Project 3: Databases Winter 2021

Objectives

- Be able to write a range of SQL queries to extract data from a relational database
- Gain experience writing interactive command line programs that support a range of commands and options
- Use Plot.ly to create bar charts

Overview

You will write a program that allows a user to retrieve information about gourmet chocolate bars. This data was originally retrieved from Kaggle (chocolate bars) and a JSON data source (countries), but you will be working with a cleaned-up version of the data already available in a Sqlite data base file. You will add the ability for a user to issue several different types of queries which will be presented either directly in the shell or in the form of a bar plot.

Knowledge Required

The contents of each part are covered in lecture as follows:
- Part 1 (SQL):
  - Lecture Notes (Week 9-11)
  - W3schools SQL tutorials
- Part 2 (Implement Menu):
  - Previously covered.
- Part 3 (Plot.ly):
  - Week 11
So, one strategy is to implement Part 1 & Part 2 this week, and then implement part 3 after that.

Getting started

Starter code:
- available at https://github.com/umsi-amadaman/Proj3_2021
- The starter repo contains three files: proj3_choc.py and proj3_choc_test.py and a help.txt file. Run proj3_choc_test.py to check if your functions implemented in Part 1 return expected values.

Database:
- Create a .gitignore file and add "choc.sqlite" to it, along with any other files and folders that you don't wish to push to GitHub (e.g. __pycache__).
- Download choc.sqlite and move it into your cloned starter code folder.

What to submit

Push your solution code to the GitHub repo before deadline, and submit the repo URL on Canvas. **Please don't push the .sqlite file and other files and folders that don't contain your solution code.** Also make sure that you write your name and uniqname in the comment at the top of proj3_choc.py. This will make our awesome IAs' lives easier!

Also make sure that:
- If you create additional file that your program relies on, remember to push those to github.
- Don't submit another commit after submission. If you do, submit a new screenshot on canvas. Also, any commit after deadline will result in late penalty.

Part 1: Implement Query Interface

As the first step, you will implement a function 'process_command' that takes a **command string** and returns a list of tuples representing records that match the query. The command string represents a query that the user wants to run, however you will need to translate from the user command inputs to the corresponding SQL query. In other words, you will need to parse the command (as you have done with interactive input throughout the semester) and construct the appropriate SQL query based on the input.

Spend some time looking at choc.sqlite in your DB Browser. Look at both the structure of the database and the data itself. Run some sample queries using "Execute SQL."

Here are some notes on the structure of the two tables in choc.sqlite:

Table: Bars

This table contains data on the bars that have been sold, so the companies referenced here are the **sellers** and the bean origins refer to the **sources** of beans!

Column Name	Type	Description
Id (primary key)	Integer	Primary key, assigned by DB
Company	Text	Name of the company which makes the bar
SpecificBeanBarName	Text	The name of the bar itself, or sometimes the name of the bean
REF	Text	Reference number
ReviewDate	Text	Date review was conducted
CocoaPercent	Real	% of cocoa in the bar
CompanyLocationId	Integer	Foreign key - points to Countries, location of chocolate company
Rating	Real	Rating given by chocolate experts
BeanType	Text	Category of the cocoa bean
BroadBeanOriginId	Integer	Foreign key - points to Countries, origin country

			of beans used to produce beans

Table: Countries

This table contains country data on countries and will become useful when **JOIN**ed with Bars.

Column Name	Type	Description
Id (primary key)	Integer	Primary key, assigned by DB
Alpha2	Text	2 letter country code
Alpha3	Text	3 letter country code
EnglishName	Text	English name for country
Region	Text	Broad region where country is located.
Subregion	Text	More specific subregion where country is located.
Population	Integer	Country's population
Area	Real	Country's area in km2

Commands

The `process_command()` function takes a string and returns a list of tuples. The string is a command that would be entered by a user (for example `bars sell region=Europe cocoa bottom 5`). The return would be the records that match the query, including the fields that are relevant to the type of command. There are four high-level commands, each of which can take a set of options. Each option has a default value that must be used if the option is omitted. Details for each of the high-level commands and their options are as follows:

High level commands
All commands must begin with one of the high level commands as the first term. The four valid commands are **bars, companies, countries,** and **regions.** The high-

level command indicates what type of entity the query will return. A **bar** query will return a set of records, each representing a particular chocolate bar. A **companies** query will return records representing different companies. **Countries** and **regions** will return records representing individual countries and regions, respectively.

