Reddit Data Project Final Report

Shaun Saini, Avik Samanta , Faith A Dennis , Bach Xuan Tran

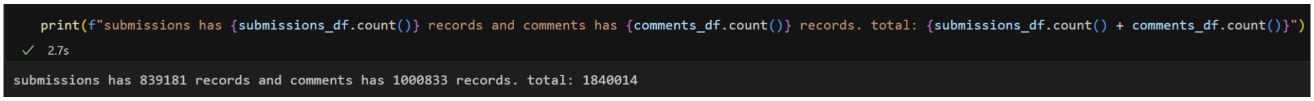
Dataset Selection

The dataset selected for this project is a subset of Reddit data from 2015-2023 across the top 40,000 subreddits available online for download. We selected Reddit data because we are interested in learning more about building a machine learning data pipeline, and we felt this data was more suitable for a wide range of ML uses. For example, Reddit data captures more complexity in language than Yelp data making it more generalizable to a handful of NLP tasks. There are TBs of free Reddit data available for training and building models, and Reddit offers a free API that can be used to later expand our project to include real time data extraction to get us close to streaming.

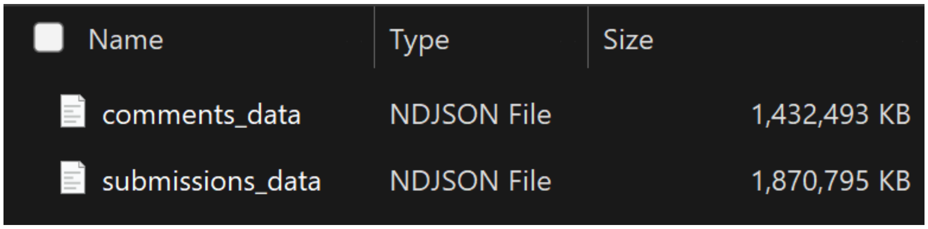
We loaded roughly 2-3GB of data by process of downloading compressed data organized into folders and unzipping them into NDJSONs. The entire dataset was organized into folders by subreddit. Due to the size of the entire dataset available to us, we were unable to load the entire set and sample from it. Further, we found that some of the subreddits were NSFW, so we selected a subset of the subreddits to download that we believed were more appropriate, such as the "stocks" and "psychology" subreddits. The data came organized into submissions and comments. Data from different subreddits were aggregated into the same file during the unzipping process via a python script. Below are screenshots showing the record numbers and the NDJSON sizes of the submission data and comment data. The total # of records ended up being 1.8M. KB sizes combined are >1GB.

We were unable to upload the compressed data into the repository due to the size and Github upload limits, so data was shared via drive. A link to the data we used is provided in the data readme in the repository. The original data source is available at: https://academictorrents.com/details/56aa49f9653ba545f48df2e33679f014d2829c10

Screenshots verifying data requirements:



```
print(f"submissions has {submissions_df.count()} records and comments has {comments_df.count()} records. total: {submissions_df.count() + comments_df.count()}")
✓ 2.7s
submissions has 839181 records and comments has 1000833 records. total: 1840014
```



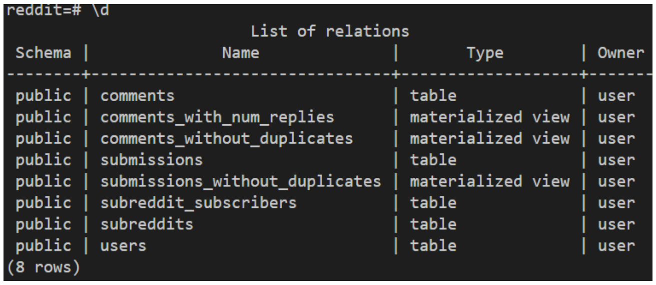
Name	Type	Size
comments_data	NDJSON File	1,432,493 KB
submissions_data	NDJSON File	1,870,795 KB

Database Structure

- Show the PostgreSQL database schema

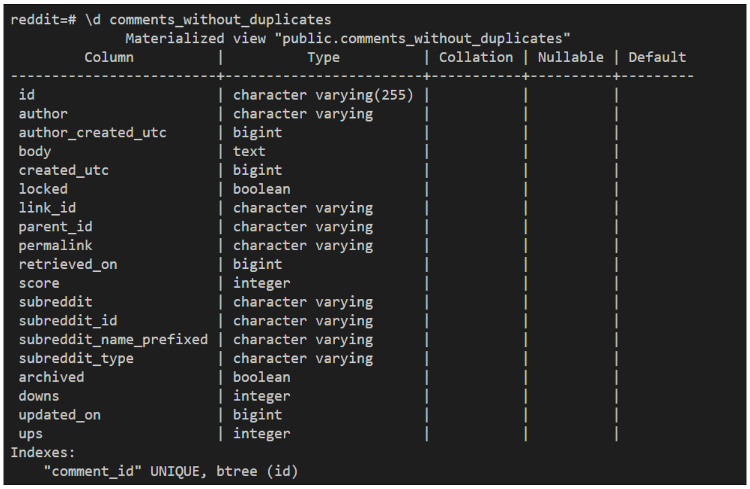
– For each table show the result of running \d [table_name] in a psql console.

Below we have the schema for the entire database, as we can see the underlying tables comments and submissions have duplicate values when uploading the json to the psql server, which is why created materialized views comments_without_duplicates and submissions_without_duplicates, so that we could avoid pkey constraint issues and instead create indexes on primary keys.



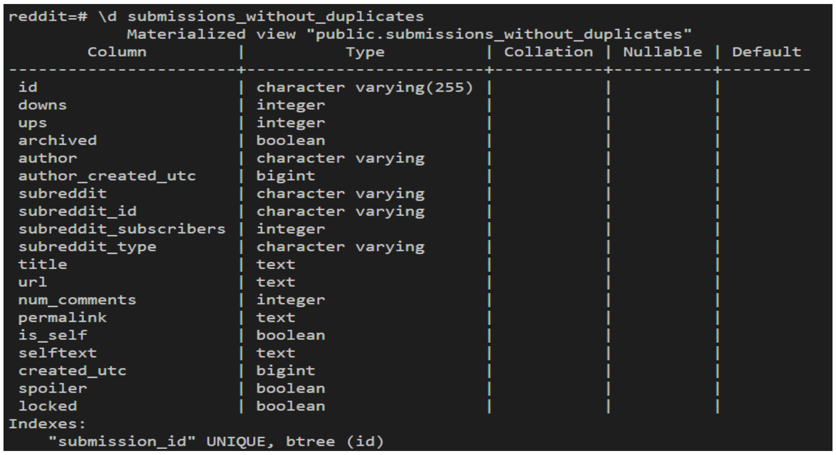
```
reddit=# \d
                          List of relations
 Schema |              Name              |       Type        | Owner
--------+--------------------------------+-------------------+-------
 public | comments                       | table             | user
 public | comments_with_num_replies      | materialized view | user
 public | comments_without_duplicates    | materialized view | user
 public | submissions                    | table             | user
 public | submissions_without_duplicates | materialized view | user
 public | subreddit_subscribers          | table             | user
 public | subreddits                     | table             | user
 public | users                          | table             | user
(8 rows)
```

Below is the comments_without_duplicates materialized view, note that we have a unique index on id which is essentially a pkey constraint.



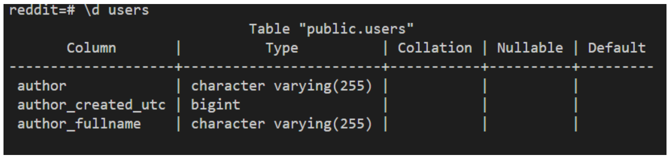
```
reddit=# \d comments_without_duplicates
             Materialized view "public.comments_without_duplicates"
         Column          |          Type          | Collation | Nullable | Default
-------------------------+------------------------+-----------+----------+---------
 id                      | character varying(255) |           |          |
 author                  | character varying      |           |          |
 author_created_utc      | bigint                 |           |          |
 body                    | text                   |           |          |
 created_utc             | bigint                 |           |          |
 locked                  | boolean                |           |          |
 link_id                 | character varying      |           |          |
 parent_id               | character varying      |           |          |
 permalink               | character varying      |           |          |
 retrieved_on            | bigint                 |           |          |
 score                   | integer                |           |          |
 subreddit               | character varying      |           |          |
 subreddit_id            | character varying      |           |          |
 subreddit_name_prefixed | character varying      |           |          |
 subreddit_type          | character varying      |           |          |
 archived                | boolean                |           |          |
 downs                   | integer                |           |          |
 updated_on              | bigint                 |           |          |
 ups                     | integer                |           |          |
Indexes:
    "comment_id" UNIQUE, btree (id)
```

Below is the submissions_without_duplicates materialized view, which has a unique index on id as a pkey constraint.



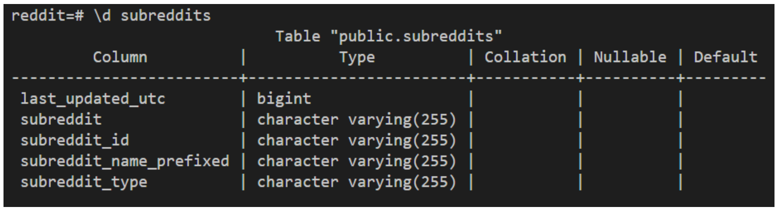
```
reddit=# \d submissions_without_duplicates
            Materialized view "public.submissions_without_duplicates"
        Column         |          Type          | Collation | Nullable | Default
-----------------------+------------------------+-----------+----------+---------
 id                    | character varying(255) |           |          |
 downs                 | integer                |           |          |
 ups                   | integer                |           |          |
 archived              | boolean                |           |          |
 author                | character varying      |           |          |
 author_created_utc    | bigint                 |           |          |
 subreddit             | character varying      |           |          |
 subreddit_id          | character varying      |           |          |
 subreddit_subscribers | integer                |           |          |
 subreddit_type        | character varying      |           |          |
 title                 | text                   |           |          |
 url                   | text                   |           |          |
 num_comments          | integer                |           |          |
 permalink             | text                   |           |          |
 is_self               | boolean                |           |          |
 selftext              | text                   |           |          |
 created_utc           | bigint                 |           |          |
 spoiler               | boolean                |           |          |
 locked                | boolean                |           |          |
Indexes:
    "submission_id" UNIQUE, btree (id)
```

Below is the users table:



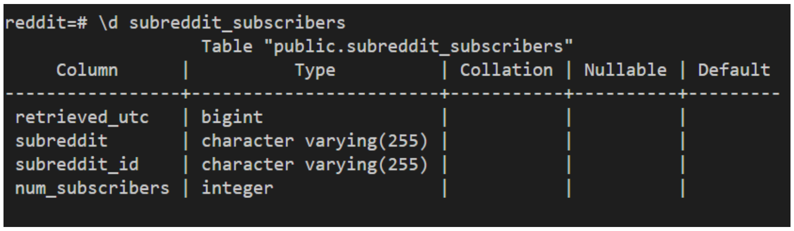
```
reddit=# \d users
                          Table "public.users"
      Column       |          Type          | Collation | Nullable | Default
-------------------+------------------------+-----------+----------+---------
 author            | character varying(255) |           |          |
 author_created_utc | bigint                 |           |          |
 author_fullname   | character varying(255) |           |          |
```

Below is the subreddits table:



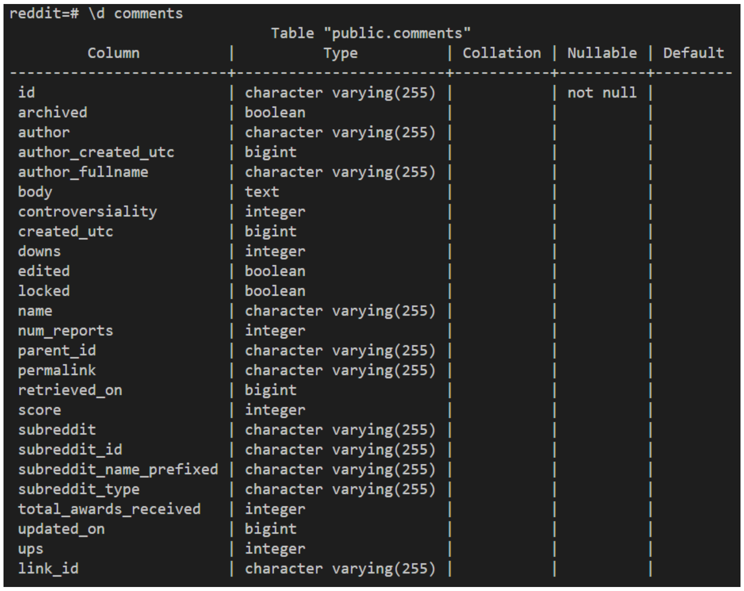
```
reddit=# \d subreddits
                             Table "public.subreddits"
         Column          |          Type          | Collation | Nullable | Default
-------------------------+------------------------+-----------+----------+---------
 last_updated_utc        | bigint                 |           |          |
 subreddit               | character varying(255) |           |          |
 subreddit_id            | character varying(255) |           |          |
 subreddit_name_prefixed | character varying(255) |           |          |
 subreddit_type          | character varying(255) |           |          |
```

Below is the subreddit_subscribers_table:



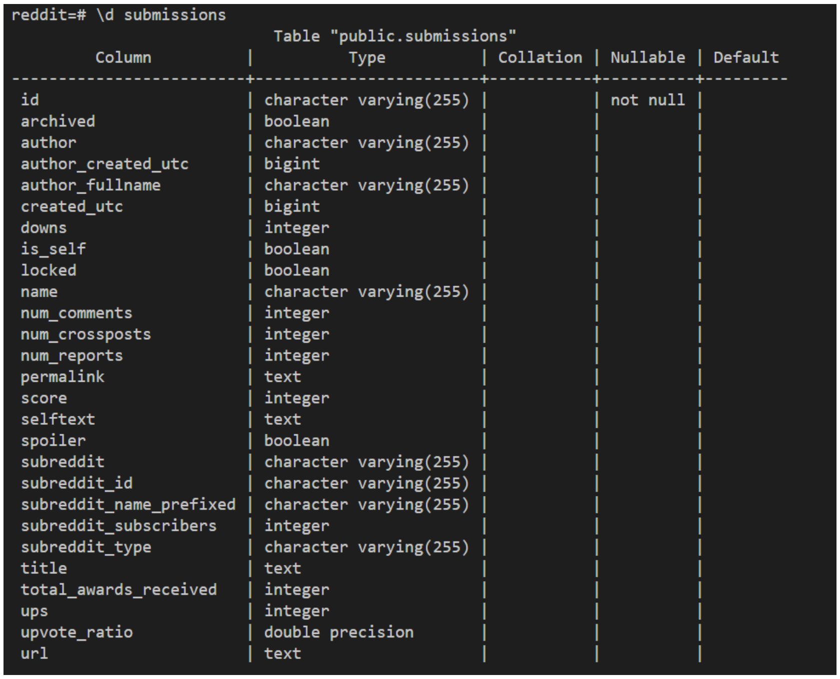
```
reddit=# \d subreddit_subscribers
                  Table "public.subreddit_subscribers"
     Column      |          Type          | Collation | Nullable | Default
-----------------+------------------------+-----------+----------+---------
 retrieved_utc   | bigint                 |           |          |
 subreddit       | character varying(255) |           |          |
 subreddit_id    | character varying(255) |           |          |
 num_subscribers | integer                |           |          |
```

Below is the underlying comments table which has the same columns as comments_without_duplicates, but no pkey constraint:



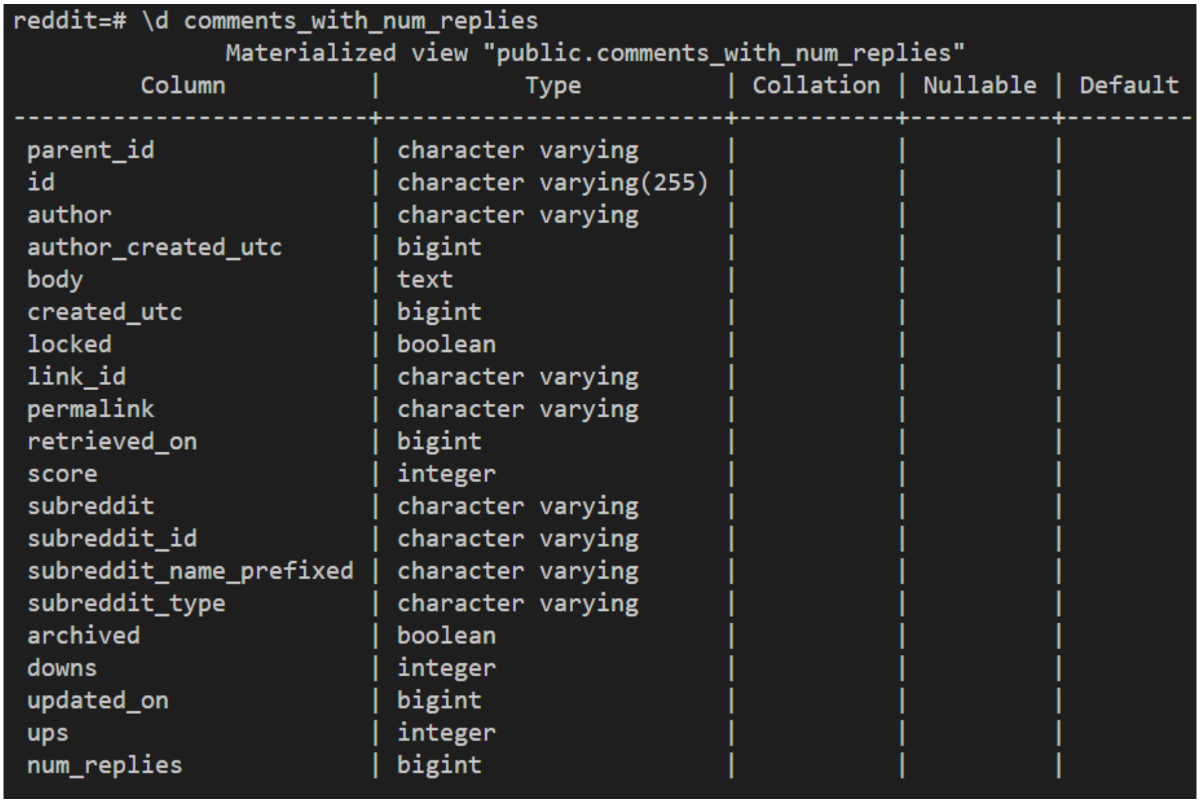
```
reddit=# \d comments
                              Table "public.comments"
         Column          |          Type          | Collation | Nullable | Default
-------------------------+------------------------+-----------+----------+---------
 id                      | character varying(255) |           | not null |
 archived                | boolean                |           |          |
 author                  | character varying(255) |           |          |
 author_created_utc      | bigint                 |           |          |
 author_fullname         | character varying(255) |           |          |
 body                    | text                   |           |          |
 controversiality        | integer                |           |          |
 created_utc             | bigint                 |           |          |
 downs                   | integer                |           |          |
 edited                  | boolean                |           |          |
 locked                  | boolean                |           |          |
 name                    | character varying(255) |           |          |
 num_reports             | integer                |           |          |
 parent_id               | character varying(255) |           |          |
 permalink               | character varying(255) |           |          |
 retrieved_on            | bigint                 |           |          |
 score                   | integer                |           |          |
 subreddit               | character varying(255) |           |          |
 subreddit_id            | character varying(255) |           |          |
 subreddit_name_prefixed | character varying(255) |           |          |
 subreddit_type          | character varying(255) |           |          |
 total_awards_received   | integer                |           |          |
 updated_on              | bigint                 |           |          |
 ups                     | integer                |           |          |
 link_id                 | character varying(255) |           |          |
```

Below is the underlying submissions table with the same columns as submissions_without_duplicates but no pkey constraint:



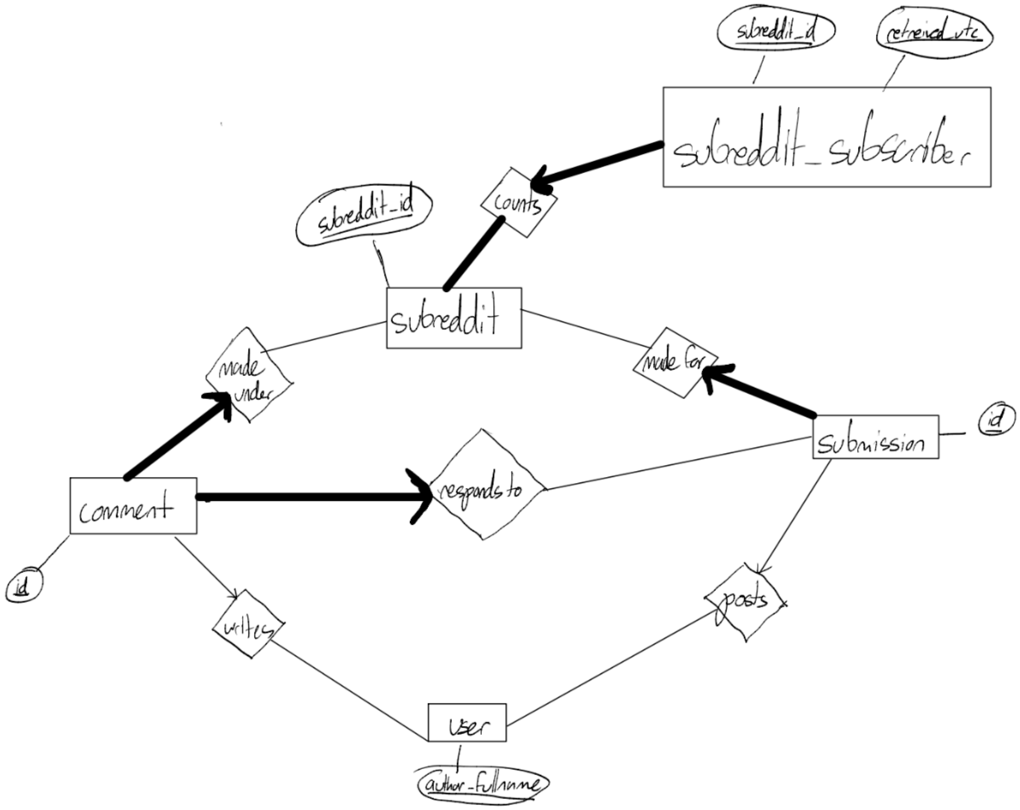
```
reddit=# \d submissions
                            Table "public.submissions"
         Column          |          Type          | Collation | Nullable | Default
-------------------------+------------------------+-----------+----------+---------
 id                      | character varying(255) |           | not null |
 archived                | boolean                |           |          |
 author                  | character varying(255) |           |          |
 author_created_utc      | bigint                 |           |          |
 author_fullname         | character varying(255) |           |          |
 created_utc             | bigint                 |           |          |
 downs                   | integer                |           |          |
 is_self                 | boolean                |           |          |
 locked                  | boolean                |           |          |
 name                    | character varying(255) |           |          |
 num_comments            | integer                |           |          |
 num_crossposts          | integer                |           |          |
 num_reports             | integer                |           |          |
 permalink               | text                   |           |          |
 score                   | integer                |           |          |
 selftext                | text                   |           |          |
 spoiler                 | boolean                |           |          |
 subreddit               | character varying(255) |           |          |
 subreddit_id            | character varying(255) |           |          |
 subreddit_name_prefixed | character varying(255) |           |          |
 subreddit_subscribers   | integer                |           |          |
 subreddit_type          | character varying(255) |           |          |
 title                   | text                   |           |          |
 total_awards_received   | integer                |           |          |
 ups                     | integer                |           |          |
 upvote_ratio            | double precision       |           |          |
 url                     | text                   |           |          |
```

Below is the comments_with_num_replies materialized view which has the same columns as comments_without_duplicates as well as an additional num_replies column:



```
reddit=# \d comments_with_num_replies
                Materialized view "public.comments_with_num_replies"
         Column          |          Type          | Collation | Nullable | Default
-------------------------+------------------------+-----------+----------+---------
 parent_id               | character varying      |           |          |
 id                      | character varying(255) |           |          |
 author                  | character varying      |           |          |
 author_created_utc      | bigint                 |           |          |
 body                    | text                   |           |          |
 created_utc             | bigint                 |           |          |
 locked                  | boolean                |           |          |
 link_id                 | character varying      |           |          |
 permalink               | character varying      |           |          |
 retrieved_on            | bigint                 |           |          |
 score                   | integer                |           |          |
 subreddit               | character varying      |           |          |
 subreddit_id            | character varying      |           |          |
 subreddit_name_prefixed | character varying      |           |          |
 subreddit_type          | character varying      |           |          |
 archived                | boolean                |           |          |
 downs                   | integer                |           |          |
 updated_on              | bigint                 |           |          |
 ups                     | integer                |           |          |
 num_replies             | bigint                 |           |          |
```

- Show an ER diagram of your database



The ER diagram does not explicitly show all non primary key attributes, but they are the other columns in the database schema. Note that in this diagram comments and submissions are comments_without_duplicates and submissions_without_duplicates respectively.

System and Database Setup

Postgres Setup

We found when uploading that the json contained a few duplicate entries for comments (68/100,833) as well as some for submissions (181/89,181). It seems like the duplicate entries for the comments and submissions may have been due to how the data was collected/scraped. From further analysis, one possible explanation is that the dataset contains results from multiple data retrievals, resulting in some submissions and comments being included multiple times.

Different data retrievals:

```
reddit=# SELECT retrieved_on, COUNT(*) FROM comments GROUP BY
retrieved_on ORDER BY COUNT(*) DESC LIMIT 10;
 retrieved_on | count
--------------+-------
              | 60358
   1425948671 |    16
   1425855694 |    15
   1425948673 |    13
   1693494772 |    13
   1425948672 |    12
   1425967436 |    11
   1426022210 |    10
   1425855580 |    10
   1693560207 |    10
(10 rows)
```

… resulting in the same comment being duplicated in the dataset.

```
reddit=# SELECT id, retrieved_on FROM comments WHERE id = 'dmehvq5';
   id    | retrieved_on
---------+--------------
 dmehvq5 |   1504556732
 dmehvq5 |   1504556732
 dmehvq5 |   1506386549
(3 rows)
```

To circumvent duplicate entries preventing us from creating primary keys, we decided to `create materialized_views` (comments_without_duplicates, submissions_without_duplicates), leaving the underlying comments and submissions tables. We then created unique indexes on the materialized views of the comments's id and submissions's id column respectively, effectively creating primary keys.

Postgres Server

We are self-hosting a shared PostgreSQL server, providing 24/7 uptime to ensure consistent data access and availability for all team members. We are running **PostgreSQL 17.0** inside a Docker container.

Postgres Setup

After performing EDA on the dataset, which only contains `submission` and `comments` originally, we selected relevant attributes from both entity sets to define the schemas for the database, which can be viewed here. In order to upload the data to Postgres, we decided to use the Python `pandas` library as the intermediate tool, motivated by our familiarity with the library and its support for data preprocessing steps. We construct `pandas.Dataframe` from entries read from `ndjson` files, precisely 1000 lines each. We then select the relevant columns and upload the data to the database using `Dataframe.to_sql()` method. Our approach works but relies heavily on sequential processing, leaving room for future optimization.

After the `submissions` and `comments` data is uploaded successfully into the corresponding Postgres tables, we begin the normalizing steps based on our ER diagram. We construct the `users` and `subreddits` tables from the original two tables. This step is rather straightforward as the data is already on Postgres we can query `submissions` and `comments` to make new tables.

`subreddits`

Entries in `submissions` and `comments` both have attributes `subreddit`, `subreddit_id`, `subreddit_name_prefixed`, `subreddit_type`. Entries in `submissions` also contain `subreddit_subscribers`, which is the number of subscribers to the subreddit. This is useful to track the number of subscribers to a subreddit over time and is our motivation for an additional table `subreddit_subscribers` which contains time series data of subreddits' subscribers count over time.

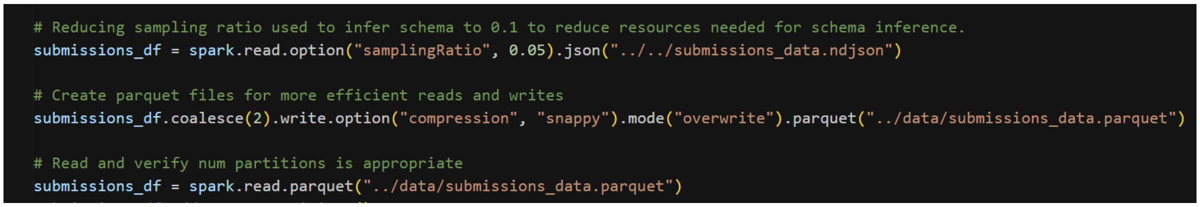
We query `submissions` to retrieve the most recent post where all subreddit data is available, representing the latest "subreddit data fetch" that provides the most up-to-date information about a subreddit. Since entries in `comments` also contain the same data granularity, we take `UNION ALL` of the query on both table tables and select the latest data entry of a subreddit (the one with larger `created_utc`) to insert to the table. We apply the same principles to construct the `users` table.

The code for database upload steps can be found here.

Spark Setup

All Spark operations were completed in the spark_processing.ipynb notebook in the repository code directory. First, a Spark session was created and the NDJSON data was read from both files. During the read, no schema was explicitly stated which utilized Spark's schema inference ability. To reduce memory cost, the sample size used to infer the schema was reduced to 5 percent of the data and the number of partitions used was reduced by manually assigning a default partition of 2 and coalescing when doing read and writes. The loaded data was then written into parquet files to make later reads and writes of the data more efficient (original read and write are commented out after the parquet file is created). Additional read and writes are necessary because Spark does not persist data. 2 was chosen as the default number of partitions after testing out varying numbers of partitions to determine the maximum number of partitions that could be used without running into memory errors due to Spark's memory requirements for managing each partition and performing operations on the data in each node.

Screenshot of using Spark to read to a dataframe and write to parquet:



```
# Reducing sampling ratio used to infer schema to 0.1 to reduce resources needed for schema inference.
submissions_df = spark.read.option("samplingRatio", 0.05).json("../../submissions_data.ndjson")

# Create parquet files for more efficient reads and writes
submissions_df.coalesce(2).write.option("compression", "snappy").mode("overwrite").parquet("../data/submissions_data.parquet")

# Read and verify num partitions is appropriate
submissions_df = spark.read.parquet("../data/submissions_data.parquet")
```

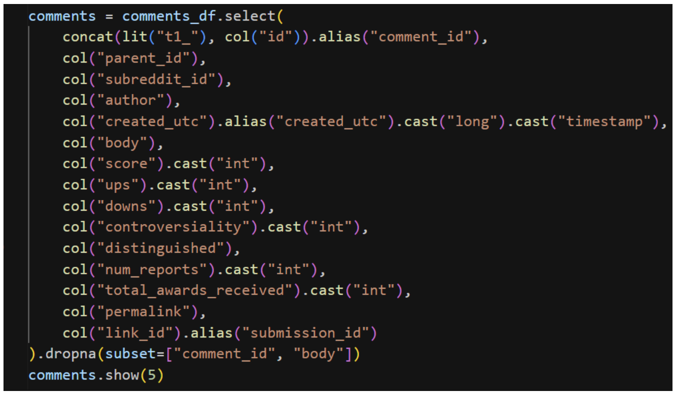
Once loaded, the schema was visualized and the dataframe described to determine what columns were available and which columns were null. Of the available columns in both dataframes, the most relevant columns were selected. The columns were used to produce the following normalized dataframes: users, subreddits, subscriber_counts, submissions, and comments. We wanted the schema of the Spark dataframes we fabricated to closely resemble the tables in postgres. Doing this will make it easier to load data directly from Spark to postgres making the pipeline more seamless in later updates.

