ChatData

Sqlite

We are using a file system called **sqlite**. It looks and acts like a real single user relational database (RDB). sqlite3 comes packaged with python. You do not need to install the library. You simply import it (as below).

- We are using Pandas to write and read to sqlite. Pandas will manage a lot of the complexity of dealing with a RDB. There are other ways of reading and writing to a RDB that are VERY common and often used in production systems.
- The most common other way is to read or write to the RDB row by row. As you can see Pandas puts the entire dataframe into the RDB or extracts a new dataframe from the RDB. These are not row by row operations. These are set operations. Set operations ARE more efficient. However, it is common to use row by row operations to avoid needing large memory computers to hold the dataframes. One row only takes a few bytes. An entire dataframe could be many gigabytes and even petabytes of data. Obviously you will end up with problems with your compute resource if your files are this big. This will not happen to you in this activity and may never happen to you while you are a data analyst. Please just be aware of this.
- This activity is primarily about querying a RDB, so, we are using a simple way that Pandas provides. If you want to learn the more complex way, just google 'reading and wrting to a relational database using python'. There are many resources to learn from.
- The data types that sqlite supports are quite limited. for example, it does not have a DATE type. This is not a challenge for this project. However, more sophisticated database systems,such as Postgresql, have a large array of data types, such as "DATE", etc. that give those systems additional capability.
- All of the SQL queries could also be performed on the Pandas DataFrames directly. You may want to try this yourself for comparison (but make sure you do the SQL queries first, as this is an exercise in using SQL!).

SQL Magic

Within the Jupyter notebook we will be using something called **SQL Magic**. This provides a convenient way to write SQL queries directly into code cells in the notebook and to read the results back into a Pandas DataFrame. This makes working with SQL much easier!

Note that you may need to update your sqlite version using `conda update -c anaconda sqlite` in order for the SQL Magic to work correctly.

The Data Analysis Lifecycle

The sections in this notebook follow the stages of the Data Analysis Lifecycle introduced in an earlier activity. The stages are:

- Acquire
- Transform
- Organise
- Analyse
- Communicate
- Maintain

The requirements document for this project is Template SQL queries.xlsx.

Preliminary Steps: Create a Database

First let's import Sqlite and the other libraries we will need.

In [1]:
```
# Import Libraries
import numpy as np
import pandas as pd
import sqlite3
```

In [2]:
```
#!pip install ipython-sql
```

Create the Database

Now we will create the Sqlite database. Here is some code that does this for you. We use the `%load_ext` magic command to load the SQL Magic extension and then use `%sql` to connect to the database.

In [3]:
```
# If the db does not exist, sqlite will create it.
con = sqlite3.connect('chatdata.db')

# loads sql magic
%load_ext sql

# connects sql magic command to the correct db
%sql sqlite:///chatdata.db
```

Drop the `queries` table if it already exists

The queries table will be our record of the queries created to answer the questions from the requirements spreadsheet. As we will be running this Jupyter notebook a few times, let's drop (i.e. remove) the queries table so that we start fresh each time. Here is the code to do this. We use the `%%sql` magic command to tell Jupyter that we are going to write SQL in the cell.

In [4]:
```
%%sql
DROP TABLE IF EXISTS 'queries'
```

```
 * sqlite:///chatdata.db
Done.
[]
```

Out[4]:

Task 1: Load the Data

Now we can start loading the data. The tables will be created as we load the data.

Note that some of these files are quite large, so make sure you have plenty of free memory!

Lifecycle Stages: Transform and Organise

The data has already been processed into 3 clean data files ready for this project:

- queries.csv
- posts.csv
- comments.csv

We will load these files into our database.

Lifecycle Stage: Acquire

The data can be found in the OpenClassrooms instructions for this activity.

Load Comments Data into a `comments` table

Now we will load the data from the csv files into our sqlite database.

First we load the csv file into a Pandas dataframe:

In [5]:
```
# Read the comments dataset into a DataFrame
comments = pd.read_csv('comments.csv')

# Explore the first 5 rows
comments.head()
```

Out[5]:

	Id	PostId	Score	Text	CreationDate	UserId
0	723182	385124	0	@BenBolker I don't understand. The fit cannot ...	2019-01-01 00:06:39	78575
1	723183	385124	3	You can't add *less* than $(-\min(y))$, but you...	2019-01-01 00:09:22	2126
2	723186	385137	0	nice. If you felt like doing the work it would...	2019-01-01 00:32:11	2126
3	723187	385137	0	i.e. `emdbook::curve3d(-sum(dnbinom(y,mu=mu,si...	2019-01-01 00:40:36	2126

	Id	PostId	Score	Text	CreationDate	UserId
4	723188	385134	0	Don't you mean "so variance should be $\sigma^...	2019-01-01 00:41:28	112141

Now take the comments dataframe and push the data into the Sqlite database table called 'comments':

In [6]:
```
# Load comments into sqlite
comments.to_sql('comments', con, if_exists='replace', index=False)

# read back in to prove that it worked
sql = 'SELECT * FROM comments'
comments = pd.read_sql(sql, con)
comments.head()
```

Out[6]:

	Id	PostId	Score	Text	CreationDate	UserId
0	723182	385124	0	@BenBolker I don't understand. The fit cannot ...	2019-01-01 00:06:39	78575
1	723183	385124	3	You can't add *less* than $(-\min(y))$, but you...	2019-01-01 00:09:22	2126
2	723186	385137	0	nice. If you felt like doing the work it would...	2019-01-01 00:32:11	2126
3	723187	385137	0	i.e. `emdbook::curve3d(-sum(dnbinom(y,mu=mu,si...	2019-01-01 00:40:36	2126
4	723188	385134	0	Don't you mean "so variance should be $\sigma^...	2019-01-01 00:41:28	112141

Load Other CSVs

Now that you have seen the code for loading in the comments.csv now do the same to read and write the posts.csv and users.csv to sqlite.

TODO: Enter code in the following cells. Insert as many cells as you want to do this.

In [7]:
```
# Read the post dataset into a DataFrame
posts = pd.read_csv('posts.csv')

# Explore the first 5 rows
posts.head()
```

Out[7]:

	Id	PostTypeId	AcceptedAnswerId	ParentId	CreationDate	Score	ViewCount	
0	423497	1	423511	0	2019-08-24 09:39:31	2	68	<p>From w href="https://e
1	423498	1	0	0	2019-08-24 09:47:42	1	24	<p>I am curr local sensi

	Id	PostTypeId	AcceptedAnswerId	ParentId	CreationDate	Score	ViewCount	
2	423499	1	0	0	2019-08-24 09:48:26	1	56	<p>I'm student in
3	423500	2	0	215865	2019-08-24 09:57:01	0	0	<p>Maybe yc this <a href="h
4	423502	2	0	423286	2019-08-24 10:44:52	3	0	<blockquote> Is my approacl

5 rows × 21 columns

In [8]:
```
# load post DataFrame into sqlite
posts.to_sql('posts', con, if_exists='replace', index=False)

# read back in to prove that it worked
sql = 'SELECT * FROM posts'
posts = pd.read_sql(sql, con)
posts.head()
```

Out[8]:

	Id	PostTypeId	AcceptedAnswerId	ParentId	CreationDate	Score	ViewCount	
0	423497	1	423511	0	2019-08-24 09:39:31	2	68	<p>From w href="https://e
1	423498	1	0	0	2019-08-24 09:47:42	1	24	<p>I am curr local sensi
2	423499	1	0	0	2019-08-24 09:48:26	1	56	<p>I'm student in
3	423500	2	0	215865	2019-08-24 09:57:01	0	0	<p>Maybe yc this <a href="h
4	423502	2	0	423286	2019-08-24 10:44:52	3	0	<blockquote> Is my approacl

5 rows × 21 columns

In [9]:
```
# Read the users dataset into a DataFrame
users = pd.read_csv('users.csv')

# Explore the first 5 rows
users.head()
```

Out[9]:

	Id	Reputation	CreationDate	DisplayName	LastAccessDate	WebsiteUrl	Location	Abou
0	157607	31	2017-04-17 14:50:42	user157607	2019-07-23 16:44:08	NaN	NaN	I
1	157656	101	2017-04-17 20:08:20	user102859	2019-06-26 13:42:13	NaN	NaN	I
2	157704	133	2017-04-18 05:10:47	jupiar	2019-11-25 13:32:27	NaN	Shanghai, China	<p>Origi from the I hav Underç
3	157709	155	2017-04-18 06:39:18	farmer	2019-02-17 19:44:24	NaN	NaN	I
4	157755	101	2017-04-18 12:56:17	Miki P	2019-08-12 17:02:21	NaN	NaN	I

In [10]:
```
# load post DataFrame into sqlite
users.to_sql('users', con, if_exists='replace', index=False)

# read back in to prove that it worked
sql = 'SELECT * FROM users'
users = pd.read_sql(sql, con)
users.head()
```

Out[10]:

	Id	Reputation	CreationDate	DisplayName	LastAccessDate	WebsiteUrl	Location	Abou
0	157607	31	2017-04-17 14:50:42	user157607	2019-07-23 16:44:08	None	None	N
1	157656	101	2017-04-17 20:08:20	user102859	2019-06-26 13:42:13	None	None	N
2	157704	133	2017-04-18 05:10:47	jupiar	2019-11-25 13:32:27	None	Shanghai, China	<p>Origi from the I hav Underç
3	157709	155	2017-04-18 06:39:18	farmer	2019-02-17 19:44:24	None	None	N
4	157755	101	2017-04-18 12:56:17	Miki P	2019-08-12 17:02:21	None	None	N

TODO: Drop Duplicates

Look for and drop any duplicates in all 3 of the tables (if they exist). Use Pandas to do this. If you find duplicates, you will need to rewrite the table.

TODO: Enter code in the following cells.

In [11]:
```
# Find duplicates in the comments DataFrame
comments[comments.duplicated()]
```

Out[11]:

Id	PostId	Score	Text	CreationDate	UserId

In [12]:
```
# Find duplicates in the posts DataFrame
posts[posts.duplicated()]
```

Out[12]:

Id	PostTypeId	AcceptedAnswerId	ParentId	CreationDate	Score	ViewCount	Body	OwnerUserId

0 rows × 21 columns

In [13]:
```
# Find duplicates in the posts DataFrame
users[users.duplicated()]
```

Out[13]:

Id	Reputation	CreationDate	DisplayName	LastAccessDate	WebsiteUrl	Location	AboutMe	View

> There are no duplicates in the 3 datasets

Review the Data

Spend some time reviewing the data. Understand what data we have, think about how that data can be used to assist in the initiative of understanding how ChatData is used in the real world. Is the data organised in a way that would lend itself to being manged in a relational database? How would the different tables be connected? What are the primary and foreign keys? Would this give you a 3NF model?

Also think about security and ethics. Is there personal data in here? Could individuals be identified through this data? It it ethical to use the data in this way? You will be asked to comment on these questions later!

Use the code below to help you.

In [14]:
```
users.columns
```

Out[14]:
```
Index(['Id', 'Reputation', 'CreationDate', 'DisplayName', 'LastAccessDate',
       'WebsiteUrl', 'Location', 'AboutMe', 'Views', 'UpVotes', 'DownVotes',
       'ProfileImageUrl', 'AccountId'],
      dtype='object')
```

In [15]:
```
users.head()
```

Out[15]:

	Id	Reputation	CreationDate	DisplayName	LastAccessDate	WebsiteUrl	Location	Abou
0	157607	31	2017-04-17 14:50:42	user157607	2019-07-23 16:44:08	None	None	N
1	157656	101	2017-04-17 20:08:20	user102859	2019-06-26 13:42:13	None	None	N
2	157704	133	2017-04-18 05:10:47	jupiar	2019-11-25 13:32:27	None	Shanghai, China	<p>Origi from the I hav Underg
3	157709	155	2017-04-18 06:39:18	farmer	2019-02-17 19:44:24	None	None	N
4	157755	101	2017-04-18 12:56:17	Miki P	2019-08-12 17:02:21	None	None	N

In [16]:
```
comments.columns
```

Out[16]:
```
Index(['Id', 'PostId', 'Score', 'Text', 'CreationDate', 'UserId'], dtype='object')
```

In [17]:
```
comments.head()
```

Out[17]:

	Id	PostId	Score	Text	CreationDate	UserId
0	723182	385124	0	@BenBolker I don't understand. The fit cannot ...	2019-01-01 00:06:39	78575
1	723183	385124	3	You can't add *less* than $(-\min(y))$, but you...	2019-01-01 00:09:22	2126
2	723186	385137	0	nice. If you felt like doing the work it would...	2019-01-01 00:32:11	2126
3	723187	385137	0	i.e. `emdbook::curve3d(-sum(dnbinom(y,mu=mu,si...	2019-01-01 00:40:36	2126
4	723188	385134	0	Don't you mean "so variance should be $\sigma^...	2019-01-01 00:41:28	112141

In [18]:
```
posts.columns
```

Out[18]:
```
Index(['Id', 'PostTypeId', 'AcceptedAnswerId', 'ParentId', 'CreationDate',
       'Score', 'ViewCount', 'Body', 'OwnerUserId', 'OwnerDisplayName',
       'LastEditorUserId', 'LastEditorDisplayName', 'LastEditDate',
       'LastActivityDate', 'Title', 'Tags', 'AnswerCount', 'CommentCount',
       'FavoriteCount', 'ClosedDate', 'CommunityOwnedDate'],
      dtype='object')
```

In [19]:
```
posts.head()
```

Out[19]:

	Id	PostTypeId	AcceptedAnswerId	ParentId	CreationDate	Score	ViewCount	
0	423497	1	423511	0	2019-08-24 09:39:31	2	68	<p>From w href="https://e

	Id	PostTypeId	AcceptedAnswerId	ParentId	CreationDate	Score	ViewCount	
1	423498	1	0	0	2019-08-24 09:47:42	1	24	<p>I am curr local sensi
2	423499	1	0	0	2019-08-24 09:48:26	1	56	<p>I'm student in
3	423500	2	0	215865	2019-08-24 09:57:01	0	0	<p>Maybe y this <a href="h
4	423502	2	0	423286	2019-08-24 10:44:52	3	0	<blockquote> Is my approacl

5 rows × 21 columns

Working with Sqlite and SQL Magic

In this section let's spend a little time understanding a bit more about how we can work with Sqlite within Jupyter.

Let's look at 2 ways to query the sqlite database: using SQL Magic or using Pandas. Either way is fine for this project.

Writing queries with SQL Magic

You will now need to write some queries to get answers to the questions in the requirements.

For single-line queries, start the cell with `%sql` and simply enter your query:

In [20]:
```
# This is an example
%sql SELECT COUNT(*) FROM comments
```

```
 * sqlite:///chatdata.db
Done.
```

Out[20]:

COUNT(*)
50000

For multi line sql statements use `%%sql` as follows. This tells Jupyter that *everything* in this cell should be interpreted as sql. So, NO comments other statements are allowed:

In [21]:
```
%%sql
SELECT Id, PostId, Score, Text
```

```
    FROM comments
        LIMIT 5
```

```
 * sqlite:///chatdata.db
Done.
```

Out[21]:

	Id	PostId	Score	
723182	385124	0	@BenBolker I don't understand. The fit cannot be done for the negative y. So ir
723183	385124	3	You can't add *less* than $(-\min(y))$, but you could add *more*. I'm going to stop ar
723186	385137	0	nice. If you felt like doing the v
723187	385137	0	$emdb\infty k::curve3d\Big(-\sum(dnb \in om(y, \mu = \mu, size = size, \log = TRUE)), a$
723188	385134	0	

Writing queries with Pandas

Another way to write queries is to use pandas:

In [22]:

```
sql = """
SELECT Id, PostId, Score, Text
    FROM comments
        LIMIT 5
        """
result = pd.read_sql(sql, con)
result
```

Out[22]:

	Id	PostId	Score	Text
0	723182	385124	0	@BenBolker I don't understand. The fit cannot ...
1	723183	385124	3	You can't add *less* than $(-\min(y))$, but you...
2	723186	385137	0	nice. If you felt like doing the work it would...
3	723187	385137	0	i.e. `emdbook::curve3d(-sum(dnbinom(y,mu=mu,si...
4	723188	385134	0	Don't you mean "so variance should be $\sigma^...