The options specify how the results are to be filtered (e.g., by countries/regions) or sorted (e.g., by ratings or cocoa%).

Each of these commands yield different results:

bars:
- Lists chocolate bars, according to the specified parameters.
- Valid options: none, country, region, sell, source, ratings, cocoa, ~~number_of_bars~~, top, bottom, <integer>
- Returns a list of <integer> 6-tuple records:
  (Specific Bean Bar Name, Company Name, Company Location, Rating, Cocoa Percent, Broad Bean Origin)

companies:
- Lists chocolate bars sellers according to the specified parameters. Only companies that **sell more than 4** bars are listed in results.
- Valid options: none, country, region, ~~sell~~, ~~source~~, ratings, cocoa, number_of_bars, top, bottom, <integer>
- Returns a list of <integer> 3-tuple records:
  (Company Name, Company Location, <agg: avg rating, avg cocoa, or number_of_bars>)

countries:
- Lists countries according to specified parameters. Only countries that sell/source **more than 4** bars are listed in results.
- Valid options: none, ~~country~~, region, sell, source, ratings, cocoa, number_of_bars, top, bottom, <integer>
- Returns a list of <integer> 3-tuple records:
  (Country, Region, <agg: avg rating, avg cocoa, or number_of_bars>)

regions:

- Lists regions according to specified parameters. Only regions that sell/source **more than 4** bars are listed in results.
- Valid options: ~~none~~, ~~country~~, ~~region~~, sell, source, ratings, cocoa, number_of_bars, top, bottom, <integer>
- Returns a list of <integer> 2-tuple records:
  (Region, <agg: avg rating, avg cocoa, or number_of_bars>)

The column "agg" refers to the requested aggregation according to supplied parameters (i.e., average rating, average cocoa percent, or number of bars).

Options for Commands

Each command supports a set of options to filter and sort the results, as detailed below:

- **[none|country=<alpha2>| region=<name>]**, default=**none**
  - Specifies a country or region within which to limit the results (e.g. if country is set to 'AO' then either only bars with *beans from*—if "source" is specified—or *companies located in*—if "sell" is specified—Angola will be considered depending on wether the parameter sell or source is set).
  - If the chosen command is 'countries', then "country" is invalid as an option. If the chosen command is 'regions', then country, region, and none are all invalid options.
  - If the chosen command is 'bars' and 'none' is chosen as a parameter, then the 'sell/source' parameter does not impact the results.
- **[sell|source]**, default=**sell**
  - Specifies whether to filter **countries/regions** in a query based on **sellers** (Company Locations) or **bean sources** (Bean Originator Locations).
  - If the chosen command is companies, then source is not an option since companies can only be sellers in our data base. Countries and regions can be sources (or bean originators).
- **[ratings|cocoa|number_of_bars]**, default=**ratings**
  - Specifies whether to sort by rating, cocoa percentage, or the number of different types of bars.
  - If the chosen command is 'companies', 'countries' or 'regions', then this paramter also specifies what <agg> will become. It is then either the average

rating, average cocoa percentage, or the amount of bars. All of these are aggregated values.
  - The option 'number_of_bars' is not available if the chosen command is 'bars', because it would be just one for each record since each record represents a bar in the database.
- [**top**|**bottom**], default=**top**
  - Specifies whether to list results in descending (top) or ascending (bottom) order.
- <**integer**>, default=**10**
  - Specifies the number results returned.

Based on this summary of the interface, **command strings** for the 'process_command' function can look like the following examples:

Example 1: Return the "bottom" 8 records in terms of ratings (i.e., the lowest rated bars) that originate from Brazil.

