Predicted_Points_All_Players

March 15, 2025

[1289]:
```
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
```

This code scrapes several betting odds from Oddschecker.com, converts the odds to percentages and calculates predicted points for players in the next full gameweek in Fantasy Premier League according to the percentages. In addition to selenium, webdriver has to be installed also. Webdrivers run or drive a browser from inside of your code. Version of webdriver has to match the version of your browser.

Assisting and Goalscoring odds for players are usually available couple of days before the game, so this script is very likely to return empty file or a file containing a lot of missing values if there are still several days until the first game of the gameweek.

Added new functionalities to the code (Predicted_Manager_Points and Predicted_Player_Points) to better match a player from the Oddschecker website with the correct player from the FPL API. Additionally, the code has been improved to allow handling of game weeks where a team may play more than one match.

[1290]:
```
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}")
fixtures = response.json()
```

[1291]:
```
game_weeks = defaultdict(list)
for fixture in fixtures:
    game_weeks[fixture["event"]].append(fixture)
for event in sorted(game_weeks.keys()):
    if all(not fixture['finished'] for fixture in game_weeks[event]):
        next_gameweek =  event
        break
    else:
        next_gameweek = None
```

[1292]:
```
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()
teams_data = data['teams']
players_data = data['elements']
```

[1293]:
```
TEAM_NAMES_ODDSCHECKER = {
    "Nott'm Forest": "Nottingham Forest",
    "Wolves": "Wolverhampton",
    "Spurs": "Tottenham",
    }
```

[1294]:
```
next_gw_fixtures = [fixture for fixture in fixtures if (fixture['event'] ==␣
 ↪next_gameweek) and (fixture['started'] == False)]
team_id_to_name = {team['id']: team['name'] for team in teams_data}
```

[1295]:
```
print("Next Gameweek Fixtures:")
print('')
teams_playing = defaultdict(int)
for fixture in next_gw_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(team_id_to_name[fixture['team_h']], 'v.',␣
 ↪team_id_to_name[fixture['team_a']])
```

```
Next Gameweek Fixtures:
```

```
Everton v. West Ham
Ipswich v. Nott'm Forest
Man City v. Brighton
Southampton v. Wolves
Bournemouth v. Brentford
Arsenal v. Chelsea
Fulham v. Spurs
Leicester v. Man Utd
```

[1296]:
```python
# Function to normalize and prepare names for comparison
def prepare_name(name):
    """
    Normalizes a name by converting to lowercase, removing accents, and␣
 ↪splitting into tokens.
    """

    # Replace Scandinavian letters with their ASCII equivalents
    scandinavian_replacements = {
        'ø': 'o',
        'å': 'a',
        'æ': 'ae',
        'Ø': 'O',
        'Å': 'A',
        'Æ': 'AE',
    }
    for scandinavian_char, ascii_char in scandinavian_replacements.items():
        name = name.replace(scandinavian_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_name2 = cleaned_name.replace("'", '')
    # Convert to lowercase and split into tokens
    return cleaned_name2.lower().split()
```

[1297]:
```python
def teams_league_positions_mapping(teams):
    return {team['id']: team['position'] for team in teams}
```

[1298]:
```python
def position_mapping(data):
    return {et["id"]: et["singular_name_short"] for et in data["element_types"]}
```

[1299]:
```python
def prepare_nickname(nickname):
    nickname1 = nickname
    nickname2 = nickname
    index = nickname1.find(".")
```

```
    while (index != -1):
        if index != len(nickname1) - 1:
            nickname1 = nickname1[:index] + ' ' + nickname1[index+1:]
            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
```

[1300]:
```
def player_dict_constructor(players_data, element_types, team_id_to_name):
    # Initialize player_dict to store lists of values for each key
    player_dict = defaultdict(lambda: defaultdict(list))

    for player in players_data:
        player_name = player["first_name"] + " " + player["second_name"]
        nickname = player['web_name']
        nickname1, nickname2 = prepare_nickname(nickname)

        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_NAMES_ODDSCHECKER.
 ↪get(team_id_to_name[player["team"]], team_id_to_name[player["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

    return player_dict
```

