Predicted_Points_All_Players

May 22, 2025

1 FPL Predicted Points: Data Pipeline and Odds Integration

This notebook predicts Fantasy Premier League (FPL) points for all players in the next gameweek(s) by integrating historical FPL data, live fixture information, and bookmaker odds from Oddschecker. The workflow includes:

1. **Data Fetching:** Retrieve FPL teams, players, and fixtures from the official API.
2. **Data Preparation:** Normalize and aggregate historical and current season stats for teams and players.
3. **Odds Scraping:** Use Selenium to scrape match and player odds from Oddschecker.
4. **Probability Calculation:** Convert odds to probabilities for match outcomes, player goals, assists, and saves.
5. **Prediction Calculation:** Combine probabilities and historical rates to estimate expected points for each player.
6. **Results Output:** Save detailed and summary results to Excel, and print top 5 predicted players by position.

[1]:
```
# Import all required libraries for data fetching, processing, and web scraping.
import requests
import pandas as pd
from bs4 import BeautifulSoup

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException
from selenium.common.exceptions import TimeoutException
from selenium.common.exceptions import ElementClickInterceptedException
from selenium.webdriver.common.action_chains import ActionChains
import undetected_chromedriver as uc
import time
from fractions import Fraction
from collections import defaultdict
from unicodedata import normalize
from itertools import zip_longest
import os
```

```
import math
import csv
import ast
import chardet
import typing
import statistics
```

1.1 Overview

This notebook scrapes betting odds from Oddschecker.com, converts the odds to probabilities, and calculates predicted points for Fantasy Premier League (FPL) players for the next full gameweek. It uses Selenium for web scraping and matches player/team names between Oddschecker and the FPL API. The script also handles cases where teams play multiple matches in a gameweek and includes improved player matching logic.

1.2 Data Fetching and Preparation

The following cells define functions to fetch FPL data (teams, players, fixtures), normalize names, and aggregate historical and current season stats for teams and players.

[2]:
```
def get_all_fixtures() -> list:
    """
    Fetch all Premier League fixtures from the FPL API.

    Returns:
        list: A list of fixture dictionaries, each containing details about a␣
 ↪scheduled or completed match.

    Raises:
        Exception: If the API request fails.
    """
    url = "https://fantasy.premierleague.com/api/fixtures/"
    response = requests.get(url)
    if response.status_code != 200:
        raise Exception(f"Failed to fetch fixtures: {response.status_code}")
    # Get all fixtures from FPL API
    return response.json()
```

[3]:
```
def get_next_gws(fixtures: list, extra_gw: str = 'False') -> list:
    """
    Find the next gameweek(s) that have not yet started.

    Args:
        fixtures (list): List of fixture dictionaries from the FPL API.
        extra_gw (str): If 'True', return the next two gameweeks; otherwise,␣
 ↪return only the next gameweek.

    Returns:
```

```
        list: A list containing the next gameweek(s) as integers.
    """
    game_weeks = defaultdict(list)
    for fixture in fixtures:
        game_weeks[fixture["event"]].append(fixture)
    next_gameweek = None
    for event in sorted(game_weeks.keys()):
        if all(not fixture['finished_provisional'] for fixture in␣
 ↪game_weeks[event]):
            next_gameweek = event
            break
    if next_gameweek is None:
        raise Exception("No upcoming gameweek found.")
    if extra_gw == 'True':
        return [next_gameweek, next_gameweek + 1]
    else:
        return [next_gameweek]
```

[4]:
```
# Mapping of team names from Oddschecker to FPL API team names for consistency.
TEAM_NAMES_ODDSCHECKER = {
    "Nott'm Forest": "Nottingham Forest",
    "Wolves": "Wolverhampton",
    "Spurs": "Tottenham",
    }
```

[5]:
```
# Mapping of player names from Oddschecker to FPL API player names for␣
 ↪consistency.
PLAYER_NAMES_ODDSCHECKER = {
    "Diogo Jota": "Diogo Teixeira Da Silva",
    "Yegor Yarmolyuk": "Yehor Yarmoliuk"
    }
```

[6]:
```
def fetch_fpl_data() -> tuple:
    """
    Fetch all FPL data from the API, including teams and players.

    Returns:
        tuple: (data, teams_data, players_data, team_id_to_name,␣
 ↪player_id_to_name)
            - data: Full API response as a dictionary.
            - teams_data: List of team dictionaries.
            - players_data: List of player dictionaries.
            - team_id_to_name: Mapping from team ID to team name (with␣
 ↪Oddschecker mapping).
            - player_id_to_name: Mapping from player ID to full player name.
    """
    url = "https://fantasy.premierleague.com/api/bootstrap-static/"
```

```
    response = requests.get(url)
    if response.status_code != 200:
        raise Exception(f"Failed to fetch teams: {response.status_code}")
    data = response.json()
    # Get team data from FPL API
    teams_data = data['teams']
    # Get player data from FPL API
    players_data = data['elements']
    # A dictionary containing the team name corresponding to each team id
    team_id_to_name = {int(team['id']): TEAM_NAMES_ODDSCHECKER.
↪get(team['name'], team['name']) for team in teams_data}
    player_id_to_name = {int(player['id']): player["first_name"] + " " +␣
↪player['second_name'] for player in players_data}

    return data, teams_data, players_data, team_id_to_name, player_id_to_name
```

[7]:
```
def get_next_fixtures(fixtures: list, next_gws: list) -> list:
    # Return fixtures for the next full gameweek(s) that have not started yet.
    return [fixture for fixture in fixtures if (fixture['event'] in next_gws)␣
↪and (fixture['started'] == False)]
```

[8]:
```
def print_and_store_next_fixtures(next_fixtures: list, team_id_to_name: dict)␣
↪-> dict:
    """
    Print and return the teams playing in the next gameweek(s).

    Args:
        next_fixtures (list): List of fixture dictionaries for the next␣
↪gameweek(s).
        team_id_to_name (dict): Mapping from team ID to team name.

    Returns:
        dict: Count of games for each team in the next gameweek(s).
    """
    print("Predicted Points Will Be Calculated for The Following Fixtures:")
    print('')
    teams_playing = defaultdict(int)
    for fixture in next_fixtures:
        teams_playing[TEAM_NAMES_ODDSCHECKER.
↪get(team_id_to_name[fixture['team_h']], team_id_to_name[fixture['team_h']])]␣
↪+= 1
        teams_playing[TEAM_NAMES_ODDSCHECKER.
↪get(team_id_to_name[fixture['team_a']], team_id_to_name[fixture['team_a']])]␣
↪+= 1
        print(f"GW{fixture['event']} {team_id_to_name[fixture['team_h']]} v.␣
↪{team_id_to_name[fixture['team_a']]}")
```

```
    print('')
    return teams_playing
```

[9]:
```
def prepare_name(name: str) -> list:
    """
    Normalize a name for robust comparison by converting to lowercase, removing␣
 ↪accents, and splitting into tokens.

    Args:
        name (str): The name to normalize.

    Returns:
        list: List of capitalized tokens from the cleaned name.
    """
    # Replace foreign letters with their ASCII equivalents
    foreign_replacements = {
        'ø': 'o',
        'å': 'a',
        'æ': 'ae',
        'ä': 'a',
        'ö': 'o',
        'ú': 'u',
        'ü': 'u',
        'é': 'e',
        'ñ': 'n',
        'ï': 'i',
        'í': 'i',
        'ã': 'a',
        'á': 'a',
        'č': 'c',
        'ć': 'c',
        'š': 's'
    }
    for foreign_char, ascii_char in foreign_replacements.items():
        name = name.lower().replace(foreign_char, ascii_char)

    # Normalize the name to handle accents and foreign characters
    normalized_name = normalize('NFKD', name).encode('ascii', 'ignore').
 ↪decode('ascii')

    cleaned_name = normalized_name.replace('-', ' ')
    cleaned_name = cleaned_name.replace("'", '')
    # Split into tokens
    name_tokens = cleaned_name.split()
    cap_tokens = [token.capitalize() for token in name_tokens]
    return cap_tokens
```

[10]:
```python
def get_pos_range(position: int) -> str:
    """
    Return the league position range string for a given position (1-5, 6-10, etc.).

    Args:
        position (int): League position.

    Returns:
        str: Position range as string.
    """
    if position <= 5:
        return '1-5'
    elif position <= 10:
        return '6-10'
    elif position <= 15:
        return '11-15'
    elif position <= 20:
        return '16-20'
    else:
        return 'Unknown'
```

[11]:
```python
def get_team_template(pos_22_23: int, pos_23_24: int, pos: int) -> dict:
    """
    Create a template dictionary for storing team statistics, initialized to default values.

    Args:
        pos_22_23 (int): Team's position in 2022/23 season.
        pos_23_24 (int): Team's position in 2023/24 season.
        pos (int): Current league position.

    Returns:
        dict: Team statistics template.
    """
    team_template = {'League Position': pos,
        '22/23 League Position': pos_22_23,
        '23/24 League Position': pos_23_24,
        'ELO': 1000,
        'Home ELO': 1000,
        'Away ELO': 1000,
        'Home ELO 22/23': 1000,
        'Away ELO 22/23': 1000,
        'Home ELO 23/24': 1000,
        'Away ELO 23/24': 1000,
        'Home ELO 24/25': 1000,
        'Away ELO 24/25': 1000,
```

```
        'Home Goals': 0,
        'Away Goals': 0,
        'Home Assists': 0,
        'Away Assists': 0,
        'Goals Conceded Home': 0,
        'Goals Conceded Away': 0,
        'Home Games Played': 0,
        'Away Games Played': 0,
        'Home Goalkeeper Saves': 0,
        'Away Goalkeeper Saves': 0,
        '22/23 Home Goals': 0,
        '22/23 Away Goals': 0,
        '22/23 Home Assists': 0,
        '22/23 Away Assists': 0,
        '22/23 Goals Conceded Home': 0,
        '22/23 Goals Conceded Away': 0,
        '22/23 Home Goalkeeper Saves': 0,
        '22/23 Away Goalkeeper Saves': 0,
        '23/24 Home Goals': 0,
        '23/24 Away Goals': 0,
        '23/24 Home Assists': 0,
        '23/24 Away Assists': 0,
        '23/24 Goals Conceded Home': 0,
        '23/24 Goals Conceded Away': 0,
        '23/24 Home Goalkeeper Saves': 0,
        '23/24 Away Goalkeeper Saves': 0,
        'Home Games Against 1-5': 0,
        'Home Goals Against 1-5': 0,
        'Home Goals Conceded Against 1-5': 0,
        'Home Games Against 6-10': 0,
        'Home Goals Against 6-10': 0,
        'Home Goals Conceded Against 6-10': 0,
        'Home Games Against 11-15': 0,
        'Home Goals Against 11-15': 0,
        'Home Goals Conceded Against 11-15': 0,
        'Home Games Against 16-20': 0,
        'Home Goals Against 16-20': 0,
        'Home Goals Conceded Against 16-20': 0,
        'Away Games Against 1-5': 0,
        'Away Goals Against 1-5': 0,
        'Away Goals Conceded Against 1-5': 0,
        'Away Games Against 6-10': 0,
        'Away Goals Against 6-10': 0,
        'Away Goals Conceded Against 6-10': 0,
        'Away Games Against 11-15': 0,
        'Away Goals Against 11-15': 0,
        'Away Goals Conceded Against 11-15': 0,
```

```
        'Away Games Against 16-20': 0,
        'Away Goals Against 16-20': 0,
        'Away Goals Conceded Against 16-20': 0,
        '22/23 Home Games Against 1-5': 0,
        '22/23 Home Goals Against 1-5': 0,
        '22/23 Home Goals Conceded Against 1-5': 0,
        '22/23 Home Games Against 6-10': 0,
        '22/23 Home Goals Against 6-10': 0,
        '22/23 Home Goals Conceded Against 6-10': 0,
        '22/23 Home Games Against 11-15': 0,
        '22/23 Home Goals Against 11-15': 0,
        '22/23 Home Goals Conceded Against 11-15': 0,
        '22/23 Home Games Against 16-20': 0,
        '22/23 Home Goals Against 16-20': 0,
        '22/23 Home Goals Conceded Against 16-20': 0,
        '22/23 Away Games Against 1-5': 0,
        '22/23 Away Goals Against 1-5': 0,
        '22/23 Away Goals Conceded Against 1-5': 0,
        '22/23 Away Games Against 6-10': 0,
        '22/23 Away Goals Against 6-10': 0,
        '22/23 Away Goals Conceded Against 6-10': 0,
        '22/23 Away Goals Against 11-15': 0,
        '22/23 Away Games Against 11-15': 0,
        '22/23 Away Goals Conceded Against 11-15': 0,
        '22/23 Away Games Against 16-20': 0,
        '22/23 Away Goals Against 16-20': 0,
        '22/23 Away Goals Conceded Against 16-20': 0,
        '23/24 Home Games Against 1-5': 0,
        '23/24 Home Goals Against 1-5': 0,
        '23/24 Home Goals Conceded Against 1-5': 0,
        '23/24 Home Games Against 6-10': 0,
        '23/24 Home Goals Against 6-10': 0,
        '23/24 Home Goals Conceded Against 6-10': 0,
        '23/24 Home Games Against 11-15': 0,
        '23/24 Home Goals Against 11-15': 0,
        '23/24 Home Goals Conceded Against 11-15': 0,
        '23/24 Home Games Against 16-20': 0,
        '23/24 Home Goals Against 16-20': 0,
        '23/24 Home Goals Conceded Against 16-20': 0,
        '23/24 Away Games Against 1-5': 0,
        '23/24 Away Goals Against 1-5': 0,
        '23/24 Away Goals Conceded Against 1-5': 0,
        '23/24 Away Games Against 6-10': 0,
        '23/24 Away Goals Against 6-10': 0,
        '23/24 Away Goals Conceded Against 6-10': 0,
        '23/24 Away Goals Against 11-15': 0,
        '23/24 Away Games Against 11-15': 0,
```

```
        '23/24 Away Goals Conceded Against 11-15': 0,
        '23/24 Away Games Against 16-20': 0,
        '23/24 Away Goals Against 16-20': 0,
        '23/24 Away Goals Conceded Against 16-20': 0,}
    return team_template
```

[12]:
```
def get_player_template(team_name: str, minutes: int, starts: int) -> dict:
    """
    Create a template dictionary for storing player statistics, initialized to␣
 ↪default values.

    Args:
        team_name (str): Name of the player's team.
        minutes (int): Total minutes played.
        starts (int): Number of starts.

    Returns:
        dict: Player statistics template.
    """
    player_template = {
            'Team': team_name,
            'Minutes': minutes,
            'Starts': starts,
            'Home Games Played for Current Team': 0,
            'Away Games Played for Current Team': 0,
            'Home Goals for Current Team': 0,
            'Away Goals for Current Team': 0,
            'Home Assists for Current Team': 0,
            'Away Assists for Current Team': 0,
            'Goalkeeper Saves for Current Team': 0,
            '22/23 Home Games Played for Current Team': 0,
            '22/23 Away Games Played for Current Team': 0,
            '22/23 Home Goals for Current Team': 0,
            '22/23 Away Goals for Current Team': 0,
            '22/23 Home Assists for Current Team': 0,
            '22/23 Away Assists for Current Team': 0,
            '22/23 Goalkeeper Saves for Current Team': 0,
            '23/24 Home Games Played for Current Team': 0,
            '23/24 Away Games Played for Current Team': 0,
            '23/24 Home Goals for Current Team': 0,
            '23/24 Away Goals for Current Team': 0,
            '23/24 Home Assists for Current Team': 0,
            '23/24 Away Assists for Current Team': 0,
            '23/24 Goalkeeper Saves for Current Team': 0,
            'BPS for Current Team': 0,
            '22/23 BPS for Current Team': 0,
            '23/24 BPS for Current Team': 0,
```

```
            'Home Games Against 1-5': 0,
            'Home Goals Against 1-5': 0,
            'Home Assists Against 1-5': 0,
            'Home Games Against 6-10': 0,
            'Home Goals Against 6-10': 0,
            'Home Assists Against 6-10': 0,
            'Home Games Against 11-15': 0,
            'Home Goals Against 11-15': 0,
            'Home Assists Against 11-15': 0,
            'Home Games Against 16-20': 0,
            'Home Goals Against 16-20': 0,
            'Home Assists Against 16-20': 0,
            'Away Games Against 1-5': 0,
            'Away Goals Against 1-5': 0,
            'Away Assists Against 1-5': 0,
            'Away Games Against 6-10': 0,
            'Away Goals Against 6-10': 0,
            'Away Assists Against 6-10': 0,
            'Away Games Against 11-15': 0,
            'Away Goals Against 11-15': 0,
            'Away Assists Against 11-15': 0,
            'Away Games Against 16-20': 0,
            'Away Goals Against 16-20': 0,
            'Away Assists Against 16-20': 0,
            '22/23 Home Games Against 1-5': 0,
            '22/23 Home Goals Against 1-5': 0,
            '22/23 Home Assists Against 1-5': 0,
            '22/23 Home Games Against 6-10': 0,
            '22/23 Home Goals Against 6-10': 0,
            '22/23 Home Assists Against 6-10': 0,
            '22/23 Home Games Against 11-15': 0,
            '22/23 Home Goals Against 11-15': 0,
            '22/23 Home Assists Against 11-15': 0,
            '22/23 Home Games Against 16-20': 0,
            '22/23 Home Goals Against 16-20': 0,
            '22/23 Home Assists Against 16-20': 0,
            '22/23 Away Games Against 1-5': 0,
            '22/23 Away Goals Against 1-5': 0,
            '22/23 Away Assists Against 1-5': 0,
            '22/23 Away Games Against 6-10': 0,
            '22/23 Away Goals Against 6-10': 0,
            '22/23 Away Assists Against 6-10': 0,
            '22/23 Away Games Against 11-15': 0,
            '22/23 Away Goals Against 11-15': 0,
            '22/23 Away Assists Against 11-15': 0,
            '22/23 Away Games Against 16-20': 0,
            '22/23 Away Goals Against 16-20': 0,
```

```
            '22/23 Away Assists Against 16-20': 0,
            '23/24 Home Games Against 1-5': 0,
            '23/24 Home Goals Against 1-5': 0,
            '23/24 Home Assists Against 1-5': 0,
            '23/24 Home Games Against 6-10': 0,
            '23/24 Home Goals Against 6-10': 0,
            '23/24 Home Assists Against 6-10': 0,
            '23/24 Home Games Against 11-15': 0,
            '23/24 Home Goals Against 11-15': 0,
            '23/24 Home Assists Against 11-15': 0,
            '23/24 Home Games Against 16-20': 0,
            '23/24 Home Goals Against 16-20': 0,
            '23/24 Home Assists Against 16-20': 0,
            '23/24 Away Games Against 1-5': 0,
            '23/24 Away Goals Against 1-5': 0,
            '23/24 Away Assists Against 1-5': 0,
            '23/24 Away Games Against 6-10': 0,
            '23/24 Away Goals Against 6-10': 0,
            '23/24 Away Assists Against 6-10': 0,
            '23/24 Away Games Against 11-15': 0,
            '23/24 Away Goals Against 11-15': 0,
            '23/24 Away Assists Against 11-15': 0,
            '23/24 Away Games Against 16-20': 0,
            '23/24 Away Goals Against 16-20': 0,
            '23/24 Away Assists Against 16-20': 0,}
    return player_template
```