SELECT Company, SpecificBeanBarName, sell.EnglishName, Rating, CocoaPercent, source.EnglishNameFROM BarsJOIN Countries as sellON Bars.CompanyLocationId = sell.IdJOIN Countries as sourceON Bars.BroadBeanOriginId = source.IdWHERE source.Alpha2 = 'BR'ORDER BY Rating ASC LIMIT 8

command:

```
process_command("bars country=BR source ratings bottom 8")
```

return value:

```
[('Brazil Rio Doce', 'Artisan du Chocolat', 'United Kingdom of
Great Britain and Northern Ireland', 1.75, 0.72, 'Brazil'),
('Bahia', 'Bahen & Co.', 'Australia', 2.5, 0.7, 'Brazil'), ('M
onte Alegre, D. Badero', "Akesson's (Pralus)", 'Switzerland',
2.75, 0.75, 'Brazil'), ('Monte Alegre, 3 diff. plantations',
'AMMA', 'Brazil', 2.75, 0.85, 'Brazil'), ('Fazenda Camboa', 'A
rete', 'United States of America', 2.75, 0.75, 'Brazil'), ('Br
azil, Batch 20316', 'Beehive', 'United States of America', 2.7
5, 0.8, 'Brazil'), ('Monte Alegre, D. Badaro, Raw, Organic',
'Fearless (AMMA)', 'United States of America', 2.75, 0.75, 'Br
azil'), ('Brazil', 'Q Chocolate', 'Brazil', 2.75, 0.55, 'Brazi
l')]
```

Example 2: Return the top 12 companies located in Europe in terms of bars sold.

command:

```
process_command("companies region=Europe number_of_bars 12")
```

return value:
```
[('Bonnat', 'France', 27), ('Pralus', 'France', 25), ('A. Mori
n', 'France', 23), ('Domori', 'Italy', 22), ('Valrhona', 'Fran
ce', 21), ('Hotel Chocolat (Coppeneur)', 'United Kingdom of Gr
eat Britain and Northern Ireland', 19), ('Coppeneur', 'German
y', 18), ('Zotter', 'Austria', 17), ('Artisan du Chocolat', 'U
nited Kingdom of Great Britain and Northern Ireland', 16), ('S
zanto Tibor', 'Hungary', 15), ('Pierre Marcolini', 'Belgium',
14), ('Amedei', 'Italy', 13)]
```

Example 3: Return the top 10 countries in Asia in terms of the average cocoa level across all bars sold by selling companies located in Asia

command:
```
process_command("countries region=Asia sell cocoa top")
```

return value (note that only 3 are returned, because there are only 3 countries from Asia that sell more than bars):
```
[('Viet Nam', 'Asia', 0.7454545454545454), ('Japan', 'Asia',
0.7076470588235293), ('Israel', 'Asia', 0.7055555555555556),
('Korea (Republic of)', 'Asia', 0.7)]
```

Example 4: Return the top 3 bean-producing (source) regions in terms of ratings (the default)

command:
```
process_command("regions source top 3")
```

return value:
```
[('Oceania', 3.268939393939394), ('Asia', 3.227272727272727),
('Africa', 3.202067669172932)]
```

The starter code provides a skeleton for 'process_command' in the starter code and recommend that you implement/add commands iteratively, i.e. implementing 'bars' first with all the options (test it), then 'companies' and so on... Try to avoid code duplication and don't put everything into one function. You are strongly encouraged to add more functions (at least one for each command). Think also about how to ensure readability e.g. by naming variables well or using concepts you have learned this term to better structure your code. Also take a look at the example outputs for commands in Part 2. These could help you for grasping what kinds of results to expect for the different commands and parameters.

Part 2: Interactive Capabilities

Now that you have implemented the ability to process command strings, the next step is to allow a user to interactively input commands and to nicely format the results for presentation.

The user should be allowed to input commands and receive results as long as they want, and enter "exit" when they are done (which will quit the program). Most of the heavy lifting of this program will be handled by Part 1. All that is required for Part 2 is
- prompting the user for input
- formatting the output "nicely"
  - You don't have to print out column names. Your output should look like the one we provide in our examples below.
  - You should use fixed-width columns to ensure column alignment. If the printed value in a field is greater than 12 characters, you should show a trimmed value in the fixed-width field (e.g. "United State...").
  - We provide you with this documentation for how to do string formatting in Python, but you may also choose your own approach (e.g. other libraries)
- adding basic error handling (i.e., not crashing the program on invalid inputs)

