Term Report
Databases

AskReddit Dataset Analysis using SQLite and Python

April 11, 2021

M2 Statistics & Econometrics,
Toulouse School of Economics

Sherman **ALINE**
Ankita **BHATTACHARYA**

Toulouse School of Economics

Contents

Introduction

This project takes us through the routine of modelling and storing a subset of a large dataset, which in our case is the Ask Reddit dataset. Our reduced dataset contains 4 million rows and this document shall outline the procedure we employed to accomplish the said objectives and the challenges encountered in the process. The reason behind using a subset rather than the original dataset is two-fold : reducing redundancies in the latter and bring it down to a size of around 3 GB including indices, that our computers could manage.

Subsequently, we conducted an *exploratory* analysis of our data that contributed to the visualization of this large dataset. Not only have we presented simple descriptive statistics, we have also tried to capture the most representative comments and users by different variables such as the average score or karma, the gilds or reddit gold they received, and controversiality to name a few. We have tried to conduct a comprehensive analysis into the user and comment characteristics of the database and hope the reader enjoys read

Presenting the database

The reduced dataset contains 20 columns that have been modelled in the relational schema given below. A single table is not efficient compared to smaller, interconnected tables for two reasons : it is inefficient in terms of database functionality and does not help us visualize the relationships that exist within the table.

As a result, we were provided the relational schema stated below that groups the 20 columns under 11 tables. These tables Our database will thus contain these 11 tables. Primary keys are

underlined and foreign keys are followed by the # sign as per the convention seen throughout the course.

- **AUTHOR**(author)
- **DISTINGUISHED**(distinguished)
- **CONTROVERSY**(controversiality)
- **REMOVAL**(removal_reason)
- **SCORE**(id#, score, ups, downs, score_hidden, gilded)
- **PARENT**(parent_id, link_id)
- **SUBREDDIT**(subreddit_id, subreddit)
- **COMMENT**(id, created_utc, name, body, edited, author_flair_css_class, author_flair_text, author#, controversiality#, subreddit_id#)
- **IS_DITINGUISHED**(id#, distinguished#)
- **REMOVED**(id#, removal_reason#)
- **DEPENDS**(id#, parent_id#)

Here we see that the key `id` is a primary and foreign key for several tables. This is a common convention in modelling relational databases. We shall now proceed to the first out of the two tasks.

Section I

1. Creating and populating the database

At the outset, we used the entity/relationship diagram to understand how the relations flow across tables - this is vital to both parts of this twofold task, while creating and populating the database. Here is why — we must take into consideration the foreign key dependencies across tables before creating and dropping tables and then inserting data into these tables. We understand that SQLite is relatively flexible when it comes to dropping tables without respecting foreign key dependencies — given two tables A and B, where B has a foreign key referenced from A, dropping A before having dropped B will not show an error in the SQLite

server, however, we try to respect this condition here in order for universal applicability of the queries.

SQL queries input into the **SQLite interface** are provided below. We choose the interesting excerpts of our code. The script entitled AskReddit_SQL.sql is to be executed to reproduce the results that are given below.

We first created the database entitled reddit.db by using the following query. This database will store all the created tables which in turn will contain the data to be used for Section II - Exploration of the data.

```
-- Creating the database reddit
.open reddit.db
```

Once this is in place, we can now proceed to creating the tables.

a. Dropping tables

We first must ensure that the tables we are going to create do not already exist. This step also serves to erase any previously existing data in these tables. We will drop tables respecting the foreign dependencies, i.e., we drop the tables that have foreign keys referenced before dropping the tables from where the said foreign keys are referenced.

The table `tbl_comment` or the "comment" table is dropped after tables `depends`, `removed`, `score` and `is_distinguished` as these four tables are linked to the `tbl_comment` through the key `id`.

```
DROP TABLE IF EXISTS depends;
DROP TABLE IF EXISTS removed;
DROP TABLE IF EXISTS score;
DROP TABLE IF EXISTS is_distinguished;
DROP TABLE IF EXISTS tbl_comment;
DROP TABLE IF EXISTS author;
DROP TABLE IF EXISTS distinguished;
```

```
DROP TABLE IF EXISTS controversy;
DROP TABLE IF EXISTS subreddit;
DROP TABLE IF EXISTS removal;
DROP TABLE IF EXISTS tbl_parent;
```

The queries below will drop the additional tables we created to store the imported data from the .csv files. The tables, as we shall see below will feed the data into the database tables.

```
DROP TABLE IF EXISTS depends_dt;
DROP TABLE IF EXISTS removed_dt;
DROP TABLE IF EXISTS score_dt;
DROP TABLE IF EXISTS is_distinguished_dt;
DROP TABLE IF EXISTS tbl_comment_dt;
DROP TABLE IF EXISTS author_dt;
DROP TABLE IF EXISTS distinguished_dt;
DROP TABLE IF EXISTS controversy_dt;
DROP TABLE IF EXISTS subreddit_dt;
DROP TABLE IF EXISTS removal_dt;
DROP TABLE IF EXISTS tbl_parent_dt;
```

b. Creating tables

We may now proceed to creating the tables as follows. SQLite ignores the numerical limits imposed in parentheses for the variables, but they are included anyway for reproducibility in other SQL environments[1]. Each of the tables is created and constraints are added, with primary keys prefixed with '`pk_`' and secondary keys prefixed with '`fk_`'. In case a table has several primary keys -- consider the example of the table `is_distinguished` we see that these keys can be defined in a single command line. Each foreign key is to referenced on a separate line, clearly indicating the table it has been referenced from.

We state two queries below to illustrate the creation of the tables

```
CREATE TABLE IF NOT EXISTS author(
      author VARCHAR(255),
```

[1] https://www.sqlite.org/datatype3.html

```
      CONSTRAINT pk_author PRIMARY KEY (author)
);

CREATE TABLE IF NOT EXISTS distinguished(
      distinguished VARCHAR(9), -- 9 is chosen because the
longest possible word in this field is 'moderator', having 9
letters.
      CONSTRAINT pk_distinguished PRIMARY KEY (distinguished)
);

CREATE TABLE IF NOT EXISTS controversy(
      controversiality NUMBER(1),
      CONSTRAINT pk_controversy PRIMARY KEY
(controversiality)
);

CREATE TABLE IF NOT EXISTS removal(
      removal_reason VARCHAR(225),
      CONSTRAINT pk_removal_reason PRIMARY KEY
(removal_reason)
);

CREATE TABLE IF NOT EXISTS subreddit(
      subreddit_id VARCHAR(255),
      subreddit VARCHAR(255),
      CONSTRAINT pk_subreddit PRIMARY KEY (subreddit_id)
);

CREATE TABLE IF NOT EXISTS tbl_comment(
      id VARCHAR(255),
      created_utc INTEGER,
      name VARCHAR(255),
      cl_body VARCHAR (255),
      edited INTEGER,
      author_flair_css_class VARCHAR(255),
      author_flair_text VARCHAR(255),
      author VARCHAR(255),
      controversiality NUMBER(1),
      subreddit_id VARCHAR(255) ,
      CONSTRAINT pk_tbl_comment PRIMARY KEY (id),
      CONSTRAINT fk_tbl_comment_author FOREIGN KEY (author)
REFERENCES author(author),
      CONSTRAINT fk_tbl_comment_controversiality FOREIGN KEY
(controversiality) REFERENCES controversy(controversiality)
);
```

```
CREATE TABLE IF NOT EXISTS is_distinguished(
      id VARCHAR(255),
      distinguished VARCHAR(255),
      CONSTRAINT pk_is_distinguished PRIMARY KEY (id,
distinguished),
      CONSTRAINT fk_is_distinguished_id FOREIGN KEY (id)
REFERENCES tbl_comment (id),
      CONSTRAINT fk_is_distinguished_distinguished FOREIGN
KEY (distinguished) REFERENCES distinguished (distinguished)
);

CREATE TABLE IF NOT EXISTS score(
      id NUMBER(2),
      score NUMBER(2),
      ups INTEGER,
      downs INTEGER,
      score_hidden BOOLEAN,
      gilded INTEGER,
      CONSTRAINT pk_score PRIMARY KEY (id),
      CONSTRAINT fk_score_id FOREIGN KEY (id) REFERENCES
tbl_comment(id)
);

CREATE TABLE IF NOT EXISTS tbl_parent(
      parent_id VARCHAR(255),
      link_id VARCHAR(255),
      CONSTRAINT pk_parent PRIMARY KEY (parent_id)
);

CREATE TABLE IF NOT EXISTS removed(
      id VARCHAR(255),
      removal_reason VARCHAR(255),
      CONSTRAINT pk_removed PRIMARY KEY (id, removal_reason),
      CONSTRAINT fk_removed_id FOREIGN KEY (id) REFERENCES
tbl_comment (id),
      CONSTRAINT fk_removed_removal_reason FOREIGN KEY
(removal_reason) REFERENCES removal (removal_reason)
);

CREATE TABLE IF NOT EXISTS depends (
      id VARCHAR(255),
      parent_id VARCHAR(255),
      CONSTRAINT pk_depends PRIMARY KEY (id, parent_id),
      CONSTRAINT fk_depends_id FOREIGN KEY (id) REFERENCES
```

```
tbl_comment(id),
      CONSTRAINT fk_depends_parent_id FOREIGN KEY (parent_id)
REFERENCES parent (parent_id)
);
```

c. Viewing the schema

After having created the tables, we must firstly ensure if all the tables were created. This is simply seen with the command below, which gives us the output captured in figure 1. Note how we do not need the semi colon ';' for SQLite to understand when the command ends.

```
--list all available tables
.tables
```

```
sqlite> .tables
sqlite> .tables
author            distinguished     removed           tbl_comment
controversy       is_distinguished  score             tbl_parent
depends           removal           subreddit
```

