2ID70, Assignment 2 – Big Data **Deadline** 26 March 2025

The solutions should be zipped and submitted via Canvas, before the deadline. It is **not sufficient** to submit the solutions to the course's server.

This assignment includes four questions:

1. Load and prepare data in Spark (**10pt**)
2. Query the data **using** SparkSQL (**30pt**)
3. Query the data **without** SparkSQL (**30pt**)
4. Identify potential epidemics (**30pt**)

Input data

You will use the following data files as inputs:

Patients.csv (patientId int, patientName character(100), address character(200), dateOfBirth character(10))

Prescriptions.csv (prescriptionId int, medicineId int, dosage character(100))

Diagnoses.csv (patientId int, doctorId int, date character(10), diagnosis character(200), prescriptionId int)

The data can be downloaded from Canvas.

Notice that there can be multiple lines with the same prescriptionId, in file prescriptions.csv, meaning that a single prescription can include more than one medicine.

Setup

The assignments in this course are set up in a way that allows all groups to test their work for performance under the same conditions. There is a server available to which each group can submit its work. This server will run your code and provide feedback about performance metrics as well as the output of your code. A document with more detailed information on how to connect to- and interact with this server in general is available in Canvas.

For this assignment, you will need to upload your application to your group's home directory on the server. As described in the general guide, the application can be in the form of a JAR file or ZIP file and must be named `app.jar` or `app.zip`, respectively. You do not need to upload any data for this assignment. The data will be made available on HDFS and you can refer to it in your source code as `'/patients.csv'`, `'/prescriptions.csv'`, `'/diagnoses.csv'` (note the forward slash, which is important for the server deployment, but will not work when you run it locally).

Constraints

Your code should be fully parallelizable and fully distributed over the Spark cluster. The code that will be executed on the master node (the coordinator) should be minimal, i.e., ONLY for distributing the computation, collecting the final answer and printing the final results.

Submission in canvas

Submitting your code to the server for execution is not enough to receive a grade. You also need to submit your solution in Canvas before the deadline. Each group should submit their work in Canvas only once. You should submit a single ZIP archive. The ZIP archive should contain only a directory called **`group-n`** (e.g., if you are group number 3, the directory should be called group-3). The directory's contents depend on the programming language you chose to work with.

Java

If you built your solution in Java, the directory `group-n` should have the following contents:

- app.jar (this should be the same JAR you uploaded to the server and thus be executable there)
- src (a directory containing any .java source files you used to build app.jar)

You are not allowed to modify the signatures (input/output) of the methods/functions in the provided template.

Python

If you built your solution in Python, the directory `group-n` should have the following contents:

- main.py (this should be the same script you uploaded to the server and thus be executable there)
- src (a directory containing any other .py source files. If none exist, this directory can be omitted)

You are not allowed to modify the signatures (input/output) of the methods/functions in the provided template.

1. Load and prepare data in Spark (10pt)

Load the data in both RDD and Dataset/Dataframe format into Spark. The RDDs should be returned by the function named Q1 (see the template code). The Datasets/Dataframes should be tied to the SparkSession to enable querying with SparkSQL from the session (Java, Python)

Also notice that the file containing the patients is partially corrupted, and contains lines that do not follow the correct format. You need to filter these lines out. A line is invalid when it contains more or fewer attributes than expected, or when the patientId is not a number.

Hint: it might make coding easier if you break (some of) the dates to separate attributes during loading. For example, a date of format yyyy-mm-dd can be broken to 3 different attributes: dd, for the day of the month, mm for the month of the year, and yyyy for the year.

To verify that the loading of the data was successful, you can print the number of patients locally. There should be a total of 1.999.828 records left after filtering. Make sure to remove this printing when uploading to the server.

Deliverable: Spark code (Java or Python) in a method/function named `Q1()`. All code for this question should be contained in this method/function.

2. Query the data using SparkSQL (30pt)

Using the return value of Q1 as an input, execute the following queries using SparkSQL:

1. Find the number of patients that were born in 1999. (**10pt**)
2. Find the date in 2024 where the number of diagnoses reached its maximum value. You can assume that there is only one such date. (**10pt**)
3. Find the date in 2024 where the prescription with the maximum number of medicines was administered. You can assume that there is only one such date. (**10pt**)

You should use data frames and data sets, <u>but the computation needs to be done using SparkSQL, rather than in pure Java or Python</u>. The results for each query should be stored in variables named q21, q22 and q23 respectively. The code should print these results as follows:

```
>> [q21: result1]
>> [q22: result2]
>> [q23: result3]
```

For example, a possible answer (for different input data) could look like this:

```
>> [q21: 17]
>> [q22: 2024-03-21]
>> [q23: 2024-02-11]
```

Deliverable: Spark code (Java or Python) in a method/function named `Q2()`. All code for this question (including the code for printing the number of records per relation) should be contained in this method/function.

