Mini-project 1:

Hadoop ecosystem

| Asmaa IsmailID: 201600780Anhar Hassan ID: 201601171 |  |
| --- | --- |

# 

# DATA ANALYSIS:

We started out by visually inspecting the data fields and format by displaying a sample of the dataset file. We also referred to the dataset documentation provided in this [source](https://github.com/reddit-archive/reddit/wiki/JSON) which documents the objects retrieved by the reddit API. We focus mainly on the ***‘comment’*** structure and its fields since our dataset is a set of user comments retrieved from reddit. Every line in the dataset represents a comment in JSON format and carries the following fields:

( we interpreted these fields by referring to the linked documentation- and using a common sense when information is wasn’t available)

| Author | The username of the comment writer. |
| --- | --- |
| Score | The net score (upvotes-downvotes) |
| Archived | Boolean for whether or not the comment was archived |
| ID | The comment’s unique id. |
| Distinguished | to allow determining whether they have been distinguished by moderators/admins. null = not distinguished. moderator = the green [M]. admin = the red [A]. special = various other special distinguishes. |
| Body | Text body of the comment. |
| created\_utc | the time of creation of the comment in UTC epoch-second format. Note that neither of these ever have a non-zero fraction |
| Downs | The number of downvotes. |
| name | its unique ID which forms a compact encoding of a globally unique ID on reddit. |
| link\_id | ID of the link this comment is in |
| score\_hidden | Boolean. Whether the comment score is hidden. |
| edited | Boolean. Whether the comment was edited. |
| subreddit\_id | Unique id for the parent subreddit. |
| controversiality | an integer of either 0 or 1, denoting whether the comment is considered controversial. |
| author\_flair\_text | the text of the author's flair. subreddit specific |
| guilded | the number of times this comment received reddit gold |
| author\_flair\_css\_class | the CSS class of the author's flair. subreddit specific |
| retrieved\_on | the time of posting in UTC epoch-second format. |
| ups | The number of upvotes. |
| parent\_id | The id of the POST that this comment belongs to. |
| subreddit | The name of the subreddit this comment belongs to. |

## **Analysis observations:**

1- We noticed that the downvote count for all comments in the dataset was zero.

2- We noticed that the upvote counts had negative numbers and were equal to the scores in all fields, we suspect that downvotes were omitted after being subtracted from the upvotes and the net score only remained.

3- We found that the controversiality scoring for all comments was zero.

4- There was no ‘replies’ field in any of the data. We expected to find a reply field given the information in the [source](https://github.com/reddit-archive/reddit/wiki/JSON#comment-implements-votable--created) and the second requirement in the project.

# Challenges and optimization:

## Challenges:

1- Runtime and processing the entire data locally:

We had difficulty downloading, extracting the entire dataset, as well as processing all of it on our local computers without causing a crash or waiting on a very long runtime. We solved the this issue by:

* Using Colab instead of our local machines which significantly sped up the download and processing of the data.
* Running the mapreduce jobs on the .bz2 compressed file directly instead of the extracted 32 Gbs which saved a lot of time.
* We did a data cleaning job before proceeding with the analysis which reduced the data size and consequently the runtime.

2- Storing the inputs and outputs in the localhost’s space of colab caused them to disappear when colab is closed and consumed the entire available desk space on the host:

* We solved this by integrating with google drive and setting up the mapreduce operations to take their inputs and dump their output on drive.

## 

## 

## 

## Optimizations:

1- As mentioned above, we ran a cleaning job prior to the data processing in which we:

* Dumped the unnecessary fields in the Json, which weren't used in the analysis but added so many gigabytes to the data size.
* Summarize the comment body field (which could be several lines of text long) into a single descriptive word denoting the topic of the comment; which is all we will need for the analysis requirements.

These cleaning steps reduced the data file size from 32 Gbs to a little under 7.5 Gbs, and by working a .bz2 compressed version of this, we could run a single mapreduce job in around 10~20 mins.

2- Using NLTK to carry out the text processing and the topic modelling instead of doing it by hand saves a lot of time and effort.

# Code Pipeline:

## STEP1 : THE DATA CLEANING JOB:

This mapreduce job runs on the entire dataset. The goal is to discard the data fields we don’t need and keep only the fields we need for our analysis which are: t***he subreddit, author, topic (retrieved from body), controversiality, ups, downs, and score.*** Here, we also carry out a simple 1-gram topic modelling function to summarize the comment body into a single descriptive word.