Figure 2 : Listing all tables in the reddit database

We can also explore the structure of a table after its creation - this is recommended if we want to have a quick reminder of the primary/foreign keys associated with this table and also error-proofing the schema of the tables. We output the result in figure 3 for a table to give the reader an idea of the output.

```
--get the description of a table (here: score)
.schema score
```

```
sqlite> .schema score
CREATE TABLE score(
id NUMBER(2),
score NUMBER(2),
ups INTEGER,
downs INTEGER,
score_hidden BOOLEAN,
gilded INTEGER,
CONSTRAINT pk_score PRIMARY KEY (id),
CONSTRAINT fk_score_id FOREIGN KEY (id) REFERENCES tbl_comment(id)
);
```

Figure 3 : Exploring the schema of the `score` *table of the reddit database*

d. Inserting data

The original unified .csv file exp_askreddit.csv containing all the AskReddit data was split into several small .csv files each meant to be inserted into the table the file was named after. This was done to achieve greater efficiency in data insertion given the size of the data that is to be inserted. Only for the tables `is_distinguished`, `removed` and `depends`, we had to make use of the original .csv file as the splitted .csv files were not provided. However, this did not have a significant impact on the insertion time. The populating of the database can be presented in two parts :

i. Importing the .csv files

In order to import the .csv files we must first set the mode to CSV to instruct the command-line shell program to interpret the input file as a CSV file. The command below does the job.

```
--switch to csv mode
.mode csv
```

Now we can safely import the .csv files as a temporary table. For example, we import the author.csv corresponding to the `author` table created above and store it as a temporary table called `author_dt. We also import the` exp_askreddit.csv to insert data into the three tables mentioned above. The queries below show how .csv file corresponding to each table have

been imported.

```
.import exp_author.csv author_dt
.import exp_distinguihshed.csv distinguished_dt
.import exp_controverse.csv controversy_dt
.import exp_removal.csv removal_dt
.import exp_score.csv score_dt
.import exp_parent.csv parent_dt
.import exp_subreddit.csv subreddit_dt
.import exp_comment.csv comment_dt
.import exp_askreddit.csv askreddit_dt
```

ii. Inserting data from the temporary tables into the database tables

This sub-section details the insertion of the data from the temporary tables into the tables of reddit.db. The queries are comprehensively documented in the SQL script. We illustrate the process of inserting data using a query for the `score` table

```
INSERT INTO score
SELECT id, score, ups, downs, score_hidden, gilded
      FROM score_dt;
```

Upon the insertion of data into each table, we can run the following command to obtain the tables inserted in each table, as shown in figure 4.

```
SELECT name,
      SUM("pgsize")
FROM "dbstat"
WHERE name NOT LIKE '%sqlite%'
GROUP BY name;
```

```
author      10891264
controvers  4096
depends     111403008
distinguis  4096
is_controv  74551296
is_disting  70750208
parent      41177088
removal     4096
removed     70348800
score       91201536
subreddit   4096
tbl commen  915931136
```

Output 1 : Obtaining the number of lines inserted in the score table

e. Creating indices

Indexes are used in relational databases and can be used to enforce the uniqueness of the elements in a row. This will improve the speed of joins across tables and thus improve the performance of the query as we shall see in Section II.

Given below is an index called `id_comments` created on the `id` column of the `tbl_comment` or the comments table. Similarly, `id_score` is created for the

```
CREATE UNIQUE INDEX id_comments
ON tbl_comment (id);

CREATE UNIQUE INDEX id_score
ON score (id);
```

f. Subsection of 5MB

We shall present this part at the end of the paper after having presented all our analyses on the most representative part of the database.

Section II

This section presents a concise investigation of the database. We shall first perform an exploratory analysis of our database before proceeding to more complex queries.

1. What is inside the database ?

Let us start by obtaining some descriptive statistics on the comments table, with regard to the length of the comment in characters.

```
SELECT ROUND(AVG(LENGTH(cl_body)),2) AS mean_length,
        MAX(LENGTH(cl_body)) AS max_length,
        MIN(LENGTH(cl_body)) AS min_length
FROM tbl_comment;
```

The SQLite output for the character count is given below. The format was obtained using

```
.mode box
```

mean_length	max_length	min_length
143.57	10006	0

Output 2: Descriptive statistics of the comments' character count

Now let us look at how the score is distributed.Output 3 shows that the distribution is likely to be skewed, as we shall see as we visualize the data.

```
SELECT ROUND(AVG(score), 2) AS mean_score,
       MAX(score) AS max_score,
       MIN(score) AS min_score
FROM score;
```

mean_score	max_score	min_score
12.61	6761	-333

Output 3: Descriptive statistics of the score

Let us see how many Reddit golds (gilded) comments have been awarded. This information is derived from the `score` table.

```
SELECT MAX(gilded), MIN(gilded) FROM score;
```

MAX(gilded)	MIN(gilded)
12	0

Output 4: Descriptive statistics of the gilded feature

Let us briefly look at the timeline of the dataset, i.e., first and last posting date.

```
SELECT DATETIME(MIN(created_utc), 'unixepoch') AS
first_post, DATETIME(MAX(created_utc), 'unixepoch') AS
last_post
        FROM tbl_comment;
```

first_post	last_post
2015-05-01 00:00:00	2015-05-31 23:59:59

Output 5: Post timeline : May, 2015

Now we proceed to making some visualizations. In order to do this, we used SQlite in a Python programme. SQLite3 can be integrated with Python using the `sqlite3` module. Let us first configure this step by creating a connection to the `project_reddit.db` created in Section I. This now gives us access to the contents of the database.

```
# Create a connection to the database
conn = sqlite3.connect('project_reddit.db')
# Get an object to execute SQL commands and retrieve data
c = conn.cursor()
```

SQL queries were then used to obtain results that were then plotted using `matplotlib` on Python. We have provided the explanations for 2 of the 5 queries, those that warranted further explanation - all these queries are found in the .py script handed. We divide this sub-section into 2 parts :

a. Comment-specific visualizations

A key question while doing this exploratory analysis was whether the score of the comments were driven by the length. The intuition was mostly, extremely long comments are less likely to receive a high score compared to relatively shorted ones as they are less likely to be read in its entirety. We see that there is no clear relation but relatively shorter comments do outperform extremely long ones.

We shall illustrate using the query that was used to produce figure 4. The SQL query (in orange) calculates the character count of each comment and the corresponding score. The end result is a list that provides the character count of the comment and the corresponding aggregation, or the mean score of the comment. The results are then extracted from this list,

and we obtain the plot shown in figure 4.

```
store = list(c.execute(''' SELECT length(comm.cl_body) as char_count,
                                  AVG(sc.score) as karma
                                  FROM tbl_comment comm, score sc
                                  WHERE comm.id = sc.id
                                  GROUP BY char_count
                                  ORDER BY karma DESC ''' ))
char_count = [store[i][0] for i in range(len(store))]
karma = [store[i][1] for i in range(len(store))]

plt.scatter(x = char_count, y = karma, color = "purple")
plt.xlabel('Character count of the comment', fontsize = 12)
plt.ylabel('Comment karma', fontsize = 13)
plt.title('Plotting the karma by the length of the comment')
plt.show()
```

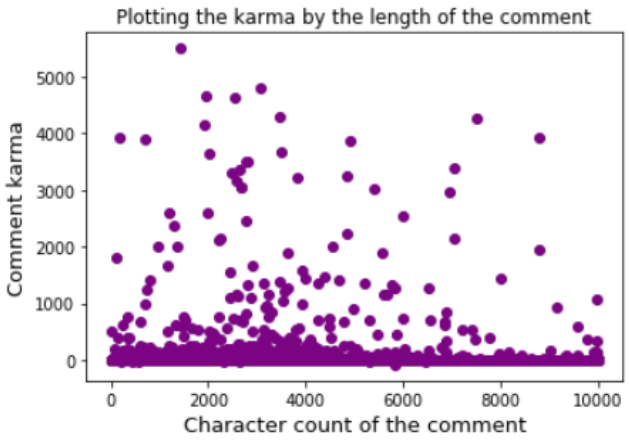


Figure 4: Scatter plot showing the distribution of comment karma by the character count of the comment.

Next we look at the average comment karma by distinction of the author - special, unspecified or moderator. Reddit users would know that moderator comments are essentially made to request proper language, pertinence to the original post etc. It is thus normal that the average score is close to zero. The difference between the average score between special users and the unspecified is remarkable.

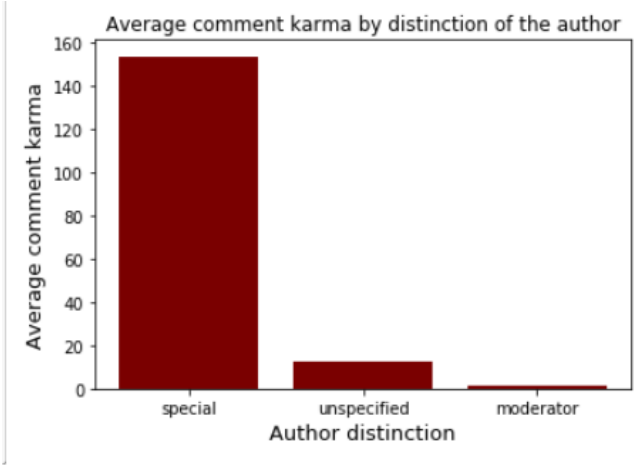


Figure 5 ; Average score by author distinction

Let us look at the number of times a comment was gilded, or in other words received Reddit gold which can be interpreted as an award. We can safely say that there is a generally positive correlation between the former and the average score that the comment receives.