[1301]:
```
def fetch_all_match_links(next_gw_fixtures, team_id_to_name,␣
↪teams_positions_map, driver):
    driver.get("https://www.oddschecker.com/football/english/premier-league/")

    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, 5)
    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_gw_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)
        match_title = home_team + " v " + away_team

        # Find match link
        match_link = driver.find_element(By.XPATH, f"//
↪a[@title='{match_title}'][@href]")
        href = match_link.get_attribute("href")

        matches_details[match_title] = {}
        matches_details[match_title]['home_team'] = home_team
        matches_details[match_title]['away_team'] = away_team
        matches_details[match_title]['Underdog Bonus'] = Underdog_Bonus
        matches_details[match_title]['Link'] = href

    return matches_details
```

[1302]:
```
def fetch_win_market_odds(match_dict, driver, player_dict):
    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')

    try:
        driver.get(link)
        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.find(' ') != -1:
                                odd_text = odd_text.replace(' ', '')
```

```
                            odd_fraction = Fraction(odd_text)
                            odds_list.append(odd_fraction)
                        odds_dict[list(odds_dict)[i]] = odds_list
                        i += 1

                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

                except Exception as e:
                    print("Could not get average odds for Home Win, Away␣
↪Win and/or Draw", e)
                    home_win_prob = 0
                    away_win_prob = 0
                    draw_prob = 0
            except Exception as e:
                print("Couldn't get odds for Win Market", e)
                home_win_prob = 0
                away_win_prob = 0
                draw_prob = 0

        except Exception as e:
            print("Couldn't find Win Market All Odds Section")
            home_win_prob = 0
            away_win_prob = 0
            draw_prob = 0

    except Exception as e:
        print("Could not open Compare All Odds on Win Market, e")
        home_win_prob = 0
        away_win_prob = 0
        draw_prob = 0

except Exception as e:
    print("Could not find Win Market header, e")
    home_win_prob = 0
    away_win_prob = 0
    draw_prob = 0
```

```
    for player in player_dict:
        if player_dict[player]['Position'] == 'MNG':
            if player_dict[player]['Team'] == home_team:
                player_dict[player]['Win Probability'].append(home_win_prob)
                player_dict[player]['Draw Probability'].append(draw_prob)
                if Underdog_Bonus == 'Home':
                    player_dict[player]['Manager Bonus'].append('True')
                else:
                    player_dict[player]['Manager Bonus'].append('False')

            if player_dict[player]['Team'] == away_team:
                player_dict[player]['Win Probability'].append(away_win_prob)
                player_dict[player]['Draw Probability'].append(draw_prob)
                if Underdog_Bonus == 'Away':
                    player_dict[player]['Manager Bonus'].append('True')
                else:
                    player_dict[player]['Manager Bonus'].append('False')
```

[1303]:
```
def fetch_odds(odd_type, driver):
    wait = WebDriverWait(driver, 5)
    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.find(' ') != -1:
                                    odd_text = odd_text.replace(' ', '')
                                odd_fraction = Fraction(odd_text)
                                odds_list.append(odd_fraction)
                            odds_dict[list(odds_dict)[i]] = odds_list
                            i += 1
                        header.click()
                        time.sleep(1)
                        return odds_dict
                    except Exception as e:
                        print("Couldn't get odds for ", list(odds_dict)[i])
                except Exception as e:
                    print("Couldn't get odds for ", outcome, " ", e)
            except Exception as e:
                print(f"Couldn't find {odd_type} All Odds Section", e)
        except Exception as e:
            print(f"Couldn't click Compare All Odds on {odd_type}")
        header.click()
        time.sleep(1)
    except Exception as e:
        print(f"Couldn't find or expand section: {odd_type}")
```

