WEB SCRAPING WITH SELENIUM (FLASH SCORE)

May 13, 2024

Using selenium to extract information from web pages

Created by: David Zerpa

1 Data extraction phase

1.1 Import of necesary libraries

[1]:
```
# Libraries for Google Chrome services and to handle DataFrames

from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.options import Options
import pandas as pd
import chromedriver_autoinstaller
import re

# Libraries to interact with the page
from selenium.common.exceptions import TimeoutException,␣
↪ElementClickInterceptedException
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

1.2 Creating services in Google Chrome to run session called “Driver”

[2]:
```
def setup_selenium():
    # Make sure the latest version of ChromeDriver is installed and configured
    chromedriver_autoinstaller.install()

    # Setting options for Chrome
    options = Options()

    options.add_argument("--start-maximized")

    # Uncomment the following line if you want to run Chrome in headless mode
```

```
    # options.add_argument('--headless')
    options.add_argument('--disable-gpu')  # Disable GPU (useful for headless mode)
    options.add_argument('--no-sandbox')  # Bypass OS security model, only in Chrome
    options.add_argument('--disable-dev-shm-usage')  # Overcome resource limitations

    # Configure the service
    service = Service(executable_path=chromedriver_autoinstaller.install())

    # Configure the service
    driver = webdriver.Chrome(service=service, options=options)
    return driver
```

1.3 Creating functions to interact with website

Until accessing the page where the results of LA LIGA are

[3]:
```
def accept_cookies(driver):
    try:
        # Navigate to the main page
        driver.get("https://www.flashscore.com")  # Adjust if the URL is different
        # Wait until the cookie acceptance button is clickable
        cookie_button = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.ID, "onetrust-accept-btn-handler"))
        )
        cookie_button.click()
        print("Cookies accepted successfully.")
    except Exception as e:
        print(f"Error accepting cookies: {e}")
```

[4]:
```
def navigate_to_laliga(driver):

    accept_cookies(driver)
    try:

        # Wait until the page is fully loaded (you can adjust the waiting logic as needed)
        WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.CSS_SELECTOR, "#my-leagues-list"))
        )

        # Use JavaScript to click on the element
```

```
        laliga_link = driver.find_element(By.CSS_SELECTOR, "#my-leagues-list >␣
↪div:nth-child(1) > div:nth-child(6) > a > span.leftMenu__text")
        driver.execute_script("arguments[0].click();", laliga_link)

        print("Navigated to LaLiga page successfully.")
    except Exception as e:
        print(f"Error navigating to LaLiga: {e}")
```

[5]:
```
def click_results_tab(driver):
    try:
        # Wait until the results tab is clickable
        results_tab = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.ID, "li2"))  # Using the ID for the␣
↪results tab
        )
        results_tab.click()
        print("Results tab clicked successfully.")
    except Exception as e:
        print(f"Error clicking on results tab: {e}")
```

[6]:
```
def click_show_more_matches(driver):
    try:
        while True:
            try:
                # Use WebDriverWait to ensure the element is refreshed and␣
↪clickable
                WebDriverWait(driver, 5).until(
                    EC.element_to_be_clickable((By.CSS_SELECTOR, "a.event__more.
↪event__more--static"))
                )
                # Find the "Show more matches" button each time to avoid␣
↪StaleElementReferenceException
                show_more_button = driver.find_element(By.CSS_SELECTOR, "a.
↪event__more.event__more--static")
                # Attempt to click the button
                show_more_button.click()
                #print("Clicked 'Show more matches' button.")
            except ElementClickInterceptedException:
                # If click is intercepted, try clicking via JavaScript
                show_more_button = driver.find_element(By.CSS_SELECTOR, "a.
↪event__more.event__more--static")
                driver.execute_script("arguments[0].click();", show_more_button)
                #print("Clicked 'Show more matches' button using JavaScript.")
            except StaleElementReferenceException:
                # Handle case where element goes stale as loop tries to␣
↪interact with it
```

```
                #print("Encountered a stale element, retrying...")
                continue
    except TimeoutException:
        # When no more "Show more matches" button is found, the loop exits
        #print("No more 'Show more matches' buttons to click.")
        pass
    except Exception as e:
        pass
        #print(f"Error while clicking 'Show more matches' button: {e}")
```

[7]:
```
def extract_match_data(driver,season_text):
    matches = []
    current_round = None  # Variable to keep track of the current round number

    # Assuming you have already navigated to the correct page and handled␣
 ↪cookies, etc.
    # Find all elements that might be rounds or matches
    elements = driver.find_elements(By.CSS_SELECTOR, "div.leagues--static >␣
 ↪div, div.leagues--static > div > div")

    match = re.search(r'(\d{4})/(\d{4})', season_text)
    if match:
        first_year = match.group(1)  # El primer grupo de captura (primer año)
        second_year = match.group(2)  # El segundo grupo de captura (segundo␣
 ↪año)
    else:
        first_year = None  # O maneja como prefieras si no se encuentra el␣
 ↪patrón
        second_year = None

    for element in elements:
        class_attr = element.get_attribute('class')

        # Check if the element is a round
        if "event__round--static" in class_attr:
            current_round = element.text.replace("ROUND ", "").strip()

        # If the element is a match, process the match information
        elif "event__match--static" in class_attr and current_round:
            date_time = element.find_element(By.CSS_SELECTOR, "div.
 ↪event__time").text
            date_parts = date_time.split(' ')[0].split('.')
            if len(date_parts) >= 2:
                day, month = date_parts[0], date_parts[1]  # Safely unpack day␣
 ↪and month
                # Calculate year based on the month
                year = second_year  if int(month) <= 7 else first_year
```

```
                full_date = f"{year}-{month}-{day}"  # Format date as YYYY-MM-DD
            else:
                continue  # Skip this match if date format is incorrect

            home_team = element.find_element(By.CSS_SELECTOR, "div.
↪event__participant--home").text
            away_team = element.find_element(By.CSS_SELECTOR, "div.
↪event__participant--away").text
            home_score = element.find_element(By.CSS_SELECTOR, "div.
↪event__score--home").text
            away_score = element.find_element(By.CSS_SELECTOR, "div.
↪event__score--away").text

            match_data = {
                "Round": current_round,
                "Date": full_date,
                "Home Team": home_team,
                "Away Team": away_team,
                "Home Score": int(home_score),
                "Away Score": int(away_score),
                "Total Goals": int(home_score) + int(away_score),
                "Result": determine_result(int(home_score), int(away_score))
            }
            matches.append(match_data)

    return matches
```

[8]:
```
def determine_result(home_score, away_score):
    if home_score > away_score:
        return "Home"
    elif home_score < away_score:
        return "Away"
    else:
        return "Tie"
```

[9]:
```
def navigate_to_archive(driver):
    # Use existing function to navigate to the La Liga page
    navigate_to_laliga(driver)
    # Additional navigation to click the 'Archive' tab
    archive_tab = WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.CSS_SELECTOR, "#li5"))  # ID of the␣
↪'Archive' tab
    )
    archive_tab.click()
```

[10]:
```
def get_season_links(driver):
    # Extract links for all seasons
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.CSS_SELECTOR, "div.archive__season␣
 ↪a"))
    )
    season_links = driver.find_elements(By.CSS_SELECTOR, "div.archive__season␣
 ↪a")
    return [(season.text, season.get_attribute('href')) for season in␣
 ↪season_links]
```

[11]:
```
def extract_data_for_all_seasons(driver):
    # Navigate to the archive section to get links to each season
    navigate_to_archive(driver)
    seasons = get_season_links(driver)
    all_matches = []

    # Iterate over each season's link
    for season_text, season_url in seasons:
        driver.get(season_url)
        # Use existing functions to interact with the results page of each␣
 ↪season
        click_results_tab(driver)
        click_show_more_matches(driver)
        season_matches = extract_match_data(driver,season_text)
        # Append the season information to each match record
        for match in season_matches:
            match['Season'] = season_text
        all_matches.extend(season_matches)
        print("{} Season Extracted".format(season_text))
    return all_matches
```

[12]:
```
def main():
    driver = setup_selenium()
    try:
        # Extract data for all seasons and store in a DataFrame
        all_matches_data = extract_data_for_all_seasons(driver)
        matches_df = pd.DataFrame(all_matches_data)
        # Optionally: Save the data to a CSV file
        return matches_df
    finally:
        # Ensure the driver is quit properly
        driver.quit()
```

1.4 Running the algorithm

[13]:
```
matches = main()
```