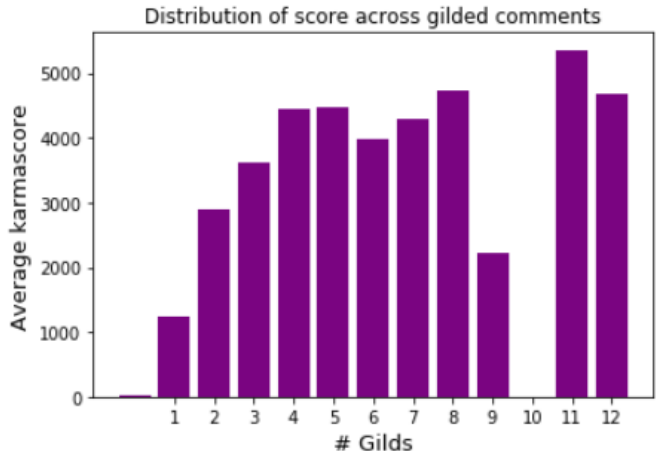


Figure 6 : Score of the comment by the number of times a comment was gilded.

b. Time specific characteristics

The key question here is when are Reddit users most active - we have shown the evolution of user activity by hour of the day and day of the week.

Let us dissect the query for the Reddit user by hour of the day activity below.

In order to perform time specific analyses, we first instructed SQlite using the `datetime()` function to define the `created_utc` column, previously defined in `TEXT` format to a `UNIX` timestamp. Unix time is a single signed number that increments every second, which makes it easier for computers to store and manipulate than conventional date systems. Interpreter programs can then convert it to a human-readable format. The Unix epoch is the time 00:00:00 UTC on 1 January 1970.

Following this, the `strftime()` function extracts the hour indicated by '**%H**' from the given time variable. This makes it straightforward to then calculate the number of comments made by the hour. The same was done for the days of the week, where we used '**%w**' to instruct SQLite to extract the week and then we aggregate the comments over the days of the week;

```
store = list(c.execute(''' SELECT  strftime('%H', datetime(created_utc,
                           'unixepoch')) as hour_of_post,
                                  COUNT(*) FROM tbl_comment
                                  GROUP BY hour_of_post '''))
hours = [store[i][0] for i in range(len(store))]
total_posts = [store[i][1] for i in range(len(store))]
plt.plot_date(hours, total_posts, color='green', linestyle='dashed',
marker='o',
     markerfacecolor='maroon', markersize=12)
plt.xticks(ticks = hours[0::2])
plt.xlabel('Hours, May 2015', fontsize = 12)
plt.ylabel('# of comments', fontsize = 12)
plt.title('Comments, by hours')
plt.show()
```

The figure below shows user activity by the hour for the entire month of May 2015. Clearly Reddit users are more active during the evening, from around 15h to 2am as shown in Figure 7. Figure 8 shows that on May 2015, users have mostly been active on Fridays, and less on week days.

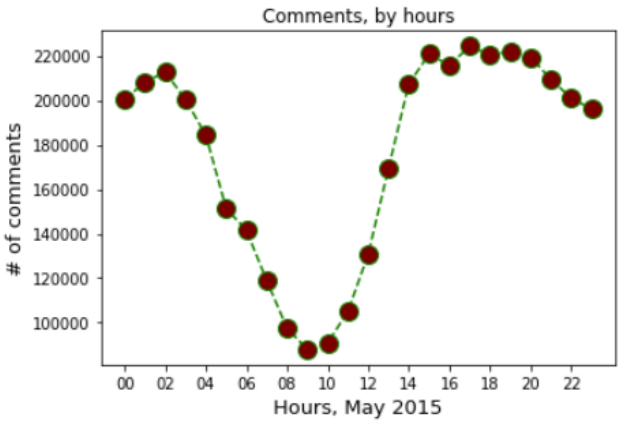


Figure 7 : Comments by hours of the day

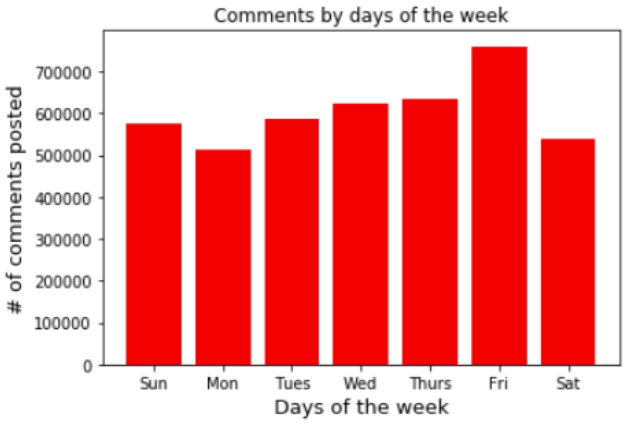


Figure 8 : Comments by the days of the week

Let's examine some variables in our dataset and see if they have valuable information or not. First we look at the flairs and controversiality.

```
SELECT DISTINCT author_flair_css_class,
                author_flair_text,
                controversiality
FROM tbl_comment
LIMIT 10;
```

	0
	1

Output 6 : Exploring the controversiality variable

We see that both flair variables have only empty strings in them and that controversy is a binary variable for each comment.

```
SELECT DISTINCT downs,
                score_hidden,
                gilded
FROM score;
```

0	0	0
0	0	1
0	1	0
0	0	2
0	0	3
0	0	4
0	1	1
0	0	5
0	0	11
0	0	12
0	0	8
0	0	9
0	0	6
0	0	7

Output 7 : Downvotes, Hidden and Gilded columns

Looking at downvotes, hidden, and gilded. We find that hidden is binary and folded can be any integer value. Strangely, downvotes only contains zeroes, so we investigate further.

```
SELECT score,
       ups
FROM score
WHERE score != ups;
```

<result is blank>

Output 8 : The score is equal to the ups column

```
SELECT count(ups)
FROM score
WHERE score < 0;
```

result: 113438

Output 9 : Downs contain no information

From these two commands we can discern that downs contains no information, and furthermore that score is redundant to ups. Usually we would expect ups and downs to both be positive numbers such that ups - downs = score. Since the information was not available, all downvotes are set to zero and comments with more downvotes than upvotes simply are assigned a negative score and value to ups.

Next we look at some distributions and quantiles for different variables. To do this, we export the data to python.

```
SELECT author,
       sum(1)
FROM tbl_comment
GROUP BY author;
```

Output 10 : Qunatiles of the posts by authors

0.1	1
0.4	1
0.5	2
0.6	3
0.7	4
0.8	6
0.9	13
0.93	18
0.95	24
0.97	36
1	312007

Here, we examine how many posts users write. The median is only 2 posts!

```
SELECT count(author)
FROM
  (SELECT author,
          sum(1) AS cnt
   FROM tbl_comment
   GROUP BY author)
WHERE cnt > 36;
```

The output is 17110 rows. This implies that users with more than 36 posts are in the 97th percentile, only 17,110 users in our database.

Next we look at quantiles for upvotes :

```
SELECT ups
FROM score;
```

Output 11: Quantiles for upvotes

0.1	1
0.6	1
0.7	2

0.8	*3*
0.9	*6*
0.93	*10*
0.95	*16*
0.97	*36*
0.99	*212*
1	*6761*

```
SELECT count(id)
FROM score
WHERE ups > 212;
```

The output is 42256. The 99th percentile contains 42256 comments with upvotes above 212.

Last, we look at the quantiles for number of replies each comment receives. We rely on these results in our main analysis, so we save them to a temporary table.

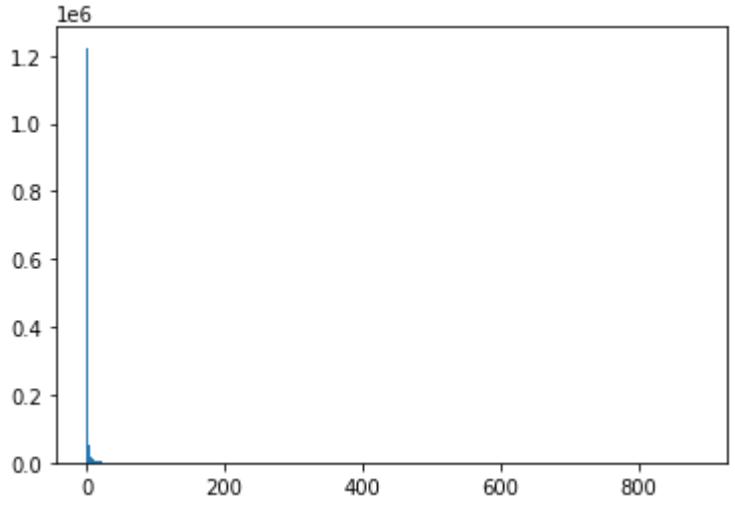
```
CREATE TEMP TABLE replycount(id VARCHAR(255),
                             replies INTEGER);


INSERT INTO replycount
SELECT substr(parent_id, -7) AS id,
      count(id) AS replies
FROM depends
WHERE parent_id not like 't3%'
GROUP BY parent_id
ORDER BY replies DESC;

SELECT *
FROM replycount;
```

Note: for this query we must exclude posts starting with `t3_`. The reason is that according to the reddit API, this prefix indicates the content is an original post, with its own link on the front page, rather than a comment on an original post. Of course, original posts will have very higher numbers of replies compared to comments so we don't want to consider them. Additionally, the content of these posts are not handled the same way as the id starting with `t3` and so in this dataset we can not link it back to anything.
For the remaining `parent_id` which all start with t1_, we use `substr()` to remove the `t1_` prefix and match it to its `id` in format of the rest of the database.

