```
# -*- coding: utf-8 -*-
"""project fifa.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1shQn2vGf4Sm61a67ESQlmAv0EM8SO7I9

# KPI's
Problem Statement:
With FIFA being in the blood as 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 show all information about all the World Cups in the 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.

Project is about Fifa_World_Cup data analysis, but we are going to test it on a machine learning algorithm also...As there is not enough data to build machine learning model on top, bu
Thankue
"""

import numpy as np
import pandas as pd

df=pd.read_csv('/content/WorldCupMatches.csv')

df.head(60)

df1= pd.read_csv('/content/WorldCupPlayers.csv')

df1.head(50)

df2= pd.read_csv('/content/WorldCups.csv')

df2.head(30)

df.count()

df1.count()

df2.count()

df.info()

df1.info()

df2.info()

"""# Data Preprocessing"""

# Count the number of NaN values in each column
nan_count = df.isna().sum()
print(nan_count)

# Count the number of NaN values in each column
nan_count = df1.isna().sum()
print(nan_count)

# Count the number of NaN values in each column
nan_count = df2.isna().sum()
print(nan_count)

# Merge the datasets on the common columns "roundID" and "matchID"
merged_df = pd.merge(df, df1, on=["RoundID", "MatchID"], how="left")

# Replace null values with "unknown"
merged_df = merged_df.dropna()

merged_df.count()

df.head(2)

df.info()

df1.head(2)

df1.info()

df2.head(2)

df2.info()

merged_df1 = pd.merge(df, df2, on=['Year'], how='inner')

merged_df1.count()

merged_df.head(5)

merged_df1.head(5)

merged_df1.columns

latest_df= merged_df1.drop(columns=['Referee','Assistant 1','Assistant 2' ])

latest_df.columns

latest_df.head(2)

latest_df = latest_df.rename(columns={'Attendance_x': 'Attendance'})

latest_df.columns

latest_df = latest_df.drop(columns=['Attendance_y'])

latest_df.columns

latest_df.head(2)

"""#New columns Winning Status will tell the story that which team actually won the match as it will help in further analysis

"""

def comparing(latest_df):
    if(int(latest_df['Home Team Goals']) == int(latest_df['Away Team Goals'])):
```

```
        latest_df['Winning Status'] = 0
    elif(int(latest_df['Home Team Goals']) > int(latest_df['Away Team Goals'])):
        latest_df['Winning Status'] = 1
    else:
        latest_df['Winning Status'] = 2
    return latest_df

data= latest_df.apply(comparing, axis='columns')

data.head(5)

###Just checking the relationship between halftime and teams...it shows the probability or outcomes of teams scores in first half

def checking(latest_df):
    if(int(latest_df['Half-time Home Goals']) > int(latest_df['Half-time Away Goals'])):
        latest_df['Half-time lead'] = 1
    elif(int(latest_df['Half-time Home Goals']) < int(latest_df['Half-time Away Goals'])):
        latest_df['Half-time lead'] = 2
    else:
        latest_df['Half-time lead'] = 0
    return latest_df
data= data.apply(checking, axis='columns')

data.head()

"""dropping home team initials and away team initials because both are same as home team name and away team name"""

#data = data.drop(columns=['Country', 'Winner', 'Runners-Up', 'Third', 'Fourth', 'QualifiedTeams', 'Home Team Initials', 'Away Team Initials'])

data.columns

data.head(5)

data['Year'] = data['Year'].astype(int)

data.info()

#outliers will not effect because there might be change that home team scores more goals than average
import seaborn as sns

sns.boxplot(data=df['Home Team Goals'])

"""If a team has a higher number of matches won during extra time, it could indicate that they are better at handling high-pressure situations or have more stamina compared to other te

# Create a function to determine the winning time based on the "Win conditions" column
def get_winning_time(win_conditions):
    if "after extra time" in win_conditions:
        return "Extra Time"
    elif "on penalties" in win_conditions:
        return "Penalties"
    else:
        return "Regular Time"

# Apply the function to create the "Winning Time" column
data["Winning Time"] = data["Win conditions"].apply(get_winning_time)

data.head(50)

data.drop('Win conditions', axis=1, inplace=True)

"""# Exploratory Data Analysis

"""

data.head(2)

data.columns

import plotly.express as px

# group by year and winning status, count the number of qualified teams
df = data.groupby(['Year', 'Winning Status'])['QualifiedTeams'].count().reset_index()

# create stacked bar chart
fig = px.bar(df, x='Year', y='QualifiedTeams', color='Winning Status',
             title='Qualified Teams with Winning Status')
fig.show()



#this shows the impact of attendence on teams, when hover
# create a new column called "outcome" that indicates the outcome of the match
data['outcome'] = data.apply(lambda row: 'win' if row['Winner'] == row['Home Team Name']
                                         else 'lose' if row['Winner'] == row['Away Team Name']
                                         else 'draw', axis=1)

# create a scatter plot of attendance vs. match outcome
fig = px.scatter(data, x='Attendance', y='outcome', color='outcome',
                 hover_data=['Home Team Name', 'Away Team Name'],
                 title='Impact of Attendance on Match Outcome')

fig.update_layout(xaxis_title='Attendance', yaxis_title='Outcome',
                  legend_title='Outcome', showlegend=False)

fig.show()

#this will show times the teams loss the world-cup final
# filter data to only include finals and where there is a runner-up
finals_data = data[(data['Stage'] == 'Final') & (data['Runners-Up'].notnull())]

# group by runners-up and count the number of instances
loss_counts = finals_data.groupby('Runners-Up')['Year'].count().reset_index()

# create the bar chart
fig = px.bar(loss_counts, x='Runners-Up', y='Year', title='Teams that Lost the World Cup Final')
fig.update_layout(xaxis_title='Team', yaxis_title='Count')
fig.show()

#this shows how many were scored in every final of fifa-world cup
# create a new DataFrame with goals scored in each final
final_goals = data.loc[data['Stage'] == 'Final', ['Year', 'Home Team Name', 'Home Team Goals',
                                                  'Away Team Goals', 'Away Team Name']]
final_goals['Total Goals'] = final_goals['Home Team Goals'] + final_goals['Away Team Goals']

# create a bar chart of goals scored in each final
fig = px.bar(final_goals, x='Year', y='Total Goals', color='Home Team Name',
             title='Goals Scored in Every World Cup Final')
fig.show()

data.columns
```

```
#eda3
data['Winning Margin'] = data['Home Team Goals'] - data['Away Team Goals']
extra_time_matches = data[data['Winning Time'] == 'Extra Time']
import matplotlib.pyplot as plt

plt.hist(extra_time_matches['Winning Margin'])
plt.xlabel('Winning Margin')
plt.ylabel('Frequency')
plt.title('Distribution of Winning Margins in Matches with Extra Time')
plt.show()


###In the boxplot, the x-axis represents the possible values of the winning margin.