Creating Tables with Referential Integrity

When we loaded the csv files into Sqlite database tables, Sqlite created the tables for us behind the scenes. Let's inspect this a bit more.

We can see how Sqlite created the tables by querying the `sqlite_master` table, which Sqlite uses to keep track of what objects have been created in the database:

In [23]:

```
%%sql
select sql from sqlite_master
```

```
 * sqlite:///chatdata.db
Done.
```

Out[23]:

sql
CREATE TABLE "comments" ("Id" INTEGER, "PostId" INTEGER, "Score" INTEGER, "Text" TEXT, "CreationDate" TEXT, "UserId" INTEGER)
CREATE TABLE "posts" ("Id" INTEGER, "PostTypeId" INTEGER, "AcceptedAnswerId" INTEGER, "ParentId" INTEGER, "CreationDate" TEXT, "Score" INTEGER, "ViewCount" INTEGER, "Body" TEXT, "OwnerUserId" INTEGER, "OwnerDisplayName" TEXT, "LastEditorUserId" INTEGER, "LastEditorDisplayName" TEXT, "LastEditDate" TEXT, "LastActivityDate" TEXT, "Title" TEXT, "Tags" TEXT, "AnswerCount" INTEGER, "CommentCount" INTEGER, "FavoriteCount" INTEGER, "ClosedDate" TEXT, "CommunityOwnedDate" TEXT)
CREATE TABLE "users" ("Id" INTEGER, "Reputation" INTEGER, "CreationDate" TEXT, "DisplayName" TEXT, "LastAccessDate" TEXT, "WebsiteUrl" TEXT, "Location" TEXT, "AboutMe" TEXT, "Views" INTEGER, "UpVotes" INTEGER, "DownVotes" INTEGER, "ProfileImageUrl" TEXT, "AccountId" INTEGER)

The above results show the `CREATE TABLE` statements that could be used by Sqlite to recreate the tables with the exact same structure.

The problem with the CREATE TABLE statements above is that they don't enforce **referential integrity**. In other words, they don't ensure that every UserId and PostId in the comments table refers to an actual UserId and PostId in the users and posts tables. At the moment. we can insert any old number here, and even have multiple users with the same Id! One of the advantages of working with relational databases is that they can enforce the correct uniqueness and relationships in the data, but at the moment we are not using that feature. So let's fix that...

First, let's drop the original tables:

In [24]:
```
%%sql
DROP TABLE comments;
DROP TABLE users;
DROP TABLE posts;
```

```
 * sqlite:///chatdata.db
Done.
Done.
Done.
```

Out[24]: []

Prove that this worked by selecting the names of the tables back. We should have no tables:

In [25]:
```
%%sql
SELECT name FROM sqlite_master WHERE type='table'
    ORDER BY name
```

```
 * sqlite:///chatdata.db
Done.
```

Out[25]:

name

In Sqlite we need to enable the enforcement of foreign key constraints:

In [26]:
```
%%sql
PRAGMA foreign_keys=ON;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[26]: []

Now recreate the users table with a **primary key constraint** by copying the CREATE TABLE statement from above and adding the NOT NULL PRIMARY KEY clause to the Id:

In [27]:
```
%%sql
CREATE TABLE "users" (
"Id" INTEGER NOT NULL PRIMARY KEY,
"Reputation" INTEGER,
"CreationDate" TEXT,
"DisplayName" TEXT,
"LastAccessDate" TEXT,
"WebsiteUrl" TEXT,
"Location" TEXT,
"AboutMe" TEXT,
"Views" INTEGER,
"UpVotes" INTEGER,
"DownVotes" INTEGER,
"ProfileImageUrl" TEXT,
"AccountId" INTEGER
);
```

```
 * sqlite:///chatdata.db
Done.
```

Out[27]: []

Now do the same for the posts table:

TODO: Complete the following code cell

In [28]:
```
# Run the CREATE TABLE statement for the posts table, including the primary key cons
```

In [29]:
```
#%%sql
#DROP TABLE posts
```

In [30]:
```
%%sql
CREATE TABLE "posts" (
"Id" INTEGER NOT NULL PRIMARY KEY,
"PostTypeId" INTEGER,
"AcceptedAnswerId" INTEGER,
"ParentId" INTEGER,
"CreationDate" TEXT,
"Score" INTEGER,
"ViewCount" INTEGER,
"Body" TEXT,
"OwnerUserId" INTEGER,
"OwnerDisplayName" TEXT,
"LastEditorUserId" INTEGER,
"LastEditorDisplayName" TEXT,
"LastEditDate" TEXT,
"LastActivityDate" TEXT,
"Title" TEXT,
"Tags" TEXT,
"AnswerCount" INTEGER,
"CommentCount" INTEGER,
"FavoriteCount" INTEGER,
"ClosedDate" TEXT,
"CommunityOwnedDate" TEXT,
FOREIGN KEY(OwnerUserId) REFERENCES users(Id)
);
```

Out[30]:
```
 * sqlite:///chatdata.db
Done.
[]
```

Now for the comments table. We need to add the primary key constraint on the id here as we did for users and posts, but we also need to add FOREIGN KEY constraints on the UserId and PostId. Read the documentation here and find our how to do that: https://www.sqlite.org/foreignkeys.html. Then create the comments table with the correct constraints:

TODO: Complete the following code cell

In [31]:
```
# Run the CREATE TABLE statement for the comments table, including the primary key c
```

In [32]:
```
%%sql
CREATE TABLE "comments" (
"Id" INTEGER PRIMARY KEY,
"PostId" INTEGER,
"Score" INTEGER,
"Text" TEXT,
"CreationDate" TEXT,
```

```
 "UserId" INTEGER,
  FOREIGN KEY(PostId) REFERENCES posts(Id),
  FOREIGN KEY(UserId) REFERENCES users(Id)
 )
```

```
 * sqlite:///chatdata.db
Done.
```

Out[32]:
```
[]
```

Now we can re-insert the data into these constrained tables. First users:

In [33]:
```
# Insert data into the new users table
users.to_sql('users', con, if_exists='append', index=False)
```

Now posts:

TODO: Complete the following code cell

In [34]:
```
# Insert data into the new posts table
posts.to_sql('posts',con,if_exists='append',index=False)
```

Finally comments, which references the users and posts tables:

TODO: Complete the following code cell

In [35]:
```
# Insert data into the new comments table
comments.to_sql('comments',con,if_exists='append',index=False)
```

Now check that we have the 3 new table definitions in Sqlite:

In [36]:
```
%%sql
SELECT name FROM sqlite_master WHERE type='table'
    ORDER BY name
```

```
 * sqlite:///chatdata.db
Done.
```

Out[36]:

name
comments
posts
users

We now have all the data in tables in Sqlite and the tables will enforce the referential integrity.

Example Query and Pattern for Tasks 2 and 3

As you work through the next tasks, you will need to:

1. Prepare the Sqlite query to answer the question
2. Test it
3. Insert it into the `queries` table, so we have a record of it for others.

This is the process that we want you to follow for this project while completing it.

Let's see an example of this by answering the following question:

Which 5 users have viewed the most times and what is the sum of those views per user?

Prepare the Sqlite query

First, let's write the query:

In [37]:
```
sql = """
SELECT Id, SUM(Views) AS TotalViews
    FROM Users
        GROUP BY Id
            ORDER BY TotalViews DESC
                LIMIT 5
                """
result = pd.read_sql(sql, con) # con is the connection to the database
result
```

Out[37]:

	Id	TotalViews
0	919	85180
1	4253	35119
2	805	34637
3	7290	32639
4	3277	29255

Test the query

You can optionally prove the query worked by performing the same query in Pandas:

In [38]:
```
results = users.groupby(['Id']).sum().sort_values('Views', ascending = False)[:5]
results['Views']
```

Out[38]:
```
Id
919     85180
4253    35119
805     34637
7290    32639
3277    29255
Name: Views, dtype: int64
```

Insert the query into the `queries` table

Now we need to put this query into the `queries` table in sqlite. Remember we want these queries to be accessible to everybody that should have access to them. We do not want people writing and rewriting the same queires over and over again. The easiest thing to do is create a dictionary with the values and insert these into the queries table. Note that the values are provided as lists as we are inserting a list of values (i.e. a number of rows) into the table. In this case the number of rows is 1, so we have lists of 1 item.

So here, we have a column called 'task' with a list of values, a column called 'action' with a list of values, etc.

In [39]:
```
query_dict = {
              'task': ['Single Table Queries'],
              'action': ['Which 5 users have viewed the most times and what is the s
              'query': [sql]
             }
query_dict
```

Out[39]:
```
{'task': ['Single Table Queries'],
 'action': ['Which 5 users have viewed the most times and what is the sum of those v
iews per user?'],
 'query': ['\nSELECT Id, SUM(Views) AS TotalViews\n    FROM Users\n        GROUP BY
Id\n            ORDER BY TotalViews DESC\n                LIMIT 5\n
']}
```

Now that you have the data structure (query_dict) containing the data, create a pandas dataframe that holds those values:

In [40]:
```
queries = pd.DataFrame(query_dict)
queries
```

Out[40]:

	task	action	query
0	Single Table Queries	Which 5 users have viewed the most times and w...	\nSELECT Id, SUM(Views) AS TotalViews\n FRO...

Now load that pandas dataframe (queries) into the sqlite table called queries. In this case, you use append NOT replace. You will be adding to this tables as you go thru this project.

In [41]:
```
# load query into sqlite
queries.to_sql('queries', con, if_exists='append', index=False)

# read back in to prove that it worked
sql = 'SELECT * FROM queries'
queries = pd.read_sql(sql, con)
queries.head()
```

Out[41]:

	task	action	query
0	Single Table Queries	Which 5 users have viewed the most times and w...	\nSELECT Id, SUM(Views) AS TotalViews\n FRO...

It is likely that as you iterate thru this notebook you will create some duplicate entries in the query table. Not a big deal. Just tell sql or pandas (much easier in pandas) to drop the duplicates! If you drop the duplicates in Pandas you will have to write the entire dataframe back to sqlite. Otherwise it is changed in memory in Pandas but NOT in sqlite on the disk drive (or SDD). Just do this drop at the end of this notebook so that you are not constantly dealing with this.

So, to summarise, as you go through the following tasks you need to:

- answer the question in sql
- prove it in pandas (if you want to)
- put the query into the queries table

Task 1 (continued): Insert the CREATE TABLE Statements into the `queries` Table

Now that we understand how to populate the `queries` table, let's insert the CREATE TABLE statements into it. First let's define a function to help us insert into the queries table:

In [42]:
```
# Define a function that will insert into the queries table
def store_query(task, action, query):
    query_dict = {
                  'task': [task],
                  'action': [action],
                  'query': [query]
                 }

    # put query into the query_dict
    queries = pd.DataFrame(query_dict)

    # load query into sqlite
    queries.to_sql('queries', con, if_exists='append', index=False)
```

Now we can specify the queries and call the above function to store them. The first one is done for you.

In [43]:
```
sql = """
    CREATE TABLE "comments" (
    "Id" INTEGER,
    "PostId" INTEGER,
    "Score" INTEGER,
    "Text" TEXT,
    "CreationDate" TEXT,
    "UserId" INTEGER
    )
    """
store_query("Task 1", "Create table comments", sql)
```

Let's prove it works by selecting back from the queries table:

In [44]:
```
# Prove it works
%sql SELECT * FROM queries
```

```
 * sqlite:///chatdata.db
Done.
```

Out[44]:

task	action	query
Single Table Queries	Which 5 users have viewed the most times and what is the sum of those views per user?	SELECT Id, SUM(Views) AS TotalViews FROM Users GROUP BY Id ORDER BY TotalViews DESC LIMIT 5
Task 1	Create table comments	CREATE TABLE "comments" ("Id" INTEGER, "PostId" INTEGER,

```
"Score" INTEGER,
"Text" TEXT,
"CreationDate" TEXT,
"UserId" INTEGER
)
```

Insert the other CREATE TABLE statements into the `queries` table.

Follow the above pattern to complete these code cells:

TODO: Complete the following code cells

In [45]:
```
# Insert the CREATE TABLE for posts into the queries table
sql = """
    CREATE TABLE "posts" (
        "Id" INTEGER NOT NULL PRIMARY KEY,
        "PostTypeId" INTEGER,
        "AcceptedAnswerId" INTEGER,
        "ParentId" INTEGER,
        "CreationDate" TEXT,
        "Score" INTEGER,
        "ViewCount" INTEGER,
        "Body" TEXT,
        "OwnerUserId" INTEGER,
        "OwnerDisplayName" TEXT,
        "LastEditorUserId" INTEGER,
        "LastEditorDisplayName" TEXT,
        "LastEditDate" TEXT,
        "LastActivityDate" TEXT,
        "Title" TEXT,
        "Tags" TEXT,
        "AnswerCount" INTEGER,
        "CommentCount" INTEGER,
        "FavoriteCount" INTEGER,
        "ClosedDate" TEXT,
        "CommunityOwnedDate" TEXT,
        FOREIGN KEY(OwnerUserId) REFERENCES users(Id)
        )
    """
store_query("Task 1", "Create table posts", sql)
```

In [46]:
```
# Insert the CREATE TABLE for users into the queries table
sql = """
    CREATE TABLE "users" (
        "Id" INTEGER NOT NULL PRIMARY KEY,
        "Reputation" INTEGER,
        "CreationDate" TEXT,
        "DisplayName" TEXT,
        "LastAccessDate" TEXT,
        "WebsiteUrl" TEXT,
        "Location" TEXT,
        "AboutMe" TEXT,
        "Views" INTEGER,
        "UpVotes" INTEGER,
        "DownVotes" INTEGER,
        "ProfileImageUrl" TEXT,
        "AccountId" INTEGER
```

```
        )
    """
store_query("Task 1", "Create table users", sql)
```

In [47]:

```
# Confirm that the data is stored
sql="""
     SELECT * FROM queries
    """
queries=pd.read_sql(sql,con)
queries
```

Out[47]:

	task	action	query
0	Single Table Queries	Which 5 users have viewed the most times and w...	\nSELECT Id, SUM(Views) AS TotalViews\n FRO...
1	Task 1	Create table comments	\n CREATE TABLE "comments" (\n "Id" INTE...
2	Task 1	Create table posts	\n CREATE TABLE "posts" (\n "Id" INT...
3	Task 1	Create table users	\n CREATE TABLE "users" (\n "Id" INT...

Count the Number of Rows in Each Table

Run some queries to count the number of rows in each of the tables. Don't forget to insert the query into the queries table.

TODO: Complete the following code cells

In [48]:

```
# Count the number of rows in the comments table
sql = """
    SELECT count(*)
    FROM "comments"
    """
count_row_comments=pd.read_sql(sql,con)

store_query("Task 1", "Count the number of rows in the comments table", sql)
count_row_comments
```

Out[48]:

	count(*)
0	50000

In [49]:

```
%%sql
DELETE FROM queries WHERE action = "Run the query to select 5 random rows from the p
```

Out[49]:

```
 * sqlite:///chatdata.db
0 rows affected.
[]
```

In [50]:

```
# Count the number of rows in the users table
sql = """
    SELECT count(*)
    FROM "users"
    """
```

```
count_row_users=pd.read_sql(sql,con)

store_query("Task 1", "Count the number of rows in the users table", sql)
count_row_users
```

Out[50]:

	count(*)
0	18412

In [51]:

```
# Count the number of rows in the posts table
sql = """
    SELECT count(*)
    FROM "posts"
    """
count_row_posts=pd.read_sql(sql,con)

store_query("Task 1", "Count the number of rows in the posts table", sql)
count_row_users
```

Out[51]:

	count(*)
0	18412

Do some Random Checks on the Data

Let's write some queries that select 5 random rows from each table. The queries are provided here:

```
select * from Comments order by random() limit 5;
select * from Users order by random() limit 5;
select * from Posts order by random() limit 5;
```

Enter the queries into the code cells below and insert the queries into the `queries` table in the same was as you did for the CREATE TABLE statements.

TODO: Complete the following code cells

In [52]:

```
# Run the query to select 5 random rows from the comments table
```

In [53]:

```
%%sql
SELECT * FROM comments
ORDER BY RANDOM()
LIMIT 5;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[53]:

Id	PostId	Score	Text	CreationDate	UserId
728986	387128	0	@RamiroScorolli: You might want to contact the authors of these articles directly to see if they have derived these results themselves or they reviewed them from somewhere.	2019-01-19 22:51:29	11852

750482	400810	0	@baruuum Thanks for the comment! The way your wrote out the underspecified sparked a new attempt. I've now proved the statement in full generality. (I did this with deterministic regressors X, Z, but the same approach should work with random regressors as well.	2019-04-03 09:53:43	213470
747874	399281	1	The statistic model to be used is determined by the physical model. E.g. when you are dealing with Michaelis-Menten kinetics then the formula is of the form $y = \frac{ax}{b+x}$. Depending on how precise and exact you want to be you could do some approach with more advanced determination of error propagation. But, I guess that most people just use a simple and regular non-linear least squares regression and use the associated confidence intervals (E. g. assuming iid normal errors). This can be done in R with a function nls which is part of the $stats$ package, which is a core package in R.	2019-03-25 10:44:00	164061
747843	399141	0	@Ertxiem - but then the question is, what is 'similar'. It would be better to have some hard arguments :)	2019-03-25 08:58:27	242257
743796	396708	0	A couple of comments. Identifying the null and alternate hypotheses have nothing to do with what you *expect* to happen or what you *want* to happen or what would be *interesting*. So the null hypothesis is that the incomes of men and women are equal. We may not expect the null hypothesis to be true. And maybe we want the null hypothesis to be true. But these considerations don't help you identify the null hypothesis. For probably every test you'll be studying, the null is always that there is no association, no correlation, no difference between groups, and so on.	2019-03-11 09:09:12	166526

In [54]:
```
# insert the queries into the queries table
sql ="""
    SELECT * FROM comments
    ORDER BY RANDOM()
    LIMIT 5;
    """

store_query('task 1','Run the query to select 5 random rows from the comments table'
```

In [55]:
```
# Run the query to select 5 random rows from the posts table
```

In [56]:
```
%%sql
SELECT * FROM posts
ORDER BY RANDOM()
LIMIT 5;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[56]:

Id	PostTypeId	AcceptedAnswerId	ParentId	CreationDate	Score	ViewCount	
389181	2	0	210758	2019-01-25 19:37:38	0	0	<p>Generall Matrix from a cov cases. My reaso which only bec href="https://multi

measurement-
fisher-informati

<p>
unconsciously a
many operatior
being much ea
points, so each dia
database colum
Likelihood Es
statistics about the
This is an apples
reverse enginee

<p>• The sec
attempted to w
only way to val
have been to make
Matrix derived in
resulting matrix
came up with thi
list, but it really is
the data poin

<p>• Even if all
and methods of d
able to determ
more records/t
design, pre
particular mo
Bound. That still
Big Data
populations we're
for a billion livin
the whole huma
rows of cell da
septillion human
such gaps has no
more relia
database tables
the need
somewhat, giv

<p>There may be
can be used to cal
Minimum Desc
taking into accc

href="https://multi
measurement-
fisher-informati
are probably m
Information ca
Information of ag
The kicker is that
to just take t
calculate; resortin
expensive step.
Information from c
most exotic use ca

							href="https://multi measurement- fisher-information- Alexander; Verha “A Tutorial on Fi href=" noreferrer">http:/ have not had a c <p>Model 1: < <p>Model 2: <(<p>Model 3: <(<p>Of course, the model <p>This is my firs <p>I have a set of). However, due to my data. If I r <p>Also, I do not want to test for (am already using
418890	2	0	418874	2019-07-23 23:26:07	1	0	
435566	1	0	0	2019-11-11 14:50:28	1	30	
431275	1	0	0	2019-10-13 17:11:22	0	27	<p>Suppo randor $\mathcal{N}(\mu, \theta)$</ $Y = \frac{1}{n}\sum_{i=1}^{n}(X_i$ container">. container">$d(y)$ = <p>I want to comp <p>This is my first assume, that by <s <p> $R(\theta, d) =$.

							<p>However, I ha <span class="m
							<p>EDIT: Usi
							$R(\theta, d) = E_\theta(\theta^2)$
							<p>I'm rather ne
394975	1	0	0	2019-02-28 22:06:48	0	47	<p>I hav intervention intervention had a experimental stu There is c wondering h variable? H

In [57]:
```
# insert the queries into the queries table
sql ="""
    SELECT * FROM posts
    ORDER BY RANDOM()
    LIMIT 5;
    """

store_query('task 1','Run the query to select 5 random rows from the posts table',sq
```

In [58]:
```
# Run the query to select 5 random rows from the users table
```

In [59]:
```
%%sql
SELECT * FROM users
ORDER BY RANDOM()
LIMIT 5;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[59]:

Id	Reputation	CreationDate	DisplayName	LastAccessDate	WebsiteUrl	Locati
261005	43	2019-09-27 05:29:02	Azka	2019-11-13 13:36:00	None	Nc
84130	111	2015-08-04 19:54:42	Jonathan	2019-11-15 17:17:09	http://www.acridmedia.com	Nc
261089	11	2019-09-28 00:24:21	Ajinkya Ghadge	2019-11-30 21:31:27	None	Nc
240379	101	2019-03-08 10:46:38	relatively_random	2019-11-18 11:18:59	None	Nc
241135	29	2019-03-14 14:21:51	Mathijs	2019-11-27 11:28:21	None	Nc

In [60]:
```
# insert the queries into the queries table
sql ="""
    SELECT * FROM users
    ORDER BY RANDOM()
    LIMIT 5;
    """

store_query('task 1','Run the query to select 5 random rows from the users table',sq
```

In [61]:
```
# Confirm that the queries table was updated correctly
```

In [62]:
```
%%sql
SELECT * FROM queries
```

```
 * sqlite:///chatdata.db
Done.
```

Out[62]:

task	action	query
Single Table Queries	Which 5 users have viewed the most times and what is the sum of those views per user?	SELECT Id, SUM(Views) AS TotalViews FROM Users GROUP BY Id ORDER BY TotalViews DESC LIMIT 5
Task 1	Create table comments	CREATE TABLE "comments" ("Id" INTEGER, "PostId" INTEGER, "Score" INTEGER, "Text" TEXT, "CreationDate" TEXT, "UserId" INTEGER)
Task 1	Create table posts	CREATE TABLE "posts" ("Id" INTEGER NOT NULL PRIMARY KEY, "PostTypeId" INTEGER, "AcceptedAnswerId" INTEGER, "ParentId" INTEGER, "CreationDate" TEXT, "Score" INTEGER, "ViewCount" INTEGER, "Body" TEXT, "OwnerUserId" INTEGER, "OwnerDisplayName" TEXT, "LastEditorUserId" INTEGER, "LastEditorDisplayName" TEXT, "LastEditDate" TEXT, "LastActivityDate" TEXT, "Title" TEXT, "Tags" TEXT, "AnswerCount" INTEGER, "CommentCount" INTEGER,

		"FavoriteCount" INTEGER, "ClosedDate" TEXT, "CommunityOwnedDate" TEXT, FOREIGN KEY(OwnerUserId) REFERENCES users(Id))
Task 1	Create table users	CREATE TABLE "users" ("Id" INTEGER NOT NULL PRIMARY KEY, "Reputation" INTEGER, "CreationDate" TEXT, "DisplayName" TEXT, "LastAccessDate" TEXT, "WebsiteUrl" TEXT, "Location" TEXT, "AboutMe" TEXT, "Views" INTEGER, "UpVotes" INTEGER, "DownVotes" INTEGER, "ProfileImageUrl" TEXT, "AccountId" INTEGER)
Task 1	Count the number of rows in the comments table	SELECT count(*) FROM "comments"
Task 1	Count the number of rows in the users table	SELECT count(*) FROM "users"
Task 1	Count the number of rows in the posts table	SELECT count(*) FROM "posts"
task 1	Run the query to select 5 random rows from the comments table	SELECT * FROM comments ORDER BY RANDOM() LIMIT 5;
task 1	Run the query to select 5 random rows from the posts table	SELECT * FROM posts ORDER BY RANDOM() LIMIT 5;
task 1	Run the query to select 5 random rows from the users table	SELECT * FROM users ORDER BY RANDOM() LIMIT 5;

Task 2: Create Single Table Queries

Lifecycle Stage: Analyze

We can now start the analysis with our single-table queries. First we need to create a new computed column to help with one of the queries. The code below creates a column called LEN_BODY which is the length of the BODY text:

In [63]:
```
%%sql
ALTER TABLE posts ADD COLUMN LEN_BODY INT
```

```
 * sqlite:///chatdata.db
Done.
```

Out[63]: []

In [64]:
```
%%sql
UPDATE posts SET LEN_BODY = LENGTH(BODY)
```

```
 * sqlite:///chatdata.db
42234 rows affected.
```

Out[64]: []

Single Table Queries

From the Template SQL Queries there are a series of queries requested. You are responsible for coding the result. Use the pattern above to accomplish the coding objective. Copy and paste the question (column B Action) into a markdown cell and answer the query. Use the pattern that you have been using above.

TODO: Enter your code below. Just keep on inserting cells as you need them.

How many posts have 0 comments?

In [65]:
```
%%sql
SELECT count(*) as Count FROM posts WHERE CommentCount = 0;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[65]:

Count
21713

In [66]:
```
# Confirm using Pandas
count=posts[posts['CommentCount']==0]
len(count)
```

Out[66]: 21713

In [67]:
```
# insert the queries into the queries table
sql ="""
    SELECT count(*) as Count FROM posts
    WHERE CommentCount = 0;
    """

store_query('task 2','How many posts have 0 comments',sql)
```

How many posts have 1 comments?

In [68]:
```
%%sql
SELECT count(*) as Count FROM posts WHERE CommentCount = 1;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[68]:

Count
6460

In [69]:
```
# Confirm using Pandas
count=posts[posts['CommentCount']==1]
len(count)
```

Out[69]: 6460

In [70]:
```
# insert the queries into the queries table
sql ="""
    SELECT count(*) as Count FROM posts
    WHERE CommentCount = 1;
    """

store_query('task 2','How many posts have 1 comments',sql)
```

How many posts have 2 comments or more?

In [71]:
```
%%sql
SELECT count(*) as Count FROM posts WHERE CommentCount >= 2;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[71]:

Count
14061

In [72]:
```
# Confirm using Pandas
count=posts[posts['CommentCount']>=2]
len(count)
```

Out[72]: 14061

In [73]:
```
# insert the queries into the queries table
sql ="""
    SELECT count(*) as Count FROM posts
    WHERE CommentCount >= 2;
    """

store_query('task 2','How many posts have 2 comments or more',sql)
```

Find the 5 posts with the highest viewcount

In [74]:
```
%%sql
SELECT * FROM posts
    ORDER BY ViewCount DESC
        LIMIT 5;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[74]:

Id	PostTypeId	AcceptedAnswerId	ParentId	CreationDate	Score	ViewCount

388566	1	388582	0	2019-01-22 15:16:47	56	19542	Statement C ca Statement Twc as a r <p>Now, I pe difference at all be When w interchangeable I've been chall now, and am <p>My default i people drawn u population of hum them to die as a re do consider this qu <p>My c <p>Q1) Is my equivalent t <p>Q2) Is unu my defa <p>Q3) If you such that to state the first is mis please provide a <p>Let's put aside does not specifical assume that that us also put aside a of the claim itself <p>As best I ca heard so far seem to different in <p>For the interpret it as as 1 of deaths caused b reason, default to the second along set of 80 people, c die in a car accider equivalent claim). interpretation of be to read it as number of peop == number of de I'm not interpretation (stronger assur

							innate statistical
394118	1	394128	0	2019-02-24 14:07:11	64	16317	<p>A human c instances of a ca reasonable accura etc. When identify trams an seen just a few. Si one with eac network was not t <p>What is it that missing that prev learn way qu
431370	1	431397	0	2019-10-14 11:29:21	77	11723	<p>It seems v people that a gi high accuracy (say more false positive situations, namely positives is v <p>I see people m when ar screenings, or measures etc succinctly des
398646	1	398653	0	2019-03-21 01:19:52	61	9850	<p>Does this ph have a name? good, terse, jargo would help me e <p>Apologies if t this. If s <p>The title c href="https://www 019-00857-9" re up against statis <p>Valentin Amrh McShane and m for an end to hype <p>and later c <p>Again, we a values, confidenc measures — the dichotomizatio not, as well as c statistical measure

<p>I think I car
does not say
because one "
other does not. Bu
much more depth

<p>Towards
summary i
summarize th
those of us wh

<p>When talking
be

<p>First, ju
the values m
given the assump
outside it are in

<p>Sec
equally compa

<p>Third
which it ca
compute

<p>Last, a
humble: compa
the correctness
used to com

href="https:/

src="https:/
alt="Nature: Scient

434128	1	434579	0	2019-11-01 13:07:36	73	6718	<p>I am designing analysis with a l program aims t basic tasks in da summarizatior

<p>I would like
world examples
intuition
necessary. I'
statistical fails", but
The data involv

<p>A perfect exa
is the Berkel
illustrates Simpsor
is memo

<p>Historical ca
Snow's analysis of
is a g

<p>There are a l
data (selection b
medical s

<p>A lot of "stat
of variable select
interested in p
areas -- li

In [75]:
```
# Confirm using Pandas
posts.sort_values(by='ViewCount',ascending=False)[:5]
```

Out[75]:

	Id	PostTypeId	AcceptedAnswerId	ParentId	CreationDate	Score	ViewCount	
2756	388566	1	388582	0	2019-01-22 15:16:47	56	19542	\n<li
One (S1):
7166	394118	1	394128	0	2019-02-24 14:07:11	64	16317	<p>A hun
age 2 ne
36077	431370	1	431397	0	2019-10-14 11:29:21	77	11723	<p>It
counter
10833	398646	1	398653	0	2019-03-21 01:19:52	61	9850	<p>The
Commer
38317	434128	1	434579	0	2019-11-01 13:07:36	73	6718	<p>I am
one year

5 rows × 21 columns

In [76]:
```
# insert the queries into the queries table
sql ="""
    SELECT * FROM posts
    ORDER BY ViewCount DESC
    LIMIT 5;
    """

store_query('task 2','Find the 5 posts with the highest viewcount',sql)
```