Figure 9 : Number of replies by comments



Output 12 : Quantiles for replies

0.1	1
0.7	1
0.9	3
0.97	7
0.99	16
0.999	69
0.9999	188
0.99999	357.9515
0.999999	629.56905
1	884

Number of replies is extremely skewed, with most users receiving one reply (see Figure 9). Even the median is 1 reply, and only 90th percentile comments get 3 replies, as is shown in Output 12.

```
SELECT sum(1)
FROM replycount
WHERE replies > 16;
```

result: 13126

```
SELECT count(DISTINCT t.author)
FROM replycount AS r
LEFT JOIN tbl_comment AS t ON t.id == r.id
WHERE r.replies > 16;
```

result: 10410

```
SELECT count(DISTINCT author)
FROM tbl_comment;
```

result: 570735

```
SELECT count(DISTINCT id)
FROM tbl_comment;
```

result: 4234970

So this (10410/570735) = 1.8% of authors produce (13126/4234970) = 0.3% of comments.

2. Finding Interesting Comments & Authors

A. What is Interesting?

On other social media platforms such as Twitter, controversy can increase engagement. This is because users only way of voicing their disagreement or disapproval is through replying. However on Reddit, downvoting is another way users may be able to voice their disapproval (although this behavior is explicitly discouraged). When a post is marked as controversial, it means that a comment has a ratio of upvotes and downvotes which is close to 0.5.

Additionally, all social media platforms care about engagement. Most social media platforms have investors, and engagement metrics are crucial for signalling that your platform is successful to investors.

Based on these factors, we are curious about users who generate a lot of engagement and also get significant numbers of downvotes as well as upvotes. Are controversial users 'rewarded' with lots of replies? Are users or comments with lots of replies often controversial? We believe these questions are interesting as it may reveal something different about incentives on reddit in comparison to other platforms.

In order to do this investigation, we have three steps. First, we examine comments which generate many replies. Second, we examine users with high average comment controversy. Finally, we take the intersection of both groups at the 99th percentile and examine this. We define as an interesting user, these users corresponding to this group, and interesting comments as those which they post.

A.1. High-Reply Comments

Based on the quantiles from the earlier section, we know that the 99th percentile of most-replied comments is 16. We use this as our cutoff to examine the subpopulation of posters who have a high-reply comment.

```
SELECT r.replies,
       s.ups,
       s.score_hidden,
       s.gilded,
       t.controversiality,
       t.author,
       length(t.cl_body),
       t.edited,
       s.id
FROM
  (SELECT *
   FROM replycount
   WHERE replies > 16) AS r
LEFT JOIN score AS s ON s.id = r.id
LEFT JOIN tbl_comment AS t ON t.id = r.id;
```

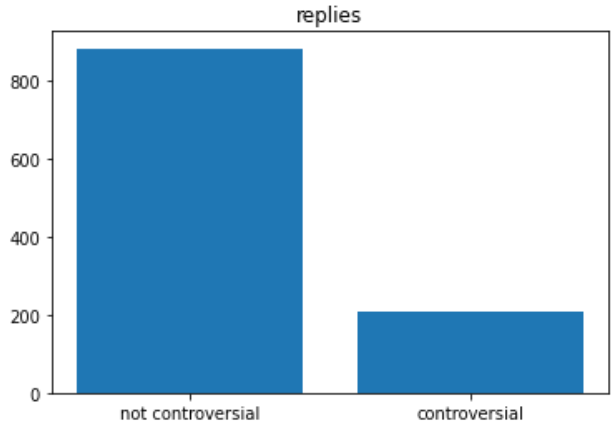


Figure 10. Count of controversial and non-controversial comments

From Figure 10,we see that controversial comments represent only about one sixth of high-reply comments.

```
SELECT avg(controversiality)
FROM tbl_comment;
```

result: 0.012330193602316

However here we can see that this ratio is much higher than overall. (This may be in part because comments default to 1 upvote and 0 downvotes at time of posting. Since we don't have access to separate counts of ups and downs we cannot reliably separate out these

comments.)

Next we examine some scatter plots with replies on the y-axis and colours for controversial vs non-controversial comments.

Controversial (0) v/s Not controversial (1)

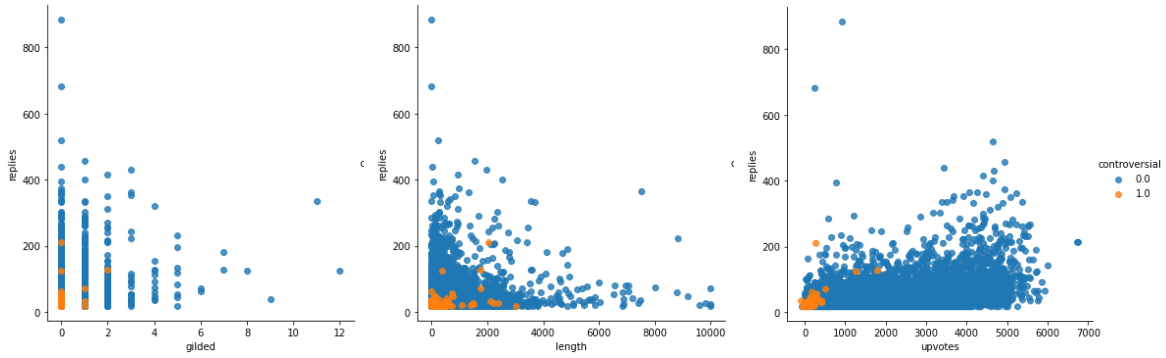


Figure 11 : Gilded *Figure 12: comment length* *Figure 13 : Upvotes*

We see that controversy is mostly correlated with no gildeds (Figure 11), lengths of around 1000 (Figure 12), and much fewer upvotes (Figure 13) than non-controversial posts. Also, there may be a slight positive relationship between number of upvotes and number of replies. This makes sense as a higher score means your comment will be viewed by more people.

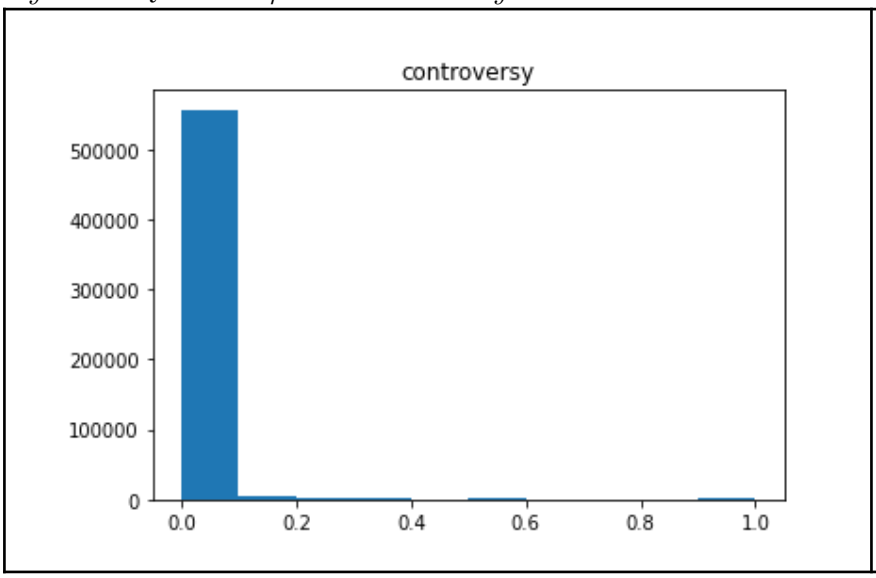
A.2. Controversial Users

Next we examine controversial users. First we look at all users, and look for trends between other variables and high controversy levels, then we look at a subset after choosing a cutoff. We calculate controversy for a user by taking the number of controversial posts divided by all posts they have made.

```
SELECT t.author,
       avg(t.controversiality),
       avg(s.ups),
       sum(s.gilded),
       avg(length(t.cl_body)),
       avg(t.edited),
       count(t.id)
FROM tbl_comment AS t
LEFT JOIN score AS s ON t.id == s.id
GROUP BY author
```

```
ORDER BY avg(controversiality) DESC;
```

Figure 14 : Quantiles of the controversiality column



Output 13: Table of Quantiles

Quantiles

0.95	0.011673
0.96	0.032258
0.97	0.0625
0.98	0.125
0.99	0.333333
0.995	0.5
0.9963	0.666667
0.999	1
1	1

We see that users with very small average controversy make up the overwhelming majority of the distribution. Our bin-size prevents us from getting a better idea of how they are distributed at the small sizes though. From the quantile table on the left, we can see that 99.5% of users have average controversy less than 0.5. Also, we see that the 99th percentile is 0.3333, which will be our cutoff later.

