M2 Statistics Econometrics / Data Science for Social Sciences

Data Bases project

Reddit Comment Analysis: from platform-wide trends to the anatomy of a viral post

Authors:

BASTIEN Emma - 21900089
GOARDET Marie - 21900104
PIZZETTA Nathan - 21900028

Contents

0 Introduction

Reddit is one of the most active and content-rich platforms on the internet, where users engage in discussions across a wide range of communities known as subreddits. Among these, AskReddit is one of the most popular and dynamic, built entirely around user-generated questions and open conversations.

In this project, we focus on a subset of Reddit data: all comments posted in the AskReddit subreddit during the month of May 2015. Despite this restriction, the dataset remains rich, with thousands of posts and millions of user comments. Each comment is associated with metadata such as scores, authors, timestamps, and reply structures, allowing for both large-scale and fine-grained analysis.

Our central question is: "What factors drive user engagement—measured by scores, controversies, and sentiment—within the AskReddit community, and how can we model and analyze a month's worth of data to uncover both broad trends and in-depth insights on a single highly active post?"

To answer this question, our work follows a five-part structure:
1. Creating the database : we begin by modeling AskReddit's data as a relational schema, including comments, authors, subreddits, scores systems, and relationships between comments and parent posts.

2. Loading the database : we then populate this structure with the AskReddit-May-2015 datasets, ensuring data consistency and referential integrity to allow efficient querying.

3. Initial exploration of the database : we perform an exploratory analysis to understand overall comment activity: volume of comments, comment density per post, posting rhythms.

4. Detailed exploration of the scores : we examine how scores vary across comments, including statistical differences between controversial and non-controversial comments, and explore what drives high or low scoring behavior.

5. Focus on a specific post : finally, we zoom in on one specific post that statistically stands out, based on its total number of comments and overall score. We analyze this post in detail as a case study of sentimental analysis.

1 Creating the database

To analyze the AskReddit data efficiently, we first need to structure it properly in a relational database. The original data is provided as 11 separate CSV files, each corresponding to the content of one of the tables in our schema. In this section, we outline the design choices and the steps taken to create the database. In the next part, we will load the data from these CSV files into the database after the schema has been created.

1.1 Schemas

The data will be structured based on an entity-relationship model (ER model), which reflects the relationships between the different entities such as authors, comments, and scores. This model ensures that the database is normalized, reducing redundancy and improving query performance.

Entity-Relationship Schema

The following ER diagram shows the entities and relationships of our database:

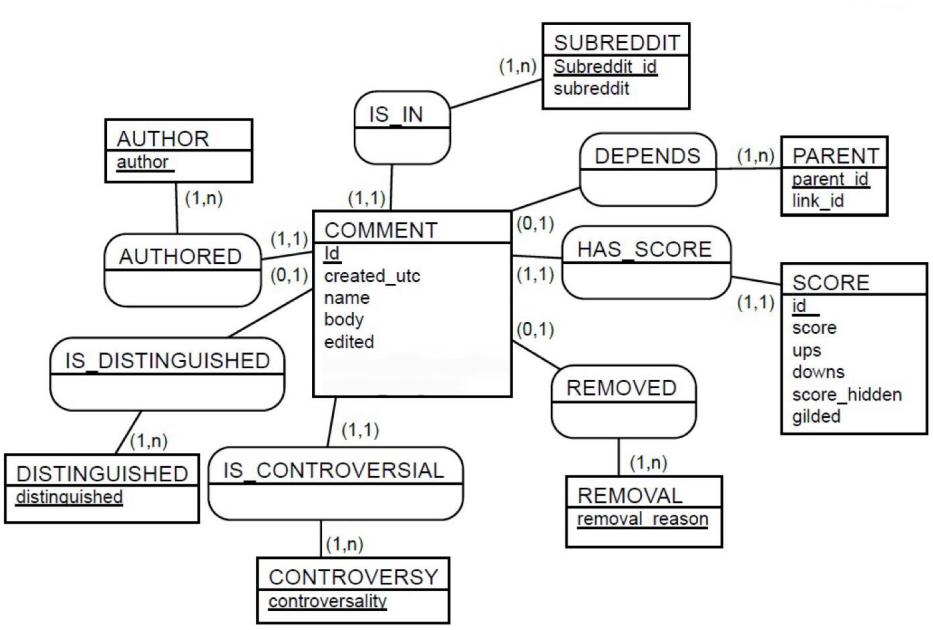

Figure 1: Entity/Relationship diagram of the database

The entities (shown in the squared boxes) represent the core tables :

- **AUTHOR** - Stores information about the user who authored a comment.
- **COMMENT** - Contains details of the comment (time of creation, body of the comment...)
- **SUBREDDIT** - Represents the subreddit to which a comment belongs (here in our case, AskReddit)
- **SCORE** - Holds data on comment scoring (upvotes, downvotes, hidden status).
- **PARENT** - Captures the parent-child relationship between comments.
- **DISTINGUISHED** - Indicates if a comment is distinguished (special, moderator...).
- **CONTROVERSY** - Holds data about the controversy level of a comment.
- **REMOVAL** - Stores information about why a comment was removed.

Then in this ER schema, we have the linking tables (represented by the rounded boxes) that define the relationships between the different entities.

Finally, the cardinalities define how many records in one table can relate to records in another table with the following matching :

- **(1,1)** - Exactly one record must match.
- **(1,n)** - One record in the first table can match multiple records in the second table.
- **(0,1)** - A record in the first table may or may not have a matching record in the second table.

Here are some examples from the diagram :

- A **COMMENT** must have exactly one **AUTHOR** → (1,1).
- A **COMMENT** can have one or no **PARENT** → (0,1).
- A **COMMENT** must have exactly one **SCORE** and a **SCORE** must belong to exactly one **COMMENT** → (1,1).
- A **SUBREDDIT** can have multiple comments → (1,n).

Relational Schema

Now, the relational schema translates the ER model into a database schema using tables (core and linking ones) and keys as follows :

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

Where the underlined field(s) in each table indicates its primary key, which uniquely identifies records within a table and the # symbol denotes a foreign key, which is a field in one table that refers to the primary key in another table, establishing a relationship between the two.
Also, we will explain in the following parts why some of the linking tables appearing in the ER diagram are not considered in the relational schema nor will be created in the database.

Process Schema

The following diagram illustrates the process followed to create the database. Our goal here is to show how our SQL code for the creation of the database was executed at a high level before going into the details in the following part.

Figure 2: Process Schema

First, we established a connection between our Python script and the Oracle database with the following code :

Listing 1: Connection to Oracle via Python.

```
oradb_username = ""
oradb_password = ORADB_PWD
oradb_hostname = ""
oradb_port = ""
oradb_sid = ""

dsn = f"(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST={oradb_hostname
   })(PORT={oradb_port}))(CONNECT_DATA=(SID={oradb_sid})))"

oradb_connection = oracledb.connect(
    user=oradb_username,
    password=oradb_password,
    dsn=dsn
    )
oradb_cursor = oradb_connection.cursor()
print(oradb_connection)
```

From there, all SQL code, including table creation and querying, was executed directly from Python using a cursor object. Here is an example of how it worked in our notebook:

Listing 2: Example of how we executed our queries in practice.

```
query = """
CREATE TABLE AUTHOR (
        author VARCHAR2(50) PRIMARY KEY
        )
)"""
oradb_cursor.execute(query)
```

This approach allowed us to encapsulate all database interactions within Python code, while still using SQL queries that were executed via Oracle. From now on, we will only include in this report the SQL scripts for our queries, syntactically colored. Please note that these queries were executed in a Jupyter notebook, but they can also be run directly in the Oracle SQL Developer interface, which we also used. In the following sections, we will present the query results either from the SQL Developer interface, from the notebook execution, or sometimes summarize them in a report table (for outputs too long to screenshot), fo the sake of space efficiency management of this report.

1.2 Data Dictionary

The data dictionary describes each column in detail, including the data type and the purpose of the variable.

