Report Data Base - Chérif YAKER & Lucas MONTEIRO - M2 STAT-ECO - TSE

For this project, we have decided to use SQLite since it is a light, pratical and fast database tool. Moreover, we have made the project only through the command line since it is fast, simple and we didn't want to be constraint by the heavyness of a software such as Oracle. Note that we have made the project on Linux Fedora (Workstation 37).

The project is organized as follow :

- In the first part, we set the environment and prepare all the files. We create the database and the database's schema (no data are loaded).
- In the second part, we describe the changes that have been done on data before loading them and we then load all the cleaned data.
- In the third part, we only execute few queries to give some indications about the size of each table. This part is quite short since we quickly start the analysis in the fourth part
- In the fourth part, we analyze the content of the database through several dimensions : score, comment, controversiality, authorship
- Finally in the fifth part, we extract a subset of the database.

We have tried to be explicit on each stage of the project, in order to be as much reproductible as possible.

Part 1 : Database creation

Setup and preparation

The project can be found in the git repository https://github.com/lucasmonteiro4/data-base.git

The architecture of the project is as follow :

```
data_base/
    - data/
        - raw/
        - cleaned/
        - data_dictionary
    - logic/
        - sql_scripts/
        - python_scripts/
    - doc/
        - report.md
    project.db
    README.me
```

data/ folder contains all original data that are going to be use for the project.

data/raw/ contains all the raw data (directly extracted from the source, see below for details).

data/cleaned/ contains data that have been preprocessed, these data are the one that are actually used.

The original data are not in the repo since it is too heavy (> 1GB) to store it in the GitHub repository. Hence, raw data have to be placed in the folder **data/raw/**.

For information, **raw/** folder contains : - askreddit_author.csv - askreddit_distinguihshed.csv - askreddit_controverse.csv - askreddit_parent.csv - askreddit_removal.csv - askreddit_comment.csv - askreddit_score.csv - askreddit_depends.csv - askreddit_is_distinguihshed.csv

cleaned/ is empty, it is normal it will host the data once they are preprocessed (also too heavy, to get them : follow instruction in Part 2 and Python scripts).

logic/ contains all the code that we execute(SQL for database, Python for preprocessing).

doc/ contains all the necessary documentation to understand the project (*e.g.* this report).

Once we have detailed the content of each folder, we can now clone the git repository and we select it as current directory.

One can also separatly recover **data/**, **logic/** and **doc/** and put them in a folder named **data_base**.

```
$ git clone https://github.com/lucasmonteiro4/data-base.git data_base
$ cd data_base
```

Once the repository is initialized, we have to put all the raw data into the **raw/** folder. Once this is done, we can create the database schema (we create the database after having done all the cleaning (*i.e.* at the end of Part 2), we do this in order to avoid to enter into SQlite, then quit it to execute Python files then again enter into SQlite. We first execute Python scripts in shell and then we enter into SQlite, create the database etc., it is more convenient).

Database creation

To create the schema of the database (that we will name *project.db*), we write a SQl script that we will execute and that will create all the tables. This script, named *script_part1.sql*, is in the folder **logic/**.

Here is an example for the table **author_**. Through all the project, we indicate table names with an underscore (this is done to avoid problems with author as table name and author as column name).

First we use `drop table if exists` statements to avoid the problem of rewriting tables. To drop tables, we start from the ones with no foreign keys (no

reference) to the ones with foreign keys.

```
drop table if exists removed_ ;
drop table if exists depends_ ;
drop table if exists is_distinguished_ ;
drop table if exists score_ ;
drop table if exists comment_ ;
drop table if exists subreddit_ ;
drop table if exists parent_ ;
drop table if exists removal_ ;
drop table if exists controversy_ ;
drop table if exists distinguished_ ;
drop table if exists author_ ;
```

We now create each table.

Auhor

```
create table author_ (
    author TEXT
    , PRIMARY KEY (author)
);
```

DISTINGUISHED

```
create table distinguished_ (
    distinguished TEXT
    , PRIMARY KEY (distinguished)
);
```

CONTROVERSY

```
create table controversy_ (
    controversiality
    , PRIMARY KEY (controversiality)
);
```

REMOVAL

```
create table removal_ (
    removal_reason TEXT
    , PRIMARY KEY (removal_reason)
);
```

PARENT

```
create table parent_ (
```

```
    parent_id TEXT
    , link_id TEXT
    , PRIMARY KEY (parent_id)
);
```

