STAT 29000 Project 10

Topics: Python, SQL, SQLAlchemy

Motivation: There will be many times when your data won't come in the prominent csv format. Often times you'll need to interact with a database for your work. SQLAlchemy is a very popular toolkit that enables direct interaction with databases using SQL or an ORM (object relational mapper) within the Python environment. ORM's can (very roughly) be a convenient way to interact with a database using syntax and concepts that are closer to the language.

Context: We took a brief hiatus from Python to learn how to build tools using R and `shiny`. Now we will continue to learn about Python and the excellent SQLAlchemy library. This library will enable you to connect to and interact with databases while staying in the Python environment.

Scope: Python, SQL, and the SQLAlchemy library.

First, we must install the most up-to-date packages `requests` and `beautifulsoup4`. In scholar, open up a shell and type the following:

```
python3.6 -m pip install sqlalchemy --user
```

For the following questions, please copy and paste the following code at the top of your Jupyter notebook. We are simply importing useful Python modules that are required for this project. Be sure to replace "PURDUEALIAS" with your Purdue username. Note that if you get an error after running this chunk of code, you may need to click on Kernel->Restart.

```
# the following two lines tell notebook.scholar.rcac.purdue.edu's default python interpreter
# that it should look in ~/.local/lib/python3.6/site-packages for packages as well
# if you do not include these two lines at the very top of your notebook, you won't
# be able to import and use the packages we installed at earlier times/dates
import sys
sys.path.append("/home/PURDUEALIAS/.local/lib/python3.6/site-packages")
```

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

`/class/datamine/data/spring2020/stat29000project10examples.ipynb`.

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

Use the template found here or on scholar:

`/class/datamine/data/spring2020/stat29000project10template.ipynb`

to submit your solutions.

After each problem, we've provided you with a list of keywords. These keywords could be package names, functions, or important terms that will point you in the right direction when trying to solve the problem, and give you accurate terminology that you can further look into online. You are not required to utilize all the given keywords. You will receive full points as long as your code gives the correct result.

Question 1: SQL review

1a. *(.5 pts)* In 1-3 sentences, explain the difference between an inner join and left join.

Item(s) to submit:
- 1-3 sentences explaining the difference between inner join and left join.

1b. *(.5 pts)* Logical opeations in SQL have order of operations, similarly to how multiplication, addition, and subtraction have order of operation in math. For example, the expression `1 + 2 * 4 - 2 * 10` could be re-written using parenthesis to indicate the order of operation as `((1 + (2 * 4)) - (2 * 10))`. Re-write the following SQL statement to include parenthesis indicating the correct order of operation: `X AND Y OR Z OR U AND V AND P OR T`.

Item(s) to submit:
- Modifed SQL statement with parenthesis indicating order of operations.

1c. *(1 pt)* If you run the following two SQL commands, back-to-back, will everything run as expected or not? Why?

```
SELECT s.FirstName, s.LastName, g.Grade FROM students AS s, grades AS g WHERE g.grades > 90;
SELECT s.FirstName, s.LastName FROM students WHERE g.grades > 90;
```

Keywords: *aliasing in SQL*

Item(s) to submit:
- 1-3 sentences explaining why the SQL commands will or will not work when run back-to-back.

Question 2:

For the following questions, use the non ORM way of querying data using regular SQL.

Use the following code to connect to a `read-only` database containing IMDB information (note that we are assuming you are using https://notebook.scholar.rcac.purdue.edu):

```
import sqlalchemy as db
engine = db.create_engine("sqlite:////class/datamine/data/spring2020/imdb.db")
```

Note: *Titles in the `titles` table can be referring to either an individual episode of a tv show, or the name of the tv show itself. This is why the `episodes` table has two columns: `episode_title_id` and `show_title_id`. The former will get an episode's title from the `title` table, and the latter will get the title of the tv show itself. The `ratings` table is the same as the `titles` table in that it contains ratings for individual episodes as well as the show itself.*

2a. *(1 pt)* List the tables in the database, then, loop through and print the column names of each table

Keywords: *MetaData, tables, c or columns*

Item(s) to submit:
- The names of the tables in the `imdb.db` database.
- The names of the columns for every table in the `imdb.db` database, printed.

