MapReduce: An Olympic Games case study

G. Bouziotopoulos, V. Konstantakos
MSc in Data Science: Big Data Management
NCSR "Demokritos" & University of the Peloponnese

1 Introduction

The modern Olympic Games are leading international sporting events with thousands of athletes participating in a variety of competitions. They were created based on the ancient Olympic Games, which were held in Olympia, Greece, from the 8th century BC to the 4th century AD. The modern Olympics include all the Games from Athens 1896 to Rio 2016 and are considered the world's foremost sports competition. However, the Olympics are more than just a quadrennial multi-sport world championship. They are also a lens through which we can understand the global history, including political power dynamics, women's empowerment, and the evolving values of society.

In this report, we study some major patterns in Olympic history using the MapReduce paradigm. Which athletes have won gold medals? Who are the top athletes of all time? How many female athletes have participated per team, and how did this number change over the years? By answering these questions, our goal is to highlight some interesting aspects of the Olympic history, while showcasing the MapReduce framework in different programming languages (e.g., Java, Pig).

First, we describe the necessary data processing steps to handle the peculiarities of our data. We then present our approach for each question separately. In particular, each section includes pseudocode, examples, and a summary of our findings. We conclude with a comparison between the two implementations.

2 System requirements

The project was implemented both in Java and Pig, using Linux (PoP!_OS and Ubuntu distributions).

Java requirements

- **Hadoop 3.3** was installed using the instructions and requirements from the official site - **here**, in standalone mode.
- **Java 8** was used (*OpenSDK version 1.8.0_292*).
- **Apache Maven** Version 3.6.3.

For the Java part the **Jetbrains Intelij IDE** was used and the project was build using the maven framework.

Apache Pig requirements

- **Hadoop 2.10.1** was installed in standalone mode, using the instructions and requirements from the official site - **here**.
- **Java 8** was used (*OpenSDK version 1.8.0_292*).
- **Apache Pig 0.17.0** was used to get the latest features and operators.

All the Pig scripts were run in local mode from the command line. We used the interactive mode (i.e., grunt shell) for exploratory analysis, and the batch mode to obtain the final results. Specifically, the resulting files can be obtained using the "*pig -x local*" command for each Pig script. Detailed information about the implementation is included in the provided source code.

Python requirements

Python version **3.7.4** was used for plotting purposes. The required libraries and their corresponding versions are:

- **Pandas 0.25.1**.
- **Matplotlib 3.1.1**.
- **Seaborn 0.9.0**.

3 Project Structure

The project structure is outlined in Figure 1. It includes the following directories:

- **data**: contains the input files
- **java_app**: the main directory of the java application.
  - **src**: contains the source code files.
  - **target**: contains the compiled source code files (this directory can be created when compiling the source files).
- **output_files**: directory that contains the output files.
  - **java**: contains the java output files, including a directory for each java main.
  - **pig**: contains the pig output files, including a directory for each script.
- **pig_scripts**: contains the pig scripts, along with the libraries that were used.

4 Data Processing

After an exploratory data analysis using the "*Exploratory Analyser*" Java file - which uses a Map-Reduce job for parsing the data - we observed that the following columns have missing values, and calculated their percentages using the provided Pig "*Preprocessing*" script:

- Age: 3.49%
- Height: 22.19%
- Weight: 23.19%

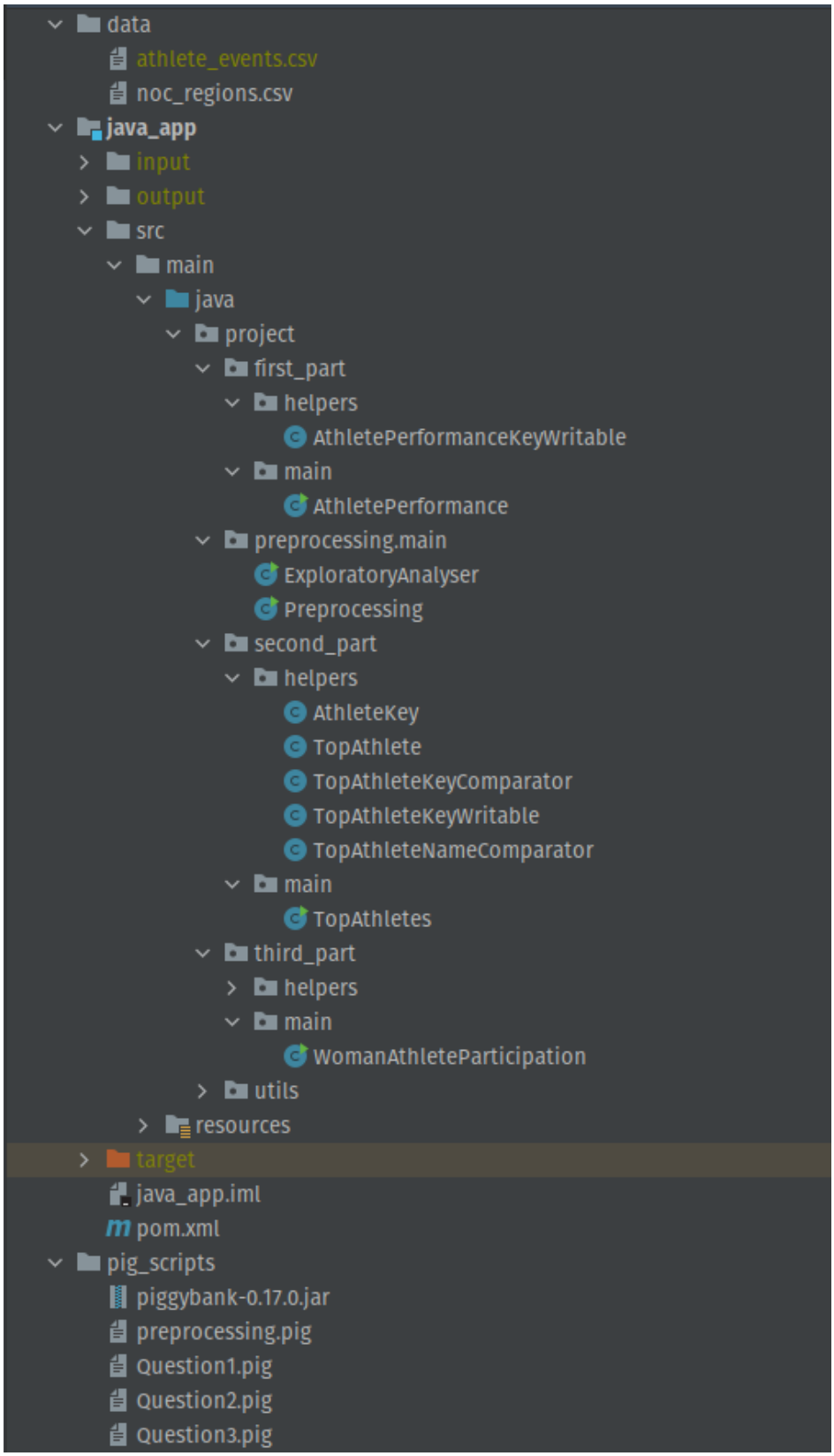


Figure 1: Project Structure

- Medal: 85.33%

Since we are dealing with real data, we decided not to fill or remove any of these values, but rather to keep them as they are. We made that choice because if an athlete's age, height or weight are **null**, they are not available or unknown to us. Regarding the **null** values for medals, the interpretation was that an athlete with **null** medals is an athlete that has not won a medal in the respective competition.

After handling **null values**, we encountered another issue. There is a big time gap between the first Olympic games (Athens 1896) and the last ones (Rio 2016) in the data. Therefore, many teams have different names although they may belong to the same **NOC**. In order to handle this we used the **noc_regions.csv** from the following **link** and added the following two columns, which were missing:

- BAH, Bahamas.
- AZE, Azerbaijan.

A mapping was constructed between **regions** and **NOCs**, and all the team names were replaced accordingly. The process is described within the "*Preprocessing*" Java file which does the data preprocessing through a Map-Reduce job. The files can be executed as following:

Given the current project structure, from inside the **java_app** directory, the following command will perform the exploratory data analysis and output the results to the *output_files/java/exploratory/part-r-00000* file.

```
mvn compile exec:java -Dexec.mainClass=project.preprocessing.main.
    ExploratoryAnalyser -Dexec.args="../data/athlete_events.csv ../output_files/
    java/exploratory"
```

In addition, this command will perform the data preprocessing and output the results to the *output_files/java/preprocessing/part-r-00000* file.

```
mvn compile exec:java -Dexec.mainClass=project.preprocessing.main.Preprocessing -
    Dexec.args="../data/noc_regions.csv ../data/athlete_events.csv ../output_files
    /java/preprocessing"
```

5 Question 1: Gold Medal count

5.1 Pseudocode

The algorithm that was used for counting the number of gold medals is outlined in Algorithm 1. A schematic representation of the implemented MapReduce algorithm can be seen in Figure 2.

5.2 Implementation

For the Java part, the "*AthletePerformance*" file is used. Assuming that the steps of the preprocessing have been run and the input file **"output_files/java/preprocessing/part-r-00000"** has been created, from inside the **java_app** directory run:

```
mvn compile exec:java -Dexec.mainClass=project.first_part.main.AthletePerformance
    -Dexec.args="../output_files/java/preprocessing/part-r-00000 ../output_files/
   java/part_1"
```

The output will be: **output_files/java/part_1/part-r-00000**.

Similarly, the "*Question1*" Pig script can be run on the same input file to obtain the same results with the Java execution (see Section 8).