##A winning margin of -2 or -1 means that the away team won the match by 2 or 1 goal(s) respectively.
###A winning margin of 0 means that the match was drawn.
###A winning margin of 1 or 2 means that the home team won the match by 1 or 2 goal(s) respectively.
###The y-axis represents the frequency or count of matches with that particular winning margin when extra time was played.



data.columns

#eda5

# Create a new column 'HT Lead' that contains the difference between the half-time home goals and half-time away goals
data['HT Lead'] = data['Half-time Home Goals'] - data['Half-time Away Goals']

# Create a new column 'Result' that contains the outcome of the match
data['Result'] = 'Draw'
data.loc[data['Home Team Goals'] > data['Away Team Goals'], 'Result'] = 'Home Team Win'
data.loc[data['Away Team Goals'] > data['Home Team Goals'], 'Result'] = 'Away Team Win'

# Analyze the impact of half-time leads on the final outcome of the match
df = data.groupby('HT Lead')['Result'].value_counts(normalize=True).unstack()

# Plot the results using a stacked bar chart
df.plot(kind='bar', stacked=True, figsize=(12,8))
plt.title('Impact of Half-time Leads on Match Outcome')
plt.xlabel('Half-time Lead')
plt.ylabel('Proportion of Matches')
plt.show()

data.columns

data.head(5)



#eda7
# Group the data by country and count the number of matches played in each country
matches_per_country = data.groupby('Country')['MatchID'].count()

# Sort the data by number of matches played in each country in descending order
matches_per_country = matches_per_country.sort_values(ascending=False)

# Create a bar plot to visualize the distribution
plt.figure(figsize=(15, 6))
matches_per_country.plot(kind='bar')
plt.title('Number of Matches Played in Each Country')
plt.xlabel('Country')
plt.ylabel('Number of Matches Played')
plt.show()

#eda8
#distribution of winning margins in matches where penalty shootouts were played
# Filter the dataset to include only matches where penalty shootouts were played
penalty_shootout_matches = data[data["Winning Status"] == "won after PSO"]

# Create a new column to calculate the winning margin in penalty shootouts
penalty_shootout_matches["Penalty Shootout Margin"] = penalty_shootout_matches["Home Team Goals"] - penalty_shootout_matches["Away Team Goals"]

# Plot the distribution of winning margins using a histogram
plt.hist(penalty_shootout_matches["Penalty Shootout Margin"], bins=range(-5, 6), edgecolor="black")
plt.xlabel("Winning Margin in Penalty Shootouts")
plt.ylabel("Number of Matches")
plt.title("Distribution of Winning Margins in Penalty Shootouts")
plt.show()

#Creates a new column named "Shootout Winner" by applying a lambda function to each row of the DataFrame. The lambda function checks if the match ended in a draw and then checks which
# Create a new column for the shootout winner
data['Shootout Winner'] = data.apply(lambda x: x['Home Team Name'] if x['Home Team Goals'] > x['Away Team Goals'] and x['Winning Status'] == 'Draw' else x['Away Team Name'] if x['Hom

# Create a new column for the winning team
data['Winning Team'] = data.apply(lambda x: x['Home Team Name'] if x['Home Team Goals'] > x['Away Team Goals'] else x['Away Team Name'] if x['Home Team Goals'] < x['Away Team Goals']
filtered_df = data.loc[(data['Shootout Winner'] != 'N/A') & (data['Winning Team'] == data['Away Team Name'])]

data.columns

#this shows the number of goals through the year, and it means the number of goals are increasing as the years are progressing.
# group the data by year and count the goals scored
goals_by_year = data.groupby('Year')['GoalsScored'].count().reset_index()

# create a line chart
fig = px.line(goals_by_year, x='Year', y='GoalsScored', title='Goals Scored During the Years')

fig.update_layout(xaxis_title='Year', yaxis_title='Goals Scored')

fig.show()

data.tail(59)

data['extra_time_winners'] = (data['Winning Time'] == 'Extra Time') & \
                             (((data['Half-time Home Goals'] < data['Half-time Away Goals']) & (data['Home Team Goals'] > data['Away Team Goals'])) |
                             ((data['Half-time Home Goals'] > data['Half-time Away Goals']) & (data['Home Team Goals'] < data['Away Team Goals']))) & \
                             (data['Home Team Goals'] != data['Away Team Goals'])

import matplotlib.pyplot as plt

# Count the number of games that match the filter conditions
num_extra_time_winners = data['extra_time_winners'].sum()

# Count the number of games that don't match the filter conditions
num_non_extra_time_winners = len(data) - num_extra_time_winners

# Create a pie chart
labels = ['Extra Time Winners', 'Non-Extra Time Winners']
sizes = [num_extra_time_winners, num_non_extra_time_winners]
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.axis('equal')
```

```
plt.title('Extra Time Winners vs Non-Extra Time Winners')
plt.show()

###the above analysis shows that when extra-time is given the team that was losing during half-time loses the match.