**Input**: The entire raw dataset.

**Output**: a clean version of the dataset with the same json format, but with fewer key values.

**The mapper:** simply reads the json line by line, and emits the key value pairs of interest: ['subreddit', 'author','body', 'score', 'downs', 'ups', 'controversiality','replies'].

**The reducer:** uses NLTK to do the following cleaning steps on the body text :

1- remove URLS and emojis

2- tokenize and convert to lowercase

3- remove stop words

4 - remove non-english words

5- remove special characters / punctuation

6- lemmatize

After that, the topic is found by counting the remaining tokens and retrieving the most common one, and the sentiment analysis is done using NLTK.

## STEP2 : REQUIREMENT 1:

Uses 3 mapreduce jobs chained one after the other. According to our understanding of the requirement, we need to first find the top subreddits- which we consider to be the most frequent subreddits in the dataset- then we find the most frequent topic in each of these top subreddits, then we find the most common topic for every username that commented in these top subreddits.

### Job 1: counting the frequency of each subreddit

**Input**: The entire clean dataset.

**Output**: 3 entries separated by “|~|” : the name of the subreddit, a dictionary of all the usernames that commented in this subreddit as keys, and a list of the topics these usernames commented (within the subreddit) as a value. We used the |~| pattern as a separator because it's a very unusual pattern to avoid it being present in any subreddit name or in any text body by chance.

#### Ex: **some\_subreddit|~|{‘user1’:[topic1, topic2], ‘user2’:[topic1,topic3]}**

**The mapper:** extracts the subreddit, author, and topic fields from each json line, then emits the: subreddit, {author: topic} as dict, a value of 1-all separated by |~|.

**The reducer:** takes the subreddit as key with two values: a value of 1, which is accumulated to yield the total count of the subreddit, and a dict value {author:topic} which is also accumulated in a defaultdict structure where for every subreddit we add the current author as a key and append the topic into its list of values.

### Job2: sorting the lines by subreddit frequency:

The goal here is to sort the previous output by subreddit count.

**Input**: the output of job 1.

**Output**: the same output as in job 1 except it's ordered by the subreddits count, and the subreddit count is now the key.

#### ex : count|~|subreddit|~|{‘author1’:[topic,topic2],....}

**Mapper**: passes the subreddit counts as keys (\*-1 for descending order), and the subreddit, and author-topic dictionary as values.

Reducer: outputs the same input it receives.

### Job3: most popular topic calculation (per subreddit and per topic)

Here we filter out the top N subreddits (we set N=500 for now), then we count the topic occurrences per subreddit and per author and select the highest.

**Input**: output of job2.

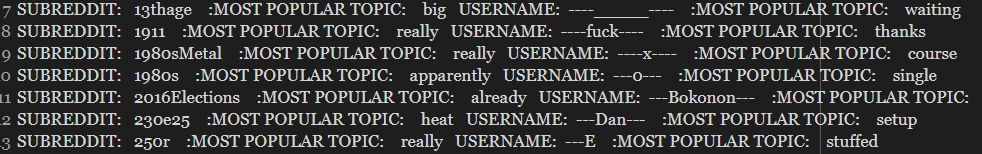
**Output**: we treat the subreddit and author as independent but print them in one line in the format:

#### SUBREDDIT: subreddit\_name MOST COMMON TOPIC: topic AUTHOR: username MOST COMMON TOPIC: topic

**Mapper:** separates the subreddit and the list of its topics from the author and its topic list. And sends each of them as key and their topic list as value separately. The topic list of the subreddit is simply the result of chaining all the topic lists from in the topic-author dictionary. The author’s topic list is its value entry in the topic-author dictionary. We distinguish the author key value pair from the subreddit key value pair by appending a ‘/r’ to the subreddit name, and ‘/a’ in the author name.

**Reducer:** first checks whether the incoming key is a username or a subreddit by checking the appended ‘/r’ or ‘/a’. If the key is a subreddit, the most common topic is just the most frequent element in its topic list. If the key is an author, we have to accumulate value over all subreddits in a defaultdict before finding the most common topic from the accumulated list.