Table 1: Full Data Dictionary

Table	Variable	Type	Description
AUTHOR	author	String	Account name of the comment author. Null if deleted.
COMMENTS	id	String	Unique comment identifier.
	created_utc	Integer	Unix timestamp of comment creation.
	name	String	Full name of the comment (e.g., t1_abc).
	body	String	Raw textual content of the comment.
	edited	Integer	0 if never edited, otherwise timestamp.
	author_flair_css_class	String	Author's flair style (subreddit-specific).
	author_flair_text	String	Author's flair text (subreddit-specific).
	subreddit_id	String	ID of the subreddit this comment belongs to.
SCORE	id	String	Comment ID (foreign key).
	score	Integer	Net score = ups - downs.
	ups	Integer	Number of upvotes.
	downs	Integer	Number of downvotes (always 0).
	score_hidden	Integer	Whether the score was hidden.
	gilded	Integer	Number of Reddit Gold awards.
PARENT	parent_id	String	ID of the parent comment or post.
	link_id	String	ID of the original submission (thread).
SUBREDDIT	subreddit_id	String	Unique identifier of the subreddit.
	subreddit	String	Name of the subreddit (e.g., AskReddit).
CONTROVERSY	controversiality	Integer	Flag: 1 if controversial, else 0.
REMOVAL	removal_reason	String	Reason for removal, if applicable.
IS_DISTINGUISHED	id	String	Comment ID.
	distinguished	String	Type of distinction (e.g., moderator, special).
REMOVED	id	String	ID of the removed comment.
	removal_reason	String	Reason this comment was removed.
DEPENDS	id	String	ID of the comment.
	parent_id	String	ID of the parent comment or post.
DISTINGUISHED	distinguished	String	Type of distinction (e.g., moderator, special).

1.3 Database Modelling Choices and Assumptions

In the design and implementation of the database, several key modeling decisions and assumptions were made to balance technical constraints, data integrity, and query performance, especially given the large volume of data involved. Below are the key points regarding these choices:

- **Entities with a Single Attribute**
  For technical reasons, certain entities in the database schema contain only one attribute. Examples of such entities include `author`, `distinguished`, and `controversy`. While this is generally not considered an optimal modeling practice, it was deemed acceptable in this case due to the nature of the data and the volume involved.

- **Relationships with (1,1) Cardinalities**
  In cases where relationships have (1,1) cardinalities on both sides, the typical approach in relational modeling is to combine the related entities into a single table. However, in our specific case, entities such as the `COMMENTS` and `SCORE` tables were modeled as two separate tables. This decision was made intentionally to reduce the size of each individual table, thereby improving the speed of query execution when processing them separately.

- **Excluded Attributes from the Comments Table**
  Certain attributes in the `COMMENTS` table, namely `author_flair_css_class` and `author_flair_text`, were not included in the final database schema. This decision was based on the fact that these columns only contained NA values in the original dataset (CSV files). As a result, these attributes were deemed redundant for the purpose of this analysis and were omitted to reduce unnecessary data storage. If in the future the data in these columns becomes more relevant, they could be added to the schema.

- **Creation of Linking Tables Based on Cardinality**
  We only created the linking tables that correspond to relationships with a (1,n) cardinality, as indicated in the relational schema. Linking tables for (0,1) or (1,1) relationships were intentionally excluded, as they are not required to maintain data integrity in the current context. For example, (1,1) relationships are typically better represented by placing the foreign key directly in one of the tables involved in the relationship. By excluding these unnecessary linking tables, we avoid unnecessary complexity and improve the efficiency of the schema.

- **Retention of Removed and Removal Tables**
  Despite the `REMOVED` and `REMOVAL` CSV files being empty, they were retained in the database schema for completeness purposes. Even though they are not currently used in our analysis, these tables may hold potential future value. Retaining these tables ensures that the database schema remains consistent with the original data structure, which could be important for future data migrations, data verification, or other analyses. Additionally, if there were another reason for the removal of entries beyond simply being "legal" it might be beneficial to keep the associated tables for completeness.

- **Subreddit Table - Minimal Data**
  The `SUBREDDIT` table contains only a single entry, "AskReddit". Although this

table might appear unlikely to be used in the current analysis, it was retained in the database model. We decided to keep this table in case if additional subreddits be included in the dataset at a later stage, the structure is already in place. This ensures that the database schema can easily accommodate future changes without requiring major modifications.

In conclusion, these modeling decisions ensures that the database remains efficient while maintaining flexibility for future data expansion and modifications.

1.4 SQL Code for Database Creation

Now, let us deep dive in the SQL code that was used to create the database and all its tables:

Listing 3: Creation of the database structure

```
CREATE TABLE AUTHOR (
    author VARCHAR2(50) PRIMARY KEY
        );

CREATE TABLE DISTINGUISHED (
    distinguished VARCHAR2(50) PRIMARY KEY
        );

CREATE TABLE CONTROVERSY (
    controversiality NUMBER(1) PRIMARY KEY
        );

CREATE TABLE REMOVAL (
    removal_reason VARCHAR2(100) PRIMARY KEY
        );

CREATE TABLE SUBREDDIT (
    subreddit_id VARCHAR2(20) PRIMARY KEY,
    subreddit    VARCHAR2(100)
        );

CREATE TABLE PARENT (
    parent_id VARCHAR2(20) PRIMARY KEY,
    link_id   VARCHAR2(20)
        );

CREATE TABLE COMMENTS (
    id                      VARCHAR2(20) PRIMARY KEY,
    created_utc             NUMBER(19),
    name                    VARCHAR2(50),
    body                    CLOB,
    edited                  NUMBER,
    author                  VARCHAR2(50),
    controversiality        NUMBER(1),
    subreddit_id            VARCHAR2(20),
```

```
    FOREIGN KEY (author) REFERENCES AUTHOR(author),
    FOREIGN KEY (controversiality) REFERENCES CONTROVERSY(controversiality),
    FOREIGN KEY (subreddit_id) REFERENCES SUBREDDIT(subreddit_id)
        );

CREATE TABLE SCORE (
    id            VARCHAR2(20) PRIMARY KEY,
    score         NUMBER,
    ups           NUMBER,
    downs         NUMBER,
    score_hidden  NUMBER,
    gilded        NUMBER,
    FOREIGN KEY (id) REFERENCES COMMENTS(id)
        );

CREATE TABLE IS_DISTINGUISHED (
    id             VARCHAR2(20),
    distinguished  VARCHAR2(50),
    PRIMARY KEY (id, distinguished),
    FOREIGN KEY (id) REFERENCES COMMENTS(id),
    FOREIGN KEY (distinguished) REFERENCES DISTINGUISHED(distinguished)
        );

CREATE TABLE REMOVED (
    id              VARCHAR2(20),
    removal_reason  VARCHAR2(100),
    PRIMARY KEY(id, removal_reason),
    FOREIGN KEY (id) REFERENCES COMMENTS(id),
    FOREIGN KEY (removal_reason) REFERENCES REMOVAL(removal_reason)
        );

CREATE TABLE DEPENDS (
    id          VARCHAR2(20),
    parent_id   VARCHAR2(20),
    PRIMARY KEY(id, parent_id),
    FOREIGN KEY (id) REFERENCES COMMENTS(id),
    FOREIGN KEY (parent_id) REFERENCES PARENT(parent_id)
        );
```

Overall, for the data types of each variable, we followed the specifications outlined in the data dictionary (type column). The field type `NUMBER` was used for integer values, while `VARCHAR2` was applied for string values, with the length of the field varying based on the specific requirements of each attribute. The only exception to the standard data types was the use of the `CLOB` field for the `body` attribute of the `COMMENTS` table. This was chosen due to the potentially large size of the text data stored in this attribute, as comments can vary significantly in length and may exceed the limits of standard string types.

2 Loading the Database

Once the database schema is created, the next step is to load the data from the CSV files into the database. The data was processed and cleaned using Python before being inserted into the Oracle database.

2.1 Schema and Code

The following schema shows the data loading process:

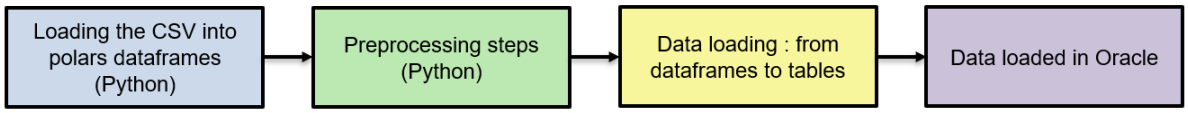

Figure 3: Data Loading schema

We will now go into the details on how those different steps were implemented.