#matches played per year


matches_per_year = data['Year'].value_counts().sort_index()
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
plt.bar(matches_per_year.index, matches_per_year.values, width=0.6, color='blue')
plt.xticks(matches_per_year.index, rotation=90)
plt.xlabel('Year')
plt.ylabel('Number of Matches')
plt.title('Number of Matches Played per Year')
plt.show()



# Group the data by year and winning team
winners_by_year = data.groupby(['Year', 'Winner']).size().reset_index(name='Wins')

# Pivot the data so that the winning teams become columns and the number of wins become values
winners_pivot = winners_by_year.pivot(index='Year', columns='Winner', values='Wins')

# Plot the data as a stacked bar chart
winners_pivot.plot(kind='bar', stacked=True, figsize=(10,6))
plt.title('World Cup Winners by Year')
plt.xlabel('Year')
plt.ylabel('Number of Wins')
plt.show()

### home team victories
# Create a new column to store the number of home team victories
data['Home Team Victories'] = 0

# Iterate over the rows of the dataframe
for index, row in data.iterrows():

    # Check if the home team won the match
    if row['Home Team Goals'] > row['Away Team Goals']:

        # Increment the number of home team victories by 1
        data.loc[index, 'Home Team Victories'] += 1

data.columns

#away team victories
# Create a new column to store the number of away team victories
data['Away Team Victories'] = 0

# Iterate over the rows of the dataframe
for index, row in data.iterrows():

    # Check if the away team won the match
    if row['Away Team Goals'] > row['Home Team Goals']:

        # Increment the number of away team victories by 1
        data.loc[index, 'Away Team Victories'] += 1




import matplotlib.pyplot as plt

# Get the count of home team victories, away team victories, and drawn matches
home_wins = data['Home Team Victories'].sum()
away_wins = data['Away Team Victories'].sum()
draws = len(data) - home_wins - away_wins

# Create a list of values and labels for the pie chart
values = [home_wins, away_wins, draws]
labels = ['Home Team Victories', 'Away Team Victories', 'Drawn Matches']

# Create the pie chart
plt.pie(values, labels=labels)

# Set the title and count of the chart
plt.title('Distribution of Match Results\nHome Team Victories: {}\nAway Team Victories: {}\nDrawn Matches: {}'.format(home_wins, away_wins, draws))

# Show the chart
plt.show()

fig, ax = plt.subplots(figsize = (10,5))
sns.despine(right = True)
g = sns.barplot(x = 'Year', y = 'Attendance', data = data)
g.set_xticklabels(g.get_xticklabels(), rotation = 80)
g.set_title('Attendance Per Year')

#========================================

fig, ax = plt.subplots(figsize = (10,5))
sns.despine(right = True)
g = sns.barplot(x = 'Year', y = 'QualifiedTeams', data = data)
g.set_xticklabels(g.get_xticklabels(), rotation = 80)
g.set_title('Qualified Teams Per Year')

#========================================

fig, ax = plt.subplots(figsize = (10,5))
sns.despine(right = True)
g = sns.barplot(x = 'Year', y = 'GoalsScored', data = data)
g.set_xticklabels(g.get_xticklabels(), rotation = 80)
g.set_title('Goals Scored by Teams Per Year')


#========================================


fig, ax = plt.subplots(figsize = (10,5))
sns.despine(right = True)
g = sns.barplot(x = 'Year', y = 'MatchesPlayed', data = data)
g.set_xticklabels(g.get_xticklabels(), rotation = 80)
g.set_title('Matches  by Teams Per Year')
```

```
data.columns
```

```
data.columns
```

```
import plotly.express as px

# create a bar chart of the number of World Cup tournaments held by year
fig = px.bar(data.groupby('Year').size().reset_index(name='Count'), x='Year', y='Count', title='Number of World Cup Matches by Year')
fig.show()
```

```
import plotly.express as px

# create a pie chart of the distribution of matches by stage
fig = px.pie(data, names='Stage', title='Distribution of Matches by Stage')
fig.show()
```

```
import plotly.express as px

# create a bar chart of the top 10 stadiums by number of matches held
fig = px.bar(data.groupby('Stadium').size().reset_index(name='Count').sort_values('Count', ascending=False).head(10), x='Stadium', y='Count', title='Top 10 Stadiums by Number of Match
fig.show()
```

```
import plotly.express as px

# create a bar chart of the top 10 cities by number of matches held
fig = px.bar(data.groupby('City').size().reset_index(name='Count').sort_values('Count', ascending=False).head(10), x='City', y='Count', title='Top 10 Cities by Number of Matches Held'
fig.show()
```

```
import plotly.express as px

# create a bar chart of the top 10 home teams by number of matches played
fig = px.bar(data.groupby('Home Team Name').size().reset_index(name='Count').sort_values('Count', ascending=False).head(10), x='Home Team Name', y='Count', title='Top 10 Home Teams by
fig.show()
```

```
import plotly.express as px

# create a histogram of the distribution of home team goals
fig = px.histogram(data, x='Home Team Goals', nbins=10, title='Distribution of Home Team Goals')
fig.show()
```

```
import plotly.express as px

# create a histogram of the distribution of away team goals
fig = px.histogram(data, x='Away Team Goals', nbins=10, title='Distribution of Away Team Goals')
fig.show()
```

```
import plotly.express as px

# create a bar chart of the top 10 away teams by number of matches played
fig = px.bar(data.groupby('Away Team Name').size().reset_index(name='Count').sort_values('Count', ascending=False).head(10), x='Away Team Name', y='Count', title='Top 10 Away Teams by
fig.show()
```

```
import plotly.express as px

# create a histogram of the distribution of half-time home goals
fig = px.histogram(data, x='Half-time Home Goals', nbins=10, title='Distribution of Half-time Home Goals')
fig.show()
```

```
import plotly.express as px

# create a histogram of the distribution of half-time away goals
fig = px.histogram(data, x='Half-time Away Goals', nbins=10, title='Distribution of Half-time Away Goals')

# display the plot
fig.show()
```

```
#this shows total goals by everyteam
# combine the home team goals and away team goals into one column
data['Total Goals'] = data['Home Team Goals'] + data['Away Team Goals']

# group by team name and sum the total goals
goals_by_team = data.groupby('Home Team Name')['Total Goals'].sum().reset_index()

# create the bar chart
fig = px.bar(goals_by_team, x='Home Team Name', y='Total Goals', title='Total Goals by Team')
fig.show()
```

```
#pie chart shows the proportion of matches in the dataset that were won by each of the three possible winning statuses: 'Home Team', 'Away Team', and 'Draw'.