SAMPLE OF THE OUTPUT:

****

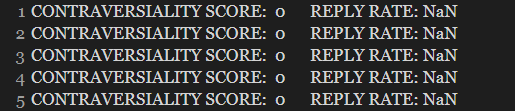
## 

## STEP3 : REQUIREMENT 2:

Rate of replies compared to controversiality of comment/post. We approached this requirement by doing two alternative solutions: one for the rate of replies ***to the comment*** given the controversiality score. Referring to the [source](https://github.com/reddit-archive/reddit/wiki/JSON#comment-implements-votable--created), we expected to have the replies field in the comment json (which is a list of all the replies to this comment). We wrote a map reduce job that counts the elements in the replies in the replies entry , and compares it to the controversiality score for this comment.

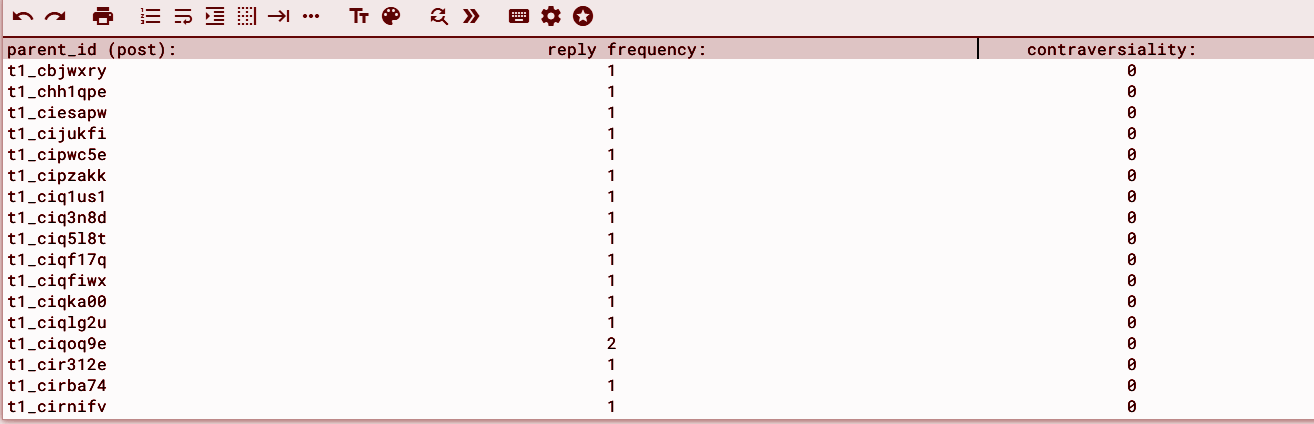
HOWEVER: we noted that the replies field was missing from the entire dataset and the controversiality score for all entries was zero.

SAMPLE OUTPUT FOR APPROACH 1:



Approach 2: we tried to find the rate of comments to each post versus its controversiality. We calculated the comment frequency per post by calculating the number of occurrences of each parent\_id field by a wordcount algorithm. We also accumulated the controversiality scores across all the entries for each parent id.

SAMPLE OUTPUT:



We cannot infer any correlation because the controversiality score is always zero.

## STEP4 : REQUIREMENT 3:

This requirement is implemented based on summing the upvotes “since

we discovered that it is equal to the score in the whole dataset”, we had

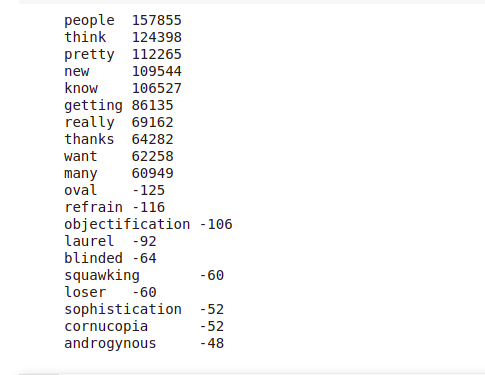
to manually ignore certain spammy/ meaningless topics like “get, someone

, NaN,,,etc”, we already expected them due to the poor topic modelling technique.

Input : The entire clean dataset.

Output: a pair of (topic, and its associated vote sum).

Output Screenshot:



Mapper: Prints the topic, and its associated vote from the dataset.