```
CHROME >= 115, using mac-arm64 as architecture identifier
Cookies accepted successfully.
Navigated to LaLiga page successfully.
Results tab clicked successfully.
LaLiga 2023/2024 Season Extracted
Results tab clicked successfully.
LaLiga 2022/2023 Season Extracted
Results tab clicked successfully.
LaLiga 2021/2022 Season Extracted
Results tab clicked successfully.
LaLiga 2020/2021 Season Extracted
Results tab clicked successfully.
LaLiga 2019/2020 Season Extracted
Results tab clicked successfully.
LaLiga 2018/2019 Season Extracted
Results tab clicked successfully.
LaLiga 2017/2018 Season Extracted
Results tab clicked successfully.
LaLiga 2016/2017 Season Extracted
Results tab clicked successfully.
Primera Division 2015/2016 Season Extracted
Results tab clicked successfully.
Primera Division 2014/2015 Season Extracted
Results tab clicked successfully.
Primera Division 2013/2014 Season Extracted
Results tab clicked successfully.
Primera Division 2012/2013 Season Extracted
Results tab clicked successfully.
Primera Division 2011/2012 Season Extracted
Results tab clicked successfully.
Primera Division 2010/2011 Season Extracted
Results tab clicked successfully.
Primera Division 2009/2010 Season Extracted
Results tab clicked successfully.
Primera Division 2008/2009 Season Extracted
Results tab clicked successfully.
Primera Division 2007/2008 Season Extracted
Results tab clicked successfully.
Primera Division 2006/2007 Season Extracted
Results tab clicked successfully.
Primera Division 2005/2006 Season Extracted
Results tab clicked successfully.
Primera Division 2004/2005 Season Extracted
Results tab clicked successfully.
```

```
Primera Division 2003/2004 Season Extracted
Results tab clicked successfully.
Primera Division 2002/2003 Season Extracted
Results tab clicked successfully.
Primera Division 2001/2002 Season Extracted
Results tab clicked successfully.
Primera Division 2000/2001 Season Extracted
Results tab clicked successfully.
Primera Division 1999/2000 Season Extracted
Results tab clicked successfully.
Primera Division 1998/1999 Season Extracted
Results tab clicked successfully.
Primera Division 1997/1998 Season Extracted
Results tab clicked successfully.
Primera Division 1996/1997 Season Extracted
Results tab clicked successfully.
Primera Division 1995/1996 Season Extracted
Results tab clicked successfully.
Primera Division 1994/1995 Season Extracted
Results tab clicked successfully.
Primera Division 1993/1994 Season Extracted
Results tab clicked successfully.
Primera Division 1992/1993 Season Extracted
Results tab clicked successfully.
Primera Division 1991/1992 Season Extracted
Results tab clicked successfully.
Primera Division 1990/1991 Season Extracted
Results tab clicked successfully.
Primera Division 1989/1990 Season Extracted
```

1.5 Creating the DataFrame

[14]:
```
matches.Date = pd.to_datetime(matches.Date)
matches['Season'] = matches['Season'].str.extract(r'(\d{4}/\d{4})')
matches = matches[['Season', 'Round', 'Date', 'Home Team', 'Away Team', 'Home␣
↪Score',
                   'Away Score','Total Goals', 'Result']]
```

[17]:
```
#original_matches = matches
```

[18]:
```
matches = original_matches
```

[19]:
```
matches.head(-5)
```

[19]:

	Season	Round	Date	Home Team	Away Team	Home Score	\
0	2023/2024	34	2024-05-05	Rayo Vallecano	Almeria	0	
1	2023/2024	34	2024-05-05	Sevilla	Granada CF	3	

```
2      2023/2024    34 2024-05-05        Valencia       Alaves           0
3      2023/2024    34 2024-05-05      Celta Vigo   Villarreal           3
4      2023/2024    34 2024-05-05         Osasuna        Betis           0
…          …   …        …               …           …          …
13453  1989/1990     1 1989-09-03        Cadiz CF  CD Logrones           0
13454  1989/1990     1 1989-09-03         Osasuna     Mallorca           1
13455  1989/1990     1 1989-09-03       R. Oviedo    Castellon           1
13456  1989/1990     1 1989-09-03     Real Madrid        Gijon           2
13457  1989/1990     1 1989-09-03         Sevilla     Tenerife           1

       Away Score  Total Goals Result
0               1            1   Away
1               0            3   Home
2               1            1   Away
3               2            5   Home
4               2            2   Away
…             …          …     …
13453           1            1   Away
13454           0            1   Home
13455           1            2    Tie
13456           0            2   Home
13457           0            1   Home

[13458 rows x 9 columns]
```

[20]:
```
matches.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13463 entries, 0 to 13462
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Season       13463 non-null  object
 1   Round        13463 non-null  object
 2   Date         13463 non-null  datetime64[ns]
 3   Home Team    13463 non-null  object
 4   Away Team    13463 non-null  object
 5   Home Score   13463 non-null  int64
 6   Away Score   13463 non-null  int64
 7   Total Goals  13463 non-null  int64
 8   Result       13463 non-null  object
dtypes: datetime64[ns](1), int64(3), object(5)
memory usage: 946.7+ KB
```

2 Variable engineering phase

[21]:
```
# Function to calculate points per match
def calculate_points(result):
    if result == 'Home':
        return 3
    elif result == 'Tie':
        return 1
    else:
        return 0
```

2.1 HOME FORM

[22]:
```
def last_5_home_form(group):
    # Calculates the cumulative sum of points and then uses the rolling␣
 ↪function to sum the last 5 results
    # Use min_periods=1 to ensure that the sum is calculated even with fewer␣
 ↪than 5 games available
    group['Last 5 Home Form'] = group['Home Points'].rolling(window=5,␣
 ↪min_periods=1).sum().shift()
    return group
```

[23]:
```
def home_form(matches):
    # Sort the data by season, home team, and date
    matches.sort_values(['Season', 'Home Team', 'Date'], inplace=True)
    # Apply the function to create a new column of points
    matches['Home Points'] = matches['Result'].apply(calculate_points)
    # Apply the function to each group of 'Season' and 'Home Team'
    matches = matches.groupby(['Season', 'Home Team']).apply(last_5_home_form)
    matches = matches.sort_values('Date', ascending=False)
    matches['Last 5 Home Form'].fillna(0, inplace=True)
    return matches
```

[24]:
```
matches = home_form(matches)
```

[25]:
```
matches.head()
```

[25]:

	Season	Round	Date	Home Team	Away Team	Home Score	Away Score	Total Goals	Result	Home Points	Last 5 Home Form
2	2023/2024	34	2024-05-05	Valencia	Alaves	0	1	1	Away	0	6.0
3	2023/2024	34	2024-05-05	Celta Vigo	Villarreal	3	2	5	Home	3	7.0
0	2023/2024	34	2024-05-05	Rayo Vallecano	Almeria	0					
4	2023/2024	34	2024-05-05	Osasuna	Betis	0					
1	2023/2024	34	2024-05-05	Sevilla	Granada CF	3					

0	1	1	Away	0	9.0
4	2	2	Away	0	6.0
1	0	3	Home	3	10.0

2.2 AWAY FORM

[26]:
```
def calculate_away_points(result):
    """Calculate points for the away team based on match results."""
    if result == 'Away':
        return 3
    elif result == 'Tie':
        return 1
    else:
        return 0
```

[27]:
```
def last_5_away_form(group):
    """Calculate the rolling sum of points from the last 5 away matches."""
    # Calculate the rolling sum of the last 5 games' points, shifting not to␣
 ↪include the current game
    group['Last 5 Away Form'] = group['Away Points'].rolling(window=5,␣
 ↪min_periods=1).sum().shift()

    return group
```

[28]:
```
def away_form(matches):
    """Calculate and append the away team's form over the last 5 matches."""
    # Sort the data by season, away team, and date
    matches.sort_values(['Season', 'Away Team', 'Date'], inplace=True)

    # Apply the function to create a new column of points for the away team
    matches['Away Points'] = matches['Result'].apply(calculate_away_points)

    # Apply the function to calculate the last 5 away games form for each group␣
 ↪of 'Season' and 'Away Team'
    matches = matches.groupby(['Season', 'Away Team']).apply(last_5_away_form)