# group by winning status and count the number of instances
winning_status_counts = data.groupby('Winning Status')['Year'].count().reset_index()

# create the pie chart
fig = px.pie(winning_status_counts, values='Year', names='Winning Status', title='Distribution of Winning Status')
fig.show()
```

```
import plotly.express as px

# convert datetime column to datetime object
data['Datetime'] = pd.to_datetime(data['Datetime'], format='%d %b %Y - %H:%M', errors='coerce')

# group by year and calculate the mean winning margin
winning_margin_by_year = data.groupby('Year')['Winning Margin'].mean().reset_index()

# create the line plot
fig = px.line(winning_margin_by_year, x='Year', y='Winning Margin', title='Mean Winning Margin by Year')
fig.show()
```

```
import plotly.express as px

fig = px.scatter(data, x='Half-time lead', y='Result', title='Relationship Between Half-Time Lead and Full-Time Result')
fig.show()
```

```
import plotly.express as px

fig = px.scatter(data, x='Half-time lead', y='Winning Margin', title='Half-time Lead vs Winning Margin')
fig.show()
```

```
import plotly.express as px

# create a scatter plot of half-time leads for the home and away teams
fig = px.scatter(data, x='Half-time Home Goals', y='Half-time Away Goals', color='Result', title='Half-time Lead by Home and Away Teams')
fig.update_layout(xaxis_title='Half-time Home Goals', yaxis_title='Half-time Away Goals')
```

```
# display the plot
fig.show()



import plotly.graph_objs as go

# extract runner-up, third, and fourth teams
runner_up = data['Runners-Up'].value_counts().head(10)
third_place = data['Third'].value_counts().head(10)
fourth_place = data['Fourth'].value_counts().head(10)

# create bar chart for each team's finish
fig = go.Figure()
fig.add_trace(go.Bar(x=runner_up.index, y=runner_up.values, name='Runners-Up'))
fig.add_trace(go.Bar(x=third_place.index, y=third_place.values, name='Third Place'))
fig.add_trace(go.Bar(x=fourth_place.index, y=fourth_place.values, name='Fourth Place'))

# update layout
fig.update_layout(title='FIFA World Cup Top Teams (Runner-Up, Third, Fourth)',
                  xaxis_title='Team',
                  yaxis_title='Number of Finishes')

# display the plot
fig.show()

import plotly.express as px

# count the number of times each team has won the World Cup
winners = data.groupby('Winner')['Year'].nunique().reset_index(name='Wins')

# create a bar chart of the number of World Cup wins for each team
fig = px.bar(winners, x='Winner', y='Wins', title='Number of World Cup Wins by Team')

# display the plot
fig.show()



import plotly.express as px

# create a dataframe with stadium and number of home team victories
home_victories = data[data['Home Team Goals'] > data['Away Team Goals']]
home_victories_by_stadium = home_victories.groupby('Stadium')['Home Team Name'].count().reset_index()
home_victories_by_stadium.rename(columns={'Home Team Name': 'Number of Home Victories'}, inplace=True)

# create a bar chart of home team victories by stadium
fig = px.bar(home_victories_by_stadium, x='Stadium', y='Number of Home Victories',
             title='Number of Home Team Victories in Different Stadiums')
fig.show()

import plotly.express as px

# create a dataframe with the number of away team victories in each stadium
away_victories_by_stadium = data.groupby(['Stadium', 'Away Team Name'])['Away Team Victories'].sum().reset_index()

# create a grouped bar chart
fig = px.bar(away_victories_by_stadium, x='Stadium', y='Away Team Victories', color='Away Team Name', title='Away Team Victories by Stadium')

# rotate x-axis labels for better readability
fig.update_layout(xaxis_tickangle=-45)

# display the chart
fig.show()



import pandas as pd
import plotly.express as px


# create a new dataframe with the count of qualified teams by country
qualified_teams = data.groupby(['Country'])['QualifiedTeams'].count().reset_index()

# create a bar chart of the number of qualified teams by country
fig = px.bar(qualified_teams, x='Country', y='QualifiedTeams',
             title='Number of qualified teams by country')
fig.show()



import plotly.express as px

fig = px.histogram(data, x='Half-time lead', color='Result',
                   barmode='group', nbins=20,
                   category_orders={"Result": ["H", "D", "A"]},
                   title='Frequency of Half-time Lead Outcomes for Home and Away Teams')

fig.update_layout(xaxis_title='Half-time Lead', yaxis_title='Frequency',
                  legend_title='Result', barmode='group')

fig.show()

import plotly.express as px

fig = px.histogram(data, x='Stadium', title='Number of Matches Played in Each Stadium')
fig.update_layout(xaxis={'categoryorder':'total descending'})
fig.show()

import pandas as pd
import plotly.express as px



# Create a dataframe with the count of matches played in each country
matches_by_country = data.groupby(['Country']).size().reset_index(name='Matches Played')

# Create a choropleth map showing the number of matches played in each country
fig = px.choropleth(matches_by_country, locations='Country', locationmode='country names', color='Matches Played',
                    projection='natural earth', title='Number of Matches Played by Country')
fig.show()

# filter dataframe for matches that went into extra time
extra_time_matches = data[data['extra_time_winners'].notnull()]

# group by stadium and count the number of matches
matches_by_stadium = extra_time_matches.groupby('Country')['MatchID'].count().reset_index()

# plot bar chart of matches by stadium
```

```
fig = px.bar(matches_by_stadium, x='Country', y='MatchID', title='Matches with Extra Time by Country')
fig.show()

"""Index(['Year', 'Datetime', 'Stage', 'Stadium', 'City', 'Home Team Name',
       'Home Team Goals', 'Away Team Goals', 'Away Team Name', 'Attendance',
       'Half-time Home Goals', 'Half-time Away Goals', 'RoundID', 'MatchID',
       'Home Team Initials', 'Away Team Initials', 'Country', 'Winner',
       'Runners-Up', 'Third', 'Fourth', 'GoalsScored', 'QualifiedTeams',
       'MatchesPlayed', 'Winning Status', 'Half-time lead', 'Winning Time',
       'Winning Margin', 'HT Lead', 'Result', 'Shootout Winner',
       'Winning Team', 'extra_time_winners', 'Home Team Victories'],
      dtype='object')"""

import plotly.express as px

# create a bar chart of the average attendance by stadium
fig = px.bar(data_frame=data, x='Stadium', y='Attendance',
             title='Average Attendance by Stadium')
fig.show()

"""Index(['Year', 'Datetime', 'Stage', 'Stadium', 'City', 'Home Team Name',
       'Home Team Goals', 'Away Team Goals', 'Away Team Name', 'Attendance',