To produce the dataframes, the desired columns from the dataframe were selected, renamed, and cast if appropriate. For fabricated dataframes where id numbers would be treated as primary key values, such as users and subreddits, duplicates were dropped in order to enforce uniqueness. For subscriber_counts and comments, null rows for specific columns were dropped as having a row without a timestamp or a comment without a body did not provide value.

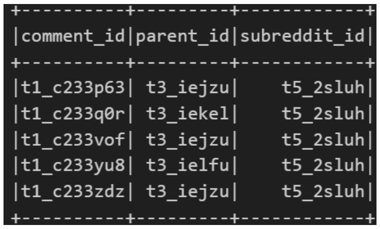
During this process, we visually discerned a couple discrepancies between columns that reference ids. Spark makes it easy to quickly visualize entity discrepancies. Some columns reference an id number that includes a prefix to indicate the data type being referenced and others are missing this indicator. To resolve this, all columns where ids were missing an indicator were processed. Fortunately, the columns with this discrepancy all contained like types, so no involved entity resolution methods were needed. For example, ids in the comment id column were missing the "t1_" prefix whereas ids in the parent id column contained them. This fix involved a simple pass to concatenate the "t1_" prefix to each object in the comment id column.

Columns with more than a set percentage of null values were also dropped to maintain data integrity. Columns that were deemed imputable for later use were marked with a TODO. For example, null subscriber counts with a timestamp and subreddit id can be imputed using runs and upvote values can be imputed using ups and downs. These are tasks that are not an immediate priority but would be worth revisiting to improve the robustness of the data.

Screenshots showing some of the preprocessing steps mentioned (e.g. selecting columns, dropping nulls, resolving the id discrepancies, etc) and the comment_id column after resolution:



```
comments = comments_df.select(
    concat(lit("t1_"), col("id")).alias("comment_id"),
    col("parent_id"),
    col("subreddit_id"),
    col("author"),
    col("created_utc").alias("created_utc").cast("long").cast("timestamp"),
    col("body"),
    col("score").cast("int"),
    col("ups").cast("int"),
    col("downs").cast("int"),
    col("controversiality").cast("int"),
    col("distinguished"),
    col("num_reports").cast("int"),
    col("total_awards_received").cast("int"),
    col("permalink"),
    col("link_id").alias("submission_id")
).dropna(subset=["comment_id", "body"])
comments.show(5)
```



```
+----------+---------+------------+
|comment_id|parent_id|subreddit_id|
+----------+---------+------------+
|t1_c233p63| t3_iejzu|    t5_2sluh|
|t1_c233q0r| t3_iekel|    t5_2sluh|
|t1_c233vof| t3_iejzu|    t5_2sluh|
|t1_c233yu8| t3_ielfu|    t5_2sluh|
|t1_c233zdz| t3_iejzu|    t5_2sluh|
+----------+---------+------------+
```

PostgreSQL Tasks and Queries

For this part of the project, we are interested in understanding the Reddit data we currently have in our database. The queries selected explore relationships in engagement. These relationships allow us to better understand the connectivity between different entities in our database and offer insight into user behavior, which are often subjects of interest for social media and entertainment platforms like Reddit.

Task / Query 1 : Which subreddits had the most subscribers according to the data?

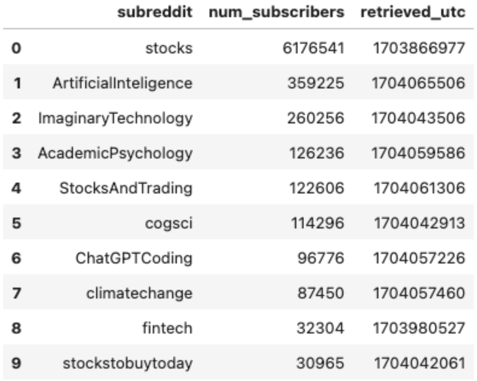
Subreddit counts were obtained as time series data, so as a sanity check of our normalized data we wanted to run a simple query to find out which subreddits had the highest number of subscribers at their latest recording in the database. This query would provide us insight into what some of the latest dates of collection were for our dataset as well as what subreddits may hold the most influence in our dataset.

Because this query is achievable as a simple query in Postgres, we thought it would be a good simple benchmark for evaluating the use of Postgres. One question we might want to know is in what cases does it make sense to load data into Postgres for a machine learning pipeline? Are some queries better suited for a normalized SQL database system? We felt this task would be suited for both giving us insight about the extracted data and helping us evaluate Postgres.

Due to the nature of the subscriptions we tracked in our database, each subreddit in our subreddit_subscribers table has multiple entries with different timestamps represented by numerical UTC. Thus for each table we sought to select the most recent timestamp for each subreddit to asses the subscriber count at the most recent time upon data collection

Our approach was to use a CTE to first extract a view of the data just displaying subreddit entries with their latest timestamp update. Afterwards we then select the entries from the view and order by the num_subscriber count in descending order to see the subreddits with the most subscribers. This is a reasonable solution given the timestamp updates our data has, and it allows us to understand the largest subreddits in our dataset in a fairly easy manner.

The performance analysis gives us an execution time of 1.52 seconds and a planning time of 0.083 ms. The query uses quicksort with 25KB memory and external merge sorting for handling distinct subreddit records.



	subreddit	num_subscribers	retrieved_utc
0	stocks	6176541	1703866977
1	ArtificialInteligence	359225	1704065506
2	ImaginaryTechnology	260256	1704043506
3	AcademicPsychology	126236	1704059586
4	StocksAndTrading	122606	1704061306
5	cogsci	114296	1704042913
6	ChatGPTCoding	96776	1704057226
7	climatechange	87450	1704057460
8	fintech	32304	1703980527
9	stockstobuytoday	30965	1704042061

Task / Query 2 : Do controversial comments or higher scoring comments get more replies?

We wanted to see whether comments with a large number of replies were correlated with comments with a high score (difference between upvotes and downvotes). We created a materialized view of the number of replies for each comment utilizing the fact that comments have a parent_id entry and joining the comments table with itself on the parent_id and id. Materializing the view is useful as we will definitely see many queries using num_replies for each comment as it is a way to gauge engagement. Using the materialized view, we created a series of CTE's (one getting the top 10,000 comments with the highest score, and another getting the top 10,000 comments with the most number of replies), and then counted how many comments were in both CTE's. Shockingly, the count was only 97, showing that very few top scoring comments are also comments with the top number of replies. The code and result of the query is in the appendix. The query itself took 559ms to run using a series of sequential scans, sorts, and hash joins.

Task / Query 3: Who are the most active users?

To gauge this we decided to find the users who had the highest total number of comments and submissions for each subreddit. Notice that there may have been some users that only submit posts or comments thus taking the sum of the two allows us to more accurately gauge who the most active users are. The only caveat with the query is that since users can delete their accounts, there may be people who have made many submissions and/or comments but deleted their account, furthermore there may be people with multiple alt accounts that are more active, but given the limitations taking the total number of submissions and comments is the best we can do to gauge most active users. The query took 1.45 seconds to run, and used a hash full join to join the two CTE's. The query and output are in the appendix section. We had to use a full outer join to make sure that users that haven't created submissions or haven't created comments are still included to see their activity, furthermore we had to use the coalesce function so that null values would be changed to 0 to allow for the addition of comments and submissions.

Non-Relational Tools Comparison

Tool Choice: Apache Spark

Task / Query 1 : Preprocessing + EDA

Based on internet searches and exploring Reddit's developer and API website, Reddit does not offer comprehensive documentation about the data retrieved via its API. Further, the data comes in a semi-structured JSON format with each object containing varying sets of attributes. For this reason, we wanted a flexible way to quickly load the data, understand it, and process it for Postgres.

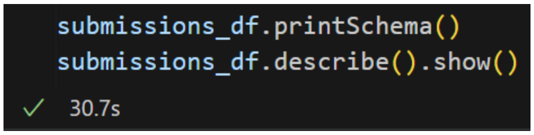
This task seemed like a good task for which to evaluate the Apache Spark tool. Spark is designed for batch processing, supporting the efficient processing of large amounts of data and making it useful for machine learning pipelines and applicable to our goals. We thought it made sense to use it for our Reddit data pipeline also because later tasks may involve scaling the data which Spark is also intended to be useful for. Also, Spark supports easy integration with other tools, such as Kafka and Postgres, making it easy to integrate it with streaming data down the road or loading data into Postgres which was our downstream task for this project. For the current task at hand though, Spark, like Pandas, offers a variety of functions designed for EDA and processing making it a viable option for this phase of the project.

Spark allowed us to immediately infer a schema, load the data into a dataframe object, and describe key statistics about it, saving us time upfront to try to manually parse the data. Lowering the sampling percentage for the schema inference process was used to increase performance by decreasing the data Spark processed to infer the schema saving minutes of time (>5 mins to run prior). Increasing the number of partitions improved runtime as well, but was limited by the available memory on the device used. We used available functions for preprocessing and visualizing the data to validate the normalized data frames looked as we wanted them to. These runtimes were satisfactory. The performance of removing null columns based on a threshold varied.