    # Sort the DataFrame by date in descending order after processing
    matches = matches.sort_values('Date', ascending=False)
    matches['Last 5 Away Form'].fillna(0, inplace=True)

    return matches
```

[29]:
```
matches = away_form(matches)
```

[30]:
```
matches.head()
```

[30]:
```
      Season Round       Date       Home Team   Away Team  Home Score  \
3  2023/2024    34 2024-05-05      Celta Vigo  Villarreal           3
2  2023/2024    34 2024-05-05        Valencia      Alaves           0
1  2023/2024    34 2024-05-05         Sevilla  Granada CF           3
0  2023/2024    34 2024-05-05  Rayo Vallecano     Almeria           0
4  2023/2024    34 2024-05-05         Osasuna       Betis           0

   Away Score  Total Goals Result  Home Points  Last 5 Home Form  Away Points  \
3           2            5   Home            3               7.0            0
2           1            1   Away            0               6.0            3
1           0            3   Home            3              10.0            0
0           1            1   Away            0               9.0            3
4           2            2   Away            0               6.0            3

   Last 5 Away Form
3              11.0
2               4.0
1               2.0
0               5.0
4               6.0
```

2.3 Historical home vs away form

[31]:
```
def calculate_historical_form(matches):

    """Calculate the historical form of the home team against the same away team."""
    # Ensure data is sorted by date for each group to apply the correct calculations
    matches = matches.sort_values(by=['Home Team', 'Away Team', 'Date'])

    # Apply the points calculation
    matches['Home Points'] = matches['Result'].apply(calculate_points)

    # Group by home and away teams, then calculate the rolling sum and shift
    matches['HomeHistoricalPointsVSAway'] = matches.groupby(['Home Team', 'Away Team'])['Home Points'] \
        .transform(lambda x: x.rolling(window=5, min_periods=1).sum().shift())
    matches['AwayHistoricalPointsVSHome'] = matches.groupby(['Home Team', 'Away Team'])['Away Points'] \
        .transform(lambda x: x.rolling(window=5, min_periods=1).sum().shift())

    # Fill NaN values which appear for the first few games where there aren't 5 previous games
    matches['HomeHistoricalPointsVSAway'].fillna(0, inplace=True)
    matches['AwayHistoricalPointsVSHome'].fillna(0,inplace=True)
    matches = matches.sort_values('Date', ascending=False)
```

```
    return matches
```

[32]:
```
matches_test = calculate_historical_form(matches)
```

2.4 Total points both Teams (Historical)

[33]:
```
def calculate_points_before_each_match(df):
    # Sort the DataFrame by 'Season' and 'Date' and make a copy to avoid␣
↪setting with copy warning
    df_sorted = df.sort_values(by=['Season', 'Date']).copy()
    df_sorted['Season Home Cumulative Points'] = 0
    df_sorted['Season Away Cumulative Points'] = 0

    # Iterate over each unique team found in either 'Home Team' or 'Away Team'
    for team in pd.unique(df_sorted[['Home Team', 'Away Team']].values.
↪ravel('K')):
        # Filter matches where the team was either home or away
        is_home_or_away = (df_sorted['Home Team'] == team) | (df_sorted['Away␣
↪Team'] == team)
        team_matches = df_sorted[is_home_or_away].copy()

        # Calculate points for each match depending on whether the team was␣
↪home or away
        team_matches['Points'] = team_matches.apply(
            lambda x: x['Home Points'] if x['Home Team'] == team else x['Away␣
↪Points'], axis=1
        )

        # Calculate cumulative points up to each match
        team_matches['Cumulative Points'] = team_matches['Points'].cumsum().
↪shift(fill_value=0)

        # Assign calculated cumulative points back to the original sorted␣
↪DataFrame
        for index, row in team_matches.iterrows():
            if row['Home Team'] == team:
                df_sorted.loc[index, 'Historical Home Cumulative Points'] =␣
↪row['Cumulative Points']
            if row['Away Team'] == team:
                df_sorted.loc[index, 'Historical Away Cumulative Points'] =␣
↪row['Cumulative Points']
    df_sorted = df_sorted.sort_values('Date', ascending=False)
    return df_sorted
```

[34]:
```
matches_test = calculate_points_before_each_match(matches_test)
```

2.5 Total points both Teams (Per Season)

[35]:
```
def calculate_seasonal_points(df):
    # Sort the DataFrame by 'Season' and 'Date' and make a copy to avoid␣
↪setting with copy warning
    df_sorted = df.sort_values(by=['Season', 'Date']).copy()
    df_sorted['Season Home Cumulative Points'] = 0
    df_sorted['Season Away Cumulative Points'] = 0

    # Process each season separately
    for season in df_sorted['Season'].unique():
        season_data = df_sorted[df_sorted['Season'] == season]
        # Iterate over each unique team found in either 'Home Team' or 'Away␣
↪Team' within the season
        for team in pd.unique(season_data[['Home Team', 'Away Team']].values.
↪ravel('K')):
            # Filter matches where the team was either home or away in the␣
↪current season
            is_home_or_away = (season_data['Home Team'] == team) |␣
↪(season_data['Away Team'] == team)
            team_matches = season_data[is_home_or_away].copy()

            # Calculate points for each match depending on whether the team was␣
↪home or away
            team_matches['Points'] = team_matches.apply(
                lambda x: x['Home Points'] if x['Home Team'] == team else␣
↪x['Away Points'], axis=1
            )

            # Calculate cumulative points up to each match, resetting each␣
↪season
            team_matches['Cumulative Points'] = team_matches['Points'].cumsum().
↪shift(fill_value=0)

            # Assign calculated cumulative points back to the original sorted␣
↪DataFrame
            for index, row in team_matches.iterrows():
                if row['Home Team'] == team:
                    df_sorted.loc[index, 'Season Home Cumulative Points'] =␣
↪row['Cumulative Points']
                if row['Away Team'] == team:
                    df_sorted.loc[index, 'Season Away Cumulative Points'] =␣
↪row['Cumulative Points']
    df_sorted = df_sorted.sort_values('Date', ascending=False)
    return df_sorted
```

[36]:
```
matches_test = calculate_seasonal_points(matches_test)
```

2.6 Method that extracts information prior to the next match to make future predictions

[43]:
```
def get_upcoming_match_stats(matches, home_team, away_team):
    """
    Retrieve current match statistics for a specific matchup based on␣
↪historical performance, recent form,
    and updated cumulative points based on the latest match results.

    Parameters:
    - matches (DataFrame): The DataFrame containing all match data.
    - home_team (str): The name of the home team.
    - away_team (str): The name of the away team.

    Returns:
    - DataFrame: A DataFrame with the most relevant recent stats for the␣
↪matchup.
    """
    # Filter by matches where the specified teams are home and away for␣
↪historical points
    specific_matches = matches[(matches['Home Team'] == home_team) &␣
↪(matches['Away Team'] == away_team)]
    specific_matches = specific_matches.sort_values('Date', ascending=False)

    # Filter the recent matches of each team regardless of their opponent for␣
↪the form of the last 5 matches
    recent_home_matches = matches[matches['Home Team'] == home_team].head(5)
    recent_away_matches = matches[matches['Away Team'] == away_team].head(5)

    # Extract recent form statistics
    home_form = recent_home_matches['Home Points'].sum()
    away_form = recent_away_matches['Away Points'].sum()

    # Extract the last available historical points if previous matches exist
    historical_home_pointsVsAway = specific_matches['Home Points'].head(5).
↪sum() if not specific_matches.empty else 0
    historical_away_pointsVsHome = specific_matches['Away Points'].head(5).
↪sum() if not specific_matches.empty else 0

    # Get the last match for each team to extract cumulative points including␣
↪the latest match
    last_home_match = matches[(matches['Home Team'] == home_team) |␣
↪(matches['Away Team'] == home_team)].sort_values('Date', ascending=False).
↪iloc[0]
```

```
    last_away_match = matches[(matches['Home Team'] == away_team) |␣
↪(matches['Away Team'] == away_team)].sort_values('Date', ascending=False).
↪iloc[0]

