STA 141B_HW3

SQL STATS EXCHANGE

```
library(RSQLite)
library(DBI)
library(lubridate)
library(sqldf)
library(dplyr)
```

CONNECT TO POSTTYPEID MAP

```
# Set working directory
setwd("~/Documents/UCD Classes/22-23/Spring Q23 Classes/STA 141B/HW3/HW3_Stuff/Schema:Descriptions")
# Connect to Database
db = dbConnect(SQLite(), "stats.stackexchange.db")
```

```
# Get tables
dbListTables(db)
```

```
##  [1] "BadgeClassMap"     "Badges"            "CloseReasonMap"
##  [4] "Comments"          "LinkTypeMap"       "PostHistory"
##  [7] "PostHistoryTypeId" "PostLinks"         "PostTypeIdMap"
## [10] "Posts"             "TagPosts"          "Users"
## [13] "VoteTypeMap"       "Votes"
```

Questions

1. How many users are there?

```
dbGetQuery(db, "SELECT COUNT(DISTINCT Id) as USERS FROM Users")
```

```
##    USERS
## 1 321677
```

A user can be interpreted in various ways. I interpreted as the number of unique user IDs, so I used COUNT DISTINCT ID to query the number of users. I found 321677 different users.

Tables Used: Users

2. How many users joined since 2020? (Hint: Convert the CreationDate to a year.)

```
dbGetQuery(db, "SELECT COUNT(*) AS Users_2020, strftime('%Y-%m-%d %H:%M:%f', CreationDate) AS CreationDa
           strftime('%Y', CreationDate) AS Year, Id
           FROM Users WHERE
           Year >= '2020'")
```

```
##   Users_2020            CreationDate Year     Id
## 1     100796 2020-01-01 01:06:38.380 2020 269763
```

I used strftime to extract the year, and found that 100796 users joined since 2020.

Tables Used: Users

3. How many users joined each year? Describe this with a plot, commenting on any anomalies.

I counted the total tuples in Users and grouped the tuples by Year.

```
user_per_year = dbGetQuery(db, "SELECT COUNT(*) AS Users_2020, strftime('%Y', CreationDate) AS Year
                           FROM Users
                           GROUP BY Year")
```

```
user_per_year
```

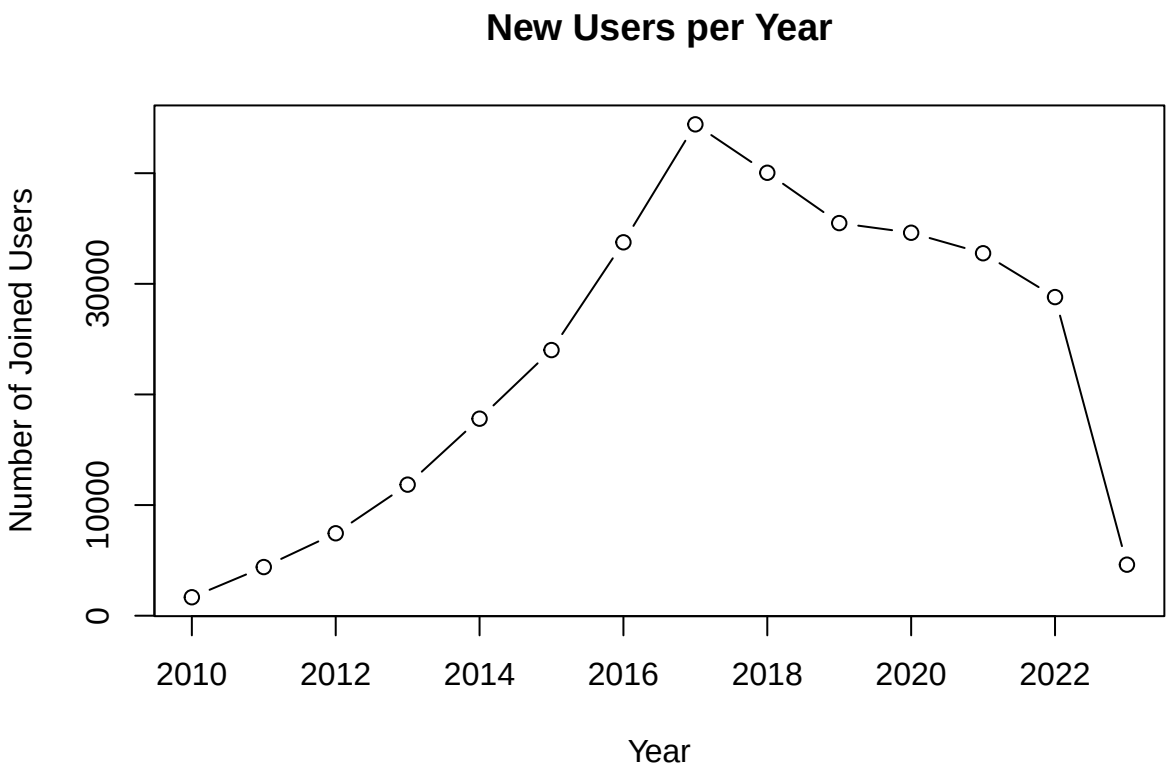
```
##    Users_2020 Year
## 1        1668 2010
## 2        4396 2011
## 3        7450 2012
## 4       11846 2013
## 5       17809 2014
## 6       24012 2015
## 7       33753 2016
## 8       44416 2017
## 9       40040 2018
## 10      35491 2019
## 11      34617 2020
## 12      32765 2021
## 13      28801 2022
## 14       4613 2023
```

```
# Double check all users are groups = 321677
sum(user_per_year$Users_2020)
```

```
## [1] 321677
```

I grouped the users by year and counted the frequency of users, and then double checked if the original number of users was the same after the data manipulation.

```
plot(user_per_year$Year,user_per_year$Users_2020, xlab = "Year", ylab = "Number of Joined Users", main =
```



From the plot above, we observe that there was a huge drop in users that joined between 2022-2023.

Tables Used: Users

4. How many different types of posts are there in the Posts table? Get the description of the types from the PostTypeIdMap table. In other words, create a table with the description of each post type and the number of posts of that type, and arrange it from most to least occurrences.

I found the different types of posts in the Posts table by querying the DISTINCT PostTypeIds. In the POSTS table, I found 7 different types of posts. However, I noticed that the schema showed that there are actually 8 different types of posts.

```
# Different types of Posts
dbGetQuery(db, "SELECT DISTINCT PostTypeId FROM Posts")
```

```
##   PostTypeId
## 1          1
## 2          2
## 3          7
## 4          5
## 5          4
## 6          6
## 7          3
```

Using the PostTypeIdMap table, I found 8 distinct descriptions of the post types.

```
# Different types of descriptions
dbGetQuery(db, "SELECT DISTINCT value FROM PostTypeIdMap")
```

```
##                                                        value
## 1                                                   Question
## 2                                                     Answer
## 3                                          Orphaned tag wiki
## 4                                           Tag wiki excerpt
## 5                                                   Tag wiki
## 6                                         Moderator nomination
## 7 Wiki placeholder (seems to only be the election description)
## 8                                              Privilege wiki
```

The count (8) in the PostTypeIdMap is not congruent to the count of PostTypeIds (7) in the Posts table. Notice how there is no PostTypeId = 8 in the Posts table. From this, I reasoned that the Posts table did not contain any posts with PostTypeId = 8 and the corresponding description of "Privilege wiki".

```
dbGetQuery(db,"SELECT * FROM Posts
           WHERE PostTypeId = 8")
```

```
##  [1] Id                    PostTypeId            AcceptedAnswerId
##  [4] CreationDate          Score                 ViewCount
##  [7] Body                  OwnerUserId           LastActivityDate
## [10] Title                 Tags                  AnswerCount
## [13] CommentCount          ContentLicense        LastEditorDisplayName
## [16] LastEditDate          LastEditorUserId      CommunityOwnedDate
## [19] ParentId              OwnerDisplayName      ClosedDate
## [22] FavoriteCount
## <0 rows> (or 0-length row.names)
```

We see that this is true. To get the number of posts for each description, I used COUNT to count the total number of rows and GROUP BY PostTypeId to group the number of counts with its corresponding PostTypeId.

```
# Number of posts for description
dbGetQuery(db, "SELECT COUNT(*) AS Posts, PostTypeId, value FROM Posts
           JOIN PostTypeIdMap
           ON Posts.PostTypeId = PostTypeIdMap.id GROUP BY PostTypeId ORDER BY Posts DESC")
```

```
##    Posts PostTypeId
## 1 204370          1
## 2 197928          2
## 3   1444          5
## 4   1444          4
## 5     23          6
## 6      6          3
## 7      5          7
##                                                        value
## 1                                                   Question
## 2                                                     Answer
## 3                                                   Tag wiki
```

```
## 4                                                 Tag wiki excerpt
## 5                                               Moderator nomination
## 6                                                  Orphaned tag wiki
## 7 Wiki placeholder (seems to only be the election description)
```

Tables Used: Posts, PostTypeIdMap

5. How many posted questions are there?

```
dbGetQuery(db, "SELECT COUNT(PostTypeId) AS Questions FROM Posts
           JOIN PostTypeIdMap
           ON Posts.PostTypeId = PostTypeIdMap.Id
           WHERE PostTypeIdMap.Id = 1")
```

```
##   Questions
## 1    204370
```

We want to double check that the PostTypeId = 1 in Posts is actually a question. To do this, I joined PostTypeIdMap to Posts by the corresponding Id link, and then found the answer by finding the joined tuples that had Id = 1 for PostTypeIdMap. I found 204370 counts of type 'question'. Another way to confirm this is by using the last question: there are also 204370 counts of questions.