[13]:
```
def construct_team_and_player_data(
    fpl_data: dict,
    team_id_to_name: dict,
    player_id_to_name: dict,
    fixtures: list
) -> tuple:
    """
    Build and return two dictionaries:
      1. Team statistics (goals, assists, games played, saves, etc.)
      2. Player statistics (games/goals/assists/saves for current team, etc.)

    Args:
        fpl_data (dict): FPL API data.
        team_id_to_name (dict): Mapping from team ID to team name.
        player_id_to_name (dict): Mapping from player ID to player name.
        fixtures (list): List of fixture dictionaries.

    Returns:
        tuple: (team_data, player_data)
    """
```

```
    teams = fpl_data['teams']
    elements = fpl_data['elements']

    team_data = {}
    player_data = defaultdict(lambda: defaultdict(float))

    fixtures = [fixture for fixture in fixtures if (fixture['finished'] ==␣
↪True)]

    fixtures_23_24 = []
    with open('fixtures.csv', newline='') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            # Convert the 'stats' field from a string to a Python object (list␣
↪of dictionaries)
            if 'stats' in row:
                row['stats'] = ast.literal_eval(row['stats'])
            fixtures_23_24.append(row)

    with open('teams.csv', newline='') as csvfile2:
        reader2 = csv.DictReader(csvfile2)
        teams_23_24 = [row for row in reader2]

    fixtures_22_23 = []
    with open('fixtures_22_23.csv', newline='') as csvfile3:
        reader3 = csv.DictReader(csvfile3)
        for row in reader3:
            # Convert the 'stats' field from a string to a Python object (list␣
↪of dictionaries)
            if 'stats' in row:
                row['stats'] = ast.literal_eval(row['stats'])
            fixtures_22_23.append(row)

    with open('teams_22_23.csv', newline='') as csvfile4:
        reader4 = csv.DictReader(csvfile4)
        teams_22_23 = [row for row in reader4]

    with open('player_idlist_22_23.csv', 'rb') as f:
        result = chardet.detect(f.read())
    if result is None:
        raise ValueError("Could not detect encoding of player_idlist_22_23.csv")
    else:
        with open('player_idlist_22_23.csv', newline='',␣
↪encoding=result['encoding']) as csvfile5:
            reader5 = csv.DictReader(csvfile5)
            player_idlist_22_23 = [row for row in reader5]
```

```
    with open('player_idlist_23_24.csv', 'rb') as f2:
        result2 = chardet.detect(f2.read())
    if result2 is None:
        raise ValueError("Could not detect encoding of player_idlist_23_24.csv")
    else:
        with open('player_idlist_23_24.csv', newline='',␣
↪encoding=result2['encoding']) as csvfile6:
            reader6 = csv.DictReader(csvfile6)
            player_idlist_23_24 = [row for row in reader6]

    team_id_to_name_22_23 = {int(team['id']): TEAM_NAMES_ODDSCHECKER.
↪get(team['name'], team['name']) for team in teams_22_23}
    team_id_to_name_23_24 = {int(team['id']): TEAM_NAMES_ODDSCHECKER.
↪get(team['name'], team['name']) for team in teams_23_24}

    player_id_to_name_22_23 = {int(player['id']): player["first_name"] + " " +␣
↪player['second_name'] for player in player_idlist_22_23}
    player_id_to_name_23_24 = {int(player['id']): player["first_name"] + " " +␣
↪player['second_name'] for player in player_idlist_23_24}

    season_23_24_team_positions = {
        'Man City': 1,
        'Arsenal': 2,
        'Man Utd': 8,
        'Newcastle': 7,
        'Liverpool': 3,
        'Brighton': 11,
        'Aston Villa': 4,
        'Tottenham': 5,
        'Brentford': 16,
        'Fulham': 14,
        'Crystal Palace': 10,
        'Chelsea': 6,
        'Wolverhampton': 15,
        'West Ham': 9,
        'Bournemouth': 13,
        'Nottingham Forest': 17,
        'Everton': 12,
        'Sheffield Utd': 20,
        'Burnley': 19,
        'Luton': 18
        }

    season_22_23_team_positions = {
        'Man City': 1,
        'Arsenal': 2,
        'Man Utd': 3,
```

```
        'Newcastle': 4,
        'Liverpool': 5,
        'Brighton': 6,
        'Aston Villa': 7,
        'Tottenham': 8,
        'Brentford': 9,
        'Fulham': 10,
        'Crystal Palace': 11,
        'Chelsea': 12,
        'Wolverhampton': 13,
        'West Ham': 14,
        'Bournemouth': 15,
        'Nottingham Forest': 16,
        'Everton': 17,
        'Leicester': 18,
        'Leeds': 19,
        'Southampton': 20
        }

    # Initialize team data set to 0
    for team in teams:
        team_name_key = team['name'] if team['name'] is not None else ""
        team_name = TEAM_NAMES_ODDSCHECKER.get(team_name_key, team_name_key)
        pos_22_23 = season_22_23_team_positions.get(team_name, 21)
        pos_23_24 = season_23_24_team_positions.get(team_name, 21)
        pos_current = team.get('position', 21)
        team_data[team_name] = defaultdict(float)
        team_data[team_name].update(get_team_template(pos_22_23, pos_23_24,␣
↪pos_current))

    for player in elements:
        name = " ".join(prepare_name(player_id_to_name[player['id']]))
        team_name_key = player['team'] if player['team'] is not None else ""
        team_name_lookup = team_id_to_name.get(team_name_key, "Unknown")
        team_name = TEAM_NAMES_ODDSCHECKER.get(team_name_lookup,␣
↪team_name_lookup)
        if team_name is None:
            team_name = ""
        minutes = player['minutes']
        starts = player['starts']
        player_data[name] = defaultdict(float)
        player_data[name].update(get_player_template(team_name, minutes,␣
↪starts))

    k_factor = 20 # K-factor for ELO rating system

    for fixture in fixtures_22_23:
```

```
        home_team_id = int(fixture['team_h'])
        away_team_id = int(fixture['team_a'])
        home_team_name = TEAM_NAMES_ODDSCHECKER.
↪get(team_id_to_name_22_23[home_team_id], team_id_to_name_22_23[home_team_id])
        away_team_name = TEAM_NAMES_ODDSCHECKER.
↪get(team_id_to_name_22_23[away_team_id], team_id_to_name_22_23[away_team_id])
        home_pos_22_23 = season_22_23_team_positions.get(home_team_name, 21)
        away_pos_22_23 = season_22_23_team_positions.get(away_team_name, 21)
        home_pos_23_24 = season_23_24_team_positions.get(home_team_name, 21)
        away_pos_23_24 = season_23_24_team_positions.get(away_team_name, 21)
        team_data[home_team_name] = team_data.get(home_team_name,␣
↪defaultdict(float, get_team_template(home_pos_22_23, home_pos_23_24, 21)))
        team_data[away_team_name] = team_data.get(away_team_name,␣
↪defaultdict(float, get_team_template(away_pos_22_23, away_pos_23_24, 21)))

        # Ensure team_data always contains defaultdict(float)
        if not isinstance(team_data.get(home_team_name), defaultdict):
            team_data[home_team_name] = defaultdict(float,␣
↪team_data[home_team_name])
        if not isinstance(team_data.get(away_team_name), defaultdict):
            team_data[away_team_name] = defaultdict(float,␣
↪team_data[away_team_name])

        # Update ELO rankings
        home_goals = int(fixture['team_h_score'])
        away_goals = int(fixture['team_a_score'])

        home_pos_range = get_pos_range(home_pos_22_23)
        away_pos_range = get_pos_range(away_pos_22_23)

        home_games_against_string = f"22/23 Home Games Against {away_pos_range}"
        home_goals_against_string = f"22/23 Home Goals Against {away_pos_range}"
        home_goals_conceded_against_string = f"22/23 Home Goals Conceded␣
↪Against {away_pos_range}"
        home_assists_against_string = f"22/23 Home Assists Against␣
↪{away_pos_range}"

        away_games_against_string = f"22/23 Away Games Against {home_pos_range}"
        away_goals_against_string = f"22/23 Away Goals Against {home_pos_range}"
        away_goals_conceded_against_string = f"22/23 Away Goals Conceded␣
↪Against {home_pos_range}"
        away_assists_against_string = f"22/23 Away Assists Against␣
↪{home_pos_range}"

        team_data[away_team_name][away_games_against_string] += 1
        team_data[away_team_name][away_goals_against_string] += away_goals
```

```
        team_data[away_team_name][away_goals_conceded_against_string] +=␣
↪home_goals

        team_data[home_team_name][home_games_against_string] += 1
        team_data[home_team_name][home_goals_against_string] += home_goals
        team_data[home_team_name][home_goals_conceded_against_string] +=␣
↪away_goals

        home_overall_elo = team_data[home_team_name]['ELO']
        away_overall_elo = team_data[away_team_name]['ELO']

        home_elo = team_data[home_team_name]['Home ELO']
        away_elo = team_data[away_team_name]['Away ELO']

        home_elo_22_23 = team_data[home_team_name]['Home ELO 22/23']
        away_elo_22_23 = team_data[away_team_name]['Away ELO 22/23']

        expected_home = 1 / (10 ** (-(home_elo - away_elo) / 400) + 1)
        expected_away = 1 / (10 ** (-(away_elo - home_elo) / 400) + 1)

        expected_home_22_23 = 1 / (10 ** (-(home_elo_22_23 - away_elo_22_23) /␣
↪400) + 1)
        expected_away_22_23 = 1 / (10 ** (-(away_elo_22_23 - home_elo_22_23) /␣
↪400) + 1)

        expected_home_overall = 1 / (10 ** (-(home_overall_elo -␣
↪away_overall_elo) / 400) + 1)
        expected_away_overall = 1 / (10 ** (-(away_overall_elo -␣
↪home_overall_elo) / 400) + 1)

        if home_goals > away_goals:
            actual_home = 1
            actual_away = 0
        elif home_goals < away_goals:
            actual_home = 0
            actual_away = 1
        else:
            actual_home = 0.5
            actual_away = 0.5

        # Calculate the margin of victory
        goal_difference = abs(home_goals - away_goals)
        margin_multiplier = 1.5 if goal_difference == 2 else 1.75 if␣
↪goal_difference == 3 else 1.75 + ((goal_difference - 3) / 8) if␣
↪goal_difference >= 4 else 1
```

```
        home_elo_change = k_factor * (actual_home - expected_home) *␣
↪margin_multiplier
        away_elo_change = k_factor * (actual_away - expected_away) *␣
↪margin_multiplier

        home_elo_change_22_23 = k_factor * (actual_home - expected_home_22_23)␣
↪* margin_multiplier
        away_elo_change_22_23 = k_factor * (actual_away - expected_away_22_23)␣
↪* margin_multiplier

        home_overall_elo_change = k_factor * (actual_home -␣
↪expected_home_overall) * margin_multiplier
        away_overall_elo_change = k_factor * (actual_away -␣
↪expected_away_overall) * margin_multiplier

        team_data[home_team_name]['Home ELO'] += home_elo_change
        team_data[away_team_name]['Away ELO'] += away_elo_change

        team_data[home_team_name]['Home ELO 22/23'] += home_elo_change_22_23
        team_data[away_team_name]['Away ELO 22/23'] += away_elo_change_22_23

        team_data[home_team_name]['ELO'] += home_overall_elo_change
        team_data[away_team_name]['ELO'] += away_overall_elo_change

        # Add values to both dictionaries by fixture
        for stat in fixture['stats']:
            if stat['identifier'] == 'bps':
                for pair in stat['a']:
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_22_23[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == away_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_22_23[pair['element']])):
                            player_data[player]['22/23 Away Games Played for␣
↪Current Team'] += 1
                            player_data[player]['22/23 BPS for Current Team']␣
↪+= int(pair['value'])
                            player_data[player][away_games_against_string] += 1

                for pair in stat['h']:
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_22_23[pair['element']]))) == None:
                        continue
                    for player in player_data:
```

```
                    if player_data[player]["Team"] == home_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_22_23[pair['element']])):
                        player_data[player]['22/23 Home Games Played for␣
↪Current Team'] += 1
                        player_data[player]['22/23 BPS for Current Team']␣
↪+= int(pair['value'])
                        player_data[player][home_games_against_string] += 1

            if stat['identifier'] == 'goals_scored':
                for pair in stat['a']:
                    team_data[away_team_name]['22/23 Away Goals'] +=␣
↪int(pair['value'])
                    team_data[home_team_name]['22/23 Goals Conceded Home'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_22_23[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == away_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_22_23[pair['element']])):
                            player_data[player]['22/23 Away Goals for Current␣
↪Team'] += int(pair['value'])
                            player_data[player][away_goals_against_string] +=␣
↪int(pair['value'])

                for pair in stat['h']:
                    team_data[home_team_name]['22/23 Home Goals'] +=␣
↪int(pair['value'])
                    team_data[away_team_name]['22/23 Goals Conceded Away'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_22_23[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == home_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_22_23[pair['element']])):
                            player_data[player]['22/23 Home Goals for Current␣
↪Team'] += int(pair['value'])
                            player_data[player][home_goals_against_string] +=␣
↪int(pair['value'])

            if stat['identifier'] == 'assists':
                for pair in stat['a']:
                    team_data[away_team_name]['22/23 Away Assists'] +=␣
↪int(pair['value'])
```

```
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_22_23[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == away_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_22_23[pair['element']])):
                            player_data[player]['22/23 Away Assists for Current␣
↪Team'] += int(pair['value'])
                            player_data[player][away_assists_against_string] +=␣
↪int(pair['value'])

                for pair in stat['h']:
                    team_data[home_team_name]['22/23 Home Assists'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_22_23[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == home_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_22_23[pair['element']])):
                            player_data[player]['22/23 Home Assists for Current␣
↪Team'] += int(pair['value'])
                            player_data[player][home_assists_against_string] +=␣
↪int(pair['value'])

            if stat['identifier'] == 'saves':
                for pair in stat['a']:
                    team_data[away_team_name]['22/23 Away Goalkeeper Saves'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_22_23[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == away_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_22_23[pair['element']])):
                            player_data[player]['22/23 Goalkeeper Saves for␣
↪Current Team'] += int(pair['value'])

                for pair in stat['h']:
                    team_data[home_team_name]['22/23 Home Goalkeeper Saves'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_22_23[pair['element']]))) == None:
                        continue
                    for player in player_data:
```

```
                    if player_data[player]["Team"] == home_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_22_23[pair['element']])):
                        player_data[player]['22/23 Goalkeeper Saves for␣
↪Current Team'] += int(pair['value'])

  for fixture in fixtures_23_24:
      home_team_id = int(fixture['team_h'])
      away_team_id = int(fixture['team_a'])
      home_team_name = TEAM_NAMES_ODDSCHECKER.
↪get(team_id_to_name_23_24[home_team_id], team_id_to_name_23_24[home_team_id])
      away_team_name = TEAM_NAMES_ODDSCHECKER.
↪get(team_id_to_name_23_24[away_team_id], team_id_to_name_23_24[away_team_id])
      home_pos_22_23 = season_22_23_team_positions.get(home_team_name, 21)
      away_pos_22_23 = season_22_23_team_positions.get(away_team_name, 21)
      home_pos_23_24 = season_23_24_team_positions.get(home_team_name, 21)
      away_pos_23_24 = season_23_24_team_positions.get(away_team_name, 21)
      team_data[home_team_name] = team_data.get(
          home_team_name, defaultdict(float,␣
↪get_team_template(home_pos_22_23, home_pos_23_24, 21))
      )
      team_data[away_team_name] = team_data.get(
          away_team_name, defaultdict(float,␣
↪get_team_template(away_pos_22_23, away_pos_23_24, 21))
      )

      # Ensure team_data always contains defaultdict(float)
      if not isinstance(team_data.get(home_team_name), defaultdict):
          team_data[home_team_name] = defaultdict(float,␣
↪team_data[home_team_name])
      if not isinstance(team_data.get(away_team_name), defaultdict):
          team_data[away_team_name] = defaultdict(float,␣
↪team_data[away_team_name])

      # Update ELO rankings
      home_goals = int(fixture['team_h_score'])
      away_goals = int(fixture['team_a_score'])

      home_pos_range = get_pos_range(home_pos_23_24)
      away_pos_range = get_pos_range(away_pos_23_24)

      home_games_against_string = f"23/24 Home Games Against {away_pos_range}"
      home_goals_against_string = f"23/24 Home Goals Against {away_pos_range}"
      home_goals_conceded_against_string = f"23/24 Home Goals Conceded␣
↪Against {away_pos_range}"
      home_assists_against_string = f"23/24 Home Assists Against␣
↪{away_pos_range}"
```

```
        away_games_against_string = f"23/24 Away Games Against {home_pos_range}"
        away_goals_against_string = f"23/24 Away Goals Against {home_pos_range}"
        away_goals_conceded_against_string = f"23/24 Away Goals Conceded␣
↪Against {home_pos_range}"
        away_assists_against_string = f"23/24 Away Assists Against␣
↪{home_pos_range}"

        team_data[away_team_name][away_games_against_string] += 1
        team_data[away_team_name][away_goals_against_string] += away_goals
        team_data[away_team_name][away_goals_conceded_against_string] +=␣
↪home_goals

        team_data[home_team_name][home_games_against_string] += 1
        team_data[home_team_name][home_goals_against_string] += home_goals
        team_data[home_team_name][home_goals_conceded_against_string] +=␣
↪away_goals

        home_overall_elo = team_data[home_team_name]['ELO']
        away_overall_elo = team_data[away_team_name]['ELO']

        home_elo = team_data[home_team_name]['Home ELO']
        away_elo = team_data[away_team_name]['Away ELO']

        home_elo_23_24 = team_data[home_team_name]['Home ELO 23/24']
        away_elo_23_24 = team_data[away_team_name]['Away ELO 23/24']

        expected_home = 1 / (10 ** (-(home_elo - away_elo) / 400) + 1)
        expected_away = 1 / (10 ** (-(away_elo - home_elo) / 400) + 1)

        expected_home_23_24 = 1 / (10 ** (-(home_elo_23_24 - away_elo_23_24) /␣
↪400) + 1)
        expected_away_23_24 = 1 / (10 ** (-(away_elo_23_24 - home_elo_23_24) /␣
↪400) + 1)

        expected_home_overall = 1 / (10 ** (-(home_overall_elo -␣
↪away_overall_elo) / 400) + 1)
        expected_away_overall = 1 / (10 ** (-(away_overall_elo -␣
↪home_overall_elo) / 400) + 1)

        if home_goals > away_goals:
            actual_home = 1
            actual_away = 0
        elif home_goals < away_goals:
            actual_home = 0
            actual_away = 1
```

```
        else:
            actual_home = 0.5
            actual_away = 0.5

        # Calculate the margin of victory
        goal_difference = abs(home_goals - away_goals)
        margin_multiplier = 1.5 if goal_difference == 2 else 1.75 if␣
↪goal_difference == 3 else 1.75 + ((goal_difference - 3) / 8) if␣
↪goal_difference >= 4 else 1

        home_elo_change = k_factor * (actual_home - expected_home) *␣
↪margin_multiplier
        away_elo_change = k_factor * (actual_away - expected_away) *␣
↪margin_multiplier

        home_elo_change_23_24 = k_factor * (actual_home - expected_home_23_24)␣
↪* margin_multiplier
        away_elo_change_23_24 = k_factor * (actual_away - expected_away_23_24)␣
↪* margin_multiplier

        home_overall_elo_change = k_factor * (actual_home -␣
↪expected_home_overall) * margin_multiplier
        away_overall_elo_change = k_factor * (actual_away -␣
↪expected_away_overall) * margin_multiplier

        team_data[home_team_name]['Home ELO'] += home_elo_change
        team_data[away_team_name]['Away ELO'] += away_elo_change

        team_data[home_team_name]['Home ELO 23/24'] += home_elo_change_23_24
        team_data[away_team_name]['Away ELO 23/24'] += away_elo_change_23_24

        team_data[home_team_name]['ELO'] += home_overall_elo_change
        team_data[away_team_name]['ELO'] += away_overall_elo_change

        # Add values to both dictionaries by fixture
        for stat in fixture['stats']:
            if stat['identifier'] == 'bps':
                for pair in stat['a']:
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_23_24[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == away_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_23_24[pair['element']])):
                            player_data[player]['23/24 Away Games Played for␣
↪Current Team'] += 1
```

```
                            player_data[player]['23/24 BPS for Current Team']␣
↪+= int(pair['value'])
                            player_data[player][away_games_against_string] += 1

                for pair in stat['h']:
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_23_24[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == home_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_23_24[pair['element']])):
                            player_data[player]['23/24 Home Games Played for␣
↪Current Team'] += 1
                            player_data[player]['23/24 BPS for Current Team']␣
↪+= int(pair['value'])
                            player_data[player][home_games_against_string] += 1

            if stat['identifier'] == 'goals_scored':
                for pair in stat['a']:
                    team_data[away_team_name]['23/24 Away Goals'] +=␣
↪int(pair['value'])
                    team_data[home_team_name]['23/24 Goals Conceded Home'] +=␣
↪int(pair['value'])