Input: Olympic Games Data
Output: Athlete's ID, Name, Gold medals

```
class Mapper
    method map(rowkey id, tuple t)
      if t.medal == 'Gold' then
        emit(id, 1)

class Reducer
    method reduce(key id, counts [c_1, ··· , c_n])
      sum = 0
      for count c in counts do
        sum += c
      emit(id, sum)
```

Algorithm 1: Gold Medal Count

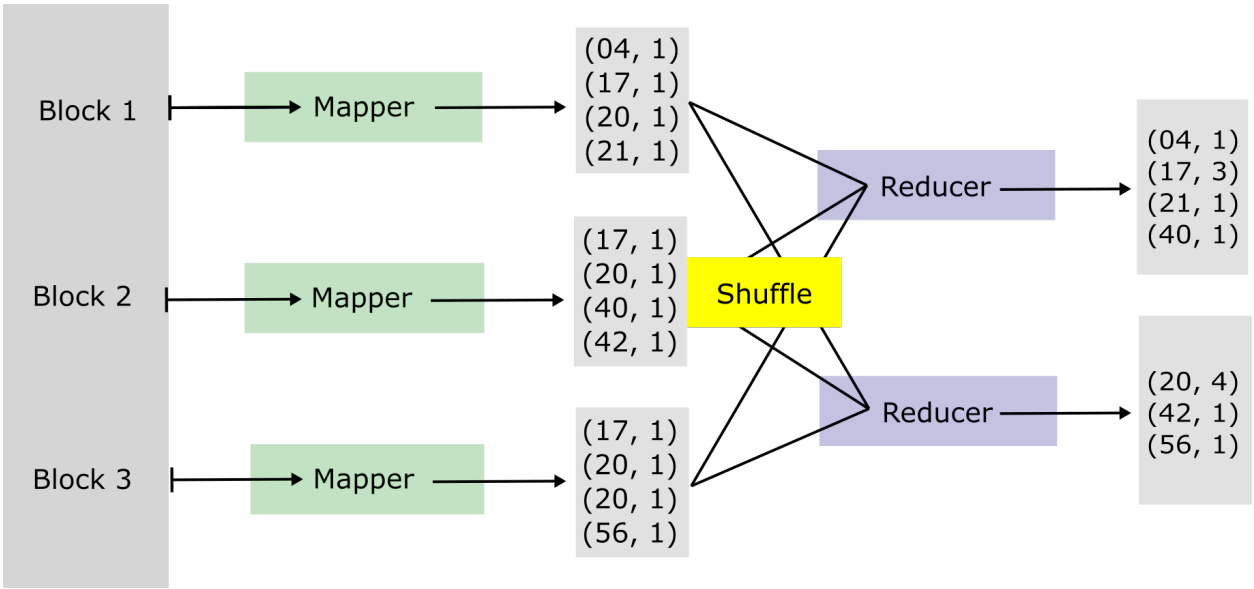


Figure 2: Gold Medal count in MapReduce

5.3 Results

We present a small portion of the obtained results in Table 1. We also illustrate the athletes with the most gold medals in Figure 3.

Table 1: Question 1 Results

ID	Name	Sex	Gold Medals
4	Edgar Lindenau Aabye	M	1
17	Paavo Johannes Aaltonen	M	3
20	Kjetil Andr Aamodt	M	4
21	Ragnhild Margrethe Aamodt	F	1
40	Roald Edgar Aas	M	1
42	Thomas Valentin Aas	M	1
56	Ren Abadie	M	1
72	Aleksey Aleksandrovich Abalmasov	M	1
73	Luc Abalo	M	2
76	Jouan Patrice Abanda Etong	M	1

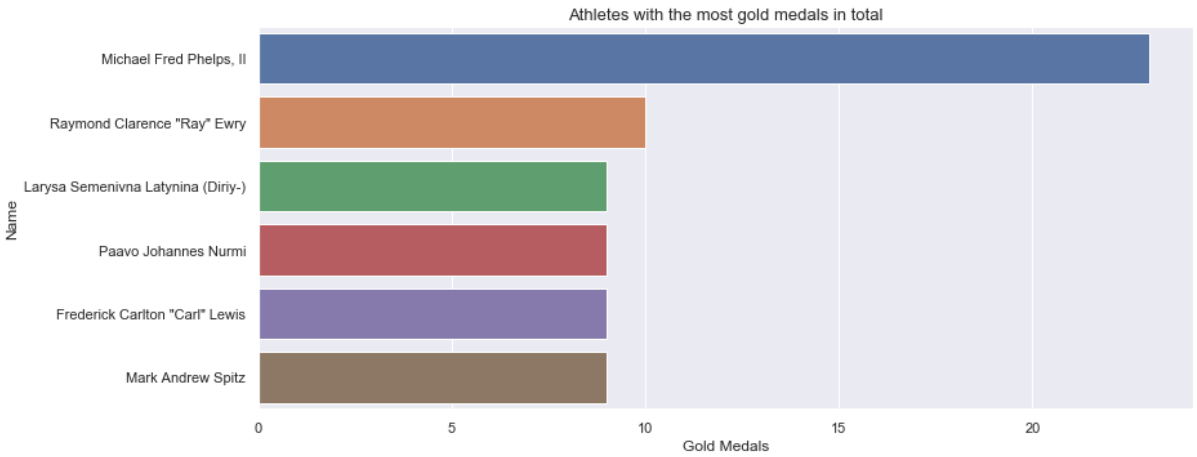


Figure 3: Question 1 Results

6 Question 2: Top Athletes

6.1 Pseudocode

The algorithm that was used for determining the top athletes of all time is outlined in Algorithm 2. A schematic demonstration with certain data examples is shown in Figure 4.