Tables Used: Posts, PostTypeIdMap

6. What are the top 50 most common tags on questions? For each of the top 50 tags on questions, how many questions are there for each tag.

```
dbGetQuery(db, "SELECT Tag, COUNT(*) AS Tag_Count
           FROM TagPosts
           GROUP BY Tag
           ORDER BY Tag_Count DESC LIMIT 5")
```

```
##                Tag Tag_Count
## 1                r     28495
## 2       regression     28146
## 3 machine-learning     19355
## 4      time-series     13745
## 5      probability     11894
```

I found the tags first by using the TagPosts table. The table includes the question ID and its corresponding tag(names). In order to find the most common tags, all I needed to do was to create a new column for counting the total number of tags, group by the distinct tags, and find which tags had the greatest counts. For finding questions for each tag, it is given that each tag is associated with a unique question ID, so the number of questions is just equal to the tag count.

Tables Used: TagPosts

7. How many tags are in most questions?

```
number_tags = dbGetQuery(db, "SELECT COUNT(Tag) AS Number_Tags,Id as Question,Tag AS TagName FROM TagPos
           GROUP BY Id
           ORDER BY Number_Tags DESC")

head(number_tags)
```

```
##   Number_Tags Question             TagName
## 1           5   608403               arima
## 2           5   608395 confidence-interval
## 3           5   608380                   r
## 4           5   608379    machine-learning
## 5           5   608371         probability
## 6           5   608359            logistic
```

What is the range of tag counts for all the individual questions? I used COUNT(Tag) and ID and then grouped by ID to get the tag counts for every individual ID/question. To find the range of tag counts, I used the table function and found that there are only 1-5 tags associated with every question.

```
table(number_tags$Number_Tags)
```

```
##
##     1     2     3     4     5
## 20163 49278 59782 43249 31898
```

```
max(table(number_tags$Number_Tags))
```

```
## [1] 59782
```

Lastly, to find the most frequent number of tags in most questions, I used table() to count the frequencies of number of tags and chose the one with the greatest count by max(). I found that in most questions (59782), there are 3 tags.

Tables Used: TagPosts

8.) How many answers are there?

```
dbGetQuery(db, "SELECT COUNT(*) AS Answers FROM Posts
           JOIN PostTypeIdMap
           ON Posts.PostTypeId = PostTypeIdMap.Id
           WHERE PostTypeIdMap.Id = 2")
```

```
##   Answers
## 1  197928
```

This is similar to question 5. I did the same thing but set the PostTypeIdMap.Id = 2, which corresponds to Answer. I found 197928 answers.

Tables Used: Posts, PostTypeIdMap

9.) What's the most recent question (by date-time) in the Posts table?

```
dbGetQuery(db, "SELECT strftime('%Y-%m-%d %H:%M:%f', CreationDate) AS CreationDate, Title, Id FROM Posts
           ORDER BY CreationDate DESC
           LIMIT 1")
```

```
##              CreationDate
## 1 2023-03-05 05:10:18.393
##                                                                    Title
## 1 Are there any papers or methods that combine mcmc and variational inference
##       Id
## 1 608405
```

I found the most recent question by using the CreationDate column in Posts, which relates to the date a question was created. I ordered the CreationDate in descending order, and found that the most recent question in the Posts table was "Are there any papers or methods that combine mcmc and variational inference".

Tables Used: Posts

- **Find it on the stats.exchange.com Web site and provide the URL.**

https://stats.stackexchange.com/questions/608405/are-there-any-papers-or-methods-that-combine-mcmc-and-variational-inference

I found this by simply using the base url: https://stats.stackexchange.com and then provided the PostId to find the question. This actually leads to a 404 and said that "This question was voluntarily removed by its author".

- **How would we map a question in the Posts table to the corresponding SO URL?**

To map a question in the Posts table to the StackOverflow URL we would just put in the QuestionId into the base URL followed by the path questions/.

For example, if we wanted to map the question "Eliciting priors from experts", which corresponds to a PostId of 1, we would simply do:

https://stats.stackexchange.com/questions/1

10.) For the 10 users who posted the most questions:

- How many questions did they post? • What are the users' names?
- When did they join SO?
- What is their Reputation?
- What country do they have in their profile?

Note: Instead of answering the sub-questions one by one, I kept adding query commands to the one chunk of code below. All the answers to the questions can be found in the data frame below.

```
top_10 = dbGetQuery(db, "SELECT COUNT(PostTypeId) AS Questions, Users.Id, DisplayName, Users.CreationDa
FROM Posts
           JOIN Users
           ON Posts.OwnerUserId = Users.Id
           WHERE PostTypeId = 1
           GROUP BY Users.Id
           ORDER BY Questions DESC
           LIMIT 10")

top_10
```

```
##    Questions     Id                 DisplayName             Joined_Date
## 1        349  77179                  stats_noob 2015-05-14T21:12:31.790
## 2        298   1005                         Tim 2010-08-19T15:31:09.537
## 3        264   9162 user1205901 -          2012-02-13T02:09:08.377
## 4        255  53690               Richard Hardy 2014-08-08T10:57:13.613
## 5        236 108150                  user321627 2016-03-10T14:45:28.010
## 6        192 113777                   Haitao Du 2016-04-27T20:51:38.203
## 7        184  28986              Charlie Parker 2013-08-09T19:20:37.540
## 8        180  40252      An old man in the sea. 2014-02-14T13:18:39.917
## 9        166 163242                The Pointer 2017-05-30T00:13:41.380
## 10       164  56211                       rnso 2014-09-22T08:35:18.697
```

```
##    Reputation Country
## 1           1
## 2       18497
## 3       11859
## 4       60742  Europe
## 5        2478
## 6       34665
## 7        6286
## 8        5330
## 9        1344
## 10       9299
```

Since the User.Id and PostTypeId are in different tables, I joined the User and Posts table and matched the corresponding links. This gave back a table that included the User.Id if it was in Post.OwnerUserId, and ensured that the User who asked the question matched the correct question post. After, I counted the number of posts where PostTypeId = 1 for a given User.Id and grouped by User.Id, which gave me the question counts per user. The other sub-questions were relatively straightforward.

Tables Used: Posts, Users

11.) Following from the previous questions, for the 10 users who posted the most questions, how many gold, silver and bronze badges does each of these 10 individuals have?

```
badges = dbGetQuery(db, "SELECT Badges.Id,UserId,Class,value,Name FROM Badges
                    JOIN BadgeClassMap
                    ON Badges.Class = BadgeClassMap.id")
head(badges,3)
```

```
##   Id UserId Class  value    Name
## 1  1      5     3 Bronze Teacher
## 2  2      6     3 Bronze Teacher
## 3  3      8     3 Bronze Teacher
```

My first idea was to join the Badges and Top_10_Users table (from Question 10) to get the Usernames corresponding to the badges. Before I did that, I created a new d.f named badges that classified the classes of badges (1,2,3) as Gold,Silver,and Bronze, respectively. After this, I constructed the query below

Tables Used: Badges, BadgeClassMap

```
sqldf("SELECT DisplayName, top_10.Id AS User_Id, value, COUNT(value) AS Count
FROM top_10
      JOIN badges
      ON top_10.Id = badges.UserId
      GROUP BY top_10.Id, value
      ORDER BY DisplayName")
```

```
##                    DisplayName User_Id  value Count
## 1       An old man in the sea.   40252 Bronze    65
## 2       An old man in the sea.   40252   Gold     4
## 3       An old man in the sea.   40252 Silver    27
## 4               Charlie Parker   28986 Bronze   122
## 5               Charlie Parker   28986   Gold    13
## 6               Charlie Parker   28986 Silver    65
## 7                    Haitao Du  113777 Bronze   228
```

```
## 8                    Haitao Du  113777   Gold    19
## 9                    Haitao Du  113777 Silver   130
## 10               Richard Hardy   53690 Bronze   237
## 11               Richard Hardy   53690   Gold    12
## 12               Richard Hardy   53690 Silver   114
## 13                 The Pointer  163242 Bronze    44
## 14                 The Pointer  163242 Silver    22
## 15                         Tim    1005 Bronze   205
## 16                         Tim    1005   Gold    32
## 17                         Tim    1005 Silver   116
## 18                        rnso   56211 Bronze   109
## 19                        rnso   56211   Gold    18
## 20                        rnso   56211 Silver    59
## 21                  stats_noob   77179 Bronze    68
## 22                  stats_noob   77179   Gold     2
## 23                  stats_noob   77179 Silver    32
## 24 user1205901 -            9162 Bronze   161
## 25 user1205901 -            9162   Gold    26
## 26 user1205901 -            9162 Silver    88
## 27                  user321627  108150 Bronze    63
## 28                  user321627  108150   Gold     4
## 29                  user321627  108150 Silver    20
```

I joined the Top_10_Users with the Badges on their linked Ids. To find the distribution of Gold,Silver,and Bronze badges for the 10 unique users, I counted the total number of badges, and grouped by the User.Id and Badge Type. This returns a table that includes the 10 users with their respective badge counts.