[1304]:
```
def get_player_over_probs(odd_type, odds_dict, player_dict):
    if odd_type == "Player Assists":
        odds_for = ['Over 0.5', 'Over 1.5', 'Over 2.5']
    if odd_type == "Goalkeeper Saves":
        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:
                odd = sum(odds_list)/len(odds_list)
```

```
                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 else 0
            else:
                continue
            try:
                for p in player_dict:
                    # Prepare the player name for comparison
                    player_tokens = prepare_name(p)
                    webname_tokens = prepare_name(name)
                    matched_name = None

                    # 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:
                    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)
                        matched_name = None
                        nickname1 = player_dict[p]['Nickname']
                        nickname2 = player_dict[p]['Nickname2']
                        nickname_tokens = prepare_name(nickname2)

                        if nickname1 in name and (all(token in nickname_tokens␣
↪for token in webname_tokens) or all(token in webname_tokens for token in␣
↪nickname_tokens)):
                            matched_name = p
                            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)
```

[1305]:
```
def get_total_goals_over_probs(odds_dict, team):
    try:
        for team_odd, odds_list in odds_dict.items():
            if team_odd == "Over 0.5":
                team_over_05_odd = sum(odds_list)/len(odds_list)
            if team_odd == "Over 1.5":
                team_over_15_odd = sum(odds_list)/len(odds_list)
            if team_odd == "Over 2.5":
                team_over_25_odd = sum(odds_list)/len(odds_list)
            if team_odd == "Over 3.5":
                team_over_35_odd = sum(odds_list)/len(odds_list)
            if team_odd == "Over 4.5":
                team_over_45_odd = sum(odds_list)/len(odds_list)
        try:
            team_over_05_prob = (1/(float(Fraction(team_over_05_odd)) + 1)) if␣
↪team_over_05_odd else 0
            team_over_15_prob = (1/(float(Fraction(team_over_15_odd)) + 1)) if␣
↪team_over_15_odd else 0
            team_over_25_prob = (1/(float(Fraction(team_over_25_odd)) + 1)) if␣
↪team_over_25_odd else 0
            team_over_35_prob = (1/(float(Fraction(team_over_35_odd)) + 1)) if␣
↪team_over_35_odd else 0
            team_over_45_prob = (1/(float(Fraction(team_over_45_odd)) + 1)) if␣
↪team_over_45_odd else 0
            try:
                team_0_goal_prob = 1 - team_over_05_prob
                team_5_goal_prob = team_over_45_prob
                team_1_goal_prob = team_over_05_prob - team_over_15_prob if␣
↪team_over_05_prob and team_over_15_prob else team_over_05_prob
                team_2_goal_prob = team_over_15_prob - team_over_25_prob if␣
↪team_over_15_prob and team_over_25_prob else max((1 - team_over_25_prob -␣
↪team_1_goal_prob - team_0_goal_prob), 0)
                team_3_goal_prob = team_over_25_prob - team_over_35_prob if␣
↪team_over_25_prob and team_over_35_prob else max((1 - team_over_35_prob -␣
↪team_2_goal_prob - team_1_goal_prob - team_0_goal_prob), 0)
```

```
            team_4_goal_prob = team_over_35_prob - team_over_45_prob if␣
↪team_over_35_prob and team_over_45_prob else max((1 - team_over_45_prob -␣
↪team_3_goal_prob -  team_2_goal_prob - team_1_goal_prob - team_0_goal_prob),␣
↪0)
        except Exception as e:
            print(f"Couldnt calculate probabilities for Total {team.
↪capitalize()} Goals", e)
    except Exception as e:
        print(f"Couldnt calculate probabilities for Total {team.
↪capitalize()} Over Goals", e)
    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}
  except Exception as e:
    print(f"Couldnt find probabilities from odds_dict for Total {team.
↪capitalize()} Over Goals", e)
```