       'Half-time Home Goals', 'Half-time Away Goals', 'RoundID', 'MatchID',
       'Home Team Initials', 'Away Team Initials', 'Country', 'Winner',
       'Runners-Up', 'Third', 'Fourth', 'GoalsScored', 'QualifiedTeams',
       'MatchesPlayed', 'Winning Status', 'Half-time lead', 'Winning Time',
       'Winning Margin', 'HT Lead', 'Result', 'Shootout Winner',
       'Winning Team', 'extra_time_winners', 'Home Team Victories'],
      dtype='object')"""

# showing the relationship between attendance and number of goals scored:

# Create a scatter plot
plt.scatter(data['Attendance'], data['GoalsScored'])
plt.xlabel('Attendance')
plt.ylabel('Goals Scored')
plt.title('Relationship between Attendance and Goals Scored')
plt.show()

data.columns



#Assumption: Brazil is the most successful team

import matplotlib.pyplot as plt

# Create a dictionary of the number of World Cups won by each country
world_cup_counts = data['Winner'].value_counts().to_dict()

# Separate Brazil's count from the others
brazil_count = world_cup_counts.pop('Brazil')
other_counts = sum(world_cup_counts.values())

# Create a list of labels and counts for the pie chart
labels = ['Brazil', 'Other Countries']
counts = [brazil_count, other_counts]

# Create the pie chart
plt.pie(counts, labels=labels, autopct='%1.1f%%', colors=['green', 'lightgray'], startangle=90)
plt.axis('equal')
plt.title('Number of World Cups Won by Brazil vs Other Countries')
plt.show()

# Assumption: Germany has been consistent throught out the tournaments

# Set the style for the plot
sns.set_style("whitegrid")

# Filter the data to only include instances where Germany played
germany_data = data[data['Home Team Name'].str.contains('Germany') | data['Away Team Name'].str.contains('Germany')]

# Create a bar plot of the number of times Germany has reached the semifinals
sns.barplot(x="Year", y="Stage", data=germany_data[germany_data['Stage'] == 'Semi-finals'], estimator=len, palette="Blues_d")

# Set the plot title and axis labels
plt.title("Germany's Semifinal Appearances in the FIFA World Cup")
plt.xlabel("Year")
plt.ylabel("Number of Semifinal Appearances")

# Rotate the x-axis tick labels for better readability
plt.xticks(rotation=45)

# Show the plot
plt.show()




import seaborn as sns
import matplotlib.pyplot as plt

# Filter the data to include only matches that reached the semifinals
semifinals_data = data[data['Stage'] == 'Semi-finals']

# Set the figure size
plt.figure(figsize=(12, 8))

# Create a bar plot of the number of times each team has reached the semifinals
sns.barplot(x="Country", y="Year", data=semifinals_data, estimator=len, palette="Blues_d", order=semifinals_data['Country'].value_counts().index)

# Set the plot title and axis labels
plt.title('Number of times each team reached the semifinals', fontsize=16)
plt.xlabel('Country', fontsize=14)
plt.ylabel('Number of times', fontsize=14)

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Display the plot
plt.show()

import seaborn as sns

# Create a countplot of the number of times a penalty shootout was required in the final
sns.countplot(x="Shootout Winner", data=data[data['Stage'] == 'Final'])

# Add plot labels and title
plt.xlabel("Penalty Shootout Winner")
```

```
plt.ylabel("Number of Times")
plt.title("Number of Times Penalty Shootout was Required in the Final")
plt.show()

data.columns

#Assumption: Home teams have an advantage in the World Cup.


# Filter data to include only matches where there was a winning team
winning_teams = data[data['Winning Team'].notnull()]

# Create a new column to indicate if the winning team was the home team or not
winning_teams['Home Team Win'] = winning_teams['Winning Team'] == winning_teams['Home Team Name']

# Create a countplot to visualize the number of times the home team won vs. the away team won
sns.countplot(x='Home Team Win', data=winning_teams, palette=["red", "green"])

# Set axis labels and title
plt.xlabel("Did the Home Team Win?")
plt.ylabel("Number of Matches")
plt.title("Home Team Advantage in the World Cup")

# Show the plot
plt.show()

# Assumption: The World Cup has become more competitive over time.

import matplotlib.pyplot as plt
import seaborn as sns

# Create a line plot showing the total number of goals scored each year
plt.figure(figsize=(12, 8))
sns.set_style("darkgrid")
sns.lineplot(x="Year", y="GoalsScored", data=data, linewidth=2.5, color='blue')

# Add labels and title
plt.xlabel("Year")
plt.ylabel("Total Goals Scored")
plt.title("Total Number of Goals Scored in the World Cup Over Time")
plt.show()

!pip install country_converter

# Assumption: Teams from certain continents are more successful in the World Cup.
# this tells the story that Europe is dominating, because football is mostly played in Europe and parts of North America
import country_converter as coco

# Create a dictionary to map country names to their continent
continent_dict = {}
for country in data['Country'].unique():
    continent_dict[country] = coco.convert(names=country, to='Continent')

# Create a new column with the continent for each team
data['Continent'] = data['Country'].replace(continent_dict)

# Clean up some country names
data.replace({
    'Czechoslovakia': 'Czech Republic',
    'German DR': 'Germany',
    'Soviet': 'Russia',
    'Serbia and Montenegro': 'Serbia'
}, inplace=True)

# Group data by continent and winning team
continent_data = data.groupby(['Continent', 'Winning Team']).size().reset_index(name='count')

# Pivot data to create a table of victories by continent and winning team
pivot_data = continent_data.pivot(index='Continent', columns='Winning Team', values='count').fillna(0)

# Create a horizontal stacked bar chart to show the number of victories by continent and winning team
sns.set_style("whitegrid")
sns.set_palette("husl")
pivot_data.plot(kind='barh', stacked=True, figsize=(10, 6))
plt.xlabel("Number of Victories")
plt.ylabel("Continent")
plt.title("Number of World Cup Victories by Continent")
plt.legend(title="Winning Team", bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()

# Create a pie chart showing the percentage of total victories for each continent
continent_victories = data.groupby('Continent')['Winning Team'].count()
explode = [0.1 if continent == 'Europe' else 0 for continent in continent_victories.index]
plt.pie(continent_victories, labels=continent_victories.index, autopct='%1.1f%%', startangle=90, explode=explode)
plt.axis('equal')
plt.title("Percentage of World Cup Victories by Continent")
plt.show()

#Assumption: The highest-scoring World Cup match had a lot of goals.