Reducer: We put the data in a collection, sum over each topic, and return

the highest 10 score, and the lowest 10 scores, in the reduced we, as well, exclude

the undesirable topics.

## STEP5 :THE CREATIVE REQUIREMENT :

We implemented two reqs:

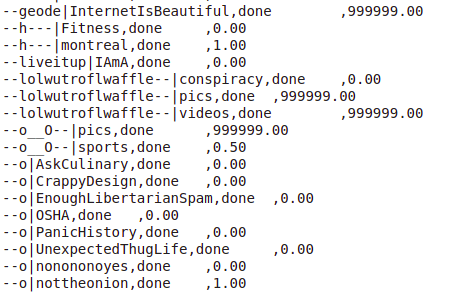
1- Ration of negative / positive attitude of comments per subreddits/users:

This requirement consists of two jobs, and since it was quite unclear, hence, we implemented two job1’s:

* First we did sentiment analysis during the cleaning process, hence we have an attribute in the dataset for the sentiment.
* The first job would return a key and a pair of numbers for the positive counts and the negative counts respectively.

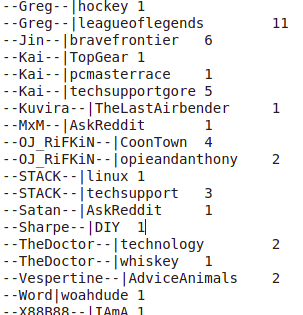
The unclear part was whether the user and the subreddits should be joined together, or separate, hence we implemented the two: one in the folder named job1\_joint and another job1\_seperate.

The second job divides the negative counts over the positive counts, and assigns users with 0 positive counts to a large number “99999”.



2- User recommendation system: was designed based on counting the numbers of the subreddits over which a user had commented, then returning the subreddit with the largest count. Further optimization could have been done, as we were considering taking the sentiment ratio into account, but after further scrutinization, it is noted that the chronological data of the user’s comments would have helped drastically, so ended up with that.

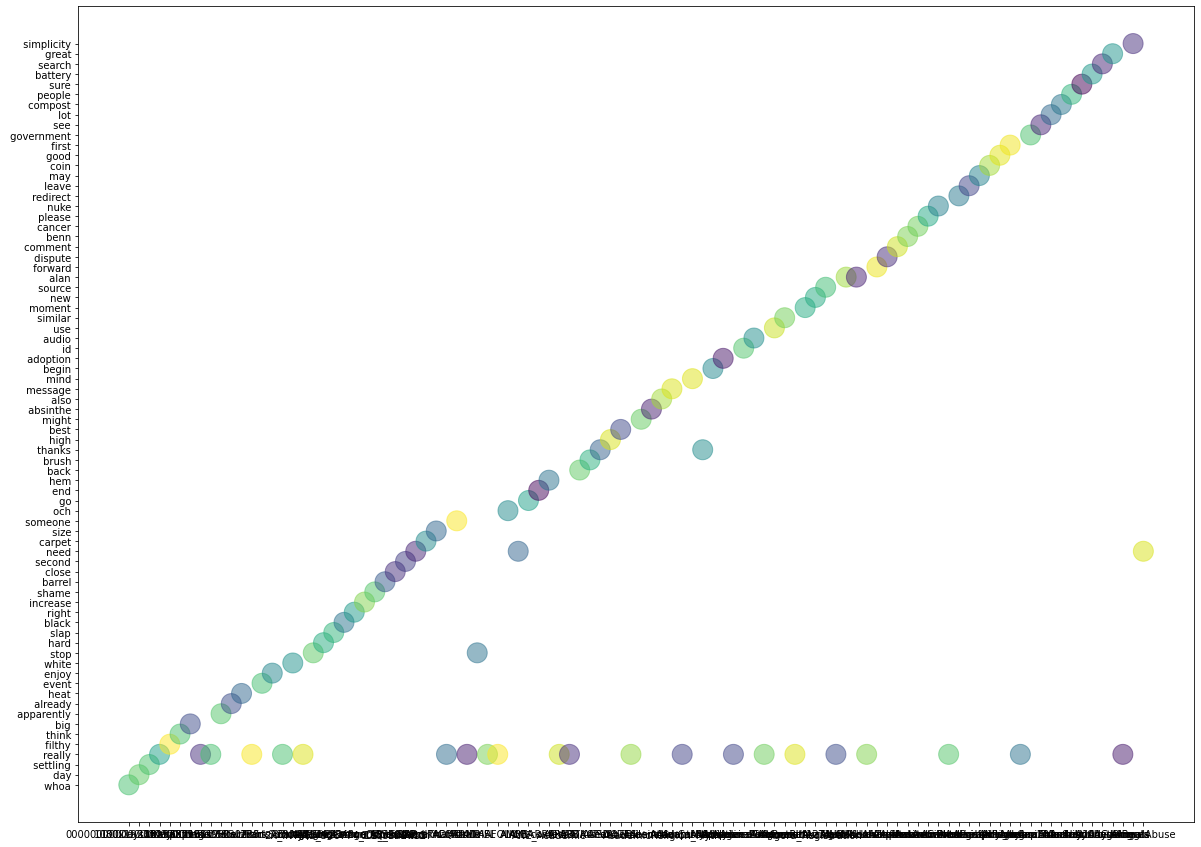
A sample of the output:



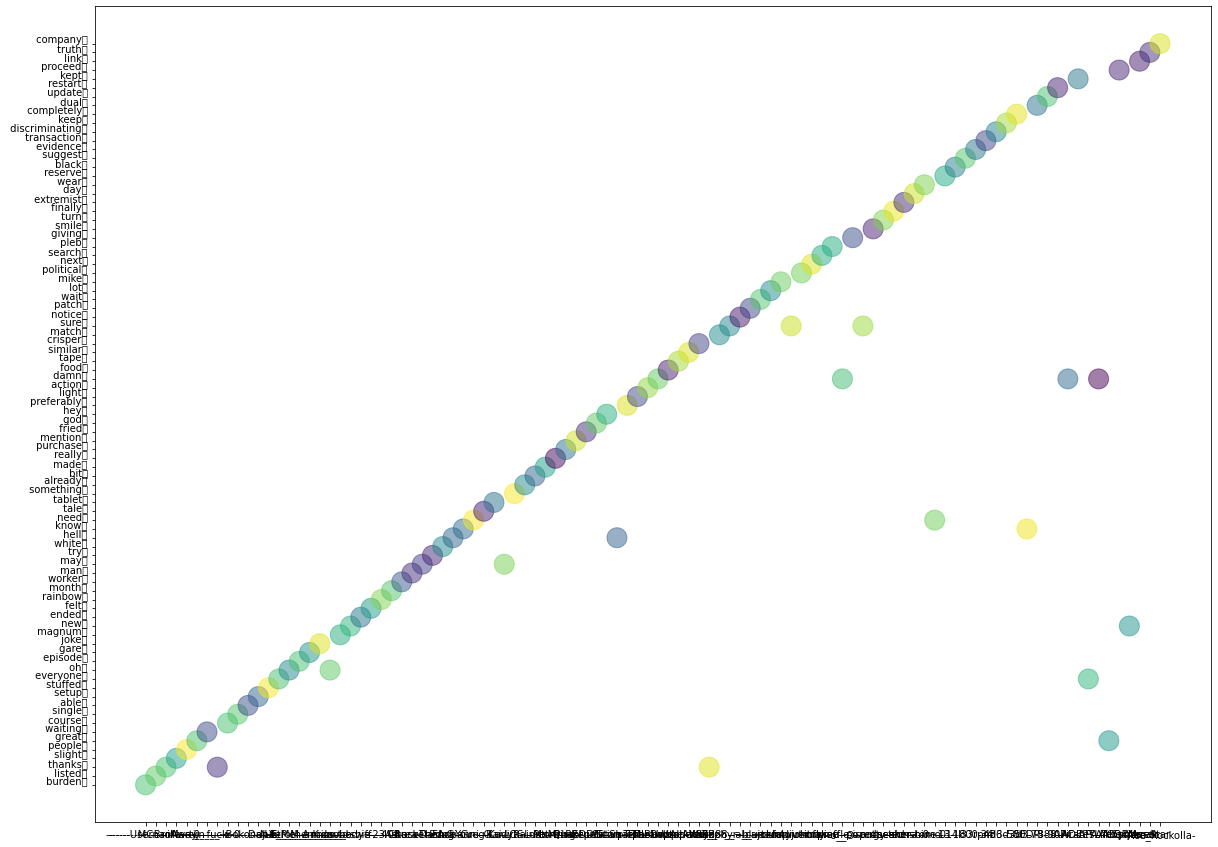
# IIII. VISUALIZATIONS:

## REQUIREMENT 1 VISUALIZATIONS:

In this part we had two different plots one for Topic against user:



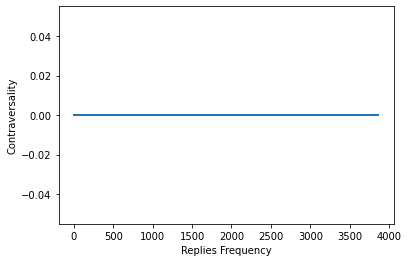
And another for the Topic against the subreddit:



These two show a linear trend, which means it’s rare to find multiple topics associated with the same subreddit/user and vise versa, we justify that by the fact that we used 1-gram for topic modelling which is naive”and we noticed examples like keep and kept, where it considered them two different topic, further optimization for the topic modelling technique would have solved this.”, also few outliers can be seen within this sample “of 100 points”

## REQUIREMENT 2 VISUALIZATIONS:

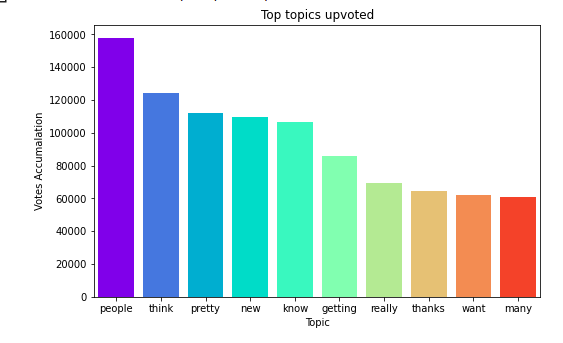
In Both approaches to this problem, we ended with this graph:



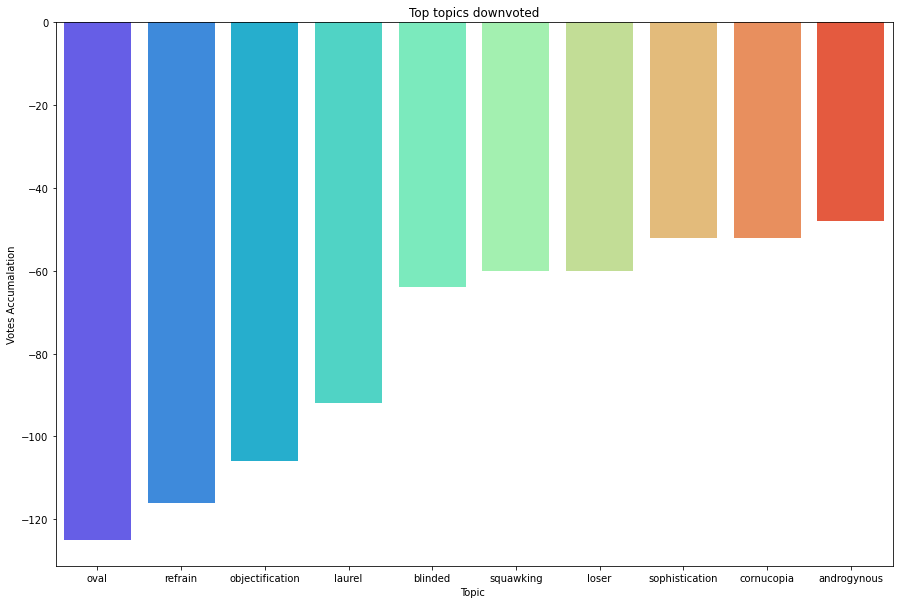
As already the contraversality is equal to 0, and the rate of replies is not provided in the data, we think this is the unreasonable requirement.