                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_23_24[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == away_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_23_24[pair['element']])):
                            player_data[player]['23/24 Away Goals for Current␣
↪Team'] += int(pair['value'])
                            player_data[player][away_goals_against_string] +=␣
↪int(pair['value'])

                for pair in stat['h']:
                    team_data[home_team_name]['23/24 Home Goals'] +=␣
↪int(pair['value'])
                    team_data[away_team_name]['23/24 Goals Conceded Away'] +=␣
↪int(pair['value'])

                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_23_24[pair['element']]))) == None:
                        continue
                    for player in player_data:
```

```
                    if player_data[player]["Team"] == home_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_23_24[pair['element']])):
                        player_data[player]['23/24 Home Goals for Current␣
↪Team'] += int(pair['value'])
                        player_data[player][home_goals_against_string] +=␣
↪int(pair['value'])

            if stat['identifier'] == 'assists':
                for pair in stat['a']:
                    team_data[away_team_name]['23/24 Away Assists'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_23_24[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == away_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_23_24[pair['element']])):
                        player_data[player]['23/24 Away Assists for Current␣
↪Team'] += int(pair['value'])
                        player_data[player][away_assists_against_string] +=␣
↪int(pair['value'])

                for pair in stat['h']:
                    team_data[home_team_name]['23/24 Home Assists'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_23_24[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == home_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_23_24[pair['element']])):
                        player_data[player]['23/24 Home Assists for Current␣
↪Team'] += int(pair['value'])
                        player_data[player][home_assists_against_string] +=␣
↪int(pair['value'])

            if stat['identifier'] == 'saves':
                for pair in stat['a']:
                    team_data[away_team_name]['23/24 Away Goalkeeper Saves'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_23_24[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == away_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_23_24[pair['element']])):
```

```
                            player_data[player]['23/24 Goalkeeper Saves for␣
↪Current Team'] += int(pair['value'])

                for pair in stat['h']:
                    team_data[home_team_name]['23/24 Home Goalkeeper Saves'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name_23_24[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == home_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name_23_24[pair['element']])):
                            player_data[player]['23/24 Goalkeeper Saves for␣
↪Current Team'] += int(pair['value'])

    # Process each gameweek
    for fixture in fixtures:
        home_team_id = int(fixture['team_h'])
        away_team_id = int(fixture['team_a'])
        home_team_name = TEAM_NAMES_ODDSCHECKER.
↪get(team_id_to_name[home_team_id], team_id_to_name[home_team_id])
        away_team_name = TEAM_NAMES_ODDSCHECKER.
↪get(team_id_to_name[away_team_id], team_id_to_name[away_team_id])
        home_pos = team_data[home_team_name]['League Position']
        away_pos = team_data[away_team_name]['League Position']
        # Update ELO rankings
        home_goals = fixture['team_h_score']
        away_goals = fixture['team_a_score']

        home_pos_range = get_pos_range(home_pos)
        away_pos_range = get_pos_range(away_pos)

        home_games_against_string = f"Home Games Against {away_pos_range}"
        home_goals_against_string = f"Home Goals Against {away_pos_range}"
        home_goals_conceded_against_string = f"Home Goals Conceded Against␣
↪{away_pos_range}"
        home_assists_against_string = f"Home Assists Against {away_pos_range}"

        away_games_against_string = f"Away Games Against {home_pos_range}"
        away_goals_against_string = f"Away Goals Against {home_pos_range}"
        away_goals_conceded_against_string = f"Away Goals Conceded Against␣
↪{home_pos_range}"
        away_assists_against_string = f"Away Assists Against {home_pos_range}"

        team_data[away_team_name][away_games_against_string] += 1
        team_data[away_team_name][away_goals_against_string] += away_goals
```

```
        team_data[away_team_name][away_goals_conceded_against_string] +=␣
↪home_goals

        team_data[home_team_name][home_games_against_string] += 1
        team_data[home_team_name][home_goals_against_string] += home_goals
        team_data[home_team_name][home_goals_conceded_against_string] +=␣
↪away_goals

        # Increment games played for both teams
        team_data[home_team_name]['Home Games Played'] += 1
        team_data[away_team_name]['Away Games Played'] += 1

        home_overall_elo = team_data[home_team_name]['ELO']
        away_overall_elo = team_data[away_team_name]['ELO']

        home_elo = team_data[home_team_name]['Home ELO']
        away_elo = team_data[away_team_name]['Away ELO']

        home_elo_24_25 = team_data[home_team_name]['Home ELO 24/25']
        away_elo_24_25 = team_data[away_team_name]['Away ELO 24/25']

        expected_home = 1 / (10 ** (-(home_elo - away_elo) / 400) + 1)
        expected_away = 1 / (10 ** (-(away_elo - home_elo) / 400) + 1)

        expected_home_24_25 = 1 / (10 ** (-(home_elo_24_25 - away_elo_24_25) /␣
↪400) + 1)
        expected_away_24_25 = 1 / (10 ** (-(away_elo_24_25 - home_elo_24_25) /␣
↪400) + 1)

        expected_home_overall = 1 / (10 ** (-(home_overall_elo -␣
↪away_overall_elo) / 400) + 1)
        expected_away_overall = 1 / (10 ** (-(away_overall_elo -␣
↪home_overall_elo) / 400) + 1)

        if home_goals > away_goals:
            actual_home = 1
            actual_away = 0
        elif home_goals < away_goals:
            actual_home = 0
            actual_away = 1
        else:
            actual_home = 0.5
            actual_away = 0.5

        # Calculate the margin of victory
        goal_difference = abs(home_goals - away_goals)
```

```
        margin_multiplier = 1.5 if goal_difference == 2 else 1.75 if␣
↪goal_difference == 3 else 1.75 + ((goal_difference - 3) / 8) if␣
↪goal_difference >= 4 else 1

        home_elo_change = k_factor * (actual_home - expected_home) *␣
↪margin_multiplier
        away_elo_change = k_factor * (actual_away - expected_away) *␣
↪margin_multiplier

        home_elo_change_24_25 = k_factor * (actual_home - expected_home_24_25)␣
↪* margin_multiplier
        away_elo_change_24_25 = k_factor * (actual_away - expected_away_24_25)␣
↪* margin_multiplier

        home_overall_elo_change = k_factor * (actual_home -␣
↪expected_home_overall) * margin_multiplier
        away_overall_elo_change = k_factor * (actual_away -␣
↪expected_away_overall) * margin_multiplier

        team_data[home_team_name]['Home ELO'] += home_elo_change
        team_data[away_team_name]['Away ELO'] += away_elo_change

        team_data[home_team_name]['Home ELO 24/25'] += home_elo_change_24_25
        team_data[away_team_name]['Away ELO 24/25'] += away_elo_change_24_25

        team_data[home_team_name]['ELO'] += home_overall_elo_change
        team_data[away_team_name]['ELO'] += away_overall_elo_change

        # Add values to both dictionaries by fixture
        for stat in fixture['stats']:
            if stat['identifier'] == 'bps':
                for pair in stat['a']:
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == away_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name[pair['element']])):
                            player_data[player]['Away Games Played for Current␣
↪Team'] += 1
                            player_data[player]['BPS for Current Team'] +=␣
↪int(pair['value'])
                            player_data[player][away_games_against_string] += 1
                for pair in stat['h']:
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name[pair['element']]))) == None:
```

```
                    continue
                for player in player_data:
                    if player_data[player]["Team"] == home_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name[pair['element']])):
                        player_data[player]['Home Games Played for Current␣
↪Team'] += 1
                        player_data[player]['BPS for Current Team'] +=␣
↪int(pair['value'])
                        player_data[player][home_games_against_string] += 1

            if stat['identifier'] == 'goals_scored':
                for pair in stat['a']:
                    team_data[away_team_name]['Away Goals'] +=␣
↪int(pair['value'])
                    team_data[home_team_name]['Goals Conceded Home'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == away_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name[pair['element']])):
                            player_data[player]['Away Goals for Current Team']␣
↪+= int(pair['value'])
                            player_data[player][away_goals_against_string] +=␣
↪int(pair['value'])
                for pair in stat['h']:
                    team_data[home_team_name]['Home Goals'] +=␣
↪int(pair['value'])
                    team_data[away_team_name]['Goals Conceded Away'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == home_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name[pair['element']])):
                            player_data[player]['Home Goals for Current Team']␣
↪+= int(pair['value'])
                            player_data[player][home_goals_against_string] +=␣
↪int(pair['value'])
            if stat['identifier'] == 'assists':
                for pair in stat['a']:
                    team_data[away_team_name]['Away Assists'] +=␣
↪int(pair['value'])
```

```
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == away_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name[pair['element']])):
                            player_data[player]['Away Assists for Current␣
↪Team'] += int(pair['value'])
                            player_data[player][away_assists_against_string] +=␣
↪int(pair['value'])
                for pair in stat['h']:
                    team_data[home_team_name]['Home Assists'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == home_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name[pair['element']])):
                            player_data[player]['Home Assists for Current␣
↪Team'] += int(pair['value'])
                            player_data[player][home_assists_against_string] +=␣
↪int(pair['value'])
            if stat['identifier'] == 'saves':
                for pair in stat['a']:
                    team_data[away_team_name]['Away Goalkeeper Saves'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == away_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name[pair['element']])):
                            player_data[player]['Goalkeeper Saves for Current␣
↪Team'] += int(pair['value'])
                for pair in stat['h']:
                    team_data[home_team_name]['Home Goalkeeper Saves'] +=␣
↪int(pair['value'])
                    if player_data.get(" ".
↪join(prepare_name(player_id_to_name[pair['element']]))) == None:
                        continue
                    for player in player_data:
                        if player_data[player]["Team"] == home_team_name and␣
↪player == " ".join(prepare_name(player_id_to_name[pair['element']])):
                            player_data[player]['Goalkeeper Saves for Current␣
↪Team'] += int(pair['value'])
```

```
  for team in team_data:
      team_data[team]['HFA'] = float(team_data[team]['Home ELO'] -␣
↪team_data[team]['Away ELO']) if team_data[team]['Away ELO'] != 0 else 0

      team_data[team]['Goalkeeper Saves per Home Game'] =␣
↪float(team_data[team]['Home Goalkeeper Saves']/team_data[team]['Home Games␣
↪Played']) if team_data[team]['Home Games Played'] != 0 else 0
      team_data[team]['Goalkeeper Saves per Away Game'] =␣
↪float(team_data[team]['Away Goalkeeper Saves']/team_data[team]['Away Games␣
↪Played']) if team_data[team]['Away Games Played'] != 0 else 0
      team_data[team]['Goals per Game'] = float((team_data[team]['Home␣
↪Goals'] + team_data[team]['Away Goals'])/(team_data[team]['Home Games␣
↪Played'] + team_data[team]['Away Games Played'])) if (team_data[team]['Home␣
↪Games Played'] + team_data[team]['Away Games Played']) != 0 else 0
      team_data[team]['Goals per Home Game'] = float(team_data[team]['Home␣
↪Goals']/team_data[team]['Home Games Played']) if team_data[team]['Home Games␣
↪Played'] != 0 else 0
      team_data[team]['Goals per Away Game'] = float(team_data[team]['Away␣
↪Goals']/team_data[team]['Away Games Played']) if team_data[team]['Away Games␣
↪Played'] != 0 else 0
      team_data[team]['Goals Conceded per Game'] =␣
↪float((team_data[team]['Goals Conceded Home'] + team_data[team]['Goals␣
↪Conceded Away'])/(team_data[team]['Home Games Played'] +␣
↪team_data[team]['Away Games Played'])) if (team_data[team]['Home Games␣
↪Played'] + team_data[team]['Away Games Played']) != 0 else 0
      team_data[team]['Goals Conceded per Home Game'] =␣
↪float(team_data[team]['Goals Conceded Home']/team_data[team]['Home Games␣
↪Played']) if team_data[team]['Home Games Played'] != 0 else 0
      team_data[team]['Goals Conceded per Away Game'] =␣
↪float(team_data[team]['Goals Conceded Away']/team_data[team]['Away Games␣
↪Played']) if team_data[team]['Away Games Played'] != 0 else 0

      team_data[team]['Goals per Home Game Against 1-5'] =␣
↪float(team_data[team]['Home Goals Against 1-5']/team_data[team]['Home Games␣
↪Against 1-5']) if team_data[team]['Home Games Against 1-5'] != 0 else 0
      team_data[team]['Goals Conceded per Home Game Against 1-5'] =␣
↪float(team_data[team]['Home Goals Conceded Against 1-5']/
↪team_data[team]['Home Games Against 1-5']) if team_data[team]['Home Games␣
↪Against 1-5'] != 0 else 0
      team_data[team]['Goals per Home Game Against 6-10'] =␣
↪float(team_data[team]['Home Goals Against 6-10']/team_data[team]['Home Games␣
↪Against 6-10']) if team_data[team]['Home Games Against 6-10'] != 0 else 0
```

```
        team_data[team]['Goals Conceded per Home Game Against 6-10'] =␣
↪float(team_data[team]['Home Goals Conceded Against 6-10']/
↪team_data[team]['Home Games Against 6-10']) if team_data[team]['Home Games␣
↪Against 6-10'] != 0 else 0
        team_data[team]['Goals per Home Game Against 11-15'] =␣
↪float(team_data[team]['Home Goals Against 11-15']/team_data[team]['Home␣
↪Games Against 11-15']) if team_data[team]['Home Games Against 11-15'] != 0␣
↪else 0
        team_data[team]['Goals Conceded per Home Game Against 11-15'] =␣
↪float(team_data[team]['Home Goals Conceded Against 11-15']/
↪team_data[team]['Home Games Against 11-15']) if team_data[team]['Home Games␣
↪Against 11-15'] != 0 else 0
        team_data[team]['Goals per Home Game Against 16-20'] =␣
↪float(team_data[team]['Home Goals Against 16-20']/team_data[team]['Home␣
↪Games Against 16-20']) if team_data[team]['Home Games Against 16-20'] != 0␣
↪else 0
        team_data[team]['Goals Conceded per Home Game Against 16-20'] =␣
↪float(team_data[team]['Home Goals Conceded Against 16-20']/
↪team_data[team]['Home Games Against 16-20']) if team_data[team]['Home Games␣
↪Against 16-20'] != 0 else 0

        team_data[team]['Goals per Away Game Against 1-5'] =␣
↪float(team_data[team]['Away Goals Against 1-5']/team_data[team]['Away Games␣
↪Against 1-5']) if team_data[team]['Away Games Against 1-5'] != 0 else 0
        team_data[team]['Goals Conceded per Away Game Against 1-5'] =␣
↪float(team_data[team]['Away Goals Conceded Against 1-5']/
↪team_data[team]['Away Games Against 1-5']) if team_data[team]['Away Games␣
↪Against 1-5'] != 0 else 0
        team_data[team]['Goals per Away Game Against 6-10'] =␣
↪float(team_data[team]['Away Goals Against 6-10']/team_data[team]['Away Games␣
↪Against 6-10']) if team_data[team]['Away Games Against 6-10'] != 0 else 0
        team_data[team]['Goals Conceded per Away Game Against 6-10'] =␣
↪float(team_data[team]['Away Goals Conceded Against 6-10']/
↪team_data[team]['Away Games Against 6-10']) if team_data[team]['Away Games␣
↪Against 6-10'] != 0 else 0
        team_data[team]['Goals per Away Game Against 11-15'] =␣
↪float(team_data[team]['Away Goals Against 11-15']/team_data[team]['Away␣
↪Games Against 11-15']) if team_data[team]['Away Games Against 11-15'] != 0␣
↪else 0
        team_data[team]['Goals Conceded per Away Game Against 11-15'] =␣
↪float(team_data[team]['Away Goals Conceded Against 11-15']/
↪team_data[team]['Away Games Against 11-15']) if team_data[team]['Away Games␣
↪Against 11-15'] != 0 else 0
```

```
        team_data[team]['Goals per Away Game Against 16-20'] =␣
↪float(team_data[team]['Away Goals Against 16-20']/team_data[team]['Away␣
↪Games Against 16-20']) if team_data[team]['Away Games Against 16-20'] != 0␣
↪else 0
        team_data[team]['Goals Conceded per Away Game Against 16-20'] =␣
↪float(team_data[team]['Away Goals Conceded Against 16-20']/
↪team_data[team]['Away Games Against 16-20']) if team_data[team]['Away Games␣
↪Against 16-20'] != 0 else 0

        team_data[team]['22/23 Goalkeeper Saves per Home Game'] =␣
↪float(team_data[team]['22/23 Home Goalkeeper Saves']/19)
        team_data[team]['22/23 Goalkeeper Saves per Away Game'] =␣
↪float(team_data[team]['22/23 Away Goalkeeper Saves']/19)
        team_data[team]['22/23 Goals per Home Game'] =␣
↪float(team_data[team]['22/23 Home Goals']/19)
        team_data[team]['22/23 Goals per Away Game'] =␣
↪float(team_data[team]['22/23 Away Goals']/19)
        team_data[team]['22/23 Goals Conceded per Home Game'] =␣
↪float(team_data[team]['22/23 Goals Conceded Home']/19)
        team_data[team]['22/23 Goals Conceded per Away Game'] =␣
↪float(team_data[team]['22/23 Goals Conceded Away']/19)

        team_data[team]['22/23 Goals per Home Game Against 1-5'] =␣
↪float(team_data[team]['22/23 Home Goals Against 1-5']/team_data[team]['22/23␣
↪Home Games Against 1-5']) if team_data[team]['22/23 Home Games Against 1-5']␣
↪> 0 else 0
        team_data[team]['22/23 Goals Conceded per Home Game Against 1-5'] =␣
↪float(team_data[team]['22/23 Home Goals Conceded Against 1-5']/
↪team_data[team]['22/23 Home Games Against 1-5']) if team_data[team]['22/23␣
↪Home Games Against 1-5'] > 0 else 0
        team_data[team]['22/23 Goals per Home Game Against 6-10'] =␣
↪float(team_data[team]['22/23 Home Goals Against 6-10']/team_data[team]['22/
↪23 Home Games Against 6-10']) if team_data[team]['22/23 Home Games Against␣
↪6-10'] > 0 else 0
        team_data[team]['22/23 Goals Conceded per Home Game Against 6-10'] =␣
↪float(team_data[team]['22/23 Home Goals Conceded Against 6-10']/
↪team_data[team]['22/23 Home Games Against 6-10']) if team_data[team]['22/23␣
↪Home Games Against 6-10'] > 0 else 0
        team_data[team]['22/23 Goals per Home Game Against 11-15'] =␣
↪float(team_data[team]['22/23 Home Goals Against 11-15']/team_data[team]['22/
↪23 Home Games Against 11-15']) if team_data[team]['22/23 Home Games Against␣
↪11-15'] > 0 else 0
        team_data[team]['22/23 Goals Conceded per Home Game Against 11-15'] =␣
↪float(team_data[team]['22/23 Home Goals Conceded Against 11-15']/
↪team_data[team]['22/23 Home Games Against 11-15']) if team_data[team]['22/23␣
↪Home Games Against 11-15'] > 0 else 0
```

```
        team_data[team]['22/23 Goals per Home Game Against 16-20'] =␣
↪float(team_data[team]['22/23 Home Goals Against 16-20']/team_data[team]['22/
↪23 Home Games Against 16-20']) if team_data[team]['22/23 Home Games Against␣
↪16-20'] > 0 else 0
        team_data[team]['22/23 Goals Conceded per Home Game Against 16-20'] =␣
↪float(team_data[team]['22/23 Home Goals Conceded Against 16-20']/
↪team_data[team]['22/23 Home Games Against 16-20']) if team_data[team]['22/23␣
↪Home Games Against 16-20'] > 0 else 0

        team_data[team]['22/23 Goals per Away Game Against 1-5'] =␣
↪float(team_data[team]['22/23 Away Goals Against 1-5']/team_data[team]['22/23␣
↪Away Games Against 1-5']) if team_data[team]['22/23 Away Games Against 1-5']␣
↪> 0 else 0
        team_data[team]['22/23 Goals Conceded per Away Game Against 1-5'] =␣
↪float(team_data[team]['22/23 Away Goals Conceded Against 1-5']/
↪team_data[team]['22/23 Away Games Against 1-5']) if team_data[team]['22/23␣
↪Away Games Against 1-5'] > 0 else 0
        team_data[team]['22/23 Goals per Away Game Against 6-10'] =␣
↪float(team_data[team]['22/23 Away Goals Against 6-10']/team_data[team]['22/
↪23 Away Games Against 6-10']) if team_data[team]['22/23 Away Games Against␣
↪6-10'] > 0 else 0
        team_data[team]['22/23 Goals Conceded per Away Game Against 6-10'] =␣
↪float(team_data[team]['22/23 Away Goals Conceded Against 6-10']/
↪team_data[team]['22/23 Away Games Against 6-10']) if team_data[team]['22/23␣
↪Away Games Against 6-10'] > 0 else 0
        team_data[team]['22/23 Goals per Away Game Against 11-15'] =␣
↪float(team_data[team]['22/23 Away Goals Against 11-15']/team_data[team]['22/
↪23 Away Games Against 11-15']) if team_data[team]['22/23 Away Games Against␣
↪11-15'] > 0 else 0
        team_data[team]['22/23 Goals Conceded per Away Game Against 11-15'] =␣
↪float(team_data[team]['22/23 Away Goals Conceded Against 11-15']/
↪team_data[team]['22/23 Away Games Against 11-15']) if team_data[team]['22/23␣
↪Away Games Against 11-15'] > 0 else 0
        team_data[team]['22/23 Goals per Away Game Against 16-20'] =␣
↪float(team_data[team]['22/23 Away Goals Against 16-20']/team_data[team]['22/
↪23 Away Games Against 16-20']) if team_data[team]['22/23 Away Games Against␣
↪16-20'] > 0 else 0
        team_data[team]['22/23 Goals Conceded per Away Game Against 16-20'] =␣
↪float(team_data[team]['22/23 Away Goals Conceded Against 16-20']/
↪team_data[team]['22/23 Away Games Against 16-20']) if team_data[team]['22/23␣