# Filter for the highest-scoring match
max_goals = data['Total Goals'].max()
match = data[data['Total Goals'] == max_goals].iloc[0]

# Create a bar plot of the goals scored in the match
sns.set_style("whitegrid")
sns.set_palette("husl")
fig, ax = plt.subplots(figsize=(10, 6))
goals = match[['Home Team Goals', 'Away Team Goals']]
ax.bar(goals.index, goals.values, color=['blue', 'red'])
ax.set_xlabel("Team")
ax.set_ylabel("Goals")
ax.set_title("Goals Scored in the Highest-Scoring World Cup Match")
plt.show()

data.columns



# The country that hosts the World Cup has an advantage over other teams.

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Load the data
data1 = pd.read_csv("/content/WorldCupMatches.csv")
data1 = pd.merge(data1, data[['Year', 'Country', 'Winner']], on='Year', how='left')
```

```
# Drop missing values in the 'Stage' column
data1.dropna(subset=['Stage'], inplace=True)

# Subset the data to only include World Cup matches
data1 = data1[data1['Stage'].str.contains('Group|Round of 16|Quarter|Semi|Final')]

# Create a new column to indicate whether the match was played in the host country or not
data1['Host Country'] = data1['Country'] == data1['Home Team Name']

# Calculate the win percentage for the host country in matches played on their home soil versus matches played elsewhere
host_wins = data1[data1['Home Team Name'] == data1['Winner']]
away_wins = data1[data1['Away Team Name'] == data1['Winner']]

host_win_pct = len(host_wins[host_wins['Host Country'] == True]) / len(data1[data1['Host Country'] == True])
away_win_pct = len(away_wins[away_wins['Host Country'] == False]) / len(data1[data1['Host Country'] == False])

# Visualize the win percentage for the host country in home matches versus away matches
sns.set_style("whitegrid")
sns.set_palette("husl")

fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=['Host Country', 'Away'], y=[host_win_pct, away_win_pct], palette='husl')
ax.set_xlabel("Match Location")
ax.set_ylabel("Win Percentage")
ax.set_title("Win Percentage for Host Country in Home Matches versus Away Matches")

plt.show()

#The team that scores the first goal in a knockout match is more likely to win the match.

import pandas as pd
import matplotlib.pyplot as plt



# Filter the data to include only knockout matches
data2 = data[data['Stage'].str.contains('Round of 16|Quarter|Semi|Final')]

# Create a new column to indicate whether the team that scored the first goal won the match or not
data2['First Goal Winner'] = data2.apply(lambda row: row['Home Team Name'] if row['Home Team Goals'] > row['Away Team Goals'] else row['Away Team Name'] if row['Away Team Goals'] > ro
data2['First Goal'] = data2.apply(lambda row: row['Home Team Name'] if row['Home Team Goals'] > row['Away Team Goals'] else row['Away Team Name'] if row['Away Team Goals'] > row['Home

# Count the number of matches won and lost/drawn by the team that scored the first goal
first_goal_wins = data2[data2['First Goal'] != 'No Goal']['First Goal Winner'].value_counts()
first_goal_losses = data2[data2['First Goal Winner'] != data2['First Goal']][['Home Team Name', 'Away Team Name']].stack().value_counts()

# Calculate the win percentage for the team that scored the first goal
first_goal_win_pct = first_goal_wins / (first_goal_wins + first_goal_losses)

# Plot the win percentage on a bar chart
ax = first_goal_win_pct.plot(kind='bar', figsize=(10,6))
ax.set_title('Win Percentage for Team Scoring First Goal in Knockout Matches')
ax.set_xlabel('Team')
ax.set_ylabel('Win Percentage')
plt.show()

data.columns

import pandas as pd



# Create a new column "Won First Match"
data["Won First Match"] = "Draw"
data.loc[data["Home Team Goals"] > data["Away Team Goals"], "Won First Match"] = "Yes"
data.loc[data["Home Team Goals"] < data["Away Team Goals"], "Won First Match"] = "No"

# Print the first few rows of the updated dataset
print(data.head())

#Assumption: Teams that win their opening match are more likely to advance to the knockout stages.

# Create a new column to indicate whether a team won their opening match
data["Won First Match"] = (data["Home Team Goals"] > data["Away Team Goals"]) | ((data["Home Team Goals"] == data["Away Team Goals"]) & (data["Home Team Name"] < data["Away Team Name"

# Create a new column to indicate which stage a match was played in
def get_stage(row):
    if row["Stage"] in ["Group 1", "Group 2", "Group 3", "Group 4", "Group 5", "Group 6", "Group 7", "Group 8"]:
        return "Group Stage"
    elif row["Stage"] == "Round of 16":
        return "Knockout Stage"
    elif row["Stage"] == "Quarter-finals":
        return "Knockout Stage"
    elif row["Stage"] == "Semi-finals":
        return "Knockout Stage"
    elif row["Stage"] == "Match for third place":
        return "Knockout Stage"
    elif row["Stage"] == "Final":
        return "Knockout Stage"

data["Match Stage"] = data.apply(get_stage, axis=1)

# Calculate the proportion of teams that won their opening match and advanced to the knockout stages
won_opening_match = data[data["Won First Match"] == 1]
advanced_to_knockout = won_opening_match[won_opening_match["Match Stage"] == "Knockout Stage"]
won_and_advanced = len(advanced_to_knockout) / len(won_opening_match)

# Calculate the proportion of teams that lost their opening match and still advanced to the knockout stages
lost_opening_match = data[data["Won First Match"] == 0]
advanced_to_knockout = lost_opening_match[lost_opening_match["Match Stage"] == "Knockout Stage"]
lost_and_advanced = len(advanced_to_knockout) / len(lost_opening_match)

# Visualize the results
labels = ["Won First Match and Advanced", "Lost First Match and Advanced"]
values = [won_and_advanced, lost_and_advanced]

plt.bar(labels, values)
plt.title("Proportion of Teams that Advanced to Knockout Stage")
plt.show()




"""
# Statistical Assumptions checks"""

data.columns
```

```
#performing chi-square to test assumption that the team leads half-time, wins the match
#If the p-value is less than the significance level (usually 0.05), you can reject the null hypothesis and conclude that there is a significant association between the two variables.