## REQUIREMENT 3 VISUALIZATIONS:

We have plotted two graphs: one for the top 10 with the highest votes:



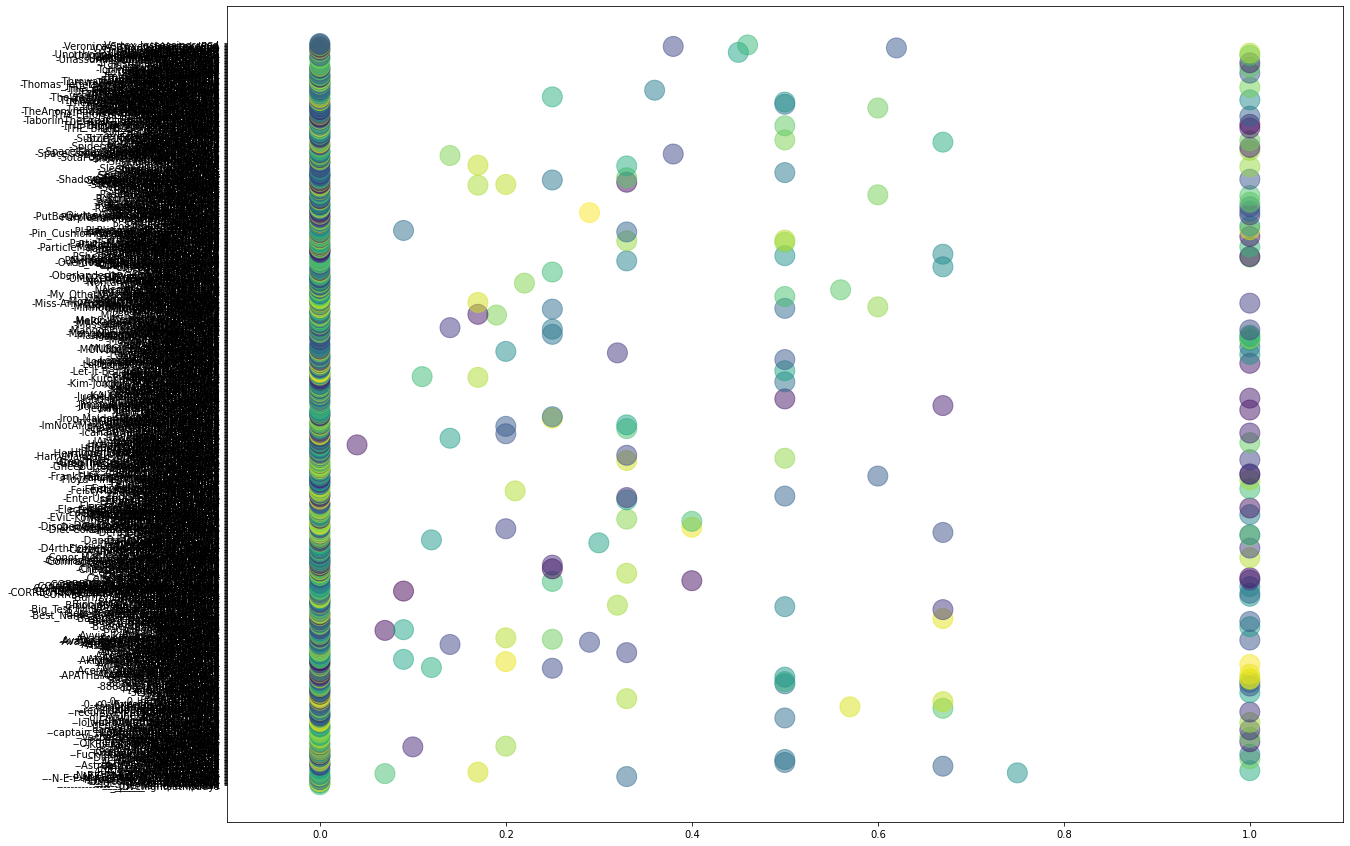
And another for the lowest 10 downvoted top:



They look reasonable, I had to manually remove topics preceding them as they were naive “ like NaN, get..etc” due to the usage of 1-gram for topic modelling.

## EXTRA REQUIREMENT VISUALIZATIONS:

We plotted the author|subreditted vs. the percentage of the sentiments of negative over positive values:



The graph looks reasonable, I had to truncate it though, as in case a user has 0 positive sentiments, it will be assigned a value of 99999, hence, I focused on the area of interest “0~1” in plotting.

[Here](https://drive.google.com/drive/u/2/folders/1gVD6jQDj0Dv4NZuFHJlQ_rSqWwJoAYxm) is the link to all the codes and scripts.