Find the top 5 posts with the highest scores

In [77]:
```
%%sql
SELECT * FROM posts
    ORDER BY Score DESC
        LIMIT 5;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[77]:

Id	PostTypeId	AcceptedAnswerId	ParentId	CreationDate	Score	ViewCount	
431397	2	0	431370	2019-10-14 14:29:36	101	0	<p>Yes there fallacy</stro paradox</strong <a href="htt
394128	2	0	394118	2019-02-24 15:44:44	100	0	<p>I caution aga biological an "neural networks' into expecti learning sh biologica
							<p>As an exampl the reasoning in t neural networ provided you amount of tr (car/tram/bike/ai
							<p>By contrast out a car the c after it has seer obviously differen accounts for vanilla image class of picking up ob after "birth." I thi (1) the relative self-teaching me
							<p>The origin body of the quest examples." Rela trained using comi
							<p>I will
							<h1>"How d image benchma
							<p>For the sake o because it i portion is compose

per class. Each ima
the labeled ima
fps video, yo

<p>A child of 2
daily has roughly
direct observa
adults (labels). (Th
how many minutes
the world.) Mor
many objects bey

<p>So there are a
exposure to mo
data than the C
volume are
models in general
a neural network is
neural network
training dat
resolution availab
10 images, so the

<p>The CIFAR-1
because th
multiple passes
will see, using bin
three-dime
different lightin

<p>The anecdote

<h1>"How can neu

<p>A child is enc
that new ca
with

href="/questions
title="show ques
rel="tag">trans
adaptation
<p>In comme
and few-shot lea

href="
clas
'reinforcemen
learning add
perspective, essent
error experimen
spec

<p>It's probably
paradigms are ge
new computer visio
models to new

							because the p instances of malwa index the inte goals of a child one is done in a in organic r
							<p>As an aside the CIFAR-10 pr recognize 6000 this wouldn't there would stil divers <p>*We don't pr
426878	2	0	426873	2019-09-11 23:23:31	93	0	<p> class easy</s direct ma
							<p>This is a simplicity of logi
							<p>What logis <span class= multiply them interesting thing input and ou weight corres class="math-co account when co by taking the wei <span class= image resolution important for the <p><img src='
							<p>Now take a loo two digits (i.e pixel's intensity c
							<p>Nov class="math-conta that's empty picked up on image, it counts to recognize zero and high-level f

<p>Same thi
. It alway

<p>The rest of the
little imagir
container">2</s
, the <s
the <span class="
numbers are a bi
the logistic r

<p>Through th
very good chan

<p>The code to re

<pre class="la

from tensorflow

mnist

x = t
y =

cross_e

tf.train.GradientDes

correct_pred =
accuracy = tf.r

x_batch, y
sess.run(

l_tr, a_tr = sess

							l_ts, a_ts = sess
							print('Test /
							plt.imshow Duminil, cmap
							<p>To me "1 in The denominator i
388578	2	0	388566	2019-01-22 15:48:47	80	0	<p>There's amb You really mean "1 can just as easily b
							<p>I'm all probability or fr about the
							<p>It seems given diagnosti generate massive some situ positives is very
431370	1	431397	0	2019-10-14 11:29:21	77	11723	<p>I see people r for wider p surveillanc succinctly de
							<p>Does this ph Failing th intuition/ex
							<p>Apologi please

In [78]:
```
# Confirm using Pandas
posts.sort_values(by='Score',ascending=False)[:5]
```

Out[78]:

	Id	PostTypeId	AcceptedAnswerId	ParentId	CreationDate	Score	ViewCount	
36101	431397	2	0	431370	2019-10-14 14:29:36	101	0	<p General
7174	394128	2	0	394118	2019-02-24 15:44:44	100	0	<p>I c exp
32528	426878	2	0	426873	2019-09-11 23:23:31	93	0	t Even th
2767	388578	2	0	388566	2019-01-22 15:48:47	80	0	<p> deaths..
36077	431370	1	431397	0	2019-10-14 11:29:21	77	11723	<p coun

5 rows × 21 columns

In [79]:

```
# insert the queries into the queries table
sql ="""
    SELECT * FROM posts
    ORDER BY Score DESC
    LIMIT 5;
    """

store_query('task 2','Find the 5 posts with the highest scores',sql)
```

What are the 5 most frequent scores on posts?

In [80]:

```
%%sql
SELECT Score ,count(*) as count FROM posts
    GROUP BY Score
        ORDER BY count DESC
            LIMIT 5;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[80]:

Score	count
0	19888
1	11867
2	5094
3	2228
4	1059

In [81]:

```
# Confirm using Pandas
posts.groupby("Score").count().sort_values(by='Id',ascending=False)['Id'][0:5]
```

Out[81]:
```
Score
0    19888
1    11867
2     5094
3     2228
4     1059
Name: Id, dtype: int64
```

In [82]:
```
# insert the queries into the queries table
sql ="""
    SELECT Score ,count(*) as count FROM posts
    GROUP BY Score
    ORDER BY count DESC
    LIMIT 5;
    """

store_query('task 2','What are the 5 most frequent scores on posts',sql)
```

How many posts have the keyword "data" in their tags?

In [83]:
```
%%sql
SELECT count(*) as count FROM posts
    WHERE Tags LIKE "%data%";
```

```
 * sqlite:///chatdata.db
Done.
```

Out[83]:

count
2242

In [84]:
```
# Confirm using Pandas
len(posts[posts['Tags'].str.contains('data')==True])
```

Out[84]:
```
2242
```

In [85]:
```
# insert the queries into the queries table
sql ="""
    SELECT count(*) as count FROM posts
    WHERE Tags LIKE "%data%";
    """

store_query('task 2','How many posts have the keyword "data" in their tags',sql)
```

What are the 5 most frequent commentcount for posts?

In [86]:
```
%%sql
SELECT CommentCount ,count(*) as count FROM posts
    GROUP BY CommentCount
        ORDER BY count DESC
            LIMIT 5;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[86]:

CommentCount	count
0	21713
1	6460

2	4966
3	3063
4	2026

In [87]:
```
# Confirm using Pandas
posts.groupby("CommentCount").count().sort_values(by='Id',ascending=False)['Id'][0:5
```

Out[87]:
```
CommentCount
0    21713
1     6460
2     4966
3     3063
4     2026
Name: Id, dtype: int64
```

In [88]:
```
# insert the queries into the queries table
sql ="""
    SELECT CommentCount ,count(*) as count FROM posts
    GROUP BY CommentCount
    ORDER BY count DESC
    LIMIT 5;
    """

store_query('task 2','What are the 5 most frequent commentcount for posts',sql)
```

How many posts have an accepted answer?

In [89]:
```
%%sql
SELECT count(*) as Count FROM posts WHERE AcceptedAnswerId <> 0;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[89]:

Count
5341

In [90]:
```
# Confirm using Pandas
len(posts[posts['AcceptedAnswerId']!=0])
```

Out[90]:
```
5341
```

In [91]:
```
# insert the queries into the queries table
sql ="""
    SELECT count(*) as Count FROM posts
    WHERE AcceptedAnswerId <> 0;
    """

store_query('task 2','How many posts have an accepted answer',sql)
```

What is the average reputation of table users?

In [92]:
```
%%sql
SELECT ROUND(AVG(Reputation),2) as AverageReputation FROM users;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[92]:

AverageReputation
312.35

In [93]:
```
# Confirm using Pandas
round(users['Reputation'].mean(),2)
```

Out[93]:
```
312.35
```

In [94]:
```
# insert the queries into the queries table
sql ="""
    SELECT ROUND(AVG(Reputation),2) as AverageReputation FROM users;
    """

store_query('task 2','What is the average reputation of table users',sql)
```

What are the min and max reputation of users?

In [95]:
```
%%sql
SELECT ROUND(MIN(Reputation),2) as MinReputation,
ROUND(MAX(Reputation),2) as MaxReputation FROM users;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[95]:

MinReputation	MaxReputation
1.0	228662.0

In [96]:
```
# Confirm using Pandas
round(users['Reputation'].min(),2),round(users['Reputation'].max(),2)
```

Out[96]:
```
(1, 228662)
```

In [97]:
```
# insert the queries into the queries table
sql ="""
    SELECT ROUND(MIN(Reputation),2) as MinReputation,
    ROUND(MAX(Reputation),2) as MaxReputation FROM users;
    """

store_query('task 2','What are the min and max reputation of users',sql)
```

What is the length of the body of 5 most viewed posts?

In [98]:
```
%%sql
SELECT ViewCount as MostViewed ,Len_Body as LengthBody FROM posts
    ORDER BY ViewCount DESC
        LIMIT 5;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[98]:

MostViewed	LengthBody
19542	2270

16317	512
11723	811
9850	2148
6718	1172

In [99]:
```
# Confirm using Pandas
post=pd.read_sql("""Select * From posts""",con)
post[['ViewCount','LEN_BODY']].sort_values(by='ViewCount',ascending=False)[0:5]
```

Out[99]:

	ViewCount	LEN_BODY
2695	19542	2270.0
7105	16317	512.0
36077	11723	811.0
10772	9850	2148.0
38317	6718	1172.0

In [100…
```
# insert the queries into the queries table
sql ="""
    SELECT ViewCount as MostViewed ,Len_Body as LengthBody FROM posts
    ORDER BY ViewCount DESC
    LIMIT 5;
    """

store_query('task 2','What is the length of the body of 5 most viewed posts',sql)
```

How many different locations are there in the users table?

In [101…
```
%%sql
SELECT count(DISTINCT(location)) as LocationCount FROM (
    SELECT * FROM users
        GROUP BY location
    );
```

```
 * sqlite:///chatdata.db
Done.
```

Out[101…

LocationCount
1900

In [102…
```
# Confirm using Pandas
users['Location'].nunique()
```

Out[102…
```
1900
```

In [103…
```
# insert the queries into the queries table
sql ="""
    SELECT count(DISTINCT(location)) as LocationCount FROM (
    SELECT * FROM users
        GROUP BY location
    );
```

```
    """

store_query('task 2','How many different locations are there in the users table',sql
```

What are the top 5 locations of users?

In [104…

```
%%sql
SELECT Location,count(*) as LocationCount FROM users
    GROUP BY Location
        ORDER BY LocationCount DESC
            LIMIT 5;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[104…

Location	LocationCount
None	13335
Germany	117
India	100
United States	69
Paris, France	66

In [105…

```
# Confirm using Pandas
users.groupby('Location',dropna=False).count().sort_values('Id',ascending=False)['Id
```

Out[105…

```
Location
NaN              13335
Germany            117
India              100
United States       69
Paris, France       66
Name: Id, dtype: int64
```

In [106…

```
# insert the queries into the queries table
sql ="""
    SELECT Location,count(*) as LocationCount FROM users
    GROUP BY Location
    ORDER BY LocationCount DESC
    LIMIT 5;
    """

store_query('task 2','What are the top 5 locations of users',sql)
```

Rank the days of the week from highest to lowest in terms of the volume of ViewCount as a percentage.

In [107…

```
%%sql
SELECT  CASE
            WHEN WeekDayNumber = 0 THEN 'Sunday'
            WHEN WeekDayNumber = 1 THEN 'Monday'
            WHEN WeekDayNumber = 2 THEN 'Tuesday'
            WHEN WeekDayNumber = 3 THEN 'Wednesday'
            WHEN WeekDayNumber = 4 THEN 'Thursday'
            WHEN WeekDayNumber = 5 THEN 'Friday'
            ELSE 'Saturday'
    END as WeekDay,Percentage_ViewCount
FROM (SELECT CAST (strftime('%w', CreationDate) AS INTEGER) as WeekDayNumber,
```

```
            round( cast(sum(ViewCount) as float)/cast((select sum(ViewCount) from pos
FROM posts
GROUP BY WeekDayNumber
ORDER BY Percentage_ViewCount DESC)
```

```
 * sqlite:///chatdata.db
Done.
```

Out[107…

WeekDay	Percentage_ViewCount
Thursday	16.82
Wednesday	16.77
Tuesday	16.26
Monday	15.81
Friday	13.56
Sunday	11.9
Saturday	8.89

In [108…

```
# Confirm using Pandas
post['CreationDate']=pd.to_datetime(post.CreationDate, format='%Y-%m-%d %H:%M:%S')
post['week']=post['CreationDate'].dt.dayofweek
sum=post.groupby('week')['ViewCount'].sum()
sum2=sum.sum()
newsum=sum/sum2*100
round(newsum.sort_values(ascending=False),2)
```

Out[108…

```
week
3    16.82
2    16.77
1    16.26
0    15.81
4    13.56
6    11.90
5     8.89
Name: ViewCount, dtype: float64
```

In [109…

```
# insert the queries into the queries table
sql ="""
SELECT  CASE
            WHEN WeekDayNumber = 0 THEN 'Sunday'
            WHEN WeekDayNumber = 1 THEN 'Monday'
            WHEN WeekDayNumber = 2 THEN 'Tuesday'
            WHEN WeekDayNumber = 3 THEN 'Wednesday'
            WHEN WeekDayNumber = 4 THEN 'Thursday'
            WHEN WeekDayNumber = 5 THEN 'Friday'
            ELSE 'Saturday'
    END as WeekDay,Percentage_ViewCount
FROM (SELECT CAST (strftime('%w', CreationDate) AS INTEGER) as WeekDayNumber,
          round( cast(sum(ViewCount) as float)/cast((select sum(ViewCount) from pos
FROM posts
GROUP BY WeekDayNumber
ORDER BY Percentage_ViewCount DESC);
    """

store_query('task 2','Rank the days of the week from highest to lowest in terms of t
```

Task 3: Cross Table Queries

Lifecycle Stage: Analyze

Let's continue the analysis with our multi-table queries.

Cross Table Queries

From the Template SQL Queries there are a series of queries requested. You are responsible for coding the result. Use the pattern above to accomplish the coding objective. Copy and paste the question (column B Action) into a markdown cell and answer the query. Use the pattern that you have been using above.

TODO: Enter your code below. Just keep on inserting cells as you need them.

How many posts have been created by a user that has a filled out the "AboutMe" section?

In [110…

```
%%sql
SELECT count(*) FROM users
INNER JOIN posts ON posts.OwnerUserId = users.Id
WHERE AboutMe <> "NULL";
```

```
 * sqlite:///chatdata.db
Done.
```

Out[110…

count(*)
17189

In [111…

```
# Confirm using Pandas
mergeT= posts.merge(users,how='inner',left_on='OwnerUserId',right_on='Id')
len(mergeT[mergeT['AboutMe'].notnull()])
```

Out[111…

```
17189
```

In [112…

```
# insert the queries into the queries table
sql ="""
    SELECT count(*) FROM users
    INNER JOIN posts ON posts.OwnerUserId = users.Id
    WHERE AboutMe <> "NULL";
    """

store_query('task 3','How many posts have been created by a user that has a filled o
```

Considering only the users with an "AboutMe," how many posts are there per user?