To get a better idea, we try selecting only users who have had at least one controversial post (this is equivalent to having a non-zero average controversiality)

```
SELECT count(DISTINCT author)
FROM
  (SELECT author
   FROM tbl_comment AS t
   GROUP BY author
   HAVING avg(controversiality) > 0);
```

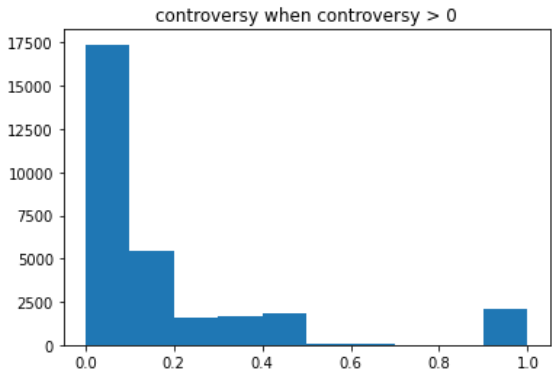


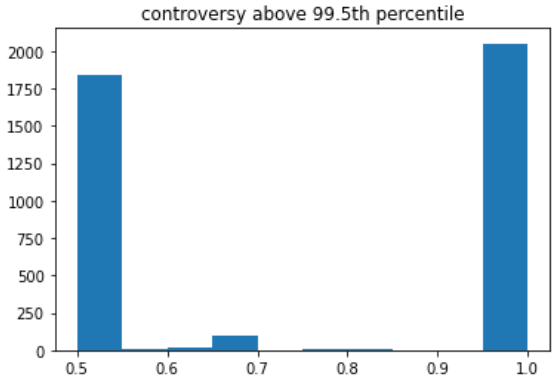
Figure 15. Users with at least 1 controversial post

From figure 15, seeing the difference in the y-axis, this has dramatically shrunk the resulting query size. We also notice there is a surprisingly large outlier group near 1.0, as well as a large drop off at 5.0. So of the subset of users who have more than half their comments marked controversial, most of them have all their comments as controversial. This definitely points to specific behaviour patterns, where users with high average controversy rarely post and/or when they do, it is to say something which other users have a hard time deciding whether to upvote or downvote.

To examine the users who have more than half their comments marked as controversial, we again take a subset.

```
SELECT count(DISTINCT author)
   FROM
   (SELECT author
    FROM tbl_comment AS t
    GROUP BY author
    HAVING avg(controversiality) >= 0.5);
```

Figure 16 : Examining controversy for users having more than half their comments marked as controversial,



The query results in 4032 observations. And slightly more than half of these users have ALL their comments marked as controversial!

Next we look for any relationship of 'controversy' with other variables as shown in the figures 17, 18 and 19 below.

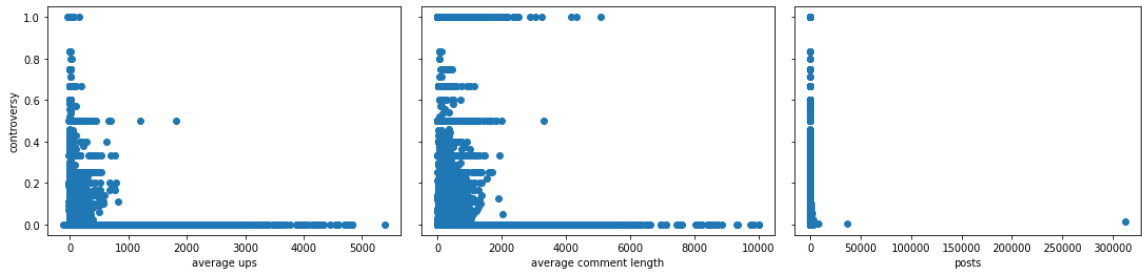


Figure 17 : With Avg. ups *Figure 18 :With Avg. Comment length* *Fig 19: With Posts*

Considering the distribution of controversy among authors, these results don't show any visual relationship between controversy and the characteristics of average upvotes (Figure 17), average comment length (Figure 18), or total number of posts (Figure 19).
We do see some strange outliers which we decided to investigate further.

```
SELECT t.author,
       avg(t.controversiality),
       avg(s.ups),
       sum(s.gilded),
       avg(length(t.cl_body)),
       avg(t.edited),
       count(t.id) AS totposts
FROM tbl_comment AS t
LEFT JOIN score AS s ON t.id == s.id
GROUP BY author
HAVING totposts > 10000
ORDER BY avg(controversiality) DESC;
```

	author	controversy	averageups	totgilded	avglength	posts
27620	[deleted]	0.015016	4.336576	12	21.757749	312007
30129	AutoModerator	0.002493	1.000975	1	974.928881	36910

Output 14: Controversy, Ups, Total Gilded, Character length and number of posts by groups of authors

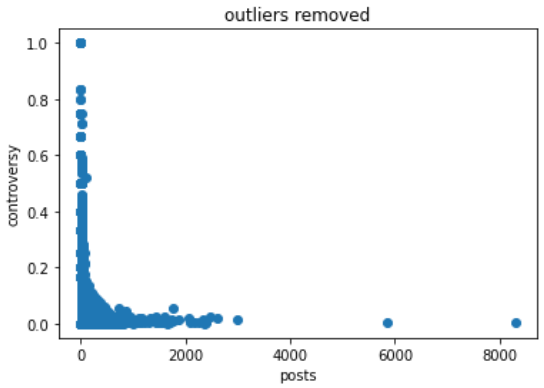


Figure 20 : Controversy by posts, upon removing outliers

The two outliers which had extremely large numbers of posts were [deleted], which is not one person but a variety of people who deleted their account. The second outlier is AutoModerator which is a robot.

After removing the outliers we can see that posters which high controversy levels have very few posts. So as a user posts more their controversy level is likely to decrease. This again can be intuitive when we consider that the majority of posts do not get any upvotes or downvotes, instead staying at 1. Thus when one posts more they are more likely to have posts which nobody sees, and thus decrease their average controversy score.

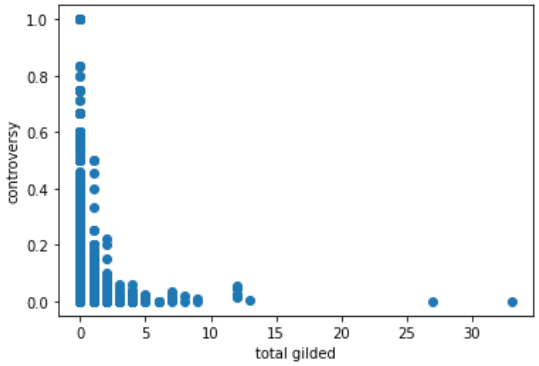


Figure 21 : Gilded by controversiality after removing outliers

Here we observe that **controversial posts never get gilded**, especially above the threshold of 0.5. Again we examine the two outliers!

```
SELECT t.author,
       avg(t.controversiality),
       avg(s.ups),
       sum(s.gilded) AS totgild,
       avg(length(t.cl_body)),
       avg(t.edited),
```

```
      count(t.id)
FROM tbl_comment AS t
LEFT JOIN score AS s ON t.id == s.id
GROUP BY author
HAVING totgild > 25
ORDER BY avg(controversiality) DESC;
```

	author	controversy	averageups	totgilded	avglength	avgedited	posts
384756	Poem_for_your_sprog	0.0	2643.047619	33	446.714286	3.408863e+08	21
504084	Donald_Keyman	0.0	728.125000	27	122.602564	2.019537e+08	312

Figure 22 : Outlier inspection for authors by controversiality, ups, gilded, comment length, and posts

It looks like the first one is some gimmicks account, based on the name and the post length it shows he probably posts original poems. The other one, Donald Keyman, is a well respected user who has helped many students with SAT studies.

A.3. Intersection

We rely on two groups to generate our intersection.

- Users who have a comment in the 99th percentile of most-replies
- Users in the 99th percentile of average controversy for all their comments.

```
SELECT t.author,
       avg(t.controversiality),
       avg(r.replies),
       avg(s.ups),
       avg(s.score_hidden),
       sum(s.gilded),
       avg(length(t.cl_body)),
       avg(t.edited),
       count(t.id)
FROM
  (SELECT *
   FROM replycount
   WHERE replies > 16) AS r
LEFT JOIN tbl_comment AS t ON t.id == r.id
LEFT JOIN score AS s ON t.id == s.id
GROUP BY t.author
```

```
HAVING avg(controversiality) >= 0.3333;
```

	author	controversy	avgreplies	avgups	avghidden	avggilded	avglength	avgedited	posts
0	Algamain	1.0	18.0	-49.0	0.0	1	522.0	0.000000e+00	1
1	Allegretta	1.0	26.0	43.0	0.0	0	101.0	0.000000e+00	1
2	Avizard	0.5	24.5	724.5	0.0	0	53.5	0.000000e+00	2
3	BaneOfWindmills	0.5	29.0	326.5	0.0	0	1186.5	0.000000e+00	2
4	BangedLoose	1.0	23.0	4.0	0.0	0	46.0	0.000000e+00	1
...	...	...	...	...	...	...	...	...	...
63	phalseprofits	1.0	27.0	67.0	0.0	0	1479.0	1.432910e+09	1
64	prisoner76	1.0	17.0	10.0	0.0	0	45.0	0.000000e+00	1
65	tharealbenjy	1.0	23.0	129.0	0.0	0	153.0	0.000000e+00	1
66	wereux	1.0	17.0	-11.0	0.0	0	221.0	0.000000e+00	1
67	wishforagiraffe	1.0	18.0	61.0	0.0	0	478.0	0.000000e+00	1

68 rows × 9 columns

Figure 23: The results are a subset of 68 users, .00119% of authors in our database.

Next, we generate pair-plots in Python to examine the distributions and the relationship between different variables. Pair plots plot each of our variables on both the x and y axis, so that we can see scatter plots for each combination of variables.