2.2 Preprocessing Steps

As explained in the process schema above, we started by importing our CSV files into Python using the `polars` library, which allowed us to efficiently handle large tabular data. Then, before loading the data into the database, the following preprocessing steps were performed directly on the dataframes in Python to handle missing values:

1. dropped the columns `author flair css class` and `author flair text` from the Comments dataframe (named `df comments`) as explained earlier:

Listing 4: Dropping 2 columns of *df comments* in Python.

```
dataframes["comments"] = dataframes["comments"].drop(["
    author_flair_css_class", "author_flair_text"])
```

2. replaced the missing values for both `Distinguished` and the `Is Distinguished` dataframes by the string "Not Distinguished". Indeed, as indicated in the data dictionary, a null value for the distinguished attribute simply corresponds to being 'Not distinguished". Here is an example of the code used for the `Distinguished` dataframe:

Listing 5: Handling missing values in the *df distinguished* in Python.

```
dataframes["distinguished"] = dataframes["distinguished"].
    with_columns([
pl.col(c).fill_null("Not distinguished") for c in
    dataframes["distinguished"].columns
])
```

2.3 Data Loading

To load data from Python dataframes into an Oracle database, the *"write to db manually"* function was implemented. This function automates the insertion process and handles large datasets efficiently. Here is its python code:

Listing 6: definition of the write to db manually function in Python.

```
def write_to_db_manually(df_in:pl.DataFrame, conn, table_name,
    batch_size=10):
    # Get a cursor
    cursor = conn.cursor()
    columns = ",".join(df_in.columns)
    col_idx = ",".join([f":{item}" for item in df_in.columns])
    sql     = f"INSERT INTO {table_name}({columns}) VALUES ({
        col_idx})"
    rows = df_in.to_struct().to_list()
    cursor.executemany(sql, rows)
    conn.commit()
```

More precisely, the function takes the following parameters:

- *df in* which is the input polars DataFrame containing the data
- *conn*, corresponding to the active Oracle database connection (in our case, the one we established in the python script at the beginning of our analysis)
- *table name* which corresponds to the targeted table name in the database (ie. amongst those we created in the first part of this report).
- *batch size* which is the number of rows inserted per batch (default = 10).

The function creates a SQL INSERT statement dynamically based on the dataframe's columns. The data is converted to a list of tuples and inserted using cursor.executemany for batch processing. The loading is committed with conn.commit() to save the changes.

Now the next steps consist in using this function to load the data into the tables, by doing so dataframe by dataframe as follows:

Listing 7: From dataframes to tables of the database : loading of the data in Python.

```
for name, df in dataframes.items():
    name = name.upper()
    try:
        write_to_db_manually(df, oradb_connection, name)
        print(f"Loaded {len(df)} rows into {name}.")
    except oracledb.DatabaseError as db_err:
        # Handle duplicates or existing data
        error_code = db_err.args[0].code if db_err.args else None
        print(f"DatabaseError on {name} (Code={error_code}): {
            db_err}")
        pass
    except Exception as e:
        # Catch any other error
        print(f"Unexpected error on {name}: {e}")
        pass
```

Now, let us check that, after loading, we get the same number of rows in the CSVs (ie. loaded in the polars dataframes) and in the tables.

First, we implemented Python code to get the number of rows for each one of the dataframes and got the following results :

```
DataFrame      Rows
------------------------
authors        570735
comments       4234970
controversy    2
depends        4234970
distinguished  3
is_distinguished4234970
parent         1464558
removal        1
removed        0
score          4234970
subreddit      1
```

Figure 4: Number of rows of each CSV file (Python dataframes)

Now to get the number of rows of each one of the tables we created and in which we loaded the data from the CSV files, we implemented the following queries :

Listing 8: Counting number of rows in each table

```
SELECT COUNT(*) FROM author;
SELECT COUNT(*) FROM distinguished;
SELECT COUNT(*) FROM controversy;
SELECT COUNT(*) FROM removal;
SELECT COUNT(*) FROM removed;
SELECT COUNT(*) FROM subreddit;
SELECT COUNT(*) FROM parent;
SELECT COUNT(*) FROM comments;
SELECT COUNT(*) FROM score;
SELECT COUNT(*) FROM is_distinguished;
SELECT COUNT(*) FROM depends;
```

and get the same results:

```
Number of rows in author: 570735
Number of rows in distinguished: 3
Number of rows in controversy: 2
Number of rows in removal: 1
Number of rows in removed: 0
Number of rows in subreddit: 1
Number of rows in parent: 1464558
Number of rows in comments: 4234970
Number of rows in score: 4234970
Number of rows in is_distinguished: 4234970
Number of rows in depends: 4234970
```

Figure 5: Number of rows of each oracle table

2.4 What Was Not Loaded

Now, in the previous part, you might have noticed that we do not execute the *write to db manually* function for the `REMOVED` table. Indeed, as stated before, the CSV file for the removed-related data was completely empty. Thus no data could be loaded in the `REMOVED` table and its number of rows is thus equal to 0.

3 Initial exploration of the database

First, before performing SQL queries, let us note that we have used JOIN operations instead of directly selecting columns from a table like `COMMENTS` in order to significantly improve both efficiency and accuracy. For example, instead of directly accessing the *author* column from the `COMMENTS` table, it's better to join the `COMMENTS` table with the `AUTHOR` table using a common key (like author). This is because directly querying the *author* column involves text-based searches, which are generally slower and less efficient. Text searches require scanning the entire column, which can be resource-intensive, especially with a large dataset as ours. On the other hand, performing a JOIN relies on indexed keys, which allows the database to quickly locate and match records using optimized indexing structures. This reduces the search space and improves query execution time. JOINS also help maintain data integrity by ensuring that the author information is consistent and up-to-date, since the data is stored in a single place (the `AUTHOR` table) rather than being duplicated across multiple tables. This makes the database more efficient and scalable.

Before performing any analysis, it is important to first get a sense of the content of each table. This step allows us to better understand the structure of the database, verify the integrity of the data, and identify potentially useful attributes for our later queries.

To do so, we begin by inspecting a few rows from each table. For example, to inspect the `AUTHOR` table, we use the following query:

Listing 9: Inspecting the first 5 rows from the AUTHOR table

```
SELECT *
  FROM author
FETCH FIRST 5 ROWS ONLY;
```

	AUTHOR
1	jesse9o3
2	beltfedshooter
3	InterimFatGuy
4	JuanTutrego
5	dcblackbelt

Figure 6: First 5 rows from the AUTHOR table

The table obtains displays 5 authors that jesse9o3, beltfedshooter, InterimFatGuy, JuanTutrego and dcblackbelt.

We implemented the same query for all tables and the following table summarizes the structure and sample values of all relevant tables in our database:

Table	Description / Sample Values
`author`	List of author usernames, e.g. `jesse9o3`, `beltfedshooter`
`distinguished`	Roles assigned to comments: `Not distinguished`, `moderator`, `special`
`controversy`	Binary controversiality flag: `0` or `1`
`removal`	Reason for comment removal, e.g. `legal`
`subreddit`	Here we only have `AskReddit` and its ID
`parent`	The post id `parent_id` and its `link_id` to access the post online
`comment`	Comments data: `id`, `created_utc` the time of creation of the comment, `name`, `body`, `edited` the time of editionauthor, `controversiality`
`score`	score data: `score`, `ups`, `downs`, `score_hidden`, `gilded`
`is_distinguished`	Links `id` to `distinguished` status
`depends`	Links `id` (comment) to `parent_id` (reply structure)

Moreover, we can have access to the type of the columns of each table :

Listing 10: Retrieving column names and data types

```
SELECT column_name, data_type, nullable
  FROM all_tab_columns
 WHERE table_name = 'AUTHOR';

SELECT column_name, data_type, nullable
  FROM all_tab_columns
 WHERE table_name = 'DISTINGUISHED';

SELECT column_name, data_type, nullable
  FROM all_tab_columns
 WHERE table_name = 'CONTROVERSY';

SELECT column_name, data_type, nullable
  FROM all_tab_columns
 WHERE table_name = 'REMOVAL';

SELECT column_name, data_type, nullable
  FROM all_tab_columns
 WHERE table_name = 'SUBREDDIT';

SELECT column_name, data_type, nullable
  FROM all_tab_columns
 WHERE table_name = 'PARENT';
```

```
SELECT column_name, data_type, nullable
  FROM all_tab_columns
 WHERE table_name = 'COMMENTS';

SELECT column_name, data_type, nullable
  FROM all_tab_columns
 WHERE table_name = 'SCORE';

SELECT column_name, data_type, nullable
  FROM all_tab_columns
 WHERE table_name = 'IS_DISTINGUISHED';

SELECT column_name, data_type, nullable
  FROM all_tab_columns
 WHERE table_name = 'DEPENDS';
```