↪Away Games Against 16-20'] > 0 else 0

        team_data[team]['23/24 Goalkeeper Saves per Home Game'] =␣
↪float(team_data[team]['23/24 Home Goalkeeper Saves']/19)
        team_data[team]['23/24 Goalkeeper Saves per Away Game'] =␣
↪float(team_data[team]['23/24 Away Goalkeeper Saves']/19)
```

```
        team_data[team]['23/24 Goals per Home Game'] =␣
↪float(team_data[team]['23/24 Home Goals']/19)
        team_data[team]['23/24 Goals per Away Game'] =␣
↪float(team_data[team]['23/24 Away Goals']/19)
        team_data[team]['23/24 Goals Conceded per Home Game'] =␣
↪float(team_data[team]['23/24 Goals Conceded Home']/19)
        team_data[team]['23/24 Goals Conceded per Away Game'] =␣
↪float(team_data[team]['23/24 Goals Conceded Away']/19)

        team_data[team]['23/24 Goals per Home Game Against 1-5'] =␣
↪float(team_data[team]['23/24 Home Goals Against 1-5']/team_data[team]['23/24␣
↪Home Games Against 1-5']) if team_data[team]['23/24 Home Games Against 1-5']␣
↪> 0 else 0
        team_data[team]['23/24 Goals Conceded per Home Game Against 1-5'] =␣
↪float(team_data[team]['23/24 Home Goals Conceded Against 1-5']/
↪team_data[team]['23/24 Home Games Against 1-5']) if team_data[team]['23/24␣
↪Home Games Against 1-5'] > 0 else 0
        team_data[team]['23/24 Goals per Home Game Against 6-10'] =␣
↪float(team_data[team]['23/24 Home Goals Against 6-10']/team_data[team]['23/
↪24 Home Games Against 6-10']) if team_data[team]['23/24 Home Games Against␣
↪6-10'] > 0 else 0
        team_data[team]['23/24 Goals Conceded per Home Game Against 6-10'] =␣
↪float(team_data[team]['23/24 Home Goals Conceded Against 6-10']/
↪team_data[team]['23/24 Home Games Against 6-10']) if team_data[team]['23/24␣
↪Home Games Against 6-10'] > 0 else 0
        team_data[team]['23/24 Goals per Home Game Against 11-15'] =␣
↪float(team_data[team]['23/24 Home Goals Against 11-15']/team_data[team]['23/
↪24 Home Games Against 11-15']) if team_data[team]['23/24 Home Games Against␣
↪11-15'] > 0 else 0
        team_data[team]['23/24 Goals Conceded per Home Game Against 11-15'] =␣
↪float(team_data[team]['23/24 Home Goals Conceded Against 11-15']/
↪team_data[team]['23/24 Home Games Against 11-15'])if team_data[team]['23/24␣
↪Home Games Against 11-15'] > 0 else 0
        team_data[team]['23/24 Goals per Home Game Against 16-20'] =␣
↪float(team_data[team]['23/24 Home Goals Against 16-20']/team_data[team]['23/
↪24 Home Games Against 16-20']) if team_data[team]['23/24 Home Games Against␣
↪16-20'] > 0 else 0
        team_data[team]['23/24 Goals Conceded per Home Game Against 16-20'] =␣
↪float(team_data[team]['23/24 Home Goals Conceded Against 16-20']/
↪team_data[team]['23/24 Home Games Against 16-20']) if team_data[team]['23/24␣
↪Home Games Against 16-20'] > 0 else 0

        team_data[team]['23/24 Goals per Away Game Against 1-5'] =␣
↪float(team_data[team]['23/24 Away Goals Against 1-5']/team_data[team]['23/24␣
↪Away Games Against 1-5']) if team_data[team]['23/24 Away Games Against 1-5']␣
↪> 0 else 0
```

```
        team_data[team]['23/24 Goals Conceded per Away Game Against 1-5'] =␣
↪float(team_data[team]['23/24 Away Goals Conceded Against 1-5']/
↪team_data[team]['23/24 Away Games Against 1-5']) if team_data[team]['23/24␣
↪Away Games Against 1-5'] > 0 else 0
        team_data[team]['23/24 Goals per Away Game Against 6-10'] =␣
↪float(team_data[team]['23/24 Away Goals Against 6-10']/team_data[team]['23/
↪24 Away Games Against 6-10']) if team_data[team]['23/24 Away Games Against␣
↪6-10'] > 0 else 0
        team_data[team]['23/24 Goals Conceded per Away Game Against 6-10'] =␣
↪float(team_data[team]['23/24 Away Goals Conceded Against 6-10']/
↪team_data[team]['23/24 Away Games Against 6-10']) if team_data[team]['23/24␣
↪Away Games Against 6-10'] > 0 else 0
        team_data[team]['23/24 Goals per Away Game Against 11-15'] =␣
↪float(team_data[team]['23/24 Away Goals Against 11-15']/team_data[team]['23/
↪24 Away Games Against 11-15']) if team_data[team]['23/24 Away Games Against␣
↪11-15'] > 0 else 0
        team_data[team]['23/24 Goals Conceded per Away Game Against 11-15'] =␣
↪float(team_data[team]['23/24 Away Goals Conceded Against 11-15']/
↪team_data[team]['23/24 Away Games Against 11-15']) if team_data[team]['23/24␣
↪Away Games Against 11-15'] > 0 else 0
        team_data[team]['23/24 Goals per Away Game Against 16-20'] =␣
↪float(team_data[team]['23/24 Away Goals Against 16-20']/team_data[team]['23/
↪24 Away Games Against 16-20']) if team_data[team]['23/24 Away Games Against␣
↪16-20'] > 0 else 0
        team_data[team]['23/24 Goals Conceded per Away Game Against 16-20'] =␣
↪float(team_data[team]['23/24 Away Goals Conceded Against 16-20']/
↪team_data[team]['23/24 Away Games Against 16-20']) if team_data[team]['23/24␣
↪Away Games Against 16-20'] > 0 else 0

    for player in player_data:
        games_played = max((player_data[player]['Home Games Played for Current␣
↪Team'] + player_data[player]['Away Games Played for Current Team']),␣
↪player_data[player]['Starts'])
        player_data[player]['Minutes per Game'] =␣
↪float(player_data[player]['Minutes']/games_played) if games_played != 0 else␣
↪0

        player_data[player]['Goals per Home Game'] =␣
↪float(player_data[player]['Home Goals for Current Team']/
↪player_data[player]['Home Games Played for Current Team']) if␣
↪player_data[player]['Home Games Played for Current Team'] != 0 else 0
        player_data[player]['Goals per Home Game Against 1-5'] =␣
↪float(player_data[player]['Home Goals Against 1-5']/
↪player_data[player]['Home Games Against 1-5']) if player_data[player]['Home␣
↪Games Against 1-5'] != 0 else 0
```

```
        player_data[player]['Assists per Home Game'] =␣
↪float(player_data[player]['Home Assists for Current Team']/
↪player_data[player]['Home Games Played for Current Team']) if␣
↪player_data[player]['Home Games Played for Current Team'] != 0 else 0
        player_data[player]['Assists per Home Game Against 1-5'] =␣
↪float(player_data[player]['Home Assists Against 1-5']/
↪player_data[player]['Home Games Against 1-5']) if player_data[player]['Home␣
↪Games Against 1-5'] != 0 else 0
        player_data[player]['Goals per Home Game Against 6-10'] =␣
↪float(player_data[player]['Home Goals Against 6-10']/
↪player_data[player]['Home Games Against 6-10']) if player_data[player]['Home␣
↪Games Against 6-10'] != 0 else 0
        player_data[player]['Assists per Home Game Against 6-10'] =␣
↪float(player_data[player]['Home Assists Against 6-10']/
↪player_data[player]['Home Games Against 6-10']) if player_data[player]['Home␣
↪Games Against 6-10'] != 0 else 0
        player_data[player]['Goals per Home Game Against 11-15'] =␣
↪float(player_data[player]['Home Goals Against 11-15']/
↪player_data[player]['Home Games Against 11-15']) if␣
↪player_data[player]['Home Games Against 11-15'] != 0 else 0
        player_data[player]['Assists per Home Game Against 11-15'] =␣
↪float(player_data[player]['Home Assists Against 11-15']/
↪player_data[player]['Home Games Against 11-15']) if␣
↪player_data[player]['Home Games Against 11-15'] != 0 else 0
        player_data[player]['Goals per Home Game Against 16-20'] =␣
↪float(player_data[player]['Home Goals Against 16-20']/
↪player_data[player]['Home Games Against 16-20']) if␣
↪player_data[player]['Home Games Against 16-20'] != 0 else 0
        player_data[player]['Assists per Home Game Against 16-20'] =␣
↪float(player_data[player]['Home Assists Against 16-20']/
↪player_data[player]['Home Games Against 16-20']) if␣
↪player_data[player]['Home Games Against 16-20'] != 0 else 0

        player_data[player]['Goals per Away Game'] =␣
↪float(player_data[player]['Away Goals for Current Team']/
↪player_data[player]['Away Games Played for Current Team']) if␣
↪player_data[player]['Away Games Played for Current Team'] != 0 else 0
        player_data[player]['Goals per Away Game Against 1-5'] =␣
↪float(player_data[player]['Away Goals Against 1-5']/
↪player_data[player]['Away Games Against 1-5']) if player_data[player]['Away␣
↪Games Against 1-5'] != 0 else 0
        player_data[player]['Assists per Away Game'] =␣
↪float(player_data[player]['Away Assists for Current Team']/
↪player_data[player]['Away Games Played for Current Team']) if␣
↪player_data[player]['Away Games Played for Current Team'] != 0 else 0
```

```
        player_data[player]['Assists per Away Game Against 1-5'] =␣
↪float(player_data[player]['Away Assists Against 1-5']/
↪player_data[player]['Away Games Against 1-5']) if player_data[player]['Away␣
↪Games Against 1-5'] != 0 else 0
        player_data[player]['Goals per Away Game Against 6-10'] =␣
↪float(player_data[player]['Away Goals Against 6-10']/
↪player_data[player]['Away Games Against 6-10']) if player_data[player]['Away␣
↪Games Against 6-10'] != 0 else 0
        player_data[player]['Assists per Away Game Against 6-10'] =␣
↪float(player_data[player]['Away Assists Against 6-10']/
↪player_data[player]['Away Games Against 6-10']) if player_data[player]['Away␣
↪Games Against 6-10'] != 0 else 0
        player_data[player]['Goals per Away Game Against 11-15'] =␣
↪float(player_data[player]['Away Goals Against 11-15']/
↪player_data[player]['Away Games Against 11-15']) if␣
↪player_data[player]['Away Games Against 11-15'] != 0 else 0
        player_data[player]['Assists per Away Game Against 11-15'] =␣
↪float(player_data[player]['Away Assists Against 11-15']/
↪player_data[player]['Away Games Against 11-15']) if␣
↪player_data[player]['Away Games Against 11-15'] != 0 else 0
        player_data[player]['Goals per Away Game Against 16-20'] =␣
↪float(player_data[player]['Away Goals Against 16-20']/
↪player_data[player]['Away Games Against 16-20']) if␣
↪player_data[player]['Away Games Against 16-20'] != 0 else 0
        player_data[player]['Assists per Away Game Against 16-20'] =␣
↪float(player_data[player]['Away Assists Against 16-20']/
↪player_data[player]['Away Games Against 16-20']) if␣
↪player_data[player]['Away Games Against 16-20'] != 0 else 0

        player_data[player]['Average BPS per Game'] =␣
↪float(player_data[player]['BPS for Current Team']/(player_data[player]['Home␣
↪Games Played for Current Team'] + player_data[player]['Away Games Played for␣
↪Current Team'])) if (player_data[player]['Home Games Played for Current␣
↪Team'] + player_data[player]['Away Games Played for Current Team']) != 0␣
↪else 0

        player_data[player]['22/23 Goals per Home Game'] =␣
↪float(player_data[player]['22/23 Home Goals for Current Team']/
↪player_data[player]['22/23 Home Games Played for Current Team']) if␣
↪player_data[player]['22/23 Home Games Played for Current Team'] != 0 else 0
        player_data[player]['22/23 Goals per Home Game Against 1-5'] =␣
↪float(player_data[player]['22/23 Home Goals Against 1-5']/
↪player_data[player]['22/23 Home Games Against 1-5']) if␣
↪player_data[player]['22/23 Home Games Against 1-5'] != 0 else 0
```

```
        player_data[player]['22/23 Assists per Home Game'] =␣
↪float(player_data[player]['22/23 Home Assists for Current Team']/
↪player_data[player]['22/23 Home Games Played for Current Team']) if␣
↪player_data[player]['22/23 Home Games Played for Current Team'] != 0 else 0
        player_data[player]['22/23 Assists per Home Game Against 1-5'] =␣
↪float(player_data[player]['22/23 Home Assists Against 1-5']/
↪player_data[player]['22/23 Home Games Against 1-5']) if␣
↪player_data[player]['22/23 Home Games Against 1-5'] != 0 else 0
        player_data[player]['22/23 Goals per Home Game Against 6-10'] =␣
↪float(player_data[player]['22/23 Home Goals Against 6-10']/
↪player_data[player]['22/23 Home Games Against 6-10']) if␣
↪player_data[player]['22/23 Home Games Against 6-10'] != 0 else 0
        player_data[player]['22/23 Assists per Home Game Against 6-10'] =␣
↪float(player_data[player]['22/23 Home Assists Against 6-10']/
↪player_data[player]['22/23 Home Games Against 6-10']) if␣
↪player_data[player]['22/23 Home Games Against 6-10'] != 0 else 0
        player_data[player]['22/23 Goals per Home Game Against 11-15'] =␣
↪float(player_data[player]['22/23 Home Goals Against 11-15']/
↪player_data[player]['22/23 Home Games Against 11-15']) if␣
↪player_data[player]['22/23 Home Games Against 11-15'] != 0 else 0
        player_data[player]['22/23 Assists per Home Game Against 11-15'] =␣
↪float(player_data[player]['22/23 Home Assists Against 11-15']/
↪player_data[player]['22/23 Home Games Against 11-15']) if␣
↪player_data[player]['22/23 Home Games Against 11-15'] != 0 else 0
        player_data[player]['22/23 Goals per Home Game Against 16-20'] =␣
↪float(player_data[player]['22/23 Home Goals Against 16-20']/
↪player_data[player]['22/23 Home Games Against 16-20']) if␣
↪player_data[player]['22/23 Home Games Against 16-20'] != 0 else 0
        player_data[player]['22/23 Assists Against 16-20'] =␣
↪float(player_data[player]['22/23 Home Assists Against 16-20']/
↪player_data[player]['22/23 Home Games Against 16-20']) if␣
↪player_data[player]['22/23 Home Games Against 16-20'] != 0 else 0

        player_data[player]['22/23 Goals per Away Game'] =␣
↪float(player_data[player]['22/23 Away Goals for Current Team']/
↪player_data[player]['22/23 Away Games Played for Current Team']) if␣
↪player_data[player]['22/23 Away Games Played for Current Team'] != 0 else 0
        player_data[player]['22/23 Goals per Away Game Against 1-5'] =␣
↪float(player_data[player]['22/23 Away Goals Against 1-5']/
↪player_data[player]['22/23 Away Games Against 1-5']) if␣
↪player_data[player]['22/23 Away Games Against 1-5'] != 0 else 0
        player_data[player]['22/23 Assists per Away Game'] =␣
↪float(player_data[player]['22/23 Away Assists for Current Team']/
↪player_data[player]['22/23 Away Games Played for Current Team']) if␣
↪player_data[player]['22/23 Away Games Played for Current Team'] != 0 else 0
```

```
        player_data[player]['22/23 Assists per Away Game Against 1-5'] =␣
↪float(player_data[player]['22/23 Away Assists Against 1-5']/
↪player_data[player]['22/23 Away Games Against 1-5']) if␣
↪player_data[player]['22/23 Away Games Against 1-5'] != 0 else 0
        player_data[player]['22/23 Goals per Away Game Against 6-10'] =␣
↪float(player_data[player]['22/23 Away Goals Against 6-10']/
↪player_data[player]['22/23 Away Games Against 6-10']) if␣
↪player_data[player]['22/23 Away Games Against 6-10'] != 0 else 0
        player_data[player]['22/23 Assists per Away Game Against 6-10'] =␣
↪float(player_data[player]['22/23 Away Assists Against 6-10']/
↪player_data[player]['22/23 Away Games Against 6-10']) if␣
↪player_data[player]['22/23 Away Games Against 6-10'] != 0 else 0
        player_data[player]['22/23 Goals per Away Game Against 11-15'] =␣
↪float(player_data[player]['22/23 Away Goals Against 11-15']/
↪player_data[player]['22/23 Away Games Against 11-15']) if␣
↪player_data[player]['22/23 Away Games Against 11-15'] != 0 else 0
        player_data[player]['22/23 Assists per Away Game Against 11-15'] =␣
↪float(player_data[player]['22/23 Away Assists Against 11-15']/
↪player_data[player]['22/23 Away Games Against 11-15']) if␣
↪player_data[player]['22/23 Away Games Against 11-15'] != 0 else 0
        player_data[player]['22/23 Goals per Away Game Against 16-20'] =␣
↪float(player_data[player]['22/23 Away Goals Against 16-20']/
↪player_data[player]['22/23 Away Games Against 16-20']) if␣
↪player_data[player]['22/23 Away Games Against 16-20'] != 0 else 0
        player_data[player]['22/23 Assists per Away Game Against 16-20'] =␣
↪float(player_data[player]['22/23 Away Assists Against 16-20']/
↪player_data[player]['22/23 Away Games Against 16-20']) if␣
↪player_data[player]['22/23 Away Games Against 16-20'] != 0 else 0

        player_data[player]['22/23 Average BPS per Game'] =␣
↪float(player_data[player]['22/23 BPS for Current Team']/
↪(player_data[player]['22/23 Home Games Played for Current Team'] +␣
↪player_data[player]['22/23 Away Games Played for Current Team'])) if␣
↪(player_data[player]['22/23 Home Games Played for Current Team'] +␣
↪player_data[player]['22/23 Away Games Played for Current Team']) != 0 else 0

        player_data[player]['23/24 Goals per Home Game'] =␣
↪float(player_data[player]['23/24 Home Goals for Current Team']/
↪player_data[player]['23/24 Home Games Played for Current Team']) if␣
↪player_data[player]['23/24 Home Games Played for Current Team'] != 0 else 0
        player_data[player]['23/24 Goals per Home Game Against 1-5'] =␣
↪float(player_data[player]['23/24 Home Goals Against 1-5']/
↪player_data[player]['23/24 Home Games Against 1-5']) if␣
↪player_data[player]['23/24 Home Games Against 1-5'] != 0 else 0
```

```
        player_data[player]['23/24 Assists per Home Game'] =␣
↪float(player_data[player]['23/24 Home Assists for Current Team']/
↪player_data[player]['23/24 Home Games Played for Current Team']) if␣
↪player_data[player]['23/24 Home Games Played for Current Team'] != 0 else 0
        player_data[player]['23/24 Assists per Home Game Against 1-5'] =␣
↪float(player_data[player]['23/24 Home Assists Against 1-5']/
↪player_data[player]['23/24 Home Games Against 1-5']) if␣
↪player_data[player]['23/24 Home Games Against 1-5'] != 0 else 0
        player_data[player]['23/24 Goals per Home Game Against 6-10'] =␣
↪float(player_data[player]['23/24 Home Goals Against 6-10']/
↪player_data[player]['23/24 Home Games Against 6-10']) if␣
↪player_data[player]['23/24 Home Games Against 6-10'] != 0 else 0
        player_data[player]['23/24 Assists per Home Game Against 6-10'] =␣
↪float(player_data[player]['23/24 Home Assists Against 6-10']/
↪player_data[player]['23/24 Home Games Against 6-10']) if␣
↪player_data[player]['23/24 Home Games Against 6-10'] != 0 else 0
        player_data[player]['23/24 Goals per Home Game Against 11-15'] =␣
↪float(player_data[player]['23/24 Home Goals Against 11-15']/
↪player_data[player]['23/24 Home Games Against 11-15']) if␣
↪player_data[player]['23/24 Home Games Against 11-15'] != 0 else 0
        player_data[player]['23/24 Assists per Home Game Against 11-15'] =␣
↪float(player_data[player]['23/24 Home Assists Against 11-15']/
↪player_data[player]['23/24 Home Games Against 11-15']) if␣
↪player_data[player]['23/24 Home Games Against 11-15'] != 0 else 0
        player_data[player]['23/24 Goals per Home Game Against 16-20'] =␣
↪float(player_data[player]['23/24 Home Goals Against 16-20']/
↪player_data[player]['23/24 Home Games Against 16-20']) if␣
↪player_data[player]['23/24 Home Games Against 16-20'] != 0 else 0
        player_data[player]['23/24 Assists pe Homer Game Against 16-20'] =␣
↪float(player_data[player]['23/24 Home Assists Against 16-20']/
↪player_data[player]['23/24 Home Games Against 16-20']) if␣
↪player_data[player]['23/24 Home Games Against 16-20'] != 0 else 0

        player_data[player]['23/24 Goals per Away Game'] =␣
↪float(player_data[player]['23/24 Away Goals for Current Team']/
↪player_data[player]['23/24 Away Games Played for Current Team']) if␣
↪player_data[player]['23/24 Away Games Played for Current Team'] != 0 else 0
        player_data[player]['23/24 Goals per Away Game Against 1-5'] =␣
↪float(player_data[player]['23/24 Away Goals Against 1-5']/
↪player_data[player]['23/24 Away Games Against 1-5']) if␣
↪player_data[player]['23/24 Away Games Against 1-5'] != 0 else 0
        player_data[player]['23/24 Assists per Away Game'] =␣
↪float(player_data[player]['23/24 Away Assists for Current Team']/
↪player_data[player]['23/24 Away Games Played for Current Team']) if␣
↪player_data[player]['23/24 Away Games Played for Current Team'] != 0 else 0
```

```
        player_data[player]['23/24 Assists per Away Game Against 1-5'] =␣
↪float(player_data[player]['23/24 Away Assists Against 1-5']/
↪player_data[player]['23/24 Away Games Against 1-5']) if␣
↪player_data[player]['23/24 Away Games Against 1-5'] != 0 else 0
        player_data[player]['23/24 Goals per Away Game Against 6-10'] =␣
↪float(player_data[player]['23/24 Away Goals Against 6-10']/
↪player_data[player]['23/24 Away Games Against 6-10']) if␣
↪player_data[player]['23/24 Away Games Against 6-10'] != 0 else 0
        player_data[player]['23/24 Assists per Away Game Against 6-10'] =␣
↪float(player_data[player]['23/24 Away Assists Against 6-10']/
↪player_data[player]['23/24 Away Games Against 6-10']) if␣
↪player_data[player]['23/24 Away Games Against 6-10'] != 0 else 0
        player_data[player]['23/24 Goals per Away Game Against 11-15'] =␣
↪float(player_data[player]['23/24 Away Goals Against 11-15']/
↪player_data[player]['23/24 Away Games Against 11-15']) if␣
↪player_data[player]['23/24 Away Games Against 11-15'] != 0 else 0
        player_data[player]['23/24 Assists per Away Game Against 11-15'] =␣
↪float(player_data[player]['23/24 Away Assists Against 11-15']/
↪player_data[player]['23/24 Away Games Against 11-15']) if␣
↪player_data[player]['23/24 Away Games Against 11-15'] != 0 else 0
        player_data[player]['23/24 Goals per Away Game Against 16-20'] =␣
↪float(player_data[player]['23/24 Away Goals Against 16-20']/
↪player_data[player]['23/24 Away Games Against 16-20']) if␣
↪player_data[player]['23/24 Away Games Against 16-20'] != 0 else 0
        player_data[player]['23/24 Assists per Away Game Against 16-20'] =␣
↪float(player_data[player]['23/24 Away Assists Against 16-20']/
↪player_data[player]['23/24 Away Games Against 16-20']) if␣
↪player_data[player]['23/24 Away Games Against 16-20'] != 0 else 0

        player_data[player]['23/24 Average BPS per Game'] =␣
↪float(player_data[player]['23/24 BPS for Current Team']/
↪(player_data[player]['23/24 Home Games Played for Current Team'] +␣
↪player_data[player]['23/24 Away Games Played for Current Team'])) if␣
↪(player_data[player]['23/24 Home Games Played for Current Team'] +␣
↪player_data[player]['23/24 Away Games Played for Current Team']) != 0 else 0

    team_data_df = pd.DataFrame.from_dict(team_data, orient='index')
    team_data_df.index.name = 'Team'
    player_data_df = pd.DataFrame.from_dict(player_data, orient='index')
    player_data_df.index.name = 'Player'

    with pd.ExcelWriter(f"historical_data_output.xlsx") as writer:
        team_data_df.to_excel(writer, sheet_name='Teams')
        player_data_df.to_excel(writer, sheet_name='Players')

    return team_data, player_data
```