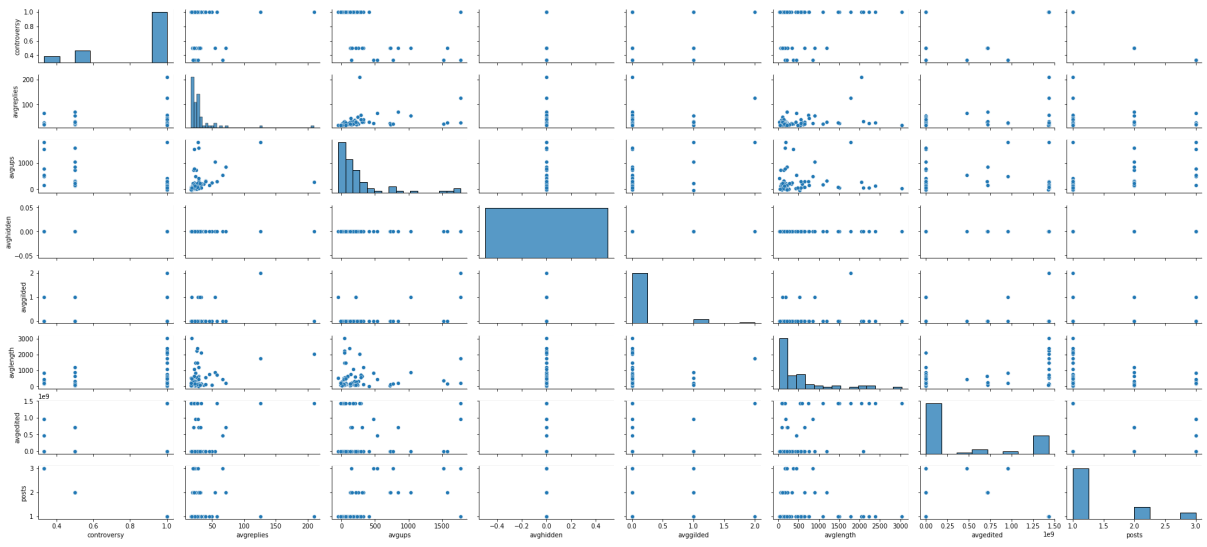
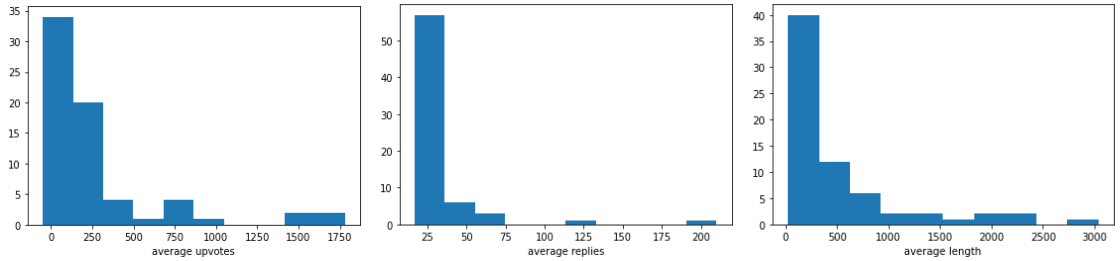


Figure 24 : Pair plots - Controversy, avg. replies, avg. ups, avg. gilded, avg. length and avg. edited.

We observe in this subset that there is a positive relationship between average replies

and average upvotes for a user. We also observe that most users only have one post and at most have three posts. Users with more posts have drastically lower controversy levels.

From the pairplot we will magnify the interesting distributions and comment on them.



We see that these users tend to have short posts, with 25 or fewer replies and average upvotes between 0 and 250.

Next, we will look at the timing of posts for this subset of users and compare it with the entire set of users.

```
SELECT strftime('%H', datetime(tbl_comment.created_utc, 'unixepoch')) AS
hour_of_post,
       COUNT(tbl_comment.id),
       m.author
FROM
  (SELECT *
   FROM
      (SELECT *
      FROM replycount
      WHERE replies > 16) AS r
   LEFT JOIN tbl_comment AS t ON t.id == r.id
   GROUP BY t.author
   HAVING avg(controversiality) >= 0.3333) AS m
LEFT JOIN tbl_comment ON m.author == tbl_comment.author
GROUP BY hour_of_post;
```

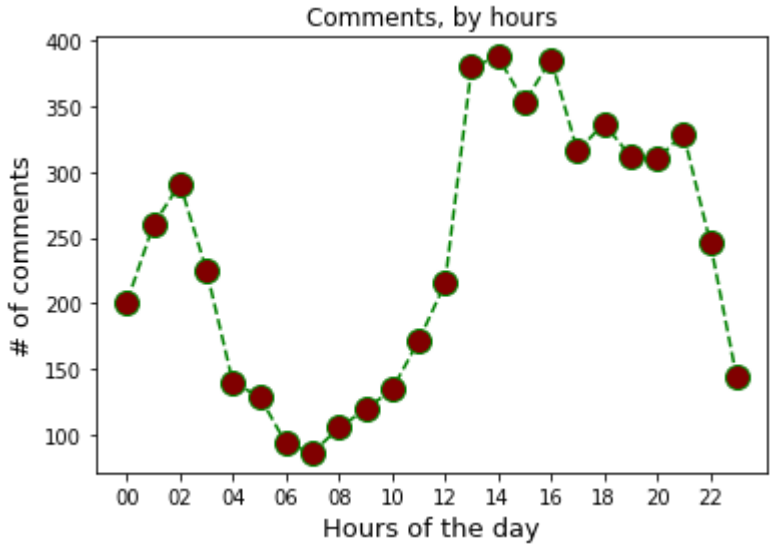


Figure 28 : Focus on subset : number of comments by the hours of the day

Compared to the overall set, these users are relatively less active in the early morning hours.

3. Representative Subset of the Database

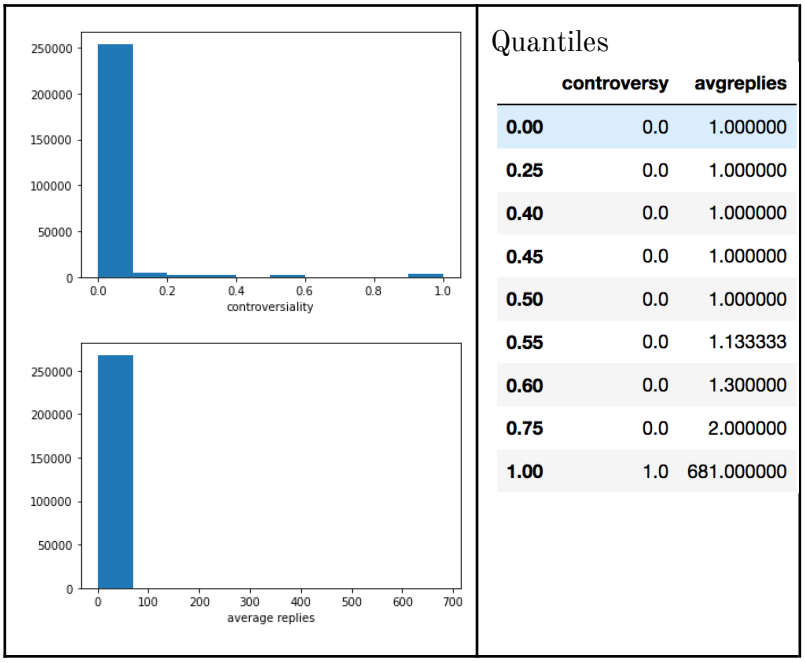
In order to capture a representative group of comments, we first focus on a representative group of users. The motivation of this is our belief that users with similar behaviors will comment similarly. If we just capture comments directly, there may be outlier comments from users with different behavior, and these may bias aspects of our descriptive statistics. After capturing this set we then extract a further subset of representative comments from the representative users.

Additionally, we consider two different sets of representative users, those who have average controversiality of 0 and those who have average controversiality of 1. This is motivated by the very large outlier population of users with average controversiality of 1. We would like to compare the behavior of both of these, otherwise similar, populations.

In practice, we first examine the distribution of average controversy and number of replies among all users.

```
SELECT t.author,
```

```
      avg(t.controversiality),
      avg(r.replies)
FROM
  (SELECT *
   FROM replycount) AS r
LEFT JOIN tbl_comment AS t ON t.id == r.id
LEFT JOIN score AS s ON t.id == s.id
GROUP BY t.author;
```



Quantiles

	controversy	avgreplies
0.00	0.0	1.000000
0.25	0.0	1.000000
0.40	0.0	1.000000
0.45	0.0	1.000000
0.50	0.0	1.000000
0.55	0.0	1.133333
0.60	0.0	1.300000
0.75	0.0	2.000000
1.00	1.0	681.000000

Figure 29 : Distribution of controversiality of posts and its avg. replies

We define representative users as within 5% deviation of the median in terms of average number of replies. For representative comments, we separate two groups by controversial and not controversial. We do this because we are interested in comparing the differences between these two groups and the 45th to 55th percentiles are all 0 average controversy for users, which is not very interesting.