Table	Column	Data Type
AUTHOR	AUTHOR	VARCHAR2
DISTINGUISHED	DISTINGUISHED	VARCHAR2
CONTROVERSY	CONTROVERSIALITY	NUMBER
REMOVAL	REMOVAL_REASON	VARCHAR2
SUBREDDIT	SUBREDDIT_ID SUBREDDIT	VARCHAR2 VARCHAR2
PARENT	PARENT_ID LINK_ID	VARCHAR2 VARCHAR2
COMMENTS	ID CREATED_UTC NAME BODY EDITED AUTHOR CONTROVERSIALITY SUBREDDIT_ID	VARCHAR2 NUMBER VARCHAR2 CLOB NUMBER VARCHAR2 NUMBER VARCHAR2
SCORE	ID SCORE UPS DOWNS SCORE_HIDDEN GILDED	VARCHAR2 NUMBER NUMBER NUMBER NUMBER NUMBER
IS_DISTINGUISHED	ID DISTINGUISHED	VARCHAR2 VARCHAR2
DEPENDS	ID PARENT_ID	VARCHAR2 VARCHAR2

Another important information is the number of rows in each table :

Listing 11: Counting number of rows in each table

```
SELECT COUNT(*) FROM author;
SELECT COUNT(*) FROM distinguished;
SELECT COUNT(*) FROM controversy;
SELECT COUNT(*) FROM removal;
SELECT COUNT(*) FROM removed;
SELECT COUNT(*) FROM subreddit;
```

```
SELECT COUNT(*) FROM parent;
SELECT COUNT(*) FROM comments;
SELECT COUNT(*) FROM score;
SELECT COUNT(*) FROM is_distinguished;
SELECT COUNT(*) FROM depends;
```

```
Number of rows in author: 570735
Number of rows in distinguished: 3
Number of rows in controversy: 2
Number of rows in removal: 1
Number of rows in removed: 0
Number of rows in subreddit: 1
Number of rows in parent: 1464558
Number of rows in comments: 4234970
Number of rows in score: 4234970
Number of rows in is_distinguished: 4234970
Number of rows in depends: 4234970
```

Figure 7: Number of rows of each oracle table

To begin exploring the dynamics of user activity, we plotted the number of comments posted each day throughout May 2015 in the AskReddit subreddit. This visualization provides a high-level view of participation trends over time, helping us identify potential spikes in engagement or daily posting rhythms.

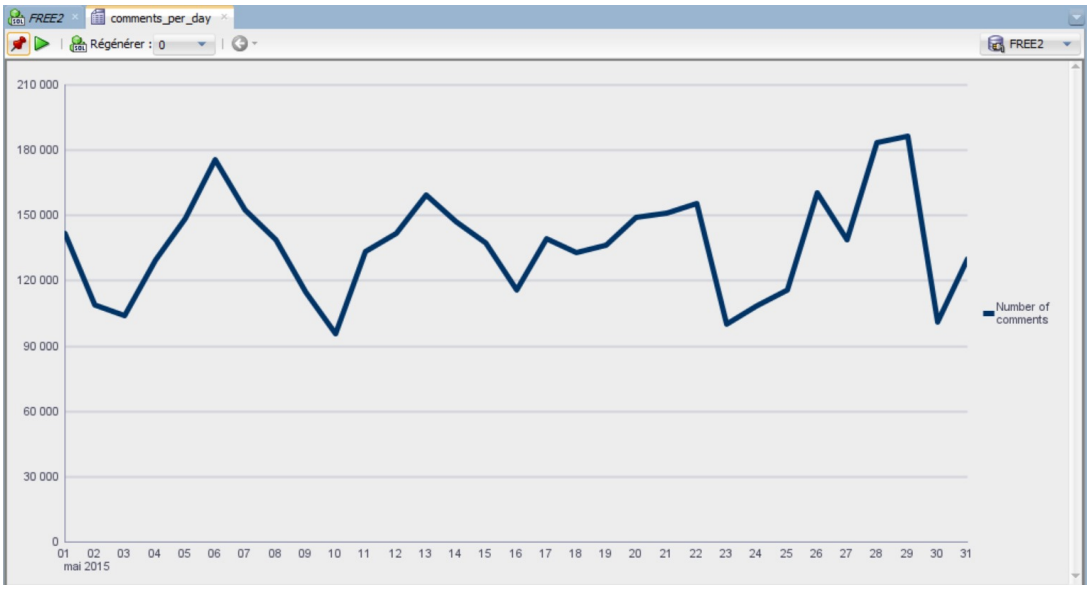

Figure 8: Number of comments per day in the AskReddit subreddit (May 2015)

This figure is the result of the following query that we have used directly in the visualization tools of SQL Developer :

Listing 12: Number of comments posted each day throughout May 2015 in the AskReddit subreddit

```
SELECT
```

```
  TRUNC(DATE '1970-01-01' + NUMTODSINTERVAL(created_utc, 'SECOND')) AS "Date",
  'Number of comments' AS "Serie",
  COUNT(*) AS "Valeur"
FROM comments
GROUP BY TRUNC(DATE '1970-01-01' + NUMTODSINTERVAL(created_utc, 'SECOND'))
ORDER BY "Date";
```

After analyzing general trends in comment activity, we now turn to a more detailed examination of specific elements within the dataset. Understanding what types of comments and users stand out the most in terms of visibility, controversy, and engagement appears essential to better understand the structure and dynamics of the database. First, it is essential to take a closer look at the most prominent authors. One effective way to identify the top 5 is to consider the users who have posted the highest number of comments.

Listing 13: Top 5 most active authors by number of comments

```
SELECT a.author,
       COUNT(*) AS number_of_comments
  FROM comments c
  JOIN author a ON a.author = c.author
 GROUP BY a.author
 ORDER BY number_of_comments DESC
 FETCH FIRST 5 ROWS ONLY;
```

	AUTHOR	NUMBER_OF_COMMENTS
1	[deleted]	312007
2	AutoModerator	36910
3	Late_Night_Grumbler	8298
4	BiagioLargo	5843
5	-_-Equinox666-_-	2989

Here, we can see that the top 1 author corresponds to "[deleted]", which seems odd. We suspect that it could be an aggregation of all authors that posted during our period but who were deleted at some points. Also, the 2nd author that commented the most, "AutoModerator", if we inspect the body of its comment, seems to be a bot. All in all, to have better insights for our analysis on the real authors that commented, we exclude these two first authors.

	AUTHOR	NUMBER_OF_COMMENTS
1	Late_Night_Grumbler	8298
2	BiagioLargo	5843
3	-_-Equinox666-_-	2989
4	KubrickIsMyCopilot	2601
5	Megaross	2479

We now turn our attention to the most controversial comments in the dataset. Highlighting the top 5 allows us to better understand the kinds of content that sparked the most disagreement or polarized reactions within the community.

Listing 14: Counting the number of controversial comments

```
SELECT COUNT(*) AS total_controversial_comments
  FROM comments c
  JOIN controversy co ON c.controversiality = co.controversiality
 WHERE co.controversiality = 1;
```

Out of a total of 4,234,970 comments in the dataset, only 52,218 were marked as controversial. This represents approximately 1.23% of all comments. Such a low proportion suggests that controversial comments are relatively rare within the AskReddit subreddit during May 2015.

Following the same logic, we also examined the number of comments that were edited after being posted. Out of a total of 4,234,970 comments, 80,620 were edited, representing approximately 1.90% of all comments.

In addition to controversy, it's also insightful to examine which users received a formal distinction from the platform. Among these, the "special" status stands out as a rare marker. Identifying the best whose contributions were most frequently marked as "special" provides a way to highlight users who may have had a significant or recognized impact within the community. Same logic for the other distinctions.

Listing 15: Top 2 commenters by type of distinction

```
SELECT
    a.author,
    COUNT(
        CASE
            WHEN d.distinguished = 'special'
            THEN 1
        END
    ) AS special_comments,
    COUNT(
        CASE
            WHEN d.distinguished = 'moderator'
            THEN 1
        END
    ) AS moderator_comments,
    COUNT(
        CASE
            WHEN d.distinguished = 'Not distinguished'
            THEN 1
        END
    ) AS not_distinguished_comments
FROM comments c
JOIN author a ON c.author = a.author
JOIN is_distinguished id ON c.id = id.id
JOIN distinguished d ON id.distinguished = d.distinguished
GROUP BY a.author
ORDER BY special_comments DESC
FETCH FIRST 2 ROWS ONLY;
```

	AUTHOR	SPECIAL_COMMENTS	MODERATOR_COMMENTS	NOT_DISTINGUISHED_COMMENTS
1	kn0thing	4	0	11

After identifying standout users and specific comment characteristics, we now turn our attention to a more focused analysis of one of the most informative variables in the dataset: the comment score and author karma (score for an author). The score is a central indicator of a comment's visibility and reception within the community, as it reflects how users voted—positively or negatively—on a given contribution. Understanding how scores are distributed and what factors influence them is essential to interpreting engagement dynamics, identifying impactful content, and ultimately selecting a relevant subset of the data for more detailed analysis.