In [113…

```
%%sql
SELECT OwnerUserId,count(*)
FROM (SELECT * FROM users
INNER JOIN posts ON posts.OwnerUserId = users.Id
WHERE AboutMe <> "NULL")
GROUP BY OwnerUserId;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[113…

OwnerUserId	count(*)

-1	1
22	2
25	4
29	1
104	1
159	9
161	1
183	5
196	1
199	1
211	19
230	2
253	3
511	1
582	1
600	1
601	1
686	386
795	8
805	230
809	1
858	5
887	4
892	28
919	203
1036	1
1098	1
1191	12
1212	3
1219	1
1329	13
1352	285
1375	1
1390	51
1739	2
1764	38
1818	1

1934	11
1973	1
2024	1
2040	6
2116	1
2126	59
2129	3
2244	1
2392	19
2485	1
2546	3
2669	11
2860	2
2894	1
3002	1
3014	1
3058	3
3219	1
3277	5
3287	1
3382	274
3385	1
3437	1
3443	1
3601	3
3770	1
3812	1
3834	1
3919	8
4060	1
4087	1
4195	2
4253	71
4370	2
4544	1
4598	77
4704	2

4737	2
4901	1
4925	1
4976	1
5053	1
5176	59
5191	3
5221	1
5410	1
5419	1
5503	9
5535	1
5619	3
5739	4
5827	1
5829	5
5836	2
6040	6
6192	1
6244	1
6329	1
6378	1
6454	1
6633	38
6867	2
6872	1
6965	3
7071	35
7194	5
7224	233
7250	8
7290	35
7486	96
7555	44
7616	10
7641	2
7644	2

7828	177
7835	7
7947	1
7962	84
7980	2
8013	212
8246	1
8291	1
8297	1
8336	84
8371	1
8401	2
8433	1
8451	1
8515	2
8605	2
9007	1
9136	1
9162	15
9167	3
9233	1
9320	1
9330	2
9354	1
9394	9
9494	1
9496	1
9519	2
9964	43
10183	1
10361	1
10416	8
10461	1
10479	10
10519	1
10619	7
10639	1

10688	2
10790	1
10812	19
10826	1
10849	4
10875	1
10908	1
11004	5
11032	2
11284	1
11363	1
11404	1
11438	1
11633	2
11697	1
11708	1
11849	4
11852	85
11867	9
11887	435
11930	1
11969	1
11984	1
12147	6
12359	1
12603	6
12615	1
12647	1
12833	3
12923	19
13047	5
13071	1
13351	1
13409	3
13429	5
13526	1
13538	2

13562	2
13610	1
13634	11
13668	1
13680	1
13849	2
13987	1
13991	1
14076	8
14187	1
14188	2
14233	5
14305	1
14306	12
14346	3
14421	1
14465	1
14671	1
14730	3
14803	1
15943	1
15973	5
15997	1
16049	1
16087	1
16175	3
16229	1
16253	1
16319	2
16367	47
16491	2
16709	9
16852	1
16897	1
16939	5
16974	3
17023	3

17060	1
17072	37
17230	6
17295	4
17459	4
17661	1
17672	1
17760	5
17908	11
17924	1
17935	1
18152	2
18168	1
18192	5
18215	1
18331	2
18364	1
18417	14
18465	1
18494	1
18539	6
18549	1
18578	1
18591	2
19815	2
19891	1
19960	5
20148	2
20213	2
20221	24
20256	1
20352	2
20395	1
20499	1
20519	6
20839	2
20872	1

20921	1
20980	10
20995	1
21095	1
21129	1
21356	1
21389	1
21743	1
21817	3
21846	22
21967	2
22047	46
22075	1
22228	1
22311	140
22403	3
22452	6
22531	5
23732	1
23770	1
23772	1
23801	7
23853	22
24000	1
24028	4
24030	3
24073	6
24097	2
24194	1
24432	1
24462	1
24575	1
24612	3
24617	1
24669	1
24778	1
24900	1

24905	11
25162	1
25186	7
25212	2
25222	1
25224	1
25311	1
25424	2
25629	1
25741	2
25794	1
26008	1
26179	1
26190	1
26327	1
26392	1
26501	1
26879	2
26909	2
26948	171
26959	1
27051	1
27148	4
27158	1
27276	1
27433	6
27512	1
27535	1
27608	2
27782	5
27881	4
28185	4
28218	2
28370	1
28436	2
28666	8
28740	17

28746	32
28943	1
28978	7
29036	1
29187	1
29234	10
29238	1
29316	1
29537	2
29596	2
29655	2
29690	1
29694	5
29770	3
29949	6
30005	23
30150	1
30187	1
30202	3
30351	18
30451	3
30545	1
30566	2
30589	1
30611	1
30754	1
30759	2
30822	9
30837	1
30838	2
31007	2
31313	13
31420	1
31634	1
31740	2
31978	13
32037	3

32065	1
32184	1
32304	1
32310	1
32462	1
32477	8
34638	3
34826	3
34927	2
34994	1
35026	2
35034	2
35115	1
35131	5
35396	1
35398	16
35413	1
35448	1
35452	34
35483	3
35584	36
35639	2
35664	1
35715	1
35802	2
35989	230
35998	1
36031	2
36041	124
36113	1
36206	3
36245	1
36251	1
36268	1
36294	2
36312	2
36381	1

36415	1
36423	4
36447	1
36462	1
36519	1
36540	10
36552	1
36593	1
36602	1
36682	6
36769	2
36867	1
37041	2
37243	2
37280	2
37306	2
37405	2
37420	1
37428	1
37549	1
37632	11
37793	19
37796	6
37888	1
38007	2
38012	1
38063	3
38160	2
38425	1
39582	2
39601	7
39630	19
39662	3
39684	2
39710	3
39770	4
39778	1

39785	1
39803	1
39829	6
39849	1
40036	2
40065	1
40242	7
40252	21
40295	1
40313	6
40316	1
40383	2
40602	1
40634	1
40747	10
41167	4
41197	1
41220	1
41251	1
41289	2
41317	1
41749	2
41870	1
42004	5
42264	1
42280	1
42429	1
42651	1
42884	1
42902	1
42955	1
43120	1
43159	4
43288	1
43379	1
43450	7
43559	1

43625	4
43634	1
43745	1
43775	20
43812	1
43834	1
43849	2
43890	1
43953	1
44075	1
44112	2
44129	4
44136	1
44137	1
44269	24
44591	1
44731	1
44760	4
45374	1
45654	1
45752	1
45804	1
45904	2
45916	1
45941	2
46098	2
46422	1
46478	8
46652	2
46790	1
46932	1
48052	1
48101	9
48184	1
48250	37
48279	2
48448	5

48515	1
48591	26
48610	7
48754	1
49024	3
49037	1
49071	1
49132	1
49251	1
49303	3
49358	1
49507	1
49611	1
49613	2
49793	1
51872	1
51993	2
52007	1
52103	1
52150	1
52151	2
52234	1
52261	1
52306	1
52367	5
52372	1
52396	2
52562	1
52898	1
53098	5
53215	1
53419	1
53438	2
53456	6
53690	98
53866	1
53922	1

54083	7
54091	6
54092	1
54184	1
54186	1
54426	3
54736	1
54740	8
54883	1
54923	2
55245	2
55410	2
55412	1
55422	1
55548	2
55587	2
55661	1
55831	2
56208	1
56211	13
56282	1
56352	2
56364	1
56472	1
56828	1
56844	1
56940	1
57111	13
57345	2
58450	5
58560	1
58602	1
58792	1
59093	5
59188	2
59395	2
59549	1

59602	1
59641	1
59667	2
59920	1
60065	3
60066	1
60233	1
60256	1
60261	23
60323	3
60593	1
60613	13
60909	9
60997	1
61018	6
61053	1
61283	1
61332	2
61389	3
61496	8
61510	1
61705	2
61780	2
61826	1
61867	1
61883	1
61927	1
61998	6
62021	1
62060	5
62245	5
62350	4
62549	4
63666	2
63677	5
63691	1
63739	2

63862	1
63868	1
64093	1
64098	24
64122	2
64136	6
64249	1
64254	2
64263	14
64370	1
64380	1
64544	2
64552	6
64572	2
64591	1
66266	1
66341	3
66383	5
66428	4
66461	1
66597	1
66741	1
66768	1
66773	1
66985	2
67042	1
67137	13
67363	1
67395	1
67427	1
67556	2
67613	1
67799	46
68006	3
68057	3
68124	1
68146	1

68224	1
68268	7
68301	7
68313	2
68576	10
68965	10
69010	1
69015	1
69177	1
69230	1
69240	1
69247	2
69260	1
69374	1
69398	1
69435	1
69625	1
69644	1
69673	4
69993	1
70093	2
70202	1
70304	1
70569	1
70690	1
70879	6
71012	3
71343	1
71348	1
71365	1
71439	1
71524	2
71565	1
72126	8
72127	1
72305	1
72352	4

72684	1
72690	2
72782	2
72959	1
73129	27
73219	1
73619	3
73682	1
73794	4
74054	3
74164	1
74396	1
74500	18
74563	1
74737	10
74752	1
74762	2
74789	1
75173	10
75224	1
76649	2
76815	1
76915	2
76937	3
76981	43
77202	1
77427	1
77446	6
77478	3
77508	2
77630	2
77838	1
77852	4
78004	3
78316	1
78398	5
78492	1

78511	1
78564	2
78565	1
78569	2
78575	5
78726	1
78773	1
78846	1
78857	7
78886	1
78964	3
79079	1
79100	8
79175	2
79186	1
79351	1
79526	1
79611	1
79671	2
79749	1
79768	2
80086	2
80262	1
80429	1
80624	3
80904	3
80914	1
80915	1
80956	1
81019	3
81243	2
81360	1
81432	4
81865	11
81883	1
81943	1
81974	16

82122	1
82217	1
82224	1
82338	5
82389	1
82393	1
82404	3
82750	3
82777	1
82897	2
82898	5
82939	1
83065	10
83419	2
83571	2
83578	2
83585	2
83845	1
83923	1
83937	1
83956	2
84006	1
84114	1
84189	1
84253	2
84301	1
85590	1
85665	545
85767	6
85906	4
85943	1
85987	1
86001	1
86112	1
86202	1
86652	122
86816	2

86935	1
86998	1
87062	1
87164	2
87166	1
87305	10
87437	1
87448	1
87558	1
87568	2
87579	1
88719	1
88723	2
88764	1
88790	8
88869	1
89012	5
89016	1
89406	1
89458	1
89466	1
89471	6
89549	1
89612	1
89653	1
89777	1
89799	4
89891	1
89906	1
90006	1
90102	2
90298	1
90427	2
90527	1
90563	1
90621	3
90687	2

90731	1
90833	2
91142	1
91336	1
91355	1
91387	1
91427	1
91469	1
91558	1
91627	1
91695	2
91762	25
91928	2
92082	1
92235	1
92336	1
92417	1
92423	7
92493	3
92505	1
92522	9
92769	2
92811	5
92932	2
92979	1
93018	1
93054	2
93079	2
93111	2
93223	1
93302	1
93401	2
93480	1
93498	2
93595	2
93623	1
93661	1

93663	2
93702	1
93788	7
93802	1
93965	1
94052	5
94117	1
94368	1
94379	1
94700	2
94747	3
94790	1
94799	2
94994	1
95000	2
95008	2
95027	1
95098	19
95187	1
95436	2
95445	1
95505	1
95569	9
95889	1
95965	3
95978	1
96023	2
96105	1
96216	4
96296	1
96336	1
96404	1
96533	2
96646	1
96662	1
96761	1
96825	5

96830	1
96901	1
96971	1
96972	1
96979	1
97205	1
97470	4
97501	1
97577	11
97594	1
97653	1
97810	2
97818	1
97925	6
98094	1
98156	1
98188	2
98271	1
98379	1
98405	1
98420	1
98464	2
98619	1
98685	1
98694	1
98942	2
98943	1
99274	32
99438	1
99565	3
99612	1
99688	2
99791	3
99818	13
99882	1
99938	1
100048	1

100080	2
100121	1
100235	1
100279	1
100305	1
100366	1
100369	3
100439	4
100507	4
100673	21
100853	1
101097	2
101144	7
101156	2
101183	1
101426	72
101455	1
101486	1
101791	1
101801	3
101905	1
101974	4
102265	14
102291	1
102541	16
102554	2
102879	4
103003	6
103034	1
103055	1
103090	1
103153	68
103197	3
103257	1
103342	1
103407	11
103525	1

103653	9
103671	1
105160	1
105241	1
105344	1
105368	8
105510	13
105868	4
106001	2
106101	1
106118	1
106247	1
106433	1
106606	1
106747	1
106978	7
107010	1
107126	3
107209	1
107356	1
107572	1
108308	1
108524	1
108729	1
108755	1
108877	47
108944	2
108959	1
109055	1
109273	1
109372	18
109561	1
109646	1
109647	2
109718	2
109945	2
110211	1

110248	1
110298	3
110423	4
110833	20
110896	1
110940	1
111054	1
111081	1
111105	1
111259	155
111319	2
111380	10
111389	9
111402	2
111683	2
111684	1
112094	3
112219	1
112553	2
112731	1
112733	8
112912	10
112987	3
113273	1
113337	1
113777	12
113779	1
113793	1
113819	1
113959	3
114025	1
114028	1
114055	1
114062	2
114237	1
114271	1
114345	8

114486	1
114773	2
114903	1
114940	4
115107	9
115361	1
115497	1
115634	133
115704	1
115764	2
116047	2
116056	1
116195	180
116507	1
116549	2
116757	1
116837	1
116844	2
117143	1
117159	7
117509	1
117573	3
117574	11
117705	19
117731	1
118959	3
119202	1
119251	1
119261	71
119293	1
119499	1
119515	1
119523	1
119920	20
119946	2
120118	2
120468	1

120566	2
120569	2
120639	1
120781	1
120788	10
120831	14
120913	1
121027	1
121128	15
121270	25
121489	2
121490	5
121493	1
121522	179
121883	1
121917	2
122024	1
122077	2
122137	2
122209	1
122236	2
122311	10
122381	1
122448	1
122472	5
122527	1
122792	2
123056	2
123097	7
123199	1
123277	2
123462	1
123694	1
123840	1
124079	1
124097	1
124361	1

124547	3
124548	1
124590	1
124694	3
124771	1
124881	1
124899	1
125417	2
125431	1
125481	16
126589	3
126619	1
126648	1
126741	3
126755	1
126931	33
126943	1
126972	1
127258	2
127281	39
127482	1
127724	2
127790	1
127913	5
128078	2
128284	21
128303	1
128318	1
128610	39
128677	7
128824	2
128897	1
128901	1
128938	1
128939	8
129066	1
129321	22

129343	2
129418	1
129463	1
129880	1
129973	1
130078	2
130153	2
130195	3
130523	1
131079	1
131293	1
131377	2
131402	3
131412	1
131477	1
131490	1
131834	1
131965	1
131977	1
132024	2
132098	1
132099	1
132195	6
132217	12
132253	1
132448	1
132564	4
132593	1
132651	4
132772	1
132808	4
132817	2
132899	3
132971	6
132997	1
133080	1
133284	5

133336	4
133377	2
133457	2
133603	1
133645	1
133695	1
133720	3
133742	6
133901	1
134183	2
134754	1
134859	3
134925	1
135291	6
135324	1
135392	2
135567	3
135866	5
136102	4
136150	1
136282	3
136563	2
136596	1
136746	7
136845	5
137008	2
137066	23
137088	3
137200	2
137207	1
137299	3
137387	1
137431	2
137591	8
137593	16
137616	1
137746	1

137828	2
137921	31
137926	8
138067	3
138135	4
138143	1
138249	1
138341	2
138414	4
138601	1
138616	1
138658	1
138708	1
138831	1
138984	1
139186	1
139232	3
139291	1
139325	3
139501	1
139605	2
139869	3
139878	1
139903	1
139982	1
140058	1
140215	1
140300	60
140365	10
140368	2
140381	1
140662	4
140775	1
140832	2
140965	2
141278	1
141327	1

141486	1
141526	2
141599	13
141683	1
141708	1
141867	3
141956	7
142022	1
142039	3
142062	1
142200	2
142281	1
142294	1
142317	4
142322	2
142334	1
142789	1
142914	8
142975	1
142976	8
142978	8
143028	6
143089	1
143094	8
143190	2
143414	3
143446	15
143472	3
143496	1
143546	1
143602	1
143653	2
143973	1
144019	1
144020	1
144096	1
144296	2

144600	2
144604	3
144733	2
145124	1
145263	2
145272	1
145370	1
145425	1
145472	6
145508	1
145620	1
145666	1
145856	2
145920	1
145930	1
145945	1
145995	3
146147	1
146404	2
146552	3
146599	2
146642	2
146898	1
146911	1
147349	1
147592	1
147675	1
147695	13
147874	1
147940	1
148053	1
148084	1
148138	11
148159	1
148446	1
148455	21
148714	1

148774	6
148912	1
148919	1
148952	4
148990	1
149000	1
149088	1
149147	1
149199	1
149260	3
149268	2
149340	2
149415	3
149459	1
149515	1
149550	6
149607	1
149824	3
149885	2
149964	1
150025	3
150323	3
150531	1
150543	1
150722	1
150771	1
150865	3
151100	5
151480	1
151512	1
151562	1
151648	2
151832	1
151870	2
152406	3
152425	3
152477	1

152490	2
152553	1
152597	1
152685	1
152809	7
153107	4
153318	1
153370	1
153478	1
153554	2
153776	1
153877	2
153988	1
154048	2
154064	1
154143	1
154264	3
154325	1
154376	1
154617	1
154747	15
154925	5
154943	1
155465	1
155520	1
155650	1
155656	2
155843	3
156077	1
156522	1
156553	1
156678	3
156741	1
156742	5
156818	2
156984	2
157252	8

157316	1
157437	1
157455	1
157457	3
157484	1
157575	2
157704	1
158003	1
158565	61
158662	1
158806	1
158856	9
159516	1
159551	2
159591	2
159624	1
159650	1
159701	4
159832	1
159900	1
159914	2
159954	4
160061	1
160062	1
160116	2
160184	2
160339	4
160638	1
160713	7
160753	1
160761	6
160779	1
160840	1
160897	1
160918	1
161081	9
161135	1

161151	3
161383	2
161484	5
161581	1
161757	27
161759	1
161877	1
161882	2
162001	2
162344	1
162468	1
162482	7
162507	2
162524	1
162527	2
162659	2
162776	2
162826	1
162835	2
162960	1
162986	6
163067	1
163114	5
163230	1
163290	6
163503	3
163516	1
163572	54
163678	1
163908	1
163978	1
164053	1
164061	117
164072	1
164423	1
164455	3
164461	1

164546	2
164615	1
164775	1
164855	3
164986	1
164998	1
165004	2
165034	1
165094	1
165429	1
165508	2
165710	1
165748	1
165754	1
165776	2
165900	1
165923	1
165991	1
166032	1
166270	13
166327	7
166493	1
166514	8
166526	81
166615	4
166943	1
167044	2
167138	7
167493	1
167624	1
167828	1
167879	1
167982	1
168173	1
168306	10
168369	1
168388	1

168479	2
168492	1
168495	1
168734	1
169051	1
169126	3
169422	1
169444	2
169512	1
169621	1
171066	7
171223	2
171355	1
171488	1
171498	3
171583	1
171586	8
171828	1
171913	1
171914	4
172077	1
172098	7
172131	3
172270	1
172296	6
172323	1
172333	1
172392	12
172666	7
172766	1
172820	1
172842	1
172854	2
172899	1
173042	4
173082	435
173225	2

173339	3
173758	1
173999	1
174042	12
174198	6
174217	2
174245	10
174286	1
174293	1
174439	2
174574	2
174596	2
174653	4
174654	1
174733	1
174787	1
174809	1
174870	1
175363	1
175371	1
175409	1
175484	7
175557	4
175623	1
175656	1
175687	7
176045	2
176132	1
176140	1
176202	85
176418	1
176425	1
176493	1
176529	1
176776	1
176804	2
176908	1

177127	6
177186	1
177219	1
177355	1
177387	1
177677	1
177779	1
177787	1
177861	2
178054	3
178087	1
178089	2
178121	4
178248	1
178467	1
178503	1
178587	1
178613	10
178665	1
178784	1
178898	14
178899	1
179052	1
179053	2
179103	7
179463	1
179474	1
179506	1
179509	1
179608	1
180021	1
180540	23
180676	1
180717	3
180771	1
180867	1
180917	1

181114	2
181225	1
181233	1
181242	1
181389	2
181572	1
181784	1
181921	2
181953	1
182153	3
182175	1
182241	1
182258	3
182328	6
182363	1
182410	2
182431	1
182482	1
182823	1
183171	46
183270	3
183333	4
183445	1
183508	5
183628	1
183634	21
183735	1
183800	3
184005	2
184119	1
184126	1
184252	3
184264	1
184392	1
184879	1
185111	1
185126	1

185137	1
185235	2
185677	1
185695	3
185727	1
185816	1
185831	1
185965	1
185969	1
186092	5
186106	1
186174	11
186178	1
186229	1
186538	1
186619	2
186771	6
186834	3
186988	1
187064	5
187105	1
187201	2
187264	1
187312	1
187321	1
187376	1
187467	1
187602	1
187816	1
187819	1
187836	1
187862	1
187873	1
187993	1
188037	1
188088	1
188090	1

188124	4
188353	7
188525	2
188682	1
188779	1
188854	4
188928	11
188976	1
189023	1
189235	2
189296	3
189492	3
189508	1
189692	4
189721	1
189966	1
190085	2
190163	1
190386	3
190872	2