[14]:
```python
def teams_league_positions_mapping(teams: list) -> dict:
    """
    Return a mapping from team ID to league position.

    Args:
        teams (list): List of team dictionaries.

    Returns:
        dict: Mapping from team ID to league position.
    """
    return {team['id']: team['position'] for team in teams}
```

[15]:
```python
def position_mapping(data: dict) -> dict:
    """
    Return a mapping from element_type ID to player position short name (e.g.,␣
 ↪'GKP', 'DEF').

    Args:
        data (dict): FPL API data.

    Returns:
        dict: Mapping from element_type ID to position short name.
    """
    return {et["id"]: et["singular_name_short"] for et in data["element_types"]}
```

[16]:
```python
def prepare_nickname(nickname: str) -> tuple:
    """
    Clean and generate two versions of a player's nickname for matching␣
 ↪purposes.

    Args:
        nickname (str): The player's nickname.

    Returns:
        tuple: Two cleaned nickname strings.
    """
    nickname1 = nickname.replace("'", '')
    nickname2 = nickname.replace("'", '')
    index = nickname1.find(".")
    while (index != -1):
        if index != len(nickname1) - 1:
            nickname1 = nickname1[:index] + ' ' + nickname1[index+1:].strip()
            if nickname1.find(".") != -1:
                nickname1 = nickname1[index+1:]
            index = nickname1.find(".")
        else:
            nickname1 = nickname1[:index]
```

```
            index = nickname1.find(".")

    index2 = nickname2.find(".")
    while (index2 != -1):
        if index2 != len(nickname2) - 1:
            nickname2 = nickname2[index2+1:]
            index2 = nickname2.find(".")
        else:
            nickname2 = nickname2[:index2]
            index2 = nickname2.find(".")

    nickname1 = nickname1.replace("-", " ").replace("'", '')
    nickname2 = nickname2.replace("-", " ").replace("'", '')
    return nickname1, nickname2
```

[17]:
```
def player_dict_constructor(
    players_data: list,
    team_stats_dict: dict,
    player_stats_dict: dict,
    element_types: dict,
    team_id_to_name: dict
) -> dict:
    """
    Build a dictionary with detailed stats for every player from the FPL API.

    Args:
        players_data (list): List of player dictionaries.
        team_stats_dict (dict): Team statistics.
        player_stats_dict (dict): Player statistics.
        element_types (dict): Mapping from element_type ID to position.
        team_id_to_name (dict): Mapping from team ID to team name.

    Returns:
        dict: Player details dictionary.
    """
    # Initialize player_dict to store lists of values for each key
    player_dict = defaultdict(lambda: defaultdict(list))

    for player in players_data:
        first_name = " ".join(prepare_name(player["first_name"]))
        second_name = " ".join(prepare_name(player["second_name"]))
        player_name = first_name + " " + second_name
        nickname = player['web_name']
        nickname1, nickname2 = prepare_nickname(nickname)
        team = TEAM_NAMES_ODDSCHECKER.get(team_id_to_name[player["team"]],␣
↪team_id_to_name[player["team"]])
```

```
        player_dict[player_name]['Nickname'] = [nickname1.strip()] if nickname1␣
↪!= None else ["Unknown"]
        player_dict[player_name]['Nickname2'] = [nickname2.strip()] if␣
↪nickname2 != None else ["Unknown"]
        player_dict[player_name]['Position'] =␣
↪[element_types[player["element_type"]]]
        player_dict[player_name]['Team'] = [team]
        player_dict[player_name]['Chance of Playing'] =␣
↪[player['chance_of_playing_next_round'] / 100] if␣
↪player['chance_of_playing_next_round'] else [1] if player['status'] in ('a',␣
↪'d') else [0]
        games_played_of_total_home_games_ratio =␣
↪float(team_stats_dict[team]['Home Games Played']/
↪player_stats_dict[player_name]['Home Games Played for Current Team']) if␣
↪player_stats_dict[player_name]['Home Games Played for Current Team'] > 0␣
↪else 1
        games_played_of_total_away_games_ratio =␣
↪float(team_stats_dict[team]['Away Games Played']/
↪player_stats_dict[player_name]['Away Games Played for Current Team']) if␣
↪player_stats_dict[player_name]['Away Games Played for Current Team'] > 0␣
↪else 1
        games_played_of_total_games_ratio = float((team_stats_dict[team]['Home␣
↪Games Played'] + team_stats_dict[team]['Away Games Played'])/
↪(player_stats_dict[player_name]['Home Games Played for Current Team'] +␣
↪player_stats_dict[player_name]['Away Games Played for Current Team'])) if␣
↪(player_stats_dict[player_name]['Home Games Played for Current Team'] +␣
↪player_stats_dict[player_name]['Away Games Played for Current Team']) != 0␣
↪else 1
        games_played = int(player_stats_dict[player_name]['Home Games Played␣
↪for Current Team'] + player_stats_dict[player_name]['Away Games Played for␣
↪Current Team']) if int((player_stats_dict[player_name]['Home Games Played␣
↪for Current Team'] + player_stats_dict[player_name]['Away Games Played for␣
↪Current Team'])) >= int(player['starts']) else int(player['starts'])
        player_dict[player_name]['Games'] = [games_played]
        player_dict[player_name]['Average Minutes per Game'] =␣
↪[player_stats_dict[player_name].get('Minutes per Game', 90)]
        player_dict[player_name]['Average BPS per Game'] =␣
↪[player_stats_dict[player_name].get('Average BPS per Game', 0)]
        # How many goals has the player scored out of the total goals scored by␣
↪his team
```

```
        player_dict[player_name]['Share of Goals by The Team'] =␣
↪[float((player_stats_dict[player_name]["Home Goals for Current Team"] +␣
↪player_stats_dict[player_name]["Away Goals for Current Team"])/
↪(team_stats_dict[team]['Home Goals'] + team_stats_dict[team]['Away Goals']))␣
↪* games_played_of_total_games_ratio] if games_played_of_total_games_ratio <␣
↪3 else [float((player_stats_dict[player_name]["Home Goals for Current Team"]␣
↪+ player_stats_dict[player_name]["Away Goals for Current Team"])/
↪(team_stats_dict[team]['Home Goals'] + team_stats_dict[team]['Away Goals']))]
        player_dict[player_name]['Share of Home Goals by The Team'] =␣
↪[float(player_stats_dict[player_name]["Home Goals for Current Team"]/
↪team_stats_dict[team]['Home Goals']) *␣
↪games_played_of_total_home_games_ratio] if␣
↪games_played_of_total_home_games_ratio < 3 else␣
↪[float(player_stats_dict[player_name]["Home Goals for Current Team"]/
↪team_stats_dict[team]['Home Goals'])]
        player_dict[player_name]['Share of Away Goals by The Team'] =␣
↪[float(player_stats_dict[player_name]["Away Goals for Current Team"]/
↪team_stats_dict[team]['Away Goals']) *␣
↪games_played_of_total_away_games_ratio] if␣
↪games_played_of_total_away_games_ratio < 3 else␣
↪[float(player_stats_dict[player_name]["Away Goals for Current Team"]/
↪team_stats_dict[team]['Away Goals'])]
        player_dict[player_name]['Expected Goals per Game'] =␣
↪[float(player['expected_goals']) / games_played] if games_played != 0 else␣
↪[0]
        # How many assists has the player assisted out of the total assists␣
↪assisted by his team
        player_dict[player_name]['Share of Assists by The Team'] =␣
↪[float((player_stats_dict[player_name]["Home Assists for Current Team"] +␣
↪player_stats_dict[player_name]["Away Assists for Current Team"])/
↪(team_stats_dict[team]['Home Goals'] + team_stats_dict[team]['Away Goals']))␣
↪* games_played_of_total_games_ratio] if games_played_of_total_games_ratio <␣
↪3 else [float((player_stats_dict[player_name]["Home Assists for Current␣
↪Team"] + player_stats_dict[player_name]["Away Assists for Current Team"])/
↪(team_stats_dict[team]['Home Goals'] + team_stats_dict[team]['Away Goals']))]
        player_dict[player_name]['Share of Home Assists by The Team'] =␣
↪[float(player_stats_dict[player_name]["Home Assists for Current Team"]/
↪team_stats_dict[team]['Home Goals']) *␣
↪games_played_of_total_home_games_ratio] if␣
↪games_played_of_total_home_games_ratio < 3 else␣
↪[float(player_stats_dict[player_name]["Home Assists for Current Team"]/
↪team_stats_dict[team]['Home Goals'])]
```

```
        player_dict[player_name]['Share of Away Assists by The Team'] =␣
↪[float(player_stats_dict[player_name]["Away Assists for Current Team"]/
↪team_stats_dict[team]['Away Goals']) *␣
↪games_played_of_total_away_games_ratio] if␣
↪games_played_of_total_away_games_ratio < 3 else␣
↪[float(player_stats_dict[player_name]["Away Assists for Current Team"]/
↪team_stats_dict[team]['Away Goals'])]
        player_dict[player_name]['Expected Assists per Game'] =␣
↪[float(float(player['expected_assists']) / games_played)] if games_played !=␣
↪0 else [0]
        if element_types[player["element_type"]] == 'GKP':
            player_dict[player_name]['Share of Goalkeeper Saves by The Team'] =␣
↪[float(player_stats_dict[player_name]["Goalkeeper Saves for Current Team"]/
↪(team_stats_dict[team]['Home Goalkeeper Saves'] +␣
↪team_stats_dict[team]['Away Goalkeeper Saves']) *␣
↪games_played_of_total_games_ratio)] if games_played_of_total_games_ratio < 3␣
↪else [float(player_stats_dict[player_name]["Goalkeeper Saves for Current␣
↪Team"]/(team_stats_dict[team]['Home Goalkeeper Saves'] +␣
↪team_stats_dict[team]['Away Goalkeeper Saves']))]
            player_dict[player_name]['Team Goalkeeper Saves per Home Game'] =␣
↪[team_stats_dict[team]['Goalkeeper Saves per Home Game']]
            player_dict[player_name]['Team Goalkeeper Saves per Away Game'] =␣
↪[team_stats_dict[team]['Goalkeeper Saves per Away Game']]

    return player_dict
```

1.3 Odds Scraping

The following functions use Selenium to scrape match and player odds from Oddschecker, handling pop-ups and extracting relevant odds for each fixture.

[18]:
```
def fetch_all_match_links(
    next_fixtures: list,
    team_id_to_name: dict,
    teams_positions_map: dict,
    driver: "webdriver.Chrome"
) -> dict:
    """
    Scrape Oddschecker for links to all matches in the next gameweek(s).

    Args:
        next_fixtures (list): List of fixture dictionaries for the next␣
↪gameweek(s).
        team_id_to_name (dict): Mapping from team ID to team name.
        teams_positions_map (dict): Mapping from team ID to league position.
        driver (webdriver.Chrome): Selenium WebDriver instance.
```

```
    Returns:
        dict: Details for each match, including Oddschecker link and team info.
    """
    driver.get("https://www.oddschecker.com/football/english/premier-league/")
    driver.execute_script("document.body.style.zoom='33%'")
    wait = WebDriverWait(driver, 10)

    try:
        span_element = wait.until(EC.element_to_be_clickable((By.XPATH, '/html/
↪body/div[1]/div/section/h2/span[2]')))
        # Click on the <span> element (Accessing outside UK pop-up)
        span_element.click()

    except TimeoutException:
        print("Prompt for accessing outside UK did not pop up")

    wait = WebDriverWait(driver, 3)
    try:
        cookiebutton = wait.until(EC.element_to_be_clickable((By.CLASS_NAME,␣
↪'CookieBannerAcceptButton_c1mxe743')))
        # Click on the accept cookies button
        cookiebutton.click()
    except TimeoutException:
        print("Prompt for accepting Cookies did not pop up")

    except ElementClickInterceptedException:
        try:
            wait = WebDriverWait(driver, 3)
            cookiebutton = wait.until(EC.element_to_be_clickable((By.
↪CLASS_NAME, 'CookieBannerAcceptButton_c1mxe743')))
            cookiebutton.click()
        except ElementClickInterceptedException:
            print("Prompt for accepting Cookies did not pop up")

    wait = WebDriverWait(driver, 8)
    try:
        close_ad = wait.until(EC.element_to_be_clickable((By.CLASS_NAME,␣
↪'webpush-swal2-close')))
        # Click close ad button
        close_ad.click()
    except TimeoutException:
        print('Ad did not pop up')

    try:
        wait = WebDriverWait(driver, 3)
        matches_button = wait.until(EC.element_to_be_clickable((By.XPATH, "//
↪button[contains(text(), 'Matches')]")))
```

```
        matches_button.click()
    except Exception as e:
        print("Couldn't click Matches tab ", e)

    matches_details = {}
    for fixture in next_fixtures:
        home_team_id = fixture['team_h']
        away_team_id = fixture['team_a']
        home_team_name = team_id_to_name.get(home_team_id, "Unknown Team")
        away_team_name = team_id_to_name.get(away_team_id, "Unknown Team")
        home_position = teams_positions_map.get(home_team_id, "Unknown␣
↪Position")
        away_position = teams_positions_map.get(away_team_id, "Unknown␣
↪Position")
        if abs(int(home_position) - int(away_position)) >= 5:
            if home_position > away_position:
                Underdog_Bonus = 'Home'
            else:
                Underdog_Bonus = 'Away'
        else:
            Underdog_Bonus = 'None'

        home_team = TEAM_NAMES_ODDSCHECKER.get(home_team_name, home_team_name)
        away_team = TEAM_NAMES_ODDSCHECKER.get(away_team_name, away_team_name)
        if home_team == None:
            home_team = home_team_name
        if away_team == None:
            away_team = away_team_name
        match_title = home_team + " v " + away_team

        try:
            # Find match link
            match_link = driver.find_element(By.XPATH, f"//
↪a[@title='{match_title}'][@href]")
            href = match_link.get_attribute("href")
        except NoSuchElementException:
            print(f"Match link for {match_title} not found.")
            href = "Link not found"
        matches_details[match_title] = {}
        matches_details[match_title]['home_team'] = home_team
        matches_details[match_title]['away_team'] = away_team
        matches_details[match_title]['home_position'] = home_position
        matches_details[match_title]['away_position'] = away_position
        matches_details[match_title]['Underdog Bonus'] = Underdog_Bonus
        matches_details[match_title]['Link'] = href

    return matches_details
```