6.2 Implementation

Regarding the Java part, the "*TopAthletes*" file is used. Assuming that the steps of the preprocessing have been run and the input file **"output_files/java/preprocessing/part-r-00000"** has been created, from inside the **java_app** directory run:

Input: Olympic Games Data
Output: Athlete's Name, Games, Gold, Silver, Bronze, Total medals

```
class Mapper
    method map(rowkey id, tuple t)
      if t.medal == 'Gold' then
        emit ((t.name, t.games), 'Gold')
      else if t.medal == 'Silver' then
        emit ((t.name, t.games), 'Silver')
      else if t.medal == 'Bronze' then
        emit((t.name, t.games), 'Bronze')

class Reducer
    method reduce(key (name, games), medals [m_1, ..., m_n])
      gold = 0, silver = 0, bronze = 0;
      for medal m in medals do
        if m == 'Gold' then
          gold += 1;
        else if m == 'Silver' then
          silver += 1;
        else if m == 'Bronze' then
          bronze += 1;
      total_medals = bronze + silver + gold
      emit((t.name, t.games), (t.gold, t.silver, t.bronze, total_medals)
```

Algorithm 2: Top Athletes of all time

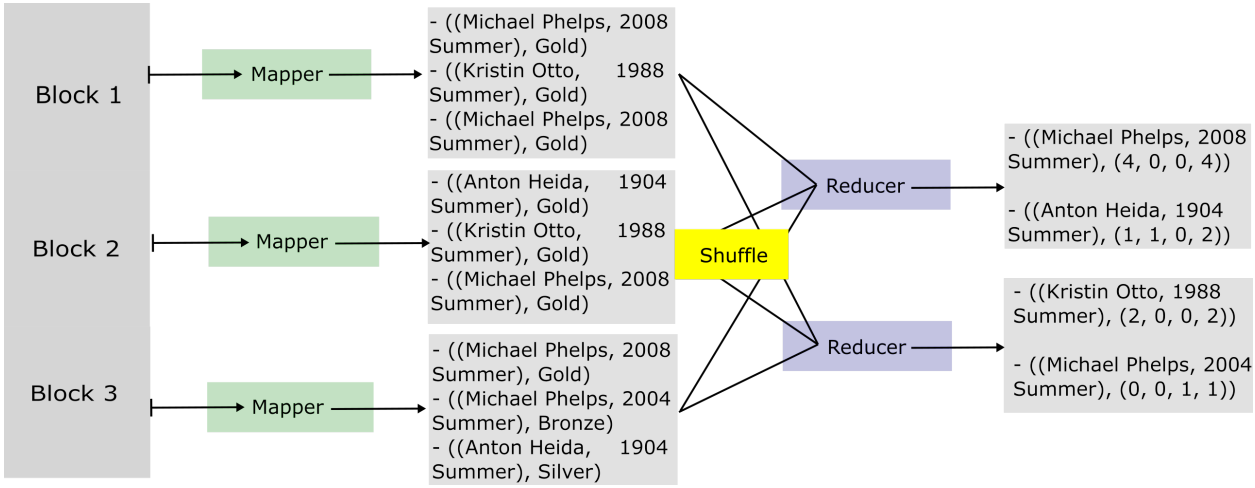


Figure 4: Top Athletes in MapReduce

```
mvn compile exec:java -Dexec.mainClass=project.second_part.main.TopAthletes -
    Dexec.args="../output_files/java/preprocessing/part-r-00000 ../output_files/
    java/part_2"
```

The output will be: **output_files/java/part_2/part-r-00000**.

Likewise, the "*Question2*" Pig script can be run using the command line to get the same results with the Java implementation.

6.3 Results

We present the obtained results in Figure 5. We also demonstrate the top athletes in Figure 6, using the provided Python script.