190992	23
191003	7
191037	9
191051	1
191241	1
191355	4
191429	1
191563	1
191568	1
191618	3
191643	6
191668	1
191758	1
191829	2
191912	1
191940	1
192050	3

192270	4
192370	6
192854	3
192949	2
193154	1
193242	1
193451	1
193524	1
193566	2
193650	1
193742	5
193952	2
194032	2
194144	2
194194	2
194312	1
194349	1
194404	1
194543	2
194589	2
194653	2
194689	1
194819	1
194957	1
195078	18
195079	1
195256	6
195381	2
195514	1
195656	1
195845	4
195908	6
195935	14
196032	1
196278	1
196304	1
196359	2

196362	1
196461	1
196582	1
196682	2
196725	2
196775	1
196936	1
196955	1
196983	1
197098	1
197211	8
197234	2
197255	3
197336	2
197385	1
197388	1
197394	1
197804	2
197829	5
197850	2
197876	2
198058	14
198145	1
198374	1
198413	1
198483	1
198613	1
198630	1
198896	10
199042	1
199063	11
199105	1
199125	17
199150	3
200366	1
200383	3
200421	1

200548	1
200664	8
200699	3
200837	2
201090	1
201176	1
201180	1
201187	2
201327	22
201369	3
201377	2
201647	1
201682	1
201712	1
202010	2
202028	2
202226	2
202274	3
202560	2
202788	8
202821	3
202926	2
202962	1
203025	12
203047	1
203074	1
203185	6
203199	2
203319	2
203358	2
203430	26
203560	8
203667	1
203672	1
203689	1
203749	1
203928	5

203934	1
203961	2
204041	2
204068	637
204209	1
204222	1
204319	12
204452	3
204636	1
204673	1
204825	11
204919	1
204986	2
205109	1
205113	2
205150	1
205282	1
205363	5
205398	1
205688	8
206184	1
206317	1
206350	5
206759	1
206785	1
206835	3
207039	2
207096	3
207118	8
207128	1
207371	3
207387	1
207448	3
207671	1
207725	3
207928	1
208154	2

208240	1
208343	10
208404	4
208417	1
208571	1
208596	1
208771	10
208860	1
208865	2
208948	2
208961	1
208995	6
209206	5
209241	3
209249	4
209428	1
209434	3
209493	1
209701	1
209713	1
209810	1
209827	1
209853	4
209878	1
209951	2
210113	1
210143	4
210172	3
210183	2
210184	1
210257	3
210398	1
210896	1
210972	2
211024	2
211052	1
211131	1

211167	1
211169	1
211232	5
211258	1
211287	1
211342	1
211395	1
211428	1
211432	4
211526	7
211613	1
211707	25
211744	1
211748	1
211755	5
211791	1
211823	1
211965	1
212031	1
212043	3
212058	3
212097	2
212274	1
212510	10
212532	1
212612	1
212845	2
212935	1
213142	1
213154	2
213258	2
213322	2
213344	5
213367	4
213376	1
213434	2
213470	3

213523	2
213584	1
213714	7
214009	1
214075	1
214127	3
214129	3
214239	1
214477	1
214519	1
214683	20
214831	1
214894	1
214956	1
215131	7
215135	4
215189	1
215218	3
215400	5
215402	6
215620	5
215670	1
215704	1
215720	1
215796	1
215817	2
215872	11
215907	9
216173	2
216288	2
216588	8
216681	2
216746	3
216773	1
216806	2
216823	1
216863	3

216894	1
216939	1
217025	4
217099	1
217197	3
217207	1
217210	4
217252	1
217263	1
217309	2
217321	1
217323	2
217348	1
217355	3
217365	1
217376	1
217385	7
217446	1
217516	3
217524	3
217591	1
217707	2
217939	1
218071	1
218088	2
218091	1
218111	4
218185	1
218196	4
218298	1
218326	3
218369	4
218373	1
218486	2
218535	1
218555	1
218559	2

218696	1
218801	1
218912	2
218916	1
219012	205
219082	1
219215	1
219322	50
219479	1
219503	5
219780	1
220141	1
220196	6
220223	1
220246	1
220307	1
220541	3
220590	6
220592	6
220630	1
220692	1
220693	1
221019	7
221043	1
221074	2
221341	1
221460	3
221478	60
221529	1
221604	2
221615	1
221761	1
221765	1
221830	1
221996	2
222162	1
222231	8

222356	6
222364	10
222416	1
222450	1
222513	2
222609	9
222788	1
222869	6
222926	3
222938	5
222939	1
223057	2
223061	1
223338	4
223662	2
223832	1
223841	3
223875	1
223918	1
224077	1
224091	1
224107	1
224112	11
224247	3
224271	31
224365	1
224433	19
224449	12
224527	1
224619	1
224638	1
224718	1
224726	2
224758	1
224766	1
224875	1
224891	1

225096	1
225121	2
225189	3
225256	6
225266	9
225485	1
225492	2
225579	13
225588	1
225642	1
225660	1
225743	2
225761	11
225762	1
225937	1
225961	43
226072	1
226084	5
226087	9
226449	1
226473	1
226675	2
226776	1
226783	1
227027	9
227116	2
227164	1
227196	1
227288	1
227348	1
227514	2
227632	2
227700	1
227828	1
227872	1
227940	1
227948	2

228051	1
228075	1
228091	3
228094	1
228138	1
228206	5
228239	1
228249	4
228345	2
228428	2
228453	5
228464	2
228479	1
228530	2
228546	6
228564	1
228569	2
228738	1
228744	1
228769	1
228809	14
228858	1
229008	1
229174	3
229255	2
229319	2
229328	1
229347	1
229360	3
229442	1
229444	1
229462	6
229563	1
229656	1
229658	1
229694	2
229720	1

229731	2
229767	1
229802	1
229816	2
229900	1
229979	9
230011	12
230018	1
230027	1
230042	3
230053	1
230074	1
230133	1
230147	1
230152	1
230219	1
230233	1
230331	9
231452	1
231479	2
231723	14
231761	6
231945	1
231948	3
231952	1
231959	3
232008	1
232034	1
232059	1
232257	1
232278	4
232363	7
232443	1
232444	16
232625	1
232657	1
232706	13

232710	3
232728	1
232738	1
232775	5
232814	1
232846	1
232855	1
232930	1
232951	1
233014	1
233027	1
233073	1
233076	2
233099	1
233191	1
233198	1
233244	2
233268	3
233294	1
233305	1
233325	2
233328	1
233347	8
233405	7
233429	3
233439	3
233515	3
233576	1
233584	2
233615	1
233618	1
233623	3
233687	2
233698	4
233712	1
233731	3
233807	2

233818	1
233819	1
233832	4
233834	1
233974	1
234050	1
234066	1
234077	1
234110	2
234116	4
234254	1
234299	1
234316	1
234340	1
234367	1
234373	1
234397	1
234410	6
234439	2
234452	1
234457	1
234464	3
234466	1
234488	35
234532	1
234558	1
234573	9
234696	1
234732	6
234749	4
234750	20
234752	1
234781	3
234794	1
234866	2
234869	1
234937	1

234957	2
235008	1
235030	1
235033	12
235035	1
235135	1
235142	1
235187	1
235199	19
235235	2
235238	1
235300	1
235322	2
235359	3
235391	1
235395	2
235406	1
235436	4
235471	1
235485	3
235497	1
235512	1
235525	1
235578	5
235620	8
235632	1
235658	1
235689	3
235734	1
235785	2
235804	1
235832	1
235845	2
235847	1
235871	1
235898	2
235927	1

235958	3
235989	1
236010	1
236034	2
236052	1
236120	3
236142	1
236170	2
236174	9
236231	1
236314	5
236337	2
236411	1
236421	4
236443	4
236452	1
236599	1
236659	1
236662	3
236755	1
236779	1
236829	1
236832	1
236934	4
236953	1
237022	4
237026	1
237058	1
237063	1
237087	1
237197	1
237256	1
237301	1
237302	1
237303	1
237321	1
237333	1

237340	1
237349	5
237379	1
237396	1
237398	1
237401	8
237418	2
237422	7
237450	1
237462	1
237463	5
237477	11
237529	1
237575	2
237586	1
237603	1
237651	1
237706	4
237732	5
237763	3
237796	15
237839	1
237912	2
237923	3
237957	3
237974	9
238031	3
238066	1
238124	1
238129	2
238201	1
238219	1
238245	1
238292	1
238310	2
238312	1
238325	2

238365	1
238395	1
238403	1
238405	1
238439	4
238448	1
238498	1
238499	160
238515	1
238522	1
238563	4
238573	1
238575	2
238595	5
238662	1
238707	1
238768	1
238813	1
238826	2
238878	1
238905	2
238934	1
238980	13
239008	2
239019	2
239028	1
239039	2
239065	1
239068	1
239098	1
239144	2
239152	2
239182	1
239183	67
239184	1
239193	1
239198	3

239222	9
239256	1
239303	1
239330	1
239343	1
239353	2
239363	1
239371	1
239395	5
239403	2
239410	5
239414	3
239430	2
239440	3
239453	1
239473	3
239481	31
239501	1
239545	1
239585	5
239587	1
239621	1
239664	26
239673	1
239684	1
239691	1
239697	1
239744	1
239841	1
239844	9
239862	1
239865	1
239877	1
239936	1
239937	1
239940	1
239945	1

239947	1
239955	1
239983	1
239999	6
240028	1
240039	1
240049	8
240054	1
240082	1
240131	1
240171	1
240184	1
240202	1
240231	1
240244	1
240259	2
240261	1
240273	1
240288	1
240306	1
240307	5
240325	1
240344	1
240383	1
240385	1
240386	1
240396	1
240432	1
240450	2
240520	1
240550	3
240576	1
240581	2
240588	1
240615	3
240620	4
240624	1

240648	1
240655	1
240672	1
240680	1
240699	9
240706	8
240708	1
240799	2
240802	2
240877	1
240887	2
240893	2
240935	34
240936	2
240962	1
241088	2
241093	45
241122	1
241135	4
241137	22
241170	7
241179	1
241258	1
241290	1
241297	1
241304	1
241404	1
241431	1
241432	1
241460	2
241545	1
241562	1
241567	1
241581	1
241606	4
241617	1
241629	2

241671	1
241690	1
241731	1
241755	4
241763	1
241805	1
241806	1
241873	1
241914	1
241943	1
241975	1
241999	4
242096	1
242168	1
242170	2
242191	3
242257	1
242260	2
242268	2
242280	2
242299	2
242315	1
242319	9
242320	1
242321	2
242348	1
242356	1
242386	2
242402	1
242512	4
242515	14
242527	2
242572	1
242585	1
242646	1
242731	5
242732	1

242770	2
242777	1
242819	1
242831	1
242832	1
242845	1
242862	2
242885	3
242904	1
242905	1
242920	1
242991	1
243014	3
243020	1
243022	1
243038	1
243043	1
243052	1
243090	1
243227	2
243286	2
243318	1
243445	1
243451	1
243473	1
243520	3
243543	1
243589	2
243602	1
243655	2
243685	1
243688	1
243714	2
243716	1
243729	1
243759	1
243766	2

243772	1
243794	1
243811	2
243859	1
243891	1
244024	1
244055	1
244057	2
244064	1
244197	1
244272	1
244279	1
244367	18
244373	1
244458	1
244464	1
244470	4
244475	1
244515	2
244525	1
244526	2
244533	1
244568	3
244600	2
244653	1
244704	1
244721	20
244724	1
244728	3
244745	2
244747	1
244832	1
244854	3
244873	4
244880	2
244888	3
244916	1

244925	1
244949	2
244973	1
244999	2
245002	3
245006	4
245027	3
245036	1
245070	1
245080	1
245149	1
245304	1
245330	1
245332	1
245338	1
245351	2
245361	1
245397	1
245458	2
245521	1
245557	1
245590	2
245592	2
245594	1
245608	1
245612	1
245619	2
245707	2
245746	2
245769	24
245811	1
245849	3
245935	3
245949	1
245961	1
245989	1
246028	1

246047	1
246062	7
246073	1
246130	1
246320	1
246349	1
246360	1
246432	4
246436	1
246472	1
246507	1
246541	1
246561	2
246576	3
246589	8
246597	2
246647	1
246661	1
246694	1
246701	1
246734	2
246788	1
246808	1
246835	14
246844	1
246858	1
246865	1
246929	4
247004	1
247065	1
247067	1
247115	1
247122	3
247132	1
247159	3
247185	3
247196	1

247242	2
247246	1
247300	1
247359	1
247363	1
247479	15
247491	1
247497	1
247507	1
247542	1
247561	1
247605	1
247748	1
247779	1
247804	1
247844	2
247850	1
247876	2
247879	2
247880	1
247884	2
247889	2
247930	7
247939	1
247958	3
247972	10
247974	1
248045	1
248056	2
248060	1
248083	1
248173	1
248258	4
248285	21
248324	1
248347	2
248359	1

248362	1
248428	1
248456	1
248468	1
248508	1
248532	1
248533	2
248535	1
248546	1
248568	1
248583	1
248584	3
248590	5
248647	3
248665	1
248667	1
248701	1
248726	1
248735	4
248780	1
248799	4
248811	1
248819	1
248839	1
248847	1
248852	1
248902	1
248915	1
249001	1
249005	2
249033	1
249096	2
249113	1
249148	1
249205	22
249229	1
249243	1

249244	2
249269	1
249270	1
249279	3
249286	1
249317	2
249345	2
249469	2
249471	1
249482	7
249490	7
249553	1
249554	1
249596	1
249609	1
249611	9
249612	1
249613	1
249628	1
249691	1
249710	1
249727	1
249729	1
249746	2
249756	1
249758	1
249867	1
249877	1
249883	1
249892	1
249966	1
249968	2
249976	1
250008	1
250040	1
250056	4
250066	1

250080	1
250082	1
250091	1
250097	1
250103	3
250108	8
250143	1
250158	1
250176	3
250216	1
250277	2
250298	6
250325	4
250342	1
250350	1
250372	13
250405	1
250408	1
250476	1
250500	1
250598	10
250623	18
250668	2
250674	4
250697	1
250712	2
250739	1
250744	1
250749	1
250820	6
250848	2
250857	1
250872	1
250899	15
250914	2
250918	1
250922	2

250942	1
250967	1
250969	1
251005	1
251014	1
251040	5
251043	1
251064	1
251081	1
251094	1
251143	1
251144	1
251177	1
251233	15
251286	2
251318	1
251368	1
251447	5
251453	3
251466	2
251472	1
251511	3
251523	1
251535	1
251547	1
251566	3
251584	1
251610	5
251645	1
251658	1
251685	2
251692	4
251725	1
251744	2
251750	3
251838	19
251858	2

251875	6
251893	1
251894	1
251901	2
251911	1
251937	1
251941	3
251943	1
252010	1
252013	1
252027	1
252032	1
252051	25
252071	1
252081	1
252092	1
252142	2
252158	2
252165	1
252202	5
252204	1
252244	2
252306	1
252382	1
252474	8
252480	1
252482	1
252495	4
252518	1
252563	1
252594	1
252604	1
252634	1
252635	2
252638	6
252669	1
252695	1

252753	1
252754	2
252759	1
252767	1
252773	1
252823	1
252871	3
252904	1
252905	1
253052	2
253095	1
253099	1
253113	1
253117	3
253215	2
253217	1
253250	88
253313	1
253319	1
253397	1
253401	1
253439	1
253447	1
253450	3
253453	1
253455	1
253468	1
253511	1
253650	1
253710	1
253728	13
253839	1
253909	1
253929	1
253951	1
254091	1
254142	1

254146	1
254193	2
254291	14
254295	1
254312	2
254341	1
254345	2
254418	1
254488	2
254521	2
254559	1
254595	2
254603	1
254607	1
254615	1
254618	1
254672	1
254687	1
254741	1
254786	1
254797	1
254848	1
254849	2
254878	1
254992	1
255005	1
255026	10
255090	2
255178	16
255184	1
255195	1
255198	1
255219	1
255272	6
255292	1
255300	5
255339	1

255362	1
255365	1
255379	1
255382	3
255387	1
255408	4
255414	1
255439	2
255489	4
255500	1
255508	1
255555	1
255592	2
255601	1
255642	1
255671	2
255688	1
255740	1
255779	1
255791	1
255833	2
255840	1
255855	1
255886	1
255908	1
255925	1
255932	2
256086	2
256124	2
256202	1
256205	1
256224	1
256266	1
256270	2
256282	1
256316	1
256329	1

256408	1
256432	1
256460	1
256467	1
256515	1
256518	1
256586	1
256609	1
256626	1
256716	2
256798	4
256925	1
256929	1
256955	5
256958	1
256959	1
256962	1
256994	1
257001	1
257006	1
257070	1
257079	1
257114	2
257335	1
257384	2
257449	1
257465	2
257504	1
257676	2
257714	3
257726	1
257728	1
257767	1
257774	5
257827	2
257890	1
257897	1

257912	2
257931	1
258009	1
258059	1
258074	1
258296	1
258315	1
258339	1
258346	1
258450	1
258484	1
258565	2
258581	1
259581	1
259630	1
259657	1
259673	1
259679	1
259700	2
259747	1
259806	1
259919	3
259947	1
260020	14
260029	1
260031	1
260043	1
260112	1
260167	10
260169	1
260178	1
260249	1
260303	2
260320	1
260434	1
260475	3
260533	1

260548	1
260637	1
260660	2
260705	1
260710	1
260763	2
260776	2
260789	2
260797	3
260812	2
260856	4
260888	2
260905	6
260997	1
261017	1
261026	1
261169	1
261177	1
261193	1
261288	1
261309	1
261340	1
261347	1
261365	2
261381	6
261394	1
261410	1
261446	1
261472	1
261529	1
261590	1
261591	1
261599	1
261611	1
261707	1
261723	3
261888	1

261900	1
261955	3
261996	1
262058	1
262077	1
262146	1
262168	3
262182	1
262229	1
262365	1
262408	1
262444	1
262546	1
262563	2
262594	1
262624	3
262637	2
262651	1
262660	2
262728	1
262748	4
262781	1
262786	1
262811	1
262847	2
262851	1
262906	1
262913	6
262948	18
262965	2
263055	2
263059	1
263075	1
263188	2
263189	2
263196	1
263208	1

263317	4
263327	1
263415	1
263508	2
263607	1
263612	1
263640	1
263644	1
263718	1
263743	2
263813	1
263816	1
263862	1
263905	2
264000	2
264024	2
264076	1
264098	1
264103	1
264105	1
264121	1
264222	2
264266	1
264286	1
264343	1
264392	2
264405	1
264451	3
264464	3
264569	1
264581	4
264652	1
264726	1
264742	2
264775	1
264860	1
264893	2

264900	1
264926	1
264944	4
264985	1
265039	2
265055	1
265058	2
265069	3
265096	1
265147	2
265148	5
265150	6
265175	1
265177	1
265196	1
265272	1
265293	1
265454	1
265479	1
265510	2
265548	1
265590	3
265598	1
265658	2
265690	1
265697	1
265718	1
265760	1
265881	2
265916	1
265996	2
265997	1
266046	1
266071	7
266072	1
266089	1
266096	1

266110	1
266113	1
266129	1
266147	2
266148	2
266161	19
266182	2
266197	1
266203	1
266212	2
266257	1
266258	1
266317	1
266434	1
266455	1
266473	3
266480	1
266492	2
266507	1
266517	1
266545	2
266565	1
266574	1
266702	1
266722	1
266833	1
266859	1
266878	1
266946	1
267024	1
267053	1
267133	1
267172	1
267205	1
267232	1
267322	1

In [114...
```
# Confirm using Pandas
(mergeT[mergeT['AboutMe'].notnull()]).groupby('OwnerUserId')['PostTypeId'].count()
```

Out[114...
```
OwnerUserId
-1         1
 22        2
 25        4
 29        1
 104       1
          ..
 267133    1
 267172    1
 267205    1
 267232    1
 267322    1
Name: PostTypeId, Length: 3366, dtype: int64
```

In [115...
```
# insert the queries into the queries table
sql ="""
   SELECT OwnerUserId,count(*)
    FROM (SELECT * FROM users
    INNER JOIN posts ON posts.OwnerUserId = users.Id
    WHERE AboutMe <> "NULL")
    GROUP BY OwnerUserId;
    """

