18.10.2023

# Football Team Evolution

## First week - Progress Report

*Damian Urbański,*
*Jakub Mikuła,*
*Michał Wiśniewski*

**Project Overview:** The goal of our project is to trace the evolution of three football teams: Real Madrid, FC Barcelona, and Atletico de Madrid, by analyzing every match they have played since 1970. We aim to create links between players who participated in winning and loosing matches to understand how the teams have evolved over the years.

**Key Achievements:**

1. **Data Source Discovery:** Our project began with the significant challenge of sourcing detailed match data, including player information, scores, and match dates. After thorough research, we identified the website "https://www.bdfutbol.com/" as a valuable data source for our selected teams. This website provides a comprehensive repository of match records for our teams dating back to 1970.

2. **Web Scraping Script Development:** In the first week, we successfully developed a Python web scraping script that can access the "https://www.bdfutbol.com/" website, extract relevant data, and save it in CSV files for further analysis. This script is a crucial component of our data collection process.

   search_for_players: This function scrapes player names from match pages on the website. It takes the URL ending, team names, and the target team for the project as parameters.

   The **search_for_players** method is designed to retrieve a list of players' names from a specific website based on certain criteria, such as the teams involved in a project. Here is a high-level description of how it works:

   1. The method takes four input parameters: **ending_of_url** (a URL identifier), **team1** (the first team for the project), **team2** (the second team for the project), and **team_for_project** (the team being considered for the project).

   2. It constructs a URL by combining the base URL with the **ending_of_url** parameter to create the specific web page to scrape.

   3. It sends an HTTP GET request to the constructed URL using the **requests.get** method and stores the response in the **response** variable.

   4. If the response status code is 200 (indicating a successful request), it proceeds to parse the HTML content of the page using BeautifulSoup.

5. It searches for all HTML tables with the class name 'taula_estil' on the page and stores them in the **sections** variable.
6. It iterates over four sections, indexed from 0 to 3, using the **section_number** variable.
7. For each section, it extracts the rows within the table, excluding the first row (header row), and stores them in the **rows** variable.
8. It then iterates over each row, extracting the columns within the row, with a particular focus on the player's name, which is stored in the **player_name** variable.
9. It checks the following conditions:
10. If the **player_name** is not empty and **team1** matches the **team_for_project**, and the **section_number** is either 0 or 2, it adds the **player_name** to the **players** list.
11. If the **player_name** is not empty and **team2** matches the **team_for_project**, and the **section_number** is either 1 or 3, it adds the **player_name** to the **players** list.
12. Finally, it returns the **players** list, which contains the names of the players that meet the specified criteria based on the teams involved in the project.

**BeautifulSoup** is a class from the **bs4** (Beautiful Soup 4) library.

**BeautifulSoup** is used to parse the HTML content of a web page that is obtained using the **requests.get** method. It helps in transforming the raw HTML content into a structured and navigable object, which allows the code to search for specific elements in the HTML, such as tables with a specific class attribute (**taula_estil** in this case), and extract data from them.

save_to_excel: This function saves player data to an Excel file, including match details and player names. It is called for each team, and data is saved in separate CSV files.

The **save_to_excel** method is designed to create an Excel file with specific data related to matches involving a given team. Here's a high-level description of how it works:

1. The method takes three input parameters: **name_of_team** (the team of interest), **name_of_excel** (the name for the Excel file to be created), and **matches_list** (a list of match data).
2. It initializes an Excel workbook using the **Workbook()** method and sets the active sheet to **sheet**.
3. It writes the column headers to the Excel sheet, setting up the structure of the data to be saved. The columns include 'WinOrLost,' 'year,' 'team1,' 'team2,' 'score,' and 'players.'
4. It initializes **workbook_row** to 2, as the first row is used for column headers.
5. The method then iterates through the **matches_list**, where each **list_row** represents match data.

6. Inside the loop, it checks if the team specified in **name_of_team** (either as team1 or team2) won the match, and assigns 1 to 'WinOrLost' if true, or 0 if false. This information is written to the Excel sheet.
7. It calls the **search_for_players** function to obtain the list of players for the given match, using the URL ending, team1, and team2 from **list_row**. The player names are retrieved and stored in the **players** list.
8. The method then populates the Excel sheet with other match-related data, such as the year, team1, team2, and score. The players' names are written to subsequent columns.
9. After processing each match, the **workbook_row** is incremented.
10. An exception handling block surrounds the main loop, where the method attempts to save the Excel file as a CSV using the provided **name_of_excel**. If an exception occurs during processing, it saves and closes the workbook and then exits the function.
11. After processing all matches, the method saves the Excel workbook as a CSV file and closes it.