These are the processing times for reference:

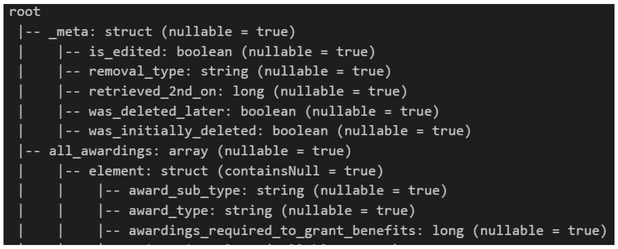
- Initial read, write, and reread of the data into Spark dataframes = ~ 3 min. 3 secs
- Subsequent reads from parquet into Spark dataframes = ~ 41.2s
- Printing both Schemas and describing the dataframes = ~ 1 min. 0 secs
- Creating and visualizing tables = <1s
- Removing columns above a threshold of null values = 15.1s (sub), 48.6s (com)

Screenshots using Spark for loading the data and preprocessing it were provided above. Screenshots of code, the visualized statistics, and a snippet of the schema of the submissions data:

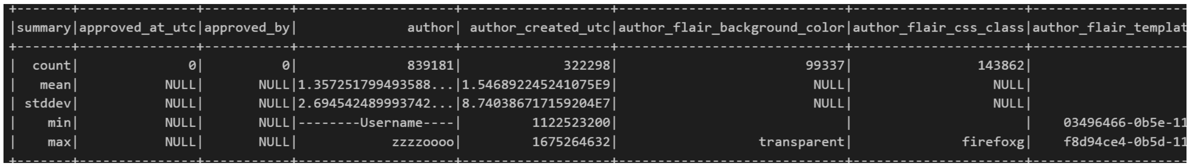


```
submissions_df.printSchema()
submissions_df.describe().show()
```

✓ 30.7s



```
root
 |-- _meta: struct (nullable = true)
 |    |-- is_edited: boolean (nullable = true)
 |    |-- removal_type: string (nullable = true)
 |    |-- retrieved_2nd_on: long (nullable = true)
 |    |-- was_deleted_later: boolean (nullable = true)
 |    |-- was_initially_deleted: boolean (nullable = true)
 |-- all_awardings: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- award_sub_type: string (nullable = true)
 |    |    |-- award_type: string (nullable = true)
 |    |    |-- awardings_required_to_grant_benefits: long (nullable = true)
```



```
+-------+---------------+-----------+--------------------+-------------------+-----------------------------+----------------------+--------------------
|summary|approved_at_utc|approved_by|              author| author_created_utc|author_flair_background_color|author_flair_css_class|author_flair_templat
+-------+---------------+-----------+--------------------+-------------------+-----------------------------+----------------------+--------------------
|  count|              0|          0|              839181|             322298|                        99337|                143862|
|   mean|           NULL|       NULL|1.357251799493588...|1.546892245241075E9|                         NULL|                  NULL|
| stddev|           NULL|       NULL|2.694542489993742...|8.740386717159204E7|                         NULL|                  NULL|
|    min|           NULL|       NULL|--------Username----|         1122523200|                             |                      |    03496466-0b5e-11
|    max|           NULL|       NULL|            zzzzoooo|         1675264632|                  transparent|              firefoxg|    f8d94ce4-0b5d-11
+-------+---------------+-----------+--------------------+-------------------+-----------------------------+----------------------+--------------------
```

Task / Query 2 : Which subreddits had the most subscribers according to the data?

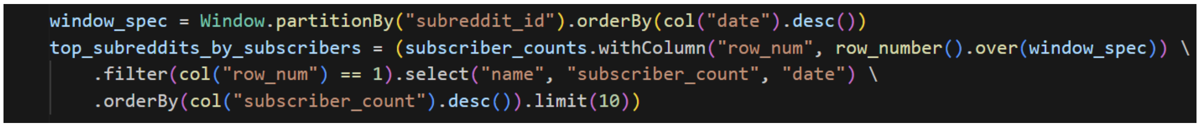
As mentioned above, for this task our goal was to understand what subreddits may hold more influence in our dataset and what some of the latest dates of collection in our dataset are. We wanted to run this query in Spark in addition to running it in Postgres to evaluate the performance of Spark as an alternative and to help us assess the usefulness of using Postgres as opposed to Spark for simple queries and sanity checks.

Because Spark offers a variety of dataframe functions similar to Pandas, achieving this kind of query was possible, but more involved as it required knowing what functions were available and combining multiple function calls to get the same results. The combined functions needed involved utilizing a window function to partition by subreddit on the subscriber counts table, rank the entries by time, then grab the entries ranked 1 for each subreddit. Then different functions were used to select the desired output columns, order the results, and limit it to top 10. Caching intermediate results via `.cache()` improved performance on a second query by 10x but consumed more memory. Due to the learning curve needed to interpret Spark memory consumption during tasks, task memory was excluded from the reported results.

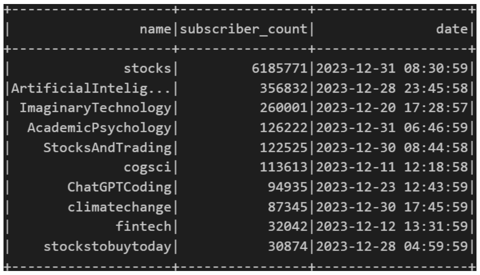
Performance runtime was obtained using python's time module:

- Processing time: ~ 889.559 ms uncached, ~85.229 ms cached
- Storage memory: 0 uncached, 366.3 MiB cached

Screenshot of the code:



```
window_spec = Window.partitionBy("subreddit_id").orderBy(col("date").desc())
top_subreddits_by_subscribers = (subscriber_counts.withColumn("row_num", row_number().over(window_spec)) \
    .filter(col("row_num") == 1).select("name", "subscriber_count", "date") \
    .orderBy(col("subscriber_count").desc()).limit(10))
```



```
+--------------------+----------------+-------------------+
|                name|subscriber_count|               date|
+--------------------+----------------+-------------------+
|              stocks|         6185771|2023-12-31 08:30:59|
|ArtificialIntelig...|          356832|2023-12-28 23:45:58|
|  ImaginaryTechnology|          260001|2023-12-20 17:28:57|
|   AcademicPsychology|          126222|2023-12-31 06:46:59|
|     StocksAndTrading|          122525|2023-12-30 08:44:58|
|              cogsci|          113613|2023-12-11 12:18:58|
|        ChatGPTCoding|           94935|2023-12-23 12:43:59|
|        climatechange|           87345|2023-12-30 17:45:59|
|             fintech|           32042|2023-12-12 13:31:59|
|     stockstobuytoday|           30874|2023-12-28 04:59:59|
+--------------------+----------------+-------------------+
```

Task / Query 3 : Do controversial comments or higher scoring comments get more replies?

Really two queries, but for this task we wanted to obtain some insight into user behavior on Reddit and compare how replies differ for controversial comments versus highly liked comments. This query is more involved than the previous query, requiring at least one join. Because we did something similar in Postgres, we thought this would be a good task for evaluating how Spark compares to Postgres on more complex queries involving joins. Spark has functions that can allow it to perform this query, however, as before, the task was more involved.

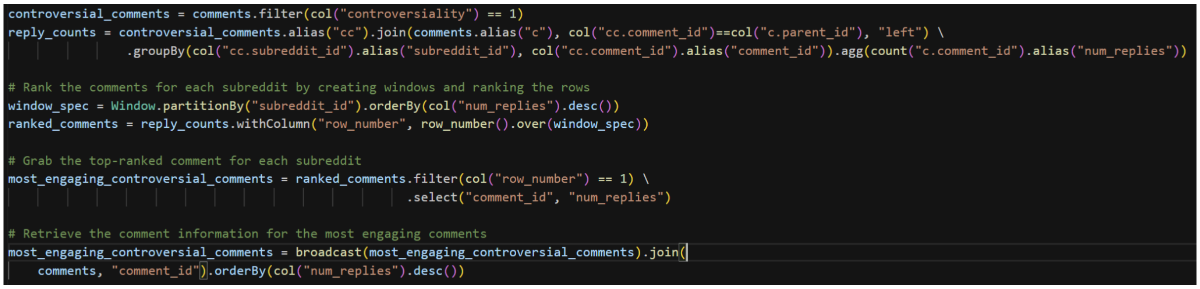
Performing both queries needed an understanding of multiple functions. Because controversiality was stored as a boolean, we first needed to filter to obtain all controversial comments. For each controversial comment, a join was used with the original comments table to obtain all replies by finding comments whose parent id matched the controversial comment id. Then group by and a count aggregation function to get the number of replies for each controversial comment. Our table was then partitioned by subreddit id and ordered by the number of replies to number each entry for use in getting the comment in each subreddit with the most replies. Finally, another join was needed with the comments table to get all comment information for each of the controversial comments with the most replies.

Due to the memory needed to perform this entire query, the final join failed. Because our table was small, a more optimal join was performed using the broadcast function. This reduced memory need and improved overall performance. Similar functions were used for finding the top scoring comments and obtaining their number of replies. Code is provided below.

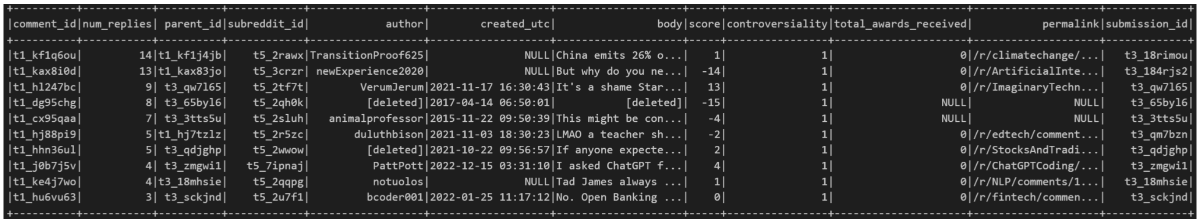
Similarly as before, caching saw ~ 10x performance increase. Performance runtime was obtained using python's time module:

- Processing time: ~ 8945.686 ms uncached, ~72.000 ms cached
- Storage memory: 0 uncached, 27.3 KiB cached

Screenshot of the code for controversial comments with most engagement and output:



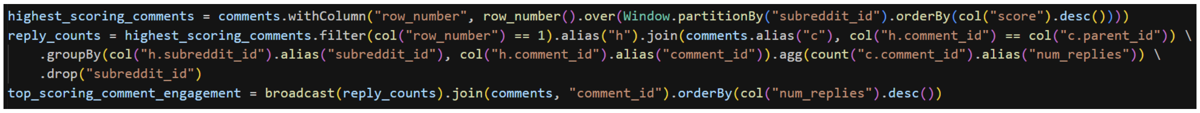
```python
controversial_comments = comments.filter(col("controversiality") == 1)
reply_counts = controversial_comments.alias("cc").join(comments.alias("c"), col("cc.comment_id")==col("c.parent_id"), "left") \
                    .groupBy(col("cc.subreddit_id").alias("subreddit_id"), col("cc.comment_id").alias("comment_id")).agg(count("c.comment_id").alias("num_replies"))

# Rank the comments for each subreddit by creating windows and ranking the rows
window_spec = Window.partitionBy("subreddit_id").orderBy(col("num_replies").desc())
ranked_comments = reply_counts.withColumn("row_number", row_number().over(window_spec))

# Grab the top-ranked comment for each subreddit
most_engaging_controversial_comments = ranked_comments.filter(col("row_number") == 1) \
                                                        .select("comment_id", "num_replies")

# Retrieve the comment information for the most engaging comments
most_engaging_controversial_comments = broadcast(most_engaging_controversial_comments).join(
    comments, "comment_id").orderBy(col("num_replies").desc())
```