store_query('task 3','Considering only the users with an "AboutMe," how many posts a
```

Not taking into account the commentcount field in the table posts, what are the Top 10 posts in terms of number of comments?

In [116...
```
%%sql
   SELECT PostId,count(comments.Id) as NumberComments FROM posts
    LEFT OUTER JOIN comments ON posts.Id = comments.PostId
    GROUP BY PostId
    ORDER BY NumberComments DESC
    LIMIT 10;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[116...

PostId	NumberComments
386853	66
386556	34
418910	31
395232	31
402987	27
386075	26
394118	24
402950	23
398828	23
396111	22

In [117…

```
# Confirm using Pandas
mergeT= posts.merge(comments,how='left',left_on='Id',right_on='PostId')
top=mergeT.groupby('PostId').count()['Text'].sort_values(ascending=False)
top.iloc[0:10]
```

Out[117…

```
PostId
386853.0    66
386556.0    34
395232.0    31
418910.0    31
402987.0    27
386075.0    26
394118.0    24
402950.0    23
398828.0    23
396111.0    22
Name: Text, dtype: int64
```

In [118…

```
# insert the queries into the queries table
sql ="""
    SELECT PostId,count(comments.Id) as NumberComments FROM posts
    LEFT OUTER JOIN comments ON posts.Id = comments.PostId
    GROUP BY PostId
    ORDER BY NumberComments DESC
    LIMIT 10;
    """
store_query('task 3','Not taking into account the commentcount field in the table po
```

What are the Top 10 posts which have the highest cummulative (post score + comment score) score?

In [119…

```
%%sql
  SELECT PostId,(TotalComScore + TotalPostScore ) as CummScore FROM
    (SELECT PostId,sum(comments.score) as TotalComScore ,sum(posts.score) as TotalPo
    LEFT OUTER JOIN comments ON posts.Id = comments.PostId
    GROUP BY PostId
    ORDER BY TotalComScore DESC
    LIMIT 10)
    ORDER BY CummScore DESC;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[119…

PostId	CummScore
394118	1778
394128	1569
398653	1021
388578	941
388566	885
398646	465
421677	392
420526	317
386571	197

```
393336          125
```

In [120…

```
# Confirm using Pandas
mergeT= posts.merge(comments,how='left',left_on='Id',right_on='PostId')
top=mergeT.groupby('PostId').sum()[['Score_x','Score_y']]
top['CummScore']=top['Score_x']+top['Score_y']
top10=top['CummScore'].sort_values(ascending=False)
top10.iloc[:10]
```

Out[120…

```
PostId
394118.0    1778.0
394128.0    1569.0
418910.0    1094.0
398653.0    1021.0
388578.0     941.0
388566.0     885.0
396818.0     808.0
418814.0     621.0
394258.0     570.0
394439.0     566.0
Name: CummScore, dtype: float64
```

In [121…

```
# insert the queries into the queries table
sql ="""
    SELECT PostId,(TotalComScore + TotalPostScore ) as CummScore FROM
    (SELECT PostId,sum(comments.score) as TotalComScore ,sum(posts.score) as TotalPo
    LEFT OUTER JOIN comments ON posts.Id = comments.PostId
    GROUP BY PostId
    ORDER BY TotalComScore DESC
    LIMIT 10)
    ORDER BY CummScore DESC;
    """
store_query('task 3','Not taking into account the commentcount field in the table po
```

Who are the top 10 users who comment the most?

In [122…

```
%%sql
SELECT UserId, users.Reputation,count(PostId) as PostsNum, count(Text) as NumComment
    LEFT OUTER JOIN comments ON users.Id = comments.UserId
    GROUP BY UserId
    ORDER BY NumComments DESC,PostsNum DESC
    LIMIT 10;
```

```
 * sqlite:///chatdata.db
Done.
```

Out[122…

UserId	Reputation	PostsNum	NumComments
919	223056	3301	3301
805	228662	1153	1153
143489	2890	1024	1024
11887	39200	805	805
85665	17391	691	691
164061	13485	540	540
22047	41385	536	536
158565	6482	504	504

7962	8030	492	492
35989	71548	470	470

In [123…

```
# Confirm using Pandas
mergeT= users.merge(comments,how='left',left_on='Id',right_on='UserId')
top=mergeT.groupby('UserId').count()['PostId'].sort_values(ascending=False)
top.iloc[0:10]
```

Out[123…

```
UserId
919.0       3301
805.0       1153
143489.0    1024
11887.0      805
85665.0      691
164061.0     540
22047.0      536
158565.0     504
7962.0       492
35989.0      470
Name: PostId, dtype: int64
```

In [124…

```
# insert the queries into the queries table
sql ="""
    SELECT UserId, users.Reputation,count(PostId) as PostsNum, count(Text) as NumCom
    LEFT OUTER JOIN comments ON users.Id = comments.UserId
    GROUP BY UserId
    ORDER BY PostsNum DESC
    LIMIT 10;
    """
store_query('task 3','Who are the top 10 users who comment the most?',sql)
```

Who are the top 10 users who post the most?

In [125…

```
%%sql
SELECT UserId, users.Reputation,count(PostId) as PostsNum FROM users
    LEFT OUTER JOIN comments ON users.Id = comments.UserId
    GROUP BY UserId
    ORDER BY  PostsNum DESC ,users.Reputation DESC
    LIMIT 10
```

```
 * sqlite:///chatdata.db
Done.
```

Out[125…

UserId	Reputation	PostsNum
919	223056	3301
805	228662	1153
143489	2890	1024
11887	39200	805
85665	17391	691
164061	13485	540
22047	41385	536
158565	6482	504
7962	8030	492

35989 71548 470

Task 4: Check the Queries Table

Now let's tidy up and check the queries table.

First let's check it's contents:

In [126…
```
%sql SELECT * FROM queries
```

```
 * sqlite:///chatdata.db
Done.
```

Out[126…

task	action	query
Single Table Queries	Which 5 users have viewed the most times and what is the sum of those views per user?	SELECT Id, SUM(Views) AS TotalViews FROM Users GROUP BY Id ORDER BY TotalViews DESC LIMIT 5
Task 1	Create table comments	CREATE TABLE "comments" ("Id" INTEGER, "PostId" INTEGER, "Score" INTEGER, "Text" TEXT, "CreationDate" TEXT, "UserId" INTEGER)
Task 1	Create table posts	CREATE TABLE "posts" ("Id" INTEGER NOT NULL PRIMARY KEY, "PostTypeId" INTEGER, "AcceptedAnswerId" INTEGER, "ParentId" INTEGER, "CreationDate" TEXT, "Score" INTEGER, "ViewCount" INTEGER, "Body" TEXT, "OwnerUserId" INTEGER, "OwnerDisplayName" TEXT, "LastEditorUserId" INTEGER, "LastEditorDisplayName" TEXT, "LastEditDate" TEXT, "LastActivityDate" TEXT, "Title" TEXT, "Tags" TEXT, "AnswerCount" INTEGER, "CommentCount" INTEGER, "FavoriteCount" INTEGER, "ClosedDate" TEXT, "CommunityOwnedDate" TEXT, FOREIGN KEY(OwnerUserId) REFERENCES users(Id))
Task 1	Create table users	CREATE TABLE "users" ("Id" INTEGER NOT NULL PRIMARY KEY, "Reputation" INTEGER,