SUBREDDIT

```
create table subreddit_ (
    subreddit_id TEXT
    , subreddit TEXT
    , PRIMARY KEY (subreddit_id)
);
```

COMMENT

We choose the following relationship : comment_ : $id and score_ : (id#), so comment_ as to be defined first and score_ second

```
create table comment_ (
    id TEXT
    , created_utc INTEGER
    , name TEXT
    , body TEXT
    , edited INTEGER
    , author_flair_css_class TEXT
    , author_flair_text TEXT
    , author TEXT
    , controversiality INTEGER
    , subreddit_id TEXT

    , PRIMARY KEY (id)
    , FOREIGN KEY (author) REFERENCES author_ (author)
    , FOREIGN KEY (controversiality) REFERENCES controversy_ (controversiality)
    , FOREIGN KEY (subreddit_id) REFERENCES subreddit_ (subreddit_id)
);
```

SCORE

```
create table score_ (
    id TEXT
    , score INTEGER
    , ups INTEGER
    , downs INTEGER

    -- no native boolean type : see https://www.sqlite.org/datatype3.html > 2.1 Boolean Dat
    , score_hidden
    , gilded INTEGER
```

```
    , PRIMARY KEY (id)
    , FOREIGN KEY (id) REFERENCES comment_(id)
);
```

IS DISTINGUISHED

```
create table is_distinguished_ (
    id TEXT
    , distinguished TEXT

    , PRIMARY KEY (id, distinguished)
    , FOREIGN KEY (id) REFERENCES comment_ (id)
    , FOREIGN KEY (distinguished) REFERENCES distinguished_ (distinguished)
);
```

REMOVED

```
create table removed_ (
    id TEXT
    , removal_reason TEXT

    , PRIMARY KEY (id, removal_reason)
    , FOREIGN KEY (id) REFERENCES comment_ (id)
    , FOREIGN KEY (removal_reason) REFERENCES removal_ (removal_reason)
);
```

DEPENDS

```
create table depends_ (
    id TEXT
    , parent_id TEXT

    , PRIMARY KEY (id, parent_id)
    , FOREIGN KEY (id) REFERENCES comment_ (id)
    , FOREIGN KEY (parent_id) REFERENCES parent_ (parent_id)
);
```

This script will be executed in SQLite after all the cleaning. So now we stay in the shell and we still don't use SQlite.

Part 2

Preprocessing

We operate some changes on the data to make some cleaning and remove headers. We execute 2 python scripts.

The first one concerns the cleaning of the file *askreddit_comment.csv*. In the field body, there are many line breaks and they have to be removed to allows SQLite to read *askreddit_comment.csv* properly, if we don't do it, SQlite will be unable to detect the 10 columns that it is waiting for (at the line break in the field *body*, it will consider it as a new row, so it will only detect 4 columns (*id*, *created_utc*, *name*, *body*) instead of the ten ones). So in the script, we take *askreddit_comment.csv*, put it as an *.txt* file (to remove linebreaks and properly treat it as a unique (very) long string), we manually create the line break after the field *subreddit_id* since it is unique and always present and finally write it as a *.csv* file in the folder **data/raw/**.

This script is in logic/python_scripts/

```
data_base/
    - data/
    - logic/
        - script.sql
        - python_scripts/
            - clean_comment.py
    - doc/
```

We execute this script (at the **data_base** level) :

```
$ python3 logic/python_scripts/clean_comment.py
```

The second python script concerns the removing to the header. When we have created the schema in the Part 1, we have created names for the columns. Hence, when we will import the data, the first row of each table will be the header in the .csv, we have to remove. (It surely exists another method, more efficient, but we did not find it.)

Here is the code to clean *author.csv* :

```python
# AUTHOR data
# open the original file
with open("data/raw/askreddit_author.csv", "r") as f:

    # prepare to write the new clean file without header into the cleaned/ folder
    with open("data/cleaned/clean_author.csv", "w") as g:

        # do not consider the first row of the .csv when writing the new one
        next(f)

        # write all the following rows of the original .csv into the new clean .csv
        for line in f:
            g.write(line)
```

All the remaining code can be found in the appendice.

This script is in logic/python_scripts/

```
data_base/
    - data/
    - logic/
        - script.sql
        - python_scripts/
            - clean_headers.py
    - doc/
```

To execute this code, we create the folder **cleaned/** into **data/** (to host the cleaned data) and run the script.

```
$ cd data
$ mkdir cleaned
$ cd -
```

Then at **data_base** level we execute the python script :

```
$ python3 logic/python_scripts/clean_headers.py
```

Hence, we have prepared data to have a nice import into the database.

