Project 4 Github Modified

April 16, 2023

1 ChatData

1.1 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.

1.2 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.

1.3 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.

2 Preliminary Steps: Create a Database

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

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

2.0.1 Create the Database

Now we will create the Sqlite database. We use the `%load_ext` magic command to load the SQL Magic extension and then use `%sql` to connect to the database.

[2]:
```
# 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.

[3]:
```
%%sql
DROP TABLE IF EXISTS 'queries'
```

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

[3]: []

3 Task 1: Load the Data

3.1 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.

3.2 Lifecycle Stage: Acquire

3.2.1 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:

[4]:
```
comments = pd.read_csv('comments.csv')
comments.head()
```

[4]:
```
        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^…

           CreationDate  UserId
0  2019-01-01 00:06:39   78575
1  2019-01-01 00:09:22    2126
2  2019-01-01 00:32:11    2126
3  2019-01-01 00:40:36    2126
4  2019-01-01 00:41:28  112141
```

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

[5]:
```
# 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()
```

[5]:
```
        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^…

           CreationDate  UserId
0  2019-01-01 00:06:39   78575
1  2019-01-01 00:09:22    2126
2  2019-01-01 00:32:11    2126
3  2019-01-01 00:40:36    2126
4  2019-01-01 00:41:28  112141
```

3.2.2 Load Other CSVs

[6]:
```
# Users data
users = pd.read_csv('users.csv')
users.head()
```

[6]:
```
        Id  Reputation         CreationDate DisplayName       LastAccessDate  \
0  157607          31  2017-04-17 14:50:42  user157607  2019-07-23 16:44:08
1  157656         101  2017-04-17 20:08:20  user102859  2019-06-26 13:42:13
2  157704         133  2017-04-18 05:10:47      jupiar  2019-11-25 13:32:27
3  157709         155  2017-04-18 06:39:18      farmer  2019-02-17 19:44:24
4  157755         101  2017-04-18 12:56:17      Miki P  2019-08-12 17:02:21

  WebsiteUrl         Location  \
0        NaN              NaN
1        NaN              NaN
2        NaN  Shanghai, China
3        NaN              NaN
4        NaN              NaN

                                             AboutMe  Views  UpVotes  \
0                                                NaN      0        0
1                                                NaN      3        0
2  <p>Originally from the U.K, I have an Undergra…      1        1
3                                                NaN     16        0
4                                                NaN      1        9

   DownVotes                                ProfileImageUrl  AccountId
0          0  https://www.gravatar.com/avatar/2efb161849efa4…   10705183
1          0      https://i.stack.imgur.com/eY4ka.jpg?s=128&g=1   10567606
2          0  https://www.gravatar.com/avatar/720e20205122c5…    9501631
3          0  https://www.gravatar.com/avatar/0f8c4bde3d8f25…   10709837
4          0  https://www.gravatar.com/avatar/af088558cd81c5…    7969290
```

[7]:
```
users.to_sql('users', con, if_exists='replace', index=False)

sql = 'SELECT * FROM users'
users = pd.read_sql(sql, con)
users.head()
```

[7]:
```
        Id  Reputation         CreationDate DisplayName       LastAccessDate  \
0  157607          31  2017-04-17 14:50:42  user157607  2019-07-23 16:44:08
1  157656         101  2017-04-17 20:08:20  user102859  2019-06-26 13:42:13
2  157704         133  2017-04-18 05:10:47      jupiar  2019-11-25 13:32:27
3  157709         155  2017-04-18 06:39:18      farmer  2019-02-17 19:44:24
4  157755         101  2017-04-18 12:56:17      Miki P  2019-08-12 17:02:21

  WebsiteUrl         Location  \
```

```
0       None             None
1       None             None
2       None  Shanghai, China
3       None             None
4       None             None

                                                AboutMe  Views  UpVotes  \
0                                                  None      0        0
1                                                  None      3        0
2  <p>Originally from the U.K, I have an Undergra…      1        1
3                                                  None     16        0
4                                                  None      1        9

   DownVotes                                     ProfileImageUrl  AccountId
0          0  https://www.gravatar.com/avatar/2efb161849efa4…   10705183
1          0      https://i.stack.imgur.com/eY4ka.jpg?s=128&g=1   10567606
2          0  https://www.gravatar.com/avatar/720e20205122c5…    9501631
3          0  https://www.gravatar.com/avatar/0f8c4bde3d8f25…   10709837
4          0  https://www.gravatar.com/avatar/af088558cd81c5…    7969290
```

[8]:
```
# Posts data
posts = pd.read_csv('posts.csv')
posts.head()
```

[8]:
```
        Id  PostTypeId  AcceptedAnswerId  ParentId         CreationDate  Score  \
0  423497           1            423511         0  2019-08-24 09:39:31      2
1  423498           1                 0         0  2019-08-24 09:47:42      1
2  423499           1                 0         0  2019-08-24 09:48:26      1
3  423500           2                 0    215865  2019-08-24 09:57:01      0
4  423502           2                 0    423286  2019-08-24 10:44:52      3

   ViewCount                                               Body  OwnerUserId  \
0         68  <p>From wikipedia <a href="https://en.wikipedi…        64552
1         24  <p>I am currently doing local sensitivity anal…        87231
2         56  <p>I'm an honours student in psychology doing …       257207
3          0  <p>Maybe you can look this <a href="https://me…       106606
4          0  <blockquote>\n  <p>Q1) Is my approach valid?</…       220643

  OwnerDisplayName  …  LastEditorDisplayName         LastEditDate  \
0              NaN  …                   NaN                  NaN
1              NaN  …                   NaN  2019-09-06 12:52:32
2              NaN  …                   NaN                  NaN
3              NaN  …                   NaN                  NaN
4              NaN  …                   NaN  2019-08-24 12:13:46

      LastActivityDate                                              Title  \
0  2019-08-24 11:38:54  When are biased estimators with lower MSE pref…
```

```
1  2019-09-06 12:52:32  How to interpret the result from local sensiti…
2  2019-08-25 08:54:17      Power Analysis in G-Power - Mixed Model Anova
3  2019-08-24 09:57:01                                                NaN
4  2019-08-24 12:13:46                                                NaN

                                Tags AnswerCount  CommentCount  \
0    <bias><unbiased-estimator><mse>           1             0
1  <sensitivity-analysis><elasticity>          1             0
2                   <anova><gpower>            2             0
3                                NaN           0             0
4                                NaN           0             1

   FavoriteCount           ClosedDate CommunityOwnedDate
0              1  2019-08-25 10:25:24                NaN
1              0                  NaN                NaN
2              0                  NaN                NaN
3              0                  NaN                NaN
4              0                  NaN                NaN

[5 rows x 21 columns]
```

[9]:
```
posts.to_sql('posts', con, if_exists='replace', index=False)

sql = 'SELECT * FROM posts'
posts = pd.read_sql(sql, con)
posts.head()
```

[9]:
```
       Id  PostTypeId  AcceptedAnswerId  ParentId         CreationDate  Score  \
0  423497           1            423511         0  2019-08-24 09:39:31      2
1  423498           1                 0         0  2019-08-24 09:47:42      1
2  423499           1                 0         0  2019-08-24 09:48:26      1
3  423500           2                 0    215865  2019-08-24 09:57:01      0
4  423502           2                 0    423286  2019-08-24 10:44:52      3

   ViewCount                                               Body  OwnerUserId  \
0         68  <p>From wikipedia <a href="https://en.wikipedi…        64552
1         24  <p>I am currently doing local sensitivity anal…        87231
2         56  <p>I'm an honours student in psychology doing …       257207
3          0  <p>Maybe you can look this <a href="https://me…       106606
4          0  <blockquote>\n  <p>Q1) Is my approach valid?</…       220643

  OwnerDisplayName  …  LastEditorDisplayName         LastEditDate  \
0             None  …                  None                 None
1             None  …                  None  2019-09-06 12:52:32
2             None  …                  None                 None
3             None  …                  None                 None
4             None  …                  None  2019-08-24 12:13:46
```

```
      LastActivityDate                                              Title  \
0  2019-08-24 11:38:54  When are biased estimators with lower MSE pref…
1  2019-09-06 12:52:32  How to interpret the result from local sensiti…
2  2019-08-25 08:54:17      Power Analysis in G-Power - Mixed Model Anova
3  2019-08-24 09:57:01                                               None
4  2019-08-24 12:13:46                                               None

                                Tags AnswerCount  CommentCount  \
0    <bias><unbiased-estimator><mse>           1             0
1  <sensitivity-analysis><elasticity>          1             0
2                    <anova><gpower>           2             0
3                               None           0             0
4                               None           0             1

   FavoriteCount           ClosedDate CommunityOwnedDate
0              1  2019-08-25 10:25:24               None
1              0                 None               None
2              0                 None               None
3              0                 None               None
4              0                 None               None

[5 rows x 21 columns]
```

3.2.3 TODO: Drop Duplicates

Look for and drop any duplicates in all 3 of the tables (if they exist).

[10]:
```
# Users
users.duplicated().sum()
```

[10]: 0

[11]:
```
# Comments
comments.duplicated().sum()
```

[11]: 0

[12]:
```
# TODO
posts.duplicated().sum()
```

[12]: 0

There seems to be no duplicates in all 3 datasets

3.3 Review the Data

[13]:
```
users.columns
```

[13]:
```
Index(['Id', 'Reputation', 'CreationDate', 'DisplayName', 'LastAccessDate',
       'WebsiteUrl', 'Location', 'AboutMe', 'Views', 'UpVotes', 'DownVotes',
       'ProfileImageUrl', 'AccountId'],
      dtype='object')
```

[14]:
```
users.head()
```

[14]:
```
       Id  Reputation         CreationDate DisplayName       LastAccessDate  \
0  157607          31  2017-04-17 14:50:42  user157607  2019-07-23 16:44:08
1  157656         101  2017-04-17 20:08:20  user102859  2019-06-26 13:42:13
2  157704         133  2017-04-18 05:10:47      jupiar  2019-11-25 13:32:27
3  157709         155  2017-04-18 06:39:18      farmer  2019-02-17 19:44:24
4  157755         101  2017-04-18 12:56:17      Miki P  2019-08-12 17:02:21

  WebsiteUrl         Location  \
0       None             None
1       None             None
2       None  Shanghai, China
3       None             None
4       None             None

                                             AboutMe  Views  UpVotes  \
0                                               None      0        0
1                                               None      3        0
2  <p>Originally from the U.K, I have an Undergra…      1        1
3                                               None     16        0
4                                               None      1        9

   DownVotes                                   ProfileImageUrl  AccountId
0          0  https://www.gravatar.com/avatar/2efb161849efa4…   10705183
1          0      https://i.stack.imgur.com/eY4ka.jpg?s=128&g=1   10567606
2          0  https://www.gravatar.com/avatar/720e20205122c5…    9501631
3          0  https://www.gravatar.com/avatar/0f8c4bde3d8f25…   10709837
4          0  https://www.gravatar.com/avatar/af088558cd81c5…    7969290
```

[15]:
```
comments.columns
```

[15]:
```
Index(['Id', 'PostId', 'Score', 'Text', 'CreationDate', 'UserId'],
dtype='object')
```

[16]:
```
comments.head()
```

[16]:
```
       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^…

          CreationDate  UserId
0  2019-01-01 00:06:39   78575
1  2019-01-01 00:09:22    2126
2  2019-01-01 00:32:11    2126
3  2019-01-01 00:40:36    2126
4  2019-01-01 00:41:28  112141
```

[17]:
```
posts.columns
```

[17]:
```
Index(['Id', 'PostTypeId', 'AcceptedAnswerId', 'ParentId', 'CreationDate',
       'Score', 'ViewCount', 'Body', 'OwnerUserId', 'OwnerDisplayName',
       'LastEditorUserId', 'LastEditorDisplayName', 'LastEditDate',
       'LastActivityDate', 'Title', 'Tags', 'AnswerCount', 'CommentCount',
       'FavoriteCount', 'ClosedDate', 'CommunityOwnedDate'],
      dtype='object')
```

[18]:
```
posts.head()
```

[18]:
```
       Id  PostTypeId  AcceptedAnswerId  ParentId         CreationDate  Score  \
0  423497           1            423511         0  2019-08-24 09:39:31      2
1  423498           1                 0         0  2019-08-24 09:47:42      1
2  423499           1                 0         0  2019-08-24 09:48:26      1
3  423500           2                 0    215865  2019-08-24 09:57:01      0
4  423502           2                 0    423286  2019-08-24 10:44:52      3

   ViewCount                                               Body  OwnerUserId  \
0         68  <p>From wikipedia <a href="https://en.wikipedi…        64552
1         24  <p>I am currently doing local sensitivity anal…        87231
2         56  <p>I'm an honours student in psychology doing …       257207
3          0  <p>Maybe you can look this <a href="https://me…       106606
4          0  <blockquote>\n  <p>Q1) Is my approach valid?</…       220643

  OwnerDisplayName  …  LastEditorDisplayName         LastEditDate  \
0             None  …                  None                 None
1             None  …                  None  2019-09-06 12:52:32
2             None  …                  None                 None
3             None  …                  None                 None
4             None  …                  None  2019-08-24 12:13:46

      LastActivityDate                                              Title  \
0  2019-08-24 11:38:54  When are biased estimators with lower MSE pref…
1  2019-09-06 12:52:32  How to interpret the result from local sensiti…
```

```
2  2019-08-25 08:54:17      Power Analysis in G-Power - Mixed Model Anova
3  2019-08-24 09:57:01                                                None
4  2019-08-24 12:13:46                                                None

                                 Tags AnswerCount  CommentCount  \
0     <bias><unbiased-estimator><mse>           1             0
1  <sensitivity-analysis><elasticity>           1             0
2                      <anova><gpower>           2             0
3                                None           0             0
4                                None           0             1

   FavoriteCount           ClosedDate CommunityOwnedDate
0              1  2019-08-25 10:25:24               None
1              0                 None               None
2              0                 None               None
3              0                 None               None
4              0                 None               None

[5 rows x 21 columns]
```

Users data

The Users data could be helpful in meeting the initiatives by creating awareness of how many users are using the Chatdata site and how often they are using it.

Comments data

The comments data could be useful in terms of showing how users interact with eachother through the comments they post and the nature of each comment.

Posts data

The posts data could assist in understanding what content Chatdata users post and how often they interact with the post features in terms of the favourites and comments.

Organisation of data

In terms of how the datasets are organised, they will fit well in terms of being managed in a relational database as a relational database is based on rows and columns, as well as being connected via a primary and foreign key, in which these 3 datasets seem to show

The primary keys are evident in all the datasets which is the ID column. For the comments table, there seems to be 2 foreign keys which are the PostID and the User ID.

Would the data give you a 3NF model?

This depends on whether the data firstly meets both the 1NF (Eliminating repeating groups) and 2NF (Removing partial dependencies) model. In terms of these datasets, they dont seem to have any repeated groups, each are individual to their own and the columns contain values that are the same type. So, they are 1NF.

The User data shows no partial dependencies. As for the comments and posts data, there are no partial dependencies too.

In order to meet a 3NF model, the data should have no transitive dependencies. In other words, the values should all depend on the primary key on the table, not on any other column.

Security and Legislation

The data in the users, comments and posts dataset could be considered as personal data since the data contains information about a users activity on Chatdata in terms of their posts, display name and comments. In terms of identifying a user, the location and display name could possibly lead to the users identity if they use their birth name as their display name. This could make the data unethical to use if a user could be identified since consent may be needed from the user. This could also cause a issue in privacy if the users activity data is being used.

4 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.

4.1 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:

[19]:
```
# This is an example
%sql SELECT COUNT(*) FROM comments
```

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

[19]:
```
[(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:

[20]:
```
%%sql
SELECT Id, PostId, Score, Text
    FROM comments
        LIMIT 5
```

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

[20]:
```
[(723182, 385124, 0, '@BenBolker I don\'t understand. The fit cannot be done for
the negative $y$. So intuitively I\'d think that in order to retain the
relativity of the  … (24 characters truncated) … rror" or "flip" it to the
positive axis? Would it be possible to adjust every point individually?
Basically e.g. take $abs(y_i)$ instead of minimums.'),
 (723183, 385124, 3, "You can't add *less* than (`-min(y)`), but you could add
```

```
*more*. I'm going to stop answering now sorry, because **judging what the 'best'
approach is … (60 characters truncated) …  of the analysis, why you need to
fit an exponential, your level of computational and statistical sophistication
and that of your audience, etc. …)"),
 (723186, 385137, 0, 'nice. If you felt like doing the work it would be nice to
generate an image/contour plot of log-likelihood as a function of (mu, theta)
and show the lines corresponding to the two `size` values .'),
 (723187, 385137, 0, 'i.e.
`emdbook::curve3d(-sum(dnbinom(y,mu=mu,size=size,log=TRUE)),\n
xlim=c(2,5),ylim=c(0.2,0.5),\n                  varnames=c("mu","s … (57
characters truncated) … ,col="red")\nabline(h=fit.what$theta,col="blue")\n`
… although doesn\'t look this is actually the answer - mu-hat is independent
of theta-hat … ?'),
 (723188, 385134, 0, 'Don\'t you mean "so variance should be
$\\sigma^2/(n\\mu^2)$"')]
```

4.2 Writing queries with Pandas

Another way to write queries is to use pandas:

[21]:
```
sql = """
SELECT Id, PostId, Score, Text
    FROM comments
        LIMIT 5
        """
result = pd.read_sql(sql, con)
result
```

[21]:

	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^…

5 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:

[22]:
```
%%sql
select sql from sqlite_master
```

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

[22]:
```
[('CREATE TABLE "comments" (\n"Id" INTEGER,\n  "PostId" INTEGER,\n  "Score"
INTEGER,\n  "Text" TEXT,\n  "CreationDate" TEXT,\n  "UserId" INTEGER\n)',),
 ('CREATE TABLE "users" (\n"Id" INTEGER,\n  "Reputation" INTEGER,\n
"CreationDate" TEXT,\n  "DisplayName" TEXT,\n  "LastAccessDate" TEXT,\n
"WebsiteUr … (17 characters truncated) … tion" TEXT,\n  "AboutMe" TEXT,\n
"Views" INTEGER,\n  "UpVotes" INTEGER,\n  "DownVotes" INTEGER,\n
"ProfileImageUrl" TEXT,\n  "AccountId" INTEGER\n)',),
 ('CREATE TABLE "posts" (\n"Id" INTEGER,\n  "PostTypeId" INTEGER,\n
"AcceptedAnswerId" INTEGER,\n  "ParentId" INTEGER,\n  "CreationDate" TEXT,\n
"Scor … (240 characters truncated) …   "Tags" TEXT,\n  "AnswerCount"
INTEGER,\n  "CommentCount" INTEGER,\n  "FavoriteCount" INTEGER,\n  "ClosedDate"
TEXT,\n  "CommunityOwnedDate" TEXT\n)',)]
```

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:

[23]:
```
%%sql
DROP TABLE comments;
DROP TABLE users;
DROP TABLE posts;
```

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

[23]:
```
[]
```

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

[24]:
```
%%sql
SELECT name FROM sqlite_master WHERE type='table'
    ORDER BY name
```

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

[24]:
```
[]
```

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

[25]:
```
%%sql
PRAGMA foreign_keys=ON;
```

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

[25]: []

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:

[26]:
```
%%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.
```

[26]: []

Now do the same for the posts table:

TODO: Complete the following code cell

[27]:
```
%%sql
CREATE TABLE "posts"(
    "Id" INTEGER NOT NULL PRIMARY KEY,
    "PostTypeId" INTEGER,
    "AcceptedAnswerId" INTEGER,
    "ParentId" INTEGER,
    "CreationDate" DATE,
    "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 user(id)
    );
```

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

[27]: []

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.