```
"CreationDate" TEXT,
"DisplayName" TEXT,
"LastAccessDate" TEXT,
"WebsiteUrl" TEXT,
"Location" TEXT,
"AboutMe" TEXT,
"Views" INTEGER,
"UpVotes" INTEGER,
"DownVotes" INTEGER,
"ProfileImageUrl" TEXT,
"AccountId" INTEGER
)
```

Task 1	Count the number of rows in the comments table	SELECT count(*) FROM "comments"
Task 1	Count the number of rows in the users table	SELECT count(*) FROM "users"
Task 1	Count the number of rows in the posts table	SELECT count(*) FROM "posts"
task 1	Run the query to select 5 random rows from the comments table	SELECT * FROM comments ORDER BY RANDOM() LIMIT 5;
task 1	Run the query to select 5 random rows from the posts table	SELECT * FROM posts ORDER BY RANDOM() LIMIT 5;
task 1	Run the query to select 5 random rows from the users table	SELECT * FROM users ORDER BY RANDOM() LIMIT 5;
task 2	How many posts have 0 comments	SELECT count(*) as Count FROM posts WHERE CommentCount = 0;
task 2	How many posts have 1 comments	SELECT count(*) as Count FROM posts WHERE CommentCount = 1;
task 2	How many posts have 2 comments or more	SELECT count(*) as Count FROM posts WHERE CommentCount >= 2;
task 2	Find the 5 posts with the highest viewcount	SELECT * FROM posts ORDER BY ViewCount DESC LIMIT 5;
task 2	Find the 5 posts with the highest scores	SELECT * FROM posts ORDER BY Score DESC LIMIT 5;
task 2	What are the 5 most frequent scores on posts	SELECT Score ,count(*) as count FROM posts GROUP BY Score ORDER BY count DESC LIMIT 5;

task 2	How many posts have the keyword "data" in their tags	SELECT count(*) as count FROM posts WHERE Tags LIKE "%data%";
task 2	What are the 5 most frequent commentcount for posts	SELECT CommentCount ,count(*) as count FROM posts GROUP BY CommentCount ORDER BY count DESC LIMIT 5;
task 2	How many posts have an accepted answer	SELECT count(*) as Count FROM posts WHERE AcceptedAnswerId <> 0;
task 2	What is the average reputation of table users	SELECT ROUND(AVG(Reputation),2) as AverageReputation FROM users;
task 2	What are the min and max reputation of users	SELECT ROUND(MIN(Reputation),2) as MinReputation, ROUND(MAX(Reputation),2) as MaxReputation FROM users;
task 2	What is the length of the body of 5 most viewed posts	SELECT ViewCount as MostViewed ,Len_Body as LengthBody FROM posts ORDER BY ViewCount DESC LIMIT 5;
task 2	How many different locations are there in the users table	SELECT count(DISTINCT(location)) as LocationCount FROM (SELECT * FROM users GROUP BY location);
task 2	What are the top 5 locations of users	SELECT Location,count(*) as LocationCount FROM users GROUP BY Location ORDER BY LocationCount DESC LIMIT 5;
task 2	Rank the days of the week from highest to lowest in terms of the volume of ViewCount as a percentage	SELECT CASE WHEN WeekDayNumber = 0 THEN 'Sunday' WHEN WeekDayNumber = 1 THEN 'Monday' WHEN WeekDayNumber = 2 THEN 'Tuesday' WHEN WeekDayNumber = 3 THEN 'Wednesday' WHEN WeekDayNumber = 4 THEN 'Thursday' WHEN WeekDayNumber = 5 THEN 'Friday' ELSE 'Saturday' END as WeekDay,Percentage_ViewCount FROM (SELECT CAST (strftime('%w', CreationDate) AS INTEGER) as WeekDayNumber, round(cast(sum(ViewCount) as float)/cast((select sum(ViewCount) from posts)as float) *100.0,2) as Percentage_ViewCount FROM posts GROUP BY WeekDayNumber ORDER BY Percentage_ViewCount DESC);
task 3	How many posts have been created by a user that has a filled out the "AboutMe" section	SELECT count(*) FROM users

		INNER JOIN posts ON posts.OwnerUserId = users.Id WHERE AboutMe <> "NULL";
task 3	Considering only the users with an "AboutMe," how many posts are there per user	SELECT OwnerUserId,count(*) FROM (SELECT * FROM users INNER JOIN posts ON posts.OwnerUserId = users.Id WHERE AboutMe <> "NULL") GROUP BY OwnerUserId;
task 3	Not taking into account the commentcount field in the table posts, what are the Top 10 posts in terms of number of comments?	SELECT PostId,count(comments.Id) as NumberComments FROM posts LEFT OUTER JOIN comments ON posts.Id = comments.PostId GROUP BY PostId ORDER BY NumberComments DESC LIMIT 10;
task 3	Not taking into account the commentcount field in the table posts, what are the Top 10 posts in terms of number of comments?	SELECT PostId,(TotalComScore + TotalPostScore) as CummScore FROM (SELECT PostId,sum(comments.score) as TotalComScore ,sum(posts.score) as TotalPostScore FROM posts LEFT OUTER JOIN comments ON posts.Id = comments.PostId GROUP BY PostId ORDER BY TotalComScore DESC LIMIT 10) ORDER BY CummScore DESC;
task 3	Who are the top 10 users who comment the most?	SELECT UserId, users.Reputation,count(PostId) as PostsNum, count(Text) as NumComments FROM users LEFT OUTER JOIN comments ON users.Id = comments.UserId GROUP BY UserId ORDER BY PostsNum DESC LIMIT 10;

Drop Duplicates

You likely have some duplicates. Lets drop them.

In [127…

```
# Read the queries table into pandas
sql = 'SELECT * FROM queries'
queries = pd.read_sql(sql, con)

# Drop duplicates
queries.drop_duplicates(inplace = True) # drop duplicates
```

Case Issues

Remember, SQL is case insensitive. Pandas IS case sensitive. Lets deal with this now by making all of the text uppercase.

In [128…

```
for col in queries.columns:
    queries[col] = queries[col].str.upper()
```

```
queries
```

Out[128...

	task	action	query
0	SINGLE TABLE QUERIES	WHICH 5 USERS HAVE VIEWED THE MOST TIMES AND W...	\nSELECT ID, SUM(VIEWS) AS TOTALVIEWS\n FRO...
1	TASK 1	CREATE TABLE COMMENTS	\n CREATE TABLE "COMMENTS" (\n "ID" INTE...
2	TASK 1	CREATE TABLE POSTS	\n CREATE TABLE "POSTS" (\n "ID" INT...
3	TASK 1	CREATE TABLE USERS	\n CREATE TABLE "USERS" (\n "ID" INT...
4	TASK 1	COUNT THE NUMBER OF ROWS IN THE COMMENTS TABLE	\n SELECT COUNT(*) \n FROM "COMMENTS" \n...
5	TASK 1	COUNT THE NUMBER OF ROWS IN THE USERS TABLE	\n SELECT COUNT(*) \n FROM "USERS" \n
6	TASK 1	COUNT THE NUMBER OF ROWS IN THE POSTS TABLE	\n SELECT COUNT(*) \n FROM "POSTS" \n
7	TASK 1	RUN THE QUERY TO SELECT 5 RANDOM ROWS FROM THE...	\n SELECT * FROM COMMENTS \n ORDER BY RA...
8	TASK 1	RUN THE QUERY TO SELECT 5 RANDOM ROWS FROM THE...	\n SELECT * FROM POSTS \n ORDER BY RANDO...
9	TASK 1	RUN THE QUERY TO SELECT 5 RANDOM ROWS FROM THE...	\n SELECT * FROM USERS \n ORDER BY RANDO...
10	TASK 2	HOW MANY POSTS HAVE 0 COMMENTS	\n SELECT COUNT(*) AS COUNT FROM POSTS \n ...
11	TASK 2	HOW MANY POSTS HAVE 1 COMMENTS	\n SELECT COUNT(*) AS COUNT FROM POSTS \n ...
12	TASK 2	HOW MANY POSTS HAVE 2 COMMENTS OR MORE	\n SELECT COUNT(*) AS COUNT FROM POSTS \n ...
13	TASK 2	FIND THE 5 POSTS WITH THE HIGHEST VIEWCOUNT	\n SELECT * FROM POSTS \n ORDER BY VIEWC...
14	TASK 2	FIND THE 5 POSTS WITH THE HIGHEST SCORES	\n SELECT * FROM POSTS \n ORDER BY SCORE...
15	TASK 2	WHAT ARE THE 5 MOST FREQUENT SCORES ON POSTS	\n SELECT SCORE ,COUNT(*) AS COUNT FROM POS...
16	TASK 2	HOW MANY POSTS HAVE THE KEYWORD "DATA" IN THEI...	\n SELECT COUNT(*) AS COUNT FROM POSTS \n ...
17	TASK 2	WHAT ARE THE 5 MOST FREQUENT COMMENTCOUNT FOR ...	\n SELECT COMMENTCOUNT ,COUNT(*) AS COUNT F...
18	TASK 2	HOW MANY POSTS HAVE AN ACCEPTED ANSWER	\n SELECT COUNT(*) AS COUNT FROM POSTS \n ...
19	TASK 2	WHAT IS THE AVERAGE REPUTATION OF TABLE USERS	\n SELECT ROUND(AVG(REPUTATION),2) AS AVERA...
20	TASK 2	WHAT ARE THE MIN AND MAX REPUTATION OF USERS	\n SELECT ROUND(MIN(REPUTATION),2) AS MINRE...
21	TASK 2	WHAT IS THE LENGTH OF THE BODY OF 5 MOST VIEWE...	\n SELECT VIEWCOUNT AS MOSTVIEWED ,LEN_BODY...
22	TASK 2	HOW MANY DIFFERENT LOCATIONS	\n SELECT COUNT(DISTINCT(LOCATION))

	task	action	query
		ARE THERE IN THE ...	AS LOCA...
23	TASK 2	WHAT ARE THE TOP 5 LOCATIONS OF USERS	\n SELECT LOCATION,COUNT(*) AS LOCATIONCOUN...
24	TASK 2	RANK THE DAYS OF THE WEEK FROM HIGHEST TO LOWE...	\nSELECT CASE \n WHEN WEEKDAYNUMBE...
25	TASK 3	HOW MANY POSTS HAVE BEEN CREATED BY A USER THA...	\n SELECT COUNT(*) FROM USERS\n INNER JO...
26	TASK 3	CONSIDERING ONLY THE USERS WITH AN "ABOUTME," ...	\n SELECT OWNERUSERID,COUNT(*) \n FROM (S...
27	TASK 3	NOT TAKING INTO ACCOUNT THE COMMENTCOUNT FIELD...	\n SELECT POSTID,COUNT(COMMENTS.ID) AS NUMB...
28	TASK 3	NOT TAKING INTO ACCOUNT THE COMMENTCOUNT FIELD...	\n SELECT POSTID,(TOTALCOMSCORE + TOTALPOST...
29	TASK 3	WHO ARE THE TOP 10 USERS WHO COMMENT THE MOST?	\n SELECT USERID, USERS.REPUTATION,COUNT(PO...

In [129...

```
# Write the now deduped uppercase dataframe back to sqlite and replace the table
queries.to_sql('queries', con, if_exists='replace', index=False)
```

Use Case

Now that we have this queries table, lets give you some ideas about how you would use it.

Suppose you wanted to find all of the queries where you did a GROUP BY:

In [130...

```
%%sql
SELECT query
    FROM queries
        WHERE query LIKE '%GROUP BY%'
```

```
 * sqlite:///chatdata.db
Done.
```

Out[130...

query
SELECT ID, SUM(VIEWS) AS TOTALVIEWS FROM USERS GROUP BY ID ORDER BY TOTALVIEWS DESC LIMIT 5
SELECT SCORE ,COUNT(*) AS COUNT FROM POSTS GROUP BY SCORE ORDER BY COUNT DESC LIMIT 5;
SELECT COMMENTCOUNT ,COUNT(*) AS COUNT FROM POSTS GROUP BY COMMENTCOUNT ORDER BY COUNT DESC LIMIT 5;

```
SELECT COUNT(DISTINCT(LOCATION)) AS LOCATIONCOUNT FROM (
SELECT * FROM USERS
GROUP BY LOCATION
);
```

```
SELECT LOCATION,COUNT(*) AS LOCATIONCOUNT FROM USERS
GROUP BY LOCATION
ORDER BY LOCATIONCOUNT DESC
LIMIT 5;
```

```
SELECT CASE
WHEN WEEKDAYNUMBER = 0 THEN 'SUNDAY'
WHEN WEEKDAYNUMBER = 1 THEN 'MONDAY'
WHEN WEEKDAYNUMBER = 2 THEN 'TUESDAY'
WHEN WEEKDAYNUMBER = 3 THEN 'WEDNESDAY'
WHEN WEEKDAYNUMBER = 4 THEN 'THURSDAY'
WHEN WEEKDAYNUMBER = 5 THEN 'FRIDAY'
ELSE 'SATURDAY'
END AS WEEKDAY,PERCENTAGE_VIEWCOUNT
FROM (SELECT CAST (STRFTIME('%W', CREATIONDATE) AS INTEGER) AS WEEKDAYNUMBER,
ROUND( CAST(SUM(VIEWCOUNT) AS FLOAT)/CAST((SELECT SUM(VIEWCOUNT) FROM POSTS)AS FLOAT)
*100.0,2) AS PERCENTAGE_VIEWCOUNT
FROM POSTS
GROUP BY WEEKDAYNUMBER
ORDER BY PERCENTAGE_VIEWCOUNT DESC);
```

```
SELECT OWNERUSERID,COUNT(*)
FROM (SELECT * FROM USERS
INNER JOIN POSTS ON POSTS.OWNERUSERID = USERS.ID
WHERE ABOUTME <> "NULL")
GROUP BY OWNERUSERID;
```

```
SELECT POSTID,COUNT(COMMENTS.ID) AS NUMBERCOMMENTS FROM POSTS
LEFT OUTER JOIN COMMENTS ON POSTS.ID = COMMENTS.POSTID
GROUP BY POSTID
ORDER BY NUMBERCOMMENTS DESC
LIMIT 10;
```

```
SELECT POSTID,(TOTALCOMSCORE + TOTALPOSTSCORE ) AS CUMMSCORE FROM
(SELECT POSTID,SUM(COMMENTS.SCORE) AS TOTALCOMSCORE ,SUM(POSTS.SCORE) AS TOTALPOSTSCORE
FROM POSTS
LEFT OUTER JOIN COMMENTS ON POSTS.ID = COMMENTS.POSTID
GROUP BY POSTID
ORDER BY TOTALCOMSCORE DESC
LIMIT 10)
ORDER BY CUMMSCORE DESC;
```

```
SELECT USERID, USERS.REPUTATION,COUNT(POSTID) AS POSTSNUM, COUNT(TEXT) AS NUMCOMMENTS
FROM USERS
LEFT OUTER JOIN COMMENTS ON USERS.ID = COMMENTS.USERID
GROUP BY USERID
ORDER BY POSTSNUM DESC
LIMIT 10;
```

Now Your Turn

Find the queries that have 'DISTINCT' in them. You can do it with the %sql command or with Pandas and sql.

TODO: Complete the following code cell

In [131…

```
%%sql
SELECT query
    FROM queries
        WHERE query LIKE '%DISTINCT%'
```

```
 * sqlite:///chatdata.db
Done.
```

Out[131…

query
SELECT COUNT(DISTINCT(LOCATION)) AS LOCATIONCOUNT FROM (SELECT * FROM USERS GROUP BY LOCATION);

Close SQLite

It is good practise to close all relational databases as soon as you are finished updating them.

In [132…

```
con.close()
```

All Done!

Great job. You now have a good idea for how to use sql and pandas with sql. You can create your own databases from csv files and you can do extensive querying using sql. These are valuable skills that will take you a long ways in todays technological world.