3. Query the data without SparkSQL (30pt)

You are now asked to write Spark code that will address the same three sub-questions of Question 2, but now **without using SparkSQL, data frames, or datasets.** This means that you should use the basic Spark functionality (map, reduce, reduceByKey, etc.), which operate on RDDs.

Again, your code should be fully parallelizable and fully distributed over the Spark cluster. The code that will be executed on the master node (the coordinator) should be minimal, i.e., ONLY for distributing the computation, collecting the final answer and printing the final results.

The input data should be the RDD produced in question 1. Your code should print the results as follows:

```
>> [q31: result1]
>> [q32: result2]
>> [q33: result3]
```

Deliverable: Spark code (Java or Python) in a method/function named `Q3()`. All code for this question (including the code for printing the number of records per relation) should be contained in this method/function.

4. Identify potential epidemics (30pt)

The following situation is interesting for the national center of infectious diseases, as it may be an early sign of an epidemic: if for the period of 1 **calendar** month (e.g., for January), a diagnosis has been the most frequent diagnosis detected by more than 50% of the registered doctors. Using this definition, there can be at most one epidemic at the same calendar month. You need to print all pairs of <Month, Diagnosis> that satisfy the above condition.

Consider the following example.

In January 2021:

- Doctor1 detected 100 cases of flu, and 30 cases of Christmas disease[1]
- Doctor2 detected 30 cases of flu, 20 cases of Morgellons Disease, and 20 case of Christmas disease
- Doctor 3 was on holidays for most of January. They only detected one case of Grover’s disease
- Doctor 4 detected 230 cases of flu, 200 cases of cold, and 1 case of allergy

In February 2021:

- Doctor1 detected 10 cases of flu, and 30 cases of Christmas disease
- Doctor2 detected 10 cases of flu, 20 cases of Morgellons Disease, and 1 case of Christmas disease
- Doctor 3 was still on holidays for most of February. They only detected one case of flu
- Doctor 4 detected 450 cases of cold, 100 cases of flu, and 1 case of itchy feet

Assume that these statistics represent **all** medical examinations in your data. With these statistics, flu was the most frequent diagnosis of 3 out of the 4 doctors in January, i.e., greater than 50% of the doctors. In February, there was no such frequent diagnosis.

So the answer should include only the pair:

2021-01,flu

And it should be printed as follows:

>> [q4: 2021-01,flu]

If there are more than one pairs in the answer, then all **should be printed one after the other, using character ‘;’ as a delimiter, and sorted chronologically on the calendar month.** For example, a possible answer is:

>> [q4: 2021-01,flu;2021-03,cold]

[1] According to ChatGPT, this and the other names you see below are real diseases/medical conditions!

Deliverable: Spark code (Java or Python) in a method/function named `Q4()`. All code for this question (including the code for printing the answer) should be contained in this method/function.

Frequently asked questions

Plagiarism: Plagiarism will not be tolerated. All parts participating in plagiarism will be reported and punished according to the university rules.

How will my submission be graded? The final submissions will be downloaded from Canvas. They will be executed and their results will be compared to the expected results. The tests will be automatic. This is why it is very important that you strictly follow the provided instructions.

Do I need to adhere strictly to the provided instructions (filenames, structures, etc)? Yes, for the testing suite to evaluate your solutions correctly.

Can I verify that my answer is correct, before submission? You can verify your submission on the small data set for which you will receive the expected output. Notice however that obtaining the same output on the small data set does not completely guarantee the correctness of a submission in general.

Further questions: For clarifications and questions about the assignment (or useful pointers), please use Canvas discussions so that your fellow students can also see the answers. For questions that cannot be asked publicly (e.g., the question reveals part of the answer), you can ask us privately during the instruction hours.

Use of ChatGPT and other assistants: As discussed during the first week, and according to the course's policy. You are allowed to use ChatGPT and similar tools, in the same way you would use a search engine, e.g., to find the reason for an exception. These tools can be used to facilitate your learning process, not to answer your homework. As a sanity check, you can think of the following question: if I ask this question to my teacher, or to a teaching assistant, will they answer? The teaching assistant and your teacher will never give you an answer that 'solves' the exercise, as this would defeat the learning goals of the exercise. They can help you by providing a different perspective, or by explaining to you why your code is very slow, or why it fails. You can use these tools in the same way. Another way to use these tools is to see alternative solutions, **after** you solve and submit your exercise (Keep in mind that ChatGPT often errs, but in a very convincing way. Therefore, the answers you get might as well be wrong.). In all cases, you are expected to fully understand the solution, to the point that you will be able to apply it in a different context (and in the exam), without the help of such tools.

An example: Your teacher and teaching assistants would never answer a question: "Write me code that does XYZ", as this answer will not help you learn – it will only help you evade the necessary hands-on practice. They would, however, look into the following question, and try to help you: "I am running Spark on my computer, but even the word-count demo code that I downloaded from the Spark webpage does not seem to use more than one core. What might be the problem?"