Finally we can open SQLite and execute the SQL file *script_part1.sql* (in the folder **logic/**) to create the schema and the tables.

We first execute sqlite3 and create the database named `"project.db"`

```
$ sqlite3 "project.db"
```

One this is done, we read the SQL script we have mentioned in the Part 1.

```
sqlite> .read logic/sql_scripts/script_part1.sql
```

Then we can check that all the tables have been created.

```
sqlite> .tables
```

```
author_            depends_           parent_            score_
comment_           distinguished_     removal_           subreddit_
controversy_       is_distinguished_  removed_
```

Database loading

We now load data into the database. To do so, we first indicate the type of data we are going to load (*csv*), we specify the separator and finally we import all cleaned data from **data/cleaned/** folder.

To do so, we have created a SQL script named *script_part2.sql* in the folder **logic/**, which load all the cleaned data into its respective tables.

```
sqlite> .read logic/sql_scripts/script_part2.sql
```

The content of *script_part2.sql* is as follow :

```
.mode csv
.separator ","

.import data/cleaned/clean_author.csv author_
.import data/cleaned/clean_distinguihshed.csv distinguihshed_
.import data/cleaned/clean_controverse.csv controversy_
.import data/cleaned/clean_removal.csv removal_
.import data/cleaned/clean_parent.csv parent_
.import data/cleaned/clean_subreddit.csv subreddit_
.import data/cleaned/clean_comment.csv comment_
.import data/cleaned/clean_score.csv score_
.import data/cleaned/clean_is_dist.csv is_distinguished_
.import data/cleaned/clean_depends.csv depends_

.headers on
```

We specify the nature of files we load, the separator of csv and we load all cleaned data into their respective tables.

Hence, we have load data into the freshly created database named *project.db* and we are ready to analyze its content.

Index creation

Part 3

In this part, we only execute few queries to have an idea of the size of each table.

To explore the partitioning of the database (*i.e.* the size that each table takes), we use a wonderful tool called sqlite3_analyzer (see https://www.sqlite.org/sqlanalyze.html for more details) which gives information about size, memory etc.

We simply run, at the **data_base/** folder level :

```
$ sqlite3_analyzer "project.db"
```

For instance, here is an extract of the long command response, for the table comment_

```
*** Table COMMENT_ w/o any indices ******************************************

Percentage of total database.....  55.1%
Number of entries.............. 4234970
Bytes of storage consumed..... 915931136
Bytes of payload.............. 827764673   90.4%
Bytes of metadata.............. 33362487     3.6%
B-tree depth..... 4
Average payload per entry..... 195.46
Average unused bytes per entry..... 12.94
```

```
Average metadata per entry..... 7.88
Average fanout..... 376.00
Non-sequential pages..... 18743          8.4%
Maximum payload per entry..... 16935
Entries that use overflow..... 1614          0.038%
Index pages used..... 589
Primary pages used..... 221257
Overflow pages used..... 1770
Total pages used..... 223616
Unused bytes on index pages..... 305523       12.7%
Unused bytes on primary pages..... 54341897      6.0%
Unused bytes on overflow pages..... 157180       2.2%
Unused bytes on all pages..... 54804600      6.0%
```

So we have really detailed information about each table (althought we only need the 2 first ones).

We start the explanatory analysis with the table *comment_* since it is the core table of the database.

```
select count(*) from comment_;  -- number of comment_ rows : 4 234 970
```

We have 4 234 970 rows, which correspond to 4 234 970 comments in the subreddit AskReddit. This confirms the good loading of data.

Another interesting fact is the number of authors/users

```
select count(*) from author_;   -- number of author_ rows : 570 735
```

We have 570 735 unique users in this subreddit AskReddit.

```
select count(*) from parent_;   -- number of parent_ rows : 1 464 558
```

```
select count(*) from score_;                -- number of score_ rows 4 234 970
select count(*) from depends_;              -- number of depends_ rows : 4 234 970
select count(*) from is_distinguished_;     -- number of is_distinguished_ rows : 4 234 970
```

For *score_*, *depends_* and *is_distinguished_*, we have 4 234 970 rows, like *comment_*, so each comment is properly linked to its score, its parent etc.

We can now move to a slighlty deeper analysis. For instance, we could be interested by the maximal score, the more recent comment etc.

```
select max(score) from score_;  -- maximal score : 6761
```

The maximal score attributed to a comment is 6 761.