    # Determine points from the last match based on the team's position (home␣
↪or away)
    if last_home_match['Home Team'] == home_team:
        last_home_points = last_home_match['Home Points']
        home_current_points = last_home_match['Season Home Cumulative Points']
        historical_home_points = last_home_match['Historical Home Cumulative␣
↪Points']
    else:
        last_home_points = last_home_match['Away Points']
        home_current_points = last_home_match['Season Away Cumulative Points']
        historical_home_points = last_home_match['Historical Away Cumulative␣
↪Points']

    if last_away_match['Home Team'] == away_team:
        last_away_points = last_away_match['Home Points']
        away_current_points = last_away_match['Season Home Cumulative Points']
        historical_away_points = last_away_match['Historical Home Cumulative␣
↪Points']
    else:
        last_away_points = last_away_match['Away Points']
        away_current_points = last_away_match['Season Away Cumulative Points']
        historical_away_points = last_away_match['Historical Away Cumulative␣
↪Points']

    # Update total cumulative points by adding the points from the last match
    home_current_points += last_home_points
    away_current_points += last_away_points
    historical_home_points += last_home_points
    historical_away_points += last_away_points
    # Prepare information to return
    upcoming_match_stats = {
        'Home Team': home_team,
        'Away Team': away_team,
        'Last 5 Home Form': home_form,
        'Last 5 Away Form': away_form,
        'HomeHistoricalPointsVSAway': historical_home_pointsVsAway,
        'AwayHistoricalPointsVSHome': historical_away_pointsVsHome,
        'Season Home Cumulative Points': home_current_points,
        'Season Away Cumulative Points': away_current_points,
        'Historical Home Cumulative Points': historical_home_points,
        'Historical Away Cumulative Points': historical_away_points
    }
```

```
    return pd.DataFrame([upcoming_match_stats])
```

[45]:
```
upcoming_match_stats = get_upcoming_match_stats(matches, 'Barcelona', 'Real␣
↪Sociedad')
upcoming_match_stats.head()
```

[45]:
```
   Home Team      Away Team  Last 5 Home Form  Last 5 Away Form  \
0  Barcelona  Real Sociedad                13                10

   HomeHistoricalPointsVSAway  AwayHistoricalPointsVSHome  \
0                          12                           3

   Season Home Cumulative Points  Season Away Cumulative Points  \
0                             73                             54

   Historical Home Cumulative Points  Historical Away Cumulative Points
0                             2823.0                             1714.0
```

[51]:
```
matches_test.head(10)
```

[51]:
```
      Season Round       Date       Home Team    Away Team  Home Score  \
4  2023/2024    34 2024-05-05         Osasuna        Betis           0
3  2023/2024    34 2024-05-05      Celta Vigo   Villarreal           3
1  2023/2024    34 2024-05-05         Sevilla   Granada CF           3
2  2023/2024    34 2024-05-05        Valencia       Alaves           0
0  2023/2024    34 2024-05-05  Rayo Vallecano      Almeria           0
6  2023/2024    34 2024-05-04          Girona    Barcelona           4
8  2023/2024    34 2024-05-04   Real Sociedad   Las Palmas           2
5  2023/2024    34 2024-05-04        Mallorca  Atl. Madrid           0
7  2023/2024    34 2024-05-04     Real Madrid     Cadiz CF           3
9  2023/2024    34 2024-05-03          Getafe   Ath Bilbao           0

   Away Score  Total Goals Result  Home Points  Last 5 Home Form  Away Points  \
4           2            2   Away            0               6.0            3
3           2            5   Home            3               7.0            0
1           0            3   Home            3              10.0            0
2           1            1   Away            0               6.0            3
0           1            1   Away            0               9.0            3
6           2            6   Home            3              13.0            0
8           0            2   Home            3               4.0            0
5           1            1   Away            0               9.0            3
7           0            3   Home            3              15.0            0
9           2            2   Away            0               8.0            3

   Last 5 Away Form  HomeHistoricalPointsVSAway  AwayHistoricalPointsVSHome  \
4               6.0                         4.0                        10.0
```

```
3              11.0                        5.0                        8.0
1               2.0                       12.0                        3.0
2               4.0                       13.0                        1.0
0               5.0                        9.0                        0.0
6              10.0                        0.0                        9.0
8               2.0                        6.0                        6.0
5               4.0                       10.0                        4.0
7               2.0                       10.0                        4.0
9               4.0                        7.0                        4.0

   Season Home Cumulative Points  Season Away Cumulative Points  \
4                             39                             49
3                             31                             45
1                             38                             21
2                             47                             38
0                             34                             14
6                             71                             73
8                             51                             37
5                             32                             64
7                             84                             26
9                             43                             58

   Historical Home Cumulative Points  Historical Away Cumulative Points
4                             1141.0                             1378.0
3                             1303.0                             1363.0
1                             1858.0                              380.0
2                             2122.0                              576.0
0                              768.0                              297.0
6                              208.0                             2823.0
8                             1711.0                              231.0
5                             1122.0                             2157.0
7                             2790.0                              334.0
9                              883.0                             1856.0
```

[41]:
```
matches_test.columns
```

[41]:
```
Index(['Season', 'Round', 'Date', 'Home Team', 'Away Team', 'Home Score',
       'Away Score', 'Total Goals', 'Result', 'Home Points',
       'Last 5 Home Form', 'Away Points', 'Last 5 Away Form',
       'HomeHistoricalPointsVSAway', 'AwayHistoricalPointsVSHome',
       'Season Home Cumulative Points', 'Season Away Cumulative Points',
       'Historical Home Cumulative Points',
       'Historical Away Cumulative Points'],
      dtype='object')
```

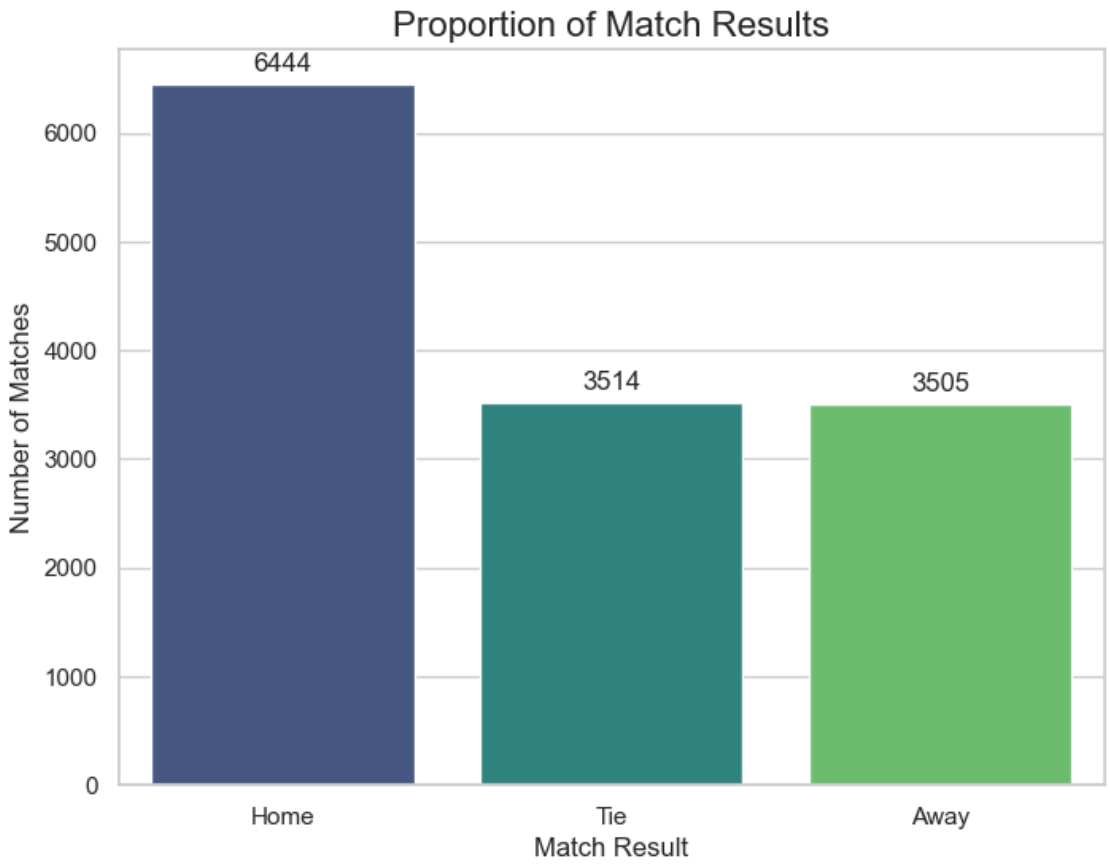
3 Exploratory data analysis phase

[42]:
```
matches = matches_test
```

[10]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

3.1 Analysis of the Proportion of Results

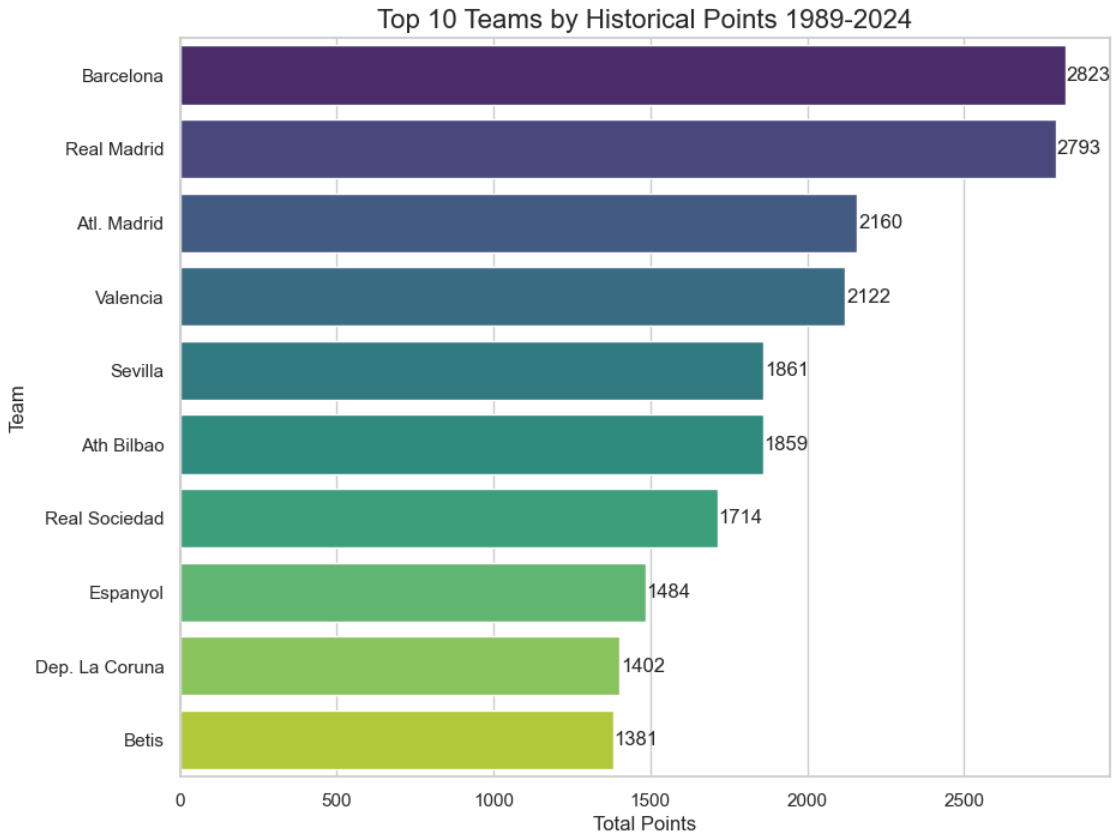
[44]:
```
result_counts = matches['Result'].value_counts()