	A	B	C	D	E	F	G	H	I	J	K
1	1	Michael Fred Phelps, II	M	23	USA	Swimming	2008 Summer	8	0	0	8
2	2	Mark Andrew Spitz	M	22	USA	Swimming	1972 Summer	7	0	0	7
3	3	Michael Fred Phelps, II	M	19	USA	Swimming	2004 Summer	6	0	2	8
4	4	Kristin Otto	F	22	Germany	Swimming	1988 Summer	6	0	0	6
5	4	Vitaly Venediktovich Shcherbo	M	20	Russia	Gymnastics	1992 Summer	6	0	0	6
6	5	Matthew Nicholas "Matt" Biondi	M	22	USA	Swimming	1988 Summer	5	1	1	7
7	5	Willis Augustus Lee, Jr.	M	31	USA	Shooting	1920 Summer	5	1	1	7
8	6	Anton Heida	M	25	USA	Gymnastics	1904 Summer	5	1	0	6
9	6	Michael Fred Phelps, II	M	31	USA	Swimming	2016 Summer	5	1	0	6
10	7	Eric Arthur Heiden	M	21	USA	Speed Skating	1980 Winter	5	0	0	5
11	7	Nedo Nadi	M	25	Italy	Fencing	1920 Summer	5	0	0	5
12	7	Paavo Johannes Nurmi	M	26	Finland	Athletics	1924 Summer	5	0	0	5
13	8	Borys Anfiyanovych Shakhlin	M	28	Russia	Gymnastics	1960 Summer	4	2	1	7
14	8	Lloyd Spencer Spooner	M	35	USA	Shooting	1920 Summer	4	1	2	7
15	8	Nikolay Yefimovich Andrianov	M	23	Russia	Gymnastics	1976 Summer	4	2	1	7
16	9	Akinori Nakayama	M	25	Japan	Gymnastics	1968 Summer	4	1	1	6
17	9	Carl Townsend Osburn	M	35	USA	Shooting	1920 Summer	4	1	1	6
18	9	Gerard Theodor Hubert Van Innis	M	54	Belgium	Archery	1920 Summer	4	2	0	6
19	9	Larysa Semenivna Latynina (Diriy-)	F	21	Russia	Gymnastics	1956 Summer	4	1	1	6
20	9	Michael Fred Phelps, II	M	27	USA	Swimming	2012 Summer	4	2	0	6
21	9	Viktor Ivanovych Chukarin	M	30	Russia	Gymnastics	1952 Summer	4	2	0	6
22	9	Viljo Eino "Ville" Ritola (Koukkari-)	M	28	Finland	Athletics	1924 Summer	4	2	0	6
23	9	Vra slavsk (-Odloilov)	F	26	Czech Rep	Gymnastics	1968 Summer	4	2	0	6
24	9	gnes Keleti-Srkny (Klein)	F	35	Hungary	Gymnastics	1956 Summer	4	2	0	6
25	10	Ecaterina (Katalin-) Szabo (-Tamas	F	16	Romania	Gymnastics	1984 Summer	4	1	0	5
26	10	John Phillips Naber	M	20	USA	Swimming	1976 Summer	4	1	0	5
27	10	Kathleen Genevieve "Katie" Ledeck	F	19	USA	Swimming	2016 Summer	4	1	0	5
28	10	Kornelia Ender (-Matthes, -Grummt	F	17	Germany	Swimming	1976 Summer	4	1	0	5
29	10	Marcus Latimer Hurley	M	20	USA	Cycling	1904 Summer	4	0	1	5
30	10	Melissa Jeanette "Missy" Franklin	F	17	USA	Swimming	2012 Summer	4	0	1	5
31	10	Simone Arianne Biles	F	19	USA	Gymnastics	2016 Summer	4	0	1	5
32	10	Vladimir Nikolayevich Artyomov	M	23	Russia	Gymnastics	1988 Summer	4	1	0	5

Figure 5: Question 2 Results in table form.

