Programming Data Science at Scale
Assignment 1

1 Introduction

This is the first practical assignment for the Programming Data Science at Scale course 2024/25. You need to use the Scala Collection API to solve problems you might encounter when working with collections. This section will give you administrative information. This is followed by the actual tasks, and finally, a description of how to submit your solutions.

1.1 Administrative Information

Deadline The assignment is due at **12:00 p.m. on October 18th**.

Deadline Extension The School of Informatics has a policy on coursework deadlines, which applies across all taught courses. Further information can be found here:

https://web.inf.ed.ac.uk/infweb/student-services/taught-students/information-for-students/information-for-all-students/your-studies/late-coursework-extension-requests

Questions All questions should go on Piazza

https://piazza.com/class/m0mg39x2zlt3i7

Feel free to discuss general techniques amongst each other unless you would reveal an answer. If your question/discussion reveals an answer, ask privately.

Marking The assignment is worth 25 marks in total and contributes 25% to the final course grade. Marks are given for correctness, efficiency, and the proper use of tools.

Marks are indicated in the right margin of the page using two numbers, e.g., 3+1 marks. The first number represents the marks achievable for correctness, while the second number represents the marks achievable for efficiency.

Submission The submission process for your solution is described in Section 4. We start marking at the deadline and only mark once. If you are submitting on time, you can submit as many times as you want and the last one will be marked. If you are submitting late, you may only submit once in total (which implies that you should not submit before the deadline) or run the risk that we will mark an old version and then penalise you for the last submission time.

2 Data

For the dataset you will use in this assignment, there is a `imdb-small-data.zip` file that can be accessed at:

https://amirsh.github.io/files/pdss24/cw1.html

You can copy the extracted `tsv` files to `imdb/src/main/resources/imdb/` for local testing and debugging.

2.1 Internet Movie Database (IMDB)

This assignment will be on processing the IMDB dataset – we have chosen a subset for these tasks to encourage you to think about how to structure your solutions to use multiple input data collections, and efficiently process structured text using Scala collections.

Please note that we have removed the first line of the `tsv` file, which contained the column names in the original dataset. We have done this for your convenience, as in your code, you can assume all lines are data. The four files used, including their structures, are detailed in Section 2.2.

Note that not all `.tsv` files are required for all questions. Consult the schema in Section 2.2 to ascertain which one(s) you require for the task at hand. Be aware that `skipVal` (`'\N'`) may be present where fields are denoted `Option`, meaning no data is present. **You are expected to account for this possibility and ignore those entries from your solutions**.

2.2 IMDB Schema Reference

The following table defines the columns in each of the provided files from the IMDB dataset to aid you in your solution design.

- `Option[T]` means either type `T` is present, or `skipVal` (`'\N'`) otherwise
- `List[T]` means a comma-delimited list of type `T` is present, e.g. `'dog,cat,bear'`, where `T := String`

INDEX	FIELD	TYPE	EXAMPLES/NOTES
			name.basics.tsv
0	nconst	String	nmXXXXXXX – Unique person/crew ID
1	primaryName	Option[String]	–
2	birthYear	Option[Int]	–
3	deathYear	Option[Int]	–
4	primaryProfession	Option[List[String]]	'editor,manager', 'actor', 'actress'
5	knownForTitles	Option[List[String]]	'tconst1,tconst2,tconst3'
			title.basics.tsv
0	tconst	String	ttXXXXXXX – Unique title ID
1	titleType	Option[String]	'tvMovie', 'short', 'movie', 'videoGame'
2	primaryTitle	Option[String]	–
3	originalTitle	Option[String]	–
4	isAdult	Int	–
5	startYear	Option[Int]	YYYY – Release year
6	endYear	Option[Int]	YYYY – End year, e.g. when a play ends.
7	runtimeMinutes	Option[Int]	–
8	genres	Option[List[String]]	'Documentary,Short,Sport'
			title.crew.tsv
0	tconst	String	Joins title.basics.tconst
1	directors	Option[List[String]]	'nmXXXXXX1,nmXXXXXX2' – Joins nconst
2	writers	Option[List[String]]	'nmXXXXXX1,nmXXXXXX2' – Joins nconst
			title.ratings.tsv
0	tconst	String	Joins title.basics.tconst
1	averageRating	Float	–
2	numVotes	Int	–

3 Tasks

Download `cw1-imdb-scala-src.zip` and extract it somewhere on your machine. You have to complete the missing implementations (specified by `???`) in `imdb/src/main/scala/imdb/ImdbAnalysis.scala`.

You are encouraged to look at the Scala API documentation while solving this exercise, which can be found here:

https://www.scala-lang.org/api/2.12.15/index.html

Consult the schema in Section 2.2 when designing your solutions in order to extract the correct data.

Task 1

◁ **Task**

Return a list containing an ordered key-value pair of genres based on their associated number of movies in descending order.

Note that a title can have more than one genre, thus it should be considered for all of them. **(3+1 marks)**

return type: List[(String, Int)]
genre: String
movies_count: Int

Task 2

◁ **Task**

Return a list of key-value pairs, where the key is the crew member's name (`primaryName`) and the value is the number of `knownForTitle` films they have worked on, released between 2010 and 2022 (inclusive). Only include crew members associated with at least three such films. Additionally, only consider titles where the `titleType` equals to `'movie'`.

Hint: There could be a crew member with a role different from a director or writer. **(4+2 marks)**

return type: List[(String, Int)]
crew_name: String
film_count: Int

Task 3

◁ **Task**

Return a list containing a descending ordered key-values pair of directors' names (`primaryName`) based on their ratings.

To calculate a director's rating, use the weighted average of their movies' ratings, with the `numVotes` as the weight. For a director to be included, they must have at least two movies, and each of those movies received a minimum of 10000 votes. Only consider titles for which `titleType` equals to `'movie'` in your calculations. **(4+3 marks)**

return type: List[(String, Float)]
director_name: String
weighted_average_ratings: Float

Task 4

◁ **Task**

Return a list of key-value pairs, where each key represents a decade from 1900 to 1999, and the value is a list of the top 5 movie genres for that decade, along with their calculated ratings.

Use a single digit to represent each decade, for example, 0 for movies with `startYear` from 1900 to 1909 inclusive. Ensure the list is sorted by decade in ascending order, and for each decade, sort the genres by their ratings in descending order. The rating for each genre should be calculated by determining the weighted average of all the movies in that genre, with the `numVotes` each movie has received as the weight. Lastly, only include titles that are equal to `'movie'` in the `titleType` field when performing the calculations. (**5+3 marks**)

return type: List[(Int, List[(String, Float)])]
decade: Int
genre: String
weighted_average_ratings: Float

4 Submissions

To submit your work, please do the following:

1. To test the correctness of your program, you can use the test command of sbt. We will run your submissions against a bigger test suite than we provided to you. So if you want full points, be sure to add your own tests to double check that you have caught all corner cases. We also check the performance of your implementation. Thus, make sure that you use the best possible implementation (e.g., immutable variables and groupBy instead of inefficient nested loops where possible).
2. To submit your code, please zip the entire imdb directory and name it as imdb.zip. Make sure that the data files are removed from the resource directory.
3. Upload the zip file to Learn.