Tables Used: Badges, Top_10

12.) For each of the following terms, how many questions contain that term: Regression, ANOVA, Data Mining, Machine Learning, Deep Learning, Neural Network.

```
dbGetQuery(db, "SELECT
  SUM(Title LIKE '%Regression%') AS Regression,
  SUM(Title LIKE '%ANOVA%') AS ANOVA,
  SUM(Title LIKE '%Data Mining%') AS Data_Mining,
  SUM(Title LIKE '%Machine Learning%') AS Machine_Learning,
  SUM(Title LIKE '%Deep Learning%') AS Deep_Learning,
  SUM(Title LIKE '%Neural Network%') AS Neural_Network
FROM Posts")
```

```
##   Regression ANOVA Data_Mining Machine_Learning Deep_Learning Neural_Network
## 1      21886  3111         117             1716           447           2703
```

I collected the sum of titles/questions that contained the exact specified term.

Tables Used: Posts

13.) Using the Posts and PostLinks tables, how many questions gave rise to a "related" or "duplicate" question?

According to the schema, "related" or "duplicated" questions are classified as LinkType.Id = 1 and 3, respectively. With this information, I found the answer by joining the Posts and PostLinks table by Id, made sure that the PostType was a question, and grouped by LinkType.

```
dbGetQuery(db, "SELECT LinkTypeId, COUNT(*) AS Count
FROM Posts
           JOIN PostLinks
           ON Posts.Id = PostLinks.PostId
           WHERE PostTypeId = 1
           GROUP BY LinkTypeId")
```

```
##   LinkTypeId Count
## 1          1 71175
## 2          3  8980
```

Tables Used: Posts, PostLinks

- **And how many responses did these questions get?**

```
dbGetQuery(db, "SELECT DISTINCT Posts.Id,Posts.Title,LinkTypeId,AnswerCount+CommentCount AS Responses
FROM Posts
           JOIN PostLinks
           ON Posts.Id = PostLinks.PostId
           WHERE PostTypeId = 1
           ORDER BY Responses DESC
           LIMIT 5")
```

```
##       Id
## 1    726
## 2    423
## 3   1337
## 4  41208
## 5 328630
##                                                                                Title
## 1                                                  Famous statistical quotations
## 2                                What is your favorite "data analysis" cartoon?
## 3                                                              Statistics Jokes
## 4                                                The Sleeping Beauty Paradox
## 5 Is ridge regression useless in high dimensions ($n \\ll p$)? How can OLS fail to overfit?
##   LinkTypeId Responses
## 1          1       154
## 2          1        89
## 3          1        81
## 4          1        60
## 5          1        60
```

I added up the answer and comment counts to measure the response count for a question. I grouped the tuples by responses, from highest to lowest.

Tables Used: Posts, PostLinks

- **How experienced were the users posting these** questions**.**

```
dbGetQuery(db, "SELECT Posts.Id,Posts.Title,Users.Reputation,Users.Upvotes
FROM Posts
           JOIN PostLinks
           ON Posts.Id = PostLinks.PostId
```

```
           JOIN Users
           ON Posts.OwnerUserId = Users.Id
           WHERE PostTypeId = 1
           ORDER BY Reputation DESC
           LIMIT 5")
```

```
##       Id                                                  Title Reputation
## 1  41208                              The Sleeping Beauty Paradox     304878
## 2   1963 Looking for good introductory treatment of meta-analysis     304878
## 3 204843             Is this the solution to the p-value problem?     304878
## 4 415435           How does entropy depend on location and scale?     304878
## 5 204843             Is this the solution to the p-value problem?     304878
##   UpVotes
## 1   32184
## 2   32184
## 3   32184
## 4   32184
## 5   32184
```

I measured experience through User Reputation and Upvotes. The values above show the most experienced users that hold the highest reputation.

Tables Used: Posts, PostLinks, Users

14.) What is the date range for the questions and answers in this database?

```
# Dates for Questions
dbGetQuery(
  db,
  "SELECT PostTypeId, MIN(strftime('%Y-%m-%d %H:%M:%f', CreationDate)) AS Start,
                      MAX(strftime('%Y-%m-%d %H:%M:%f', CreationDate)) AS End
          FROM Posts
            GROUP BY PostTypeId
  LIMIT 2"
)
```

```
##   PostTypeId                   Start                     End
## 1          1 2009-02-02 14:21:12.103 2023-03-05 05:10:18.393
## 2          2 2009-02-02 14:24:31.740 2023-03-05 04:48:34.853
```

I assume that date ranges means the range of the first and last occurrence of a question or answer in the database. I first used strftime on CreationDate, then used the MIN and MAX functions to get the range. Then, I grouped by PostTypeId to get the date ranges for all the types, and limited it to the first 2 tuples, as they corresponded to PostTypeId = 1,2, which represent question and answer.

Tables Used: Posts

15.) What question has the most comments associated with it?

```
dbGetQuery(db, "SELECT Title, COUNT(*) AS Comment FROM Comments
           JOIN Posts
           ON Comments.PostId = Posts.Id
           WHERE PostTypeId = 1
           GROUP BY Title
           ORDER BY Comment DESC
           LIMIT 1")
```

```
##                                                                                 Title
## 1 Is ridge regression useless in high dimensions ($n \\ll p$)? How can OLS fail to overfit?
##   Comment
## 1      54
```

First I wanted to join the Comments and Posts table to get the Question Title and comment count. I counted the number of tuples in comment and grouped by Title/question to get the comment counts per question. To get the questions with the most comments, I first set PostTypeId = 1 (question), and then ordered the count by descending and limited the display to 1 tuple.

Tables Used: Comments, Posts

- **How many answers are there for this question?**

```
dbGetQuery(db, "SELECT AnswerCount FROM Posts
           WHERE Title = 'Is ridge regression useless in high dimensions ($n \\ll p$)? How can OLS fail
```

```
##   AnswerCount
## 1           6
```

Tables Used: Posts

16.) How many comments are there across all posts?

```
dbGetQuery(db, "SELECT SUM(CommentCount) AS Comments FROM Posts")
```

```
##   Comments
## 1   768069
```

I used SUM(CommentCount) (instead of COUNT because CommentCount contains values > 1) to compute the total amount of comments for all PostTypeIds.

- **How many posts have a comment?**

```
dbGetQuery(db, "SELECT COUNT(DISTINCT Id) AS Post_With_Comment FROM Posts
           WHERE CommentCount > 0")
```

```
##   Post_With_Comment
## 1            229859
```

- **What is the distribution of comments per question?**

First, I wanted to find the comment counts for every unique question. I verified that all of the Titles were questions by setting PostTypeId = 1

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.1.2
```

```
distribution_coques = dbGetQuery(db, "SELECT DISTINCT Title,Posts.Id,CommentCount,Posts.PostTypeId FROM
           JOIN Posts ON
           Posts.Id = Comments.PostId
           WHERE Title is NOT NULL AND
           Title <> '' AND
           PostTypeId = 1
           ORDER BY Posts.Id")

# Look at CommentCount per Question
head(distribution_coques)
```

```
##                                                                  Title Id
## 1                                    Eliciting priors from experts  1
## 2                                                What is normality?  2
## 3 What are some valuable Statistical Analysis open source projects?  3
## 4        Assessing the significance of differences in distributions  4
## 5                The Two Cultures: statistics vs. machine learning?  6
## 6                           Locating freely available data samples  7
##   CommentCount PostTypeId
## 1            1          1
## 2            1          1
## 3            3          1
## 4            2          1
## 5           10          1
## 6            2          1
```

Verification for all Posts having PostTypeId = 1

```
# Checking if all are questions == 1
unique(distribution_coques$PostTypeId)
```

```
## [1] 1
```

```
# How many Questions have comments
nrow(distribution_coques)
```

```
## [1] 128280
```

To display the distribution of comments per question, I used a table:

```
# How many comments in number of questions
table(distribution_coques$CommentCount)
```

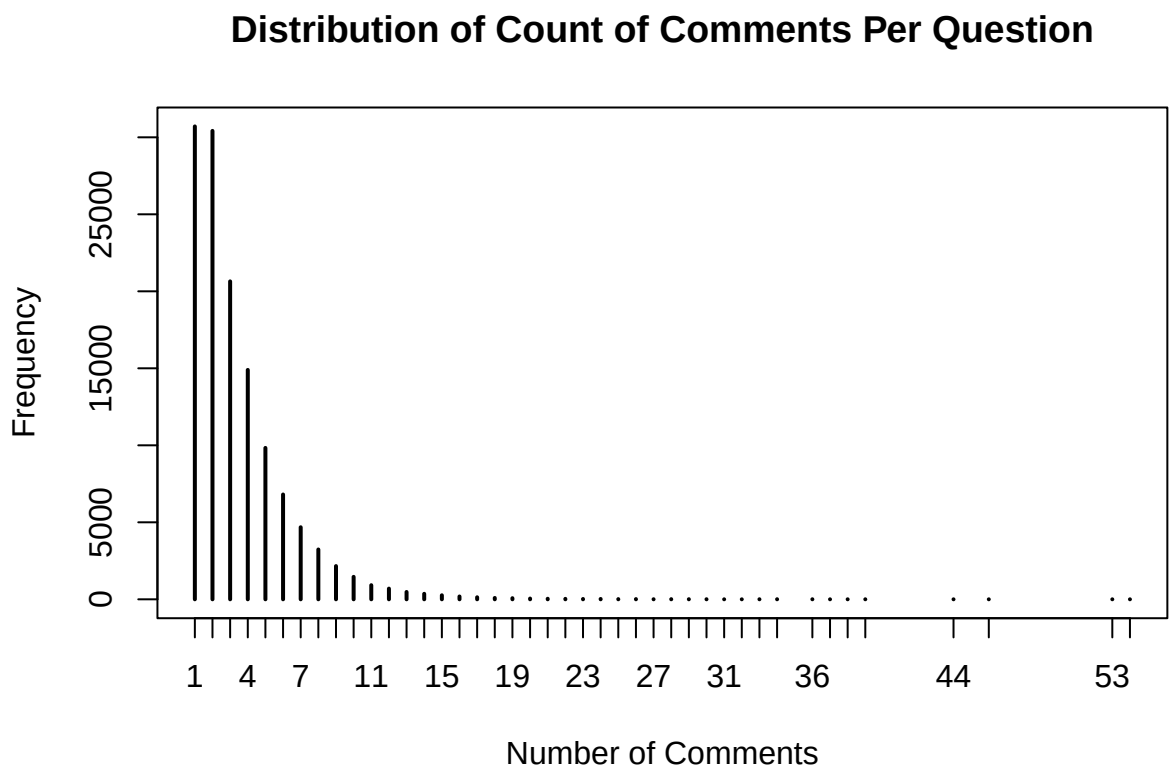
```
##
##     1     2     3     4     5     6     7     8     9    10    11    12    13
## 30707 30417 20645 14897  9832  6806  4680  3236  2158  1459   918   695   477
##    14    15    16    17    18    19    20    21    22    23    24    25    26
##   357   262   185   133    90    74    49    38    31    22    24    14    15
##    27    28    29    30    31    32    33    34    36    37    38    39    44
##     6     9     8     6     2     2     5     6     4     1     3     3     1
##    46    53    54
##     1     1     1
```

```
# Double check counts in table = # of comments for questions
sum(table(distribution_coques$CommentCount))
```

```
## [1] 128280
```

I also graphed the result for more clarity on the distribution

```
# Graphically
plot(table(distribution_coques$CommentCount), xlab = "Number of Comments", ylab = "Frequency",
     main = "Distribution of Count of Comments Per Question")
```



From the histogram above, it is clearly right skewed. Most of the data lies on the left side, which indicates that there was a greater frequency of number of comments in a smaller amount of comments.

Tables Used: Comments, Posts

17.) Is there any relationship between the number of tags on a question, the length of a question, and the number of responses (posts and comments)?

```
relationship = dbGetQuery(db, "SELECT COUNT(Tag) AS Tags, LENGTH(Title) AS Length, AnswerCount, CommentC
FROM Posts
           JOIN TagPosts
           ON Posts.Id = TagPosts.Id
           GROUP BY TagPosts.Id
           ORDER BY Tags,Length")
```

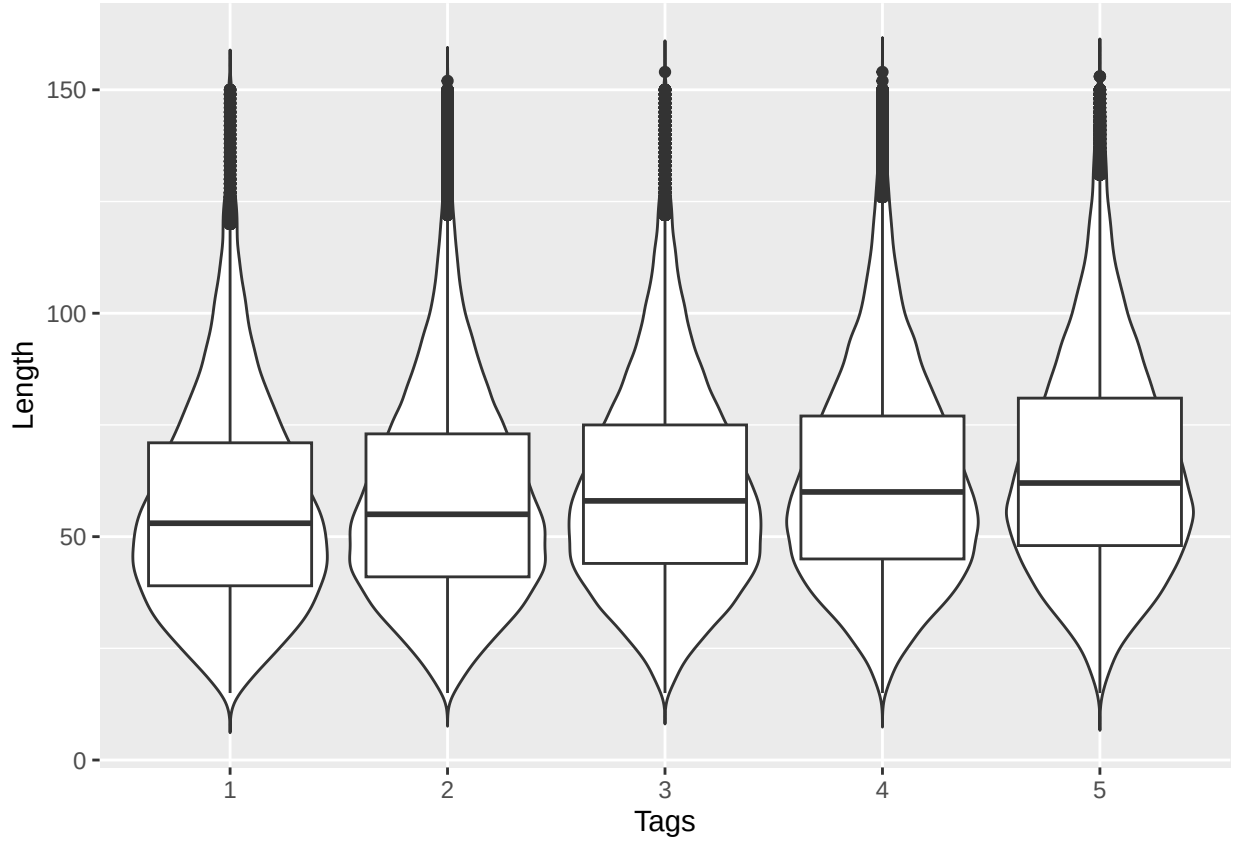
```
# As Factors
relationship$Tags = as.factor(relationship$Tags)
```

I imagine that for this question, the independent variable is number of tags, and the dependent variables being the length and number of responses. To show the relationship(s), I needed to extract the columns for number of tags, length, and responses on a question. In addition, because Tags is the independent variable, I changed their class to Factors for plotting purposes.

Tables Used: Posts, TagPosts

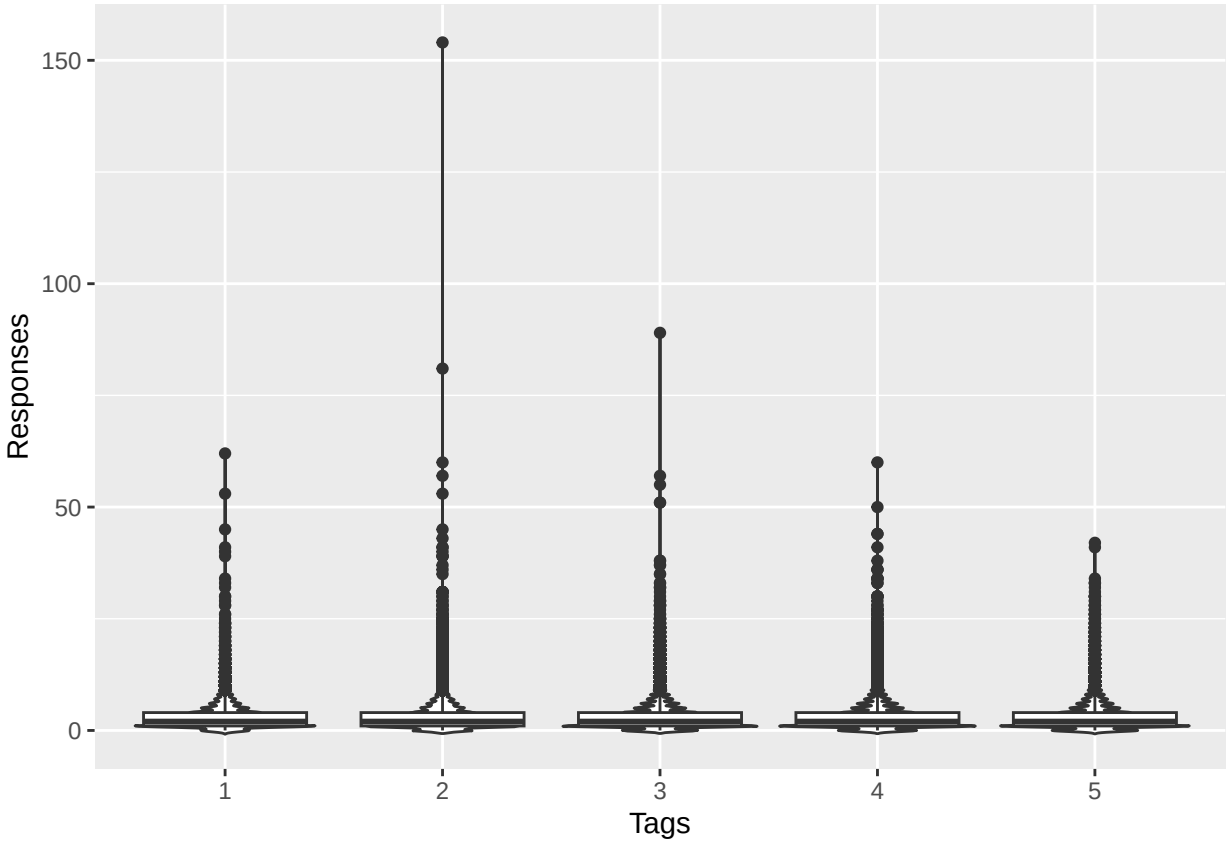
To test the relationship, I plotted the graphs with the appropriate axis.

