Massive Data Sets
Assignment No. 2

Note 1 **This assignment is to be done individually**

Note 2 Working with other people is prohibited.

Note 2 Sharing queries or files with other people is prohibited.

A note on Academic Integrity and Plagiarism

Please review the following documents:

- Standards for Professional Behaviour, Faculty of Engineering:
  `https://www.uvic.ca/engineering/assets/docs/professional-behaviour.pdf`
- Policies Academic Integrity, UVic:
  `https://www.uvic.ca/students/academics/academic-integrity/`
- Uvic's Calendar section on Plagirism:
  `https://www.uvic.ca/calendar/undergrad/index.php#/policy/Sk_0xsM_V`

Note specifically:

> Plagiarism
> Single or multiple instances of inadequate attribution of sources should result in a failing grade for the work. A largely or fully plagiarized piece of work should result in a grade of F for the course.

Submissions will be screened for plagiarism **at the end of the term**.
You are responsible for your own submission, but you could also be responsible if somebody plagiarizes your submission.

1 Objectives

After completing this assignment, you will have experience:

- Using Scala
- Programming Functionally
- Basket analysis in big datasets using the A-priori algorithm
- 500-level students only: using a library that implements the algorithm.

2 Basket analysis

In this assignment we will implement the A-Priori algorithm for analysis of pairs of items (2 passes only). We will use the following implementation details:

1. Use hash dictionaries to keep information in memory (both items, and pairs) instead of a triangular matrix (the memory cost of our program will be proportional to the number of frequent items, specifically $O(n^2)$ where n is the number of items above the threshold).
2. Do not read all the input lines in the file at once. If you do, you will receive a 0 as your grade.
3. The input is a CSV file. Note that some files use ',' and some ';' as a separator. More on that later.
4. Your program **cannot** use any `var` bindings. In other words, if you use mutation, even once, your assignment will receive 0 as a grade, irrespectively of whether your program satisfies the requirements.

3 Your task, should you choose to accept it

3.1 Install sbt

Install sbt. I am using 1.0, but any recent version should work. Install scala 2.12 or later.

3.2 Preliminaries: how to run your program

Download from Brightspace the assignment files. There are two: source code, tests and data.

You are provided 4 source code files:

```
./02_assign/build.sbt
./02_assign/src/main/scala/Main.scala
./02_assign/src/main/scala/apriori.scala
./02_assign/src/main/scala/common.scala
```

1. The sbt file is the configuration file. Do not change it. It assumes you are using scala 2.13.3 (sbt will take care of it, and it can be different than the one you manually installed).
2. Your code will be written in `apriori.scala`. This is the only source code file you will submit (hence, do not modify any other file).

Run sbt from the directory where the file **build.sbt** is located. Once inside sbt, you can use the following commands:

- `compile` to compile your program
- `run <commandline arguments>` to run your program, passing specific command line arguments.

You can also run sbt from the command line. We will test your program from the commandline using the following (this is an example test provided, more on that later):

```
sbt --error 'set showSuccess := false' 'run ./data/online.csv "," 0.01 10'
```

This command must be run from the directory where the `build.sbt` is located. It runs the program with command line arguments (more on that later). The `--error` is to suppress sbt information. The information after run is the command line arguments to the program[1]:

```
/tmp/online.csv "," 0.01 10
```

3.3 Command line options

Your program is run with the following arguments:

```
<filename> <delimiter> <MiniumuSuportThreshold> [linesToPrint]
```

- filename. csv delimited file. One record per line. You can assume there are no duplicates in a record.
- delimiter. A one character string that indicates the separator between items in a record. Use quotes around it to avoid the shell or sbt to interpret the character.
- MiniumuSuportThreshold The minimum threshold that an pair should have (between 0 and 1)
- linesToPrint. An optional parameter. If provided, print at most this number of items. If not provided, print all the items.

For example, the command:

```
sbt --error 'set showSuccess := false' 'run ./data/online.csv "," 0.01 10'
```

uses the file `/tmp/online.csv` with delimiter , (comma) and minimum support threshold of 0.01 and requests to print at most the first 10 items (see order below).

4 Part I: A-priori

Download from brigthspace the two zip files. One is the source code, one is the tests.

Your job is to implement the 3 functions in the file apriori.scala:

4.1 doFirstPass

```
def doFirstPass(threshold: Double,
                lines: Iterator[String],
                delim:String): (Int, Int, Items) = ???
```

This function takes three parameters:

1. `threshold`: The minimum threshold (between 0 and 1),
2. `lines`: the input records (lines), and

[1] It is possible to generate a Jar file and run it as any other command line program, if we so choose

3. `delim`: the delimiter of the items in each line (a string on length 1).

It should return 3 values:

1. a counter of the number of records in the stream.
2. the minimum support threshold (i.e. the minimum number of times a frequent item appears), and
3. a Map (dictionary) of the frequent items and their corresponding frequency (number of times they appear in each record—line, you can assume that items are unique in a given record).

To compute the minimum support threshold truncate the floating point number to an integer. We will keep items with this value or larger.