[1306]:
```
def add_total_goals_probs_to_dict(probs_dict, home_team, away_team,␣
↪player_dict):
  for player in player_dict:
    if (player_dict[player]['Team'] == home_team) and␣
↪(player_dict[player]['Position'] in ['MNG', 'GKP', 'DEF', 'MID']):
        player_dict[player]['Clean Sheet Probability'].
↪append(probs_dict["away_0_goal_prob"])
        if player_dict[player]['Position'] in ['GKP', 'DEF']:
            player_dict[player]['Team Concedes 1 Goals Probability'].
↪append(probs_dict["away_1_goal_prob"])
            player_dict[player]['Team Concedes 2 Goals Probability'].
↪append(probs_dict["away_2_goals_prob"])
            player_dict[player]['Team Concedes 3 Goals Probability'].
↪append(probs_dict["away_3_goals_prob"])
            player_dict[player]['Team Concedes 4 Goals Probability'].
↪append(probs_dict["away_4_goals_prob"])
            player_dict[player]['Team Concedes 5 Goals Probability'].
↪append(probs_dict["away_5_goals_prob"])
            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"]
            player_dict[player]['Goals Conceded On Average'].
↪append(home_goals_conceded_average)
        if player_dict[player]['Position'] == 'MNG':
            player_dict[player]['Team Scores 0 Goals Probability'].
↪append(probs_dict["home_0_goal_prob"])
            player_dict[player]['Team Scores 1 Goal Probability'].
↪append(probs_dict["home_1_goal_prob"])
```

```
                player_dict[player]['Team Scores 2 Goals Probability'].
↪append(probs_dict["home_2_goals_prob"])
                player_dict[player]['Team Scores 3 Goals Probability'].
↪append(probs_dict["home_3_goals_prob"])
                player_dict[player]['Team Scores 4 Goals Probability'].
↪append(probs_dict["home_4_goals_prob"])
                player_dict[player]['Team Scores 5 Goals Probability'].
↪append(probs_dict["home_5_goals_prob"])
                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"]
                player_dict[player]['Goals Scored On Average'].
↪append(home_goals_average)
        if (player_dict[player]['Team'] == away_team) and␣
↪(player_dict[player]['Position'] in ['MNG', 'GKP', 'DEF', 'MID']):
            player_dict[player]['Clean Sheet Probability'].
↪append(probs_dict["home_0_goal_prob"])
            if player_dict[player]['Position'] in ['GKP', 'DEF']:
                player_dict[player]['Team Concedes 1 Goals Probability'].
↪append(probs_dict["home_1_goal_prob"])
                player_dict[player]['Team Concedes 2 Goals Probability'].
↪append(probs_dict["home_2_goals_prob"])
                player_dict[player]['Team Concedes 3 Goals Probability'].
↪append(probs_dict["home_3_goals_prob"])
                player_dict[player]['Team Concedes 4 Goals Probability'].
↪append(probs_dict["home_4_goals_prob"])
                player_dict[player]['Team Concedes 5 Goals Probability'].
↪append(probs_dict["home_5_goals_prob"])
                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"]
                player_dict[player]['Goals Conceded On Average'].
↪append(away_goals_conceded_average)
            if player_dict[player]['Position'] == 'MNG':
                player_dict[player]['Team Scores 0 Goals Probability'].
↪append(probs_dict["away_0_goal_prob"])
                player_dict[player]['Team Scores 1 Goal Probability'].
↪append(probs_dict["away_1_goal_prob"])
                player_dict[player]['Team Scores 2 Goals Probability'].
↪append(probs_dict["away_2_goals_prob"])
                player_dict[player]['Team Scores 3 Goals Probability'].
↪append(probs_dict["away_3_goals_prob"])
                player_dict[player]['Team Scores 4 Goals Probability'].
↪append(probs_dict["away_4_goals_prob"])
```

```
                player_dict[player]['Team Scores 5 Goals Probability'].
↪append(probs_dict["away_5_goals_prob"])
                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"]
                player_dict[player]['Goals Scored On Average'].
↪append(away_goals_average)
```