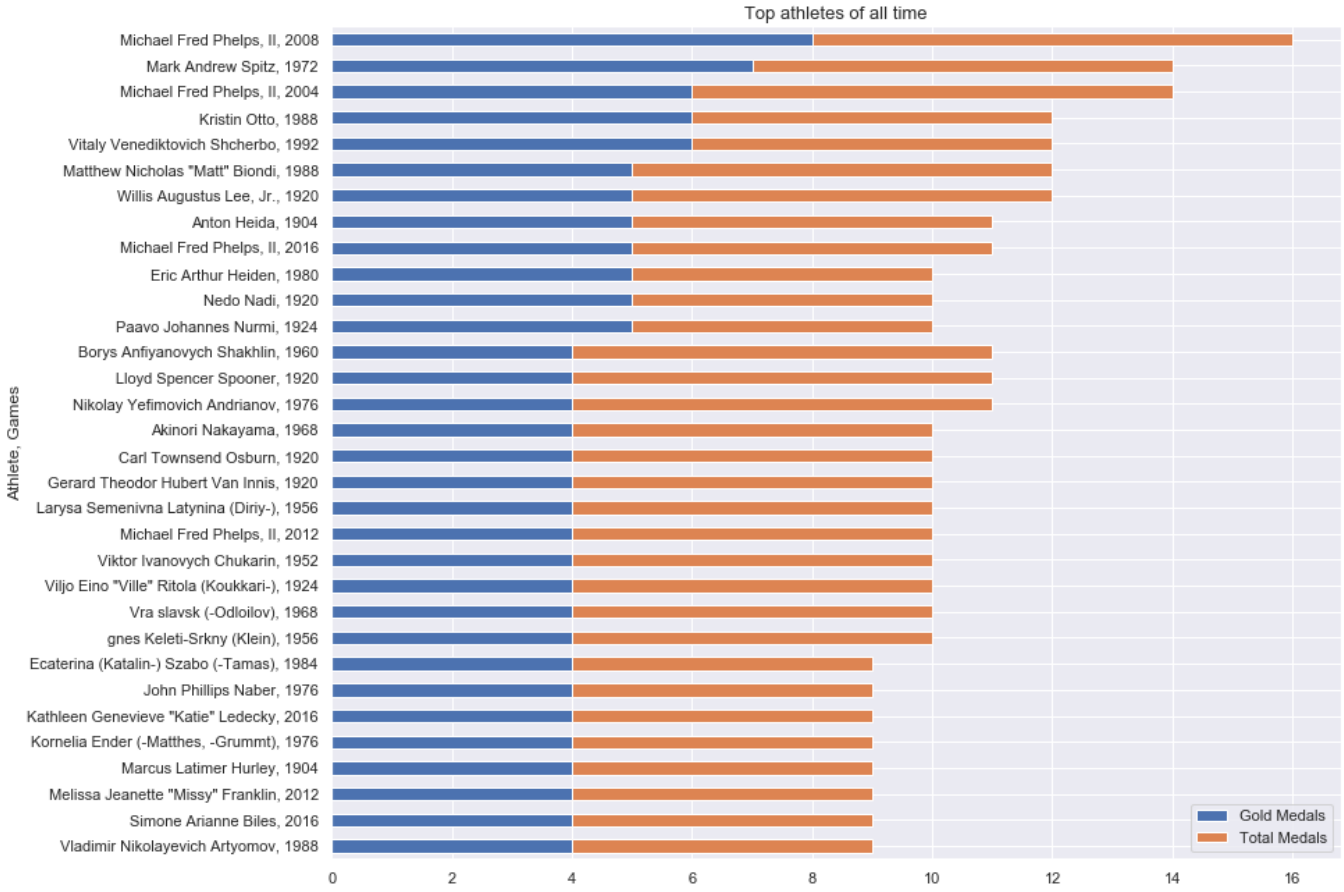


Figure 6: Question 2 Results

7 Question 3: Female Athletes

7.1 Pseudocode

The algorithm that was used to determine the number of female athletes is shown in Algorithm 3. A schematic representation of the implemented MapReduce algorithm can be seen in Figure 7.

```
Input: Olympic Games Data
Output: Games, Team, Women count, Sport

class Mapper_1
    method map(rowkey id, tuple t)
      if t.sex == 'F' then
        emit((t.games, t.team), (t.sport, t.id))

class Reducer_1
    method reduce(key (games, team), tuples [(sport_1, id_1), ..., (sport_n, id_n)])
      unique_ids = new HashSet();
      sport_count_map = new HashMap();
      for (sport, id) in tuples do
        if unique_ids.contain(id) then
          continue;
        unique_ids.puts(id);
        if sport_count_map.containsKey(sport)) then
          sport_count_map.put(sport, sport_count_map.get(sport) + 1);
        else
          sport_count_map.put(sport, 1);
      tMap = new TreeMap();
      for entry in sport_count_map do
        if tMap.containsKey(entry.getValue()) then
          list = tMap.get(entry.getValue());
        else
          list = new List();
          tMap.put(entry.getValue(), list);
        lastKey = tMap.get(tMap.lastKey());
        list.add(entry.getKey());
        if lastKey.size() > 1 then
          lastKey.sort();
        favorite_sport = String.join(" ", lastKey)
        emit(games, (team, unique_ids.size(), favorite_sport));
```

```
class Mapper_2
    method map(key games, tuple (team, women_count, sport))
      emit(key, (team, women_count, sport))

class Reducer_2
    method reduce(key games, tuples
    [(team_1, women_count_1, sport_1) ... (team_n, women_count_n, sport_n)])
      tMap = new TreeMap();
      for (team, women_count, sport) in tuples do
        if tMap.containsKey(women_count) then
          team_list = tMap.get(women_count);
        else
          team_list = new List();
          tMap.put(women_count, team_list);
        team_list.add(team, sport);
      if tMap.size() > 3 then
        tMap.remove(tMap.firstKey());
      for entry in tMap.descendingOrder() do
        list = entry.getValue();
        for tuple t in list do
          emit(games, (entry.getKey(), t.sport, t.team))
```