Now we compare different subsets.

```
SELECT t.author,
       avg(t.controversiality) AS controversy,
       avg(r.replies) AS avgreplies,
       avg(s.ups),
       avg(s.score_hidden),
       sum(s.gilded),
       avg(length(t.cl_body)),
```

```
      avg(t.edited),
      count(t.id)
FROM
  (SELECT *
   FROM replycount) AS r
LEFT JOIN tbl_comment AS t ON t.id == r.id
LEFT JOIN score AS s ON t.id == s.id
GROUP BY t.author
HAVING avgreplies >= 1
AND avgreplies <= 1.1334
AND controversy == 1;

SELECT t.author,
      avg(t.controversiality) AS controversy,
      avg(r.replies) AS avgreplies,
      avg(s.ups),
      avg(s.score_hidden),
      sum(s.gilded),
      avg(length(t.cl_body)),
      avg(t.edited),
      count(t.id)
FROM
  (SELECT *
   FROM replycount) AS r
LEFT JOIN tbl_comment AS t ON t.id == r.id
LEFT JOIN score AS s ON t.id == s.id
GROUP BY t.author
HAVING avgreplies >= 1
AND avgreplies <= 1.1334
AND controversy == 0;
```

Distribution of Controversial Users (1797 users)

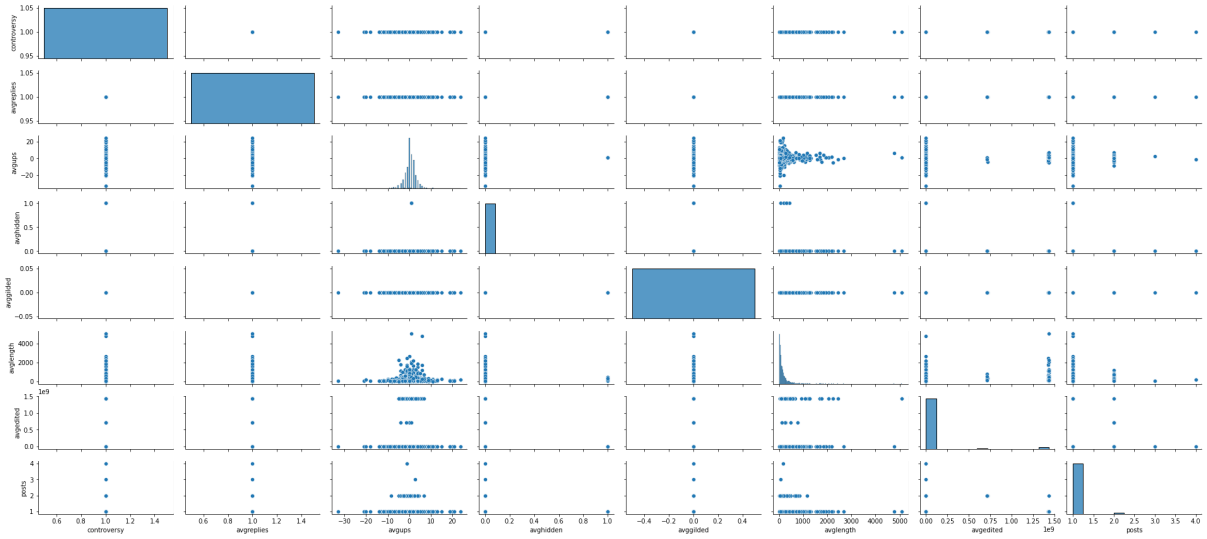


Figure 30 : Pair plot of Controversy, avg. replies, avg. ups, avg. gilded, avg. length and avg. edited for controversial users

Average upvotes are centered at zero. Longer comments are more likely to have upvotes closer to 0 or 1. Other than this we have no relationships.

Distribution of Non-Controversial Users (142775 users)

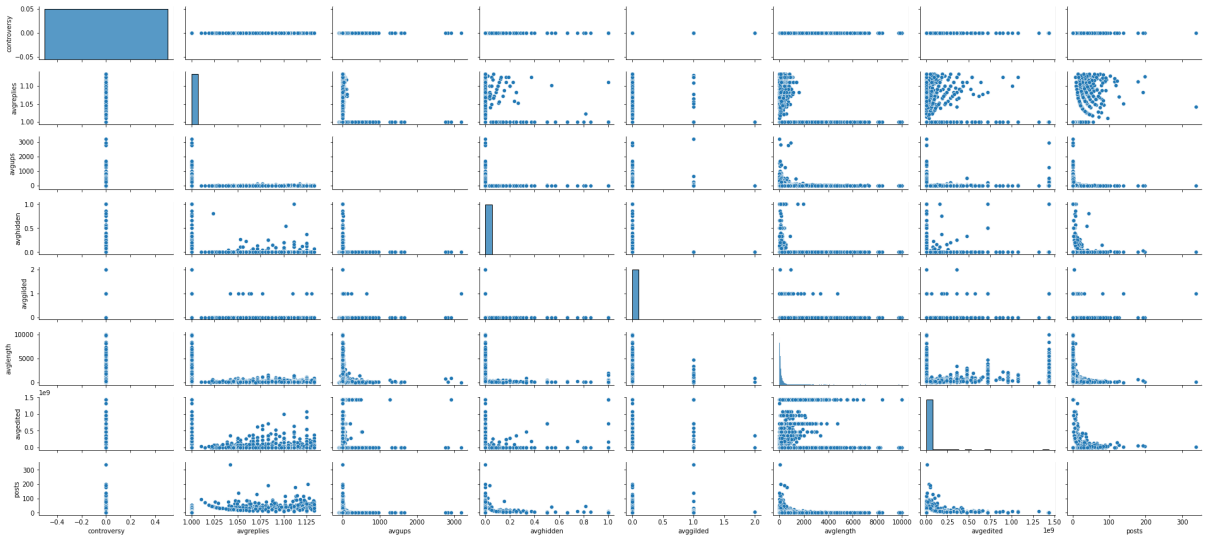


Figure 31 : Pair plot of Controversy, avg. replies, avg. ups, avg. gilded, avg. length and avg. edited for non-controversial users

Here we do not observe comments with negative karma. We also observe a positive relationship between average number of edited comments and average number of replies (when ignoring comments with only one reply).

We observe a negative relationship between total number of posts and average number of replies, which is also true for average hidden and average gilded. This supports what was

mentioned earlier about the more a user comments the less interaction through up or downvotes they receive on average.

Also, users with a high number of posts are more likely to have shorter posts on average.

In our last step, we extract all the comments corresponding to these users. Again keeping two different groups of users, we look at the distribution and relationship between the different sets of comments.

To make things faster and more organized we again use temporary tables. For this query we also perform a query on the join of two tables, then join with two other tables. This approach avoids the query going through many rows and performing slowly.

```
CREATE TEMP TABLE controversialusers(author VARCHAR(255));

CREATE TEMP TABLE noncontroversialusers(author VARCHAR(255));

INSERT INTO controversialusers
SELECT t.author
FROM
  (SELECT *
   FROM replycount) AS r
LEFT JOIN tbl_comment AS t ON t.id == r.id
LEFT JOIN score AS s ON t.id == s.id
GROUP BY t.author
HAVING avg(r.replies) >= 1
AND avg(r.replies) <= 1.1334
AND avg(t.controversiality) == 1;


INSERT INTO noncontroversialusers
SELECT t.author
FROM
  (SELECT *
   FROM replycount) AS r
LEFT JOIN tbl_comment AS t ON t.id == r.id
LEFT JOIN score AS s ON t.id == s.id
GROUP BY t.author
HAVING avg(r.replies) >= 1
AND avg(r.replies) <= 1.1334
AND avg(t.controversiality) == 0;
```

```
-- look at the distribution of comments for these groups

SELECT m.author,
       m.id,
       m.edited,
       length(m.cl_body),
       m.controversiality,
       s.ups,
       s.score_hidden,
       s.gilded,
       r.replies
FROM
  (SELECT *
   FROM controversialusers AS c
   LEFT JOIN tbl_comment AS t ON c.author == t.author) AS m
LEFT JOIN score AS s ON m.id == s.id
LEFT JOIN replycount AS r ON m.id == r.id;

SELECT m.author,
       m.id,
       m.edited,
       length(m.cl_body),
       m.controversiality,
       s.ups,
       s.score_hidden,
       s.gilded,
       r.replies
FROM
  (SELECT *
   FROM noncontroversialusers AS c
   LEFT JOIN tbl_comment AS t ON c.author == t.author) AS m
LEFT JOIN score AS s ON m.id == s.id
LEFT JOIN replycount AS r ON m.id == r.id;
```

Distribution of Comments of Controversial Users (5204 comments)

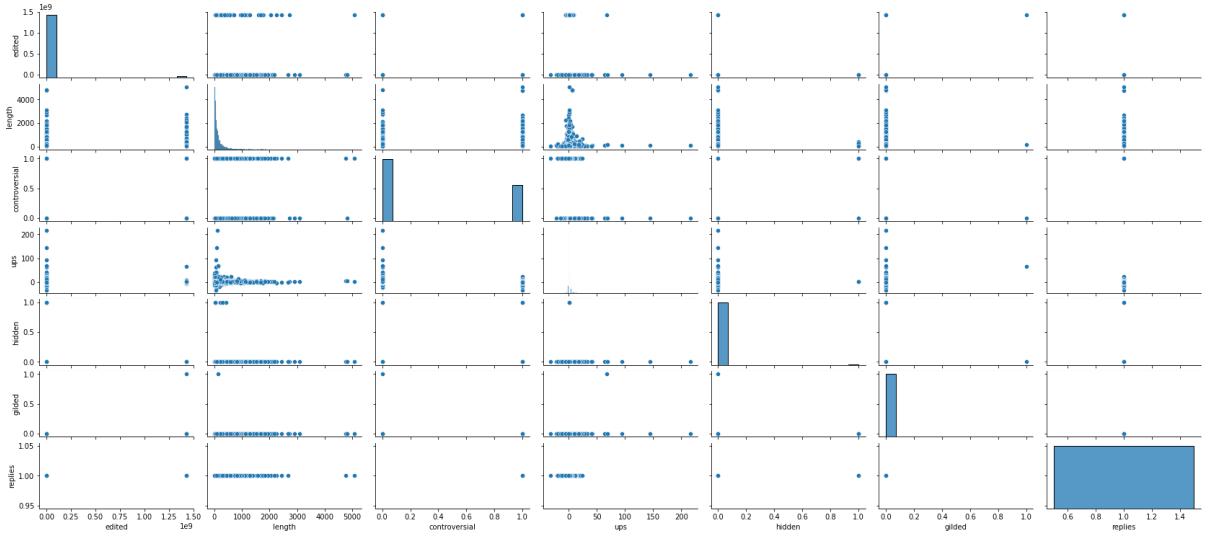


Figure 32: Pair plots of replies, gilded, hidden, upvotes, controversial comments, comment length and number of edited comments for comments coming from our subset of controversial users.