[1307]:
```
def add_probs_to_dict(odd_type, odds_dict, player_dict):
    try:
        for player_odd, odds_list in odds_dict.items():
            name = player_odd.strip()
            odd = sum(odds_list)/len(odds_list)

            for p in player_dict:
                # Prepare the player name for comparison
                player_tokens = prepare_name(p)
                webname_tokens = prepare_name(name)
                matched_name = None

                # 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:

                # Calculate and add the probability
                probability = 1/float(odd + 1)
                if probability is not None:
                    player_dict[matched_name][f"{odd_type} Probability"].
↪append(probability)
                else:
                    player_dict[matched_name][f"{odd_type} Probability"].
↪append(0)

            else:
                for p in player_dict:
                    # Prepare the player name for comparison
                    webname_tokens = prepare_name(name)
                    matched_name = None
                    nickname1 = player_dict[p]['Nickname']
                    nickname2 = player_dict[p]['Nickname2']
                    nickname_tokens = prepare_name(nickname2)
```

```
                    if nickname1 in name and (all(token in nickname_tokens for␣
↪token in webname_tokens) or all(token in webname_tokens for token in␣
↪nickname_tokens)):
                        matched_name = p
                        break

                if matched_name:
                    # Calculate and add the probability
                    probability = 1/float(odd + 1)
                    if probability is not None:
                        player_dict[matched_name][f"{odd_type} Probability"].
↪append(probability)
                    else:
                        player_dict[matched_name][f"{odd_type} Probability"].
↪append(0)

                else:
                    player_dict[name]['Nickname'] = 'Unknown'
                    player_dict[name]['Nickname2'] = 'Unknown'
                    player_dict[name]['Position'] = 'Unknown'
                    player_dict[name]['Team'] = "Unknown"
                    probability = 1/float(odd + 1)
                    if probability is not None:
                        player_dict[name][f"{odd_type} Probability"].
↪append(probability)
                    else:
                        player_dict[name][f"{odd_type} Probability"].append(0)
    except Exception as e:
        print("Couldn't get probability for ", odd_type, " ", e)
```

[1308]:
```
def calc_specific_odds(player_dict):
    for player, odds in player_dict.items():
        if odds.get("Position") in ['DEF', 'MID', 'FWD', 'Unknown']:
            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", [])

            for p25, p15, p05 in zip_longest(assisting_over_25_prob,␣
↪assisting_over_15_prob, assisting_over_05_prob, fillvalue=0):
```

```
                zero_ass_prob = 1 - p05 if p05 != 0 else 1
                three_ass_prob = p25
                one_ass_prob = p05 - p15 if p05 and p15 else max((1 - p15 -␣
↪zero_ass_prob), 0)
                two_ass_prob = p15 - p25 if p15 and p25 else max((1 -␣
↪three_ass_prob - one_ass_prob - zero_ass_prob), 0)
                ass_average = three_ass_prob * 3 + two_ass_prob * 2 +␣
↪one_ass_prob
                player_dict[player]["Assists On Average"].append(ass_average)

            for p3, p2, p1 in zip_longest(hattrick_prob, two_or_more_prob,␣
↪anytime_prob, fillvalue=0):
                zero_goal_prob = 1 - p1 if p1 != 0 else 1
                three_goals_prob = p3
                one_goal_prob = p1 - p2 if p1 and p2 else max((1 - p2 -␣
↪zero_goal_prob), 0)
                two_goals_prob = p2 - p3 if p2 and p3 else max((1 -␣
↪three_goals_prob - one_goal_prob - zero_goal_prob), 0)
                goal_average = three_goals_prob * 3 + two_goals_prob * 2 +␣
↪one_goal_prob
                player_dict[player]["Goals On Average"].append(goal_average)

        if odds.get("Position") in ('GKP'):
            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 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, fillvalue=0):
                zero_saves_prob = 1 - s05 if s05 else 1
                ten_saves_prob = s95 if s95 else 0
                one_saves_prob = s05 - s15 if s05 and s15 else max((1 - s15 -␣
↪zero_saves_prob), 0)
                two_saves_prob = s15 - s25 if s15 and s25 else max((1 -␣
↪one_saves_prob - zero_saves_prob), 0)
                three_saves_prob = s25 - s35 if s25 and s35 else max((1 -␣
↪two_saves_prob - one_saves_prob - zero_saves_prob), 0)
                four_saves_prob = s35 - s45 if s35 and s45 else max((1 -␣
↪three_saves_prob - two_saves_prob - one_saves_prob - zero_saves_prob), 0)
                five_saves_prob = s45 - s55 if s45 and s55 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 and s65 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 and s75 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 and s85 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 and s95 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
                player_dict[player]["Saves On Average"].append(saves_average)
```