4.1.1 doSecondPass

```
def doSecondPass(supportT: Int,
                 items: Items,
                 lines: Iterator[String],
                 delim:String): FreqPairs = ???
```

Takes 4 parameters:

1. `supportT`: the minimum support threshold (as an Int),
2. `items`: the Map of frequent items,
3. `lines`: the input records (lines), and
4. `delim`: the delimiter

It should return a Map that contains the frequent pairs above the minimum support threshold and their corresponding frequency.

4.2 doResults

```
def doResults(count:Int,
              items: Items,
              freqPairs: FreqPairs,
              limit: Option[Int]) = ???
```

`doResults` prints the results. It takes 4 parameters:

1. `count` is the number of records in the file (result of first pass). `items` is the frequent items (see `doFirstPass`).
2. `freqPairs` is the list of frequent pairs (result of the second pass).
3. `limit`: Print at most a given number of elements. All if `NONE`.

Use the function `printResults` to print the output (do not do it yourself, unless you are a masochist and willing to lose marks due to errors in the output). For example:

```
Running with parameters: Filename [./data/online.csv] Separator [,] Minimum relative support threshold [0.01]. Print at most 10 tuples.

25902 records, only 592 items above support threshold 259 (0.01).

SupportPair-Item                                - Freq -Support-Bought with                      - FreqPair-Confidence-     Lift
-------------------------------------------------------------------------------------------------------------------------------
   0.010887 regency tea plate green               386 0.014902 regency tea plate pink                282       0.90    60.265
   0.010501 regency tea plate roses               457 0.017643 regency tea plate pink                272       0.87    49.097
   0.012431 regency tea plate roses               457 0.017643 regency tea plate green               322       0.83    47.281
   0.010887 wooden star christmas scandinavian    515 0.019883 wooden tree christmas scandinavian    282       0.83    41.838
   0.013512 set/6 red spotty paper plates         527 0.020346 set/6 red spotty paper cups           350       0.82    40.193
   0.024863 green regency teacup and saucer      1057 0.040808 pink regency teacup and saucer        644       0.80    19.702
   0.010154 poppy's playhouse kitchen             440 0.016987 poppy's playhouse livingroom          263       0.80    46.916
   0.010115 poppy's playhouse bedroom             426 0.016447 poppy's playhouse livingroom          262       0.79    48.274
   0.013512 small dolly mix design orange bowl    528 0.020385 small marshmallows pink bowl          350       0.78    38.326
   0.023705 roses regency teacup and saucer      1120 0.043240 pink regency teacup and saucer        614       0.77    17.728
-------------------------------------------------------------------------------------------------------------------------------
Found 742 records. Printed only 10.
```

The results should be ordered according to (think order by in SQL):

1. confidence descending,
2. lift descending,
3. item frequency descending,
4. item name, ascending,
5. item name it appears with, ascending.

4.3 Tests

Download the tests from brigthspace. For each test I have provided:

1. The dataset
2. The command line
3. The expected output

Your program's output should be identical to the expected output for the same command line and input dataset.

4.4 Hints

1. The algorithm is very simple. The functions you have to write are not difficult, but it will take you some time to get familiar with the language and tools.
2. **Do not read the stream more than once per pass**. This means you need to compute its length as you are processing it.
3. In both passes, use a foldLeft. Each iteration should create a Map (and potentially other information, such as a counter of records processed) as its accumulator. Write an explicit function with explicit parameter types and return types to do each iteration of the fold (as opposed to writing it as an anonymous one). This way you can type-check and test your function. One parameter will be the accumulator (it can be a tuple) and the other the input line.

4. The initial accumulator in both cases will include an empty dictionary.
5. My functions (for each pass) are less than 20 lines long each (including empty lines).
6. As mentioned before, to output each output record use `printRecord`
7. Be careful with the output (pay attention to the last line). Your program's output should match byte-by-byte the expected output.
8. For anything not precisely specified here, see expected output and check/ask in the forum.
9. Use the type synonyms declared at the top of `common.scala`. They will help you increase the readability of your code.
10. Implement some automatic testing (e.g. using Make)
11. Running the program from inside sbt is faster than from outside.

4.5 Other information and restrictions

Violating any of the following restrictions will result in a **grade of zero**:

1. You cannot use any var binding.
2. You cannot process the input stream more than twice.
3. You cannot read the entire input stream to memory.
4. You cannot add any imports to the program.

You can share test cases but you are forbidden from sharing source code.

5 Part 2: For Graduate Students only

Use a library in Python (or any other language of your choice) that implements the A-Priori algorithm to create the same reports as in the tests provided. Your program should run stand alone with the same command line options as Part I (see 3.3). There are lots of tutorials online on how do it (e.g. using Python's mlxtend). The weight of this part will be 20% of the overall grade.

6 What to submit

Submit, via Brigthspace:

1. the file apriori.scala
2. if you are a 500-level student, a zip file called `part2.zip` with the following information:
   (a) the source code of your solution
   (b) a `readme.txt` file describing your solution and how to run it and any additional information.