# Set the aesthetic style of the plots
sns.set(style="whitegrid")

# Prepare the bar plot
plt.figure(figsize=(8, 6))
ax = sns.barplot(x=result_counts.index, y=result_counts.values,␣
 ↪palette="viridis")

# Adding labels and title
ax.set_xlabel("Match Result", fontsize=12)
ax.set_ylabel("Number of Matches", fontsize=12)
ax.set_title("Proportion of Match Results", fontsize=16)

# Annotate the bars with the exact number of matches
for p in ax.patches:
    ax.annotate(format(p.get_height(), '.0f'),
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha = 'center', va = 'center',
                xytext = (0, 9),
                textcoords = 'offset points')

plt.show()
```



3.2 Top 10 Teams with Most Historical Points 1989-2024

[50]:
```
# Sum up points for home and away by team
home_points = matches.groupby('Home Team')['Home Points'].sum()
away_points = matches.groupby('Away Team')['Away Points'].sum()

# Combine the sums of home and away points for each team
total_points = home_points.add(away_points, fill_value=0)

# Sort teams by total points and select the top 10
top_teams = total_points.sort_values(ascending=False).head(10)

# Visualization using seaborn and matplotlib
sns.set(style="whitegrid")
plt.figure(figsize=(10, 8))
ax = sns.barplot(x=top_teams.values, y=top_teams.index, palette='viridis')
ax.set_title('Top 10 Teams by Historical Points 1989-2024', fontsize=16)
ax.set_xlabel('Total Points', fontsize=12)
ax.set_ylabel('Team', fontsize=12)
```

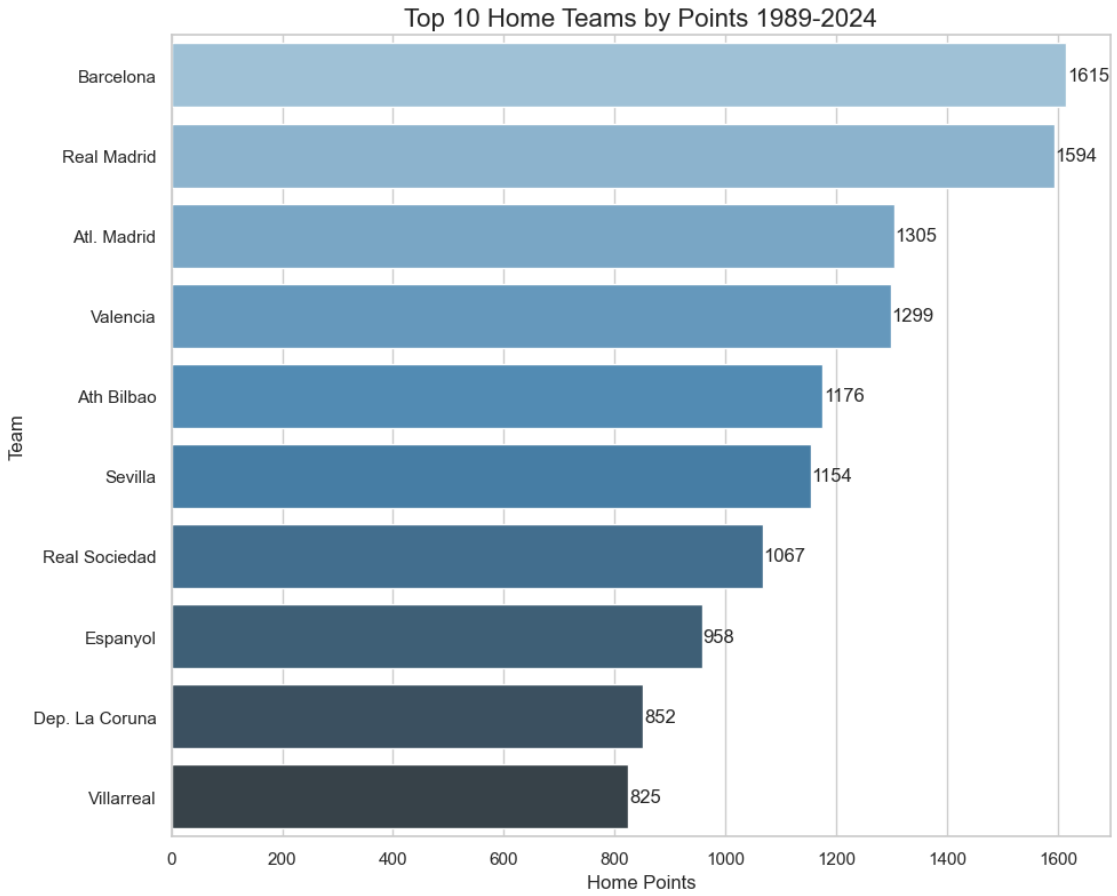
```
# Adding value labels to each bar
for p in ax.patches:
    ax.annotate(f'{int(p.get_width())}',  # format as integer
                (p.get_width(), p.get_y() + p.get_height() / 2),  # position
                ha='left', va='center',
                xytext=(0.5, 0),  # 5 points horizontal offset
                textcoords='offset points')