[19]:
```
def fetch_win_market_odds(
    match_dict: dict,
    driver: "webdriver.Chrome",
    player_dict: dict,
    team_stats_dict: dict
) -> None:
    """
    Fetch win/draw odds for a match, calculate probabilities, and update␣
 ↪manager entries in player_dict.

    Args:
        match_dict (dict): Details for a single match.
        driver (webdriver.Chrome): Selenium WebDriver instance.
        player_dict (dict): Player details dictionary.
        team_stats_dict (dict): Team statistics dictionary.
    """
    home_team = match_dict.get('home_team', 'Unknown')
    away_team = match_dict.get('away_team', 'Unknown')
    Underdog_Bonus = match_dict.get('Underdog Bonus', 'None')
    link = match_dict.get('Link', 'Link not found')
    elo_win_probs =␣
 ↪calculate_match_probabilities_with_draw(team_stats_dict[home_team]['ELO'],␣
 ↪team_stats_dict[away_team]['ELO'], team_stats_dict[home_team]['HFA'])

    if link != "Link not found":
        try:
            driver.get(link)
            driver.execute_script("document.body.style.zoom='33%'")
            wait = WebDriverWait(driver, 3)
            try:
                close_ad = wait.until(EC.element_to_be_clickable((By.
 ↪CLASS_NAME, 'webpush-swal2-close')))
                # Click close ad button
                close_ad.click()
            except TimeoutException:
                print('Ad did not pop up')
        except Exception as e:
            print("Couldn't open link ", link, " ", e)

        try:
            win_market_header = driver.find_element(By.XPATH, "//
 ↪h2[contains(text(), 'Win Market')]")
            # Expand the section if it's collapsed
            if win_market_header.get_attribute("aria-expanded") == "false":
                win_market_header.click()
                time.sleep(3)
            wait = WebDriverWait(driver, 3)
```

```
            try:
                compare_odds = wait.until(EC.element_to_be_clickable((By.XPATH,␣
↪f"//h2[contains(text(), 'Win Market')]/following-sibling::*[1]/*[1]/
↪button[contains(text(), 'Compare All Odds')]")))
                # Expand the section if it's collapsed
                if compare_odds.get_attribute("aria-expanded") == "false":
                    compare_odds.click()
                    time.sleep(3)  # Wait for the section to expand
                try:
                    odds_dict = {}
                    outcomes = driver.find_elements(By.XPATH, "//
↪h4[contains(text(), 'Win Market')]/following::a[position()<4]")
                    odds_columns = driver.find_elements(By.XPATH, "//
↪h4[contains(text(), 'Win Market')]/following::
↪div[@class='oddsAreaWrapper_o17xb9rs RowLayout_refg9ta']")
                    for outcome in outcomes:
                        outcome_string = outcome.get_attribute("innerText")
                        odds_dict[outcome_string] = []
                    i = 0
                    try:
                        for column in odds_columns:
                            odd_buttons = column.find_elements(By.XPATH, "./
↪child::button")
                            odds_list = []
                            for odd_button in odd_buttons:
                                odd_text = odd_button.get_attribute("innerText")
                                if odd_text and odd_text.find(' ') != -1:
                                    odd_text = odd_text.replace(' ', '')
                                if odd_text and odd_text.find('/') != -1:
                                    odd_fraction = Fraction(odd_text)
                                    odds_list.append(odd_fraction)
                            if len(odds_list) > 2:
                                # Include only odds that do not deviate from␣
↪the mean by more than 3 standard deviations
                                mean = sum(odds_list) / len(odds_list)
                                std = statistics.stdev(odds_list)
                                odds_list = [odd for odd in odds_list if␣
↪abs(odd - mean) <= 3 * std]
                            odds_dict[list(odds_dict)[i]] = odds_list
                            i += 1
                        print("Found odds for Win Market")
                        win_market_header.click()

                        try:
                            home_win_odd = sum(odds_dict[home_team])/
↪len(odds_dict[home_team])
```

```
                            away_win_odd = sum(odds_dict[away_team])/
↪len(odds_dict[away_team])
                            draw_odd = sum(odds_dict['Draw'])/
↪len(odds_dict['Draw'])

                            home_win_prob = 1/float(home_win_odd + 1) if␣
↪home_win_odd else 0
                            away_win_prob = 1/float(away_win_odd + 1) if␣
↪away_win_odd else 0
                            draw_prob = 1/float(draw_odd + 1) if draw_odd else 0

                            win_market_margin = home_win_prob + away_win_prob +␣
↪draw_prob
                            if win_market_margin > 1:
                                home_win_prob /= win_market_margin
                                away_win_prob /= win_market_margin
                                draw_prob /= win_market_margin

                        except Exception as e:
                            print("Could not get average odds for Home Win,␣
↪Away Win and/or Draw", e)
                            home_win_prob = elo_win_probs['Home Win␣
↪Probability']
                            away_win_prob = elo_win_probs['Away Win␣
↪Probability']
                            draw_prob = elo_win_probs['Draw Probability']
                    except Exception as e:
                        print("Couldn't get odds for Win Market", e)
                        home_win_prob = elo_win_probs['Home Win Probability']
                        away_win_prob = elo_win_probs['Away Win Probability']
                        draw_prob = elo_win_probs['Draw Probability']

                except Exception as e:
                    print("Couldn't find Win Market All Odds Section")
                    home_win_prob = elo_win_probs['Home Win Probability']
                    away_win_prob = elo_win_probs['Away Win Probability']
                    draw_prob = elo_win_probs['Draw Probability']

            except Exception as e:
                print("Could not open Compare All Odds on Win Market, e")
                home_win_prob = elo_win_probs['Home Win Probability']
                away_win_prob = elo_win_probs['Away Win Probability']
                draw_prob = elo_win_probs['Draw Probability']

        except Exception as e:
            print("Could not find Win Market header, e")
```

```
            home_win_prob = elo_win_probs['Home Win Probability']
            away_win_prob = elo_win_probs['Away Win Probability']
            draw_prob = elo_win_probs['Draw Probability']
    else:
        home_win_prob = elo_win_probs['Home Win Probability']
        away_win_prob = elo_win_probs['Away Win Probability']
        draw_prob = elo_win_probs['Draw Probability']
    for player in player_dict:
        if player_dict[player]['Team'][0] == home_team:
            player_dict[player]['Home/Away'].append('Home')
            player_dict[player]['Opponent'].append(away_team)
            if player_dict[player]['Position'][0] == 'MNG':
                player_dict[player]['Win Probability'].append(home_win_prob)
                player_dict[player]['Draw Probability'].append(draw_prob)
                player_dict[player]['ELO Win Probability'].
↪append(elo_win_probs['Home Win Probability'])
                player_dict[player]['ELO Draw Probability'].
↪append(elo_win_probs['Draw Probability'])
                if Underdog_Bonus == 'Home':
                    player_dict[player]['Manager Bonus'].append('True')
                else:
                    player_dict[player]['Manager Bonus'].append('False')
        elif player_dict[player]['Team'][0] == away_team:
            player_dict[player]['Home/Away'].append('Away')
            player_dict[player]['Opponent'].append(home_team)
            if player_dict[player]['Position'][0] == 'MNG':
                player_dict[player]['Win Probability'].append(away_win_prob)
                player_dict[player]['Draw Probability'].append(draw_prob)
                player_dict[player]['ELO Win Probability'].
↪append(elo_win_probs['Away Win Probability'])
                player_dict[player]['ELO Draw Probability'].
↪append(elo_win_probs['Draw Probability'])
                if Underdog_Bonus == 'Away':
                    player_dict[player]['Manager Bonus'].append('True')
                else:
                    player_dict[player]['Manager Bonus'].append('False')
        else:
            continue
```

[20]:
```
def fetch_odds(odd_type: str, driver: "webdriver.Chrome") -> typing.
↪Optional[dict]:
    """
    Fetch odds for a specific market (e.g., Player Assists, Goalkeeper Saves)␣
↪from Oddschecker.

    Args:
        odd_type (str): The odds market to fetch.
```

```
        driver (webdriver.Chrome): Selenium WebDriver instance.

    Returns:
        dict: Mapping from outcome to list of odds, or None if not found.
    """
    wait = WebDriverWait(driver, 2)
    try:
        # Find the section
        header = wait.until(EC.element_to_be_clickable((By.XPATH, "//h2[text()␣
↪='" + odd_type + "']")))
        # Expand the section if it's collapsed
        if header.get_attribute("aria-expanded") == "false":
            header.click()
            time.sleep(3)
        wait = WebDriverWait(driver, 5)
        try:
            compare_odds = wait.until(EC.element_to_be_clickable((By.XPATH, "//
↪h2[(text() ='" + odd_type + "')]/following-sibling::*[1]/*[1]/
↪button[contains(text(), 'Compare All Odds')]")))
            # Expand the section if it's collapsed
            if compare_odds.get_attribute("aria-expanded") == "false":
                compare_odds.click()
                time.sleep(3)  # Wait for the section to expand
            try:
                odds_dict = {}
                outcomes = driver.find_elements(By.XPATH, "//h4[(text() ='" +␣
↪odd_type + "')]/following::span[@class='BetRowLeftBetName_b1m53rgx']")
                odds_columns = driver.find_elements(By.XPATH, "//h4[(text() ='"␣
↪+ odd_type + "')]/following::div[@class='oddsAreaWrapper_o17xb9rs␣
↪RowLayout_refg9ta']")
                try:
                    for outcome in outcomes:
                        outcome_string = outcome.get_attribute("innerText")
                        odds_dict[outcome_string] = []
                    try:
                        i = 0
                        for column in odds_columns:
                            odd_buttons = column.find_elements(By.XPATH, "./
↪child::button")
                            odds_list = []
                            for odd_button in odd_buttons:
                                odd_text = odd_button.get_attribute("innerText")
                                if odd_text and odd_text.find(' ') != -1:
                                    odd_text = odd_text.replace(' ', '')
                                if odd_text and odd_text.find('/') != -1:
                                    odd_fraction = Fraction(odd_text)
                                    odds_list.append(odd_fraction)
```

```
                            if len(odds_list) > 2:
                                # Include only odds that do not deviate from␣
↪the mean by more than 3 standard deviations
                                mean = sum(odds_list) / len(odds_list)
                                std = statistics.stdev(odds_list)
                                odds_list = [odd for odd in odds_list if␣
↪abs(odd - mean) <= 3 * std]
                            odds_dict[list(odds_dict)[i]] = odds_list
                            i += 1
                        header.click()
                        time.sleep(1)
                        print("Found odds for", odd_type)
                        return odds_dict
                    except Exception as e:
                        print("Couldn't get odds for", odd_type, " ", e)
                except Exception as e:
                    print("Couldn't get odds for", odd_type, " ", e)
            except Exception as e:
                print("Couldn't find", odd_type, " All Odds Section", e)
        except Exception as e:
            print("Couldn't click Compare All Odds on", odd_type)
        header.click()
        time.sleep(1)
    except Exception as e:
        print("Couldn't find or expand section:", odd_type)
```

1.4 Probabilities, Averages and Predicted Points Calculation

These functions convert Elo ratings, historical data and bookmaker odds into probabilities and averages, which are finally combined to calculate predicted points for every player.

[21]:
```
def get_player_over_probs(
    odd_type: str,
    odds_dict: dict,
    player_dict: dict,
    home_team: str,
    away_team: str
) -> None:
    """
    Calculate player 'Over X' probabilities from odds and update player_dict.

    Args:
        odd_type (str): Odds market type.
        odds_dict (dict): Mapping from player/outcome to odds.
        player_dict (dict): Player details dictionary.
        home_team (str): Home team name.
        away_team (str): Away team name.
```

```
    """
    if odd_type == "Player Assists":
        odds_for = ['Over 0.5', 'Over 1.5', 'Over 2.5']
    else:
        odds_for = ['Over 0.5 Saves', 'Over 1.5 Saves', 'Over 2.5 Saves', 'Over␣
↪3.5 Saves', 'Over 4.5 Saves', 'Over 5.5 Saves', 'Over 6.5 Saves', 'Over 7.5␣
↪Saves', 'Over 8.5 Saves', 'Over 9.5 Saves']
    try:
        for player_odd, odds_list in odds_dict.items():
            index = player_odd.find("Over")
            odd_for = player_odd[index:].strip()
            if odd_for in odds_for:
                if len(odds_list) > 0:
                    odd = sum(odds_list)/len(odds_list)
                else:
                    odd = 0
                if odd_type == "Goalkeeper Saves":
                    name = player_odd[:index].replace("Saves", '').strip()
                    odd_for = odd_for.replace("Saves", '').strip()
                else:
                    name = player_odd[:index].strip()
                probability = (1/(float(Fraction(odd)) + 1)) if odd != 0 else 0
            else:
                continue
            try:
                matched_name = None  # Ensure matched_name is always defined
                for p in player_dict:
                    # Prepare the player name for comparison
                    player_tokens = prepare_name(p)
                    webname_tokens = prepare_name(name)

                    # Check if all tokens in one name exist in the other
                    if all(token in webname_tokens for token in player_tokens)␣
↪or all(token in player_tokens for token in webname_tokens):
                        matched_name = p
                        break

                # Add the odds to the player's dictionary
                if matched_name is not None:
                    player_dict[matched_name][f"{odd_for} {odd_type}␣
↪Probability"].append(probability)
                else:
                    for p in player_dict:
                        # Prepare the player name for comparison
                        webname_tokens = prepare_name(name)
                        nickname1 = player_dict[p]['Nickname'][0]
                        nickname2 = player_dict[p]['Nickname2'][0]
```

```
                        nickname1_tokens = prepare_name(nickname1)
                        nickname2_tokens = prepare_name(nickname2)

                        if (" ".join(nickname2_tokens) in " ".
↪join(webname_tokens) or " ".join(nickname1_tokens) in " ".
↪join(webname_tokens)) and (player_dict[p]['Team'][0] in [home_team,␣
↪away_team]):
                            matched_name = p
                            break
                        else:
                            p_name = PLAYER_NAMES_ODDSCHECKER.get(name,␣
↪"Unknown")
                            if p_name != "Unknown":
                                matched_name = p_name
                                break

                    if matched_name:
                        player_dict[matched_name][f"{odd_for} {odd_type}␣
↪Probability"].append(probability)

                    else:
                        player_dict[name]['Nickname'] = ['Unknown']
                        player_dict[name]['Nickname2'] = ['Unknown']
                        player_dict[name]['Position'] = ['Unknown']
                        player_dict[name]['Team'] = ["Unknown"]
                        player_dict[name][f"{odd_for} {odd_type} Probability"].
↪append(probability)
            except Exception as e:
                print("Couldn't update player_dict", e)
    except Exception as e:
        print("Couldn't calculate probabilities for ", odd_type, " ", e)
```

[22]:
```
def get_total_goals_over_probs(odds_dict: dict, team: str) -> typing.
↪Optional[dict]:
    """
    Calculate probabilities for total goals scored by a team using Over X odds.

    Args:
        odds_dict (dict): Mapping from outcome to odds.
        team (str): 'home' or 'away'.

    Returns:
        dict: Probabilities for 0-6+ goals scored by the team.
    """
    try:
        team_over_05_odd, team_over_15_odd, team_over_25_odd, team_over_35_odd,␣
↪team_over_45_odd, team_over_55_odd = 0,0,0,0,0,0
```

```
        for team_odd, odds_list in odds_dict.items():
            if len(odds_list) != 0:
                ave_odd = sum(odds_list)/len(odds_list)
            else:
                ave_odd = 0
            if team_odd == "Over 0.5":
                team_over_05_odd = ave_odd
            if team_odd == "Over 1.5":
                team_over_15_odd = ave_odd
            if team_odd == "Over 2.5":
                team_over_25_odd = ave_odd
            if team_odd == "Over 3.5":
                team_over_35_odd = ave_odd
            if team_odd == "Over 4.5":
                team_over_45_odd = ave_odd
            if team_odd == "Over 5.5":
                team_over_55_odd = ave_odd

        try:
            team_over_05_prob = (1/(float(Fraction(team_over_05_odd + 1)))) if␣
↪team_over_05_odd != 0 else 0
            team_over_15_prob = (1/(float(Fraction(team_over_15_odd + 1)))) if␣
↪team_over_15_odd != 0 else 0
            team_over_25_prob = (1/(float(Fraction(team_over_25_odd + 1)))) if␣
↪team_over_25_odd != 0 else 0
            team_over_35_prob = (1/(float(Fraction(team_over_35_odd + 1)))) if␣
↪team_over_35_odd != 0 else 0
            team_over_45_prob = (1/(float(Fraction(team_over_45_odd + 1)))) if␣
↪team_over_45_odd != 0 else 0
            team_over_55_prob = (1/(float(Fraction(team_over_55_odd + 1)))) if␣
↪team_over_55_odd != 0 else 0

            try:
                team_0_goal_prob = 1 - team_over_05_prob if team_over_05_prob !
↪= 0 else 0
                team_6_goal_prob = team_over_55_prob
                team_1_goal_prob = max(team_over_05_prob - team_over_15_prob,␣
↪0) if team_over_05_prob != 0 and team_over_15_prob != 0 else␣
↪team_over_05_prob
                team_2_goal_prob = max(team_over_15_prob - team_over_25_prob,␣
↪0) if team_over_15_prob != 0 and team_over_25_prob != 0 else␣
↪team_over_15_prob
                team_3_goal_prob = max(team_over_25_prob - team_over_35_prob,␣
↪0) if team_over_25_prob != 0 and team_over_35_prob != 0 else␣
↪team_over_25_prob
```

```
                team_4_goal_prob = max(team_over_35_prob - team_over_45_prob,␣
↪0) if team_over_35_prob != 0 and team_over_45_prob != 0 else␣
↪team_over_35_prob
                team_5_goal_prob = max(team_over_45_prob - team_over_55_prob,␣
↪0) if team_over_45_prob != 0 and team_over_55_prob != 0 else␣
↪team_over_45_prob

            except Exception as e:
                print(f"Couldnt calculate probabilities for Total {team.
↪capitalize()} Goals", e)
                return None
        except Exception as e:
            print(f"Couldnt calculate probabilities for Total {team.
↪capitalize()} Over Goals", e)
            return None
        return {team + '_0_goal_prob': team_0_goal_prob, team + '_1_goal_prob':␣
↪team_1_goal_prob, team + '_2_goals_prob': team_2_goal_prob, team +␣
↪'_3_goals_prob': team_3_goal_prob, team + '_4_goals_prob': team_4_goal_prob,␣
↪team + '_5_goals_prob': team_5_goal_prob, team + '_6_goals_prob':␣
↪team_6_goal_prob}
    except Exception as e:
        print(f"Couldnt find probabilities from odds_dict for Total {team.
↪capitalize()} Over Goals", e)
        return None
```