[1309]:
```
def calc_points(player_dict, teams_playing):
    for player, odds in player_dict.items():
        try:
            # Get probabilities
            team = odds.get("Team", "Unknown")
            number_of_games = teams_playing[team] if team != 'Unknown' else 1
            goals_average = odds.get("Goals On Average", [])
            ass_average = odds.get("Assists On Average", [])
            cs_odd = odds.get("Clean Sheet Probability", [])
```

```
            position = odds.get("Position", "Unknown")
            saves_average = odds.get("Saves On Average", [])
            goals_scored_average = odds.get("Goals Scored On Average", [])
            goals_conceded_average = odds.get("Goals Conceded On Average", [])

            win_probability =  odds.get('Win Probability', [])
            draw_probability =  odds.get('Draw Probability', [])
            MGR_Bonus = odds.get('Manager Bonus', [])
            chance_of_playing = odds.get("Chance of Playing", 1)

            if len(goals_average) > number_of_games or len(ass_average) >␣
↪number_of_games or len(saves_average) > number_of_games:
                print(f"{player} skipped due to data entries being higher than␣
↪number of games the player is playing")
                continue

            # Calculate points
            if position in ('MID'):
                points = chance_of_playing * (
                number_of_games *
                2 + sum(goals_average) * 5 +
                sum(ass_average) * 3 +
                sum(cs_odd))
            if position in ('DEF'):
                points = chance_of_playing * (
                number_of_games *
                2 + sum(goals_average) * 6 +
                sum(ass_average) * 3 +
                sum(cs_odd) * 4 - (sum(goals_conceded_average)/2))
            if position in ('GKP'):
                points = chance_of_playing * (
                number_of_games * 2 +
                sum(saves_average)/3 +
                sum(cs_odd) * 4 - (sum(goals_conceded_average)/2))
            if position in ('FWD'):
                points = chance_of_playing * (
                number_of_games *
                2 + sum(goals_average) * 4 +
                sum(ass_average) * 3)
            if position in ('Unknown'):
                points = chance_of_playing * (
                number_of_games *
                2 + sum(goals_average) * 4 +
                sum(ass_average) * 3)
            if position in ('MNG'):
                points = 0
                if len(win_probability) > 0:
```

```
                    for w, d, b in zip_longest(win_probability,␣
 ↪draw_probability, MGR_Bonus, fillvalue=0):
                        points += w * 6 + d * 3
                        if b == 'True':
                            points += w * 10 + d * 5
                    points += sum(cs_odd) * 2 + sum(goals_scored_average)

            player_dict[player]['Points'] = round(points, 3)
        except Exception as e:
            print(f"Could not calculate points for {player}: {e}")
```

