How to run the code

To set up a new python project it is recommended to set up a virtual python environment. This can be done by the following command on Windows 10: py -m venv bigdata

Then use this command to go into the venv: bigdata/Scripts/activate

The only truly required setup is to install the python dependencies. This can be done with the following command: pip install -r requirements.txt

To finally run the code, use this command: python main.py --input_path path\to\data

Logic behind solutions

Task 1:

In task 1 we load the csv files as text files. The first row of each file are headers, describing the data beneath. We only want to count the rows with actual data in it, so we remove the first row of each csv file with the filter function. Then we print the number of rows using the count function.

Task 2:

2.1: Firstly we need to parse each row so that we can acces the induvidual data-fields. We therefore map each row to be split on the tab character. Then we make lists of questions and answers by filtering posts by the PostTypeId. We make the comment lists the same way as posts. Since the questins, answers and comments are base64 encoded, we map the listst to be the decoded version, and then the average length of answers and questions was found by finding the length of the body and text.

2.2: First we map questions to a tuple with owner id and the creation date (as a python date object). Then we use the max and min functions to find the first and last created questions. We then parse the users the same way we did with the other data in task 2.1. We filter out every user that does not have the same id as the previously found id, and then we print out these id-s with the corresponding names.

2.3: We obtain the user id's from the questions and answers – RDDs, by first filtering away the userids that are NULL (which corresponds to the userid of "-1"), then we make a tuple with the user id and 1, which we use reducebykey(add) to sum every time a user id has an answer or a question, then we use the max function to get the userid with most questions and answers.

2.4: First we make an RDD of the badges file, parsing it as previously explained. From there we once again map a tuple of userid and 1, and use the reducebykey(add) to find out how many badges each user has. From there we filter all userids with less than 3 badges and use the count functions to produce the output.

2.5: Here we just followed the formula given in the task:

$$r_{XY} = \frac{\sum_{i=1}^{n}(X_i - \bar{x})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

We map the users to the number of upvotes for each user in the variable X and the number of downvotes in the variable Y. We use the mean function to find $\bar{Y}$. We zip the RDDs so that each row is on the format (x, y).

The numerator is calculated by mapping the zipped X and Y to the formula shown inside the sum sign, and then we sum the results. The denominator is calculated in two parts, one for X and one for Y. At the end everything is calculated together.

2.6: Here we just followed the formula given in the task:

$$H(X) = -\sum_{i=1}^{n} P(x_i)\log_2 P(x_i)$$

We map the comments to the format (userid, 1), and use reducebykey (add) to find out how many comments each user has. We store the total number of comments in a variable and we countinue to map the userd's with wit then number of comments, so that it becomes P(x). We then map thes P to the given formula.

Task 3:

3.1: To represent the graph, we only need to represent the edges, since the nodes noes not hold any extra needed information. First we map comments and posts to be on the form (post id, id of the user who commented) and (post id, id of the user who posted). Then we join these two RDDs so that the resulting format is (post id, (id of the user who commented, id of the user who posted). We do not need the post id anymore, but want to know how many times there is a connection from the user who commented to the user who posted. We therefore map the joined RDD to the format ((id of the user who commented, id of the user who posted), 1) and take reducebykey(add) to get the RDD on the format ((id of the user who commented, id of the user who posted), edge weight). At last we want the RDD to have three separate fields, so we map the result to the format (id of the user who commented, id of the user who posted, edge weight).

3.2: We create a sqlContext and use its function "createDataFrame" to create a dataframe of the RDD graph, with named columns. The "show" function displays some of the data to ensure everything is on the right format.

3.3: We group the data by the commenter ID (the id of the users who commented), sum the weights and sort the result by this sum. Using the show function, we display only the top 10 users who wrote the most comments.

3.4: We used the same technique as in subtask 3.3, bot now we group the data by poster id (the id of the users who posted), and use the “take” function to get a list of top ten users who posted the most. This is then joined with a dataframe version of the user RDD and displayed with the “show” function.

3.5: The original dataframe is stored on the location specified by the script argument –input_path.