plt.show()
```

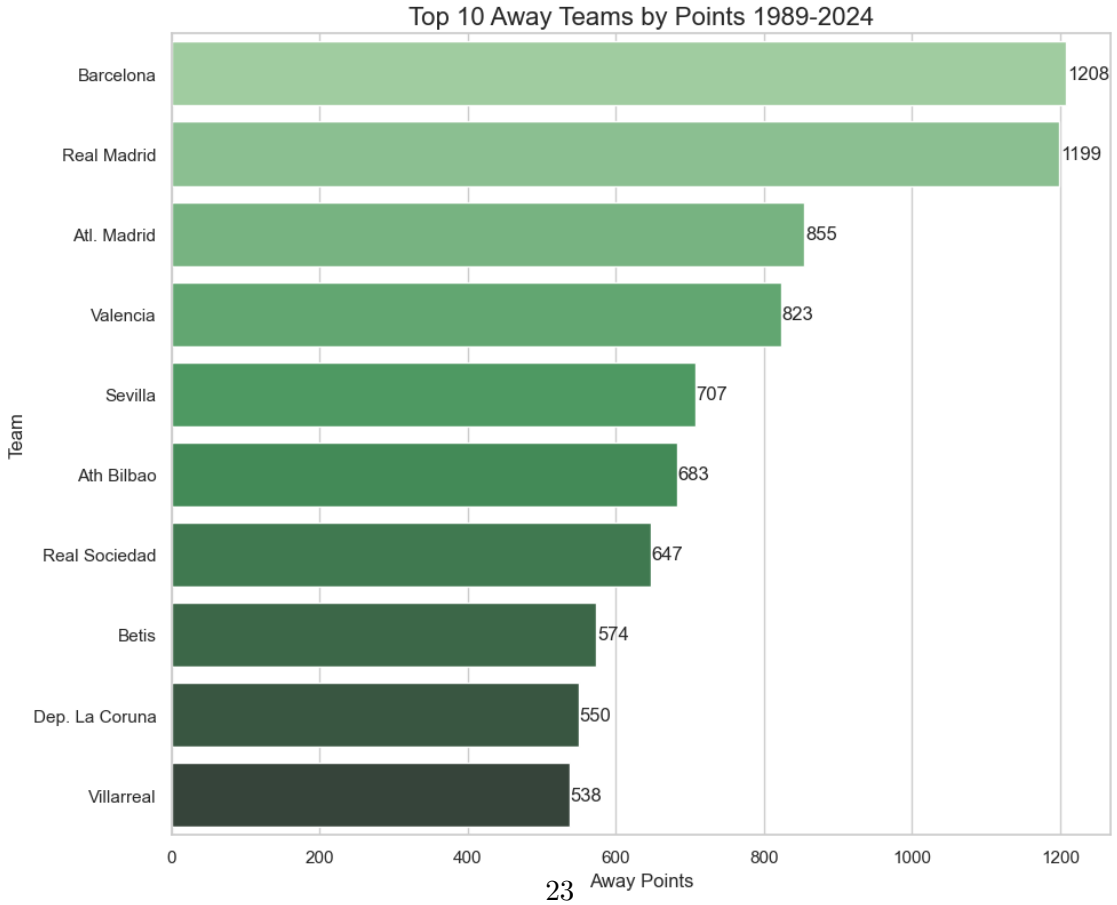


3.3 Best Home and Away Teams

[59]:
```
# Sorting and selecting the top 10 for home and away conditions
top_home_teams = home_points.sort_values(ascending=False).head(10)
top_away_teams = away_points.sort_values(ascending=False).head(10)

# Visualization using seaborn and matplotlib for Home Teams
sns.set(style="whitegrid")
fig, ax = plt.subplots(2, 1, figsize=(10, 16))  # Create 2 plots
```

```
# Plot for top home teams
sns.barplot(x=top_home_teams.values, y=top_home_teams.index, palette='Blues_d',␣
 ↪ax=ax[0])
ax[0].set_title('Top 10 Home Teams by Points 1989-2024', fontsize=16)
ax[0].set_xlabel('Home Points', fontsize=12)
ax[0].set_ylabel('Team', fontsize=12)
for p in ax[0].patches:
    ax[0].annotate(f'{int(p.get_width())}', (p.get_width(), p.get_y() + p.
 ↪get_height() / 2),
                   ha='left', va='center', xytext=(0.5, 0), textcoords='offset␣
 ↪points')

# Plot for top away teams
sns.barplot(x=top_away_teams.values, y=top_away_teams.index,␣
 ↪palette='Greens_d', ax=ax[1])
ax[1].set_title('Top 10 Away Teams by Points 1989-2024', fontsize=16)
ax[1].set_xlabel('Away Points', fontsize=12)
ax[1].set_ylabel('Team', fontsize=12)
for p in ax[1].patches:
    ax[1].annotate(f'{int(p.get_width())}', (p.get_width(), p.get_y() + p.
 ↪get_height() / 2),
                   ha='left', va='center', xytext=(0.5, 0), textcoords='offset␣
 ↪points')

plt.tight_layout()
plt.show()
```


Top 10 Home Teams by Points 1989-2024
Barcelona
Real Madrid
Atl. Madrid
Valencia
Ath Bilbao
Sevilla
Real Sociedad
Espanyol
Dep. La Coruna
Villarreal
1615
1594
1305
1299
1176
1154
1067
958
852
825
Team
0
200
400
600
800
1000
1200
1400
1600
Home Points

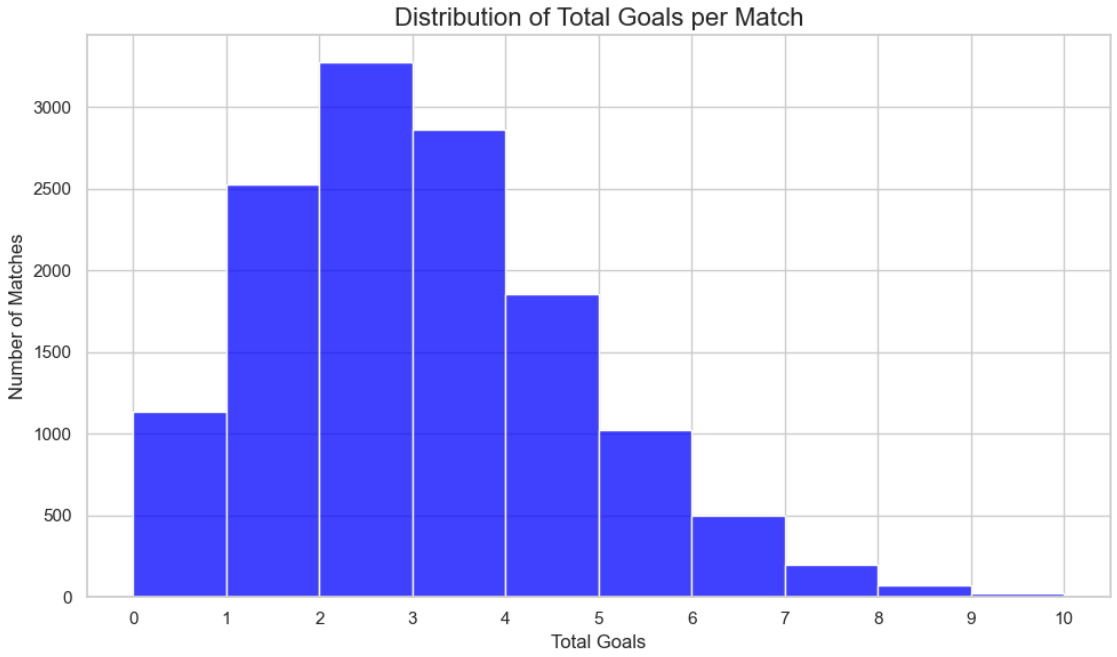

Top 10 Away Teams by Points 1989-2024
Barcelona
Real Madrid
Atl. Madrid
Valencia
Sevilla
Ath Bilbao
Real Sociedad
Betis
Dep. La Coruna
Villarreal
1208
1199
855
823
707
683
647
574
550
538
Team
0
200
400
600
800
1000
1200
Away Points

3.3.1 Evolution of Points for a Specific Team by Season

[60]:
```
team_name = "Atl. Madrid"  # Example team, replace with the team you are␣
 ↪interested in

# Filter matches for the selected team and explicitly create a copy
team_matches = matches[(matches['Home Team'] == team_name) | (matches['Away␣
 ↪Team'] == team_name)].copy()