[28]:
```
%%sql
CREATE TABLE "comments"(
    "Id" INTEGER NOT NULL PRIMARY KEY,
    "PostId" INTEGER,
    "Score" INTEGER,
    "Text" TEXT,
    "CreationDate" TEXT,
    "UserId" INTEGER,
    FOREIGN KEY(UserId) REFERENCES user(Id)
    FOREIGN KEY(PostId) REFERENCES post(Id)
);
```

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

[28]: []

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

[29]:
```
users.to_sql('users', con, if_exists='replace', index=False)
```

[29]: 18412

Now posts:

[30]:
```
posts.to_sql('posts', con, if_exists='replace', index=False)
```

[30]: 42234

Finally comments, which references the users and posts tables:

[31]:
```
# Insert data into the new comments table
# TODO
comments.to_sql('comments', con, if_exists='replace', index=False)
```

[31]:
```
50000
```

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

[32]:
```
%%sql
SELECT name FROM sqlite_master WHERE type='table'
    ORDER BY name
```

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

[32]:
```
[('comments',), ('posts',), ('users',)]
```

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

6 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.

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?

6.1 Prepare the Sqlite query

First, let's write the query:

[33]:
```
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
```

[33]:

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

```
4  3277       29255
```

6.2 Test the query

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

[34]:
```
results = users.groupby(['Id']).sum().sort_values('Views', ascending = False)[:
 ↪5]
results['Views']
```

[34]:
```
Id
919     85180
4253    35119
805     34637
7290    32639
3277    29255
Name: Views, dtype: int64
```

6.3 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.

[35]:
```
query_dict = {
              'task': ['Single Table Queries'],
              'action': ['Which 5 users have viewed the most times and what is␣
 ↪the sum of those views per user?'],
              'query': [sql]
             }
query_dict
```

[35]:
```
{'task': ['Single Table Queries'],
 'action': ['Which 5 users have viewed the most times and what is the sum of
those views 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:

[36]:
```
queries = pd.DataFrame(query_dict)
queries
```

[36]:
```
                   task                                                 action  \
0  Single Table Queries  Which 5 users have viewed the most times and w…

                                                query
0  \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.

[37]:
```
# 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()
```

[37]:
```
                   task                                                 action  \
0  Single Table Queries  Which 5 users have viewed the most times and w…

                                                query
0  \nSELECT Id, SUM(Views) AS TotalViews\n    FRO…
```

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

7 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:

[38]:
```
# 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.

[39]:
```
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:

[40]:
```
# Prove it works
%sql SELECT * FROM queries
```

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

[40]:
```
[('Single Table Queries', 'Which 5 users have viewed the most times and what is
the sum of those views per user?', '\nSELECT Id, SUM(Views) AS TotalViews\n
FROM Users\n        GROUP BY Id\n            ORDER BY TotalViews DESC\n
LIMIT 5\n                ),
 ('Task 1', 'Create table comments', '\n    CREATE TABLE "comments" (\n    "Id"
INTEGER,\n    "PostId" INTEGER,\n    "Score" INTEGER,\n    "Text" TEXT,\n
"CreationDate" TEXT,\n    "UserId" INTEGER\n    )\n    ')]
```

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

[41]:
```
# 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" DATE,
    "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 user(id)
    )
    """
store_query("Task 1", "Create table posts", sql)
```

[42]:
```
%sql SELECT * FROM queries
```

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

[42]:
```
[('Single Table Queries', 'Which 5 users have viewed the most times and what is
the sum of those views per user?', '\nSELECT Id, SUM(Views) AS TotalViews\n
FROM Users\n        GROUP BY Id\n            ORDER BY TotalViews DESC\n
LIMIT 5\n               '),
 ('Task 1', 'Create table comments', '\n    CREATE TABLE "comments" (\n    "Id"
INTEGER,\n    "PostId" INTEGER,\n    "Score" INTEGER,\n    "Text" TEXT,\n
"CreationDate" TEXT,\n    "UserId" INTEGER\n    )\n    '),
 ('Task 1', 'Create table posts', '\n    CREATE TABLE "posts" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "PostTypeId" INTEGER,\n    "AcceptedAnswerId"
INTEGER,\n    "ParentId" INTE … (372 characters truncated) … \n
"FavoriteCount" INTEGER,\n    "ClosedDate" TEXT,\n    "CommunityOwnedDate"
TEXT,\n    FOREIGN KEY(OwnerUserId) REFERENCES user(id)\n    )\n    ')]
```

[43]:
```
# 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)
```

[44]:
```
%sql SELECT * FROM queries
```

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

[44]:
```
[('Single Table Queries', 'Which 5 users have viewed the most times and what is
the sum of those views per user?', '\nSELECT Id, SUM(Views) AS TotalViews\n
FROM Users\n        GROUP BY Id\n            ORDER BY TotalViews DESC\n
LIMIT 5\n               '),
 ('Task 1', 'Create table comments', '\n    CREATE TABLE "comments" (\n    "Id"
INTEGER,\n    "PostId" INTEGER,\n    "Score" INTEGER,\n    "Text" TEXT,\n
"CreationDate" TEXT,\n    "UserId" INTEGER\n    )\n    '),
 ('Task 1', 'Create table posts', '\n    CREATE TABLE "posts" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "PostTypeId" INTEGER,\n    "AcceptedAnswerId"
INTEGER,\n    "ParentId" INTE … (372 characters truncated) … \n
"FavoriteCount" INTEGER,\n    "ClosedDate" TEXT,\n    "CommunityOwnedDate"
TEXT,\n    FOREIGN KEY(OwnerUserId) REFERENCES user(id)\n    )\n    '),
 ('Task 1', 'Create table users', '\n    CREATE TABLE "users" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "Reputation" INTEGER,\n    "CreationDate"
TEXT,\n    "DisplayName" TEXT,\n  … (82 characters truncated) … tMe" TEXT,\n
"Views" INTEGER,\n    "UpVotes" INTEGER,\n    "DownVotes" INTEGER,\n
"ProfileImageUrl" TEXT,\n    "AccountId" INTEGER\n    )\n    ')]
```

7.2 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.

[45]:
```
# Count the number of rows in the comments table
sql = """
    SELECT COUNT(*) as comment_count
    FROM comments
    """
store_query("Task 1", "Count number of rows in comments table", sql)
```

[46]:
```
%%sql
SELECT COUNT(*) as comment_count
FROM comments
```

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

[46]:
```
[(50000,)]
```

[47]:
```
%sql SELECT * FROM queries
```

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

[47]:
```
[('Single Table Queries', 'Which 5 users have viewed the most times and what is
the sum of those views per user?', '\nSELECT Id, SUM(Views) AS TotalViews\n
FROM Users\n        GROUP BY Id\n            ORDER BY TotalViews DESC\n
LIMIT 5\n                ),
 ('Task 1', 'Create table comments', '\n    CREATE TABLE "comments" (\n    "Id"
INTEGER,\n    "PostId" INTEGER,\n    "Score" INTEGER,\n    "Text" TEXT,\n
"CreationDate" TEXT,\n    "UserId" INTEGER\n    )\n    '),
 ('Task 1', 'Create table posts', '\n    CREATE TABLE "posts" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "PostTypeId" INTEGER,\n    "AcceptedAnswerId"
INTEGER,\n    "ParentId" INTE … (372 characters truncated) … \n
"FavoriteCount" INTEGER,\n    "ClosedDate" TEXT,\n    "CommunityOwnedDate"
TEXT,\n    FOREIGN KEY(OwnerUserId) REFERENCES user(id)\n    )\n    '),
 ('Task 1', 'Create table users', '\n    CREATE TABLE "users" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "Reputation" INTEGER,\n    "CreationDate"
TEXT,\n    "DisplayName" TEXT,\n  … (82 characters truncated) … tMe" TEXT,\n
"Views" INTEGER,\n    "UpVotes" INTEGER,\n    "DownVotes" INTEGER,\n
"ProfileImageUrl" TEXT,\n    "AccountId" INTEGER\n    )\n    '),
 ('Task 1', 'Count number of rows in comments table', '\n    SELECT COUNT(*) as
comment_count\n    FROM comments\n    ')]
```

[48]:
```
# Count the number of rows in the users table
sql = """
    SELECT COUNT(*) as user_count
    FROM users
    """
store_query("Task 1", "Count number of rows in users table", sql)
```

[49]:
```
%%sql
SELECT COUNT(*) as user_count
FROM users
```

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

[49]:
```
[(18412,)]
```

[50]:
```
# Count the number of rows in the posts table
sql = """
    SELECT COUNT(*) as post_count
    FROM posts
    """
store_query("Task 1", "Count number of rows in posts table", sql)
```

[51]:
```
%%sql
SELECT COUNT(*) as post_count
FROM posts
```

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

[51]:
```
[(42234,)]
```

[52]:
```
%sql SELECT * FROM queries
```

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

[52]:
```
[('Single Table Queries', 'Which 5 users have viewed the most times and what is
the sum of those views per user?', '\nSELECT Id, SUM(Views) AS TotalViews\n
FROM Users\n        GROUP BY Id\n            ORDER BY TotalViews DESC\n
LIMIT 5\n                ),
 ('Task 1', 'Create table comments', '\n    CREATE TABLE "comments" (\n    "Id"
INTEGER,\n    "PostId" INTEGER,\n    "Score" INTEGER,\n    "Text" TEXT,\n
"CreationDate" TEXT,\n    "UserId" INTEGER\n    )\n    '),
 ('Task 1', 'Create table posts', '\n    CREATE TABLE "posts" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "PostTypeId" INTEGER,\n    "AcceptedAnswerId"
INTEGER,\n    "ParentId" INTE … (372 characters truncated) … \n
"FavoriteCount" INTEGER,\n    "ClosedDate" TEXT,\n    "CommunityOwnedDate"
TEXT,\n    FOREIGN KEY(OwnerUserId) REFERENCES user(id)\n    )\n    '),
 ('Task 1', 'Create table users', '\n    CREATE TABLE "users" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "Reputation" INTEGER,\n    "CreationDate"
TEXT,\n    "DisplayName" TEXT,\n  … (82 characters truncated) … tMe" TEXT,\n
"Views" INTEGER,\n    "UpVotes" INTEGER,\n    "DownVotes" INTEGER,\n
"ProfileImageUrl" TEXT,\n    "AccountId" INTEGER\n    )\n    '),
 ('Task 1', 'Count number of rows in comments table', '\n    SELECT COUNT(*) as
comment_count\n    FROM comments\n    '),
 ('Task 1', 'Count number of rows in users table', '\n    SELECT COUNT(*) as
user_count\n    FROM users\n    '),
 ('Task 1', 'Count number of rows in posts table', '\n    SELECT COUNT(*) as
post_count\n    FROM posts\n    ')]
```

7.3 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:

[53]:
```
# Run the query to select 5 random rows from the comments table
sql = """
    SELECT * FROM COMMENTS
    ORDER BY RANDOM()
    LIMIT 5;
    """
store_query("Task 1", "Select 5 random rows from comments table", sql)
```

[54]:
```
%%sql
SELECT * FROM COMMENTS
    ORDER BY RANDOM()
    LIMIT 5;
```

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

[54]:
```
[(732279, 389935, 0, "Thank you but I still don't get it. It does not seem to
solve what I am intending on doing. What I want is: I have x subjects and every
subject creat … (287 characters truncated) … fer from each other. One
subject can create the values {15031,15029,15032} while another can create
{31,29,32} and they will have the same SD anyway.", '2019-01-30 15:04:09',
235888),
 (765181, 409586, 0, 'Those are all numerical variables from 1 to 100. The
respondant is shown a picture of a bottle with a specific type, design etc. and
rates certain parameters from 1 to 100.', '2019-05-22 14:03:47', 153364),
 (751334, 401389, 0, '+1.  Your answer is very clear and well explained.
Welcome to our site!', '2019-04-05 16:24:15', 919),
 (726635, 386853, 0, 'Thank you very much! "The -1 removes the intercept and no
longer compares each region against the reference region"  Does this mean that
it automatically compares it against 0?', '2019-01-12 22:26:25', 189757),
 (757453, 405204, 0, 'Can you edit your post to include your data, by pasting in
the result of `dput()` applied to your series?', '2019-04-26 11:20:06', 1352)]
```

[55]:
```
%sql SELECT * FROM queries
```

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

[55]:
```
[('Single Table Queries', 'Which 5 users have viewed the most times and what is
the sum of those views per user?', '\nSELECT Id, SUM(Views) AS TotalViews\n
FROM Users\n        GROUP BY Id\n            ORDER BY TotalViews DESC\n
LIMIT 5\n                '),
 ('Task 1', 'Create table comments', '\n    CREATE TABLE "comments" (\n    "Id"
INTEGER,\n    "PostId" INTEGER,\n    "Score" INTEGER,\n    "Text" TEXT,\n
"CreationDate" TEXT,\n    "UserId" INTEGER\n    )\n    '),
 ('Task 1', 'Create table posts', '\n    CREATE TABLE "posts" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "PostTypeId" INTEGER,\n    "AcceptedAnswerId"
INTEGER,\n    "ParentId" INTE … (372 characters truncated) … \n
"FavoriteCount" INTEGER,\n    "ClosedDate" TEXT,\n    "CommunityOwnedDate"
TEXT,\n    FOREIGN KEY(OwnerUserId) REFERENCES user(id)\n    )\n    '),
 ('Task 1', 'Create table users', '\n    CREATE TABLE "users" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "Reputation" INTEGER,\n    "CreationDate"
TEXT,\n    "DisplayName" TEXT,\n  … (82 characters truncated) … tMe" TEXT,\n
"Views" INTEGER,\n    "UpVotes" INTEGER,\n    "DownVotes" INTEGER,\n
"ProfileImageUrl" TEXT,\n    "AccountId" INTEGER\n    )\n    '),
 ('Task 1', 'Count number of rows in comments table', '\n    SELECT COUNT(*) as
```

```
comment_count\n    FROM comments\n    '),
 ('Task 1', 'Count number of rows in users table', '\n    SELECT COUNT(*) as
user_count\n    FROM users\n    '),
 ('Task 1', 'Count number of rows in posts table', '\n    SELECT COUNT(*) as
post_count\n    FROM posts\n    '),
 ('Task 1', 'Select 5 random rows from comments table', '\n    SELECT * FROM
COMMENTS\n    ORDER BY RANDOM()\n    LIMIT 5;\n    ')]
```

[56]:
```
# Run the query to select 5 random rows from the posts table
sql = """
    SELECT * FROM Posts
    ORDER BY RANDOM()
    LIMIT 5;
    """
store_query("Task 1", "Select 5 random rows from posts table", sql)
```

[57]:
```
%%sql
SELECT * FROM Posts
ORDER BY RANDOM()
LIMIT 5;
```

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

[57]:
```
[(400728, 2, 0, 400727, '2019-04-02 08:15:51', 2, 0, '<p>"This means their pre-
test probability was around 85%. This is higher than their post-test probability
on a positive test"</p>\n\n<p>If this state … (50 characters truncated) …
rthless. Still, it would be interesting to know about the reliability of those
estimates as well as statistical significance of that difference</p>\n', 238499,
None, 0, None, None, '2019-04-02 08:15:51', None, None, 0, 0, 0, None, None),
 (420866, 1, 0, 0, '2019-08-06 13:39:47', 2, 47, '<p>I followed this guy\'s <a
href="https://www.youtube.com/watch?v=wQ8BIBpya2k" rel="nofollow
noreferrer">tutorial</a> on YouTube. Following is the c … (998 characters
truncated) … code:</p>\n\n<ul>\n<li>Where is the input layer?</li>\n<li>How
are \'128\' neurons are chosen?</li>\n<li>And why use two hidden
layers?</li>\n</ul>\n', 194144, None, 0, None, None, '2019-08-06 15:17:32', 'How
to choose number of neurons and hidden layers?', '<neural-
networks><tensorflow><keras>', 0, 0, 0, '2019-08-06 15:42:39', None),
 (395573, 2, 0, 395196, '2019-03-04 17:11:14', 0, 0, '<p>You can try out the
code and see that your intuition is
incorrect:</p>\n\n<pre><code>set.seed(1234)\n\nx &lt;- matrix(rnorm(30*3), ncol
= 3)\n\nco … (29 characters truncated) … None of the correlations are
1.00</p>\n\n<p>However, if you change it to 30*2 matrix and ncol = 2 then all
the correlations are either 1 or -1.</p>\n', 686, None, 0, None, None,
'2019-03-04 17:11:14', None, None, 0, 2, 0, None, None),
 (407247, 1, 0, 0, '2019-05-08 11:36:51', 1, 40, '<p>Given that we know A and B
are independent and they never occur at the same time, one of them must be
impossible, no? \n<span class="math-containe … (593 characters truncated) …
```

```
k, just engagement with this on my own, to refresh my stats knowledge in
combination with a learning about data analytics (Bayes\' classifier).</p>\n',
225012, None, 225012, None, '2019-05-08 14:23:41', '2019-05-08 14:25:12', 'Does
independence and mutual exclusivity induce impossibility?', '<probability><self-
study><bayesian><conditional-probability>', 1, 4, 0, None, None),
 (431960, 2, 0, 431851, '2019-10-17 22:07:43', 9, 0, '<p>The use of Greek
letters in modern mathematics is basically a consequence of the humanist
education.  The normal distribution was introduced by Ga … (426 characters
truncated) … e been the motivation for giving them those letters.  The
equivalent Latin (rather than German) terms would likely also start with "M" and
"S".</p>\n', 0, 'user262993', 0, None, None, '2019-10-17 22:07:43', None, None,
0, 0, 0, None, None)]
```

[58]:
```
# Run the query to select 5 random rows from the users table
sql = """
    SELECT * FROM Users
    ORDER BY RANDOM()
    LIMIT 5;
    """
store_query("Task 1", "Select 5 random rows from users table", sql)
```

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

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

[59]:
```
[(209550, 111, '2018-05-25 03:19:42', 'LeGeniusII', '2019-10-30 01:27:13', None,
None, None, 3, 3, 0, 'https://i.stack.imgur.com/yMlsU.jpg?s=128&g=1', 9435951),
 (43249, 111, '2014-04-07 13:53:20', 'GrigorisG', '2019-11-15 15:31:27', None,
None, None, 5, 1, 0, 'https://www.gravatar.com/avatar/ddead34944aae545a6f960957e
4a64ff?s=128&d=identicon&r=PG', 1900996),
 (169269, 135, '2017-07-18 09:15:20', 'Lodore66', '2019-11-29 17:09:12', None,
None, None, 1, 2, 0, 'https://i.stack.imgur.com/TwuTB.jpg?s=128&g=1', 8376199),
 (248344, 1, '2019-05-18 20:59:11', 'Horstus', '2019-05-19 16:02:52', None,
None, None, 1, 0, 0, 'https://lh5.googleusercontent.com/-EtpQK4KLWig/AAAAAAAAAAI
/AAAAAAAAAZ4/Crv3ViZ_B3Y/photo.jpg?sz=128', 6160325),
 (240672, 11, '2019-03-11 08:07:12', 'Norbert Bátfai', '2019-06-20 14:26:00',
'https://arato.inf.unideb.hu/batfai.norbert/', None, '<p>Norbert Bátfai received
his M.Sc. (summa cum laude) in Computer Science in 1998 at the Kossuth Lajos
University (KLTE), Debrecen, Hungary. In 1999 … (1100 characters truncated)
… href="https://hu.linkedin.com/in/b%C3%A1tfai-norbert-863b237b" rel="nofollow
noreferrer">https://hu.linkedin.com/in/bátfai-norbert-863b237b</a></p>\n', 0, 0,
0, 'https://graph.facebook.com/1378157075536717/picture?type=large', 9824447)]
```