```
# Tags vs Length
ggplot(relationship, aes(x = Tags, y = Length)) +
  geom_violin(trim = FALSE) +
  geom_boxplot()
```



There seems to be no relationship between the number of tags and length of a question (the boxplots and the median among each tag are almost identical).

```
# Tags vs All Responses
ggplot(relationship, aes(x = Tags, y = Responses))+
  geom_violin(trim = FALSE) +
  geom_boxplot()
```



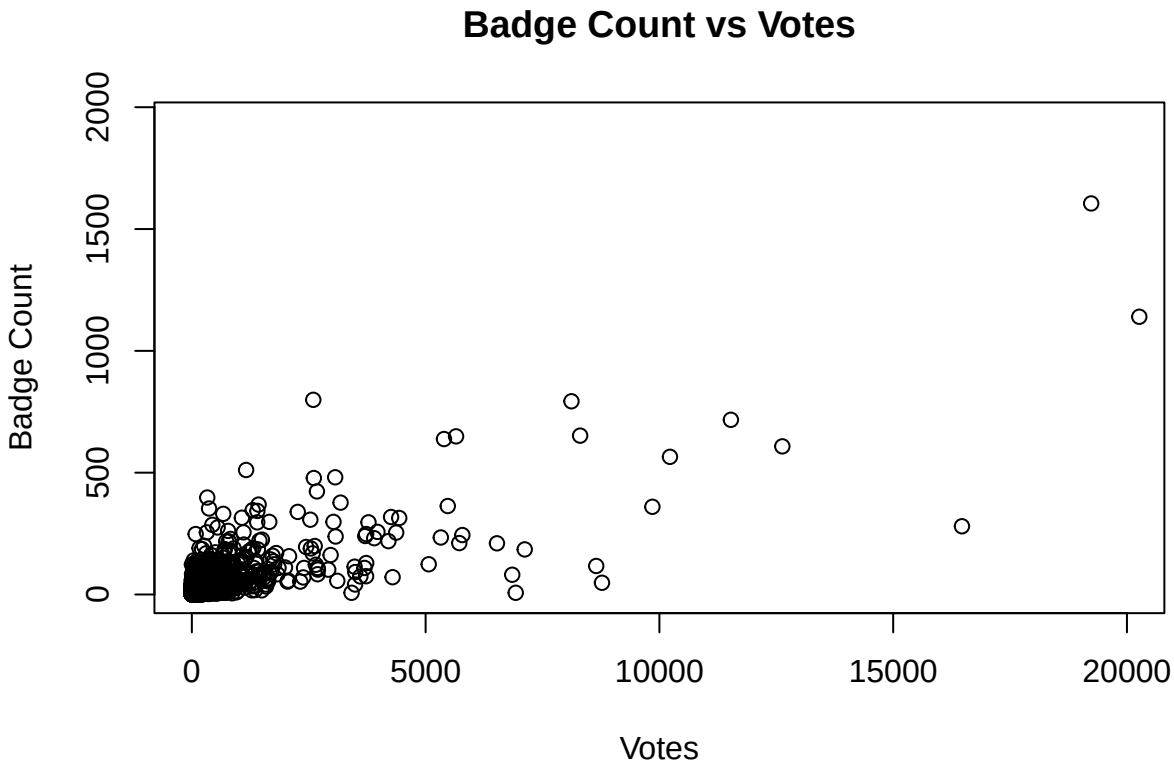
There does not seem to be much of a correlation between tag counts and the number of responses for a question either. We see that tags 2 and 3, there are outliers, however in general most of the data points for all tags have the same distribution.

18.) Do the people who vote tend to have badges?

```
vote_badges = dbGetQuery(db, "SELECT Users.Id, DisplayName, UpVotes, DownVotes, UpVotes + DownVotes AS T
           LEFT JOIN Users
           ON Users.Id = Badges.UserId
           JOIN BadgeClassMap
           ON Badges.Class = BadgeClassMap.id
           GROUP BY Users.Id")
```

Plot to display relationship below.

```
plot(vote_badges$Total_Votes, vote_badges$Badges, xlim = c(0,20000),
     xlab = "Votes", ylab = "Badge Count", main = "Badge Count vs Votes")
```



There is somewhat of a correlation. Kinda....

Tables Used: Badges, Users, BadgeClassMap

20.) How many posts have multiple different people who edit it?

```
posts_multiple_edits = dbGetQuery(db, "SELECT PostHistoryTypeId,PostId,COUNT(DISTINCT UserId) AS Multipl
           WHERE PostHistoryTypeId IN ('4','5','6')
           GROUP BY PostId
           HAVING Multiple_Editors > 1")

length(posts_multiple_edits$PostId)
```

```
## [1] 54293
```

I used the PostHistory table and extracted the unique Ids that corresponded to any form of editing. For example in the table, PostHistoryTypeId 4 had the value of "Edit Title." From this logic, I extracted all the IDs that were related to a type of edit and extracted it. I grouped by the PostId and found the values that had multiple editors. In total, I found 54293 posts with multiple editors.

Tables Used: PostHistory

Required Questions

21.) Compute the table that contains • the question, • the name of the user who posted it, • when that user joined, • their location • the date the question was first posted, • the accepted answer, • when the accepted answer was posted • the name of the user who provided the accepted answer.

```
users = dbGetQuery(db,"SELECT * FROM Users")
```

```
## Warning: Column `AccountId`: mixed type, first seen values of type integer,
## coercing other values of type string
```

```
dbGetQuery(db, "SELECT Q.Title,U.DisplayName,U.CreationDate,U.Location,Q.CreationDate,A.Body,A.CreationD
FROM Posts AS A, Posts AS Q, Users AS U, Users AS Provider
           WHERE A.Id = Q.AcceptedAnswerId AND
           U.Id = Q.OwnerUserId AND
           Provider.Id = A.OwnerUserId AND
           Q.PostTypeId = 1
           LIMIT 10")
```

```
##                                                                                Title
## 1                                                 Eliciting priors from experts
## 2                                                       What is normality?
## 3               What are some valuable Statistical Analysis open source projects?
## 4                     Assessing the significance of differences in distributions
## 5                                       Locating freely available data samples
## 6  Under what conditions should Likert scales be used as ordinal or interval data?
## 7                                    Multivariate Interpolation Approaches
## 8                                           Finding the PDF given the CDF
## 9                                 Tools for modeling financial time series
## 10                                          What is a standard deviation?
##             DisplayName            CreationDate                    Location
## 1           csgillespie 2010-07-19T19:04:52.280   Newcastle, United Kingdom
## 2                A Lion 2010-07-19T19:09:32.157
## 3                grokus 2010-07-19T19:08:29.070               United States
## 4           Jay Stevens 2010-07-19T19:09:16.917       Jacksonville, FL, USA
## 5                EAMann 2010-07-19T19:11:57.393 Tualatin, OR, United States
## 6                A Lion 2010-07-19T19:09:32.157
## 7   Christopher D. Long 2010-07-19T19:11:11.093              Versailles, KY
## 8  Mehper C. Palavuzlar 2010-07-19T19:22:10.957           Istanbul, Turkiye
## 9  Mehper C. Palavuzlar 2010-07-19T19:22:10.957           Istanbul, Turkiye
## 10         Oren Hizkiya 2010-07-19T19:25:59.420                New York, NY
##               CreationDate
## 1  2010-07-19T19:12:12.510
## 2  2010-07-19T19:12:57.157
## 3  2010-07-19T19:13:28.577
## 4  2010-07-19T19:13:31.617
## 5  2010-07-19T19:15:59.303
## 6  2010-07-19T19:17:47.537
## 7  2010-07-19T19:18:30.810
## 8  2010-07-19T19:26:04.363
## 9  2010-07-19T19:27:13.503
## 10 2010-07-19T19:27:43.860
##
## 1
## 2
## 3
## 4
Smirnov test</a>, or the like.  The two-sample Kolmogorov-Smirnov test is based on comparing differences
## 5
```