2b. *(2 pts)* You can read an sql table directly into a pandas dataframe. Run the following code:

```
import pandas as pd

# read in titles table to pandas dataframe
df = pd.read_sql_table("titles", con=engine)
```

```
# how many GB of memory is used?
df.memory_usage(deep=True).sum()/1_000_000_000
```

Yikes! That is a lot of memory – at least if you aren't on scholar or Dr. Ward's computer.

Perform the following problem using:

1. `pandas` and your newly created `df` data frame, and
2. The "pure SQL" method with `SQLAlchemy`.

Get the minimum, maximum, and average `runtime_minutes` by `type`. Which method do you prefer?

Keywords: *SQL: SELECT, MIN, MAX, AVG, GROUP BY pandas: groupby, mean, min, max*

Item(s) to submit:
- The `SQLAlchemy` code using the `execute` function and an SQL query to find the min, max, and average runtime for the titles in the `titles` table when grouped by `type` – a column in the `titles` table.
- The `pandas` code used to calculate the same information.

2c. *(1 pt)* Use the "pure SQL" method with `SQLAlchemy` to, in a single query/statement, get the `primary_title` information from the `titles` table and the episode information from the `episodes` table for the title with `title_id` "tt0108778". The result should be in ascending order first by `season_number` and then by `episode_number`.

Hint: *Note that the `show_title_id` column in the `episodes` table is the id for the show itself. The `episode_title_id` is the id for a single episode of a show.*

Item(s) to submit:
- A printed list that looks like:

```
('Friends', 'tt0583459', 'tt0108778', 1, 1)
('Friends', 'tt0583647', 'tt0108778', 1, 2)
('Friends', 'tt0583653', 'tt0108778', 1, 3)
```

for *every* episode, sorted first by `season_number`, and then by `episode_number`.

2d. *(1 pt)* Modify (2c). Use the `title_id` and write a query that results in a table containing every episode from the tv show with `show_title_id` "tt0108778" (the best tv show) in the order in which they aired.

Each row should contain all of the `episodes` table variables.

Each row should contain the following variables from the `titles` table: `title_id`, `primary_title`, `premiered`, `runtime_minutes`, and `genres`. Order in ascending order first by `season_number` and then by `episode_number`.

Item(s) to submit:
- A printed list that looks like:

```
('tt0583459', 'tt0108778', 1, 1, 'tt0583459', 'The One Where Monica Gets a Roommate', 1994, 22, 'Come
('tt0583647', 'tt0108778', 1, 2, 'tt0583647', 'The One with the Sonogram at the End', 1994, 22, 'Come
('tt0583653', 'tt0108778', 1, 3, 'tt0583653', 'The One with the Thumb', 1994, 22, 'Comedy,Romance')
```

for *every* episode, sorted first by `season_number`, and then by `episode_number`.

Question 3:

3a. *(2 pts)* One scenario where mixing some python and SQL may come in handy is when you want to perform what you did in (2d) for *many* different shows. Write a function with the following signature:

```
def get_series(engine, id: str) -> pd.DataFrame:
    pass
```

When provided with the `engine`, and an id, return a `pandas` dataframe with the same information for a tv show as found for Friends in (2d). Make sure to add column names.

Important note: *It is ok to write an unsafe function for personal use, however, to avoid SQL injection, the best practice in this situation would be to avoid using f-strings and use `sqlalchemy.sql.text()` and qmark style or named style placeholders as described here.

Keywords: *fetchall*

Item(s) to submit:

- The completed `get_series` function. Note you must add columns to your outputted dataframe.
- The output for the following calls:

```
get_series(engine, "tt0108778").head()
get_series(engine, "tt1266020").head()
```

3b. *(1 pt)* Like in (3a), create a function that interacts with the database to extract some cool information. The function can do anything you'd like – be sure to exercise your SQL skills (and your python skills if you would like). Provide 2 examples of using your function. You are NOT limited to anything here. You can use any python library to create anything you want, get creative.

Item(s) to submit:

- The function you created to interact with the database.
- 2 examples (with output) of using your function.

Project Submission:

Submit your solutions for the project at this URL: https://classroom.github.com/a/6amoLzO2 using the instructions found in the GitHub Classroom instructions folder on Blackboard.

Important note: Make sure you submit your solutions in both .ipynb and .pdf formats. We've updated our instructions to include multiple ways to convert your .ipynb file to a .pdf on scholar. You can find a copy of the instructions on scholar as well: `/class/datamine/data/spring2020/jupyter.pdf`. If for some reason the script does not work, just submit the .ipynb.