[60]:
```
%sql SELECT * FROM queries
```

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

[60]:
```
[('Single Table Queries', 'Which 5 users have viewed the most times and what is
the sum of those views per user?', '\nSELECT Id, SUM(Views) AS TotalViews\n
FROM Users\n        GROUP BY Id\n            ORDER BY TotalViews DESC\n
LIMIT 5\n                ),
 ('Task 1', 'Create table comments', '\n    CREATE TABLE "comments" (\n    "Id"
INTEGER,\n    "PostId" INTEGER,\n    "Score" INTEGER,\n    "Text" TEXT,\n
"CreationDate" TEXT,\n    "UserId" INTEGER\n    )\n    '),
 ('Task 1', 'Create table posts', '\n    CREATE TABLE "posts" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "PostTypeId" INTEGER,\n    "AcceptedAnswerId"
INTEGER,\n    "ParentId" INTE … (372 characters truncated) … \n
"FavoriteCount" INTEGER,\n    "ClosedDate" TEXT,\n    "CommunityOwnedDate"
TEXT,\n    FOREIGN KEY(OwnerUserId) REFERENCES user(id)\n    )\n    '),
 ('Task 1', 'Create table users', '\n    CREATE TABLE "users" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "Reputation" INTEGER,\n    "CreationDate"
TEXT,\n    "DisplayName" TEXT,\n  … (82 characters truncated) … tMe" TEXT,\n
"Views" INTEGER,\n    "UpVotes" INTEGER,\n    "DownVotes" INTEGER,\n
"ProfileImageUrl" TEXT,\n    "AccountId" INTEGER\n    )\n    '),
 ('Task 1', 'Count number of rows in comments table', '\n    SELECT COUNT(*) as
comment_count\n    FROM comments\n    '),
 ('Task 1', 'Count number of rows in users table', '\n    SELECT COUNT(*) as
user_count\n    FROM users\n    '),
 ('Task 1', 'Count number of rows in posts table', '\n    SELECT COUNT(*) as
post_count\n    FROM posts\n    '),
 ('Task 1', 'Select 5 random rows from comments table', '\n    SELECT * FROM
COMMENTS\n    ORDER BY RANDOM()\n    LIMIT 5;\n    '),
 ('Task 1', 'Select 5 random rows from posts table', '\n    SELECT * FROM
Posts\n    ORDER BY RANDOM()\n    LIMIT 5;\n    '),
 ('Task 1', 'Select 5 random rows from users table', '\n    SELECT * FROM
Users\n    ORDER BY RANDOM()\n    LIMIT 5;\n    ')]
```

8 Task 2: Create Single Table Queries

8.1 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:

[61]:
```
%%sql
ALTER TABLE POSTS ADD COLUMN LEN_BODY INT
```

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

[61]:
```
[]
```

[62]:
```
%%sql
UPDATE POSTS SET LEN_BODY = LENGTH(BODY)
```

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

[62]:
```
[]
```

8.1.1 Single Table Queries

How many Posts have 0 comments?

[63]:
```
%%sql
SELECT COUNT() FROM posts
WHERE "CommentCount" = 0
```

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

[63]:
```
[(21713,)]
```

[64]:
```
sql = """
SELECT COUNT() FROM posts
WHERE "CommentCount" = 0
"""

store_query("Task 2", "How many posts have 0 comments?", sql)
```

How many posts have 1 comment?

[65]:
```
%%sql
SELECT COUNT() FROM posts
WHERE "CommentCount" = 1
```

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

[65]:
```
[(6460,)]
```

[66]:
```
sql = """
SELECT COUNT() FROM posts
WHERE "CommentCount" = 1
"""

store_query("Task 2", "How many posts have 1 comment?", sql)
```

How many posts have 2 comments or more?

[67]:
```
%%sql
SELECT COUNT() FROM posts
WHERE "CommentCount" >= 2
```

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

[67]:
```
[(14061,)]
```

[68]:
```
sql = """
SELECT COUNT() FROM posts
WHERE "CommentCount" >=2
"""

store_query("Task 2", "How many posts have 2 comments or more?", sql)
```

Find 5 posts with the highest viewcount

[69]:
```
%%sql
SELECT *
FROM posts
ORDER BY ViewCount DESC
LIMIT 5
```

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

[69]:
```
[(388566, 1, 388582, 0, '2019-01-22 15:16:47', 56, 19542, '<ul>\n<li>Statement
One (S1): "One in 80 deaths is caused by a car accident."</li>\n<li>Statement
Two (S2): "One in 80 people dies as a result of a c … (2000 characters
truncated) …  interpretation (their default for S2 is a much stronger
assumption), or if they have some innate statistical sense that I\'m in fact
lacking. </p>\n', 228214, None, 164061, None, '2019-01-22 21:40:39', '2019-01-24
17:09:47', 'Is it wrong to rephrase "1 in 80 deaths is caused by a car accident"
as "1 in 80 people die as a result of a car accident?"',
'<interpretation><risk>', 9, 15, 15, None, None, 2270),
 (394118, 1, 394128, 0, '2019-02-24 14:07:11', 64, 16317, '<p>A human child at
age 2 needs around 5 instances of a car to be able to identify it with
reasonable accuracy regardless of color, make, etc. When m … (217 characters
truncated) … >What is it that artificial neural networks are missing that
prevent them from being able to learn way quicker? Is transfer learning an
answer?</p>\n', 107213, None, 7291, None, '2019-02-25 22:40:22', '2019-03-03
17:37:05', 'Why do neural networks need so many training examples to perform?',
'<neural-networks><neuroscience>', 12, 24, 38, None, None, 512),
 (431370, 1, 431397, 0, '2019-10-14 11:29:21', 77, 11723, '<p>It seems very
counter intuitive to many people that a given diagnostic test with very high
accuracy (say 99%) can generate massively more false po … (520 characters
truncated) … lp me explain it to a lay person.</p>\n\n<p>Apologies if this is
the wrong forum to ask this. If so please direct me to a more appropriate
one.</p>\n', 262594, None, 11887, None, '2019-11-28 01:44:34', '2019-11-28
01:44:34', 'Is there a name for the phenomenon of false positives
counterintuitively outstripping true positives',
'<probability><terminology><intuition>', 8, 9, 18, None, None, 811),
```

```
 (398646, 1, 398653, 0, '2019-03-21 01:19:52', 61, 9850, '<p>The title of the
Comment in Nature <a href="https://www.nature.com/articles/d41586-019-00857-9"
rel="noreferrer">Scientists rise up against statis … (1880 characters
truncated) … pft.jpg" rel="noreferrer"><img
src="https://i.stack.imgur.com/tgpft.jpg" alt="Nature: Scientists rise up
against statistical significance"></a></p>\n', 163067, None, 163067, None,
'2019-03-22 22:14:04', '2019-03-30 19:35:27', 'What does "Scientists rise up
against statistical significance" mean? (Comment in Nature)', '<statistical-
significance><p-value><bias>', 10, 7, 34, None, None, 2148),
 (434128, 1, 434579, 0, '2019-11-01 13:07:36', 73, 6718, '<p>I am designing a
one year program in data analysis with a local community college. The program
aims to prepare students to handle basic tasks in d … (891 characters
truncated) … ur in the area of variable selection and sampling design. I\'m
interested in paradoxes that occur in other areas -- like the analysis as such.
</p>\n', 14188, None, 0, None, None, '2019-11-26 00:59:15', 'Famous statistical
wins and horror stories for teaching purposes', '<mathematical-statistics><data-
visualization><experiment-design><teaching>', 13, 7, 70, None, '2019-11-01
15:12:41', 1172)]
```

[70]:
```
sql = """
SELECT *
FROM posts
ORDER BY ViewCount DESC
LIMIT 5
"""

store_query("Task 2", "Find 5 posts with the highest viewcount", sql)
```

Find 5 posts with the highest scores

[71]:
```
%%sql
SELECT *
FROM posts
ORDER BY Score DESC
LIMIT 5
```

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

[71]:
```
[(431397, 2, 0, 431370, '2019-10-14 14:29:36', 101, 0, '<p>Yes there is.
Generally it is termed <strong>base rate fallacy</strong> or more specific
<strong>false positive paradox</strong>. There is even a wikipedia article about
it: <a href="https://en.wikipedia.org/wiki/Base_rate_fallacy"
rel="noreferrer">see here</a></p>\n', 142976, None, 0, None, None, '2019-10-14
14:29:36', None, None, 0, 0, 0, None, None, 269),
 (394128, 2, 0, 394118, '2019-02-24 15:44:44', 100, 0, '<p>I caution against
expecting strong resemblance between biological and artificial neural networks.
I think the name "neural networks" is a bit dang … (5584 characters truncated)
```

```
… l volume, diversity and resolution of the training data.</p>\n\n<p>*We
don\'t presently have a tags for one-shot learning or few-shot learning.</p>\n',
22311, None, 22311, None, '2019-03-03 17:37:05', '2019-03-03 17:37:05', None,
None, 0, 15, 0, None, None, 5829),
 (426878, 2, 0, 426873, '2019-09-11 23:23:31', 93, 0, '<p><strong>tl;dr</strong>
Even though this is an image classification dataset, it remains a <strong>very
easy</strong> task, for which one can easily … (4424 characters truncated) …
and out more\n    frame1 = plt.gca()\n
frame1.axes.get_xaxis().set_visible(False)\n
frame1.axes.get_yaxis().set_visible(False)\n</code></pre>\n', 119015, None,
119015, None, '2019-09-13 14:02:28', '2019-09-13 14:02:28', None, None, 0, 6, 0,
None, None, 4627),
 (388578, 2, 0, 388566, '2019-01-22 15:48:47', 80, 0, '<p>To me "1 in 80
deaths…" is by far the clearer statement.  The denominator in your "1 in 80"
is the set of all death events and that statement ma … (282 characters
truncated) … rence set in probability or frequency assertions like this. If
you\'re talking about the proportion of deaths, then say "deaths" not "people".
</p>\n', 227039, None, 0, None, None, '2019-01-22 15:48:47', None, None, 0, 11,
0, None, None, 572),
 (431370, 1, 431397, 0, '2019-10-14 11:29:21', 77, 11723, '<p>It seems very
counter intuitive to many people that a given diagnostic test with very high
accuracy (say 99%) can generate massively more false po … (520 characters
truncated) … lp me explain it to a lay person.</p>\n\n<p>Apologies if this is
the wrong forum to ask this. If so please direct me to a more appropriate
one.</p>\n', 262594, None, 11887, None, '2019-11-28 01:44:34', '2019-11-28
01:44:34', 'Is there a name for the phenomenon of false positives
counterintuitively outstripping true positives',
'<probability><terminology><intuition>', 8, 9, 18, None, None, 811)]
```

[72]:
```
sql = """
SELECT *
FROM posts
ORDER BY Score DESC
LIMIT 5
"""

store_query("Task 2", "Find 5 posts with the highest Scores", sql)
```

What are the 5 most frequent scores on posts?

[73]:
```
%%sql
SELECT Score, COUNT(Score) AS Frequency
FROM posts
GROUP BY Score
ORDER BY COUNT(Score) DESC
LIMIT 5
```

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

```
Done.
```

[73]:
```
[(0, 19888), (1, 11867), (2, 5094), (3, 2228), (4, 1059)]
```

[74]:
```
sql = """
SELECT Score, COUNT(Score) AS Frequency
FROM posts
GROUP BY Score
ORDER BY COUNT(Score) 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?

[75]:
```
%%sql
SELECT COUNT() FROM posts
WHERE Tags LIKE "%data%"
```

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

[75]:
```
[(2242,)]
```

[76]:
```
sql = """
SELECT 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?

[77]:
```
%%sql
SELECT CommentCount, COUNT(CommentCount) AS Frequency
FROM posts
GROUP BY CommentCount
ORDER BY COUNT(CommentCount) DESC
LIMIT 5
```

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

[77]:
```
[(0, 21713), (1, 6460), (2, 4966), (3, 3063), (4, 2026)]
```

[78]:
```
sql = """
SELECT CommentCount, COUNT(CommentCount) AS Frequency
FROM posts
```

```
GROUP BY CommentCount
ORDER BY COUNT(CommentCount) DESC
LIMIT 5
"""

store_query("Task 2", "What are the 5 most frequent commentcount for posts?",␣
 ↪sql)
```

How many posts have an accepted answer?

[79]:
```
%%sql
SELECT COUNT() FROM posts
WHERE "AcceptedAnswerId" >0
```

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

[79]:
```
[(5341,)]
```

[80]:
```
sql = """
SELECT 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?

[81]:
```
%%sql
SELECT AVG(Reputation)
FROM users
```

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

[81]:
```
[(312.3509124484032,)]
```

[82]:
```
sql = """
SELECT AVG(Reputation)
FROM users
"""

store_query("Task 2", "What is the average reputation of table users?", sql)
```

What are the min and max reputation of users?

[83]:
```
%%sql
SELECT MAX(Reputation),MIN(Reputation)
FROM users
```

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

[83]:
```
[(228662, 1)]
```

[84]:
```
sql = """
SELECT MAX(Reputation),MIN(Reputation)
FROM users
"""

store_query("Task 2", "What is the min and max reputation of users?", sql)
```

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

[85]:
```
%%sql
SELECT ViewCount, LEN_BODY
FROM posts
ORDER BY ViewCount DESC
LIMIT 5
```

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

[85]:
```
[(19542, 2270), (16317, 512), (11723, 811), (9850, 2148), (6718, 1172)]
```

[86]:
```
sql = """
SELECT ViewCount, LEN_BODY
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?

[87]:
```
%%sql
SELECT COUNT (DISTINCT(Location)) AS NumberofLocations
From users
WHERE Location IS NOT NULL;
```

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

[87]:
```
[(1900,)]
```

[88]:
```
sql = """
SELECT COUNT (DISTINCT(Location)) AS NumberofLocations
From users
```

```
WHERE Location IS NOT NULL
"""

store_query("Task 2", "How many different locations are there in the users␣
 ↪table?", sql)
```

What are the top 5 locations of users?

[89]:
```
%%sql
SELECT Location, COUNT(Location) AS Frequency
FROM users
GROUP BY Location
ORDER BY COUNT(Location) DESC
LIMIT 5
```

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

[89]:
```
[('Germany', 117),
 ('India', 100),
 ('United States', 69),
 ('Paris, France', 66),
 ('London, United Kingdom', 63)]
```

[90]:
```
sql = """
SELECT Location, COUNT(Location) AS Frequency
FROM users
GROUP BY Location
ORDER BY COUNT(Location) 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

[91]:
```
%%sql
SELECT(ViewCount/CreationDate)*100.0 AS VolumeViewCount,
CreationDate,
strftime('%w', CreationDate),
CASE CAST(strftime('%w', CreationDate) as integer)
    WHEN 0 THEN 'Sunday'
    WHEN 1 THEN 'Monday'
    WHEN 2 THEN 'Tuesday'
    WHEN 3 THEN 'Wednesday'
    WHEN 4 THEN 'Thursday'
    WHEN 5 THEN 'Friday'
    ELSE 'Saturday' END AS 'DayOfWeek'
```

```
FROM posts
ORDER BY VolumeViewCount DESC
```

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

[91]:
```
[(900.0, '2019-01-22 15:16:47', '2', 'Tuesday'),
 (800.0, '2019-02-24 14:07:11', '0', 'Sunday'),
 (500.0, '2019-10-14 11:29:21', '1', 'Monday'),
 (400.0, '2019-03-21 01:19:52', '4', 'Thursday'),
 (300.0, '2019-07-23 22:15:03', '2', 'Tuesday'),
 (300.0, '2019-11-01 13:07:36', '5', 'Friday'),
 (200.0, '2019-01-02 12:20:07', '3', 'Wednesday'),
 (200.0, '2019-01-10 16:08:23', '4', 'Thursday'),
 (200.0, '2019-01-16 07:53:34', '3', 'Wednesday'),
 (200.0, '2019-01-24 04:03:12', '4', 'Thursday'),
 (200.0, '2019-02-19 19:47:46', '2', 'Tuesday'),
 (200.0, '2019-03-16 11:09:44', '6', 'Saturday'),
 (200.0, '2019-03-30 19:14:05', '6', 'Saturday'),
 (200.0, '2019-04-12 14:18:40', '5', 'Friday'),
 (200.0, '2019-04-14 06:54:13', '0', 'Sunday'),
 (200.0, '2019-04-16 11:54:06', '2', 'Tuesday'),
 (200.0, '2019-04-17 06:00:47', '3', 'Wednesday'),
 (200.0, '2019-06-16 22:48:50', '0', 'Sunday'),
 (200.0, '2019-07-15 04:11:34', '1', 'Monday'),
 (200.0, '2019-08-04 04:19:11', '0', 'Sunday'),
 (200.0, '2019-10-06 22:13:19', '0', 'Sunday'),
 (200.0, '2019-10-07 18:35:34', '1', 'Monday'),
 (200.0, '2019-10-23 23:25:35', '3', 'Wednesday'),
 (200.0, '2019-10-31 13:27:30', '4', 'Thursday'),
 (200.0, '2019-11-08 21:15:03', '5', 'Friday'),
 (100.0, '2019-01-13 17:44:56', '0', 'Sunday'),
 (100.0, '2019-01-17 00:28:21', '4', 'Thursday'),
 (100.0, '2019-01-24 21:31:31', '4', 'Thursday'),
 (100.0, '2019-01-28 14:00:54', '1', 'Monday'),
 (100.0, '2019-01-28 18:22:58', '1', 'Monday'),
 (100.0, '2019-01-31 17:17:02', '4', 'Thursday'),
 (100.0, '2019-02-04 11:32:44', '1', 'Monday'),
 (100.0, '2019-02-04 12:25:58', '1', 'Monday'),
 (100.0, '2019-02-06 11:04:20', '3', 'Wednesday'),
 (100.0, '2019-02-19 13:30:14', '2', 'Tuesday'),
 (100.0, '2019-02-22 19:30:28', '5', 'Friday'),
 (100.0, '2019-02-25 16:02:33', '1', 'Monday'),
 (100.0, '2019-02-27 15:37:16', '3', 'Wednesday'),
 (100.0, '2019-03-01 03:54:01', '5', 'Friday'),
 (100.0, '2019-03-01 14:30:28', '5', 'Friday'),
 (100.0, '2019-03-02 10:18:43', '6', 'Saturday'),
```