The **Workbook** class is part of the **openpyxl** package and is used to create a new Excel workbook or open an existing one. In this context, the Workbook class is being used to create a new Excel workbook that will be used to store the data generated by the save_to_excel function.

3. **GitHub Repository Creation:** We have created GitHub repository for our project to facilitate collaboration and version control. Our code, including the web scraping script, has been uploaded to this repository.
4. **Data Collection Initiated:** With our web scraping script in place, we initiated data collection from the chosen website. The script captures match details, player information, and match scores. The data is then structured and saved into CSV files for ease of access and analysis.
5. **Verified (players) name duplication:** We extracted and cleaned data from https://www.bdfutbol.com/en/e/e.html into three excel files inside “players check” directory. Then using “Conditional Formatting” -> Highlight Cells Rules -> Duplicate Values on column B (Àlies) and C (Nom) we verified that there are multiple duplicated values in column B, but there is no duplicated values in column C in any of chosen football clubs.
6. **Demo java script:** We managed to create and run simple java script using GraphStream library. Whole javas code is located on github in src_java/GraphExample.java file.
7. **Improved players.py:** We have modified our code in players.py to get full names of players not nicknames (this will ensure that names of players are not repeated). What is more, our modified code now allows us to get data for every match played by specific football team since 1970 (not only won matches). Then we gathered data for all 3 teams.
8. **Added readme.md file:** For the github project with the basic tasks for evolution of football teams to accomplish and the roles of the group members.

9. **Created FootballTeamEvolutionGraph class in Java**: We managed to create animated dynamic graph in Java for specific football teams. Graphs are represented year by year, from 1970 to 2023. Graphs show evolution of football teams.

10. **Modified python files:** we modified python files to gather and prepare for graphs more football teams: Burgos, Cordoba, Albacete, Algeciras, Mirandes, Numancia, Cultural Leonesa, Xerez, Eibar.

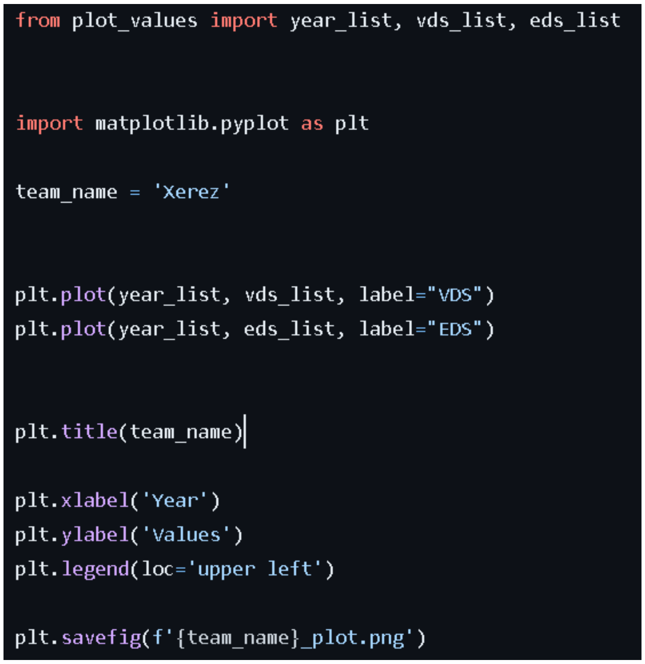
11. **Modified class FootballTeamEvolutionGraph and added plot_line.py:** We modified FootballTeamEvolution by adding a new functionality – calculating Vertical Dynamic Score (VDS) and Edge Dynamic  Score (EDS). Those values are calculated using equations provided below:

$$EDS\colon \quad \frac{|V_{t+1} \Delta V_t|}{|V_{t+1} \cup V_t|},$$

where $|V|$ indicates the number of verticals presented in set V, the $\Delta$ operator in $A \Delta B$ is defined as $A \cup B - A \cap B$.