def match_outcome(row):
    if row['Home Team Goals'] > row['Away Team Goals']:
        return 'win'
    elif row['Home Team Goals'] < row['Away Team Goals']:
        return 'loss'
    else:
        return 'draw'
data['Match Outcome'] = data.apply(match_outcome, axis=1)

import pandas as pd
from scipy.stats import chi2_contingency

# create a contingency table
contingency_table = pd.crosstab(data['Half-time lead'], data['Winning Status'])

# perform the chi-squared test
chi2, p, dof, expected = chi2_contingency(contingency_table)

# print the results
print('Chi-squared test results:')
print('Chi-squared statistic:', chi2)
print('P-value:', p)
print('Degrees of freedom:', dof)
print('Expected values:', expected)

"""chi-square value is too small ... we are considering"""

import pandas as pd
from scipy.stats import chi2_contingency

# Create a new column for whether the match went to extra time
data['Extra Time'] = data['Winning Time'] == 'Extra Time'

# Create a new column for whether the team not leading at half-time lost the match
data['Not Leading at Half-Time Lost'] = (data['Half-time lead'] == 'Away Team') & (data['Winning Status'] == 'Home Team')

# Create a contingency table
contingency_table = pd.crosstab(data['Extra Time'], data['Not Leading at Half-Time Lost'])

# Perform the chi-squared test
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Print the results
print('Chi-squared test results:')
print('Chi-squared statistic:', chi2)
print('P-value:', p)
print('Degrees of freedom:', dof)
print('Expected values:', expected)

"""Since the p-value is 1.0, which is greater than the significance level of 0.05, we fail to reject the null hypothesis. This means there is not enough evidence to conclude that the t

In other words, we cannot say there is a significant association between the half-time lead and the match outcome in matches with extra time.
"""

data.columns

import pandas as pd
from scipy.stats import pearsonr

# create a new dataframe with only relevant columns
data_subset = data[['Home Team Goals', 'Away Team Goals', 'Match Outcome']]

# calculate the correlation coefficient and p-value
corr, p = pearsonr(data_subset['Home Team Goals'] - data_subset['Away Team Goals'],
                   pd.get_dummies(data_subset['Match Outcome']).iloc[:, 0])

# print the results
print('Pearson correlation coefficient:', corr)
print('P-value:', p)

"""3rd Statistical assumption check depicts that it is not necessary that more goals means, match winner sometimes it could happen with chance or luck. We should to focus on other fact

data.columns

# Drop columns
data = data.drop(['Datetime', 'Stage','Stadium', 'City','Attendance', 'RoundID', 'MatchID','Home Team Initials', 'Away Team Initials'], axis=1)

data.columns

#data = data.drop(['Country'], axis=1)

"""# Feature Engineering and Selection """

data.info()

data.head(2)

from sklearn.preprocessing import LabelEncoder

# create a label encoder object
le = LabelEncoder()

# check if there are any non-numeric values in Home Team Name column
if data['Home Team Name'].dtype == 'object':
    # encode the Home Team Name column
    data['Home Team Name'] = le.fit_transform(data['Home Team Name'])

# check if there are any non-numeric values in Away Team Name column
if data['Away Team Name'].dtype == 'object':
    # encode the Away Team Name column
    data['Away Team Name'] = le.fit_transform(data['Away Team Name'])

# print the unique encoded values for Home Team Name
print("Unique encoded values for Home Team Name:", data['Home Team Name'].unique())

# print the unique encoded values for Away Team Name
print("Unique encoded values for Away Team Name:", data['Away Team Name'].unique())

data.head(2)

data.columns

data = data.drop(['Winner','Runners-Up', 'Third','Fourth','GoalsScored', 'QualifiedTeams', 'MatchesPlayed','Result','Shootout Winner','extra_time_winners','Extra Time','Not Leading at

data = data.drop(['Winning Time', 'HT Lead', ], axis=1)

data = data.drop(['Winning Team' ], axis=1)
```

```
data.columns

data.head(10)

data.info()

from sklearn.preprocessing import LabelEncoder

# create a LabelEncoder object
le = LabelEncoder()

# fit and transform the 'Match Outcome' column
data['Match Outcome'] = le.fit_transform(data['Match Outcome'])

data.info()

data.head(2)

#converting home team and away team goals to integer because goals are always measured in whole numbers.

data['Home Team Goals'] = data['Home Team Goals'].astype(int)
data['Away Team Goals'] = data['Away Team Goals'].astype(int)

#half-time goals are also measured in whole numbers

data['Half-time Home Goals'] = data['Half-time Home Goals'].astype(int)
data['Half-time Away Goals'] = data['Half-time Away Goals'].astype(int)

#same applies with winnning margin

data['Winning Margin'] = data['Winning Margin'].astype(int)

#now every column is in integer
data.info()

data.head(50)

data = data.drop('outcome', axis=1)

"""Now Finding Correlation for feature selection"""

#Finding correlation
# Select only numeric columns from the dataframe
numeric_cols = data.select_dtypes(include='number').columns.tolist()
data_numeric = data[numeric_cols]

# Compute the correlation matrix
corr_matrix = data_numeric.corr()

# Visualize the correlation matrix using a heatmap
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')

# Select the features with a correlation coefficient greater than 0.5
selected_features = corr_matrix[abs(corr_matrix) > 0.5].stack().reset_index().rename(columns={0: 'correlation', 'level_0': 'feature_1', 'level_1': 'feature_2'})

# Remove duplicate rows
selected_features = selected_features[selected_features['feature_1'] != selected_features['feature_2']]

# Sort the features by their correlation coefficient in descending order
selected_features = selected_features.sort_values('correlation', ascending=False)

# Print the selected features
print(selected_features)

# i can see  Match Outcome  &  Home Team Victories  =   0.917446,
# Home Team Goals   &    Winning Margin  =   0.826507
# in my use-case i want to consider match outcome as dependent variable, so i can't drop it.
#home team goals i can drop because winning margin can give me overall result that which team won the match by how many goals.