```
(100.0, '2019-03-04 12:30:04', '1', 'Monday'),
(100.0, '2019-03-06 19:35:49', '3', 'Wednesday'),
(100.0, '2019-03-09 03:39:30', '6', 'Saturday'),
(100.0, '2019-03-11 09:55:23', '1', 'Monday'),
(100.0, '2019-03-24 02:20:45', '0', 'Sunday'),
(100.0, '2019-03-26 15:57:35', '2', 'Tuesday'),
(100.0, '2019-04-07 18:26:02', '0', 'Sunday'),
(100.0, '2019-04-10 15:14:21', '3', 'Wednesday'),
(100.0, '2019-04-17 19:43:02', '3', 'Wednesday'),
(100.0, '2019-05-10 10:26:12', '5', 'Friday'),
(100.0, '2019-06-06 09:19:47', '4', 'Thursday'),
(100.0, '2019-06-10 07:44:32', '1', 'Monday'),
(100.0, '2019-06-18 16:09:18', '2', 'Tuesday'),
(100.0, '2019-07-02 20:22:07', '2', 'Tuesday'),
(100.0, '2019-07-03 12:38:14', '3', 'Wednesday'),
(100.0, '2019-07-05 20:23:40', '5', 'Friday'),
(100.0, '2019-07-06 20:21:34', '6', 'Saturday'),
(100.0, '2019-07-20 20:08:58', '6', 'Saturday'),
(100.0, '2019-07-23 15:13:12', '2', 'Tuesday'),
(100.0, '2019-07-28 23:30:04', '0', 'Sunday'),
(100.0, '2019-07-30 01:06:19', '2', 'Tuesday'),
(100.0, '2019-08-08 06:01:46', '4', 'Thursday'),
(100.0, '2019-08-11 09:47:38', '0', 'Sunday'),
(100.0, '2019-08-26 14:51:24', '1', 'Monday'),
(100.0, '2019-09-02 21:43:22', '1', 'Monday'),
(100.0, '2019-09-11 22:54:06', '3', 'Wednesday'),
(100.0, '2019-09-12 04:08:37', '4', 'Thursday'),
(100.0, '2019-09-16 06:07:17', '1', 'Monday'),
(100.0, '2019-10-17 08:48:23', '4', 'Thursday'),
(100.0, '2019-10-27 03:55:40', '0', 'Sunday'),
(100.0, '2019-11-04 00:22:31', '1', 'Monday'),
(100.0, '2019-11-20 17:34:08', '3', 'Wednesday'),
(100.0, '2019-11-21 00:50:24', '4', 'Thursday'),
(0.0, '2019-08-24 09:39:31', '6', 'Saturday'),
(0.0, '2019-08-24 09:47:42', '6', 'Saturday'),
(0.0, '2019-08-24 09:48:26', '6', 'Saturday'),
(0.0, '2019-08-24 09:57:01', '6', 'Saturday'),
(0.0, '2019-08-24 10:44:52', '6', 'Saturday'),
(0.0, '2019-08-24 10:49:58', '6', 'Saturday'),
(0.0, '2019-08-24 10:50:23', '6', 'Saturday'),
(0.0, '2019-08-24 10:53:39', '6', 'Saturday'),
(0.0, '2019-08-24 10:55:22', '6', 'Saturday'),
(0.0, '2019-08-24 11:03:09', '6', 'Saturday'),
(0.0, '2019-08-24 11:15:47', '6', 'Saturday'),
(0.0, '2019-08-24 11:20:19', '6', 'Saturday'),
(0.0, '2019-08-24 11:38:54', '6', 'Saturday'),
(0.0, '2019-08-24 12:20:03', '6', 'Saturday'),
```

```
(0.0, '2019-08-24 12:27:47', '6', 'Saturday'),
(0.0, '2019-08-24 13:07:20', '6', 'Saturday'),
(0.0, '2019-08-24 13:09:33', '6', 'Saturday'),
(0.0, '2019-08-24 13:18:31', '6', 'Saturday'),
(0.0, '2019-08-24 13:26:29', '6', 'Saturday'),
(0.0, '2019-08-24 13:29:07', '6', 'Saturday'),
(0.0, '2019-08-24 14:33:06', '6', 'Saturday'),
(0.0, '2019-08-24 14:39:09', '6', 'Saturday'),
(0.0, '2019-08-24 14:44:31', '6', 'Saturday'),
(0.0, '2019-08-24 14:46:29', '6', 'Saturday'),
(0.0, '2019-08-24 14:57:03', '6', 'Saturday'),
(0.0, '2019-08-24 15:22:08', '6', 'Saturday'),
(0.0, '2019-08-24 15:32:38', '6', 'Saturday'),
(0.0, '2019-08-24 15:34:07', '6', 'Saturday'),
(0.0, '2019-08-24 16:00:46', '6', 'Saturday'),
(0.0, '2019-08-24 16:08:06', '6', 'Saturday'),
(0.0, '2019-08-24 16:09:40', '6', 'Saturday'),
(0.0, '2019-08-24 16:24:52', '6', 'Saturday'),
(0.0, '2019-08-24 16:32:56', '6', 'Saturday'),
(0.0, '2019-08-24 16:42:45', '6', 'Saturday'),
(0.0, '2019-08-24 16:55:56', '6', 'Saturday'),
(0.0, '2019-08-24 17:19:09', '6', 'Saturday'),
(0.0, '2019-08-24 17:19:37', '6', 'Saturday'),
(0.0, '2019-08-24 17:27:57', '6', 'Saturday'),
(0.0, '2019-08-24 17:32:49', '6', 'Saturday'),
(0.0, '2019-08-24 17:44:56', '6', 'Saturday'),
(0.0, '2019-08-24 18:25:21', '6', 'Saturday'),
(0.0, '2019-08-24 18:28:55', '6', 'Saturday'),
(0.0, '2019-08-24 18:31:02', '6', 'Saturday'),
(0.0, '2019-08-24 18:53:30', '6', 'Saturday'),
(0.0, '2019-08-24 18:56:59', '6', 'Saturday'),
(0.0, '2019-08-24 19:39:30', '6', 'Saturday'),
(0.0, '2019-08-24 19:51:37', '6', 'Saturday'),
(0.0, '2019-08-24 19:58:36', '6', 'Saturday'),
(0.0, '2019-08-24 20:10:31', '6', 'Saturday'),
(0.0, '2019-08-24 20:31:23', '6', 'Saturday'),
(0.0, '2019-08-24 20:55:32', '6', 'Saturday'),
(0.0, '2019-08-24 21:28:51', '6', 'Saturday'),
(0.0, '2019-08-24 21:30:48', '6', 'Saturday'),
(0.0, '2019-08-24 21:47:51', '6', 'Saturday'),
(0.0, '2019-08-24 21:49:39', '6', 'Saturday'),
(0.0, '2019-08-24 21:59:33', '6', 'Saturday'),
(0.0, '2019-08-24 22:21:45', '6', 'Saturday'),
(0.0, '2019-08-24 22:24:40', '6', 'Saturday'),
(0.0, '2019-08-24 22:34:13', '6', 'Saturday'),
(0.0, '2019-08-24 22:47:26', '6', 'Saturday'),
(0.0, '2019-08-24 22:47:27', '6', 'Saturday'),
```

```
(0.0, '2019-01-01 00:16:33', '2', 'Tuesday'),
(0.0, '2019-01-01 00:17:09', '2', 'Tuesday'),
(0.0, '2019-01-01 01:29:22', '2', 'Tuesday'),
(0.0, '2019-01-01 01:30:22', '2', 'Tuesday'),
(0.0, '2019-01-01 02:50:26', '2', 'Tuesday'),
(0.0, '2019-01-01 03:27:55', '2', 'Tuesday'),
(0.0, '2019-01-01 03:37:15', '2', 'Tuesday'),
(0.0, '2019-01-01 03:51:05', '2', 'Tuesday'),
(0.0, '2019-01-01 04:30:50', '2', 'Tuesday'),
(0.0, '2019-01-01 05:31:15', '2', 'Tuesday'),
(0.0, '2019-01-01 06:32:41', '2', 'Tuesday'),
(0.0, '2019-01-01 06:36:37', '2', 'Tuesday'),
(0.0, '2019-01-01 07:06:40', '2', 'Tuesday'),
(0.0, '2019-01-01 08:48:35', '2', 'Tuesday'),
(0.0, '2019-01-01 09:05:27', '2', 'Tuesday'),
(0.0, '2019-01-01 09:26:16', '2', 'Tuesday'),
(0.0, '2019-01-01 10:31:57', '2', 'Tuesday'),
(0.0, '2019-01-01 11:14:50', '2', 'Tuesday'),
(0.0, '2019-01-01 11:46:18', '2', 'Tuesday'),
(0.0, '2019-01-01 12:07:28', '2', 'Tuesday'),
(0.0, '2019-01-01 13:26:21', '2', 'Tuesday'),
(0.0, '2019-01-01 13:45:16', '2', 'Tuesday'),
(0.0, '2019-01-01 13:51:31', '2', 'Tuesday'),
(0.0, '2019-01-01 14:01:32', '2', 'Tuesday'),
(0.0, '2019-01-01 14:01:44', '2', 'Tuesday'),
(0.0, '2019-01-01 14:07:32', '2', 'Tuesday'),
(0.0, '2019-01-01 14:30:56', '2', 'Tuesday'),
(0.0, '2019-01-01 14:33:02', '2', 'Tuesday'),
(0.0, '2019-01-01 14:44:41', '2', 'Tuesday'),
(0.0, '2019-01-01 15:14:20', '2', 'Tuesday'),
(0.0, '2019-01-01 15:35:55', '2', 'Tuesday'),
(0.0, '2019-01-01 15:37:53', '2', 'Tuesday'),
(0.0, '2019-01-01 16:13:32', '2', 'Tuesday'),
(0.0, '2019-01-01 16:14:16', '2', 'Tuesday'),
(0.0, '2019-01-01 16:48:30', '2', 'Tuesday'),
(0.0, '2019-01-01 16:51:11', '2', 'Tuesday'),
(0.0, '2019-01-01 17:25:36', '2', 'Tuesday'),
(0.0, '2019-01-01 17:28:02', '2', 'Tuesday'),
(0.0, '2019-01-01 17:48:36', '2', 'Tuesday'),
(0.0, '2019-01-01 19:24:25', '2', 'Tuesday'),
(0.0, '2019-01-01 19:43:47', '2', 'Tuesday'),
(0.0, '2019-01-01 20:04:22', '2', 'Tuesday'),
(0.0, '2019-01-01 20:27:16', '2', 'Tuesday'),
(0.0, '2019-01-01 20:32:11', '2', 'Tuesday'),
(0.0, '2019-01-01 20:42:20', '2', 'Tuesday'),
(0.0, '2019-01-01 21:32:06', '2', 'Tuesday'),
(0.0, '2019-01-01 21:42:00', '2', 'Tuesday'),
```

```
(0.0, '2019-01-01 21:49:38', '2', 'Tuesday'),
(0.0, '2019-01-01 21:50:32', '2', 'Tuesday'),
(0.0, '2019-01-01 21:59:58', '2', 'Tuesday'),
(0.0, '2019-01-01 22:02:28', '2', 'Tuesday'),
(0.0, '2019-01-01 22:07:42', '2', 'Tuesday'),
(0.0, '2019-01-01 22:23:06', '2', 'Tuesday'),
(0.0, '2019-01-01 22:41:52', '2', 'Tuesday'),
(0.0, '2019-01-01 22:53:29', '2', 'Tuesday'),
(0.0, '2019-01-01 23:03:48', '2', 'Tuesday'),
(0.0, '2019-01-01 23:33:22', '2', 'Tuesday'),
(0.0, '2019-01-02 00:14:24', '3', 'Wednesday'),
(0.0, '2019-01-02 00:14:53', '3', 'Wednesday'),
(0.0, '2019-01-02 00:31:06', '3', 'Wednesday'),
(0.0, '2019-01-02 00:45:51', '3', 'Wednesday'),
(0.0, '2019-01-02 00:50:00', '3', 'Wednesday'),
(0.0, '2019-01-02 00:55:22', '3', 'Wednesday'),
(0.0, '2019-01-02 00:58:56', '3', 'Wednesday'),
(0.0, '2019-01-02 01:05:12', '3', 'Wednesday'),
(0.0, '2019-01-02 01:33:36', '3', 'Wednesday'),
(0.0, '2019-01-02 01:48:06', '3', 'Wednesday'),
(0.0, '2019-01-02 02:18:12', '3', 'Wednesday'),
(0.0, '2019-01-02 02:34:37', '3', 'Wednesday'),
(0.0, '2019-01-02 02:36:33', '3', 'Wednesday'),
(0.0, '2019-01-02 02:50:42', '3', 'Wednesday'),
(0.0, '2019-01-02 03:20:11', '3', 'Wednesday'),
(0.0, '2019-01-02 03:36:05', '3', 'Wednesday'),
(0.0, '2019-01-02 03:50:55', '3', 'Wednesday'),
(0.0, '2019-01-02 04:45:59', '3', 'Wednesday'),
(0.0, '2019-01-02 05:30:35', '3', 'Wednesday'),
(0.0, '2019-01-02 05:37:48', '3', 'Wednesday'),
(0.0, '2019-01-02 06:25:37', '3', 'Wednesday'),
(0.0, '2019-01-02 06:28:04', '3', 'Wednesday'),
(0.0, '2019-01-02 06:29:30', '3', 'Wednesday'),
(0.0, '2019-01-02 07:13:34', '3', 'Wednesday'),
(0.0, '2019-01-02 07:39:24', '3', 'Wednesday'),
(0.0, '2019-01-02 08:04:08', '3', 'Wednesday'),
(0.0, '2019-01-02 08:05:55', '3', 'Wednesday'),
(0.0, '2019-01-02 08:06:34', '3', 'Wednesday'),
(0.0, '2019-01-02 08:09:46', '3', 'Wednesday'),
(0.0, '2019-01-02 08:11:47', '3', 'Wednesday'),
(0.0, '2019-01-02 08:43:16', '3', 'Wednesday'),
(0.0, '2019-01-02 08:51:12', '3', 'Wednesday'),
(0.0, '2019-01-02 08:59:15', '3', 'Wednesday'),
(0.0, '2019-01-02 08:59:34', '3', 'Wednesday'),
(0.0, '2019-01-02 09:30:51', '3', 'Wednesday'),
(0.0, '2019-01-02 09:33:16', '3', 'Wednesday'),
(0.0, '2019-01-02 09:33:54', '3', 'Wednesday'),
```

```
(0.0, '2019-01-02 09:47:45', '3', 'Wednesday'),
(0.0, '2019-01-02 10:09:05', '3', 'Wednesday'),
(0.0, '2019-01-02 10:11:44', '3', 'Wednesday'),
(0.0, '2019-01-02 10:14:46', '3', 'Wednesday'),
(0.0, '2019-01-02 10:19:41', '3', 'Wednesday'),
(0.0, '2019-01-02 10:29:50', '3', 'Wednesday'),
(0.0, '2019-01-02 10:34:20', '3', 'Wednesday'),
(0.0, '2019-01-02 10:36:22', '3', 'Wednesday'),
(0.0, '2019-01-02 11:15:34', '3', 'Wednesday'),
(0.0, '2019-01-02 11:48:06', '3', 'Wednesday'),
(0.0, '2019-01-02 11:48:35', '3', 'Wednesday'),
(0.0, '2019-01-02 11:51:00', '3', 'Wednesday'),
(0.0, '2019-01-02 11:55:26', '3', 'Wednesday'),
(0.0, '2019-01-02 12:14:31', '3', 'Wednesday'),
(0.0, '2019-01-02 12:18:18', '3', 'Wednesday'),
(0.0, '2019-01-02 12:35:10', '3', 'Wednesday'),
(0.0, '2019-01-02 13:05:39', '3', 'Wednesday'),
(0.0, '2019-01-02 13:08:49', '3', 'Wednesday'),
(0.0, '2019-01-02 13:10:50', '3', 'Wednesday'),
(0.0, '2019-01-02 13:15:02', '3', 'Wednesday'),
(0.0, '2019-01-02 13:26:37', '3', 'Wednesday'),
(0.0, '2019-01-02 13:33:12', '3', 'Wednesday'),
(0.0, '2019-01-02 13:35:39', '3', 'Wednesday'),
(0.0, '2019-01-02 13:39:57', '3', 'Wednesday'),
(0.0, '2019-01-02 13:48:32', '3', 'Wednesday'),
(0.0, '2019-01-02 13:50:39', '3', 'Wednesday'),
(0.0, '2019-01-02 14:01:01', '3', 'Wednesday'),
(0.0, '2019-01-02 14:14:59', '3', 'Wednesday'),
(0.0, '2019-01-02 14:21:49', '3', 'Wednesday'),
(0.0, '2019-01-02 14:29:02', '3', 'Wednesday'),
(0.0, '2019-01-02 14:30:14', '3', 'Wednesday'),
(0.0, '2019-01-02 14:31:44', '3', 'Wednesday'),
(0.0, '2019-01-02 14:41:48', '3', 'Wednesday'),
(0.0, '2019-01-02 14:43:13', '3', 'Wednesday'),
(0.0, '2019-01-02 14:47:31', '3', 'Wednesday'),
(0.0, '2019-01-02 15:08:16', '3', 'Wednesday'),
(0.0, '2019-01-02 15:11:53', '3', 'Wednesday'),
(0.0, '2019-01-02 15:31:35', '3', 'Wednesday'),
(0.0, '2019-01-02 15:45:36', '3', 'Wednesday'),
(0.0, '2019-01-02 15:59:02', '3', 'Wednesday'),
(0.0, '2019-01-02 16:00:36', '3', 'Wednesday'),
(0.0, '2019-01-02 16:06:11', '3', 'Wednesday'),
(0.0, '2019-01-02 16:08:37', '3', 'Wednesday'),
(0.0, '2019-01-02 16:09:54', '3', 'Wednesday'),
(0.0, '2019-01-02 16:12:43', '3', 'Wednesday'),
(0.0, '2019-01-02 16:14:17', '3', 'Wednesday'),
(0.0, '2019-01-02 16:16:05', '3', 'Wednesday'),
```

```
(0.0, '2019-01-02 16:24:02', '3', 'Wednesday'),
(0.0, '2019-01-02 16:28:36', '3', 'Wednesday'),
(0.0, '2019-01-02 16:47:17', '3', 'Wednesday'),
(0.0, '2019-01-02 17:11:34', '3', 'Wednesday'),
(0.0, '2019-01-02 17:29:35', '3', 'Wednesday'),
(0.0, '2019-01-02 17:35:06', '3', 'Wednesday'),
(0.0, '2019-01-02 17:36:48', '3', 'Wednesday'),
(0.0, '2019-01-02 17:47:07', '3', 'Wednesday'),
(0.0, '2019-01-02 18:01:42', '3', 'Wednesday'),
(0.0, '2019-01-02 18:09:35', '3', 'Wednesday'),
(0.0, '2019-01-02 18:11:22', '3', 'Wednesday'),
(0.0, '2019-01-02 18:28:23', '3', 'Wednesday'),
(0.0, '2019-01-02 18:28:35', '3', 'Wednesday'),
(0.0, '2019-01-02 18:29:31', '3', 'Wednesday'),
(0.0, '2019-01-02 14:44:15', '3', 'Wednesday'),
(0.0, '2019-01-02 19:31:00', '3', 'Wednesday'),
(0.0, '2019-01-02 19:43:27', '3', 'Wednesday'),
(0.0, '2019-01-02 20:20:47', '3', 'Wednesday'),
(0.0, '2019-01-02 20:23:15', '3', 'Wednesday'),
(0.0, '2019-01-02 20:27:13', '3', 'Wednesday'),
(0.0, '2019-01-02 20:34:13', '3', 'Wednesday'),
(0.0, '2019-01-02 20:44:11', '3', 'Wednesday'),
(0.0, '2019-01-02 20:46:29', '3', 'Wednesday'),
(0.0, '2019-01-02 21:03:25', '3', 'Wednesday'),
(0.0, '2019-01-02 21:04:38', '3', 'Wednesday'),
(0.0, '2019-01-02 21:21:40', '3', 'Wednesday'),
(0.0, '2019-01-02 21:21:55', '3', 'Wednesday'),
(0.0, '2019-01-02 21:23:21', '3', 'Wednesday'),
(0.0, '2019-01-02 21:45:04', '3', 'Wednesday'),
(0.0, '2019-01-02 21:46:51', '3', 'Wednesday'),
(0.0, '2019-01-02 21:50:50', '3', 'Wednesday'),
(0.0, '2019-01-02 21:56:36', '3', 'Wednesday'),
(0.0, '2019-01-02 22:05:31', '3', 'Wednesday'),
(0.0, '2019-01-02 22:23:37', '3', 'Wednesday'),
(0.0, '2019-01-02 22:35:46', '3', 'Wednesday'),
(0.0, '2019-01-02 22:45:37', '3', 'Wednesday'),
(0.0, '2019-01-02 22:46:04', '3', 'Wednesday'),
(0.0, '2019-01-02 22:49:31', '3', 'Wednesday'),
(0.0, '2019-01-02 22:50:05', '3', 'Wednesday'),
(0.0, '2019-01-02 22:58:12', '3', 'Wednesday'),
(0.0, '2019-01-02 23:25:44', '3', 'Wednesday'),
(0.0, '2019-01-02 23:36:57', '3', 'Wednesday'),
(0.0, '2019-01-02 23:37:23', '3', 'Wednesday'),
(0.0, '2019-01-02 23:46:16', '3', 'Wednesday'),
(0.0, '2019-01-02 18:01:31', '3', 'Wednesday'),
(0.0, '2019-01-02 19:32:41', '3', 'Wednesday'),
(0.0, '2019-01-03 00:09:35', '4', 'Thursday'),
```