We want to know the more recent comment (Table 3.1) :

```
select id
        , date(max(created_utc), 'unixepoch') as max_date
        , body
from comment_;
```

Table 3.1	id	max_date	body
	crrbeop	2015-05-31	classy way to put it

And we also want to know the older comment (Table 3.2) :

```
select id
        , date(min(created_utc), 'unixepoch') as min_date
        , body
from comment_;
```

Table 3.2 id	min_date	body
cqug90j	2015-05-01	No one has a European accent either because it doesn't exist. There are accents from Europe but not a European accent.

One interesting fact is that all comments are in a range of 1 month (May 2015).

The aspect of controversiality is interesting. One first analysis could be to know the number of controversial posts and the number of non-controversial ones (Table 3.3):

```
select count(*) as nb_posts
from comment_ c
group by c.controversiality;
```

Table 3.3		nb_posts
	controversiality = 0	4182752
	controversiality = 1	52218

We clearly notice that controversial posts represent 1.2% of the overall comments.

Part 4

In the third part, we have explored some simple characteristics about the database. Now we move to a deeper analysis. In this part, we explore score, comment, author and controversiality.

We first examine which comment has the highest score (6 761, found in Part 3), Table 4.1 :

```
select c.id
        , date(c.created_utc, 'unixepoch') as date_comment
        , c.body
        , s_max.max_score
from comment_ c, (select id as id_score
                        , max(score) as max_score
                    from score_) s_max
where s_max.id_score = c.id;
```

Table 4.1 id	date_comment	body	max_score
cr56nez	2015-05-11	Then you got yourself a one night standoff.	6761

We want to highlight authors and score. To do so, we analyze authors who have posted the most (Table 4.2) : Direct

```
-- direct way : specifying  [deleted] and AutoModerator

select a_c.*
from (select author
            , count(author) as count_author
        from comment_
        where author not in ('[deleted]', 'AutoModerator')
        group by author) a_c
order by a_c.count_author desc
limit 5;
```

Table 4.2	author	count_author
	Late_Night_Grumbler	8298
	BiagioLargo	5843
	--Equinox666--	2989
	KubrickIsMyCopilot	2601
	Megaross	2479

```
--indirect way : without specifying [deleted] and AutoModerator

select a_c.*
from (select author
            , count(author) as count_author
        from comment_
        where author not in (select a_c.author
                                from (select author
                                            , count(author) as count_author
```

```
                                        from comment_
                                        group by author) a_c
                                order by a_c.count_author desc limit 2)
        group by author) a_c
order by a_c.count_author desc
limit 5;
```

Table 4.2	author	count_author
	Late_Night_Grumbler	8298
	BiagioLargo	5843
	--*Equinox666*--	2989
	KubrickIsMyCopilot	2601
	Megaross	2479

The two queries above do the same thing : the first one is quicker but the second one does not need to specify explicitly the names, which is always preferable. The indirect way works as follows : since AutoModerator and [deleted] are by far the most active user (top 2), then we select the authors that are not in the top 2.

Once we have the most active users, we are interested by the score of the one who is the most active. Our question is : this user post a lot but do these posts are important ?

To do so, we take the previous query but we only extract the most active user. Then, we compute the average score for this user (Table 4.3). The following query use the indirect way : it is a bit heavy to write and compute but it is automatic.

```
select round(avg(s.score),2)
        , c.author
from comment_ c,
(select id as id_score
        , score
    from score_) s
where c.id = s.id_score
and c.author in (select a_c.author
                    from (select author
                                , count(author) as count_author
                            from comment_
                            where author not in (select a_c.author
                                                    from (select author
                                                                , count(author) as count_aut
                                                            from comment_
                                                            group by author) a_c
                                                    order by a_c.count_author desc limit 2)
```

```
                        group by author) a_c
                    order by a_c.count_author desc limit 1)
;
```

Table 4.3	avg_score	author
	4.75	Late_Night_Grumbler

The previous analysis started with top authors and explore their scores. Now we do the inverse : we analysis the authors with highest score and then analysis their comments.

First, we examine the 5 users that have the highest scores (Table 4.4).

```
select author, avg(s.score) as avg_score
from comment_ c, (select id as id_score, score
                  from score_) s
where c.id = s.id_score
group by author
order by avg_score desc
limit 5;
```

Table 4.4	author	avg_score
	lenaeca	5383.0
	CCorinne	4836.0
	The0isaZero	4834.0
	4eyedoracle	4815.0
	planetoiletsscareme	4755.0

Their average score ends with .0, so we suspect that their comment is unique (an average ending by .0 is quite special).