Algorithm 3: Top 3 female teams per Olympic Games

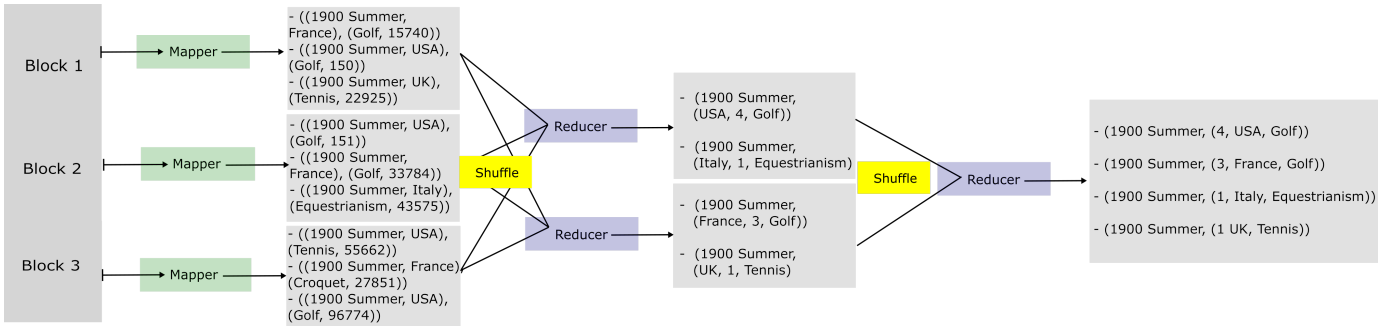


Figure 7: Female Athletes in MapReduce

7.2 Implementation

For the Java part, the "*WomanAthleteParticipation*" file is used. Assuming that the steps of the preprocessing have been run and the input file **"output_files/java/preprocessing/part-r-00000"** has been created, from inside the **java_app** directory run:

```
mvn compile exec:java -Dexec.mainClass=project.third_part.main.
    WomanAthleteParticipation -Dexec.args="../output_files/java/preprocessing/part
    -r-00000 ../output_files/java/part_3"
```

The output will be: **java_app/output/part_3/out1/part-r-00000** and **output_files/java/part_3/out2/part-r-00000** since the above file uses 2 MR Jobs; the latter one has the results.

In a similar fashion, run the "*Question3*" Pig script using the command "*pig -x local*" to confirm the results of the Java implementation.

7.3 Results

We present a small portion of the obtained results in Table 2. We also illustrate the participation of female athletes over the years in Figure 8.

Table 2: Question 3 Results

Games	Team	NOC	Female Athletes	Sport
1900 Summer	France	FRA	12	Golf
1900 Summer	USA	USA	7	Golf
1900 Summer	Czech Republic	BOH	1	Tennis
1900 Summer	Italy	ITA	1	Equestrianism
1900 Summer	Switzerland	SUI	1	Sailing
1900 Summer	UK	GBR	1	Tennis
1904 Summer	USA	USA	6	Archery
1906 Summer	Greece	GRE	5	Tennis
1906 Summer	France	FRA	1	Tennis
1908 Summer	UK	GBR	39	Archery
1908 Summer	Sweden	SWE	3	Tennis
1908 Summer	Germany	GER	2	Figure Skating

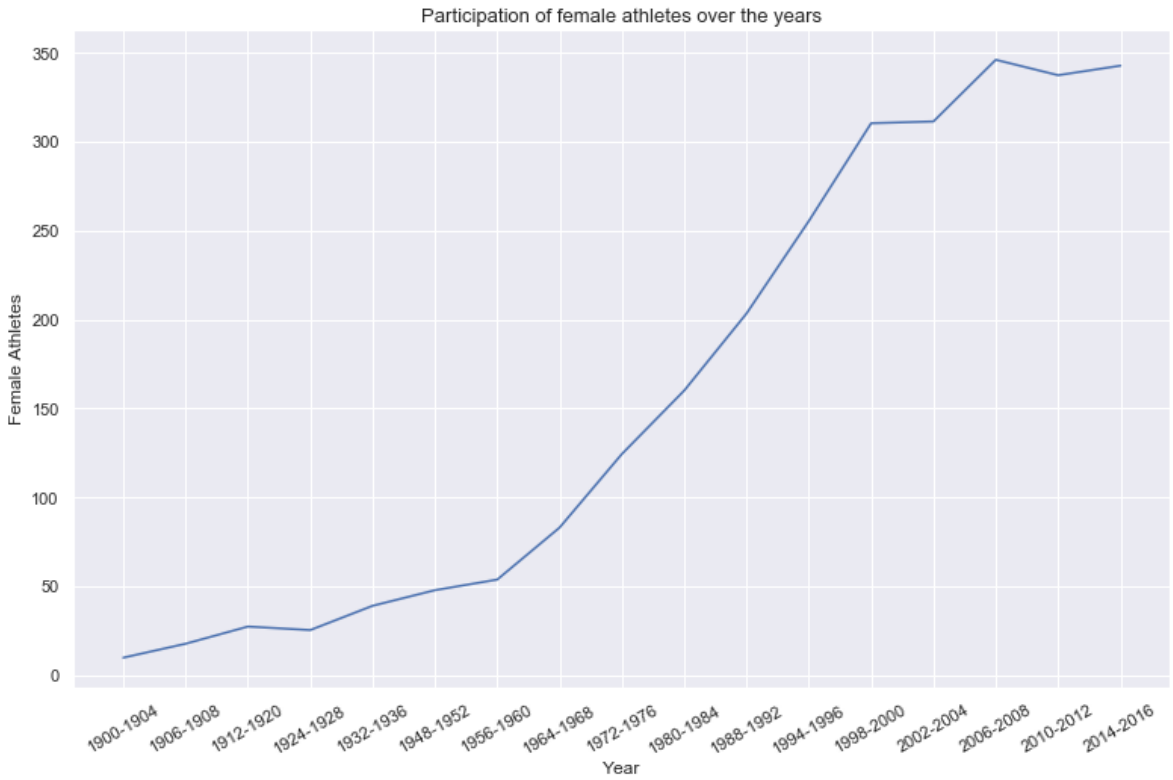
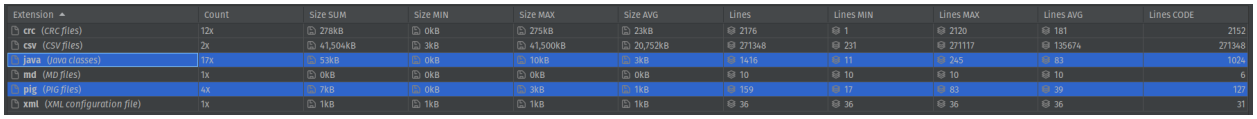