[1310]:
```
element_types = position_mapping(data)
teams_positions_map = teams_league_positions_mapping(teams_data)
player_dict = player_dict_constructor(players_data, 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_gw_fixtures, team_id_to_name,␣
 ↪teams_positions_map, driver)

for match, details in match_dict.items():
    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)

    fetch_win_market_odds(details, driver, player_dict)

    odd_type = 'Player Assists'
    ass_odds_dict = fetch_odds(odd_type, driver)
    get_player_over_probs(odd_type, ass_odds_dict, player_dict)

    odd_type = 'Goalkeeper Saves'
    saves_odds_dict = fetch_odds(odd_type, driver)
    get_player_over_probs(odd_type, saves_odds_dict, player_dict)

    odd_type = 'To Score A Hat-Trick'
    hattrick_odds_dict = fetch_odds(odd_type, driver)
    add_probs_to_dict(odd_type, hattrick_odds_dict, player_dict)

    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")
    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")
```

```
    total_combined_goals_dict = total_home_goals_probs | total_away_goals_probs

    add_total_goals_probs_to_dict(total_combined_goals_dict, home_team,␣
 ↪away_team, player_dict)

    odd_type = 'Anytime Goalscorer'
    anytime_scorer_odds_dict = fetch_odds(odd_type, driver)
    add_probs_to_dict(odd_type, anytime_scorer_odds_dict, player_dict)

    odd_type = 'To Score 2 Or More Goals'
    to_score_2_or_more_dict = fetch_odds(odd_type, driver)
    add_probs_to_dict(odd_type, to_score_2_or_more_dict, player_dict)

calc_specific_odds(player_dict)

calc_points(player_dict, teams_playing)

player_data_df = pd.DataFrame.from_dict(player_dict, orient='index')
player_data_df.index.name = 'Player'

player_points_df = player_data_df[['Position', 'Team', 'Points']]
sorted_player_points_df = player_points_df.sort_values(by=['Points'],␣
 ↪ascending=False)

with pd.ExcelWriter(f"gw_{next_gameweek}_output.xlsx") as writer:
    player_data_df.to_excel(writer, sheet_name='Data')
    sorted_player_points_df.to_excel(writer, sheet_name='Points')

driver.quit()
```

```
Ad did not pop up
Ad did not pop up
Ad did not pop up
Ad did not pop up
Ad did not pop up
Ad did not pop up
Ad did not pop up
Ad did not pop up
Ad did not pop up
Reece James skipped due to data entries being higher than number of games the
player is playing
Jack Taylor skipped due to data entries being higher than number of games the
player is playing
Jonny Evans skipped due to data entries being higher than number of games the
player is playing
```

[1311]:
```
best_mng = sorted_player_points_df[sorted_player_points_df['Position'] ==␣
 ↪'MNG'].head(1)
best_gkp = sorted_player_points_df[sorted_player_points_df['Position'] ==␣
 ↪'GKP'].head(1)
best_def = sorted_player_points_df[sorted_player_points_df['Position'] ==␣
 ↪'DEF'].head(1)
best_mid = sorted_player_points_df[sorted_player_points_df['Position'] ==␣
 ↪'MID'].head(1)
best_fwd = sorted_player_points_df[sorted_player_points_df['Position'] ==␣
 ↪'FWD'].head(1)
print("Player Predicted to Score Highest Points by Position:")
print()
print(f"Manager:     {best_mng.axes[0].tolist()[0]:20s} {best_mng.
 ↪iloc[0]['Team']:15s} {best_mng.iloc[0]['Points']:5f} Points")
print(f"Goalkeeper:  {best_gkp.axes[0].tolist()[0]:20s} {best_gkp.
 ↪iloc[0]['Team']:15s} {best_gkp.iloc[0]['Points']:5f} Points")
print(f"Defender:    {best_def.axes[0].tolist()[0]:20s} {best_def.
 ↪iloc[0]['Team']:15s} {best_def.iloc[0]['Points']:5f} Points")
print(f"Midfielder:  {best_mid.axes[0].tolist()[0]:20s} {best_mid.
 ↪iloc[0]['Team']:15s} {best_mid.iloc[0]['Points']:5f} Points")
print(f"Forward:     {best_fwd.axes[0].tolist()[0]:20s} {best_fwd.
 ↪iloc[0]['Team']:15s} {best_fwd.iloc[0]['Points']:5f} Points")
```

```
Player Predicted to Score Highest Points by Position:

Manager:     Kieran McKenna       Ipswich          7.820000 Points
Goalkeeper:  José Malheiro de Sá  Wolverhampton    3.480000 Points
Defender:    Rayan Aït-Nouri      Wolverhampton    4.304000 Points
Midfielder:  Justin Kluivert      Bournemouth      5.959000 Points
Forward:     Erling Haaland       Man City         6.039000 Points
```