[23]:
```
def add_total_goals_probs_to_dict(
    probs_dict: dict,
    home_team: str,
    away_team: str,
    player_dict: dict
) -> None:
    """
    Add calculated home/away goals probabilities to each player's dictionary.

    Args:
        probs_dict (dict): Probabilities for goals scored/conceded.
        home_team (str): Home team name.
        away_team (str): Away team name.
        player_dict (dict): Player details dictionary.
    """
    for player in player_dict:
        if player_dict[player]['Team'][0] == home_team:
            home_goals_conceded_average = probs_dict["away_1_goal_prob"] + 2 *␣
↪probs_dict["away_2_goals_prob"] + 3 * probs_dict["away_3_goals_prob"] + 4 *␣
↪probs_dict["away_4_goals_prob"] + 5 * probs_dict["away_5_goals_prob"] + 6 *␣
↪probs_dict["away_6_goals_prob"]
```

```
            player_dict[player]['Clean Sheet Probability by Bookmaker Odds'].
↪append((probs_dict["away_0_goal_prob"] + math.
↪exp(-home_goals_conceded_average)) / 2)
            player_dict[player]['Goals Conceded by Team on Average'].
↪append(home_goals_conceded_average)
            home_goals_average = probs_dict["home_1_goal_prob"] + 2 *␣
↪probs_dict["home_2_goals_prob"] + 3 * probs_dict["home_3_goals_prob"] + 4 *␣
↪probs_dict["home_4_goals_prob"] + 5 * probs_dict["home_5_goals_prob"] + 6 *␣
↪probs_dict["home_6_goals_prob"]
            player_dict[player]['Goals Scored by Team on Average'].
↪append(home_goals_average)
        if player_dict[player]['Team'][0] == away_team:
            away_goals_conceded_average = probs_dict["home_1_goal_prob"] + 2 *␣
↪probs_dict["home_2_goals_prob"] + 3 * probs_dict["home_3_goals_prob"] + 4 *␣
↪probs_dict["home_4_goals_prob"] + 5 * probs_dict["home_5_goals_prob"] + 6 *␣
↪probs_dict["home_6_goals_prob"]
            player_dict[player]['Clean Sheet Probability by Bookmaker Odds'].
↪append((probs_dict["home_0_goal_prob"] + math.
↪exp(-away_goals_conceded_average)) / 2)
            player_dict[player]['Goals Conceded by Team on Average'].
↪append(away_goals_conceded_average)
            away_goals_average = probs_dict["away_1_goal_prob"] + 2 *␣
↪probs_dict["away_2_goals_prob"] + 3 * probs_dict["away_3_goals_prob"] + 4 *␣
↪probs_dict["away_4_goals_prob"] + 5 * probs_dict["away_5_goals_prob"] + 6 *␣
↪probs_dict["away_6_goals_prob"]
            player_dict[player]['Goals Scored by Team on Average'].
↪append(away_goals_average)
```

[24]:
```python
def add_probs_to_dict(
    odd_type: str,
    odds_dict: dict,
    player_dict: dict,
    home_team: str,
    away_team: str
) -> None:
    """
    Add calculated probabilities for a specific odds market to player_dict.

    Args:
        odd_type (str): Odds market type.
        odds_dict (dict): Mapping from player/outcome to odds.
        player_dict (dict): Player details dictionary.
        home_team (str): Home team name.
        away_team (str): Away team name.
    """
    try:
```

```
        for player_odd, odds_list in odds_dict.items():
            name = player_odd.strip()
            if len(odds_list) != 0:
                odd = sum(odds_list)/len(odds_list)
            else:
                odd = 0
            probability = (1/(float(Fraction(odd)) + 1)) if odd != 0 else 0
            matched_name = None  # Ensure matched_name is always defined
            for p in player_dict:
                # Prepare the player name for comparison
                player_tokens = prepare_name(p)
                webname_tokens = prepare_name(name)
                # Check if all tokens in one name exist in the other
                if all(token in webname_tokens for token in player_tokens) or␣
↪all(token in player_tokens for token in webname_tokens):
                    matched_name = p
                    break
            # Add the odds to the player's dictionary
            if matched_name is not None:
                player_dict[matched_name][f"{odd_type} Probability"].
↪append(probability)
            else:
                for p in player_dict:
                    # Prepare the player name for comparison
                    webname_tokens = prepare_name(name)
                    nickname1 = player_dict[p]['Nickname'][0]
                    nickname2 = player_dict[p]['Nickname2'][0]
                    nickname1_tokens = prepare_name(nickname1)
                    nickname2_tokens = prepare_name(nickname2)
                    if (" ".join(nickname2_tokens) in " ".join(webname_tokens)␣
↪or " ".join(nickname1_tokens) in " ".join(webname_tokens)) and␣
↪(player_dict[p]['Team'][0] in [home_team, away_team]):
                        matched_name = p
                        break
                else:
                    p_name = PLAYER_NAMES_ODDSCHECKER.get(name, "Unknown")
                    if p_name != "Unknown":
                        matched_name = p_name
                        break

                if matched_name:
                    player_dict[matched_name][f"{odd_type} Probability"].
↪append(probability)
                else:
                    player_dict[name]['Nickname'] = ['Unknown']
                    player_dict[name]['Nickname2'] = ['Unknown']
                    player_dict[name]['Position'] = ['Unknown']
```

```
                    player_dict[name]['Team'] = ["Unknown"]
                    player_dict[name][f"{odd_type} Probability"].
↪append(probability)
    except Exception as e:
        print("Couldn't get probability for ", odd_type, " ", e)
```

[25]:
```
def calc_specific_probs(
    player_dict: dict
) -> None:
    """
    Calculate average assists, goals, and saves for each player using bookmaker␣
↪and historical data.

    Args:
        player_dict (dict): Player details dictionary.
        team_stats_dict (dict): Team statistics dictionary.
        player_stats_dict (dict): Player statistics dictionary.
    """
    for player, odds in player_dict.items():
        position = odds.get("Position", ["Unknown"])[0]
        anytime_prob = odds.get("Anytime Goalscorer Probability", [])
        two_or_more_prob = odds.get("To Score 2 Or More Goals Probability", [])
        hattrick_prob = odds.get("To Score A Hat-Trick Probability", [])
        assisting_over_05_prob = odds.get("Over 0.5 Player Assists␣
↪Probability", [])
        assisting_over_15_prob = odds.get("Over 1.5 Player Assists␣
↪Probability", [])
        assisting_over_25_prob = odds.get("Over 2.5 Player Assists␣
↪Probability", [])

        ass_share = odds.get("Share of Assists by The Team", [0])[0]
        goal_share = odds.get("Share of Goals by The Team", [0])[0]

        total_goals_bookmaker = odds.get('Goals Scored by Team on Average', [])
        total_goals_historical = odds.get('Team xG by Historical Data', [])
        total_goals_scored_average = total_goals_bookmaker if␣
↪total_goals_bookmaker != [] else total_goals_historical

        xa_per_game = odds.get("Expected Assists per Game", [0])[0]
        xg_per_game = odds.get("Expected Goals per Game", [0])[0]

        venue = odds.get("Home/Away", [])

        if position in ['DEF', 'MID', 'FWD', 'Unknown']:
            for p25, p15, p05, t_gsa, h_a in␣
↪zip_longest(assisting_over_25_prob, assisting_over_15_prob,␣
↪assisting_over_05_prob, total_goals_scored_average, venue, fillvalue=0):
```

```
                three_ass_prob = p25
                one_ass_prob = p05 - p15 if p05 != 0 and p15 != 0 else p05
                two_ass_prob = p15 - p25 if p15 != 0 and p25 != 0 else p15
                expected_assists = three_ass_prob * 3 + two_ass_prob * 2 +␣
↪one_ass_prob
                if expected_assists != 0:
                    ass_average = expected_assists
                    player_dict[player]["xA by Bookmaker Odds"].
↪append(ass_average)
                ass_average2 = ((ass_share * t_gsa) + xa_per_game) / 2 if␣
↪ass_share != 0 else xa_per_game
                player_dict[player]["xA by Historical Data"].
↪append(ass_average2)

            for p3, p2, p1, t_gsa, h_a in zip_longest(hattrick_prob,␣
↪two_or_more_prob, anytime_prob, total_goals_scored_average, venue,␣
↪fillvalue=0):
                three_goals_prob = p3
                one_goal_prob = p1 - p2 if p1 != 0 and p2 != 0 else p1
                two_goals_prob = p2 - p3 if p2 != 0 and p3 != 0 else p2
                expected_goals = three_goals_prob * 3 + two_goals_prob * 2 +␣
↪one_goal_prob
                if expected_goals != 0:
                    goal_average = expected_goals
                    player_dict[player]["xG by Bookmaker Odds"].
↪append(goal_average)
                goal_average2 = ((goal_share * t_gsa) + xg_per_game) / 2 if␣
↪goal_share != 0 else xg_per_game
                player_dict[player]["xG by Historical Data"].
↪append(goal_average2)

        if position == 'GKP':
            saves_share = odds.get("Share of Goalkeeper Saves by The Team",␣
↪[0])[0]
            team_saves_per_home_game = odds.get("Team Goalkeeper Saves per Home␣
↪Game", [0])[0]
            team_saves_per_away_game = odds.get("Team Goalkeeper Saves per Away␣
↪Game", [0])[0]
            over_05_saves = odds.get("Over 0.5 Goalkeeper Saves Probability",␣
↪[])
            over_15_saves = odds.get("Over 1.5 Goalkeeper Saves Probability",␣
↪[])
            over_25_saves = odds.get("Over 2.5 Goalkeeper Saves Probability",␣
↪[])
            over_35_saves = odds.get("Over 3.5 Goalkeeper Saves Probability",␣
↪[])
```

```
            over_45_saves = odds.get("Over 4.5 Goalkeeper Saves Probability",␣
↪[])
            over_55_saves = odds.get("Over 5.5 Goalkeeper Saves Probability",␣
↪[])
            over_65_saves = odds.get("Over 6.5 Goalkeeper Saves Probability",␣
↪[])
            over_75_saves = odds.get("Over 7.5 Goalkeeper Saves Probability",␣
↪[])
            over_85_saves = odds.get("Over 8.5 Goalkeeper Saves Probability",␣
↪[])
            over_95_saves = odds.get("Over 9.5 Goalkeeper Saves Probability",␣
↪[])

            for s95, s85, s75, s65, s55, s45, s35, s25, s15, s05, h_a in␣
↪zip_longest(over_95_saves, over_85_saves, over_75_saves, over_65_saves,␣
↪over_55_saves, over_45_saves, over_35_saves, over_25_saves, over_15_saves,␣
↪over_05_saves, venue, fillvalue=0):
                zero_saves_prob = 1 - s05
                ten_saves_prob = s95
                one_saves_prob = s05 - s15 if s05 != 0 and s15 != 0 else max((1␣
↪- s15 - zero_saves_prob), 0)
                two_saves_prob = s15 - s25 if s15 != 0 and s25 != 0 else max((1␣
↪- one_saves_prob - zero_saves_prob), 0)
                three_saves_prob = s25 - s35 if s25 != 0 and s35 != 0 else␣
↪max((1 - two_saves_prob - one_saves_prob - zero_saves_prob), 0)
                four_saves_prob = s35 - s45 if s35 != 0 and s45 != 0 else␣
↪max((1 - three_saves_prob - two_saves_prob - one_saves_prob -␣
↪zero_saves_prob), 0)
                five_saves_prob = s45 - s55 if s45 != 0 and s55 != 0 else␣
↪max((1 - four_saves_prob - three_saves_prob - two_saves_prob -␣
↪one_saves_prob - zero_saves_prob), 0)
                six_saves_prob = s55 - s65 if s55 != 0 and s65 != 0 else max((1␣
↪- five_saves_prob - four_saves_prob - three_saves_prob - two_saves_prob -␣
↪one_saves_prob - zero_saves_prob), 0)
                seven_saves_prob = s65 - s75 if s65 != 0 and s75 != 0 else␣
↪max((1 - six_saves_prob - five_saves_prob - four_saves_prob -␣
↪three_saves_prob - two_saves_prob - one_saves_prob - zero_saves_prob), 0)
                eight_saves_prob = s75 - s85 if s75 != 0 and s85 != 0 else␣
↪max((1 - seven_saves_prob - six_saves_prob - five_saves_prob -␣
↪four_saves_prob - three_saves_prob - two_saves_prob - one_saves_prob -␣
↪zero_saves_prob), 0)
                nine_saves_prob = s85 - s95 if s85 != 0 and s95 != 0 else␣
↪max((1 - eight_saves_prob - seven_saves_prob - six_saves_prob -␣
↪five_saves_prob - four_saves_prob - three_saves_prob - two_saves_prob -␣
↪one_saves_prob - zero_saves_prob), 0)
```

```
                saves_average = one_saves_prob + two_saves_prob * 2 +␣
↪three_saves_prob * 3 + four_saves_prob * 4 + five_saves_prob * 5 +␣
↪six_saves_prob * 6 + seven_saves_prob * 7 + eight_saves_prob * 8 +␣
↪nine_saves_prob * 9 + ten_saves_prob * 10
                saves_average2 = saves_share * team_saves_per_home_game if h_a␣
↪== 'Home' else saves_share * team_saves_per_away_game
                player_dict[player]["xSaves by Historical Data"].
↪append(saves_average2)
                if saves_average != 0:
                    player_dict[player]["xSaves by Bookmaker Odds"].
↪append(saves_average)
```

[26]:
```
def calc_avg_bonus_points(
    player_dict: dict,
    match_dict: dict
) -> None:
    """
    Calculate and add predicted bonus points per game for each player.

    Args:
        player_dict (dict): Player details dictionary.
        match_dict (dict): Match details dictionary.
    """
    team_bps_sum = defaultdict(list)
    for player, stats in player_dict.items():
        team = stats['Team'][0]
        bps_per_game = stats['Average BPS per Game'][0] if stats['Average BPS␣
↪per Game'] != [] else 0
        mins_per_start = stats['Average Minutes per Game'][0] if stats['Average␣
↪Minutes per Game'] != [] else 0
        if mins_per_start > 45:
            team_bps_sum[team].append(bps_per_game)

    for fixture, details in match_dict.items():
        home_team = details['home_team']
        away_team = details['away_team']
        fixture_bps = 11 * (float(sum(team_bps_sum[home_team]) /␣
↪len(team_bps_sum[home_team]))) + 11 * (float(sum(team_bps_sum[away_team]) /␣
↪len(team_bps_sum[away_team])))
        for player, stats in player_dict.items():
            if stats['Team'][0] == home_team:
                bps_ratio = float(max(player_dict[player]['Average BPS per␣
↪Game'][0], 0) / fixture_bps) if fixture_bps != 0 else 0
                player_dict[player]['Average Bonus Points per Game'].
↪append(bps_ratio * 6)
            if stats['Team'][0] == away_team:
```

```
                bps_ratio = float(max(player_dict[player]['Average BPS per␣
↪Game'][0], 0) / fixture_bps) if fixture_bps != 0 else 0
                player_dict[player]['Average Bonus Points per Game'].
↪append(bps_ratio * 6)
```

[27]:
```
def calc_team_xgs(
    home_team: str,
    away_team: str,
    team_stats_dict: dict,
    player_dict: dict
) -> None:
    """
    Estimate expected goals (xG) for both teams in a fixture and update each␣
↪player's stats.

    Args:
        home_team (str): Name of the home team.
        away_team (str): Name of the away team.
        team_stats_dict (dict): Team statistics dictionary.
        player_dict (dict): Player details dictionary.
    """
    home_pos_range = get_pos_range(team_stats_dict[home_team]['League␣
↪Position'])
    away_pos_range = get_pos_range(team_stats_dict[away_team]['League␣
↪Position'])
    home_total_goals_p90 = team_stats_dict[home_team]['Goals per Game']
    away_total_goals_p90 = team_stats_dict[away_team]['Goals per Game']
    home_goals_p90 = team_stats_dict[home_team]['Goals per Home Game']
    away_goals_p90 = team_stats_dict[away_team]['Goals per Away Game']
    home_goals_conceded_p90 = team_stats_dict[home_team]['Goals Conceded per␣
↪Home Game']
    away_goals_conceded_p90 = team_stats_dict[away_team]['Goals Conceded per␣
↪Away Game']
    home_total_goals_conceded_p90 = team_stats_dict[home_team]['Goals Conceded␣
↪per Game']
    away_total_goals_conceded_p90 = team_stats_dict[away_team]['Goals Conceded␣
↪per Game']
    home_conceded_against_string = f"Goals Conceded per Home Game Against␣
↪{away_pos_range}"
    away_conceded_against_string = f"Goals Conceded per Away Game Against␣
↪{home_pos_range}"
    home_scored_against_string = f"Goals per Home Game Against {away_pos_range}"
    away_scored_against_string = f"Goals per Away Game Against {home_pos_range}"
```

```
    home_xg = (team_stats_dict[home_team]['ELO'] /␣
↪team_stats_dict[away_team]['ELO']) * ((home_goals_p90 + home_total_goals_p90␣
↪+ away_goals_conceded_p90 + away_total_goals_conceded_p90 + 0.5 *␣
↪team_stats_dict[home_team][home_scored_against_string] + 0.5 *␣
↪team_stats_dict[away_team][away_conceded_against_string]) / 5)
    away_xg = (team_stats_dict[away_team]['ELO'] /␣
↪team_stats_dict[home_team]['ELO']) * ((away_goals_p90 + away_total_goals_p90␣
↪+ home_goals_conceded_p90 + home_total_goals_conceded_p90 + 0.5 *␣
↪team_stats_dict[away_team][away_scored_against_string] + 0.5 *␣
↪team_stats_dict[home_team][home_conceded_against_string]) / 5)

    for player, stats in player_dict.items():
        if stats['Team'][0] == home_team:
            player_dict[player]['Team xG by Historical Data'].append(home_xg)
            player_dict[player]['Team xGC by Historical Data'].append(away_xg)
            player_dict[player]["Clean Sheet Probability by Historical Data"].
↪append(math.exp(-away_xg))
        if stats['Team'][0] == away_team:
            player_dict[player]['Team xG by Historical Data'].append(away_xg)
            player_dict[player]['Team xGC by Historical Data'].append(home_xg)
            player_dict[player]["Clean Sheet Probability by Historical Data"].
↪append(math.exp(-home_xg))
```

[28]:
```
def calculate_match_probabilities_with_draw(home_elo: float, away_elo: float,␣
↪HFA: float) -> dict:
    """
    Calculate probabilities for home win, draw, and away win using Elo ratings␣
↪and a draw probability formula.

    Args:
        home_elo (float): Elo score of the home team.
        away_elo (float): Elo score of the away team.
        HFA (float): Home Field Advantage.

    Returns:
        dict: Probabilities for home win, draw, and away win.
    """
    # Calculate the Elo difference (dr)
    dr = home_elo + HFA - away_elo

    # Calculate draw probability using the given equation
    P_draw = (1 / (math.sqrt(2 * math.pi) * math.e)) * math.exp(-((dr / 200) **␣
↪2) / (2 * math.e ** 2))

    # Calculate raw probabilities for home and away wins
    P_home = (1 / (1 + 10 ** (-((home_elo + HFA) - away_elo) / 400))) - (1/2) *␣
↪P_draw
```

```
    P_away = (1 / (1 + 10 ** (-(away_elo - (home_elo + HFA)) / 400))) - (1/2) *␣
↪P_draw

    # Normalize probabilities to ensure they sum to 1
    total = P_home + P_away + P_draw
    P_home /= total
    P_away /= total
    P_draw /= total

    return {
        "Home Win Probability": P_home,
        "Draw Probability": P_draw,
        "Away Win Probability": P_away
    }
```