```
## 6  <p>Maybe too late but I add my answer anyway...</p>\\\\n\\\\n<p>It depends on what you intend to
## 7
## 8
## 9
## 10
##                CreationDate    Answer_User
## 1  2010-07-19T19:19:46.160         Harlan
## 2  2010-07-19T19:43:20.423 John L. Taylor
## 3  2010-07-19T19:14:43.050    Jay Stevens
## 4  2010-07-19T21:36:12.850 John L. Taylor
## 5  2010-07-19T19:24:18.580 Stephen Turner
## 6  2010-08-19T10:00:00.370            chl
## 7  2010-08-03T21:50:09.007 Carlos Accioly
## 8  2010-07-19T20:15:54.823           Paul
## 9  2010-07-19T19:29:06.527          Shane
## 10 2010-07-19T19:44:35.037  Neil McGuigan
```

First: I needed the Users and Posts tables.

To extract the values, I mapped a Post.Id = A.AcceptedAnswerId, User.Id = OwnerUserId, and another User Id to the posts OwnerUserId.

First map: We want to map a Post to its corresponding accepted answer in order to retrieve the accepted answer column.

Second map: We want to map a User Id to the same Post OwnerUserId above to get the user that posted the question. From this, we can extract more information about the user, such as the username, date they joined, and their location.

Third map: Finally, we want to map a new User Id to the A.OwnerUserId in order to extract the CreationDate and username of the answer.

22.) Determine the users that have only posted questions and never answered a question? (Compute the table containing the number of questions, number of answers and the user's login name for this group.) How many are there?

```
dbGetQuery(db, "SELECT OwnerUserId,Users.DisplayName,SUM(PostTypeId = 1) AS Questions,SUM(PostTypeId = 1
           JOIN Users
           ON Posts.OwnerUserId = Users.Id
           GROUP BY OwnerUserId
           HAVING Questions > 0 AND
           Answers = 0
           ORDER BY Questions DESC
           LIMIT 5")
```

```
##   OwnerUserId DisplayName Questions Answers
## 1      108150  user321627       236       0
## 2      241460   The Great       133       0
## 3       12329   user34790       128       0
## 4       67137        Dave       121       0
## 5       42198         TEX       109       0
```

Thank you to Professor Duncan for explaining this to me. Because we are looking for user and posted questions and answers, I joined the Users and Posts table. I used the SUM() function to evaluate whether there existed PostTypeId = 1 or 2, which correspond to question or answer, for a particular user. To see the counts of question and answers per user, I grouped by user ID. Finally, I created a condition where the

user must have a question count > 0 and answer count = 0 to find which users posted a question and never answered. In total, there were 76,410 of these unique users (I only showed 5 above though).

Tables Used: Users, Posts

23.) Compute the table with information for the 75 users with the most accepted answers. This table should include • the user's display name, • creation date, • location, • the number of badges they have won, – the names of the badges (as a single string) • the dates of the earliest and most recent accepted answer (as two fields) – the (unique) tags for all the questions for which they had the accepted answer (as a single string)

I split this section into two parts: In the part below, I computed the dsername, user creation date, and user location. For this, I self-joined the posts table and used the users table.

Thought Process: I want to find the users that had the most accepted answers. To do this, I need the users and posts table. I first joined where User Id = OwnerUserId. I did this because this is one of the only and best way to link the User and Posts table. Now that I have the owner of the post, I did another join with another Post table to join where the Id of the Post = AcceptedAnswerId. This gave me which users had the accepted answer to a post. To find how many accepted answers a user had, I used COUNT:

```
dbGetQuery(db,"SELECT DISTINCT U.Id,U.DisplayName,COUNT(*) AS Accepted_Count
FROM Posts AS A, Posts AS Accepted, Users AS U
           WHERE U.Id = A.OwnerUserId AND
           A.Id = Accepted.AcceptedAnswerId
           GROUP BY U.Id
           ORDER BY Accepted_Count DESC
           LIMIT 5")
```

```
##       Id DisplayName Accepted_Count
## 1    805      Glen_b           2335
## 2    919      whuber           1781
## 3  28500         EdM           1246
## 4 204068       gunes           1119
## 5  35989         Tim           1004
```

Now that I have this, I extracted the relevant information and ordered by the top 75 users.

```
top_75 = dbGetQuery(db,"SELECT DISTINCT U.Id,Accepted.PostTypeId,COUNT(*) AS Accepted_Count,U.DisplayNa
FROM Posts AS A, Posts AS Accepted, Users AS U
           WHERE U.Id = A.OwnerUserId AND
           A.Id = Accepted.AcceptedAnswerId
           GROUP BY U.Id
           ORDER BY Accepted_Count DESC
           LIMIT 75")
```

A user can have multiple accepted answers, only if it is for different questions. To find the earliest accepted question for a user, we use MIN. To find the recent accepted question for a user, we use MAX.

Next, I calculated the other questions related to badges and tags for the latter half. For this, I needed to join the top_75 table I made with the badge table in order to map the correct user id with their badge id and retrieve the badge results,names,etc.

```
badges_75 = sqldf("SELECT GROUP_CONCAT(DISTINCT badges.Name) AS BadgeName, DisplayName, CreationDate, L
FROM top_75
      JOIN badges
```

```
      ON top_75.Id = badges.UserId
      GROUP BY top_75.Id
      ORDER BY Count DESC")
```

Here is how the final result looks like with the joined tables by User.Id.

```
sqldf("SELECT top_75.DisplayName,top_75.CreationDate AS UserCreationDate,top_75.Location,badges_75.Coun
      JOIN badges_75
      ON top_75.Id = badges_75.User_Id
      LIMIT 10")
```

```
##        DisplayName        UserCreationDate               Location
## 1           Glen_b 2010-08-07T08:40:07.287        I'm right here
## 2           whuber 2010-08-13T15:29:47.140
## 3              EdM 2013-07-26T15:11:03.380
## 4            gunes 2018-04-12T10:42:43.307          Cambridge, UK
## 5              Tim 2013-12-10T21:19:06.223         Warsaw, Poland
## 6  Stephan Kolassa 2010-09-18T10:55:08.240            Switzerland
## 7            Xi'an 2011-11-05T07:56:15.903          Paris, France
## 8    Richard Hardy 2014-08-08T10:57:13.613                Europe
## 9              Ben 2017-08-10T03:27:26.793    Canberra, Australia
## 10         BruceET 2015-08-11T17:22:01.590 San Francisco Bay Area
##    NumberofBadges
## 1            1605
## 2            1942
## 3             318
## 4             129
## 5             717
## 6             638
## 7             793
## 8             363
## 9             649
## 10            122
##
## 1  Altruist,Analytical,Announcer,Autobiographer,Benefactor,Caucus,Citizen Patrol,Cleanup,Commentator
## 2
## 3
## 4
## 5
## 6
## 7
## 8
## 9
## 10
##              EarliestAnswer          RecentAccepted
## 1  2010-08-09T00:37:45.090 2023-02-23T21:21:32.110
## 2  2010-08-19T13:07:22.083 2023-03-04T19:37:09.383
## 3  2013-08-13T20:01:37.963 2023-03-01T21:03:42.703
## 4  2018-09-09T17:27:12.530 2023-01-22T10:32:17.537
## 5  2014-10-31T11:05:38.017 2023-02-25T07:34:21.380
## 6  2010-09-23T20:42:45.613 2023-03-01T19:16:24.493
## 7  2011-11-05T21:11:34.147 2023-02-22T09:13:37.127
## 8  2014-10-19T11:52:22.160 2023-03-02T14:41:06.427
```

```
## 9  2018-01-17T05:37:49.037 2023-02-28T23:31:47.337
## 10 2016-10-04T05:59:47.963 2022-07-01T19:29:14.067
##                                                                                  Tags
## 1                             <multiple-regression><multiple-comparisons><permutation-test>
## 2             <data-visualization><random-variable><kernel-smoothing><circular-statistics>
## 3                                                 <r><survival><hazard><proportional-hazards>
## 4                               <machine-learning><neural-networks><unsupervised-learning>
## 5                                                       <classification><backpropagation>
## 6                                                             <time-series><seasonality>
## 7  <bayesian><model-selection><posterior><bayes-factors><approximate-bayesian-computation>
## 8                                       <r><time-series><forecasting><diebold-mariano-test>
## 9                                                                          <time-series>
## 10                              <confidence-interval><mean><simulation><online-algorithms>
```

I did have one error that I could not fix: Getting the correct count of tags.

Tables Used: Self-join on Posts, Users

24. How many questions received no answers (accepted or unaccepted)? How many questions had no accepted answer?