```
(0.0, '2019-01-03 00:17:16', '4', 'Thursday'),
(0.0, '2019-01-03 00:26:31', '4', 'Thursday'),
(0.0, '2019-01-03 00:34:32', '4', 'Thursday'),
(0.0, '2019-01-03 00:47:41', '4', 'Thursday'),
(0.0, '2019-01-03 01:24:30', '4', 'Thursday'),
(0.0, '2019-01-03 01:48:42', '4', 'Thursday'),
(0.0, '2019-01-03 02:05:15', '4', 'Thursday'),
(0.0, '2019-01-03 02:19:03', '4', 'Thursday'),
(0.0, '2019-01-03 02:30:00', '4', 'Thursday'),
(0.0, '2019-01-03 02:35:58', '4', 'Thursday'),
(0.0, '2019-01-03 02:50:37', '4', 'Thursday'),
(0.0, '2019-01-03 02:56:21', '4', 'Thursday'),
(0.0, '2019-01-03 03:05:35', '4', 'Thursday'),
(0.0, '2019-01-03 03:23:57', '4', 'Thursday'),
(0.0, '2019-01-03 04:49:35', '4', 'Thursday'),
(0.0, '2019-01-03 05:48:22', '4', 'Thursday'),
(0.0, '2019-01-03 06:24:21', '4', 'Thursday'),
(0.0, '2019-01-03 06:32:58', '4', 'Thursday'),
(0.0, '2019-01-03 07:08:13', '4', 'Thursday'),
(0.0, '2019-01-03 07:19:52', '4', 'Thursday'),
(0.0, '2019-01-03 07:32:50', '4', 'Thursday'),
(0.0, '2019-01-03 08:07:08', '4', 'Thursday'),
(0.0, '2019-01-03 08:26:46', '4', 'Thursday'),
(0.0, '2019-01-03 08:44:24', '4', 'Thursday'),
(0.0, '2019-01-03 09:05:25', '4', 'Thursday'),
(0.0, '2019-01-03 09:29:15', '4', 'Thursday'),
(0.0, '2019-01-03 09:36:55', '4', 'Thursday'),
(0.0, '2019-01-03 09:40:37', '4', 'Thursday'),
(0.0, '2019-01-03 09:45:31', '4', 'Thursday'),
(0.0, '2019-01-03 10:13:42', '4', 'Thursday'),
(0.0, '2019-01-03 10:13:56', '4', 'Thursday'),
(0.0, '2019-01-03 10:29:21', '4', 'Thursday'),
(0.0, '2019-01-03 10:36:27', '4', 'Thursday'),
(0.0, '2019-01-03 10:39:40', '4', 'Thursday'),
(0.0, '2019-01-03 11:05:27', '4', 'Thursday'),
(0.0, '2019-01-03 11:18:21', '4', 'Thursday'),
(0.0, '2019-01-03 11:27:04', '4', 'Thursday'),
(0.0, '2019-01-03 11:30:13', '4', 'Thursday'),
(0.0, '2019-01-03 11:42:22', '4', 'Thursday'),
(0.0, '2019-01-03 11:48:59', '4', 'Thursday'),
(0.0, '2019-01-03 11:52:46', '4', 'Thursday'),
(0.0, '2019-01-03 11:53:06', '4', 'Thursday'),
(0.0, '2019-01-03 12:14:56', '4', 'Thursday'),
(0.0, '2019-01-03 12:20:38', '4', 'Thursday'),
(0.0, '2019-01-03 12:25:56', '4', 'Thursday'),
(0.0, '2019-01-03 12:45:15', '4', 'Thursday'),
(0.0, '2019-01-03 12:47:47', '4', 'Thursday'),
```

```
(0.0, '2019-01-03 12:49:50', '4', 'Thursday'),
(0.0, '2019-01-03 12:57:42', '4', 'Thursday'),
(0.0, '2019-01-03 12:58:24', '4', 'Thursday'),
(0.0, '2019-01-03 13:01:49', '4', 'Thursday'),
(0.0, '2019-01-03 13:10:11', '4', 'Thursday'),
(0.0, '2019-01-03 13:10:54', '4', 'Thursday'),
(0.0, '2019-01-03 13:13:28', '4', 'Thursday'),
(0.0, '2019-01-03 13:23:36', '4', 'Thursday'),
(0.0, '2019-01-03 13:36:19', '4', 'Thursday'),
(0.0, '2019-01-03 13:48:03', '4', 'Thursday'),
(0.0, '2019-01-03 13:49:23', '4', 'Thursday'),
(0.0, '2019-01-03 14:01:03', '4', 'Thursday'),
(0.0, '2019-01-03 14:20:35', '4', 'Thursday'),
(0.0, '2019-01-03 14:23:30', '4', 'Thursday'),
(0.0, '2019-01-03 14:26:46', '4', 'Thursday'),
(0.0, '2019-01-03 14:32:40', '4', 'Thursday'),
(0.0, '2019-01-03 14:34:43', '4', 'Thursday'),
(0.0, '2019-01-03 15:34:40', '4', 'Thursday'),
(0.0, '2019-01-03 15:37:31', '4', 'Thursday'),
(0.0, '2019-01-03 15:38:24', '4', 'Thursday'),
(0.0, '2019-01-03 15:40:14', '4', 'Thursday'),
(0.0, '2019-01-03 15:44:24', '4', 'Thursday'),
(0.0, '2019-01-03 15:47:05', '4', 'Thursday'),
(0.0, '2019-01-03 15:47:51', '4', 'Thursday'),
(0.0, '2019-01-03 16:00:46', '4', 'Thursday'),
(0.0, '2019-01-03 16:14:31', '4', 'Thursday'),
(0.0, '2019-01-03 16:26:14', '4', 'Thursday'),
(0.0, '2019-01-03 16:28:00', '4', 'Thursday'),
(0.0, '2019-01-03 16:31:53', '4', 'Thursday'),
(0.0, '2019-01-03 16:32:21', '4', 'Thursday'),
(0.0, '2019-01-03 16:49:10', '4', 'Thursday'),
(0.0, '2019-01-03 16:57:21', '4', 'Thursday'),
(0.0, '2019-01-03 17:21:59', '4', 'Thursday'),
(0.0, '2019-01-03 17:27:06', '4', 'Thursday'),
(0.0, '2019-01-03 17:31:07', '4', 'Thursday'),
(0.0, '2019-01-03 17:32:16', '4', 'Thursday'),
(0.0, '2019-01-03 17:34:49', '4', 'Thursday'),
(0.0, '2019-01-03 17:34:51', '4', 'Thursday'),
(0.0, '2019-01-03 17:36:22', '4', 'Thursday'),
(0.0, '2019-01-03 18:21:36', '4', 'Thursday'),
(0.0, '2019-01-03 18:24:31', '4', 'Thursday'),
(0.0, '2019-01-03 18:32:39', '4', 'Thursday'),
(0.0, '2019-01-03 18:48:03', '4', 'Thursday'),
(0.0, '2019-01-03 18:49:38', '4', 'Thursday'),
(0.0, '2019-01-03 18:58:33', '4', 'Thursday'),
(0.0, '2019-01-03 19:10:02', '4', 'Thursday'),
(0.0, '2019-01-03 19:15:52', '4', 'Thursday'),
```

```
(0.0, '2019-01-03 19:50:59', '4', 'Thursday'),
(0.0, '2019-01-03 19:52:40', '4', 'Thursday'),
(0.0, '2019-01-03 20:14:05', '4', 'Thursday'),
(0.0, '2019-01-03 20:16:08', '4', 'Thursday'),
(0.0, '2019-01-03 20:55:50', '4', 'Thursday'),
(0.0, '2019-01-03 20:57:16', '4', 'Thursday'),
(0.0, '2019-01-03 21:17:43', '4', 'Thursday'),
(0.0, '2019-01-03 21:26:39', '4', 'Thursday'),
(0.0, '2019-01-03 21:32:43', '4', 'Thursday'),
(0.0, '2019-01-03 21:32:55', '4', 'Thursday'),
(0.0, '2019-01-03 21:44:13', '4', 'Thursday'),
(0.0, '2019-01-03 22:02:18', '4', 'Thursday'),
(0.0, '2019-01-03 22:04:03', '4', 'Thursday'),
(0.0, '2019-01-03 22:04:26', '4', 'Thursday'),
(0.0, '2019-01-03 22:20:13', '4', 'Thursday'),
(0.0, '2019-01-03 22:21:30', '4', 'Thursday'),
(0.0, '2019-01-03 22:23:01', '4', 'Thursday'),
(0.0, '2019-01-03 22:24:25', '4', 'Thursday'),
(0.0, '2019-01-03 22:36:30', '4', 'Thursday'),
(0.0, '2019-01-03 23:09:12', '4', 'Thursday'),
(0.0, '2019-01-03 23:30:31', '4', 'Thursday'),
(0.0, '2019-01-03 23:56:37', '4', 'Thursday'),
(0.0, '2019-01-04 00:01:15', '5', 'Friday'),
(0.0, '2019-01-04 00:25:43', '5', 'Friday'),
(0.0, '2019-01-04 00:32:24', '5', 'Friday'),
(0.0, '2019-01-04 00:49:23', '5', 'Friday'),
(0.0, '2019-01-04 00:55:09', '5', 'Friday'),
(0.0, '2019-01-04 00:57:08', '5', 'Friday'),
(0.0, '2019-01-04 01:39:34', '5', 'Friday'),
(0.0, '2019-01-04 01:46:59', '5', 'Friday'),
(0.0, '2019-01-04 01:50:08', '5', 'Friday'),
(0.0, '2019-01-04 01:52:52', '5', 'Friday'),
(0.0, '2019-01-04 01:57:11', '5', 'Friday'),
(0.0, '2019-01-04 01:58:36', '5', 'Friday'),
(0.0, '2019-01-04 02:35:10', '5', 'Friday'),
(0.0, '2019-01-04 02:58:36', '5', 'Friday'),
(0.0, '2019-01-04 03:10:50', '5', 'Friday'),
(0.0, '2019-01-04 03:22:40', '5', 'Friday'),
(0.0, '2019-01-04 03:32:33', '5', 'Friday'),
(0.0, '2019-01-04 03:37:46', '5', 'Friday'),
(0.0, '2019-01-04 04:26:41', '5', 'Friday'),
(0.0, '2019-01-04 04:31:59', '5', 'Friday'),
(0.0, '2019-01-04 04:33:16', '5', 'Friday'),
(0.0, '2019-01-04 04:51:45', '5', 'Friday'),
(0.0, '2019-01-04 05:23:31', '5', 'Friday'),
(0.0, '2019-01-04 05:27:40', '5', 'Friday'),
(0.0, '2019-01-04 05:37:21', '5', 'Friday'),
```

```
(0.0, '2019-01-04 05:52:31', '5', 'Friday'),
(0.0, '2019-01-04 06:04:20', '5', 'Friday'),
(0.0, '2019-01-04 06:05:34', '5', 'Friday'),
(0.0, '2019-01-04 06:25:34', '5', 'Friday'),
(0.0, '2019-01-04 06:34:33', '5', 'Friday'),
(0.0, '2019-01-04 06:35:31', '5', 'Friday'),
(0.0, '2019-01-04 06:42:03', '5', 'Friday'),
(0.0, '2019-01-04 06:51:30', '5', 'Friday'),
(0.0, '2019-01-04 06:53:00', '5', 'Friday'),
(0.0, '2019-01-04 07:01:23', '5', 'Friday'),
(0.0, '2019-01-04 07:45:18', '5', 'Friday'),
(0.0, '2019-01-04 08:22:54', '5', 'Friday'),
(0.0, '2019-01-04 08:27:24', '5', 'Friday'),
(0.0, '2019-01-04 08:34:33', '5', 'Friday'),
(0.0, '2019-01-04 08:35:32', '5', 'Friday'),
(0.0, '2019-01-04 08:47:04', '5', 'Friday'),
(0.0, '2019-01-04 08:54:25', '5', 'Friday'),
(0.0, '2019-01-04 08:59:04', '5', 'Friday'),
(0.0, '2019-01-04 09:03:02', '5', 'Friday'),
(0.0, '2019-01-04 09:08:45', '5', 'Friday'),
(0.0, '2019-01-04 09:09:17', '5', 'Friday'),
(0.0, '2019-01-04 09:10:38', '5', 'Friday'),
(0.0, '2019-01-04 09:15:01', '5', 'Friday'),
(0.0, '2019-01-04 09:22:33', '5', 'Friday'),
(0.0, '2019-01-04 09:27:27', '5', 'Friday'),
(0.0, '2019-01-04 10:02:09', '5', 'Friday'),
(0.0, '2019-01-04 10:08:40', '5', 'Friday'),
(0.0, '2019-01-04 10:11:24', '5', 'Friday'),
(0.0, '2019-01-04 10:42:43', '5', 'Friday'),
(0.0, '2019-01-04 10:59:40', '5', 'Friday'),
(0.0, '2019-01-04 11:13:46', '5', 'Friday'),
(0.0, '2019-01-04 11:17:16', '5', 'Friday'),
(0.0, '2019-01-04 11:17:25', '5', 'Friday'),
(0.0, '2019-01-04 11:28:04', '5', 'Friday'),
(0.0, '2019-01-04 11:38:00', '5', 'Friday'),
(0.0, '2019-01-04 11:43:50', '5', 'Friday'),
(0.0, '2019-01-04 11:45:57', '5', 'Friday'),
(0.0, '2019-01-04 11:56:35', '5', 'Friday'),
(0.0, '2019-01-04 11:57:02', '5', 'Friday'),
(0.0, '2019-01-04 12:03:41', '5', 'Friday'),
(0.0, '2019-01-04 12:06:06', '5', 'Friday'),
(0.0, '2019-01-04 12:36:26', '5', 'Friday'),
(0.0, '2019-01-04 12:46:02', '5', 'Friday'),
(0.0, '2019-01-04 12:53:53', '5', 'Friday'),
(0.0, '2019-01-04 13:31:50', '5', 'Friday'),
(0.0, '2019-01-04 13:49:21', '5', 'Friday'),
(0.0, '2019-01-04 14:01:47', '5', 'Friday'),
```

```
(0.0, '2019-01-04 14:01:48', '5', 'Friday'),
(0.0, '2019-01-04 14:17:02', '5', 'Friday'),
(0.0, '2019-01-04 14:35:54', '5', 'Friday'),
(0.0, '2019-01-04 14:42:20', '5', 'Friday'),
(0.0, '2019-01-04 14:43:09', '5', 'Friday'),
(0.0, '2019-01-04 14:43:55', '5', 'Friday'),
(0.0, '2019-01-04 14:53:17', '5', 'Friday'),
(0.0, '2019-01-04 14:54:51', '5', 'Friday'),
(0.0, '2019-01-04 15:05:24', '5', 'Friday'),
(0.0, '2019-01-04 15:14:37', '5', 'Friday'),
(0.0, '2019-01-04 15:26:10', '5', 'Friday'),
(0.0, '2019-01-04 15:36:32', '5', 'Friday'),
(0.0, '2019-01-04 15:45:39', '5', 'Friday'),
(0.0, '2019-01-04 14:31:08', '5', 'Friday'),
(0.0, '2019-01-04 15:49:25', '5', 'Friday'),
(0.0, '2019-01-04 15:53:16', '5', 'Friday'),
(0.0, '2019-01-04 15:53:47', '5', 'Friday'),
(0.0, '2019-01-04 15:56:11', '5', 'Friday'),
(0.0, '2019-01-04 16:05:13', '5', 'Friday'),
(0.0, '2019-01-04 16:11:36', '5', 'Friday'),
(0.0, '2019-01-04 16:40:21', '5', 'Friday'),
(0.0, '2019-01-04 16:40:37', '5', 'Friday'),
(0.0, '2019-01-04 16:53:31', '5', 'Friday'),
(0.0, '2019-01-04 16:55:38', '5', 'Friday'),
(0.0, '2019-01-04 16:59:30', '5', 'Friday'),
(0.0, '2019-01-04 17:37:18', '5', 'Friday'),
(0.0, '2019-01-04 17:46:10', '5', 'Friday'),
(0.0, '2019-01-04 17:52:16', '5', 'Friday'),
(0.0, '2019-01-04 18:00:59', '5', 'Friday'),
(0.0, '2019-01-04 18:09:52', '5', 'Friday'),
(0.0, '2019-01-04 18:14:56', '5', 'Friday'),
(0.0, '2019-01-04 18:24:32', '5', 'Friday'),
(0.0, '2019-01-04 18:43:38', '5', 'Friday'),
(0.0, '2019-01-04 18:45:26', '5', 'Friday'),
(0.0, '2019-01-04 18:58:36', '5', 'Friday'),
(0.0, '2019-01-04 19:07:18', '5', 'Friday'),
(0.0, '2019-01-04 19:17:17', '5', 'Friday'),
(0.0, '2019-01-04 19:19:22', '5', 'Friday'),
(0.0, '2019-01-04 19:20:45', '5', 'Friday'),
(0.0, '2019-01-04 19:22:49', '5', 'Friday'),
(0.0, '2019-01-04 19:24:17', '5', 'Friday'),
(0.0, '2019-01-04 19:27:55', '5', 'Friday'),
(0.0, '2019-01-04 19:31:33', '5', 'Friday'),
(0.0, '2019-01-04 19:59:59', '5', 'Friday'),
(0.0, '2019-01-04 20:19:42', '5', 'Friday'),
(0.0, '2019-01-04 20:35:45', '5', 'Friday'),
(0.0, '2019-01-04 20:58:24', '5', 'Friday'),
```