We see that the proportion of unedited to edited comments is very large, and unrelated to length. No comments exist which are unedited and also received gilded. Unedited comments have a larger spread of upvoted but this is likely due to the higher number of them.

We see that upvotes have a mean at 0 which makes sense for controversial comments. As we know from earlier, high controversy users are likely to also be users with single-digit number of comments. We can observe from the distribution graph of controversy that about one third of comments are controversial.

Length of comments does not seem to have any relationship with other variables. We observe a minimal proportion of both hidden and gilded comments.

Distribution of Comments of Non-Controversial Users (80627 comments)

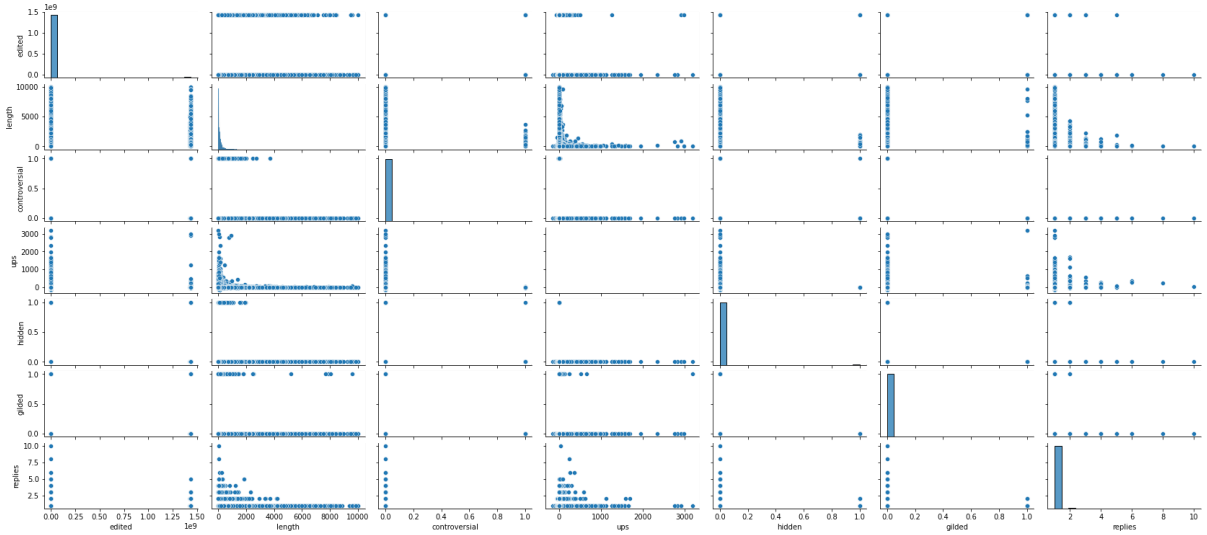


Figure 33 : Pair plots of replies, gilded, hidden, upvotes, controversial comments, comment length and number of edited comments for comments coming from our subset of non-controversial users.

We see that all variables except ups are extremely skewed to the left. For replies, there is a small number with two replies in contrast to controversial users who have zero comments which received two replies.

We also see that the number of replies to a comment decreases with length. The mechanism for this could be quite interesting. Are users not finishing reading the comment, or is there nothing more to add?

We can also see that lower length comments receive more upvotes. Similarly gilded are more often shorter.

Subsection of 5MB

We would like our subsection of the database to contain the comments and users that are most representative of the database. Here we briefly outline how we arrive at this subsection.

- We take the reply table that we used in Section II.2. to build the "`oursubset`" table.
- Here we include the controversial and non-controversial users from the representative subset as mentioned above.
- `Oursubset` contains the subset of the most interesting users. We save their `ids`
- We insert these `ids` into a new table called `fivmb`. We take only 4000 rows of each of the representative samples.

- We delete all `ids` from `tbl_comment` which do not appear in the new table we built, `fivmb`.
- To shrink it further we remove comments from `tbl_comment` which exceed 100 characters, these comments take up extra memory. We also remove `ids` of comments with very short comments.
- Finally, we use `VACUUM` which optimizes the database by offloading all the unused memory (the thing we just deleted). You must be careful with this because it overwrites the database you are currently working on, so it may be good to keep a backup.

```
-- MAKE MINI DATABASE

-- make the tables necessary to collect important features
CREATE TEMP TABLE replycount(id VARCHAR(255),
replies INTEGER);


INSERT INTO replycount
SELECT substr(parent_id, -7) AS id,
count(id) AS replies
FROM depends
WHERE parent_id not like 't3%'
GROUP BY parent_id
ORDER BY replies DESC;


CREATE TEMP TABLE controversialusers(author VARCHAR(255));
CREATE TEMP TABLE noncontroversialusers(author
VARCHAR(255));


INSERT INTO controversialusers
SELECT t.author
FROM
  (SELECT *
   FROM replycount) AS r
LEFT JOIN tbl_comment AS t ON t.id == r.id
LEFT JOIN score AS s ON t.id == s.id
GROUP BY t.author
HAVING avg(r.replies) >= 1
AND avg(r.replies) <= 1.1334
AND avg(t.controversiality) == 1;


INSERT INTO noncontroversialusers
SELECT t.author
FROM
```

```
  (SELECT *
   FROM replycount) AS r
LEFT JOIN tbl_comment AS t ON t.id == r.id
LEFT JOIN score AS s ON t.id == s.id
GROUP BY t.author
HAVING avg(r.replies) >= 1
AND avg(r.replies) <= 1.1334
AND avg(t.controversiality) == 0;


CREATE TEMP TABLE oursubset(id VARCHAR(255));


INSERT INTO oursubset
SELECT t.id
FROM
  (SELECT *
   FROM replycount
   WHERE replies > 16) AS r
LEFT JOIN tbl_comment AS t ON t.id == r.id
LEFT JOIN score AS s ON t.id == s.id
GROUP BY t.author
HAVING avg(controversiality) >= 0.3333;
-- create table which will contain ids we want to keep
CREATE TEMP TABLE fivmb(id VARCHAR(255));
-- add from the prev tables the ids to table oursubset
INSERT INTO fivmb
SELECT DISTINCT id
FROM replycount
WHERE replies > 19;

INSERT INTO fivmb
SELECT DISTINCT id
FROM oursubset;

INSERT INTO fivmb
SELECT DISTINCT id
FROM controversialusers as c
LEFT JOIN tbl_comment as t
ON t.author == c.author
limit 8000;

INSERT INTO fivmb
SELECT DISTINCT id
FROM noncontroversialusers as c
```

```
LEFT JOIN tbl_comment as t
ON t.author == c.author
limit 8000;

-- delete ids form table if theyre not in the subset we want
to keep
DELETE FROM tbl_comment
WHERE id NOT IN fivmb;

--remove overly long comments to save memory
DELETE FROM tbl_comment
WHERE length(cl_body) > 100;

--remove overly short comments which are not interesting
DELETE FROM tbl_comment
WHERE length(cl_body) < 45;

--DELETE FROM tbl_comment
--WHERE id NOT IN (SELECT DISTINCT id from tbl_comment limit
140);

--remove from all other tables based on tbl_comment
DELETE FROM is_distinguished
WHERE id NOT IN (SELECT id FROM tbl_comment);

DELETE FROM is_controversial
WHERE id NOT IN (SELECT id FROM tbl_comment);

DELETE FROM score
WHERE id NOT IN (SELECT id FROM tbl_comment);

DELETE FROM removed
WHERE id NOT IN (SELECT id FROM tbl_comment);

DELETE FROM depends
WHERE id NOT IN (SELECT id FROM tbl_comment);

DELETE FROM parent
WHERE substr(parent_id, -7) NOT IN (SELECT id FROM
tbl_comment);

DELETE FROM author
WHERE author NOT IN (SELECT author FROM tbl_comment);

-- drop the temp tables
```

```
DROP TABLE IF EXISTS controversialusers;
DROP TABLE IF EXISTS noncontroversialusers;
DROP TABLE IF EXISTS noncontroversialusers;
DROP TABLE IF EXISTS oursubset;
DROP TABLE IF EXISTS fivmb;
DROP TABLE IF EXISTS replycount;

-- !!!WARNING: OVERWRITE THE DATABASE THAT IS CURRENTLY
OPENED
-- optimizes memory by removing unused stuff
VACUUM;
```

Conclusion

As we approach the end of this project, we have seen the extensive use of both the data definition language and data manipulation language facets of SQL. We see how the flexibility of SQLite allows us to use it on Python for visualization purposes. Through several challenges of manipulating the data, we arrived at this comprehensive analysis of reddit users based on their several characteristics, like comment length, score and controversiality to name a few.

As far as our investigation of the users and comments are concerned, the most surprising result of our inquiry was the high number of outlier users at controversy $= 1$. However, it can easily be explained by their choice not to comment again after this first comment.

There is in fact a small group of users who have generated both high-reply comments and a controversial comment. But this group is quite small at only 68 users. It is likely this investigation could yield more insightful results given analysis using a lower cutoff.

Another interesting comparison was that for our high-reply controversial subset, upvotes stayed positive but for the low-reply (representative) controversial subset, average upvotes were normally dispersed around 0. This may show that, in fact, Reddit upvotes and replies are highly correlated and thus for the most part use of upvotes is not used to express an opinion. And in this way Reddit does not increase engagement through controversial content.