## Part one of the project for the Big Data Architecture course (TDT4305 spring 2021)

**Submission deadline: 11.30pm, March 2nd, 2021**

## Introduction and context

This document describes the students' tasks for the TDT4305 as the first part of the course's project. There are several main tasks that each consists of several subtasks. And it is expected that students will complete this part of the project (and its tasks) either in teams of two or individually.

The tasks and their related subtasks are shown in the table below (obviously).

**Task index 1**

| Index | Description |
|---|---|
| 1 | Load the posts.csv.gz into an RDD |
| 2 | Load the comments.csv.gz into an RDD |
| 3 | Load the users.csv.gz into an RDD |
| 4 | Load the badges.csv.gz into an RDD |
| 5 | Print the number of rows for each of four RDDs |

**Task index 2**

| Index | Description |
|---|---|
| 1 | Find the average length of the questions, answers, and comments in character |
| 2 | Find the dates when the first and the last questions were asked. Also, find the display name of users who posted those questions |
| 3 | Find the ids of users who wrote the greatest number of answers and questions. Ignore the user with OwnerUserId equal to -1 |
| 4 | Calculate the number of users who received less than three badges |
| 5 | Calculate the Pearson correlation coefficient (or Pearson's r) between the number of upvotes and downvotes cast by a user. The Pearson's r formula is: $$r_{XY} = \frac{\sum_{i=1}^{n}\left(X_i - \bar{x}\right)\left(Y_i - \bar{Y}\right)}{\sqrt{\sum_{i=1}^{n}\left(X_i - \bar{x}\right)^2}\sqrt{\sum_{i=1}^{n}\left(Y_i - \bar{Y}\right)^2}}$$ where X and Y are two numerical vectors. Do not use any Spark libraries like MLib to calculate $r_{XY}$. Instead, implement it yourself |
| 6 | Calculate the entropy of id of users (that is UserId column from comments data) who wrote one or more comments. The entropy of a (random) variable usually is calculated via the following formula: $$H(X) = -\sum_{i=1}^{n} P(x_i)\log_2 P(x_i)$$ where $P(x_i)$ is the number of occurrences of $x_i$ (as the id of a user) divided by the total number of records (or rows) in UserId column. Do not use any Spark libraries. Instead, implement H($X$) yourself |

**Task index 3**

| Index | Description |
|---|---|
| 1 | Create a graph of posts and comments. Nodes are users, and there is an edge from node $i$ to node $j$ if $i$ wrote a comment for $j$'s post. Each edge has a weight $w_{ij}$ that is the number of times $i$ has commented a post by $j$ |
| 2 | Convert the result of the previous step into a Spark DataFrame (DF) and answer the following subtasks using DataFrame API, namely using Spark SQL |
| 3 | Find the user ids of top 10 users who wrote the most comments |
| 4 | Find the display names of top 10 users who their posts received the greatest number of comments. To do so, you can load users information (or table) into a DF and join the DF from previous subtasks (that the DF containing the graph of posts and comments) with it to produce the results |

| | | | |
|---|---|---|---|
| | | 5 | Save the DF containing the information for the graph of posts and comments (from subtask 2) into a persistence format (like CSV) on your filesystem so that later could be loaded back into a Spark application's workspace |

## Additional information

Use Java, Python3, or Scala to develop and implement your solutions. Your team representative should submit a zip file before the deadline that includes these items:

1. A PDF file that briefly describes the logic behind your answers/solutions for each subtask and how to run your code
2. The code for your solution
3. And names of the people in the team

Your solution should be a runnable program or script that as its input argument accepts a directory path. Your solution must look for to read four CSV (or GZ) files in the directory path it received. And it should write the results into the STDOUT (that usually is the screen of your monitor). Suppose your solution needs to write something to the filesystem. In that case, it must do it within the directory where it received as the input argument. For example, imagine that you made a solution for task 2 (and its subtasks) in Python3 and your solution is a script named "task2.py". And your solution reads the data form the directory "/home/users/data". Your solution should normally run by executing the command shown below and spit out the result to the screen or the terminal. The command would look like this:

```
spark-submit main.py --input_path /home/users/data
```

If you are using Java, please pack everything inside a runnable Jar file. The rest would be similar to the example command mentioned above. Please avoid using any special kind of dependencies that complicates the ability to run your code. Normally, the code of your solutions should execute using standard Spark API.

Please use Spark 2.4.x or Spark 2.2.x to develop and run your solutions. Also, use Spark RDD API for implementing your solutions. The exception is that in the subtasks [3 – (2 to 5)] you should use a mixture of RDD API and DataFrame API to make your solutions.