4 Detailed exploration of the scores

In this section, we analyze score distributions, explore the role of controversiality, and compare groups using statistical tests to detect significant differences.

In the Reddit platform, each comment is assigned a score that reflects the net reception it has received from the community. This score is typically computed as the difference between upvotes and downvotes:

score = ups - downs

However, in our dataset, we observed that all downvote values are equal to zero. As a result, the score of each comment is effectively equal to the number of upvotes:

score = ups

Listing 16: Checking whether all downvote values are zero

```
SELECT
    SUM(downs) AS total_downvotes,
    CASE
        WHEN SUM(
            CASE
                WHEN downs != 0
                THEN 1
                ELSE 0
            END
        ) = 0
        THEN 'All downs are zero'
        ELSE 'Some downs are not zero'
    END AS check_result
FROM SCORE;
```

	TOTAL_DOWNVOTES	CHECK_RESULT
1	0	All downs are zero

Listing 17: Checking if comment score equals number of upvotes

```
SELECT
  COUNT(*) AS total_rows,
  SUM(CASE WHEN score = ups THEN 1 ELSE 0 END) AS same_rows,
  SUM(CASE WHEN score != ups THEN 1 ELSE 0 END) AS different_rows
FROM SCORE;
```

	TOTAL_ROWS	SAME_ROWS	DIFFERENT_ROWS
1	4234970	4234970	0

This simplification allows us to directly interpret the score as a measure of positive engagement, without needing to account for negative feedback. It also raises interesting questions about the availability or filtering of downvote data in this particular dataset.

To better understand which types of comments receive the most positive attention from the community, we first examine the top-scoring comments in the dataset. These comments represent the most upvoted content and provide insight into what kind of contributions are valued by users in the AskReddit subreddit.

Listing 18: Top 5 highest-scoring comments and score visibility

```
SELECT c.id,
       s.score,
       s.score_hidden
  FROM score s
  JOIN comments c ON s.id = c.id
 ORDER BY s.score DESC
 FETCH FIRST 5 ROWS ONLY;
```

	ID	SCORE	SCORE_HIDDEN
1	cr56nez	6761	0
2	cr5gn52	6736	0
3	cr3imvn	5992	0
4	cr8p5z9	5927	0
5	cqxx3fj	5849	0

All of the top 5 highest-scoring comments in the dataset have their scores set as visible (`score hidden = 0`). This suggests that highly upvoted comments are generally not hidden, reinforcing their visibility and further encouraging engagement.

Beyond the popularity of individual comments, it is also important to consider the cumulative impact of each user. On Reddit, this is commonly referred to as karma, which corresponds to the sum of all scores received across an author's comments. A high karma score suggests that a user's contributions are generally well received by the community.

However, this metric alone can be misleading: a user who posts frequently may accumulate a high total score simply through volume rather than through the consistent

quality or impact of their comments. Therefore, to better assess true influence, it is important to contextualize karma by considering the number of comments posted. This allows us to distinguish between highly active users and those whose individual comments have exceptional reach or approval.

In what follows, we identify the top 5 authors with the highest total karma in May 2015.

Listing 19: Top 5 authors with the highest total karma and average karma per comment

```
SELECT
    a.author,
    SUM(s.score) AS total_karma,
    COUNT(c.id) AS total_comments,
    ROUND(SUM(s.score) / COUNT(c.id), 2) AS average_karma
  FROM comments c
  JOIN author a ON c.author = a.author
  JOIN score s ON c.id = s.id
 GROUP BY a.author
 ORDER BY total_karma DESC
 FETCH FIRST 5 ROWS ONLY;
```

	AUTHOR	TOTAL_KARMA	TOTAL_COMMENTS	AVERAGE_KARMA
1	[deleted]	1353042	312007	4,34
2	Donald_Keyman	227175	312	728,13
3	kyle8998	100898	860	117,32
4	Naweezy	74654	69	1081,94
5	Irememberedmypw	65769	895	73,48

As previously stated, the user `[deleted]` is not taken into account in our interpretation, as it represents removed authors. Excluding this entry, the top-ranking author in terms of total karma is `Donald Keyman`, with over 227,000 karma across only 312 comments — an impressive average of more than 700 per comment.

In addition to comments score and author karma, Reddit also allows users to reward comments with gilding—a paid form of recognition that highlights particularly valuable or appreciated content. Gilded comments typically stand out not only for their quality, but also for the emotional, humorous, or insightful impact they have on the community. Examining the top 5 most gilded comments in the dataset allows us to better understand what kinds of contributions are considered exceptional enough to deserve this rare form of endorsement.

Listing 20: Top 5 most gilded comments and their score

```
SELECT
    c.id,
    s.score,
    SUM(s.gilded) AS gilded_count
  FROM comments c
  JOIN score s ON c.id = s.id
 WHERE gilded > 0
```

```
GROUP BY c.id, s.score
ORDER BY gilded_count DESC
FETCH FIRST 5 ROWS ONLY;
```

	ID	SCORE	GILDED_COUNT
1	cr9xc7c	4669	12
2	cr0ea66	5361	11
3	crfjkeh	2227	9
4	crcvpys	4735	8
5	cripwvz	4595	7

These five comments received the highest number of gildings in the dataset, with up to 12 awards. While their scores are generally high, the number of gildings reflects an additional layer of appreciation, often associated with emotional resonance or insightful content. Gilded comments are relatively rare, making them a useful indicator of exceptional community recognition beyond simple upvotes.

After identifying the most gilded individual comments, it is natural to extend the analysis to the authors who received the most community awards. Gilding being a premium feature, it reflects exceptional appreciation and is often reserved for comments that stand out in quality, originality, or emotional value.

However, as with karma, simply counting the total number of gildings can be misleading: a highly active user may accumulate many awards simply by posting frequently. To better assess true recognition, we also consider the average number of gildings per comment, which highlights authors whose contributions are not only numerous, but consistently impactful.

Listing 21: Top 5 authors with the highest average number of gildings per comment

```
SELECT
    a.author,
    SUM(s.gilded) AS total_gilded,
    COUNT(c.id) AS total_comments,
    ROUND(SUM(s.gilded) / COUNT(c.id), 2) AS average_nb_reddit_gold
  FROM comments c
  JOIN author a ON c.author = a.author
  JOIN score s ON c.id = s.id
 GROUP BY a.author
 ORDER BY average_nb_reddit_gold DESC
 FETCH FIRST 5 ROWS ONLY;
```

	AUTHOR	TOTAL_GILDED	TOTAL_COMMENTS	AVERAGE_NB_REDDIT_GOLD
1	GoldenScissors	7	1	7
2	ionsevin	3	1	3
3	Aquahawk911	4	2	2
4	planetoiletsscareme	2	1	2
5	DaedricDanny	4	2	2

Interestingly, all top authors in terms of average gildings per comment posted very few comments—typically just one or two. This means their individual contributions were highly impactful and appreciated, but the analysis should be interpreted with caution: a single highly gilded comment can place an author at the top of this ranking, without necessarily reflecting sustained influence. Still, this metric is useful to highlight rare but exceptional contributions within the dataset.

So far, we have focused on individual comments and authors. To gain a broader perspective on which posts generated the most valuable discussions, we now shift our attention to the parent posts—that is, the original threads to which comments are attached.

Specifically, we calculate the average score of comments per parent post, while also taking into account the number of comments each post received. This dual approach allows us to identify threads that were not only active, but whose discussions were particularly well-received by the community.

Without this normalization, a post with hundreds of low-scoring comments could appear more "important" than one with fewer but highly appreciated contributions. Therefore, this weighted analysis provides a more accurate picture of a post's overall impact and engagement quality.

Listing 22: Average score and number of comments per parent post

```
SELECT
    p.parent_id,
    p.link_id,
    COUNT(c.id) AS comment_count,
    AVG(s.score) AS average_score
  FROM COMMENTS c
  JOIN DEPENDS d ON c.id = d.id
  JOIN PARENT p ON d.parent_id = p.parent_id
  JOIN SCORE s ON c.id = s.id
 GROUP BY p.parent_id, p.link_id
 ORDER BY comment_count DESC
 FETCH FIRST 5 ROWS ONLY;
```

	PARENT_ID	LINK_ID	COMMENT_COUNT	AVERAGE_SCORE
1	t3_37pr7d	t3_37pr7d	30771	2,7
2	t3_37c2p3	t3_37c2p3	11546	6,59
3	t3_36ih74	t3_36ih74	10981	11,93
4	t3_365rlk	t3_365rlk	7703	19,47
5	t3_364iu2	t3_364iu2	7558	6,95

In this query, we chose to order the results by the number of comments per parent post rather than by average score. This decision is motivated by the goal of identifying a relevant and representative subset for deeper analysis in the final section of the report.