$$VDS\colon \quad \frac{|E_{t+1} \Delta E_t|}{|E_{t+1} \cup E_t|},$$

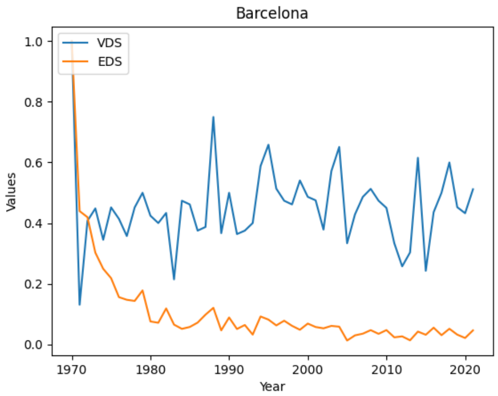
where $|E|$ indicates the number of edges presented in set E, the $\Delta$ operator in $A \Delta B$ is defined as $A \cup B - A \cap B$.
We calculate both values with an annual step, then it is saved in plot_values.py. We have also added plot_line.py file, which is responsible for creating plots for EDS and VDS. This file uses matplotlib library to create plots:

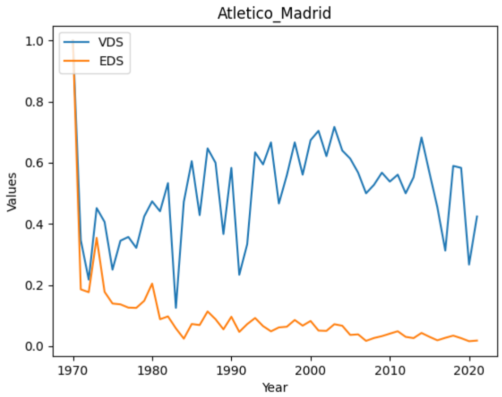


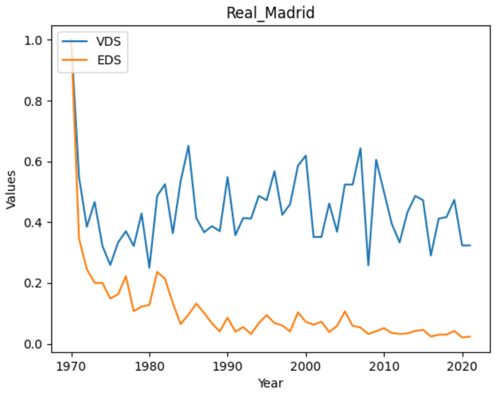
```
from plot_values import year_list, vds_list, eds_list


import matplotlib.pyplot as plt

team_name = 'Xerez'


plt.plot(year_list, vds_list, label="VDS")
plt.plot(year_list, eds_list, label="EDS")


plt.title(team_name)

plt.xlabel('Year')
plt.ylabel('Values')
plt.legend(loc='upper left')

plt.savefig(f'{team_name}_plot.png')
```

All plots are presented below:

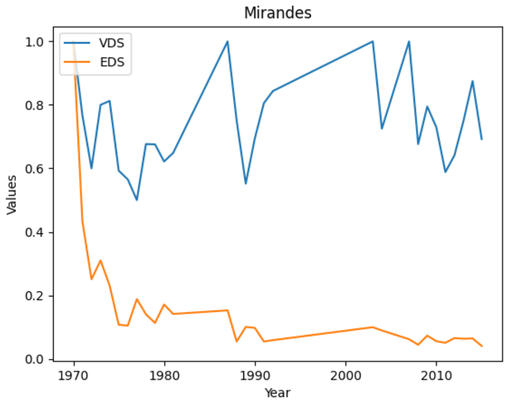
## First group:

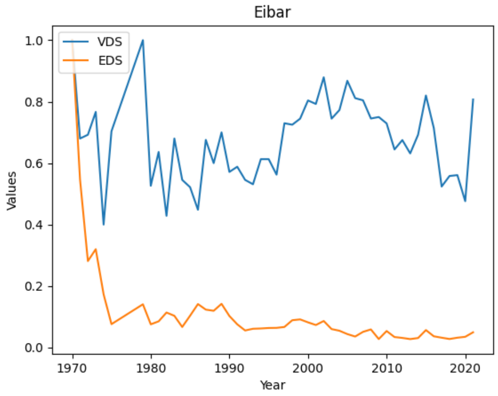


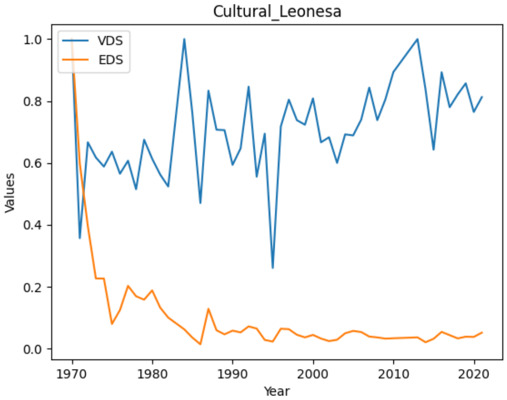


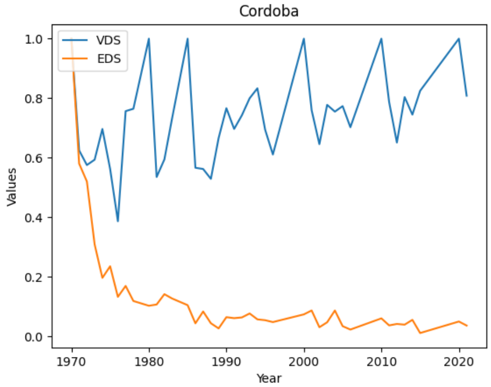


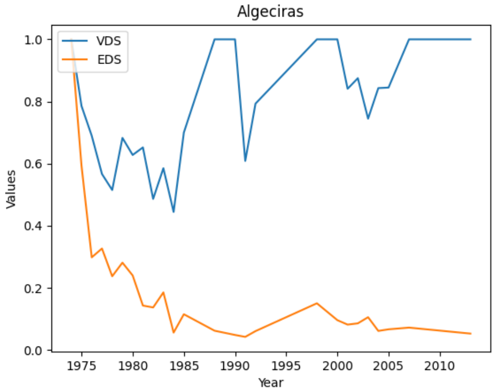
## Second group:

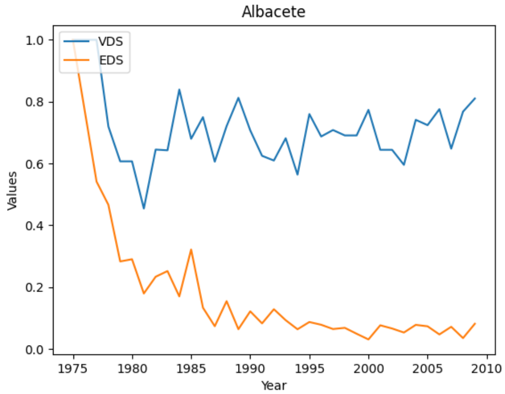


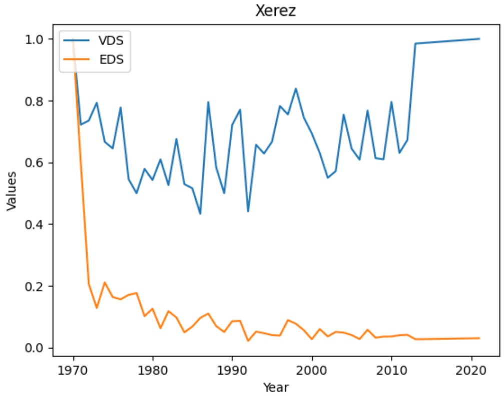


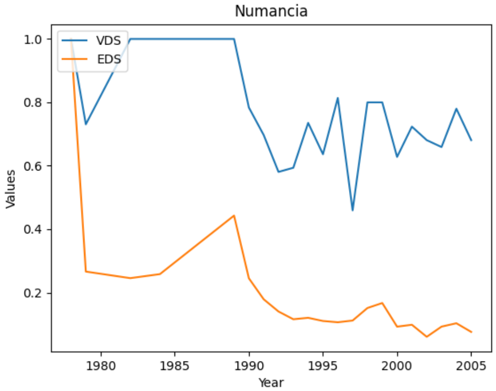












## Analysis:

In the group of teams that have been taken into our analysis, there are three well-known teams that occupy high positions in worldwide rankings (first group), and eight teams that are medium or low ranked (second group). We noticed that EDS tends to zero in every team. This suggests that squads tend to maintain a relatively constant team composition over time.

On the other hand, a difference in VDS can be noticed among the groups of teams. In the first group, VDS stays at a level between 0.2 and 0.7, with the lowest medium score observed in the Real Madrid team. Meanwhile, in the second group, VDS is observed at a higher level. It stays at level approx. 0.6 and higher. This indicates that in the first group, players in each team have developed slightly better teamwork. Winning a match is not as random among squads as in the second group of teams.