# if i am going to drop home team goals, i will also drop away team goals because it will be of  use.

data = data.drop(['Home Team Goals', 'Away Team Goals', ], axis=1)

#again checking correlation
# Check the correlation matrix
# Select only columns with numeric data types
numeric_columns = data.select_dtypes(include=[np.number])

# Compute the correlation matrix
corr_matrix = numeric_columns.corr()

# Visualize the correlation matrix using a heatmap
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')

# Select the features with a correlation coefficient greater than 0.5
selected_features = corr_matrix[abs(corr_matrix) > 0.5].stack().reset_index().rename(columns={0: 'correlation', 'level_0': 'feature_1', 'level_1': 'feature_2'})

# Remove duplicate rows
selected_features = selected_features[selected_features['feature_1'] != selected_features['feature_2']]

# Sort the features by their correlation coefficient in descending order
selected_features = selected_features.sort_values('correlation', ascending=False)

# Print the selected features
print(selected_features)

"""After doing the correlation coefficient analysis, we have dropped some features that are not contributing to dependent variable.
But it's not enough so we are going to do Univariate Feature Selection

"""

data.columns

#i am scaling this dataset, and selecting best features using selectkbest
# Encode non-numeric columns using LabelEncoder
from sklearn.feature_selection import SelectKBest, f_classif

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

for col in data.select_dtypes(include='object'):
    data[col] = le.fit_transform(data[col])

# Split the dataset into features and target variable
X = data.drop('Match Outcome', axis=1)
y = data['Match Outcome']

# Select only numeric columns for feature scaling
num_cols = X.select_dtypes(include=['float64', 'int64']).columns
```

```
# Apply feature scaling to the numeric features
X[num_cols] = (X[num_cols] - X[num_cols].mean()) / X[num_cols].std()

# Apply the f-test feature selection
k = 10
selector = SelectKBest(score_func=f_classif, k=k)
selector.fit(X, y)

# Get the selected features and their scores
selected_features = X.columns[selector.get_support()]
feature_scores = selector.scores_[selector.get_support()]

# Create a dataframe with the selected features and their scores
selected_df = pd.DataFrame({'Feature': selected_features, 'F-Score': feature_scores})

# Sort the dataframe by F-Score in descending order
selected_df = selected_df.sort_values(by='F-Score', ascending=False)

# Print the selected features and their F-Scores
print(selected_df)

"""From the above analysis there were some features not contributing like  the feature "Year" has a low F-score of 59.91830, so i am dropping but other features are much important, not



data.head(2)

data = data.drop(['Country'], axis=1)

data = data.drop(['Year'], axis=1)

data.columns

data.head(5)

"""This is final, now we are going to build model, every feature is important as per analysis, and as per domain knowledge."""

pip install statsmodels

##Checking multi-collinearity in independent features
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Drop non-numeric columns
numeric_columns = data.select_dtypes(include=[np.number])
X = numeric_columns.drop('Match Outcome', axis=1)

# Calculate VIF for each feature
vif = pd.DataFrame()
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Print the results
print(vif)

data.columns

#dropping winnning status because of highly-collinear

X = X.drop('Winning Status', axis=1)

data= data.drop('Winning Status', axis=1)

data = data.drop(['Match Stage', 'Total Goals','Won First Match','Continent'], axis=1)

data.columns

#double-checking multi-collinearity


from statsmodels.stats.outliers_influence import variance_inflation_factor

# create a dataframe with the independent variables
X = data[['Home Team Name', 'Away Team Name', 'Half-time Home Goals', 'Half-time Away Goals', 'Half-time lead', 'Winning Margin', 'Home Team Victories', 'Away Team Victories']]

# calculate VIF for each feature
vif = pd.DataFrame()
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif)

"""Now there is not any multi-collinearity in the data, we can go ahead with model building

Also there is no need for dummy variables(encoding technique) because we are already done, also data is normally districuted in sense because we have applied scaling techniques.
"""

data.columns

data.head(5)



"""# Data Preparation for Modelling

Splling data into train and test
"""

data.columns

data.head(2)

"""Checking data imbalance"""

# Compute the class distribution of the target variable
class_distribution = data['Match Outcome'].value_counts(normalize=True)
print(class_distribution)

pip install imbalanced-learn

# 2 or winning was in majority and as we are solving multi-class classification problem, it will effect the model

from imblearn.under_sampling import RandomUnderSampler

y= data[ 'Match Outcome']


# Perform undersampling
undersampler = RandomUnderSampler(sampling_strategy='majority')
X_resampled, y_resampled = undersampler.fit_resample(X, y)

# The new balanced dataset
```

```
balanced_df = pd.concat([X_resampled, y_resampled], axis=1)

print(balanced_df)

# Compute the class distribution of the target variable
class_distribution = balanced_df['Match Outcome'].value_counts(normalize=True)
print(class_distribution)

"""my data-was biased towards 2 or winner, so i was in need to balance it first before fitting the model """

balanced_df = balanced_df.drop(["Half-time Home Goals", "Half-time Away Goals", "Winning Margin"], axis=1)



# Split data into training and testing sets
from sklearn.model_selection import train_test_split
X = data.drop("Match Outcome", axis=1)
y = data["Match Outcome"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

"""# Model Building"""

from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# create an instance of the model
model = LogisticRegression(max_iter=1000)

# train the model on the training data
model.fit(X_train, y_train)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# predict the outcomes on the testing data
y_pred = model.predict(X_test)

# calculate the performance metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)

from sklearn.metrics import confusion_matrix
import seaborn as sns

# calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# plot the confusion matrix as a heatmap
sns.heatmap(cm, annot=True, cmap='Blues')

"""
This project was only about the data analysis, not any machine learning algorithm was going to be getting in use, but we tried to push data and try enhance my skills with machine learn
As In my case data was too noisy, too collinear, too imbalanced...
As my project was all about data analysis of Fifa_World_Cup I have successfully met my KPI's set forth

#Data Analysis is also done in tableau and in python too. As it answer so many questions...
#More analysis on the way in python and tableau... and the project is successfully completed.

   thankue
"""
```