```
(0.0, '2019-01-04 21:04:28', '5', 'Friday'),
(0.0, '2019-01-04 22:00:31', '5', 'Friday'),
(0.0, '2019-01-04 22:15:53', '5', 'Friday'),
(0.0, '2019-01-04 22:32:28', '5', 'Friday'),
(0.0, '2019-01-04 22:48:44', '5', 'Friday'),
(0.0, '2019-01-04 22:56:30', '5', 'Friday'),
(0.0, '2019-01-04 23:15:41', '5', 'Friday'),
(0.0, '2019-01-04 23:21:57', '5', 'Friday'),
(0.0, '2019-01-04 23:46:43', '5', 'Friday'),
(0.0, '2019-01-04 23:50:27', '5', 'Friday'),
(0.0, '2019-01-05 00:08:18', '6', 'Saturday'),
(0.0, '2019-01-05 00:23:59', '6', 'Saturday'),
(0.0, '2019-01-05 00:43:55', '6', 'Saturday'),
(0.0, '2019-01-05 01:36:55', '6', 'Saturday'),
(0.0, '2019-01-05 03:15:40', '6', 'Saturday'),
(0.0, '2019-01-05 03:17:49', '6', 'Saturday'),
(0.0, '2019-01-05 04:16:31', '6', 'Saturday'),
(0.0, '2019-01-05 04:44:07', '6', 'Saturday'),
(0.0, '2019-01-05 06:15:05', '6', 'Saturday'),
(0.0, '2019-01-05 06:21:57', '6', 'Saturday'),
(0.0, '2019-01-03 23:02:49', '4', 'Thursday'),
(0.0, '2019-01-05 06:48:47', '6', 'Saturday'),
(0.0, '2019-01-05 06:51:07', '6', 'Saturday'),
(0.0, '2019-01-05 07:27:52', '6', 'Saturday'),
(0.0, '2019-01-05 07:31:03', '6', 'Saturday'),
(0.0, '2019-01-05 07:44:31', '6', 'Saturday'),
(0.0, '2019-01-05 08:18:16', '6', 'Saturday'),
(0.0, '2019-01-05 08:41:33', '6', 'Saturday'),
(0.0, '2019-01-05 09:39:42', '6', 'Saturday'),
(0.0, '2019-01-05 10:22:12', '6', 'Saturday'),
(0.0, '2019-01-05 10:22:19', '6', 'Saturday'),
(0.0, '2019-01-05 10:40:25', '6', 'Saturday'),
(0.0, '2019-01-05 11:07:35', '6', 'Saturday'),
(0.0, '2019-01-05 11:12:19', '6', 'Saturday'),
(0.0, '2019-01-05 11:32:33', '6', 'Saturday'),
(0.0, '2019-01-05 11:58:11', '6', 'Saturday'),
(0.0, '2019-01-05 12:07:13', '6', 'Saturday'),
(0.0, '2019-01-05 12:27:32', '6', 'Saturday'),
(0.0, '2019-01-05 12:46:58', '6', 'Saturday'),
(0.0, '2019-01-05 13:03:08', '6', 'Saturday'),
(0.0, '2019-01-05 13:24:09', '6', 'Saturday'),
(0.0, '2019-01-05 13:26:03', '6', 'Saturday'),
(0.0, '2019-01-05 13:28:19', '6', 'Saturday'),
(0.0, '2019-01-05 13:30:50', '6', 'Saturday'),
(0.0, '2019-01-05 13:40:59', '6', 'Saturday'),
(0.0, '2019-01-05 14:28:31', '6', 'Saturday'),
(0.0, '2019-01-05 14:35:52', '6', 'Saturday'),
```

```
(0.0, '2019-01-05 14:45:03', '6', 'Saturday'),
(0.0, '2019-01-05 15:23:44', '6', 'Saturday'),
(0.0, '2019-01-05 15:33:37', '6', 'Saturday'),
(0.0, '2019-01-05 15:47:02', '6', 'Saturday'),
(0.0, '2019-01-05 15:49:05', '6', 'Saturday'),
(0.0, '2019-01-05 15:52:56', '6', 'Saturday'),
(0.0, '2019-01-05 16:02:58', '6', 'Saturday'),
(0.0, '2019-01-05 16:08:16', '6', 'Saturday'),
(0.0, '2019-01-05 17:02:31', '6', 'Saturday'),
(0.0, '2019-01-05 17:23:08', '6', 'Saturday'),
(0.0, '2019-01-05 17:25:01', '6', 'Saturday'),
(0.0, '2019-01-05 17:28:47', '6', 'Saturday'),
(0.0, '2019-01-05 17:31:26', '6', 'Saturday'),
(0.0, '2019-01-05 17:45:55', '6', 'Saturday'),
(0.0, '2019-01-05 17:49:00', '6', 'Saturday'),
(0.0, '2019-01-05 17:58:41', '6', 'Saturday'),
(0.0, '2019-01-05 18:04:41', '6', 'Saturday'),
(0.0, '2019-01-05 18:58:24', '6', 'Saturday'),
(0.0, '2019-01-05 19:00:32', '6', 'Saturday'),
(0.0, '2019-01-05 19:12:45', '6', 'Saturday'),
(0.0, '2019-01-05 19:26:49', '6', 'Saturday'),
(0.0, '2019-01-05 19:28:53', '6', 'Saturday'),
(0.0, '2019-01-05 19:37:41', '6', 'Saturday'),
(0.0, '2019-01-05 19:39:56', '6', 'Saturday'),
(0.0, '2019-01-05 19:46:12', '6', 'Saturday'),
(0.0, '2019-01-05 19:52:23', '6', 'Saturday'),
(0.0, '2019-01-05 19:59:20', '6', 'Saturday'),
(0.0, '2019-01-05 20:26:32', '6', 'Saturday'),
(0.0, '2019-01-05 20:33:40', '6', 'Saturday'),
(0.0, '2019-01-05 20:37:20', '6', 'Saturday'),
(0.0, '2019-01-05 21:15:38', '6', 'Saturday'),
(0.0, '2019-01-05 21:22:51', '6', 'Saturday'),
(0.0, '2019-01-05 21:41:06', '6', 'Saturday'),
(0.0, '2019-01-05 22:38:11', '6', 'Saturday'),
(0.0, '2019-01-05 22:57:59', '6', 'Saturday'),
(0.0, '2019-01-05 23:13:04', '6', 'Saturday'),
(0.0, '2019-01-05 23:19:33', '6', 'Saturday'),
(0.0, '2019-01-06 00:45:16', '0', 'Sunday'),
(0.0, '2019-01-06 00:51:17', '0', 'Sunday'),
(0.0, '2019-01-06 02:25:05', '0', 'Sunday'),
(0.0, '2019-01-06 03:10:04', '0', 'Sunday'),
(0.0, '2019-01-06 03:14:20', '0', 'Sunday'),
(0.0, '2019-01-06 03:18:42', '0', 'Sunday'),
(0.0, '2019-01-06 03:34:22', '0', 'Sunday'),
(0.0, '2019-01-06 03:58:40', '0', 'Sunday'),
(0.0, '2019-01-06 04:07:25', '0', 'Sunday'),
(0.0, '2019-01-06 04:10:34', '0', 'Sunday'),
```

```
(0.0, '2019-01-06 05:27:48', '0', 'Sunday'),
(0.0, '2019-01-06 05:55:42', '0', 'Sunday'),
(0.0, '2019-01-06 07:32:26', '0', 'Sunday'),
(0.0, '2019-01-06 07:46:44', '0', 'Sunday'),
(0.0, '2019-01-06 07:47:57', '0', 'Sunday'),
(0.0, '2019-01-06 08:05:56', '0', 'Sunday'),
(0.0, '2019-01-06 08:09:22', '0', 'Sunday'),
(0.0, '2019-01-06 08:27:53', '0', 'Sunday'),
(0.0, '2019-01-06 08:29:56', '0', 'Sunday'),
(0.0, '2019-01-06 08:30:17', '0', 'Sunday'),
(0.0, '2019-01-06 08:31:08', '0', 'Sunday'),
(0.0, '2019-01-06 08:32:30', '0', 'Sunday'),
(0.0, '2019-01-06 08:49:08', '0', 'Sunday'),
(0.0, '2019-01-06 09:04:58', '0', 'Sunday'),
(0.0, '2019-01-06 09:35:56', '0', 'Sunday'),
(0.0, '2019-01-06 09:43:07', '0', 'Sunday'),
(0.0, '2019-01-06 10:18:19', '0', 'Sunday'),
(0.0, '2019-01-06 10:51:52', '0', 'Sunday'),
(0.0, '2019-01-06 10:56:50', '0', 'Sunday'),
(0.0, '2019-01-06 10:59:40', '0', 'Sunday'),
(0.0, '2019-01-06 11:05:40', '0', 'Sunday'),
(0.0, '2019-01-06 11:05:55', '0', 'Sunday'),
(0.0, '2019-01-06 11:12:08', '0', 'Sunday'),
(0.0, '2019-01-06 11:28:05', '0', 'Sunday'),
(0.0, '2019-01-06 11:39:24', '0', 'Sunday'),
(0.0, '2019-01-06 12:01:15', '0', 'Sunday'),
(0.0, '2019-01-06 12:28:08', '0', 'Sunday'),
(0.0, '2019-01-06 13:16:33', '0', 'Sunday'),
(0.0, '2019-01-06 13:21:12', '0', 'Sunday'),
(0.0, '2019-01-06 13:28:58', '0', 'Sunday'),
(0.0, '2019-01-06 13:40:42', '0', 'Sunday'),
(0.0, '2019-01-06 13:48:21', '0', 'Sunday'),
(0.0, '2019-01-06 13:49:13', '0', 'Sunday'),
(0.0, '2019-01-06 13:49:28', '0', 'Sunday'),
(0.0, '2019-01-06 14:42:19', '0', 'Sunday'),
(0.0, '2019-01-06 14:57:29', '0', 'Sunday'),
(0.0, '2019-01-06 15:44:04', '0', 'Sunday'),
(0.0, '2019-01-06 15:45:15', '0', 'Sunday'),
(0.0, '2019-01-06 15:54:43', '0', 'Sunday'),
(0.0, '2019-01-06 16:05:04', '0', 'Sunday'),
(0.0, '2019-01-06 16:23:53', '0', 'Sunday'),
(0.0, '2019-01-06 16:44:03', '0', 'Sunday'),
(0.0, '2019-01-06 16:45:32', '0', 'Sunday'),
(0.0, '2019-01-06 16:56:20', '0', 'Sunday'),
(0.0, '2019-01-06 17:05:39', '0', 'Sunday'),
(0.0, '2019-01-06 17:13:06', '0', 'Sunday'),
(0.0, '2019-01-06 18:14:27', '0', 'Sunday'),
```

```
(0.0, '2019-01-06 18:30:54', '0', 'Sunday'),
(0.0, '2019-01-06 18:51:34', '0', 'Sunday'),
(0.0, '2019-01-06 18:54:39', '0', 'Sunday'),
(0.0, '2019-01-06 18:54:51', '0', 'Sunday'),
(0.0, '2019-01-06 19:17:57', '0', 'Sunday'),
(0.0, '2019-01-06 19:22:01', '0', 'Sunday'),
(0.0, '2019-01-06 19:28:24', '0', 'Sunday'),
(0.0, '2019-01-06 19:31:25', '0', 'Sunday'),
(0.0, '2019-01-06 19:40:13', '0', 'Sunday'),
(0.0, '2019-01-06 19:45:32', '0', 'Sunday'),
(0.0, '2019-01-06 19:48:47', '0', 'Sunday'),
(0.0, '2019-01-06 19:55:56', '0', 'Sunday'),
(0.0, '2019-01-06 20:02:58', '0', 'Sunday'),
(0.0, '2019-01-06 20:17:33', '0', 'Sunday'),
(0.0, '2019-01-06 20:20:00', '0', 'Sunday'),
(0.0, '2019-01-06 20:31:21', '0', 'Sunday'),
(0.0, '2019-01-06 21:17:25', '0', 'Sunday'),
(0.0, '2019-01-06 21:23:23', '0', 'Sunday'),
(0.0, '2019-01-06 21:36:55', '0', 'Sunday'),
(0.0, '2019-01-06 21:38:23', '0', 'Sunday'),
(0.0, '2019-01-06 21:44:38', '0', 'Sunday'),
(0.0, '2019-01-06 21:46:48', '0', 'Sunday'),
(0.0, '2019-01-06 21:49:09', '0', 'Sunday'),
(0.0, '2019-01-06 21:55:36', '0', 'Sunday'),
(0.0, '2019-01-06 21:58:39', '0', 'Sunday'),
(0.0, '2019-01-06 21:58:49', '0', 'Sunday'),
(0.0, '2019-01-06 22:01:16', '0', 'Sunday'),
(0.0, '2019-01-06 22:16:48', '0', 'Sunday'),
(0.0, '2019-01-06 22:31:56', '0', 'Sunday'),
(0.0, '2019-01-06 22:41:08', '0', 'Sunday'),
(0.0, '2019-01-06 23:13:34', '0', 'Sunday'),
(0.0, '2019-01-06 23:14:36', '0', 'Sunday'),
(0.0, '2019-01-06 23:22:23', '0', 'Sunday'),
(0.0, '2019-01-06 23:34:16', '0', 'Sunday'),
(0.0, '2019-01-06 23:38:03', '0', 'Sunday'),
(0.0, '2019-01-06 23:58:49', '0', 'Sunday'),
(0.0, '2019-01-07 00:19:07', '1', 'Monday'),
(0.0, '2019-01-07 00:27:14', '1', 'Monday'),
(0.0, '2019-01-07 00:28:43', '1', 'Monday'),
(0.0, '2019-01-07 01:17:51', '1', 'Monday'),
(0.0, '2019-01-07 01:25:29', '1', 'Monday'),
(0.0, '2019-01-07 01:34:48', '1', 'Monday'),
(0.0, '2019-01-07 01:47:38', '1', 'Monday'),
(0.0, '2019-01-07 01:54:46', '1', 'Monday'),
(0.0, '2019-01-06 15:27:46', '0', 'Sunday'),
(0.0, '2019-01-07 02:52:31', '1', 'Monday'),
(0.0, '2019-01-07 02:52:42', '1', 'Monday'),
```

```
(0.0, '2019-01-07 03:19:54', '1', 'Monday'),
(0.0, '2019-01-07 03:25:23', '1', 'Monday'),
(0.0, '2019-01-07 03:58:37', '1', 'Monday'),
(0.0, '2019-01-07 04:31:08', '1', 'Monday'),
(0.0, '2019-01-07 04:42:01', '1', 'Monday'),
(0.0, '2019-01-07 05:01:14', '1', 'Monday'),
(0.0, '2019-01-07 05:19:52', '1', 'Monday'),
(0.0, '2019-01-07 05:54:57', '1', 'Monday'),
(0.0, '2019-01-07 06:08:44', '1', 'Monday'),
(0.0, '2019-01-07 06:41:32', '1', 'Monday'),
(0.0, '2019-01-07 07:00:49', '1', 'Monday'),
(0.0, '2019-01-07 07:02:53', '1', 'Monday'),
(0.0, '2019-01-07 07:16:54', '1', 'Monday'),
(0.0, '2019-01-07 07:19:16', '1', 'Monday'),
(0.0, '2019-01-07 08:25:03', '1', 'Monday'),
(0.0, '2019-01-07 08:41:51', '1', 'Monday'),
(0.0, '2019-01-07 08:55:14', '1', 'Monday'),
(0.0, '2019-01-07 09:12:39', '1', 'Monday'),
(0.0, '2019-01-07 09:17:33', '1', 'Monday'),
(0.0, '2019-01-07 09:20:55', '1', 'Monday'),
(0.0, '2019-01-07 09:21:19', '1', 'Monday'),
(0.0, '2019-01-07 09:26:07', '1', 'Monday'),
(0.0, '2019-01-07 09:32:07', '1', 'Monday'),
(0.0, '2019-01-07 09:32:12', '1', 'Monday'),
(0.0, '2019-01-07 09:44:15', '1', 'Monday'),
(0.0, '2019-01-07 09:45:40', '1', 'Monday'),
(0.0, '2019-01-07 09:47:20', '1', 'Monday'),
(0.0, '2019-01-07 09:56:28', '1', 'Monday'),
(0.0, '2019-01-07 09:59:01', '1', 'Monday'),
(0.0, '2019-01-07 10:17:56', '1', 'Monday'),
(0.0, '2019-01-07 10:18:08', '1', 'Monday'),
(0.0, '2019-01-07 10:28:31', '1', 'Monday'),
(0.0, '2019-01-07 10:37:12', '1', 'Monday'),
(0.0, '2019-01-07 10:50:22', '1', 'Monday'),
(0.0, '2019-01-07 10:52:27', '1', 'Monday'),
(0.0, '2019-01-07 10:59:11', '1', 'Monday'),
(0.0, '2019-01-07 11:02:49', '1', 'Monday'),
(0.0, '2019-01-07 11:11:08', '1', 'Monday'),
(0.0, '2019-01-07 11:23:32', '1', 'Monday'),
(0.0, '2019-01-07 11:32:00', '1', 'Monday'),
(0.0, '2019-01-07 11:41:31', '1', 'Monday'),
(0.0, '2019-01-07 11:42:05', '1', 'Monday'),
(0.0, '2019-01-07 11:43:54', '1', 'Monday'),
(0.0, '2019-01-07 11:45:31', '1', 'Monday'),
(0.0, '2019-01-07 11:55:42', '1', 'Monday'),
(0.0, '2019-01-07 12:19:34', '1', 'Monday'),
(0.0, '2019-01-07 12:34:24', '1', 'Monday'),
```

```
(0.0, '2019-01-07 12:59:08', '1', 'Monday'),
(0.0, '2019-01-07 13:06:55', '1', 'Monday'),
(0.0, '2019-01-07 13:42:06', '1', 'Monday'),
(0.0, '2019-01-07 13:50:32', '1', 'Monday'),
(0.0, '2019-01-07 14:06:25', '1', 'Monday'),
(0.0, '2019-01-07 14:19:19', '1', 'Monday'),
(0.0, '2019-01-07 14:21:18', '1', 'Monday'),
(0.0, '2019-01-07 14:23:48', '1', 'Monday'),
(0.0, '2019-01-07 14:41:20', '1', 'Monday'),
(0.0, '2019-01-07 14:44:32', '1', 'Monday'),
(0.0, '2019-01-07 15:04:00', '1', 'Monday'),
(0.0, '2019-01-07 15:06:17', '1', 'Monday'),
(0.0, '2019-01-07 15:13:32', '1', 'Monday'),
(0.0, '2019-01-07 15:14:53', '1', 'Monday'),
(0.0, '2019-01-07 15:18:24', '1', 'Monday'),
(0.0, '2019-01-07 15:45:24', '1', 'Monday'),
(0.0, '2019-01-07 15:56:49', '1', 'Monday'),
(0.0, '2019-01-07 16:02:55', '1', 'Monday'),
(0.0, '2019-01-07 16:03:39', '1', 'Monday'),
(0.0, '2019-01-07 16:19:38', '1', 'Monday'),
(0.0, '2019-01-07 16:21:56', '1', 'Monday'),
(0.0, '2019-01-07 16:27:57', '1', 'Monday'),
(0.0, '2019-01-07 16:46:25', '1', 'Monday'),
(0.0, '2019-01-07 16:48:00', '1', 'Monday'),
(0.0, '2019-01-07 17:09:18', '1', 'Monday'),
(0.0, '2019-01-07 17:18:23', '1', 'Monday'),
(0.0, '2019-01-07 17:18:49', '1', 'Monday'),
(0.0, '2019-01-07 17:21:41', '1', 'Monday'),
(0.0, '2019-01-07 17:34:02', '1', 'Monday'),
(0.0, '2019-01-07 17:34:13', '1', 'Monday'),
(0.0, '2019-01-07 17:36:32', '1', 'Monday'),
(0.0, '2019-01-07 17:37:20', '1', 'Monday'),
(0.0, '2019-01-07 17:49:01', '1', 'Monday'),
(0.0, '2019-01-07 17:49:28', '1', 'Monday'),
(0.0, '2019-01-07 18:00:53', '1', 'Monday'),
(0.0, '2019-01-07 18:01:25', '1', 'Monday'),
(0.0, '2019-01-07 18:05:11', '1', 'Monday'),
(0.0, '2019-01-07 18:12:10', '1', 'Monday'),
(0.0, '2019-01-07 18:19:14', '1', 'Monday'),
(0.0, '2019-01-07 18:22:10', '1', 'Monday'),
(0.0, '2019-01-07 18:58:22', '1', 'Monday'),
(0.0, '2019-01-07 19:09:37', '1', 'Monday'),
(0.0, '2019-01-07 19:32:45', '1', 'Monday'),
(0.0, '2019-01-07 19:55:47', '1', 'Monday'),
(0.0, '2019-01-07 20:17:38', '1', 'Monday'),
(0.0, '2019-01-07 20:19:42', '1', 'Monday'),
(0.0, '2019-01-07 20:36:12', '1', 'Monday'),
```

