Football tournament

TM10002 – 2022-2023

Contents

Introduction

This is a group programming assignment for TM 10002, the Python Programming course for Technical Medicine master students.

Having a common way to format data allows you to automate tasks. In this assignment, you will build a tool that reads the database of a football tournament and generate a simple readable report about the tournament's outcome.

Background

The proposed model of tournament is inspired by the Football world cup. A tournament consists of two phases: the poule phase and the final phase.

Poule phase

In the **poule phase**, all 4×2^n $(n \geq 0)$ competing countries are split in groups of 4 (group 1, group 2, ..., group 2^n). The $2^{n,th}$ best countries (a country of ranking 2 is better than a country of ranking 6) are split in different groups as follow:

- the 1^{st} best country belongs to group 1,
- the 2^{nd} best country belongs to group 2,
- ...
- the $2^{n,th}$ best country belongs to group 2^n.

The rest of the countries groups are split randomly. In a group, each country plays once against each other and only the first two countries access the final phase. The **points** are distributed as follow:

- winning a match gives 3 points (e.g. France - the Netherlands: 4 - 0 gives 3 points to France);
- losing a match gives 0 points (e.g. France - the Netherlands: 4 - 0 gives 0 points to the Netherlands);
- having a draw gives 1 point (e.g. Belgium - Spain: 1 - 1 gives 1 point to each country).

By the end of the poule phase, the ranking is determined as follows: first the **number of points** is checked (a country with 10 points would be favored over a country with 3 points), then the **difference of goals** (a country that scored 15 goals and concede 3 goals would be favored over a country that scored 10 goals and concede 7 because $15 - 3 > 10 - 7$) and finally the **ranking** of the country.

Final phase

The **final phase**, consists of $n + 1$ stages. The s^{th} stage $(1 \leq s \leq n + 1)$ consists of 2^{n+1-s} matches opposing a country A to a country B.

In the 1^{st} **stage**:

- the 1^{st} match opposes the winner of group 1 (country A) to the second of group 2^n (country B),
- the 2^{nd} match opposes the winner of group 2 (country A) to the second of group $2^n - 1$ (country B),
- ...
- the $2^{n,th}$ match opposes the winner of group 2^n (country A) to the second of group 1 (country B).

In the s^{th} **stage** $(2 \leq s \leq n + 1)$:

- The m^{th} match opposes the winner of the $(2 \times m - 1)^{th}$ match of the $(s-1)^{th}$ stage (country A) to the winner of the $(2 \times m)^{th}$ match of the $(s-1)^{th}$ stage (country B).

NB: In the final phase, a draw is not possible (due to the potential extra time and penalty stroke competition).

Example

Below is an example of a tournament of $4 \times 2^1 = 8$ countries. Table 1 represents the poule phase and Figure 1 represents the final phase.

Table 1: Poule phase of the tournament. A row of the table represents: a country, *(its ranking)*, **its number of points** and ***(its difference of goals)***

Group 1		Group 2	
France *(1)*	***9 (+5)***	Moldavia *(8)*	***7 (+3)***
Netherlands *(5)*	***6 (+2)***	Italy *(2)*	***4 (+1)***
Spain *(9)*	***1 (-2)***	Germany *(6)*	***4 (+1)***
Belgium *(13)*	***1 (-5)***	Sweden *(11)*	***0 (-5)***

