STAT 19000 Project 12

Topics: R & SQL

Motivation: The ability to take a dataset from a familar format (csv, json, etc.) and move it to an sqlite database could come in handy. Databases are made for working with large amounts of data. You get high performance and the ability to create indexes, and you don't have to write special code to get portions of data from your dataset. In addition, things like creating a table and index, inserting data, and understand terms like foreign key and primary key, are definitely worthwhile learning.

Context: SQL is a really great tool to have in your belt, and in the previous project we saw how easy it is to work with databases from within R. There are *lots* of different tools to help efficiently read in data from a csv or from json. Those are excellent formats, but when you are working with anything more than a gigabyte or so (depending on your machine), the efficiency of those tools starts to degrade quickly. One should really consider moving a dataset over to an sqlite database when this happens. In this project, we will work on moving a familiar set of data from csv's to an sqlite database, and learn some more about SQL in the meantime.

Scope: SQL, RSQLite, RMariaDB & R.

You can find useful examples that walk you through relevant material here or on scholar:

`/class/datamine/data/spring2020/stat19000project12examples.pdf`.

It is highly recommended to read through these to help solve problems.

Use the template found here or on scholar:

`/class/datamine/data/spring2020/stat19000project12template.Rmd`

to submit your solutions.

Important note: From this point on, unless specified otherwise, submit your solutions as an RMarkdown (`.Rmd`) file. Place code in the provided code chunks. Submit both the raw `.Rmd` file as well as the knitted PDF.

Don't forget the very useful documentation shortcut `?`. To use, simply type `?` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help (package=PACKAGENAME)`.

Sometimes it can be helpful to see the source code of a defined function. To do so, type the function's name without the `()`.

Question 1: comparing files vs databases (mysql & elections)

We've setup a MySQL database for election data. Create a connection using `RMariaDB` to the election database using the username and password in scholar at:

`/class/datamine/data/spring2020/mysql_credentials.txt`.

Important note: If you choose to use the MySQL database to solve any of the following problems, do so *within* R using the `RSQLite` package like we learned to do in the previous project.

1a. *(1 pt)* In question 2c & 2d in project 7 last semester, you were asked:

> Using the data from the 2018 election campaign donations, save all of the information about the donors that were somehow affiliated with Purdue, into a new file called purduedonations.txt.
>
> Hint: All of the contents of the file are in capital letters, so search for PURDUE rather than Purdue.

How many such donations were made in the 2018 election campaign, from Purdue-related donors?"

Now answer the same question, but include *all* years. In addition, we only want to include doners who's employer (from the `employer` column) is "Purdue University".

Answer the question using any tool you'd like. Did you choose to use the MySQL database? Why or why not?

Item(s) to submit:
- The code used to solve the problem in code chunks.
- A sentence saying whether or not you chose to use the database, and why or why not.

1b. *(1 pt)* Get the highest donation for each year. Answer the question using any tool you'd like. Did you choose to use the MySQL database? Why or why not?

Hint: *If you choose to use SQL, this may be useful.*

Item(s) to submit:
- The code used to solve the problem in code chunks.
- A sentence saying whether or not you chose to use the database, and why or why not.

By now you can see that there are situations where SQL is a great tool to solve a problem. This isn't always the case, and many times the best thing is to load up a csv and do some analysis, or use your bash tools like awk.

1c. *(1 pt)* Rather than get the count of donations where the employer is "Purdue University", get that data and put it into a data.frame. Create a histogram where the x-axis is the year, and the y-axis is the sum of the donations for the year.

Item(s) to submit:
- The code used to solve the problem in code chunks.
- Put the resulting plot in your RMarkdown file.

Question 2:

In previous project(s) you worked with two csv files. You can find these files on scholar:

`/class/datamine/data/spring2020/rotten_tomatoes_movies.csv`

`/class/datamine/data/spring2020/rotten_tomatoes_reviews.csv`

These files can *easily* be read into R and manipulated. With that being said, if we had access to more of this data, it would absolutely become less and less feasible to do. Let's practice creating your own database using these two csv files.

2a. *(2 pts)* Look at the `movies` csv. Which column should be the primary key? List the name of each column, and which of the following 5 sqlite types we should use for each column: `NULL`, `INTEGER`, `REAL`, `TEXT`, `BLOB`. Based on that information, create an sql statement to create a table named `movies`. Make sure the types are correct for each column, and that the appropriate column is a primary key.

Item(s) to submit:

- An SQL code chunk with the `movies` create table code saved into a string.
- For example:

```
my_string <- "CREATE TABLE fake_table (
  variable1 TEXT,
  my_key INTEGER PRIMARY KEY,
  variable2 TEXT
)"
```

2b. *(2 pts)* Look at the `reviews` csv. In (3a), there was a primary key. In this table, we do not have a primary key. What type of key do we have, and which column is it? List the name of each column, and which of the 5 sqlite types we should use for each column. Based on that information, create an sql statement to create a table named `reviews`. Make sure the types are correct for each column, and that the appropriate column is identified as a certain type of key.

Item(s) to submit:

- An SQL code chunk with the `reviews` create table code saved into a string.
- For example:

```
my_string <- "CREATE TABLE fake_table (
  variable1 TEXT,
  my_key INTEGER PRIMARY KEY,
  variable2 TEXT
)"
```

2c. *(1.5 pts)* Use `dbConnect` to create a connection to a not-yet-existing database called `rt.db`. Use the `dbExecute` function from the `RSQLite` package to run the query from (2a) and create the `movies` table. Use `dbWriteTable` from the `RSQLite` package, to write the `movies` data.frame to the movies table in the database.

Item(s) to submit:

- A r code chunk with the r code used to connect to the database, create the table, and add the data.frame to the newly created table.

2d. *(1.5 pts)* Repeat (2c), but for the `reviews` data.

Item(s) to submit:

- A r code chunk with the r code used to connect to the database, create the table, and add the data.frame to the newly created table.

2e. *(optional, 0 pts)* Test out your new database, run some queries, and see if everything look and works the way you want it to.

Project Submission:

Submit your solutions for the project at this URL: https://classroom.github.com/a/rkmiEg9- using the instructions found in the GitHub Classroom instructions folder on Blackboard.

Important note: Make sure you submit your solutions in both .Rmd and .pdf formats. The PDF should be the knitted result of the .Rmd. Make sure and double check that the PDF looks okay and displays your solutions correctly.