Focusing on the most active posts ensures that we select a parent thread with significant user participation, while also checking that its average comment score remains reasonably high. In other words, we aim to prioritize posts that are both rich in content and community engagement, allowing us to justify our final focus on a meaningful and high-impact discussion thread.

Among the top parent posts, `t3 37c2p3` appeared to be a reasonable candidate for closer inspection in the final part of our analysis. With over 11,000 comments and an average score above 6.5, it combines substantial participation with solid community approval, making it both representative and qualitatively interesting.

To better understand the dynamics of engagement within Reddit discussions, we now examine whether comment scores vary according to their level of controversiality. The variable `controversiality` is a binary flag that identifies whether a comment sparked divided opinions, as perceived by Reddit's internal algorithm.

Analyzing the average score across controversial and non-controversial comments allows us to explore a fundamental question: Are controversial comments more appreciated, more ignored, or penalized by the community through lower voting?

This distinction is key to interpreting how Reddit values consensus versus provocation, and gives insight into the social dynamics of approval and dissent in large online discussions.

Listing 23: Average score by controversiality group

```
SELECT
  co.controversiality,
  COUNT(*) AS comment_count,
  AVG(s.score) AS average_score,
  STDDEV(s.score) AS std_dev
  FROM COMMENTS c
  JOIN CONTROVERSY co ON c.controversiality = co.controversiality
  JOIN SCORE s ON c.id = s.id
 GROUP BY co.controversiality
 ORDER BY co.controversiality;
```

	CONTROVERSIALITY	COMMENT_COUNT	AVERAGE_SCORE	STD_DEV
1	0	4182752	12,75	125,27
2	1	52218	0,86	13,05

The table above shows a striking difference in average scores between controversial and non-controversial comments. Comments marked as non-controversial have a much higher average score (12.75) than controversial ones (0.86), despite the controversial group still including over 50,000 comments.

However, this difference in means must be formally tested to determine whether it is statistically significant and not due to random variation. To do so, we perform a Welch's t-test, a common test used to compare the means of two groups when their variances and sample sizes differ, as is clearly the case here.

- **Null hypothesis (H0)**: the average scores of controversial and non-controversial comments are equal.
- **Alternative hypothesis (H1)**: the average scores are different.

Welch's t-statistic and degrees of freedom:

The test statistic is given by:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

And the degrees of freedom (df) are approximated by:

$$\text{df} = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{(s_1^2/n_1)^2}{n_1 - 1} + \frac{(s_2^2/n_2)^2}{n_2 - 1}}$$

Although SQL does not natively support statistical testing, we extracted all required components from the database using:

Listing 24: Computing group statistics for Welch's t-test

```
SELECT
  co.controversiality,
  COUNT(*) AS comment_count,
  AVG(s.score) AS average_score,
  STDDEV(s.score) AS std_dev
FROM COMMENTS c
JOIN CONTROVERSY co ON c.controversiality = co.controversiality
JOIN SCORE s ON c.id = s.id
GROUP BY co.controversiality
ORDER BY co.controversiality
),
welch_test AS (
  SELECT
    a.n AS n1,
    b.n AS n2,
    a.mean AS mean1,
    b.mean AS mean2,
    a.stddev AS sd1,
    b.stddev AS sd2,
    (a.mean - b.mean) AS diff_means,

    SQRT(POWER(a.stddev, 2)/a.n + POWER(b.stddev, 2)/b.n) AS std_error,

    -- t-statistic
    (a.mean - b.mean) / SQRT(POWER(a.stddev, 2)/a.n + POWER(b.stddev, 2)/b.n)
    AS t_statistic,

    -- Degrees of freedom (Welch-Satterthwaite)
    POWER((POWER(a.stddev, 2)/a.n + POWER(b.stddev, 2)/b.n), 2) /
    (
      (POWER(POWER(a.stddev, 2)/a.n, 2)) / (a.n - 1) +
      (POWER(POWER(b.stddev, 2)/b.n, 2)) / (b.n - 1)
    ) AS degrees_freedom

  FROM stats a
  JOIN stats b ON a.controversiality = 1 AND b.controversiality = 0
```

```
)
SELECT
  diff_means,
  std_error,
  t_statistic,
  degrees_freedom
FROM welch_test
```

These values were then used externally to compute the t-statistic and conclude whether the difference in scores is statistically significant.

Based on the computed statistics:

- Difference in means: -11.89
- Standard error: 0.0837
- t-statistic: -142.02
- Degrees of freedom: 237,632

The magnitude of the t-statistic is extremely large, far beyond typical critical values. This implies a p-value close to 0, well below any standard significance threshold (e.g., $\alpha = 0.05$).

We therefore reject the null hypothesis and conclude that the difference in average scores between controversial and non-controversial comments is statistically significant. In other words, the score received by a comment is indeed associated with its level of controversiality — with controversial comments receiving, on average, much lower scores.

After analyzing scores across various comment groups, including controversiality and engagement metrics, we now return to the broader question of how to identify content that stands out both quantitatively and qualitatively.

Among the most active posts, `t3_37c2p3` emerged as a compelling candidate for deeper exploration. With over 11,000 comments and an average score of approximately 6.6, it reflects a high level of community participation coupled with a solid reception. This combination makes it an ideal choice for a focused case study, allowing us to examine more closely the dynamics at play within a single high-impact discussion.

In the following section, we will analyze the structure and content of this post's comment thread to better understand what drives such extensive interaction on Reddit.

5 Focus on a specific post - sentiment analysis

5.1 Presentation of the post

To conclude our analysis, we zoom in on a specific discussion thread that stood out during our exploration of the dataset: `t3_37c2p3`, titled:

> *"High schoolers, what do you want to major in?"*

Originally posted in the AskReddit subreddit in May 2015, this post generated over 11,000 comments, making it one of the most active discussions of the month. Its average

comment score was approximately 6.6, indicating a generally positive reception by the community.

You can access the original post here: https://www.reddit.com/r/AskReddit/comments/37c2p3/high_schoolers_what_do_you_want_to_major_in/

Its estimated memory usage of our sample is 4.71 MB. This post is particularly well-suited for a focused analysis: it combines both high engagement and substantial user interaction, and its theme—youth aspirations—invites diverse and opinionated responses.

5.2 Score analysis of the post

Before diving into the analysis of our chosen post, we first needed to extract all relevant data related to it from the full dataset.

To do this efficiently, we created a materialized view — a physical subset of the data stored for faster access and querying. Unlike regular views, which are virtual and recomputed each time they are accessed, materialized views store the result of a query and can be refreshed periodically or manually.

In our case, the goal was to isolate the parent post `t3_37c2p3` along with its direct link ID, in order to filter out unrelated content. The following SQL statement was used:

Listing 25: Materialized view to isolate the selected parent post

```
CREATE MATERIALIZED VIEW parent_post_subset AS
SELECT parent_id, link_id
FROM PARENT
WHERE link_id = 't3_37c2p3'
  AND parent_id = 't3_37c2p3';
```

This subset then served as a reference to efficiently join with other tables and retrieve only the comments, scores, and metadata associated with the post we intended to study in detail.

An alternative to using a materialized view would have been to perform a direct `JOIN` between the `PARENT` and `COMMENTS` tables, using a `WHERE` clause to filter on the selected post.

While this approach is simpler and more dynamic, we chose the materialized view method for several reasons:

- Performance: The materialized view avoids scanning the entire `COMMENTS` table for every query by precomputing and storing the subset, which is especially valuable given the size of our dataset (over 4 million comments).
- Efficiency in repeated queries: Since we queried this specific post multiple times in our analysis, reusing a stored view ensured faster execution and reduced computational cost.
- Suitability for static data: Our dataset is fixed (May 2015), so there was no need for frequent refreshes. This makes a materialized view ideal in our case.

However, we also acknowledge the limitations of this approach:

- Storage overhead: The materialized view occupies additional storage space, though modest compared to the full database.

- Maintenance: If the underlying data were to change, the view would need to be refreshed manually or on schedule, which adds some complexity.

Given the static nature of our dataset and the frequency with which we queried this subset, the materialized view offered the best trade-off between performance and simplicity.

To begin our focused analysis of the post `t3_37c2p3`, we first examine the top 10 most upvoted comments in its thread.

Upvotes reflect the community's immediate and positive reaction to a comment. Studying the highest-scoring contributions offers insight into the kinds of answers that resonated most with readers—whether they were informative, relatable, humorous, or emotionally impactful.