[29]:
```
def calc_points(player_dict: dict) -> None:
    """
    Calculate predicted FPL points for each player using all available␣
↪probabilities and averages.

    Args:
        player_dict (dict): Player details dictionary.

    Updates:
        player_dict: Adds 'xP by Bookmaker Odds' and 'xP by Historical Data'␣
↪for each player.
    """
    for player, odds in player_dict.items():
        try:
            # Get probabilities
            team = odds.get("Team", ["Unknown"])[0]
            number_of_games = len(odds.get("Opponent", [])) if team !=␣
↪'Unknown' else 1
            avg_min_per_game = odds.get("Average Minutes per Game", [90])[0] if␣
↪team != 'Unknown' else 90
            goals_average1 = odds.get("xG by Bookmaker Odds", [])
            goals_average2 = odds.get("xG by Historical Data", [])
            ass_average1 = odds.get("xA by Bookmaker Odds", [])
            ass_average2 = odds.get("xA by Historical Data", [])
            cs_odds1 = odds.get("Clean Sheet Probability by Bookmaker Odds", [])
            cs_odds2 = odds.get("Clean Sheet Probability by Historical Data",␣
↪[])
            position = odds.get("Position", ["Unknown"])[0]
            saves_average1 = odds.get("xSaves by Bookmaker Odds", [])
            saves_average2 = odds.get("xSaves by Historical Data", [])
```

```
            goals_scored_team_bookmaker = odds.get('Goals Scored by Team on␣
↪Average', [])
            goals_scored_team_historical = odds.get('Team xG by Historical␣
↪Data', [])
            total_goals_scored_team_average = goals_scored_team_bookmaker if␣
↪goals_scored_team_bookmaker != [] else goals_scored_team_historical

            goals_conceded_team_bookmaker = odds.get('Goals Conceded by Team on␣
↪Average', [])
            goals_conceded_team_historical = odds.get('Team xGC by Historical␣
↪Data', [])
            total_goals_conceded_team_average = goals_conceded_team_bookmaker␣
↪if goals_conceded_team_bookmaker != [] else goals_conceded_team_historical

            win_probability =  odds.get('Win Probability', [])
            elo_win_probability =  odds.get('ELO Win Probability', [])
            draw_probability =  odds.get('Draw Probability', [])
            elo_draw_probability =  odds.get('ELO Draw Probability', [])
            MGR_Bonus = odds.get('Manager Bonus', [])
            chance_of_playing = odds.get("Chance of Playing", [1])[0] if team !
↪= 'Unknown' else 1
            avg_bonus_points = odds.get("Average Bonus Points per Game", [])

            # If there are more probability/average entries than number of␣
↪games in the gameweek for a player, skip the player
            if len(goals_average1) > number_of_games or len(ass_average1) >␣
↪number_of_games or len(saves_average1) > number_of_games:
                print(f"{player} skipped due to data entries being higher than␣
↪number of games the player is playing")
                continue
            points = 0
            points2 = 0
            ass_average1 = odds.get("Expected Assists per Game", []) if␣
↪len(ass_average1) == 0 else ass_average1
            goals_average1 = odds.get("Expected Goals per Game", []) if␣
↪len(goals_average1) == 0 else goals_average1
            saves_average1 = odds.get("xSaves by Historical Data", []) if␣
↪len(saves_average1) == 0 else saves_average1

            # Calculate points
            if position in ('MID'):
                points = chance_of_playing * (
                sum(avg_bonus_points) + number_of_games * 2 +␣
↪sum(goals_average1) * 5 +
                sum(ass_average1) * 3 + sum(cs_odds1))
```

```
                points2 = chance_of_playing * min((avg_min_per_game/90), 1) * (
                sum(avg_bonus_points) + number_of_games * 2 +␣
↪sum(goals_average2) * 5 +
                sum(ass_average2) * 3 + sum(cs_odds2))
            if position in ('DEF'):
                points = chance_of_playing * (
                sum(avg_bonus_points) + number_of_games * 2 +␣
↪sum(goals_average1) * 6 +
                sum(ass_average1) * 3 + sum(cs_odds1) * 4
                - (sum(total_goals_conceded_team_average)/2))

                points2 = chance_of_playing * min((avg_min_per_game/90), 1) * (
                sum(avg_bonus_points) + number_of_games * 2 +␣
↪sum(goals_average2) * 6 +
                sum(ass_average2) * 3 + sum(cs_odds2) * 4
                - (sum(goals_conceded_team_historical)/2))
            if position in ('GKP'):
                points = chance_of_playing * (
                sum(avg_bonus_points) + number_of_games * 2 +␣
↪sum(saves_average1)/3
                + sum(cs_odds1) * 4 - (sum(total_goals_conceded_team_average)/
↪2))

                points2 = chance_of_playing * min((avg_min_per_game/90), 1) * (
                sum(avg_bonus_points) + number_of_games * 2 +␣
↪sum(saves_average2)/3
                + sum(cs_odds2) * 4 - (sum(goals_conceded_team_historical)/2))
            if position in ('FWD'):
                points = chance_of_playing * (
                sum(avg_bonus_points) + number_of_games * 2 +␣
↪sum(goals_average1) * 4 +
                sum(ass_average1) * 3)

                points2 = chance_of_playing * min((avg_min_per_game/90), 1) * (
                sum(avg_bonus_points) + number_of_games * 2 +␣
↪sum(goals_average2) * 4 +
                sum(ass_average2) * 3)
            if position in ('Unknown'):
                points = chance_of_playing * (
                sum(avg_bonus_points) + number_of_games * 2 +␣
↪sum(goals_average1) * 4 +
                sum(ass_average1) * 3)

                points2 = 0
            if position in ('MNG'):
                points = 0
```

```
                points2 = 0
                if len(win_probability) > 0:
                    for w, elo_w, d, elo_d, b in zip_longest(win_probability,␣
↪elo_win_probability, draw_probability, elo_draw_probability, MGR_Bonus,␣
↪fillvalue=0):
                        points += w * 6 + d * 3
                        points2 += elo_w * 6 + elo_d * 3
                        # If Manager Bonus is True
                        if b == 'True':
                            points += w * 10 + d * 5
                            points2 += elo_w * 10 + elo_d * 5
                    points += sum(cs_odds1) * 2 +␣
↪sum(total_goals_scored_team_average)
                    points2 += sum(cs_odds2) * 2 +␣
↪sum(goals_scored_team_historical)

            player_dict[player]['xP by Bookmaker Odds'] = round(points, 3)
            player_dict[player]['xP by Historical Data'] = round(points2, 3)
        except Exception as e:
            print(f"Could not calculate points for {player}: {e}")
```

1.5 Main Execution: Data Integration and Prediction

The following cells execute the main workflow: fetching data, preparing structures, scraping odds, calculating probabilities, and updating player statistics.

[30]:
```
# --- Main execution: Fetch data, prepare structures, and start scraping ---
data, teams_data, players_data, team_id_to_name, player_id_to_name =␣
↪fetch_fpl_data()
fixtures = get_all_fixtures()
next_gws = get_next_gws(fixtures, extra_gw = 'False')
next_fixtures = get_next_fixtures(fixtures, next_gws)
teams_playing = print_and_store_next_fixtures(next_fixtures, team_id_to_name)
element_types = position_mapping(data)
teams_positions_map = teams_league_positions_mapping(teams_data)
team_stats_dict, player_stats_dict = construct_team_and_player_data(data,␣
↪team_id_to_name, player_id_to_name, fixtures)
player_dict = player_dict_constructor(players_data, team_stats_dict,␣
↪player_stats_dict, element_types, team_id_to_name)
driver = uc.Chrome() # Replace with the path to your WebDriver if needed
match_dict = fetch_all_match_links(next_fixtures, team_id_to_name,␣
↪teams_positions_map, driver)
```

```
Predicted Points Will Be Calculated for The Following Fixtures:

GW38 Bournemouth v. Leicester
GW38 Fulham v. Man City
GW38 Ipswich v. West Ham
```

```
GW38 Liverpool v. Crystal Palace
GW38 Man Utd v. Aston Villa
GW38 Newcastle v. Everton
GW38 Nottingham Forest v. Chelsea
GW38 Southampton v. Arsenal
GW38 Tottenham v. Brighton
GW38 Wolverhampton v. Brentford

Ad did not pop up
```

[31]:
```python
# Loop through each match, fetch odds, calculate probabilities, and update␣
 ↪player_dict.
counter = 0
for match, details in match_dict.items():
    counter += 1
    print('')
    print(f"{counter}/{len(match_dict)} Fetching odds for {match}")
    home_team_name = details.get('home_team', 'Unknown')
    away_team_name = details.get('away_team', 'Unknown')
    home_team = TEAM_NAMES_ODDSCHECKER.get(home_team_name, home_team_name)
    away_team = TEAM_NAMES_ODDSCHECKER.get(away_team_name, away_team_name)
    link = details.get('Link', 'Link not found')

    fetch_win_market_odds(details, driver, player_dict, team_stats_dict)

    if home_team is not None and away_team is not None:
        calc_team_xgs(home_team, away_team, team_stats_dict, player_dict)
    else:
        # Handle the case where home_team or away_team is None
        print("Error calculating xG by Teams: home_team or away_team is None")

    if link == 'Link not found':
        print(f"Link not found for {match}. Skipping.")
        continue

    odd_type = 'Player Assists'
    ass_odds_dict = fetch_odds(odd_type, driver)
    if ass_odds_dict:
        if home_team is not None and away_team is not None:
            get_player_over_probs(odd_type, ass_odds_dict, player_dict,␣
 ↪home_team, away_team)
        else:
            # Handle the case where home_team or away_team is None
            print("Error adding Player Assists: home_team or away_team is None")

    odd_type = 'Goalkeeper Saves'
    saves_odds_dict = fetch_odds(odd_type, driver)
```

```
    if saves_odds_dict:
        if home_team is not None and away_team is not None:
            get_player_over_probs(odd_type, saves_odds_dict, player_dict,␣
↪home_team, away_team)
        else:
            # Handle the case where home_team or away_team is None
            print("Error adding Goalkeeper Saves: home_team or away_team is␣
↪None")

    odd_type = 'To Score A Hat-Trick'
    hattrick_odds_dict = fetch_odds(odd_type, driver)
    if hattrick_odds_dict:
        if home_team is not None and away_team is not None:
            add_probs_to_dict(odd_type, hattrick_odds_dict, player_dict,␣
↪home_team, away_team)
        else:
            # Handle the case where home_team or away_team is None
            print("Error adding To Score A Hat-Trick: home_team or away_team is␣
↪None")

    odd_type = 'Total Home Goals'
    total_home_goals_dict = fetch_odds(odd_type, driver)
    total_home_goals_probs = get_total_goals_over_probs(total_home_goals_dict,␣
↪"home") if total_home_goals_dict else None

    odd_type = 'Total Away Goals'
    total_away_goals_dict = fetch_odds(odd_type, driver)
    total_away_goals_probs = get_total_goals_over_probs(total_away_goals_dict,␣
↪"away") if total_away_goals_dict else None

    total_combined_goals_dict = total_home_goals_probs | total_away_goals_probs␣
↪if total_home_goals_probs and total_away_goals_probs else None
    if total_combined_goals_dict:
        if home_team is not None and away_team is not None:
            add_total_goals_probs_to_dict(total_combined_goals_dict, home_team,␣
↪away_team, player_dict)
        else:
            # Handle the case where home_team or away_team is None
            print("Error adding Total Goals: home_team or away_team is None")

    odd_type = 'Anytime Goalscorer'
    anytime_scorer_odds_dict = fetch_odds(odd_type, driver)
    if anytime_scorer_odds_dict:
        if home_team is not None and away_team is not None:
            add_probs_to_dict(odd_type, anytime_scorer_odds_dict, player_dict,␣
↪home_team, away_team)
```

```
        else:
            # Handle the case where home_team or away_team is None
            print("Error adding Anytime Goalscorer: home_team or away_team is␣
 ↪None")

    odd_type = 'To Score 2 Or More Goals'
    to_score_2_or_more_dict = fetch_odds(odd_type, driver)
    if to_score_2_or_more_dict:
        if home_team is not None and away_team is not None:
            add_probs_to_dict(odd_type, to_score_2_or_more_dict, player_dict,␣
 ↪home_team, away_team)
        else:
            # Handle the case where home_team or away_team is None
            print("Error adding To Score 2 Or More Goals: home_team or␣
 ↪away_team is None")

driver.quit()
```

```
1/10 Fetching odds for Bournemouth v Leicester
Ad did not pop up
Found odds for Win Market
Couldn't find or expand section: Player Assists
Couldn't find or expand section: Goalkeeper Saves
Found odds for To Score A Hat-Trick
Found odds for Total Home Goals
Found odds for Total Away Goals
Found odds for Anytime Goalscorer
Found odds for To Score 2 Or More Goals

2/10 Fetching odds for Fulham v Man City
Ad did not pop up
Found odds for Win Market
Found odds for Player Assists
Found odds for Goalkeeper Saves
Found odds for To Score A Hat-Trick
Found odds for Total Home Goals
Found odds for Total Away Goals
Found odds for Anytime Goalscorer
Found odds for To Score 2 Or More Goals

3/10 Fetching odds for Ipswich v West Ham
Ad did not pop up
Found odds for Win Market
Couldn't find or expand section: Player Assists
Couldn't find or expand section: Goalkeeper Saves
Found odds for To Score A Hat-Trick
```

```
Found odds for Total Home Goals
Found odds for Total Away Goals
Found odds for Anytime Goalscorer
Found odds for To Score 2 Or More Goals

4/10 Fetching odds for Liverpool v Crystal Palace
Ad did not pop up
Found odds for Win Market
Found odds for Player Assists
Found odds for Goalkeeper Saves
Found odds for To Score A Hat-Trick
Found odds for Total Home Goals
Found odds for Total Away Goals
Found odds for Anytime Goalscorer
Found odds for To Score 2 Or More Goals

5/10 Fetching odds for Man Utd v Aston Villa
Ad did not pop up
Found odds for Win Market
Found odds for Player Assists
Found odds for Goalkeeper Saves
Found odds for To Score A Hat-Trick
Found odds for Total Home Goals
Found odds for Total Away Goals
Found odds for Anytime Goalscorer
Found odds for To Score 2 Or More Goals

6/10 Fetching odds for Newcastle v Everton
Ad did not pop up
Found odds for Win Market
Couldn't find or expand section: Player Assists
Couldn't find or expand section: Goalkeeper Saves
Found odds for To Score A Hat-Trick
Found odds for Total Home Goals
Found odds for Total Away Goals
Found odds for Anytime Goalscorer
Found odds for To Score 2 Or More Goals

7/10 Fetching odds for Nottingham Forest v Chelsea
Ad did not pop up
Found odds for Win Market
Found odds for Player Assists
Found odds for Goalkeeper Saves
Found odds for To Score A Hat-Trick
Found odds for Total Home Goals
Found odds for Total Away Goals
Found odds for Anytime Goalscorer
Found odds for To Score 2 Or More Goals
```

```
8/10 Fetching odds for Southampton v Arsenal
Ad did not pop up
Found odds for Win Market
Couldn't find or expand section: Player Assists
Couldn't find or expand section: Goalkeeper Saves
Found odds for To Score A Hat-Trick
Found odds for Total Home Goals
Found odds for Total Away Goals
Found odds for Anytime Goalscorer
Found odds for To Score 2 Or More Goals

9/10 Fetching odds for Tottenham v Brighton
Ad did not pop up
Found odds for Win Market
Couldn't find or expand section: Player Assists
Couldn't find or expand section: Goalkeeper Saves
Found odds for To Score A Hat-Trick
Found odds for Total Home Goals
Found odds for Total Away Goals
Found odds for Anytime Goalscorer
Found odds for To Score 2 Or More Goals

10/10 Fetching odds for Wolverhampton v Brentford
Ad did not pop up
Found odds for Win Market
Couldn't find or expand section: Player Assists
Couldn't find or expand section: Goalkeeper Saves
Found odds for To Score A Hat-Trick
Found odds for Total Home Goals
Found odds for Total Away Goals
Found odds for Anytime Goalscorer
Found odds for To Score 2 Or More Goals
```

1.6 Final Calculations and Output

Calculate bonus points, specific probabilities, and final predicted points for all players. Save results to Excel and print the top 5 predicted players by position.

[32]:
```
# Calculate bonus points, specific odds, and final predicted points for all␣
↪players.
calc_avg_bonus_points(player_dict, match_dict)
calc_specific_probs(player_dict)
calc_points(player_dict)
```

[33]:
```
# Create and save DataFrames with all player data and a summary of expected␣
↪points.
player_data_df = pd.DataFrame.from_dict(player_dict, orient='index')
```

```
player_data_df.index.name = 'Player'
# Convert all columns: if value is a list of length 1, replace with the value␣
 ↪contained in the list.
for col in player_data_df.columns:
    player_data_df[col] = player_data_df[col].apply(lambda x: x[0] if␣
 ↪isinstance(x, list) and len(x) == 1 else x)

# Sort players by predicted points and games played.
sorted_player_data_df = player_data_df.sort_values(by=['xP by Bookmaker Odds',␣
 ↪'xP by Historical Data'], ascending=[False, False])

# Create a summary DataFrame for quick comparison.
player_points_df = sorted_player_data_df[['Position', 'Team', 'xP by Bookmaker␣
 ↪Odds', 'xP by Historical Data']]

# Prepare filename using gameweeks.
gws_for_filename = "_".join(str(gw) for gw in next_gws)

# Save results to Excel.
with pd.ExcelWriter(f"gw_{gws_for_filename}_output.xlsx") as writer:
    sorted_player_data_df.to_excel(writer, sheet_name='Data')
    player_points_df.to_excel(writer, sheet_name='Expected Points')
```

[34]:
```
# Print the player with the highest predicted points for each position.
best_mng = player_points_df[player_points_df['Position'].apply(lambda x: 'MNG'␣
 ↪in x)].head(5)
best_gkp = player_points_df[player_points_df['Position'].apply(lambda x: 'GKP'␣
 ↪in x)].head(5)
best_def = player_points_df[player_points_df['Position'].apply(lambda x: 'DEF'␣
 ↪in x)].head(5)
best_mid = player_points_df[player_points_df['Position'].apply(lambda x: 'MID'␣
 ↪in x)].head(5)
best_fwd = player_points_df[player_points_df['Position'].apply(lambda x: 'FWD'␣
 ↪in x)].head(5)

print("Top 5 Players Predicted to Score Highest Points According to Bookmaker␣
 ↪Odds by Position:")
print()
display(best_mng)
display(best_gkp)
display(best_def)
display(best_mid)
display(best_fwd)
```

```
Top 5 Players Predicted to Score Highest Points According to Bookmaker Odds by
Position:
```

Player	Position	Team	xP by Bookmaker Odds	xP by Historical Data
Vitor Manuel De Oliveira Lopes Pereira	MNG	Wolverhampton	9.738	8.264
Mikel Arteta	MNG	Arsenal	8.394	10.483
Eddie Howe	MNG	Newcastle	8.194	7.377
Andoni Iraola	MNG	Bournemouth	8.063	7.618
Arne Slot	MNG	Liverpool	7.949	8.034

Player	Position	Team	xP by Bookmaker Odds	xP by Historical Data
David Raya Martin	GKP	Arsenal	4.542	5.742
Nick Pope	GKP	Newcastle	4.394	4.509
Kepa Arrizabalaga	GKP	Bournemouth	4.243	4.783
Mark Flekken	GKP	Brentford	3.836	4.750
Stefan Ortega Moreno	GKP	Man City	3.765	3.673

Player	Position	Team	xP by Bookmaker Odds	xP by Historical Data
Trent Alexander Arnold	DEF	Liverpool	5.131	
Fabian Schar	DEF	Newcastle	4.553	
Dean Huijsen	DEF	Bournemouth	4.529	
Riccardo Calafiori	DEF	Arsenal	4.470	
Conor Bradley	DEF	Liverpool	4.430	

Player	
Trent Alexander Arnold	4.160
Fabian Schar	4.479
Dean Huijsen	4.042
Riccardo Calafiori	3.035
Conor Bradley	1.919

Player	Position	Team	xP by Bookmaker Odds	\
Mohamed Salah	MID	Liverpool	9.061	
Bukayo Saka	MID	Arsenal	7.084	
Luis Diaz	MID	Liverpool	6.436	
Justin Kluivert	MID	Bournemouth	6.422	
Diogo Teixeira Da Silva	MID	Liverpool	6.243	

Player	xP by Historical Data
Mohamed Salah	8.091
Bukayo Saka	5.001
Luis Diaz	4.449
Justin Kluivert	4.896
Diogo Teixeira Da Silva	2.420

Player	Position	Team	\
Erling Haaland	FWD	Man City	
Ollie Watkins	FWD	Aston Villa	
Omar Marmoush	FWD	Man City	
Matheus Santos Carneiro Da Cunha	FWD	Wolverhampton	
Francisco Evanilson De Lima Barbosa	FWD	Bournemouth	

Player	xP by Bookmaker Odds	\
Erling Haaland	6.598	
Ollie Watkins	6.035	
Omar Marmoush	5.779	
Matheus Santos Carneiro Da Cunha	5.558	
Francisco Evanilson De Lima Barbosa	5.493	

Player	xP by Historical Data
Erling Haaland	5.927
Ollie Watkins	3.946
Omar Marmoush	3.693
Matheus Santos Carneiro Da Cunha	4.276
Francisco Evanilson De Lima Barbosa	4.279