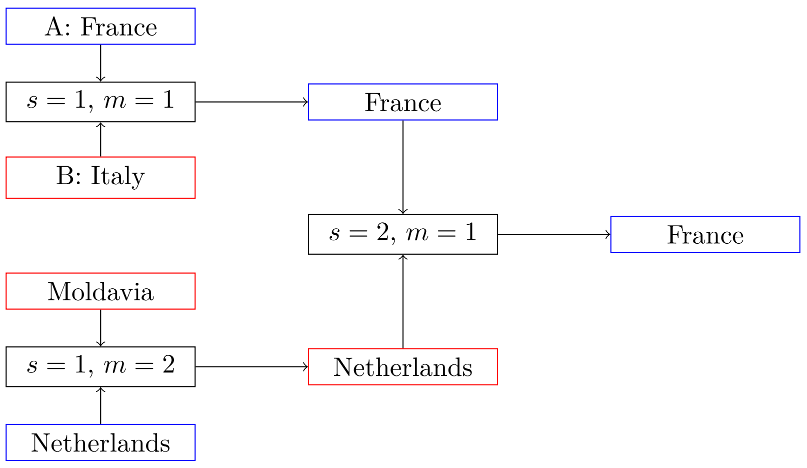

Figure 1: Final phase of the tournament. A red box denotes a losing country while a blue box denote a winning country.

Data

Structure

The information about a tournament take the form of 3 comma separated values files (csv):

- `countries.csv`: contains the country registered in the database formatted in 3 columns:
  - `id`: the country identifier (as `integers`),
  - `country`: the country name (as `strings`),
  - `ranking`: the country ranking (as `integers`)
- `poule_phase_games.csv`: contains the outcome of the match played during the poule phase formatted in 3 columns:
  - `contry_a`: the identifier of country A (as `integers`),
  - `country_b`: the identifier of country B (as `integers`),
  - `score`: the scores the match (as `strings`) formatted as `[score country A] - [score country B]`
- `final_phase_games.csv`: contains the outcome of the match played during the poule phase.
  - `stage`: the stages of match (as `integers`),
  - `match`: the numbers of the match (as `integers`),
  - `score`: the scores the match (including goals scored during the extra time and the penalty stroke competition) (as `strings`) formatted as `[score country A] - [score country B]`

Assumptions

We have the following assumptions on the data:

1. the `countries.csv`, `poule_phase_games.csv` and `final_phase_games.csv` **must** be present in the input directory;
2. the `countries.csv`, `poule_phase_games.csv` and `final_phase_games` **must** be readable and contains the right columns;
3. all countries of the tournament **must** be registered in the database;
4. some countries registered in the database **might not** participate to the tournament, these data are *unnecessary* (to the problem);
5. some scores in `poule_phase_games.csv`, `final_phase_games.csv` **might** be duplicated, these data are *redundant*;
6. some scores in `poule_phase_games.csv` and `final_phase_games.csv` **might** not follow the correct format, the data are *incorrect*;
7. *Unnecessary*, *redundant* and *incorrect* data aside, the poule and final phase **must** have the right amount of matches (see Background);
8. *Unnecessary*, *redundant* and *incorrect* data aside, the final phase **must** have the right amount of stages (see Background);
9. *Unnecessary*, *redundant* and *incorrect* data aside, the final phase **must** have the right amount of matches per stage (see Background);

10. *Unnecessary*, *redundant* and *incorrect* data aside, the poule phase **must** contain all matches in between different countries of a same group exactly once (see Background);

If any of those assumptions is violated the program **must** print an error message in both the standard output and the "Error check" section of the output report and exit.

The assignment

In this assignment, you will build a tool to make sense of a database of a football tournament. You are asked to write a python program that can read in the results of a tournament matches, compute the relevant statistics about each team, plot figures, and assemble all results in a comprehensive report that can be easily viewed. The next sections will address each of these topics in more detail, but first the commandline arguments are explained.

Reading user input

The python program should at least have the following three commandline arguments, of which only the first one (input) is mandatory:

- input directory (`--input`): specifies the directory where the input files are input
- output directory (`--output`): specifies the target output directory name, default value is current directory ('.'); if the output directory does not exist, it should be created
- overwrite (`--overwrite`): specifies if the output directory should be overwritten or not, default value is False.

To do:

1. create a python file `main.py` that accepts the above commandline arguments
2. check if the input violates assumptions 1 and 2

Pre-processing data

Data pre-processing the data is an important part of programming. In this assignment, you will check if the assumptions on the data are violated and prepare the data for analysis.