This initial overview helps characterize the tone and content that drove engagement within the post and provides a qualitative glimpse into what made this thread so successful.

Listing 26: Top 10 most upvoted comments in the selected post

```
SELECT
    c.id,
    c.body,
    s.score
  FROM score s
  JOIN comments c ON s.id = c.id
  JOIN depends d ON c.id = d.id
  JOIN parent_post_subset pps ON d.parent_id = pps.parent_id
 ORDER BY s.score DESC
 FETCH FIRST 10 ROWS ONLY;
```

	ID	BODY	SCORE
1	crlfb88	Theatre major here. Just remember, technicians work in theaters, actors ...	4579
2	crle0cg	Ctrl + F for chemistry. 0 results. Good. Don't do it. Please see [this r...	4339
3	crlgfr9	I just wanna bring to the attention here that blue collar jobs have thri...	4190
4	crlddy3	Computer Science	3427

After identifying the most upvoted comments in the thread, we now focus on another dimension of community engagement: controversy.

Analyzing the most controversial comments within the selected post allows us to highlight responses that may have sparked debates, disagreements, or conflicting opinions—providing insight into the more polarizing aspects of the discussion.

Listing 27: Top 10 most controversial comments in the selected post

```
SELECT
    c.id,
    c.body,
    s.score,
    s.gilded,
```

```
    co.controversiality
  FROM score s
  JOIN comments c ON s.id = c.id
  JOIN depends d ON c.id = d.id
  JOIN parent_post_subset pps ON d.parent_id = pps.parent_id
  JOIN controversy co ON c.controversiality = co.controversiality
 WHERE co.controversiality = 1
 ORDER BY s.score ASC
 FETCH FIRST 10 ROWS ONLY;
```

For the sake of clarity and brevity, we only display the top controversial comment here, although our query retrieved the full top 10 for internal analysis. This allows us to illustrate the type of responses that generated polarized reactions in the thread.

The comment with ID `crle6qj` reads:

> *"High schoolers?!?! STOP. I didn't declare until my 3rd year in college, don't pretend you know exactly what you want to major in while you are in high school. Take a bunch of courses in college and find out what REALLY interests you....*

This comment received a negative score of -11 and was flagged as controversial. Its tone is assertive and somewhat dismissive of the original question, which likely contributed to a mixed reception. While the author shares a valid personal experience and encourages exploration, the phrasing may have alienated readers who felt judged or dismissed.

This example highlights how controversial comments on Reddit are not necessarily offensive or extreme: they may simply express strong opinions or challenge community assumptions, thus generating both support and backlash.

After examining the most upvoted and most controversial comments, we now turn to another form of community recognition: gilding.

By analyzing the most gilded comments in the selected thread, we aim to identify contributions that made a lasting impression on readers, offering insight into what kinds of messages prompted users to go beyond a simple upvote and show support in a more visible way.

Listing 28: Top gilded comments in the selected post, ordered by number of awards

```
SELECT
    c.id,
    c.body,
    s.score,
    s.gilded
  FROM score s
  JOIN comments c ON s.id = c.id
  JOIN depends d ON c.id = d.id
  JOIN parent_post_subset pps ON d.parent_id = pps.parent_id
 WHERE s.gilded > 0
 ORDER BY s.gilded DESC, s.score DESC;
```

As with previous analysis, we chose to display only the top gilded comment here for clarity and readability, although the full top 10 list was retrieved. This allows us to focus on one standout contribution that received a special form of recognition.

The comment with ID `crlgfr9` is reproduced below:

"I just wanna bring to the attention here that blue collar jobs have thriving markets and some very good pay and compensation. [...] Thanks guys."

This comment received a score of 4190 and was awarded 1 gilding. It offers a detailed, personal, and passionate reflection on pursuing a skilled trade rather than a traditional academic path. The author's honesty, humor, and encouragement seem to have resonated strongly with the audience, making it one of the most appreciated contributions in the thread—not just in terms of votes, but through a deliberate reward.

To conclude our focused analysis of the post `t3_37c2p3`, we shift our attention from individual comments to the users themselves. By identifying the most active and influential contributors, we aim to better understand who shaped the discussion at scale.

Listing 29: Top 10 most active and influential users in the selected post

```
SELECT
    c.author,
    COUNT(c.id) AS num_comments,
    SUM(s.score) AS total_score
  FROM score s
  JOIN comments c ON s.id = c.id
  JOIN depends d ON c.id = d.id
  JOIN parent_post_subset pps ON d.parent_id = pps.parent_id
 GROUP BY c.author
 ORDER BY total_score DESC
 FETCH FIRST 10 ROWS ONLY;
```

	AUTHOR	NUM_COMMENTS	TOTAL_SCORE
1	GregoryPippenbottom	1	4579
2	langis_on	1	4339
3	Lostlittlebunny	1	4190
4	[deleted]	719	3845
5	geoffisblind	1	3427
6	Final_Hour	2	3265
7	zaj209	1	2468
8	Panzer6	1	2184
9	Ohrianna	1	1941
10	AlekRivard	1	1932

The majority of top contributors in terms of total score posted only a single comment—showing that in this thread, impact did not require high activity. Instead, a single well-written or well-timed comment was enough to generate thousands of upvotes.

After identifying the most impactful comments and users in our selected post, a natural next step is to examine the content itself. While we now understand who contributed

most and how their contributions were received, it is equally important to explore what they actually said.

By analyzing the textual content of the thread using Python, we aim to uncover patterns in language, recurring themes, sentiment, or lexical choices that may explain why certain comments stood out. This complements our quantitative findings by offering a qualitative lens on what makes a Reddit comment successful or controversial.

Such textual analysis—using tools like tokenization, word frequency, or sentiment scoring—will provide valuable insights into the mechanics of virality and community dynamics on Reddit.

5.3 Text analysis in python

Having identified the most impactful comments and users, we next turned our attention to the textual content of the discussion. Our aim was to uncover the key words, recurring themes, and sentiment patterns that characterized comments in `t3_37c2p3`. We performed the analysis in Python using a variety of natural language processing (NLP) techniques, as described below.

Data Preprocessing and Word Frequency Analysis

To begin, we decided to normalize our text by extracting each comment's textual content, removing punctuation, converting to lowercase, and filtering out stopwords (e.g., "the," "and," "of"). We leveraged both the `nltk` library and the `spaCy` model `en_core_web_sm` for tokenization and part-of-speech tagging. Two preprocessing approaches were taken:

1. *Noun-only extraction*: Using `spaCy`, we retained only nouns and discarded all other parts of speech.
2. *Full token extraction*: We kept all words (nouns, verbs, adjectives, etc.) except stopwords.

Listing 30: Illustration of text preprocessing using `spaCy` and `NLTK`.

```
def preprocess_text_v2(text):
    # Remove punctuation, convert to lowercase
    text = re.sub(r'\W', ' ', text).lower().strip()
    doc = nlp(text)

    # Extract only nouns
    words = [
        token.text for token in doc
        if token.pos_ == "NOUN"
        and token.text not in stop_words
        and len(token.text) > 2
    ]
    return words
```

We then computed term frequencies using Python's built-in `collections.Counter` class. The most common nouns included `engineering`, `science`, `school`, and `computer`, reflecting the post's focus on academic and career paths. When retaining all parts of speech, additional frequent terms such as `want`, `like`, and `really` highlighted the colloquial, advice-seeking nature of the discussion:

- `engineering: 1936`,
- `want: 1434`,
- `major: 1400`,
- `science: 1221`,
- ...

Word Clouds

To visualize these frequent terms, we generated two word clouds (Figures 9 and 10). The first focuses solely on nouns, revealing an emphasis on "engineering", "science", "school", or even "computer". The second, built from all non-stopword tokens, additionally shows high usage of words like "want", "really", or "going", underscoring how students discuss their intentions and uncertainties.