```
(0.0, '2019-01-07 20:36:33', '1', 'Monday'),
(0.0, '2019-01-07 21:03:40', '1', 'Monday'),
(0.0, '2019-01-07 21:07:33', '1', 'Monday'),
(0.0, '2019-01-07 21:12:34', '1', 'Monday'),
(0.0, '2019-01-07 21:37:23', '1', 'Monday'),
(0.0, '2019-01-07 22:21:13', '1', 'Monday'),
(0.0, '2019-01-07 22:29:13', '1', 'Monday'),
(0.0, '2019-01-07 22:31:14', '1', 'Monday'),
(0.0, '2019-01-07 22:40:08', '1', 'Monday'),
(0.0, '2019-01-07 22:43:34', '1', 'Monday'),
(0.0, '2019-01-07 22:57:04', '1', 'Monday'),
(0.0, '2019-01-07 23:09:53', '1', 'Monday'),
(0.0, '2019-01-07 23:14:41', '1', 'Monday'),
(0.0, '2019-01-07 23:26:09', '1', 'Monday'),
(0.0, '2019-01-07 23:26:38', '1', 'Monday'),
(0.0, '2019-01-07 23:52:23', '1', 'Monday'),
(0.0, '2019-01-08 02:59:30', '2', 'Tuesday'),
(0.0, '2019-01-08 03:23:21', '2', 'Tuesday'),
(0.0, '2019-01-08 03:37:31', '2', 'Tuesday'),
(0.0, '2019-01-08 04:27:35', '2', 'Tuesday'),
(0.0, '2019-01-08 04:55:35', '2', 'Tuesday'),
(0.0, '2019-01-08 05:06:18', '2', 'Tuesday'),
(0.0, '2019-01-08 05:44:20', '2', 'Tuesday'),
(0.0, '2019-01-08 05:49:24', '2', 'Tuesday'),
(0.0, '2019-01-08 06:07:51', '2', 'Tuesday'),
(0.0, '2019-01-08 06:24:29', '2', 'Tuesday'),
(0.0, '2019-01-08 06:37:48', '2', 'Tuesday'),
(0.0, '2019-01-08 06:51:34', '2', 'Tuesday'),
(0.0, '2019-01-08 07:02:21', '2', 'Tuesday'),
(0.0, '2019-01-08 07:05:53', '2', 'Tuesday'),
(0.0, '2019-01-08 07:32:43', '2', 'Tuesday'),
(0.0, '2019-01-08 07:38:03', '2', 'Tuesday'),
(0.0, '2019-01-08 07:40:52', '2', 'Tuesday'),
(0.0, '2019-01-08 07:47:19', '2', 'Tuesday'),
(0.0, '2019-01-08 08:40:56', '2', 'Tuesday'),
(0.0, '2019-01-08 09:09:50', '2', 'Tuesday'),
(0.0, '2019-01-08 09:28:46', '2', 'Tuesday'),
(0.0, '2019-01-08 09:36:50', '2', 'Tuesday'),
(0.0, '2019-01-08 10:04:27', '2', 'Tuesday'),
(0.0, '2019-01-08 10:08:27', '2', 'Tuesday'),
(0.0, '2019-01-08 10:18:51', '2', 'Tuesday'),
(0.0, '2019-01-08 10:32:22', '2', 'Tuesday'),
(0.0, '2019-01-08 10:37:29', '2', 'Tuesday'),
(0.0, '2019-01-08 10:37:35', '2', 'Tuesday'),
(0.0, '2019-01-08 10:52:21', '2', 'Tuesday'),
(0.0, '2019-01-08 10:59:17', '2', 'Tuesday'),
(0.0, '2019-01-08 11:01:02', '2', 'Tuesday'),
```

```
(0.0, '2019-01-08 11:05:13', '2', 'Tuesday'),
(0.0, '2019-01-08 11:35:15', '2', 'Tuesday'),
(0.0, '2019-01-08 11:37:55', '2', 'Tuesday'),
(0.0, '2019-01-08 11:48:25', '2', 'Tuesday'),
(0.0, '2019-01-08 11:49:12', '2', 'Tuesday'),
(0.0, '2019-01-08 11:49:32', '2', 'Tuesday'),
(0.0, '2019-01-08 12:04:53', '2', 'Tuesday'),
(0.0, '2019-01-08 12:14:01', '2', 'Tuesday'),
(0.0, '2019-01-08 12:22:14', '2', 'Tuesday'),
(0.0, '2019-01-08 12:23:04', '2', 'Tuesday'),
(0.0, '2019-01-08 12:37:16', '2', 'Tuesday'),
(0.0, '2019-01-08 12:48:27', '2', 'Tuesday'),
(0.0, '2019-01-08 12:53:52', '2', 'Tuesday'),
(0.0, '2019-01-08 12:58:17', '2', 'Tuesday'),
(0.0, '2019-01-08 13:02:24', '2', 'Tuesday'),
(0.0, '2019-01-08 13:20:47', '2', 'Tuesday'),
(0.0, '2019-01-08 13:41:19', '2', 'Tuesday'),
(0.0, '2019-01-08 13:51:18', '2', 'Tuesday'),
(0.0, '2019-01-08 13:54:04', '2', 'Tuesday'),
(0.0, '2019-01-08 13:57:06', '2', 'Tuesday'),
(0.0, '2019-01-08 14:06:20', '2', 'Tuesday'),
(0.0, '2019-01-08 14:20:14', '2', 'Tuesday'),
(0.0, '2019-01-08 14:24:12', '2', 'Tuesday'),
(0.0, '2019-01-08 14:31:35', '2', 'Tuesday'),
(0.0, '2019-01-08 14:36:16', '2', 'Tuesday'),
(0.0, '2019-01-08 14:36:49', '2', 'Tuesday'),
(0.0, '2019-01-08 14:38:53', '2', 'Tuesday'),
(0.0, '2019-01-08 14:50:40', '2', 'Tuesday'),
(0.0, '2019-01-08 14:54:04', '2', 'Tuesday'),
(0.0, '2019-01-08 14:57:45', '2', 'Tuesday'),
(0.0, '2019-01-08 15:13:53', '2', 'Tuesday'),
(0.0, '2019-01-08 15:15:37', '2', 'Tuesday'),
(0.0, '2019-01-08 15:15:48', '2', 'Tuesday'),
(0.0, '2019-01-08 15:37:33', '2', 'Tuesday'),
(0.0, '2019-01-08 15:38:24', '2', 'Tuesday'),
(0.0, '2019-01-08 15:38:30', '2', 'Tuesday'),
(0.0, '2019-01-08 16:03:56', '2', 'Tuesday'),
(0.0, '2019-01-08 16:05:07', '2', 'Tuesday'),
(0.0, '2019-01-08 16:16:38', '2', 'Tuesday'),
(0.0, '2019-01-08 16:16:55', '2', 'Tuesday'),
(0.0, '2019-01-08 16:25:19', '2', 'Tuesday'),
(0.0, '2019-01-08 16:27:23', '2', 'Tuesday'),
(0.0, '2019-01-08 16:42:26', '2', 'Tuesday'),
(0.0, '2019-01-08 16:57:04', '2', 'Tuesday'),
(0.0, '2019-01-08 17:38:45', '2', 'Tuesday'),
(0.0, '2019-01-08 18:09:50', '2', 'Tuesday'),
(0.0, '2019-01-08 19:11:59', '2', 'Tuesday'),
```

```
(0.0, '2019-01-08 19:15:18', '2', 'Tuesday'),
(0.0, '2019-01-08 19:16:39', '2', 'Tuesday'),
(0.0, '2019-01-08 19:28:03', '2', 'Tuesday'),
(0.0, '2019-01-08 19:33:16', '2', 'Tuesday'),
(0.0, '2019-01-08 19:35:49', '2', 'Tuesday'),
(0.0, '2019-01-08 19:37:40', '2', 'Tuesday'),
(0.0, '2019-01-08 19:58:42', '2', 'Tuesday'),
(0.0, '2019-01-08 20:05:33', '2', 'Tuesday'),
(0.0, '2019-01-08 20:15:15', '2', 'Tuesday'),
(0.0, '2019-01-08 20:16:17', '2', 'Tuesday'),
(0.0, '2019-01-08 20:17:55', '2', 'Tuesday'),
(0.0, '2019-01-08 20:28:55', '2', 'Tuesday'),
(0.0, '2019-01-08 20:37:53', '2', 'Tuesday'),
(0.0, '2019-01-08 20:42:38', '2', 'Tuesday'),
(0.0, '2019-01-08 20:42:50', '2', 'Tuesday'),
(0.0, '2019-01-08 20:58:50', '2', 'Tuesday'),
(0.0, '2019-01-08 21:08:44', '2', 'Tuesday'),
(0.0, '2019-01-08 21:28:34', '2', 'Tuesday'),
(0.0, '2019-01-08 21:40:38', '2', 'Tuesday'),
(0.0, '2019-01-08 22:03:35', '2', 'Tuesday'),
(0.0, '2019-01-08 22:06:18', '2', 'Tuesday'),
(0.0, '2019-01-08 22:12:08', '2', 'Tuesday'),
(0.0, '2019-01-08 22:15:49', '2', 'Tuesday'),
(0.0, '2019-01-08 22:18:44', '2', 'Tuesday'),
(0.0, '2019-01-08 22:24:21', '2', 'Tuesday'),
(0.0, '2019-01-08 22:33:37', '2', 'Tuesday'),
(0.0, '2019-01-08 22:44:22', '2', 'Tuesday'),
(0.0, '2019-01-08 22:46:09', '2', 'Tuesday'),
(0.0, '2019-01-08 22:47:27', '2', 'Tuesday'),
(0.0, '2019-01-08 22:51:02', '2', 'Tuesday'),
(0.0, '2019-01-08 23:04:02', '2', 'Tuesday'),
(0.0, '2019-01-08 23:33:32', '2', 'Tuesday'),
(0.0, '2019-01-08 23:34:43', '2', 'Tuesday'),
(0.0, '2019-01-08 23:42:01', '2', 'Tuesday'),
(0.0, '2019-01-08 23:46:55', '2', 'Tuesday'),
(0.0, '2019-01-08 23:58:38', '2', 'Tuesday'),
(0.0, '2019-01-09 00:03:53', '3', 'Wednesday'),
(0.0, '2019-01-09 00:14:46', '3', 'Wednesday'),
(0.0, '2019-01-09 00:42:04', '3', 'Wednesday'),
(0.0, '2019-01-09 00:51:03', '3', 'Wednesday'),
(0.0, '2019-01-09 01:03:48', '3', 'Wednesday'),
(0.0, '2019-01-09 01:16:59', '3', 'Wednesday'),
(0.0, '2019-01-09 01:28:00', '3', 'Wednesday'),
(0.0, '2019-01-09 01:30:14', '3', 'Wednesday'),
(0.0, '2019-01-09 01:38:58', '3', 'Wednesday'),
(0.0, '2019-01-09 01:46:31', '3', 'Wednesday'),
(0.0, '2019-01-09 02:09:31', '3', 'Wednesday'),
```

```
 (0.0, '2019-01-09 02:38:47', '3', 'Wednesday'),
 (0.0, '2019-01-09 02:52:55', '3', 'Wednesday'),
 (0.0, '2019-01-09 04:07:08', '3', 'Wednesday'),
 (0.0, '2019-01-09 04:30:13', '3', 'Wednesday'),
 (0.0, '2019-01-09 04:47:24', '3', 'Wednesday'),
 (0.0, '2019-01-09 04:47:37', '3', 'Wednesday'),
 (0.0, '2019-01-09 04:48:18', '3', 'Wednesday'),
 (0.0, '2019-01-09 05:07:18', '3', 'Wednesday'),
 (0.0, '2019-01-09 05:12:40', '3', 'Wednesday'),
 (0.0, '2019-01-09 05:14:15', '3', 'Wednesday'),
 (0.0, '2019-01-09 05:15:34', '3', 'Wednesday'),
 (0.0, '2019-01-09 05:22:05', '3', 'Wednesday'),
 (0.0, '2019-01-09 05:27:01', '3', 'Wednesday'),
 (0.0, '2019-01-09 05:30:01', '3', 'Wednesday'),
 (0.0, '2019-01-09 05:40:54', '3', 'Wednesday'),
 (0.0, '2019-01-09 05:44:40', '3', 'Wednesday'),
 (0.0, '2019-01-09 05:47:24', '3', 'Wednesday'),
 (0.0, '2019-01-09 06:07:50', '3', 'Wednesday'),
 (0.0, '2019-01-09 06:12:31', '3', 'Wednesday'),
 …]
```

[92]:
```
%%sql
SELECT
(ViewCount/CreationDate)*100.0 AS VolumeViewCount,
CreationDate,
strftime('%w', CreationDate),
CASE CAST(strftime('%w', CreationDate) as integer)
    WHEN 0 THEN 'Sunday'
    WHEN 1 THEN 'Monday'
    WHEN 2 THEN 'Tuesday'
    WHEN 3 THEN 'Wednesday'
    WHEN 4 THEN 'Thursday'
    WHEN 5 THEN 'Friday'
    ELSE 'Saturday' END AS 'DayOfWeek',
RANK() OVER(ORDER BY ViewCount DESC) AS Rank
FROM posts
WHERE VolumeViewCount !=0.0
GROUP BY VolumeViewCount
```

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

[92]:
```
[(900.0, '2019-01-22 15:16:47', '2', 'Tuesday', 1),
 (800.0, '2019-02-24 14:07:11', '0', 'Sunday', 2),
 (500.0, '2019-10-14 11:29:21', '1', 'Monday', 3),
 (400.0, '2019-03-21 01:19:52', '4', 'Thursday', 4),
 (300.0, '2019-07-23 22:15:03', '2', 'Tuesday', 5),
 (200.0, '2019-01-02 12:20:07', '3', 'Wednesday', 6),
```

```
 (100.0, '2019-01-13 17:44:56', '0', 'Sunday', 7)]
```

[93]:
```
%%sql
SELECT
CASE CAST(strftime('%w', CreationDate) as integer)
    WHEN 0 THEN 'Sunday'
    WHEN 1 THEN 'Monday'
    WHEN 2 THEN 'Tuesday'
    WHEN 3 THEN 'Wednesday'
    WHEN 4 THEN 'Thursday'
    WHEN 5 THEN 'Friday'
    ELSE 'Saturday' END AS 'DayOfWeek',
SUM(ViewCount) AS 'VolumeViewCount'
FROM posts
GROUP BY DayOfWeek
ORDER BY 2
```

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

[93]:
```
[('Saturday', 175247),
 ('Sunday', 234459),
 ('Friday', 267324),
 ('Monday', 311546),
 ('Tuesday', 320382),
 ('Wednesday', 330506),
 ('Thursday', 331507)]
```

[94]:
```
sql = """
SELECT
CASE CAST(strftime('%w', CreationDate) as integer)
    WHEN 0 THEN 'Sunday'
    WHEN 1 THEN 'Monday'
    WHEN 2 THEN 'Tuesday'
    WHEN 3 THEN 'Wednesday'
    WHEN 4 THEN 'Thursday'
    WHEN 5 THEN 'Friday'
    ELSE 'Saturday' END AS 'DayOfWeek',
SUM(ViewCount) AS 'VolumeViewCount'
FROM posts
GROUP BY DayOfWeek
ORDER BY 2
"""
store_query("Task 2", "Rank the days of the week from highest to lowest in␣
 ↪terms of the volume of ViewCount as a percentage 2", sql)
```

9 Task 3: Cross Table Queries

9.1 Lifecycle Stage: Analyze

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

9.1.1 Cross Table Queries

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

[95]:
```
%%sql
SELECT
COUNT(users.aboutme) AS TotalCount
FROM users
JOIN posts ON users.id=posts.owneruserid
```

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

[95]:
```
[(17189,)]
```

[96]:
```
sql = """
SELECT
COUNT(users.aboutme) AS TotalCount
FROM users
JOIN posts ON users.id=posts.owneruserid
"""
store_query("Task 3", "How many posts have been created by a user that has a␣
 ↪filled out the AboutMe section?", sql)
```

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

[97]:
```
%%sql
SELECT COUNT(users.aboutme) FROM USERS WHERE users.aboutme != ''
```

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

[97]:
```
[(4021,)]
```

[98]:
```
%%sql
SELECT CAST(COUNT(posts.owneruserid)/(SELECT CAST (COUNT(users.aboutme) AS␣
 ↪FLOAT) FROM USERS WHERE users.aboutme != '') AS FLOAT) AS PostsPerUser
FROM users
JOIN posts ON users.id=posts.owneruserid
WHERE users.aboutme != ''
```

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

[98]:
```
[(4.274807261875155,)]
```

[99]:
```
sql = """
SELECT CAST(COUNT(posts.owneruserid)/(SELECT CAST (COUNT(users.aboutme) AS␣
 ↪FLOAT) FROM USERS WHERE users.aboutme != '') AS FLOAT) AS PostsPerUser
FROM users
JOIN posts ON users.id=posts.owneruserid
WHERE users.aboutme != ''
"""
store_query("Task 3", "Considering only the users with an AboutMe, how many␣
 ↪posts are there per user?", sql)
```

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

[100]:
```
%%sql
SELECT COUNT(*)comments, posts.id
FROM posts
JOIN comments on posts.id=comments.postid
GROUP BY comments.postid
ORDER BY COUNT(comments.postid) DESC
LIMIT 10
```

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

[100]:
```
[(66, 386853),
 (34, 386556),
 (31, 418910),
 (31, 395232),
 (27, 402987),
 (26, 386075),
 (24, 394118),
 (23, 402950),
 (23, 398828),
 (22, 396111)]
```

[101]:
```
sql = """
SELECT COUNT(*)comments, posts.id
FROM posts
JOIN comments on posts.id=comments.postid
GROUP BY comments.postid
ORDER BY COUNT(comments.postid) DESC
LIMIT 10
"""
store_query("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?", sql)
```

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

score) score?

[102]:
```
%%sql
SELECT posts.id,
posts.score+SUM(comments.score) AS TotalScore
FROM posts
JOIN comments on posts.id=comments.postid
GROUP BY posts.id
ORDER BY TotalScore DESC
LIMIT 10
```

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

[102]:
```
[(394118, 306),
 (394128, 169),
 (388578, 141),
 (398653, 111),
 (388566, 101),
 (398646, 99),
 (421677, 96),
 (420526, 92),
 (400317, 72),
 (388582, 69)]
```

[103]:
```
sql = """
SELECT COUNT(*)comments, posts.id
FROM posts
JOIN comments on posts.id=comments.postid
GROUP BY comments.postid
ORDER BY COUNT(comments.postid) DESC
LIMIT 10
"""
store_query("Task 3", "What are the Top 10 posts which have the highest␣
 ↪cummulative (post score + comment score) score?", sql)
```

Who are the top 10 users who comment the most?

[104]:
```
%%sql
SELECT users.id, comments.userid, users.reputation, COUNT(comments.userid) AS␣
 ↪TotalComments
FROM users
JOIN comments ON users.id=comments.userid
GROUP BY comments.userid
ORDER BY COUNT(comments.userid) DESC
LIMIT 10
```

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

[104]:
```
[(919, 919, 223056, 3301),
 (805, 805, 228662, 1153),
 (143489, 143489, 2890, 1024),
 (11887, 11887, 39200, 805),
 (85665, 85665, 17391, 691),
 (164061, 164061, 13485, 540),
 (22047, 22047, 41385, 536),
 (158565, 158565, 6482, 504),
 (7962, 7962, 8030, 492),
 (35989, 35989, 71548, 470)]
```

[105]:
```
sql = """
SELECT users.id, comments.userid, users.reputation, COUNT(comments.userid) AS␣
 ↪TotalComments
FROM users
JOIN comments ON users.id=comments.userid
GROUP BY comments.userid
ORDER BY COUNT(comments.userid) 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?

