# Introduction

The solution has been written in *Python 3.7.9*, using *pyspark 3.0.1*. The project has been run inside a *conda virtual environment*, and the *requirements.txt* file can be used to set up a new environment.

The tasks have mostly been solved by creating a function for each task. This was done so that one could run just one file and get all answers. However, running all the tasks takes some time, so each tasks function can easily be commented out to skip execution.

The script can be ran using the following command if the data is located in a folder called “data” in root of the project.

$ python3 main.py

If the data folder is located somewhere else, the input\_path flag can be set. The script can then be run used the following command.

$ python3 main.py --input\_path /path/to/data/folder

The tasks are solved by doing operations of the RDDs. In retrospect, it’s apparent that doing a map operation over all RDDs upon import, to do a string to list conversion (*split(char))* would be beneficial to do only once.

# Task walkthrough

The task is mostly solved using *map*, *filter* and simple functions on the RDDs. The task descriptions are quite sort, as writing extensive descriptions of each task would be very time consuming. Have a look at the code for reference.

## Task 1.1-1.4

Loading files in to RDDs using *spark contexts* *textFile.*

## Task 1.5

Printing number of rows using count function.

## Task 2.1

Finding length of each text bodies using *map*. Finding average length using *mean*.

## Task 2.2

Filtering out only questions. Converting so only *userIds* and parsed dates are present. Finding max and min using *max* and *min* functions. Finding usernames using a RDD with users and mapping with max and min pair IF *userIds* match. Joining the user RDD and question RDD would be a more elegant solution.

## Task 2.3

Filtering out non-questions and non-answers respectively. Mapping, and reducing to get number of questions and answers for each user. Finding the max using *max* function with custom key.

## Task 2.4

Finding the userIds of each post. Mapping, and reducing to get the number of badges for each user. Filtering out users with fewer than three badges.

## Task 2.5

Mapping an RDD with each users upvotes and downvotes. Mapping out only upvotes and only downvotes to easily calculate the averages using *mean* function on each. Using *map* to find the differences for the dividend, which is then summed. Finding the squared diffs for upvotes and downvotes, which is individually summed together, then multiplicated together. The dividend is then divided by the divisor.

## Task 2.6

Mapping and reducing to find, first number of comments, then number of comments per user. Then using map each term for the summation is found, using *map* and the operations from the formula provided. Then all terms are summed together and negated.

## Task 3.1

Using *map* to find all users with display name, all posts with corresponding user id and post id, and comments with corresponding post id and user id. Joining posts and comments together (using *join* function). Mapping and reducing to find the weight of each edge (comments on posts).

## Task 3.2

Converting RDDs to DFs using *sqlContexts* *createDataFrame*, and specifying the column names.

## Task 3.3

Registering the DF of the edges as a temporarily table (*registerTempTable*) to be able to do normal SQL queries. Then doing a SQL query where the table is first grouped by *src*, then weights are summed together to form one line for each *src* id, then this is sorted and top 10 is extracted.

## Task 3.4

Doing a SQL query where the table is first grouped by *dst*, then weights are summed together to form one line for each *dst* id, then this is sorted and top 10 is extracted.

## Task 3.5

Using the built-in write function for DFs and *databricks* to create/write the csv files.