# Calculate points for each match
team_matches['Points'] = team_matches.apply(
    lambda x: x['Home Points'] if x['Home Team'] == team_name else x['Away␣
 ↪Points'], axis=1
)

# Sum points per season
season_points = team_matches.groupby('Season')['Points'].sum()

# Visualization using seaborn and matplotlib
sns.set(style="whitegrid")
plt.figure(figsize=(12, 6))
sns.lineplot(x=season_points.index, y=season_points.values, marker='o',␣
 ↪color='b')
plt.title(f'Seasonal Points Evolution for {team_name} 1989-2024', fontsize=16)
plt.xlabel('Season', fontsize=12)
plt.ylabel('Total Points', fontsize=12)
plt.xticks(rotation=45)  # Rotate labels to improve readability
plt.grid(True)
plt.tight_layout()
plt.show()
```

3.4 Distribution of Goals per Match

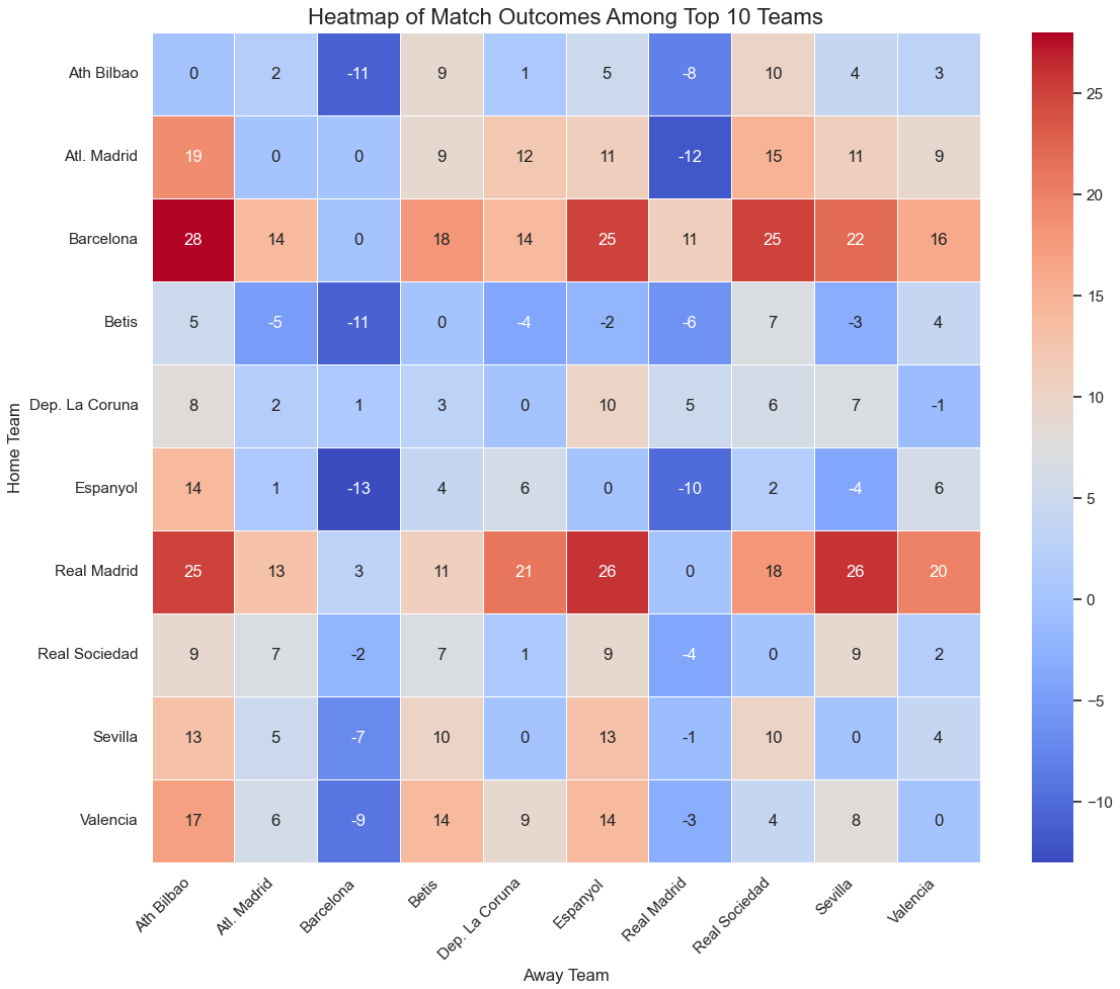
[61]:
```
# Visualization using seaborn and matplotlib
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
sns.histplot(data=matches, x='Total Goals', bins=range(0, 11), kde=False,␣
↪color='blue')
plt.title('Distribution of Total Goals per Match', fontsize=16)
plt.xlabel('Total Goals', fontsize=12)
plt.ylabel('Number of Matches', fontsize=12)
plt.xticks(range(0, 11))  # Ensuring that we cover all possible goals from 0 to␣
↪10
plt.grid(True)
plt.tight_layout()
plt.show()
```



3.5 Heat Map of Results between TOP 10 Teams with more historical points

[69]:
```
# Calculate total points for each team from previously computed home and away␣
 ↪points
home_points = matches.groupby('Home Team')['Home Points'].sum()
away_points = matches.groupby('Away Team')['Away Points'].sum()
total_points = home_points.add(away_points, fill_value=0)

# Identify the top 10 teams
top_teams = total_points.nlargest(10).index.tolist()

# Filter matches where both the home and away teams are in the top 10
filtered_matches = matches[(matches['Home Team'].isin(top_teams)) &␣
 ↪(matches['Away Team'].isin(top_teams))].copy()

# Convert the categorical 'Result' into numeric values for aggregation
result_mapping = {'Home': 1, 'Tie': 0, 'Away': -1}
filtered_matches['Result Numeric'] = filtered_matches['Result'].
 ↪map(result_mapping)

# Create a pivot table to count each type of result for home vs. away teams
result_counts = pd.pivot_table(data=filtered_matches, values='Result Numeric',␣
 ↪index='Home Team', columns='Away Team', aggfunc=np.sum, fill_value=0)

# Visualization using seaborn
```

```
sns.set(style="white")
plt.figure(figsize=(12, 10))
ax = sns.heatmap(result_counts, annot=True, fmt="d", cmap='coolwarm',␣
 ↪linewidths=.5)
plt.title('Heatmap of Match Outcomes Among Top 10 Teams', fontsize=16)
plt.xlabel('Away Team', fontsize=12)
plt.ylabel('Home Team', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



[74]:
```
matches.to_csv("LaLiga89-24-round34.csv",index=False)
```

[2]:
```
matches = pd.read_csv("LaLiga89-24-round34.csv")
```

[3]:
```
matches.head()
```

[3]:

	Season	Round	Date	Home Team	Away Team	Home Score	Away Score	Total Goals	Result	Home Points	Last 5 Home Form	Away Points	Last 5 Away Form	HomeHistoricalPointsVSAway	AwayHistoricalPointsVSHome	Season Home Cumulative Points	Season Away Cumulative Points	Historical Home Cumulative Points	Historical Away Cumulative Points	Result Numeric
0	2023/2024	34	2024-05-05	Osasuna	Betis	0	2	2	Away	0	6.0	3	6.0	4.0	10.0	39	49	1141.0	1378.0	-1
1	2023/2024	34	2024-05-05	Celta Vigo	Villarreal	3	2	5	Home	3	7.0	0	11.0	5.0	8.0	31	45	1303.0	1363.0	1
2	2023/2024	34	2024-05-05	Sevilla	Granada CF	3	0	3	Home	3	10.0	0	2.0	12.0	3.0	38	21	1858.0	380.0	1
3	2023/2024	34	2024-05-05	Valencia	Alaves	0	1	1	Away	0	6.0	3	4.0	13.0	1.0	47	38	2122.0	576.0	-1
4	2023/2024	34	2024-05-05	Rayo Vallecano	Almeria	0	1	1	Away	0	9.0	3	5.0	9.0	0.0	34	14	768.0	297.0	-1

[4]:
```
matches.columns
```

[4]:
```
Index(['Season', 'Round', 'Date', 'Home Team', 'Away Team', 'Home Score',
       'Away Score', 'Total Goals', 'Result', 'Home Points',
       'Last 5 Home Form', 'Away Points', 'Last 5 Away Form',
       'HomeHistoricalPointsVSAway', 'AwayHistoricalPointsVSHome',
       'Season Home Cumulative Points', 'Season Away Cumulative Points',
       'Historical Home Cumulative Points',
       'Historical Away Cumulative Points', 'Result Numeric'],
      dtype='object')
```