Figure 8: Question 3 Results

8 Comparison

Looking at some statistics for **Java** and **Pig** in Figure 9, we can see:

- The total lines of Java code are 1416 with comments, and 1024 without them.
- The lines of Pig code are only 159 and 127, with and without comments, respectively.



Extension	Count	Size SUM	Size MIN	Size MAX	Size AVG	Lines	Lines MIN	Lines MAX	Lines AVG	Lines CODE
crc (CRC files)	12x	278kB	0kB	275kB	23kB	2176	1	2120	181	2152
csv (CSV files)	2x	41,504kB	3kB	41,500kB	20,752kB	271348	231	271117	135674	271348
java (Java classes)	17x	53kB	0kB	10kB	3kB	1416	11	245	83	1024
md (MD files)	1x	0kB	0kB	0kB	0kB	10	10	10	10	6
pig (PIG files)	4x	7kB	0kB	3kB	1kB	159	17	83	39	127
xml (XML configuration file)	1x	1kB	1kB	1kB	1kB	36	36	36	36	31

Figure 9: Statistics for Java and Pig

Java Execution Time (rounded to 1 decimal):

- **exploratory data analysis** - 2.3 seconds
- **preprocessing** - 4.3 seconds
- **part 1** - 1.3 seconds
- **part 2** - 2.3 seconds
- **part 3** - 3.3 seconds
- **total** - 13.5 seconds

Pig Execution Time (rounded to 1 decimal):

- **preprocessing** - 14.3 seconds
- **part 1** - 11.6 seconds
- **part 2** - 22.1 seconds
- **part 3** - 14.0 seconds
- **total** - 62.0 seconds

Finally, we used the **Diffchecker** tool to compare the results between the two implementations. We confirmed that the results were identical for both approaches (e.g., Figure 10).

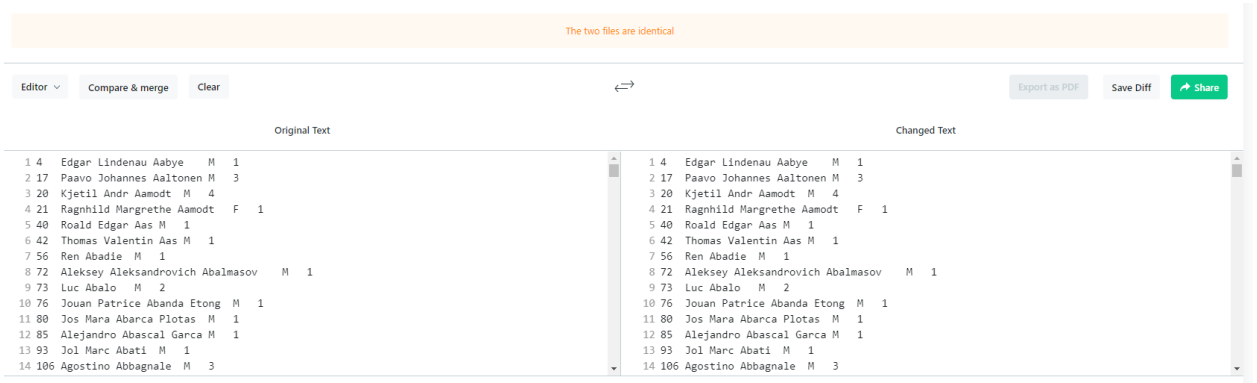


Figure 10: Comparison for Java and Pig results