```
# Checked that DISTINCT AcceptedAnswerId is empty
dbGetQuery(db, "SELECT COUNT(*) AS No_Answer FROM Posts
           WHERE AnswerCount = 0 AND
           PostTypeId = 1")
```

```
##   No_Answer
## 1     66970
```

```
dbGetQuery(db, "SELECT COUNT(*) AS No_Accepted FROM Posts
           WHERE AcceptedAnswerId = '' AND
           PostTypeId = 1")
```

```
##   No_Accepted
## 1      136365
```

All the information can be gathered from the Posts table. Because the question specifically asks for question, I set a constraint such that PostTypeId = 1. To find how many questions received no answer or had no accepted answer, respectively, I found where ANSWERCOUNT = 0 and AcceptedAnswerId is empty -> ''.

Tables Used: Posts

25.) What is the distribution of answers per posted question?

This question is identical to Q16, where I had to find the distribution of comments for each question. I used the same process.

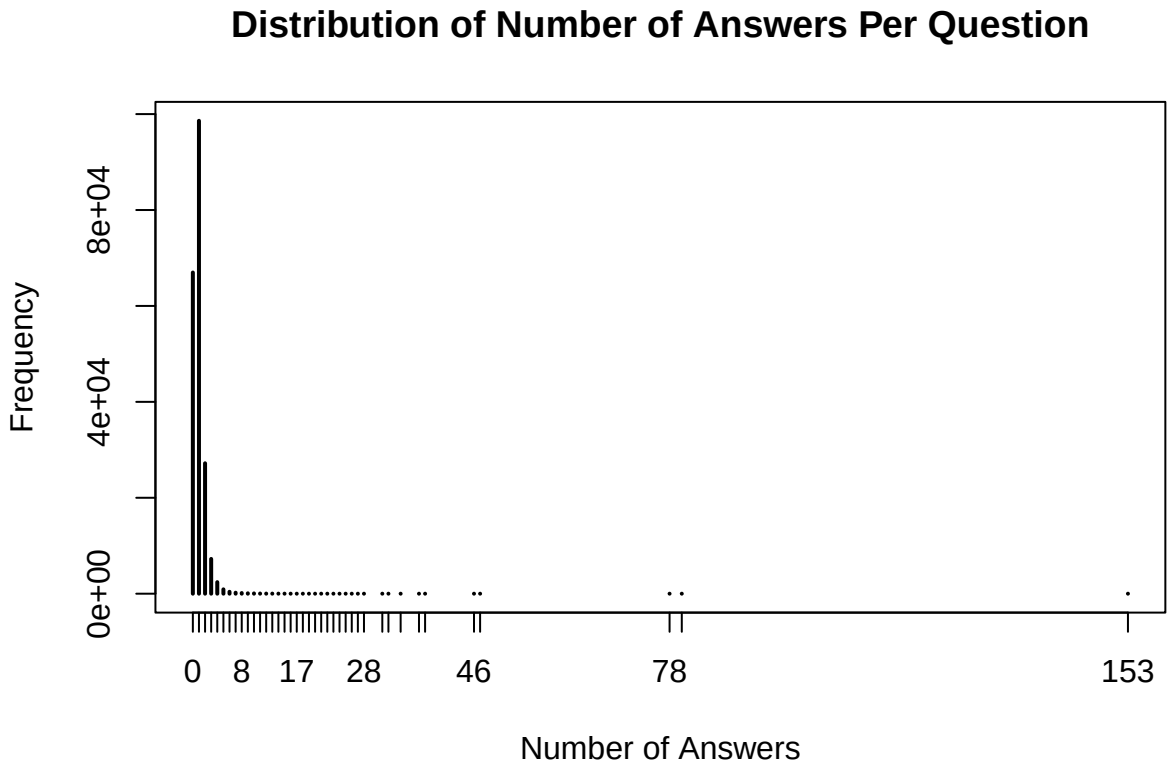
```
distribution_answers =
dbGetQuery(db, "SELECT DISTINCT Title,Id,AnswerCount FROM Posts
           WHERE PostTypeId = 1")

distribution_anscounts = table("Number of Answers" = distribution_answers$AnswerCount)
distribution_anscounts
```

```
## Number of Answers
##     0     1     2     3     4     5     6     7     8     9    10    11    12
## 66970 98602 27191  7246  2408   905   401   210   136    82    62    35    25
##    13    14    15    16    17    18    19    20    21    22    23    24    25
##    19    14    14     8     5     4     5     1     5     2     1     1     2
##    26    27    28    31    32    34    37    38    46    47    78    80   153
##     2     1     2     1     2     1     1     1     1     1     1     1     1
```

From above, we see the frequency table of the answers for each question. For example, there were 66970 questions that had no answers. This is valid as the question I computed before this (Q24) displayed the same value. To make the shape and distribution more clear, I made a plot

```
# Graphically
plot(distribution_anscounts, xlab = "Number of Answers", ylab = "Frequency", main = "Distribution of Nu
```



Very right skewed. Most of the data lies on the left side, which means that most of the questions had a few (1-3) answers per question.

Tables Used: Posts

26.) What is the length of time for a question to receive an answer? To obtaining an accepted answer?

Thought Process: For considering the length of time for a question to receive an answer, I wanted to subtract the date of the very first answer (not accepted) received by the question by the date of the posted question. I used a self join on the Posts table because we need to extract two different date times, one for the Questions and the other for the Answers.

I set Q.Id = A.ParentId because I wanted to find all the answers associated for every question. I also set the constraint of PostTypeId to be 1 for a question. Next, I extracted the relevant variables I wanted, which includes the Q.Id,A.ParentId,A.CreationDate,Q.CreationDate,and the unixepoch for the Answer and Question CreationDates. I will elaborate on using unixepoch later.

Now I have a table that looks like this

```
##    Question_Id ParentId             Answer_Date           Question_Date
## 1            1        1 2010-07-19T19:19:46.160 2010-07-19T19:12:12.510
## 2            1        1 2010-07-19T20:23:57.330 2010-07-19T19:12:12.510
## 3            1        1 2010-07-19T22:40:47.947 2010-07-19T19:12:12.510
## 4            1        1 2010-07-20T05:13:21.963 2010-07-19T19:12:12.510
## 5            1        1 2010-09-15T21:08:26.077 2010-07-19T19:12:12.510
## 6            1        1 2020-11-05T09:44:51.710 2010-07-19T19:12:12.510
## 7            2        2 2010-07-19T19:24:35.803 2010-07-19T19:12:57.157
## 8            2        2 2010-07-19T19:43:20.423 2010-07-19T19:12:57.157
## 9            2        2 2010-07-20T23:07:48.713 2010-07-19T19:12:57.157
## 10           2        2 2011-04-10T14:30:50.133 2010-07-19T19:12:57.157
##       Answers  Questions
## 1  1279567186 1279566732
## 2  1279571037 1279566732
## 3  1279579247 1279566732
## 4  1279602801 1279566732
## 5  1284584906 1279566732
## 6  1604569491 1279566732
## 7  1279567475 1279566777
## 8  1279568600 1279566777
## 9  1279667268 1279566777
## 10 1302445850 1279566777
```

Notice how we have multiple ParentIds corresponding to a QuestionId. The number of counts of a ParentId to a Question is the number of answers for that question. After I completed this step, I wanted to find the MINIMUM answer date, as this displays the quickest time a question receives an answer. To do this, I used the MIN function on A.CreationDate. In addition, because the MIN function will constrain the output (remove ParentIds that are not of minimum date), we have to use GROUP BY A.ParentId to show the other tuples.

```
dbGetQuery(db, "SELECT Q.Id AS Question_Id,A.ParentId,MIN(A.CreationDate) AS Answer_Date,Q.CreationDate
FROM Posts AS A, Posts AS Q
    WHERE Q.Id = A.ParentId
    AND Q.PostTypeId = 1
    GROUP BY A.ParentId
    LIMIT 3
    ")
```

```
##   Question_Id ParentId             Answer_Date           Question_Date
## 1           1        1 2010-07-19T19:19:46.160 2010-07-19T19:12:12.510
## 2           2        2 2010-07-19T19:24:35.803 2010-07-19T19:12:57.157
## 3           3        3 2010-07-19T19:14:43.050 2010-07-19T19:13:28.577
##      Answers  Questions
## 1 1279567186 1279566732
## 2 1279567475 1279566777
## 3 1279566883 1279566808
```

We can double check this result with the other table above. We see that the minimum date for answer and corresponding question date is correct. This is where I explain unixepoch. I used a nested function: unixepoch(strftime()) to first convert the CreationDate format of YYYY-MM-DDTHH:MM:SS.SSS into YYYY-MM-DD HH:MM:SS.SSS using strftime. After I received the strftime result, I used unixepoch to convert the converted Date into a format of seconds. Using this way, I could easily find the difference between the Answer and Question dates as they were both in seconds. Following this, I set the names of the unixepoch(strftime) values to Answers and Questions, which are represented in seconds as type int. I also had to use a subquery, as the newly created variable names in the aforementioned sentence did not exist in the original table. I also converted the seconds into minutes as my final result.