To confirm our intuition, we extract the comment of the one who has the highest average score (Table 4.5).

```
select c.id, c.body, c.author, s.score
from comment_ c, score_ s
where c.id = s.id
and author in (select ha.author
                from (select author, avg(s.score) as avg_score
                        from comment_ c, (select id as id_score, score from score_) s
                        where c.id = s.id_score
                        group by author
                        order by avg_score desc
                        limit 1) ha);
```

Table 4.5	id	body	author	score
	crcgqnq	“You get really nervous, but you shouldn’t be, because you teach statistics and no one really cares about statistics.” This was after my first semester as a Graduate teaching assistant.	lenaeca	5383

Indeed, the comment is unique. This analysis is interesting since we have been able to put some light on different aspect : the author who posts the most is not particularly important (which seem logic since quantity does not imply quality), the one who has the highest score never publish (only one comment) and the highest score (see part 3) belongs to someone who have published several times (since 6167 is not in the highest average score).

Controversiality

We now continue our analysis through the controversiality aspect, initialized in the Part 3.

To have an idea of what can be a controversial psot, we display 3 posts of that category (Table 4.6).

```
select *
from comment_
where controversiality == 1
limit 3;
```

Table 4.6	id	body	author	controversiality
	cqug921	I honestly wouldn't have believed it if I didn't live it. She made his life hell and I had a front row seat. I'm just glad I had a front row seat to her confession and firing. :)	youthfulvictim	1
	cqug94y	The implications of that varies between cultures. Don't be racist.	robondes	1
	cqug95f	You're just trying to get to the front page. I see through your facade!!!	JustMe80	1

We observe that this kind of posts implies mean posts, with bad words about others.

Social medias (such as Reddit) is often mentioned in the news to be a place where anyone can say anything and where bad words and insults are everywhere and that people share a lot this kind of posts. To explore this idea, we compare average score of controversial posts against non-controversial ones (Table 4.7).

```
select round(avg(s.score),2) as average_score
from comment_ c, (select id, score from score_) s
where c.id = s.id
group by c.controversiality;
```

Table 4.7		average_score
	controversiality = 0	12.75
	controversiality = 1	0.86

It is clear that the controversial post are not, in average, shared a lot.

We now want to mix our 2 approaches : the former with authors and the latter with controversiality. To do so, we examine the top 5 authors who are the most controversial, meaning they have the most posted controversial posts (Table 4.8).

```
select author, count(*) as count_author
from comment_
where controversiality==1
group by author
order by count_author desc
limit 5;
```

Table 4.8	author	count_author
	[deleted]	4685
	CorDeFerrum	95
	AutoModerator	92
	Megaross	60
	berke_k	58

To get rid off of *[deleted]* and *AutoModerator*, we will use the direct way as mentioned ealier (note that it is surprising to find *AutoModerator* in the controversial post, an explanation could be that *AutoModerator* posts many times and that some users could signal it as controversial; more generally we find also find *Megaross* and *KubrickIsMyCopilot*, who are also in the group of users who have the most posted) (Table 4.9):

```
select author, count(*) as count_author
from comment_
where controversiality==1
    and author not in ('[deleted]', 'AutoModerator')
group by author
order by count_author desc
limit 5;
```

Table 4.9	author	count_author
	CorDeFerrum	95
	Megaross	60

Table 4.9	author	count_author
	berke_k	58
	KubrickIsMyCopilot	57
	--Equinox666--	49

We now display some controversial posts of the *CorDeFerrum*, the one who have posted the more controversial posts (Table 4.10) :

```
select body, author
from comment_
where author like 'CorDeFerrum' and controversiality == 1
limit 3;
```

Table 4.10	body	author
	Honestly I can't tell if someone is attractive when they have dark skin, I feel like I can't "see" them. It's hard to explain.	CorDeFerrum
	Stop giving foreign aid and focus on poverty in our own country. (USA)	CorDeFerrum
	Don't give me advice. No one qualifies to give me advice.	CorDeFerrum

Part 5

To reduce the database, we operate in 3 steps : 1. Create a copy of **project.db**, named **reduced.db** 2. Randomly select 1000 rows in *comment_* 3. Recover the associated values (to reduced.comment_) in all the other tables

We execute the file *script_part5* in **logic/sql_scripts/**

```
.read logic/sql_scripts/script_part5.sql
```

This will create a subset database *reduced.db* of the initial complete database *project.db*.