Here is an example run:

```
$ python3 proj3_choc_solution.py

Enter a command: bars ratings

Chuao            Amedei         Italy            5.0  70%  Vene
zuela (B...
```

Toscano Blac...	Amedei	Italy	5.0	70%	No Country
Pablino	A. Morin	France	4.0	70%	Peru
Chuao	A. Morin	France	4.0	70%	Venezuela (B...
Chanchamayo ...	A. Morin	France	4.0	63%	Peru
Morobe	Amano	United State...	4.0	70%	Papua New Gu...
Guayas	Amano	United State...	4.0	70%	Ecuador
Porcelana	Amedei	Italy	4.0	70%	Venezuela (B...
Nine	Amedei	Italy	4.0	75%	No Country
Madagascar	Amedei	Italy	4.0	70%	Madagascar

Enter a command: bars country=US sell cocoa bottom 5

Peru, Madaga...	Ethel's Arti...	United State...	2.5	55%	No Country
Trinidad	Ethel's Arti...	United State...	2.5	55%	Trinidad and...
O'ahu, N. Sh...	Guittard	United State...	3.0	55%	United State...
O'ahu, N. Sh...	Malie Kai (G...	United State...	3.5	55%	United State...
O'ahu, N. Sh...	Malie Kai (G...	United State...	2.8	55%	United State...

Enter a command: companies region=Europe number_of_bars

Bonnat	France	27
Pralus	France	25

```
A. Morin          France          23

Domori            Italy           22

Valrhona          France          21

Hotel Chocol...   United Kingd... 19

Coppeneur         Germany         18

Zotter            Austria         17

Artisan du C...   United Kingd... 16

Szanto Tibor      Hungary         15


Enter a command: companies ratings top 8

Amedei            Italy           3.8

Patric            United State... 3.8

Idilio (Felc...   Switzerland     3.8

Benoit Nihan...   Belgium         3.7

Cacao Sampak...   Spain           3.7

Bar Au Choco...   United State... 3.6

Soma              Canada          3.6

Brasstown ak...   United State... 3.6


Enter a command: countries number_of_bars

United State...   Americas        764

France            Europe          156

Canada            Americas        125

United Kingd...   Europe          107

Italy             Europe          63

Ecuador           Americas        55

Australia         Oceania         49

Belgium           Europe          40

Switzerland       Europe          38

Germany           Europe          35
```

```
Enter a command: countries region=Asia ratings
Viet Nam         Asia              3.4
Israel           Asia              3.2
Korea (Repub... Asia              3.2
Japan            Asia              3.1


Enter a command: regions number_of_bars
Americas         1085
Europe           568
Oceania          70
Asia             46
Africa           26


Enter a command: regions ratings
Oceania          3.3
Asia             3.2
Europe           3.2
Americas         3.2
Africa           3.0


Enter a command: bad command
Command not recognized: bad command


Enter a command:


Enter a command: bars nothing
Command not recognized: bars nothing


Enter a command: exit
```

```
bye
```

Part 3: Bar plot

Now extend the command line interface to include an additional parameter called 'barplot'. It can be supplied after all other commands and parameters have been typed up. Then if provided, instead of printing the results to the console, you should use plot.ly to display a barplot which nicely visualizes the results.

Here is what should be displayed, depending on the high-level command:

bars
- x-axis: Specific Bean Bar Name
- y-axis: ratings or cocoa percentage (whichever was specified by the command)
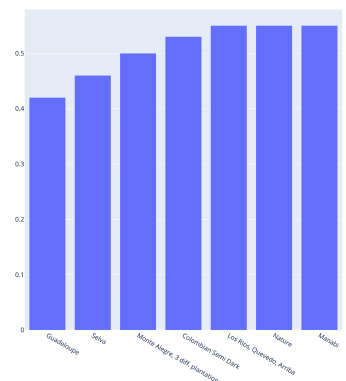
companies
- x-axis: Company Name
- y-axis: ratings/cocoa percentage/number_of_bars (whichever was specified)
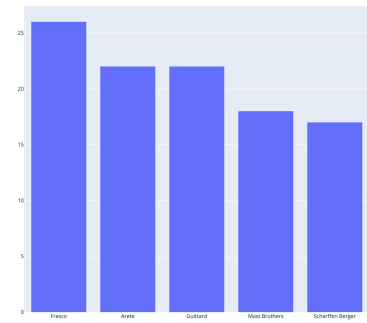
countries
- x-axis: Country Name
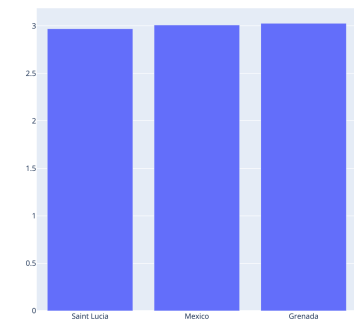- y-axis: ratings/cocoa percentage/number_of_bars (whichever was specified)