```
# Length of time for a Question to Receive an Answer
dbGetQuery(db, "SELECT Answer_Date,Question_Date,((Answers-Questions)/60) AS MinutesToReceiveAnswer,Ques
           FROM(SELECT COUNT(*),Q.Id AS Question_Id,A.ParentId,MIN(A.CreationDate) AS Answer_Date,Q.Crea
FROM Posts AS A, Posts AS Q
    WHERE Q.Id = A.ParentId
    AND Q.PostTypeId = 1
    GROUP BY A.ParentId)
    ORDER BY Question_Id
    LIMIT 5")
```

```
##               Answer_Date           Question_Date MinutesToReceiveAnswer
## 1 2010-07-19T19:19:46.160 2010-07-19T19:12:12.510                     7
## 2 2010-07-19T19:24:35.803 2010-07-19T19:12:57.157                    11
## 3 2010-07-19T19:14:43.050 2010-07-19T19:13:28.577                     1
## 4 2010-07-19T21:31:53.813 2010-07-19T19:13:31.617                   138
## 5 2010-07-19T19:18:56.800 2010-07-19T19:14:44.080                     4
##   Question_Id
## 1           1
## 2           2
## 3           3
## 4           4
## 5           6
```

Finding the length of time for an accepted answer uses almost the exact process above. The only difference is that we need to do an additional join from the Posts table to join the Answer Id to the AcceptedAnswerId to extract all the Accepted Answers for a post.

```
# Accepted Answer

dbGetQuery(db, "SELECT Answer_Date,Question_Date,((Answers-Questions)/60) AS MinutesToReceiveAnswer,Ques
(SELECT Q.Id AS Question_Id,MIN(A.CreationDate) AS Answer_Date, Q.CreationDate AS Question_Date,(unixepo
FROM Posts AS A, Posts AS Q, Posts AS Accepted
    WHERE Q.Id = A.ParentId AND
    Q.PostTypeId = 1 AND
    A.Id = Accepted.AcceptedAnswerId
    GROUP BY A.Id)
    ORDER BY Question_Id
    LIMIT 5
    ")
```

```
##               Answer_Date           Question_Date MinutesToReceiveAnswer
## 1 2010-07-19T19:19:46.160 2010-07-19T19:12:12.510                     7
## 2 2010-07-19T19:43:20.423 2010-07-19T19:12:57.157                    30
```

```
## 3 2010-07-19T19:14:43.050 2010-07-19T19:13:28.577                     1
## 4 2010-07-19T21:36:12.850 2010-07-19T19:13:31.617                   142
## 5 2010-07-19T19:24:18.580 2010-07-19T19:15:59.303                     8
##   Question_Id
## 1           1
## 2           2
## 3           3
## 4           4
## 5           7
```

Compare this to the first table answered for this question. We see that for the accepted answer table, we have one different Question_Id = 7. This means that question_id = 6 does not have an accepted answer. And this is true as I verified through checking the site

https://stats.stackexchange.com/questions/6

Tables Used: Posts, Self-join on Posts

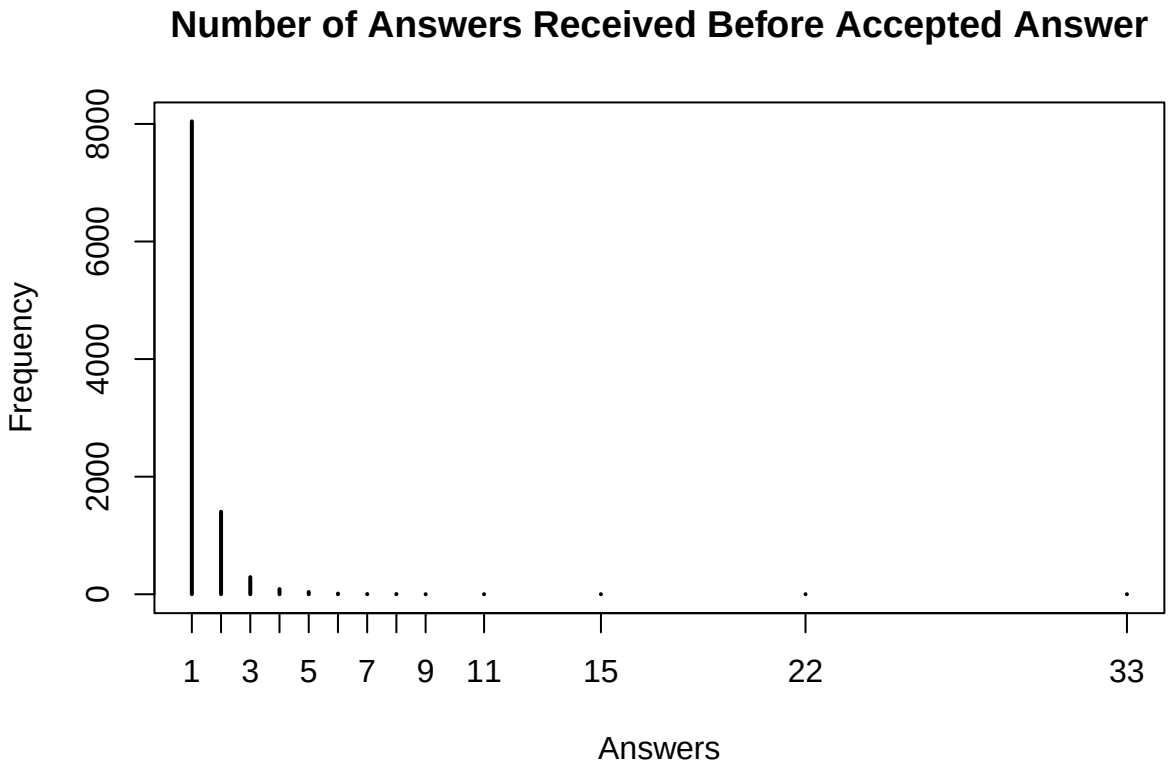
27.) How many answers are typically received before the accepted answer?

```
ansb4accept = dbGetQuery(db, "SELECT A.ParentId,Q.Id AS QuestionId,A.Id AS AnswerId,Q.AcceptedAnswerId A
FROM Posts AS A, Posts AS Q
    WHERE Q.Id = A.ParentId AND
    AcceptedId > AnswerId AND
    CAST(AcceptedId AS INT) > 0
    GROUP BY A.ParentId
    ")
```

```
table("Answer Counts" = ansb4accept$AnswersBeforeAccepted)
```

```
## Answer Counts
##    1    2    3    4    5    6    7    8    9   11   15   22   33
## 8044 1403  291   88   39   11    4    4    1    1    1    1    1
```

```
plot(table(ansb4accept$AnswersBeforeAccepted),xlab = "Answers",ylab = "Frequency",main="Number of Answe
```



My first thought was that I'm going to have to plot this as a histogram distribution to show the counts of answers and the frequencies of the counts. From this, I needed to find all of the answers that were related to a question, and the corresponding accepted answer. After we get this data, we can imagine a data frame with two column: One for AcceptedId, the other for Answer. To find how many answers were received before the accepted, I needed to find the tuples where the numerical value of AcceptedId was greater than AnswerId. Here is an example:

```
dbGetQuery(db, "SELECT A.ParentId,Q.Id AS QuestionId,A.Id AS AnswerId,Q.AcceptedAnswerId AS AcceptedId
FROM Posts AS A, Posts AS Q
    WHERE Q.Id = A.ParentId
    ORDER BY A.ParentId
    LIMIT 20
    ")
```

```
##    ParentId QuestionId AnswerId AcceptedId
## 1         1          1       15         15
## 2         1          1       98         15
## 3         1          1      154         15
## 4         1          1      218         15
## 5         1          1     2696         15
## 6         1          1   495164         15
## 7         2          2       20         59
## 8         2          2       59         59
## 9         2          2      357         59
## 10        2          2     9414         59
## 11        2          2    11969         59
```

```
## 12          2           2    17397          59
## 13          2           2    33936          59
## 14          3           3        5           5
## 15          3           3        9           5
## 16          3           3       14           5
## 17          3           3       16           5
## 18          3           3       24           5
## 19          3           3       28           5
## 20          3           3       42           5
```

In the example above, if we look at QuestionId = 1, we see that there are no acceptedIds greater than answerIds. For QuestionId = 2, however, we see that there is an AcceptedId = 59 that is greater than AnswerId, therefore in this case we would say that question Id = 2 would have one answer received before the accepted. In addition, I noticed that some values in AcceptedId were = 0, which meant that the question did not have an acceptedId. For this, I filtered out the AcceptedIds = 0. After this process, I extracted the insights.

Tables Used: Self-join on Posts