To do:

1. check if the data provided violates assumption 3
2. check if the data provided violates assumptions 7, 8, 9 and 10

3. for each parsed csv store the number rows, the number of columns, the number of *unnecessary* rows, the number of *redundant* rows and the number of *incorrect* rows.

Processing data

The objective of this part is to make sense of the data obtained in the previous step.

To do:

1. retrieve the groups of the poule phase
2. Per group and for each country (present in the poule phase) store the number of goals scored, the number of goals conceded, the number of points and the ranking in the group
3. retrieve the different matches of the final phase
4. for each country present in the final phase store the number of goals scored, the number of goals conceded and the number of match wons

Writing results

To understand better what happened during the tournament, we would like to have tables (in the form of csv files) that present the poule and the final phase.

To do:

1. for each group, produce a `<output-dir>/csv/results_poule_[group].csv` with the country name, the ranking (from `countries.csv`), the number of points and the difference of goals of each country sorted by ranking in the group (aggregating the same information as Table 1)
2. produce a `<output-dir>/csv/results_final_phase.csv` the ranking (from `countries.csv`), the difference of goals of each country sorted by number of match wons in the final phase

Vizualizing data

Vizualizing the data of the tournament will complement nicely the results

To do:

1. A vertical barplot `<output-dir>/plot/barplot.png` of the number of goals scored per country during the poule phase in descending order (with name of the country on the horizontal axis).
2. A Spearman rank correlation[[1]] plot `<output-dir>/plot/spearman.png` of the average number of scored goal per match during the whole tournament as function of the country ranking with the perfect correlation ($y = x$) represented as a dotted red line (the 5^{th} best country with the

[1] see "Spearman Rank Correlation"

7^{th} best average number of scored goal would be positioned at $x = 5$ and $y = 7$).

Producing report

Write a report in markdown (a simple text format[2]). It must contain the following sections:

- Title: `<input-directory>` Results
- Section "How Created" shows the python call used to generate the report
- Section "Input Data" summarizes the csv files read: number of rows column, number of columns, number of *unnecessary* rows, number of *redundant* rows, and the number of *incorrect* rows
- Section "Tournament" shows the data of the tables generated in the "Writing results" section
- Section "Plots" shows the plots generated in the "Visualizing data" section
- Section "Error check" reports any violations of the assumptions on the data

To do:

1. Produce the report

Some hints and tips

Each of the sections above could be a task that is programmed independently. When you define what each steps creates, and how it is 'transferred' to the next step, the work can be divided over the group.

Markdown writing is not so difficult, it is a plain text format. Still, you can generate nice output from it, and thus it is a nice format to use when creating report automatically from Python. Visual Studio Code also natively supports markdown (.md files). For better experience, you may install the following extensions:

- Markdown All in One
- Markdown Preview Enhanced (e.g. for the equations support)

There are templating solutions to create text files/reports similar to a large `format`. Something like the `jinja` package allows a more advanced way of templating that might be useful. It is not required to use such tools, but you may use them.

In markdown mode (file extension .md), you can use Ctrl+Shift+V to switch between markdown editing and markdown preview mode. You may test this on the exam-metrics file, or the toets-report-fake markdown file in the output

[2]see for example "Markdown Guide"

folder. You may also check whether you can generate an html or pdf (via html, or use pandoc) from the markdown files.

Provided with the assignment (this document in pdf) is:

- the source files (md and html) of the assignment, to illustrate how Markdown works. The html is generated using Visual Studio Code (Ctrl+Shift+P, Markdown All in One: Print current document to HTML). The pdf was generated using pandoc. You do not need to produce pdfs in this assignment.
- a directory named `clean_data` with some csv files that represent a simple example. Note that your code must be able to handle more complex examples within the bounds of the assumptions made in the section Data.
- a directory output that shows example output:
  - report-fake.md: an example (fake) report, to show how the report should look like. Note that all computed. Note that those are toy data that does not correspond to the `clean_data` folder.
  - csv: a folder with example tables. Note that those are toy data that does not correspond to the `clean_data` folder.
  - plot: a folder with example images. Note that those are toy data that does not correspond to the `clean_data` folder.