[106]:
```
%%sql
SELECT users.id, users.reputation, COUNT(posts.owneruserid) AS TotalPosts
FROM users
JOIN posts ON users.id=posts.owneruserid
GROUP BY posts.owneruserid
ORDER BY users.reputation DESC
LIMIT 10
```

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

[106]:
```
[(805, 228662, 230),
 (919, 223056, 203),
 (7290, 115531, 35),
 (686, 85077, 386),
 (28666, 75024, 8),
 (35989, 71548, 230),
 (7224, 65999, 233),
 (4253, 59952, 71),
 (1352, 59160, 285),
 (22311, 51155, 140)]
```

[107]:
```
sql = """
SELECT users.id, users.reputation, COUNT(posts.owneruserid) AS TotalPosts
```

```
FROM users
JOIN posts ON users.id=posts.owneruserid
GROUP BY posts.owneruserid
ORDER BY users.reputation DESC
LIMIT 10
"""
store_query("Task 3", "Who are the top 10 users who post the most?", sql)
```

10 Task 4: Check the Queries Table

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

First let's check it's contents:

[108]:
```
%sql SELECT * FROM queries
```

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

[108]:
```
[('Single Table Queries', 'Which 5 users have viewed the most times and what is
the sum of those views per user?', '\nSELECT Id, SUM(Views) AS TotalViews\n
FROM Users\n        GROUP BY Id\n            ORDER BY TotalViews DESC\n
LIMIT 5\n               '),
 ('Task 1', 'Create table comments', '\n    CREATE TABLE "comments" (\n    "Id"
INTEGER,\n    "PostId" INTEGER,\n    "Score" INTEGER,\n    "Text" TEXT,\n
"CreationDate" TEXT,\n    "UserId" INTEGER\n    )\n    '),
 ('Task 1', 'Create table posts', '\n    CREATE TABLE "posts" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "PostTypeId" INTEGER,\n    "AcceptedAnswerId"
INTEGER,\n    "ParentId" INTE … (372 characters truncated) … \n
"FavoriteCount" INTEGER,\n    "ClosedDate" TEXT,\n    "CommunityOwnedDate"
TEXT,\n    FOREIGN KEY(OwnerUserId) REFERENCES user(id)\n    )\n    '),
 ('Task 1', 'Create table users', '\n    CREATE TABLE "users" (\n    "Id"
INTEGER NOT NULL PRIMARY KEY,\n    "Reputation" INTEGER,\n    "CreationDate"
TEXT,\n    "DisplayName" TEXT,\n  … (82 characters truncated) … tMe" TEXT,\n
"Views" INTEGER,\n    "UpVotes" INTEGER,\n    "DownVotes" INTEGER,\n
"ProfileImageUrl" TEXT,\n    "AccountId" INTEGER\n    )\n    '),
 ('Task 1', 'Count number of rows in comments table', '\n    SELECT COUNT(*) as
comment_count\n    FROM comments\n    '),
 ('Task 1', 'Count number of rows in users table', '\n    SELECT COUNT(*) as
user_count\n    FROM users\n    '),
 ('Task 1', 'Count number of rows in posts table', '\n    SELECT COUNT(*) as
post_count\n    FROM posts\n    '),
 ('Task 1', 'Select 5 random rows from comments table', '\n    SELECT * FROM
COMMENTS\n    ORDER BY RANDOM()\n    LIMIT 5;\n    '),
 ('Task 1', 'Select 5 random rows from posts table', '\n    SELECT * FROM
Posts\n    ORDER BY RANDOM()\n    LIMIT 5;\n    '),
 ('Task 1', 'Select 5 random rows from users table', '\n    SELECT * FROM
Users\n    ORDER BY RANDOM()\n    LIMIT 5;\n    '),
```

```
 ('Task 2', 'How many posts have 0 comments?', '\nSELECT COUNT() FROM
posts\nWHERE "CommentCount" = 0\n'),
 ('Task 2', 'How many posts have 1 comment?', '\nSELECT COUNT() FROM
posts\nWHERE "CommentCount" = 1\n'),
 ('Task 2', 'How many posts have 2 comments or more?', '\nSELECT COUNT() FROM
posts\nWHERE "CommentCount" >=2\n'),
 ('Task 2', 'Find 5 posts with the highest viewcount', '\nSELECT *\nFROM
posts\nORDER BY ViewCount DESC\nLIMIT 5\n'),
 ('Task 2', 'Find 5 posts with the highest Scores', '\nSELECT *\nFROM
posts\nORDER BY Score DESC\nLIMIT 5\n'),
 ('Task 2', 'What are the 5 most frequent scores on posts?', '\nSELECT Score,
COUNT(Score) AS Frequency\nFROM posts\nGROUP BY Score\nORDER BY COUNT(Score)
DESC\nLIMIT 5\n'),
 ('Task 2', 'How many posts have the keyword data in their tags?', '\nSELECT
COUNT() FROM posts\nWHERE Tags LIKE "%data%"\n'),
 ('Task 2', 'What are the 5 most frequent commentcount for posts?', '\nSELECT
CommentCount, COUNT(CommentCount) AS Frequency\nFROM posts\nGROUP BY
CommentCount\nORDER BY COUNT(CommentCount) DESC\nLIMIT 5\n'),
 ('Task 2', 'How many posts have an accepted answer?', '\nSELECT COUNT() FROM
posts\nWHERE "AcceptedAnswerId" >0\n'),
 ('Task 2', 'What is the average reputation of table users?', '\nSELECT
AVG(Reputation)\nFROM users\n'),
 ('Task 2', 'What is the min and max reputation of users?', '\nSELECT
MAX(Reputation),MIN(Reputation)\nFROM users\n'),
 ('Task 2', 'What is the length of the body of 5 most viewed posts?', '\nSELECT
ViewCount, LEN_BODY\nFROM posts\nORDER BY ViewCount DESC\nLIMIT 5\n'),
 ('Task 2', 'How many different locations are there in the users table?',
'\nSELECT COUNT (DISTINCT(Location)) AS NumberofLocations\nFrom users\nWHERE
Location IS NOT NULL\n'),
 ('Task 2', 'What are the top 5 locations of users?', '\nSELECT Location,
COUNT(Location) AS Frequency\nFROM users\nGROUP BY Location\nORDER BY
COUNT(Location) DESC\nLIMIT 5\n'),
 ('Task 2', 'Rank the days of the week from highest to lowest in terms of the
volume of ViewCount as a percentage 2', "\nSELECT\nCASE CAST(strftime('%w',
CreationDate) as integer)\n    WHEN 0 THEN 'Sunday'\n    WHEN 1 THEN 'Monday'\n
WHEN 2 THEN 'Tuesday'\n    WHEN … (48 characters truncated) … n    WHEN 5
THEN 'Friday'\n    ELSE 'Saturday' END AS 'DayOfWeek',\nSUM(ViewCount) AS
'VolumeViewCount'\nFROM posts\nGROUP BY DayOfWeek\nORDER BY 2\n"),
 ('Task 3', 'How many posts have been created by a user that has a filled out
the AboutMe section?', '\nSELECT\nCOUNT(users.aboutme) AS TotalCount\nFROM
users\nJOIN posts ON users.id=posts.owneruserid\n'),
 ('Task 3', 'Considering only the users with an AboutMe, how many posts are
there per user?', "\nSELECT CAST(COUNT(posts.owneruserid)/(SELECT CAST
(COUNT(users.aboutme) AS FLOAT) FROM USERS WHERE users.aboutme != '') AS FLOAT)
AS PostsPerUser\nFROM users\nJOIN posts ON users.id=posts.owneruserid\nWHERE
users.aboutme != ''\n"),
 ('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?', '\nSELECT
COUNT(*)comments, posts.id\nFROM posts\nJOIN comments on
posts.id=comments.postid\nGROUP BY comments.postid\nORDER BY
COUNT(comments.postid) DESC\nLIMIT 10\n'),
 ('Task 3', 'What are the Top 10 posts which have the highest cummulative (post
score + comment score) score?', '\nSELECT COUNT(*)comments, posts.id\nFROM
posts\nJOIN comments on posts.id=comments.postid\nGROUP BY
comments.postid\nORDER BY COUNT(comments.postid) DESC\nLIMIT 10\n'),
 ('Task 3', 'Who are the top 10 users who comment the most?', '\nSELECT
users.id, comments.userid, users.reputation, COUNT(comments.userid) AS
TotalComments\nFROM users\nJOIN comments ON users.id=comments.userid\nGROUP BY
comments.userid\nORDER BY COUNT(comments.userid) DESC\nLIMIT 10\n'),
 ('Task 3', 'Who are the top 10 users who post the most?', '\nSELECT users.id,
users.reputation, COUNT(posts.owneruserid) AS TotalPosts\nFROM users\nJOIN posts
ON users.id=posts.owneruserid\nGROUP BY posts.owneruserid\nORDER BY
users.reputation DESC\nLIMIT 10\n')]
```

10.1 Drop Duplicates

You likely have some duplicates. Lets drop them.

[109]:
```
# 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
```

10.2 Case Issues

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

[110]:
```
for col in queries.columns:
    queries[col] = queries[col].str.upper()

queries
```

[110]:

	task	action	\
0	SINGLE TABLE QUERIES	WHICH 5 USERS HAVE VIEWED THE MOST TIMES AND W…	
1	TASK 1	CREATE TABLE COMMENTS	
2	TASK 1	CREATE TABLE POSTS	
3	TASK 1	CREATE TABLE USERS	
4	TASK 1	COUNT NUMBER OF ROWS IN COMMENTS TABLE	
5	TASK 1	COUNT NUMBER OF ROWS IN USERS TABLE	
6	TASK 1	COUNT NUMBER OF ROWS IN POSTS TABLE	
7	TASK 1	SELECT 5 RANDOM ROWS FROM COMMENTS TABLE	
8	TASK 1	SELECT 5 RANDOM ROWS FROM POSTS TABLE	

9	TASK 1	SELECT 5 RANDOM ROWS FROM USERS TABLE
10	TASK 2	HOW MANY POSTS HAVE 0 COMMENTS?
11	TASK 2	HOW MANY POSTS HAVE 1 COMMENT?
12	TASK 2	HOW MANY POSTS HAVE 2 COMMENTS OR MORE?
13	TASK 2	FIND 5 POSTS WITH THE HIGHEST VIEWCOUNT
14	TASK 2	FIND 5 POSTS WITH THE HIGHEST SCORES
15	TASK 2	WHAT ARE THE 5 MOST FREQUENT SCORES ON POSTS?
16	TASK 2	HOW MANY POSTS HAVE THE KEYWORD DATA IN THEIR …
17	TASK 2	WHAT ARE THE 5 MOST FREQUENT COMMENTCOUNT FOR …
18	TASK 2	HOW MANY POSTS HAVE AN ACCEPTED ANSWER?
19	TASK 2	WHAT IS THE AVERAGE REPUTATION OF TABLE USERS?
20	TASK 2	WHAT IS THE MIN AND MAX REPUTATION OF USERS?
21	TASK 2	WHAT IS THE LENGTH OF THE BODY OF 5 MOST VIEWE…
22	TASK 2	HOW MANY DIFFERENT LOCATIONS ARE THERE IN THE …
23	TASK 2	WHAT ARE THE TOP 5 LOCATIONS OF USERS?
24	TASK 2	RANK THE DAYS OF THE WEEK FROM HIGHEST TO LOWE…
25	TASK 3	HOW MANY POSTS HAVE BEEN CREATED BY A USER THA…
26	TASK 3	CONSIDERING ONLY THE USERS WITH AN ABOUTME, HO…
27	TASK 3	NOT TAKING INTO ACCOUNT THE COMMENTCOUNT FIELD…
28	TASK 3	WHAT ARE THE TOP 10 POSTS WHICH HAVE THE HIGHE…
29	TASK 3	WHO ARE THE TOP 10 USERS WHO COMMENT THE MOST?
30	TASK 3	WHO ARE THE TOP 10 USERS WHO POST THE MOST?

	query
0	\nSELECT ID, SUM(VIEWS) AS TOTALVIEWS\n FRO…
1	\n CREATE TABLE "COMMENTS" (\n "ID" INTE…
2	\n CREATE TABLE "POSTS" (\n "ID" INTEGER…
3	\n CREATE TABLE "USERS" (\n "ID" INTEGER…
4	\n SELECT COUNT(*) AS COMMENT_COUNT\n FR…
5	\n SELECT COUNT(*) AS USER_COUNT\n FROM …
6	\n SELECT COUNT(*) AS POST_COUNT\n FROM …
7	\n SELECT * FROM COMMENTS\n ORDER BY RAN…
8	\n SELECT * FROM POSTS\n ORDER BY RANDOM…
9	\n SELECT * FROM USERS\n ORDER BY RANDOM…
10	\nSELECT COUNT() FROM POSTS\nWHERE "COMMENTCOU…
11	\nSELECT COUNT() FROM POSTS\nWHERE "COMMENTCOU…
12	\nSELECT COUNT() FROM POSTS\nWHERE "COMMENTCOU…
13	\nSELECT *\nFROM POSTS\nORDER BY VIEWCOUNT DES…
14	\nSELECT *\nFROM POSTS\nORDER BY SCORE DESC\nL…
15	\nSELECT SCORE, COUNT(SCORE) AS FREQUENCY\nFRO…
16	\nSELECT COUNT() FROM POSTS\nWHERE TAGS LIKE "…
17	\nSELECT COMMENTCOUNT, COUNT(COMMENTCOUNT) AS …
18	\nSELECT COUNT() FROM POSTS\nWHERE "ACCEPTEDAN…
19	\nSELECT AVG(REPUTATION)\nFROM USERS\n
20	\nSELECT MAX(REPUTATION),MIN(REPUTATION)\nFROM…
21	\nSELECT VIEWCOUNT, LEN_BODY\nFROM POSTS\nORDE…
22	\nSELECT COUNT (DISTINCT(LOCATION)) AS NUMBERO…

```
23  \nSELECT LOCATION, COUNT(LOCATION) AS FREQUENC…
24  \nSELECT\nCASE CAST(STRFTIME('%W', CREATIONDAT…
25  \nSELECT\nCOUNT(USERS.ABOUTME) AS TOTALCOUNT\n…
26  \nSELECT CAST(COUNT(POSTS.OWNERUSERID)/(SELECT…
27  \nSELECT COUNT(*)COMMENTS, POSTS.ID\nFROM POST…
28  \nSELECT COUNT(*)COMMENTS, POSTS.ID\nFROM POST…
29  \nSELECT USERS.ID, COMMENTS.USERID, USERS.REPU…
30  \nSELECT USERS.ID, USERS.REPUTATION, COUNT(POS…
```

[111]:
```
# Write the now deduped uppercase dataframe back to sqlite and replace the table
queries.to_sql('queries', con, if_exists='replace', index=False)
```

[111]: 31

10.3 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:

[112]:
```
%%sql
SELECT query
    FROM queries
        WHERE query LIKE '%GROUP BY%'
```

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

[112]:
```
[('\nSELECT ID, SUM(VIEWS) AS TOTALVIEWS\n    FROM USERS\n        GROUP BY ID\n
ORDER BY TOTALVIEWS DESC\n                LIMIT 5\n                ',),
 ('\nSELECT SCORE, COUNT(SCORE) AS FREQUENCY\nFROM POSTS\nGROUP BY SCORE\nORDER
BY COUNT(SCORE) DESC\nLIMIT 5\n',),
 ('\nSELECT COMMENTCOUNT, COUNT(COMMENTCOUNT) AS FREQUENCY\nFROM POSTS\nGROUP BY
COMMENTCOUNT\nORDER BY COUNT(COMMENTCOUNT) DESC\nLIMIT 5\n',),
 ('\nSELECT LOCATION, COUNT(LOCATION) AS FREQUENCY\nFROM USERS\nGROUP BY
LOCATION\nORDER BY COUNT(LOCATION) DESC\nLIMIT 5\n',),
 ("\nSELECT\nCASE CAST(STRFTIME('%W', CREATIONDATE) AS INTEGER)\n    WHEN 0 THEN
'SUNDAY'\n    WHEN 1 THEN 'MONDAY'\n    WHEN 2 THEN 'TUESDAY'\n    WHEN … (48
characters truncated) … n    WHEN 5 THEN 'FRIDAY'\n    ELSE 'SATURDAY' END AS
'DAYOFWEEK',\nSUM(VIEWCOUNT) AS 'VOLUMEVIEWCOUNT'\nFROM POSTS\nGROUP BY
DAYOFWEEK\nORDER BY 2\n",),
 ('\nSELECT COUNT(*)COMMENTS, POSTS.ID\nFROM POSTS\nJOIN COMMENTS ON
POSTS.ID=COMMENTS.POSTID\nGROUP BY COMMENTS.POSTID\nORDER BY
COUNT(COMMENTS.POSTID) DESC\nLIMIT 10\n',),
 ('\nSELECT COUNT(*)COMMENTS, POSTS.ID\nFROM POSTS\nJOIN COMMENTS ON
POSTS.ID=COMMENTS.POSTID\nGROUP BY COMMENTS.POSTID\nORDER BY
COUNT(COMMENTS.POSTID) DESC\nLIMIT 10\n',),
 ('\nSELECT USERS.ID, COMMENTS.USERID, USERS.REPUTATION, COUNT(COMMENTS.USERID)
AS TOTALCOMMENTS\nFROM USERS\nJOIN COMMENTS ON USERS.ID=COMMENTS.USERID\nGROUP
```

```
BY COMMENTS.USERID\nORDER BY COUNT(COMMENTS.USERID) DESC\nLIMIT 10\n',),
 ('\nSELECT USERS.ID, USERS.REPUTATION, COUNT(POSTS.OWNERUSERID) AS
TOTALPOSTS\nFROM USERS\nJOIN POSTS ON USERS.ID=POSTS.OWNERUSERID\nGROUP BY
POSTS.OWNERUSERID\nORDER BY USERS.REPUTATION DESC\nLIMIT 10\n',)]
```

10.4 Now Your Turn

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

[113]:
```
%%sql
SELECT query
    FROM queries
        WHERE query LIKE '%DISTINCT%'
```

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

[113]:
```
[('\nSELECT COUNT (DISTINCT(LOCATION)) AS NUMBEROFLOCATIONS\nFROM USERS\nWHERE
LOCATION IS NOT NULL\n',)]
```

11 Close SQLite

[114]:
```
con.close()
```

12 All Done!

[ ]:
```

```

[ ]:
```

```