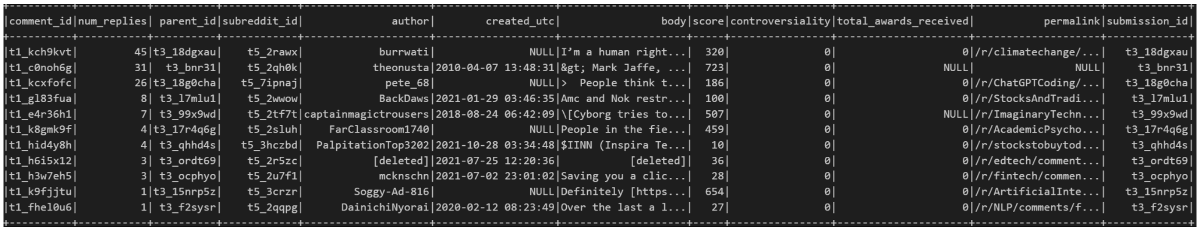
```
+----------+-----------+----------+------------+------------------+-------------------+--------------------+-----+----------------+---------------------+--------------------+-------------+
|comment_id|num_replies| parent_id|subreddit_id|            author|        created_utc|                body|score|controversiality|total_awards_received|           permalink|submission_id|
+----------+-----------+----------+------------+------------------+-------------------+--------------------+-----+----------------+---------------------+--------------------+-------------+
|t1_kf1q6ou|         14|t1_kf1j4jb|    t5_2rawx|TransitionProof625|               NULL|China emits 26% o...|    1|               1|                    0|/r/climatechange/...|   t3_18rimou|
|t1_kax8i0d|         13|t1_kax83jo|    t5_3crzr| newExperience2020|               NULL|But why do you ne...|  -14|               1|                    0|/r/ArtificialInte...|   t3_184rjs2|
|t1_hl247bc|          9| t3_qw7165|    t5_2tf7t|        VerumJerum|2021-11-17 16:30:43|It's a shame Star...|   13|               1|                    0|/r/ImaginaryTechn...|    t3_qw7165|
|t1_dg95chg|          8| t3_65byl6|    t5_2qh0k|         [deleted]|2017-04-14 06:50:01|           [deleted]|  -15|               1|                 NULL|                NULL|    t3_65byl6|
|t1_cx95qaa|          7| t3_3tts5u|    t5_2sluh|   animalprofessor|2015-11-22 09:50:39|This might be con...|   -4|               1|                 NULL|                NULL|    t3_3tts5u|
|t1_hj88pi9|          5|t1_hj7tzlz|    t5_2r5zc|       duluthbison|2021-11-03 18:30:23|LMAO a teacher sh...|   -2|               1|                    0|/r/edtech/comment...|    t3_qm7bzn|
|t1_hhn36ul|          5| t3_qdjghp|    t5_2wwow|         [deleted]|2021-10-22 09:56:57|If anyone expecte...|    2|               1|                    0|/r/StocksAndTradi...|    t3_qdjghp|
|t1_j0b7j5v|          4| t3_zmgwi1|   t5_7ipnaj|          PattPott|2022-12-15 03:31:10|I asked ChatGPT f...|    4|               1|                    0|/r/ChatGPTCoding/...|    t3_zmgwi1|
|t1_ke4j7wo|          4|t3_18mhsie|    t5_2qqpg|          notuolos|               NULL|Tad James always ...|    1|               1|                    0|/r/NLP/comments/1...|   t3_18mhsie|
|t1_hu6vu63|          3| t3_sckjnd|    t5_2u7f1|         bcoder001|2022-01-25 11:17:12|No. Open Banking ...|    0|               1|                    0|/r/fintech/commen...|    t3_sckjnd|
+----------+-----------+----------+------------+------------------+-------------------+--------------------+-----+----------------+---------------------+--------------------+-------------+
```

- Processing time: ~ 8347.364 ms uncached, ~70.393 ms cached
- Storage memory: 0 uncached, 34.8 KiB cached

Screenshot of the code for top scoring comment engagement and output:



```python
highest_scoring_comments = comments.withColumn("row_number", row_number().over(Window.partitionBy("subreddit_id").orderBy(col("score").desc())))
reply_counts = highest_scoring_comments.filter(col("row_number") == 1).alias("h").join(comments.alias("c"), col("h.comment_id") == col("c.parent_id")) \
    .groupBy(col("h.subreddit_id").alias("subreddit_id"), col("h.comment_id").alias("comment_id")).agg(count("c.comment_id").alias("num_replies")) \
    .drop("subreddit_id")
top_scoring_comment_engagement = broadcast(reply_counts).join(comments, "comment_id").orderBy(col("num_replies").desc())
```



```
+----------+-----------+----------+------------+--------------------+-------------------+--------------------+-----+----------------+---------------------+--------------------+-------------+
|comment_id|num_replies| parent_id|subreddit_id|              author|        created_utc|                body|score|controversiality|total_awards_received|           permalink|submission_id|
+----------+-----------+----------+------------+--------------------+-------------------+--------------------+-----+----------------+---------------------+--------------------+-------------+
|t1_kch9kvt|         45|t3_18dgxau|    t5_2rawx|            burrwati|               NULL|I’m a human right...|  320|               0|                    0|/r/climatechange/...|   t3_18dgxau|
|t1_c0noh6g|         31| t3_bnr31|    t5_2qh0k|           theonusta|2010-04-07 13:48:31|&gt; Mark Jaffe, ...|  723|               0|                 NULL|                NULL|     t3_bnr31|
|t1_kcxfofc|         26|t3_18g0cha|   t5_7ipnaj|             pete_68|               NULL|>  People think t...|  186|               0|                    0|/r/ChatGPTCoding/...|   t3_18g0cha|
|t1_gl83fua|          8| t3_l7mlu1|    t5_2wwow|            BackDaws|2021-01-29 03:46:35|Amc and Nok restr...|  100|               0|                    0|/r/StocksAndTradi...|    t3_l7mlu1|
|t1_e4r36h1|          7| t3_99x9wd|    t5_2tf7t|captainmagictrousers|2018-08-24 06:42:09|\[Cyborg tries to...|  507|               0|                 NULL|/r/ImaginaryTechn...|    t3_99x9wd|
|t1_k8gmk9f|          4|t3_17r4q6g|    t5_2sluh|    FarClassroom1740|               NULL|People in the fie...|  459|               0|                    0|/r/AcademicPsycho...|   t3_17r4q6g|
|t1_hid4y8h|          4| t3_qhhd4s|   t5_3hczbd|  PalpitationTop3202|2021-10-28 03:34:48|$IINN (Inspira Te...|   10|               0|                    0|/r/stockstobuytod...|    t3_qhhd4s|
|t1_h6i5x12|          3| t3_ordt69|    t5_2r5zc|           [deleted]|2021-07-25 12:20:36|           [deleted]|   36|               0|                    0|/r/edtech/comment...|    t3_ordt69|
|t1_h3w7eh5|          3| t3_ocphyo|    t5_2u7f1|            mcknschn|2021-07-02 23:01:02|Saving you a clic...|   28|               0|                    0|/r/fintech/commen...|    t3_ocphyo|
|t1_k9fjjtu|          1|t3_15nrp5z|    t5_3crzr|        Soggy-Ad-816|               NULL|Definitely [https...|  654|               0|                    0|/r/ArtificialInte...|   t3_15nrp5z|
|t1_fhel0u6|          1| t3_f2sysr|    t5_2qqpg|      DainichiNyorai|2020-02-12 08:23:49|Over the last a l...|   27|               0|                    0|/r/NLP/comments/f...|    t3_f2sysr|
+----------+-----------+----------+------------+--------------------+-------------------+--------------------+-----+----------------+---------------------+--------------------+-------------+
```

Tool Comparisons

Comparing loading the data, preprocessing, and data exploration (Task 1 for non-relational):

Postgres and Spark are like apples to oranges for this purpose primarily due to their flexibility in the data loading process. Postgres requires structured data to be loaded into it whereas Spark can load semi-structured data as well, therefore Spark is a better fit for loading the initial raw Reddit data, exploring the data, and moving it around. It offers tools for preprocessing the data and normalizing it, and supports distributed processing making it a better fit as data volume is scaled given sufficient hardware. With less data or limited hardware, the utility of Spark may decrease. This is evidenced by our partition limit of 1-2 for certain tasks.

Because Spark can also perform some exploratory queries, as in our query tasks, there may be situations in which we might choose Spark over Postgresql or Postgresql over Spark. While both Postgres and Spark are capable of our desired queries, Spark may be a good option if these queries are one offs and exploration doesn't require data persistence. This is because Spark supports loading unnormalized, unprocessed data from various sources and queries making it a flexible tool for doing both without much setup outside of configuring the Spark client. One downside is Spark may have a steeper learning curve, requiring knowledge of Spark to configure it, understand performance, and properly combine various functions to get the desired query.

For querying data that is structured, used frequently, and does not need to change often Postgresql may be a better fit. For example, in our case we performed all queries on the same data, structured, and across different days so having data that persists makes sense. Spark does not persist data and requires a separate store. Further, it does not cache intermediate results by default meaning that every query that Spark runs reloads the data from storage and processes it. A screenshot of one of the query plans is provided below showing this in action. Postgres is also more straightforward to set up and queries are typically simpler to write. …

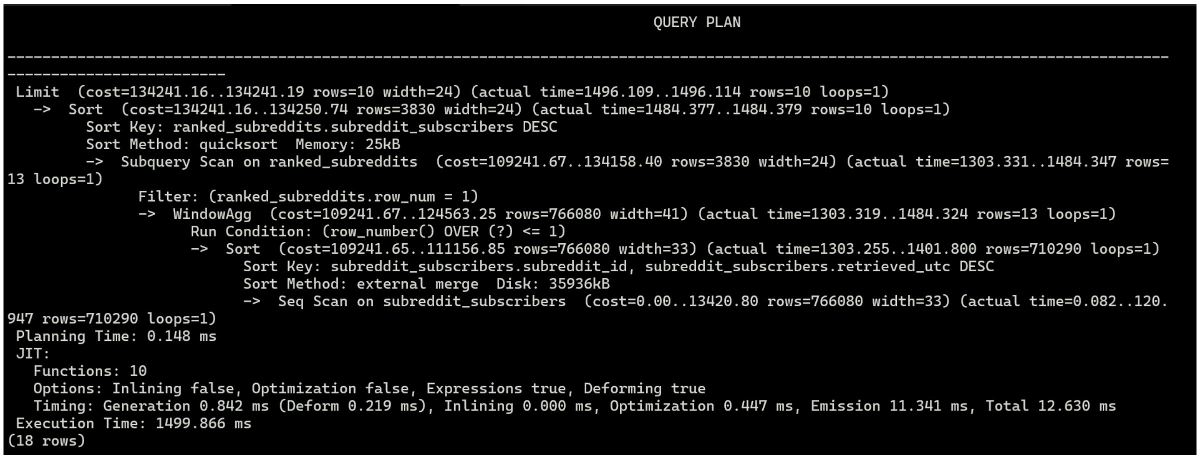
Further, Postgres manages much of the optimization process when writing queries whereas Spark is more hands-on. Learning how to read the Spark UI also features a bit more of a learning curve, screenshot provided below…

Comparing Query Execution:

Task 2: Which subreddits had the most subscribers according to the data?