[5]:
```
matches.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13463 entries, 0 to 13462
Data columns (total 20 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Season                              13463 non-null  object
 1   Round                               13463 non-null  object
 2   Date                                13463 non-null  object
 3   Home Team                           13463 non-null  object
 4   Away Team                           13463 non-null  object
 5   Home Score                          13463 non-null  int64
 6   Away Score                          13463 non-null  int64
 7   Total Goals                         13463 non-null  int64
 8   Result                              13463 non-null  object
 9   Home Points                         13463 non-null  int64
 10  Last 5 Home Form                    13463 non-null  float64
 11  Away Points                         13463 non-null  int64
 12  Last 5 Away Form                    13463 non-null  float64
 13  HomeHistoricalPointsVSAway          13463 non-null  float64
 14  AwayHistoricalPointsVSHome          13463 non-null  float64
 15  Season Home Cumulative Points       13463 non-null  int64
 16  Season Away Cumulative Points       13463 non-null  int64
 17  Historical Home Cumulative Points   13463 non-null  float64
 18  Historical Away Cumulative Points   13463 non-null  float64
 19  Result Numeric                      13463 non-null  int64
dtypes: float64(6), int64(8), object(6)
memory usage: 2.1+ MB
```

4 Predictive model development

4.1 Logistic regression

[7]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.utils.class_weight import compute_class_weight
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
```

[36]:
```
def perform_logistic_regression_balanced(matches):
    """
    Perform logistic regression to predict match results using class weights to␣
↪handle imbalance.

    Args:
    matches (DataFrame): DataFrame containing the match data with historical␣
↪and current performance metrics.

    Returns:
    str: Accuracy of the model and a brief classification report.
    """
    # Selecting features and target
    features = matches[['Last 5 Home Form', 'Last 5 Away Form',␣
↪'HomeHistoricalPointsVSAway',
                        'AwayHistoricalPointsVSHome', 'Season Home Cumulative␣
↪Points',
                        'Season Away Cumulative Points', 'Historical Home␣
↪Cumulative Points',
                        'Historical Away Cumulative Points']]
    target = matches['Result']

    # Encoding the target variable
    le = LabelEncoder()
    target_encoded = le.fit_transform(target)

    # Compute class weights
    class_weights = compute_class_weight('balanced', classes=np.
↪unique(target_encoded), y=target_encoded)
    class_weight_dict = dict(enumerate(class_weights))

    # Splitting data into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(features,␣
↪target_encoded, test_size=0.2, random_state=42)

    # Feature scaling
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Creating logistic regression model with class weights
    model = LogisticRegression(max_iter=1000, class_weight=class_weight_dict)
```

```
    # Training the model
    model.fit(X_train_scaled, y_train)

    # Making predictions
    y_pred = model.predict(X_test_scaled)

    # Evaluating the model
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred, target_names=le.
 ↪classes_)

    return model,scaler,le, accuracy, conf_matrix, class_report
```

[38]:
```
model_logistic,scaler,encoder, accuracy, conf_matrix, class_report =␣
 ↪perform_logistic_regression_balanced(matches)
```

[39]:
```
print(f"Accuracy: {accuracy}\nConfusion Matrix:\n{conf_matrix}\nClassification␣
 ↪Report:\n{class_report}")
```

```
Accuracy: 0.45673969550686966
Confusion Matrix:
[[355 193 148]
 [320 716 235]
 [273 294 159]]
Classification Report:
              precision    recall  f1-score   support

        Away       0.37      0.51      0.43       696
        Home       0.60      0.56      0.58      1271
         Tie       0.29      0.22      0.25       726

    accuracy                           0.46      2693
   macro avg       0.42      0.43      0.42      2693
weighted avg       0.46      0.46      0.45      2693
```

4.1.1 Predictions round 36 season 2023-2024 with logistic regression model

[81]:
```
def predict_next_matches(matches, model_logistic, scaler, encoder):
    """
    Predict the outcomes of upcoming matches using a logistic regression model␣
 ↪and display probabilities,
    highlighting the most probable outcome unless it's a tie.

    Args:
    matches (DataFrame): DataFrame containing historical match data.
```

```
    model_logistic (LogisticRegression): Trained logistic regression model.
    scaler (StandardScaler): Scaler object used for scaling features.
    encoder (LabelEncoder): Encoder used for encoding the target variable.
    """
    upcoming_matches = [
        ("Osasuna", "Mallorca"),
        ("Real Madrid", "Alaves"),
        ("Girona", "Villarreal"),
        ("Rayo Vallecano", "Granada CF"),
        ("Sevilla", "Cadiz CF"),
        ("Celta Vigo", "Ath Bilbao"),
        ("Getafe", "Atl. Madrid"),
        ("Las Palmas", "Betis"),
        ("Almeria", "Barcelona"),
        ("Real Sociedad", "Valencia")
    ]

    for home, away in upcoming_matches:
        match_stats = get_upcoming_match_stats(matches, home, away)
        if not match_stats.empty:
            features = match_stats[['Last 5 Home Form', 'Last 5 Away Form',␣
↪'HomeHistoricalPointsVSAway',
                                    'AwayHistoricalPointsVSHome', 'Season Home␣
↪Cumulative Points',
                                    'Season Away Cumulative Points',␣
↪'Historical Home Cumulative Points',
                                    'Historical Away Cumulative Points']]
            features_scaled = scaler.transform(features)
            probabilities = model_logistic.predict_proba(features_scaled)
            class_labels = encoder.classes_
            max_prob_index = probabilities[0].argmax()
            max_prob_class = class_labels[max_prob_index]
            max_prob_value = probabilities[0][max_prob_index]

            # Format and print the probabilities along with the class labels
            probability_output = ", ".join([f"{class_labels[i]}: {prob:.2f}"␣
↪for i, prob in enumerate(probabilities[0])])
            if True:  # Highlight only if the max probability is greater than␣
↪50% and not a tie
                print(f"Match: {home} vs {away} - Predicted Probabilities:␣
↪{probability_output} - Most Likely: [{max_prob_class}]")
            else:
                print(f"Match: {home} vs {away} - Predicted Probabilities:␣
↪{probability_output}")
        else:
```

```
            print(f"Match: {home} vs {away} - No sufficient data to predict␣
↪this match.")
```

[82]:
```
predict_next_matches(matches, model_logistic, scaler, encoder)
```

```
Match: Osasuna vs Mallorca - Predicted Probabilities: Away: 0.32, Home: 0.35,
Tie: 0.33 - Most Likely: [Home]
Match: Real Madrid vs Alaves - Predicted Probabilities: Away: 0.04, Home: 0.82,
Tie: 0.14 - Most Likely: [Home]
Match: Girona vs Villarreal - Predicted Probabilities: Away: 0.31, Home: 0.35,
Tie: 0.34 - Most Likely: [Home]
Match: Rayo Vallecano vs Granada CF - Predicted Probabilities: Away: 0.20, Home:
0.49, Tie: 0.31 - Most Likely: [Home]
Match: Sevilla vs Cadiz CF - Predicted Probabilities: Away: 0.12, Home: 0.63,
Tie: 0.25 - Most Likely: [Home]
Match: Celta Vigo vs Ath Bilbao - Predicted Probabilities: Away: 0.57, Home:
0.14, Tie: 0.29 - Most Likely: [Away]
Match: Getafe vs Atl. Madrid - Predicted Probabilities: Away: 0.69, Home: 0.08,
Tie: 0.24 - Most Likely: [Away]
Match: Las Palmas vs Betis - Predicted Probabilities: Away: 0.53, Home: 0.17,
Tie: 0.30 - Most Likely: [Away]
Match: Almeria vs Barcelona - Predicted Probabilities: Away: 0.85, Home: 0.02,
Tie: 0.14 - Most Likely: [Away]
Match: Real Sociedad vs Valencia - Predicted Probabilities: Away: 0.42, Home:
0.28, Tie: 0.30 - Most Likely: [Away]
```

As we see, 5 victories for the visitor have been predicted, and 5 victories for the home team. No ties, which reveals a problem with the model when it comes to identifying ties. We have also seen this in the evaluation of the model. We will see what the real results are.