regions
- x-axis: Region Name
- y-axis: ratings/cocoa percentage/number_of_bars (whichever was specified)

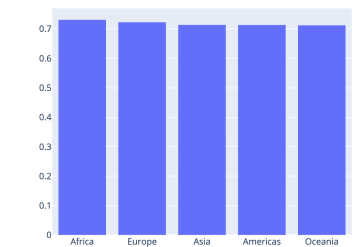| Example	Output
`bars cocoa bottom 7 barplot`	

`companies country=US number_of_bars 5 barplot`	
`countries source region=Americas bottom 3 barplot`	
`regions sell cocoa barplot`	

You will also need to extend 'help.txt' with a description of this new parameter (`barplot`) similar to the others already included.

Rubrics

Req	Part	Description	Category	Point Value
1	1	The 'process_command' functions accepts the four commands.	Code	5
2	1	The 'process_command' functions	Code	10

		accepts the parameters.		
3	1	The 'process_command' functions correctly parses [bars|companies|countries|regions] from the command string and if no command has been passed, then instead the default value is stored in an appropriate variable or data structure.	Code	5
4	2	The 'process_command' functions correctly parses [none|country= <alpha2>| region=<name>] from the command string and if none of the options have been passed, then instead the default value is stored in an appropriate variable or data structure.	Code	5
5	2	The 'process_command' functions correctly parses [ratings|cocoa|number_of_bars] from the command string and if none of the options have been passed, then instead the default value is stored in an appropriate variable or data structure.	Code	5
6	2	The 'process_command' functions correctly parses all parameters from the command string and if a parameter has not been passed, then instead the default value is stored in an appropriate variable or data structure.	Code	5
7	2	The 'bars' command works appropriately with the default parameters.	Code	10
8	3	The 'companies' command works appropriately with the default parameters.	Code	10

9	3	The 'countries' command works appropriately with the default parameters.	Code	10
10	3	The 'regions' command works appropriately with the default parameters.	Code	10
11	4	The tests for the 'bars' command all pass.	Code	5
12	4	The tests for the 'companies' command all pass.	Code	5
13	5	The tests for the 'countries' command all pass.	Code	5
14	5	The tests for the 'regions' command all pass.	Code	5
15	5	The none, country and region parameters work appropriately with all commands.	Behavior	5
16	5	The sell and source parameters work appropriately with all commands.	Behavior	5
17	5	The ratings, cocoa and number_of_bars parameters work appropriately with all commands.	Behavior	5
18	5	The top and bottom parameters work appropriately with all commands.	Behavior	10
19	5	The limit the amount of matches parameter works appropriately with all commands.	Behavior	5
20	5	When a user inputs an invalid command or parameter, you print an error message and ask the user to input again.	Behavior	5
21	5	The query menu works appropriately.	Behavior	5

22	5	You are printing records with fixed-width formatting.	Behavior	10
23	5	You truncate values that are too long appropriately and add the '…' at the end.	Behavior	5
24	5	The barplot parameter is parsed appropriately.	Behavior	5
25	5	The barplot parameter works for at least one of the four commands.	Behavior	10
26	5	The barplot parameter works appropriately with all commands.	Behavior	15
27	ALL	Submitted GitHub repo has only `proj3_choc.py` and `proj3_choc_test.py`. Or, the repo has `.gitignore` that mentions other unnecessary files (such as `__pycache__` or the database file).	Behavior	10
28	ALL	Well-constructed code that follows our guidelines.	Code	10
		Total		200