Execute query on PostgreSQL with no optimization, obtain query plan using `EXPLAIN ANALYZE`.

```
WITH ranked_subreddits AS (
      SELECT subreddit_id,
             subreddit AS name,
            num_subscribers AS subreddit_subscribers,
            retrieved_utc AS date,
            ROW_NUMBER() OVER (PARTITION BY subreddit_id ORDER BY
retrieved_utc DESC) AS row_num
      FROM subreddit_subscribers
)
SELECT
    name,subreddit_subscribers,
    TO_TIMESTAMP(date) AS readable_date
FROM ranked_subreddits
WHERE row_num = 1
ORDER BY subreddit_subscribers DESC
LIMIT 10;
```



```
                                                                          QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------
-------------------------
 Limit  (cost=134241.16..134241.19 rows=10 width=24) (actual time=1496.109..1496.114 rows=10 loops=1)
   ->  Sort  (cost=134241.16..134250.74 rows=3830 width=24) (actual time=1484.377..1484.379 rows=10 loops=1)
         Sort Key: ranked_subreddits.subreddit_subscribers DESC
         Sort Method: quicksort  Memory: 25kB
         ->  Subquery Scan on ranked_subreddits  (cost=109241.67..134158.40 rows=3830 width=24) (actual time=1303.331..1484.347 rows=
13 loops=1)
               Filter: (ranked_subreddits.row_num = 1)
               ->  WindowAgg  (cost=109241.67..124563.25 rows=766080 width=41) (actual time=1303.319..1484.324 rows=13 loops=1)
                     Run Condition: (row_number() OVER (?) <= 1)
                     ->  Sort  (cost=109241.65..111156.85 rows=766080 width=33) (actual time=1303.255..1401.800 rows=710290 loops=1)
                           Sort Key: subreddit_subscribers.subreddit_id, subreddit_subscribers.retrieved_utc DESC
                           Sort Method: external merge  Disk: 35936kB
                           ->  Seq Scan on subreddit_subscribers  (cost=0.00..13420.80 rows=766080 width=33) (actual time=0.082..120.
947 rows=710290 loops=1)
 Planning Time: 0.148 ms
 JIT:
   Functions: 10
   Options: Inlining false, Optimization false, Expressions true, Deforming true
   Timing: Generation 0.842 ms (Deform 0.219 ms), Inlining 0.000 ms, Optimization 0.447 ms, Emission 11.341 ms, Total 12.630 ms
 Execution Time: 1499.866 ms
(18 rows)
```

The query plan shows that the execution time is 1499.866 ms (approximately 1.5 seconds), which is significantly slower compared to Spark. While indexing and clustering in PostgreSQL can reduce this time, it is unlikely to match Spark's performance, especially when it is using caches. PostgreSQL excels at providing persistent data storage, whereas Spark is optimized for processing large-scale datasets through its distributed computing architecture, which enables parallel execution. A practical use case is to store data in PostgreSQL for persistence and reliability, and load it into Spark when running complex queries or processing large datasets to leverage the strengths of both tools. However, we do have to take into account that running Spark will result in additional overhead on the system.

Team Reflections

When choosing tools for data tasks, it's crucial to balance performance, usability, and scalability. For preprocessing, **Pandas** is intuitive and easy to set up as it runs within a Python interpreter, ideal for small to medium datasets, but limited by single-threaded, in-memory processing. **Apache Spark**, while requiring more setup and expertise, handles large-scale data better with support for distributed computing. For storage and querying, **PostgreSQL** excels at persistent storage, relational queries, and indexing, while Spark is better for large-scale, one-off queries, unstructured data without persistence needs. A hybrid approach, storing data in PostgreSQL and processing it in Spark, combines their strengths but adds overhead. The choice ultimately depends on data scale, ease of use, expertise and overhead costs, emphasizing the need for experimentation to find the optimal solution.

Individual Reflections

Team Member 1: Faith Dennis

I had read ahead of time that Apache Spark had a learning curve, and it did end up taking a lot of time to learn about Spark's memory constraints just to get it working. This was the most frustrating part for me, but I found it really rewarding and exciting once I got past that hurdle. Moving forward, I still have a lot to learn about Spark's UI and about the various functions and optimizations available. I found the process of normalizing the data and making tables for Postgres kind of fun. Seeing the data processed and clean is always satisfying, and I am glad I attempted this project and learned a lot about working with data tools!

Team Member 2: …

Team Member 3: Bach Tran

I was tasked with setting up the Postgres server for the group and loading the data to Postgres. This process involves a lot of EDA which I enjoyed a lot. It was a really fun experience to look at the data, try to understand it and also come up with theories to explain why certain things are the way they are: for instance why there are multiple instances of the same comment in the comment dataset. I learned a lot more about preprocessing raw data, and began to understand why some people just opt for semi-structured databases instead, as there would be considerably less work upfront!

Team Member 4: Avik Samanta:

From this project I learned about how much effort goes into planning and preprocessing a dataset in order to turn it into a relational database. I found the process of initially wrangling the missing column data and sparse data columns in our dataset, but found the process of designing the Schema and ERD diagrams very engaging and thoughtful. To design different entities and tables we were actively discussing the relationships between users and subreddits, and how the user’s actions would impact the subreddits with more submissions and activity. Thus it felt very satisfying once did write out queries after we finally got the dataset schema designed and on the server, and were able to have more meaningful discussions on analyzing the data after knowing the schema inside-out.

References (Optional)

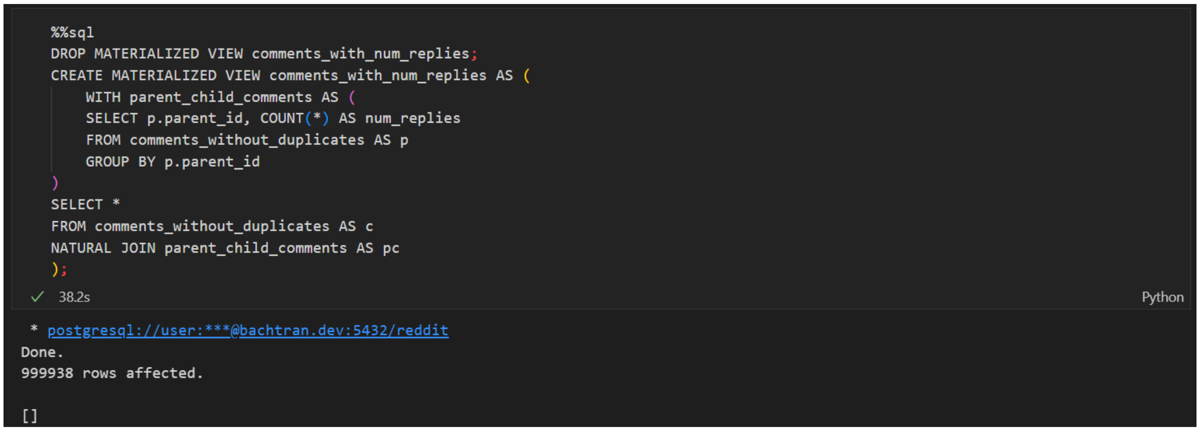
If relevant, include a reference page with citations of all outside sources used.

Appendix (Optional)

- Include links, code samples, data excerpts.

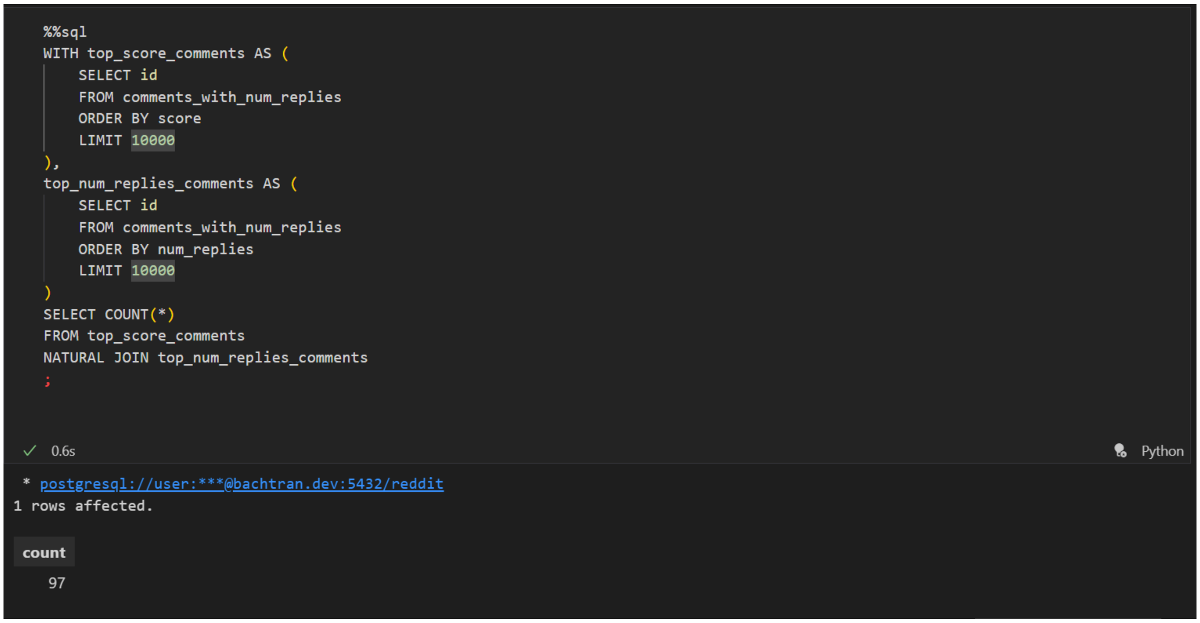
PSQL queries code:

Query 2 code snippets:



```
%%sql
DROP MATERIALIZED VIEW comments_with_num_replies;
CREATE MATERIALIZED VIEW comments_with_num_replies AS (
    WITH parent_child_comments AS (
    SELECT p.parent_id, COUNT(*) AS num_replies
    FROM comments_without_duplicates AS p
    GROUP BY p.parent_id
)
SELECT *
FROM comments_without_duplicates AS c
NATURAL JOIN parent_child_comments AS pc
);
```

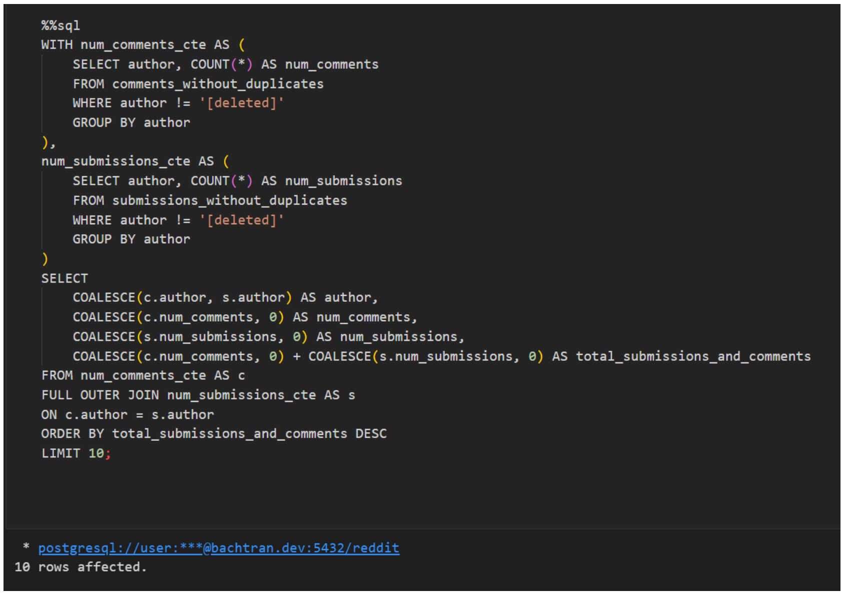
```
 * postgresql://user:***@bachtran.dev:5432/reddit
Done.
999938 rows affected.

[]
```



```
%%sql
WITH top_score_comments AS (
    SELECT id
    FROM comments_with_num_replies
    ORDER BY score
    LIMIT 10000
),
top_num_replies_comments AS (
    SELECT id
    FROM comments_with_num_replies
    ORDER BY num_replies
    LIMIT 10000
)
SELECT COUNT(*)
FROM top_score_comments
NATURAL JOIN top_num_replies_comments
;
```

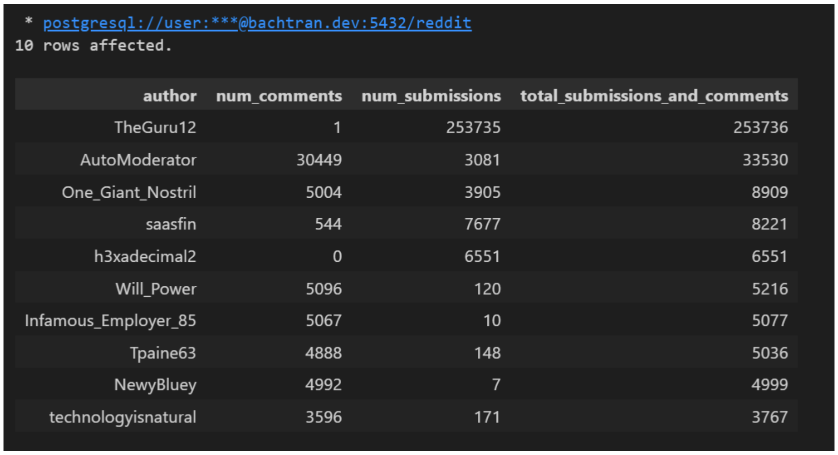
```
 * postgresql://user:***@bachtran.dev:5432/reddit
1 rows affected.

count
   97
```

Query 3 code snippets:



```
%%sql
WITH num_comments_cte AS (
    SELECT author, COUNT(*) AS num_comments
    FROM comments_without_duplicates
    WHERE author != '[deleted]'
    GROUP BY author
),
num_submissions_cte AS (
    SELECT author, COUNT(*) AS num_submissions
    FROM submissions_without_duplicates
    WHERE author != '[deleted]'
    GROUP BY author
)
SELECT
    COALESCE(c.author, s.author) AS author,
    COALESCE(c.num_comments, 0) AS num_comments,
    COALESCE(s.num_submissions, 0) AS num_submissions,
    COALESCE(c.num_comments, 0) + COALESCE(s.num_submissions, 0) AS total_submissions_and_comments
FROM num_comments_cte AS c
FULL OUTER JOIN num_submissions_cte AS s
ON c.author = s.author
ORDER BY total_submissions_and_comments DESC
LIMIT 10;
```

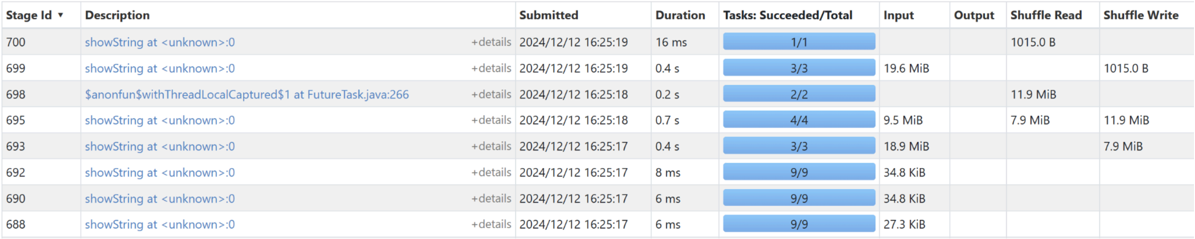
```
 * postgresql://user:***@bachtran.dev:5432/reddit
10 rows affected.
```



```
 * postgresql://user:***@bachtran.dev:5432/reddit
10 rows affected.
```

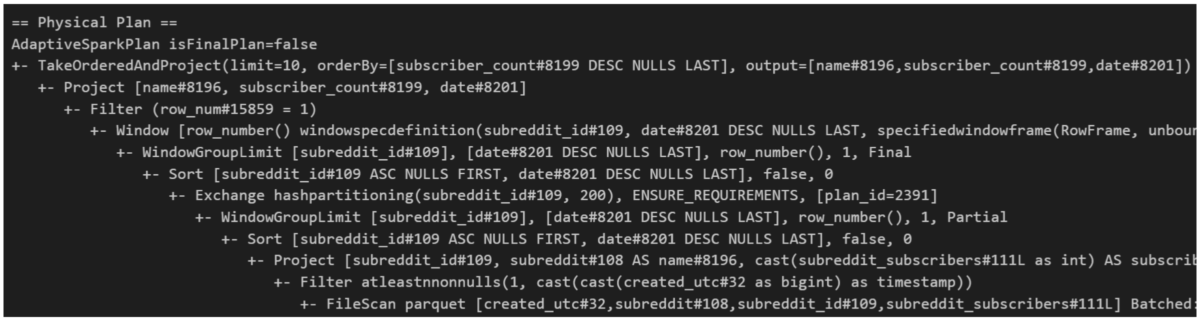
author	num_comments	num_submissions	total_submissions_and_comments
TheGuru12	1	253735	253736
AutoModerator	30449	3081	33530
One_Giant_Nostril	5004	3905	8909
saasfin	544	7677	8221
h3xadecimal2	0	6551	6551
Will_Power	5096	120	5216
Infamous_Employer_85	5067	10	5077
Tpaine63	4888	148	5036
NewyBluey	4992	7	4999
technologyisnatural	3596	171	3767

Example of how Spark displays tasks



Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
700	showString at <unknown>:0	+details	2024/12/12 16:25:19	16 ms	1/1			1015.0 B	
699	showString at <unknown>:0	+details	2024/12/12 16:25:19	0.4 s	3/3	19.6 MiB			1015.0 B
698	$anonfun$withThreadLocalCaptured$1 at FutureTask.java:266	+details	2024/12/12 16:25:18	0.2 s	2/2			11.9 MiB	
695	showString at <unknown>:0	+details	2024/12/12 16:25:18	0.7 s	4/4	9.5 MiB		7.9 MiB	11.9 MiB
693	showString at <unknown>:0	+details	2024/12/12 16:25:17	0.4 s	3/3	18.9 MiB			7.9 MiB
692	showString at <unknown>:0	+details	2024/12/12 16:25:17	8 ms	9/9	34.8 KiB			
690	showString at <unknown>:0	+details	2024/12/12 16:25:17	6 ms	9/9	34.8 KiB			
688	showString at <unknown>:0	+details	2024/12/12 16:25:17	6 ms	9/9	27.3 KiB			

Example of a physical plan for subreddits with most subscribers



```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- TakeOrderedAndProject(limit=10, orderBy=[subscriber_count#8199 DESC NULLS LAST], output=[name#8196,subscriber_count#8199,date#8201])
   +- Project [name#8196, subscriber_count#8199, date#8201]
      +- Filter (row_num#15859 = 1)
         +- Window [row_number() windowspecdefinition(subreddit_id#109, date#8201 DESC NULLS LAST, specifiedwindowframe(RowFrame, unbour
            +- WindowGroupLimit [subreddit_id#109], [date#8201 DESC NULLS LAST], row_number(), 1, Final
               +- Sort [subreddit_id#109 ASC NULLS FIRST, date#8201 DESC NULLS LAST], false, 0
                  +- Exchange hashpartitioning(subreddit_id#109, 200), ENSURE_REQUIREMENTS, [plan_id=2391]
                     +- WindowGroupLimit [subreddit_id#109], [date#8201 DESC NULLS LAST], row_number(), 1, Partial
                        +- Sort [subreddit_id#109 ASC NULLS FIRST, date#8201 DESC NULLS LAST], false, 0
                           +- Project [subreddit_id#109, subreddit#108 AS name#8196, cast(subreddit_subscribers#111L as int) AS subscrib
                              +- Filter atleastnnonnulls(1, cast(cast(created_utc#32 as bigint) as timestamp))
                                 +- FileScan parquet [created_utc#32,subreddit#108,subreddit_id#109,subreddit_subscribers#111L] Batched:
```