The procedure in the script is as follow :

- Create an copy named reduced.db, but with empty tables.

```
sqlite> attach 'reduced.db' as red_db;
```

- In the new database reduced.db, add 1000 random rows from the original project.db

```
sqlite> insert into red_db.comment_
   ...> select * from comment_ where id in (select id from comment_ order by random() limit
```

- Once we have the core *comment_*, we add the other tables, based on *comment_*.

Score

```
sqlite> insert into red_db.score_
   ...> select * from score_ where id in (select id from red_db.comment_);
```

Is Distinguished

```
sqlite> insert into red_db.is_distinguished_
   ...> select * from is_distinguished_ where id in (select id from red_db.comment_);
```

Author

```
sqlite> insert into red_db.score_
   ...> select * from score_ where id in (select id from red_db.comment_);
```

Distinguished

```
sqlite> insert into red_db2.distinguished_
   ...> select * from distinguished_;
```

Controversy

```
sqlite> insert into red_db.controversy_
   ...> select * from controversy_;
```

Subreddit

```
sqlite> insert into red_db.distinguished_
   ...> select * from distinguished_;
```

Documentation and resources

Appendice

Python script to clean *askreddit_comment.csv* of line breaks problem :

```
import csv

# open comment.csv
with open('data/raw/askreddit_comment.csv', 'r') as f:
```

```
    content = f.read()

# change format to .txt (to apply string method with python), this will remove all line bre
with open('data/raw/askreddit_comment.txt', 'w') as f:
    f.write(content)

# open the comment.txt
with open('data/raw/askreddit_comment.txt', 'r') as f:
    txt_content = f.read()

# we constraint the line break to be after the subreddit_id
# subreddit_id is unique since the project is about one subreddit (AskReddit)
# hence we can use this unique string (subreddit_id) to enforce the line break
clean_content = txt_content.replace('subreddit_id ', 'subreddit_id \n')
clean_content = clean_content.replace('t5_2qh1i ', 't5_2qh1i \n')

# we finally write the clean file into a csv file into data/raw/
with open('data/raw/worked_comment.csv', 'w') as f:
    f.write(clean_content)
```

Python script to clean headers of each *.csv* :

```
# AUTHOR data
# open the original file
with open("data/raw/askreddit_author.csv", "r") as f:

    # prepare to write the new clean file without header into the cleaned/ folder
    with open("data/cleaned/clean_author.csv", "w") as g:

        # do not consider the first row of the .csv when writing the new one
        next(f)

        # write all the following rows of the original .csv into the new clean .csv
        for line in f:
            g.write(line)



# DISTINGUISHED data
with open("data/raw/askreddit_distinguihshed.csv", "r") as f:
    with open("data/cleaned/clean_distinguihshed.csv", "w") as g:
        next(f)
        for line in f:
            g.write(line)

# CONTROVERSE data
with open("data/raw/askreddit_controverse.csv", "r") as f:
```

```
    with open("data/cleaned/clean_controverse.csv", "w") as g:
        next(f)
        for line in f:
            g.write(line)

# PARENT data
with open("data/raw/askreddit_parent.csv", "r") as f:
    with open("data/cleaned/clean_parent.csv", "w") as g:
        next(f)
        for line in f:
            g.write(line)

# REMOVAL data
with open("data/raw/askreddit_removal.csv", "r") as f:
    with open("data/cleaned/clean_removal.csv", "w") as g:
        next(f)
        for line in f:
            g.write(line)

# COMMENT data
with open("data/raw/worked_comment.csv", "r") as f:
    with open("data/cleaned/clean_comment.csv", "w") as g:
        next(f)
        next(f) #for comment we also next the second line since it is blank
        for line in f:
            g.write(line)

# SCORE data
with open("data/raw/askreddit_score.csv", "r") as f:
    with open("data/cleaned/clean_score.csv", "w") as g:
        next(f)
        for line in f:
            g.write(line)

# DEPENDS data
with open("data/raw/askreddit_depends.csv", "r") as f:
    with open("data/cleaned/clean_depends.csv", "w") as g:
        next(f)
        for line in f:
            g.write(line)

# IS DISTINGUISHED data
with open("data/raw/askreddit_is_distinguihshed.csv", "r") as f:
    with open("data/cleaned/clean_is_dist.csv", "w") as g:
        next(f)
        for line in f:
```

```
            g.write(line)

# SUBREDDIT data
with open("data/raw/askreddit_subreddit.csv", "r") as f:
    with open("data/cleaned/clean_subreddit.csv", "w") as g:
        next(f)
        for line in f:
            g.write(line)
```