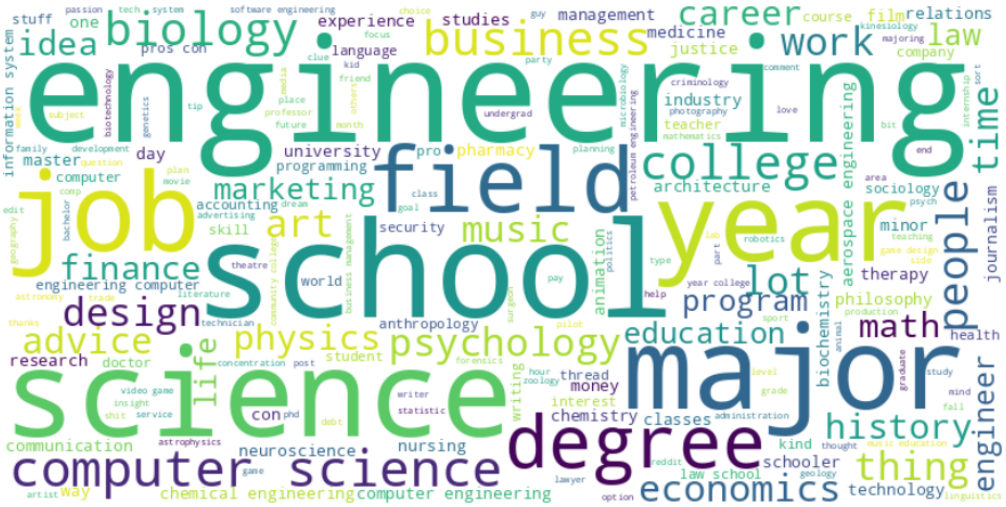

Figure 9: Word Cloud (Noun-only approach).

Figure 10: Word Cloud (All tokens).

Sentiment Analysis

To gauge the emotional tone of the comments, we employed the `TextBlob` library, which provides a polarity score ranging from "−1" (highly negative) to "+1" (highly positive). Figure 11 shows the sentiment distribution of the entire set of comments.

Listing 31: Example of applying a sentiment function with `TextBlob`.

```
def get_sentiment(text):
    return TextBlob(text).sentiment.polarity

job_dream_com["sentiment"] = job_dream_com["body"].apply(
   get_sentiment)
```

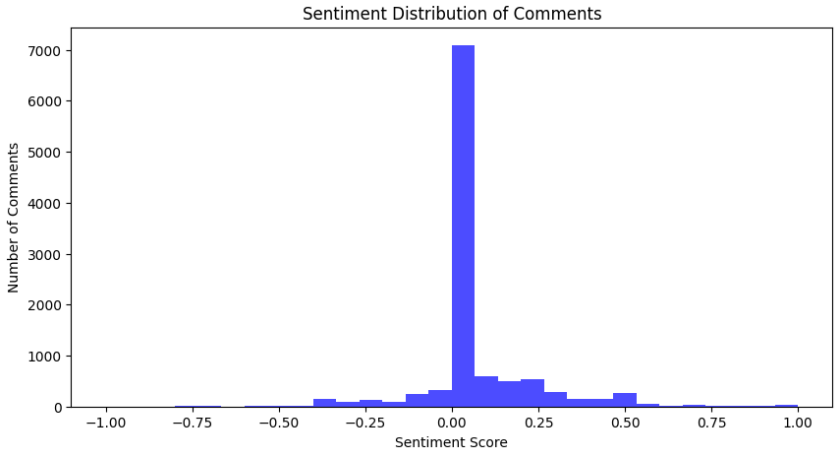

Figure 11: Sentiment Distribution of Comments.

Most comments clustered near neutral (0.0), indicating a relatively balanced or informational tone. The minority of highly positive or negative remarks often included enthusiastic endorsements of certain majors or discouragement toward specific fields. For instance, the top positive comments included phrases like *"I like business stuff"* or *"I want to be a pilot so bad"*, while more negative ones voiced frustrations: *"I really hate math"* or *"I've heard terrible things about it"*.

Named Entity Recognition (NER) and Bigrams

To deepen our linguistic analysis, we extracted named entities such as organizations, places, and nationalities using `spaCy`:

Listing 32: Extracting named entities of interest.

```
def extract_entities(text):
    doc = nlp(text)
    return [ent.text for ent in doc.ents if ent.label_ in ["ORG",
       "PERSON", "GPE", "NORP"]]
```

The most cited entities (e.g., *"Chemical Engineering," "Computer Engineering," "Neuroscience,"* etc.) spotlighted popular career paths. We also performed a bigram analysis, identifying frequent two-word phrases:

- `computer science: 511`,
- `high school: 342`,
- `want major: 253`,
- ...

showing how core topics clustered around academic decisions (*"computer science," "chemical engineering," "mechanical engineering," "high school"*).

Sentiment by Major

Finally, we compared average sentiment across a set of commonly mentioned majors (e.g., *computer science, engineering, biology, finance, law, medicine, physics,* etc.). Figure 12 shows the results, with *physics* comments exhibiting the highest average sentiment, followed closely by *marketing* and *biology*. By contrast, finance and computer science scored slightly lower on average—but still hovered in a mildly positive range.

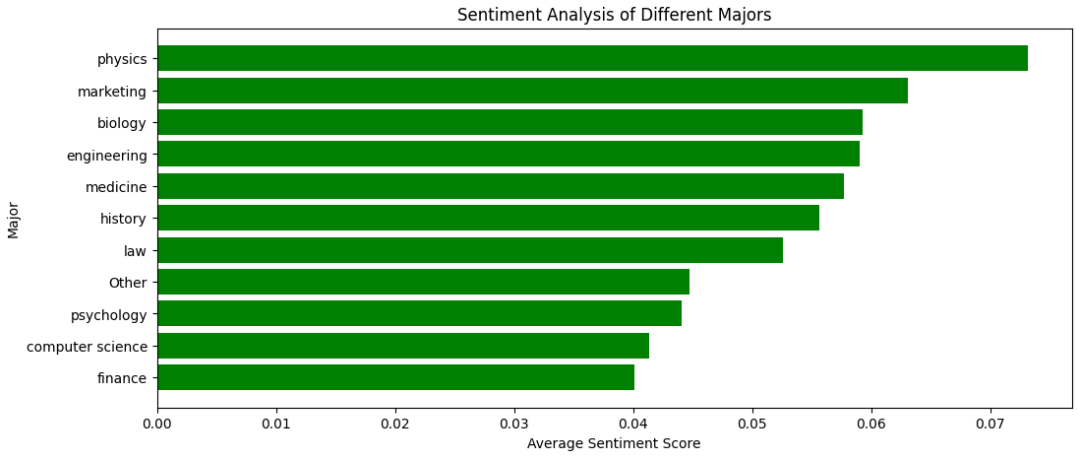

Figure 12: Average Sentiment by Major.

Interpretation Overall, the textual analysis confirms:

- **Recurring Topics:** The conversation prominently featured majors like engineering, computer science, and biology—fields frequently associated with STEM pathways.
- **Neutral to Positive Tonality:** Most comments showed near-neutral sentiment, likely reflecting a mix of personal anecdotes, practical advice, and moderate expressions of enthusiasm or concern.
- **Enthusiasm for Physics and Marketing:** Among explicitly mentioned majors, physics and marketing had slightly higher average sentiment scores, suggesting that those discussions were more upbeat or encouraging.

By combining basic frequency counts, word clouds, sentiment analysis, and simple entity/bigram extraction, we were able to derive a richer understanding of why certain fields, themes, or personal stories resonated with the community. This in-depth look at textual content helps explain both the thread's popularity and the diverse reactions—positive, negative, and controversial—that it evoked.

6 Conclusion

Reddit's AskReddit community provides a fascinating glimpse into how people share ideas, seek advice, and engage in conversation on a massive scale. By focusing on a subset—May 2015—while still retaining millions of comments, we were able to model and analyze a broad range of user interactions in a tractable yet representative manner.

Our study began with designing and creating a relational database schema tailored to the Reddit data, focusing on tables for comments, authors, scores, and parent-child relationships. We then carefully loaded the May 2015 AskReddit dataset into this database, ensuring referential integrity and consistent linking of comments to their parent posts.

Using this database as the foundation, we performed an initial exploratory analysis to understand the overall volume of activity and the posting patterns. We then conducted a deeper examination of the scoring system—covering everything from how the majority of comments cluster around low or moderate scores, to what drives certain posts or users to achieve unusually high recognition. This nuanced look at scores revealed the pivotal role of community dynamics such as controversy flags, gildings, and upvotes.

Finally, we zoomed in on one specific thread that displayed a combination of large-scale engagement and thematic breadth: high school students discussing their intended college majors. By applying sentiment analysis and text mining techniques, we uncovered how users expressed excitement or apprehension about certain fields, and we observed how highly upvoted contributions often offered relatable experiences, unique perspectives, or supportive advice.

In bringing together database modeling, data loading, exploratory statistics, and focused text analysis, this project highlights the power of structured approaches to understanding online discussions at scale. While our dataset covered only one subreddit over a single month, the methods employed can be generalized to a wide range of social media platforms and timeframes. Future work could expand this scope, incorporate more sophisticated NLP techniques, or investigate how specific events influence community behavior. Through this iterative, data-driven approach, we gain deeper insight into how large online communities exchange knowledge, convey emotion, and form collective identities.