CIS 5357 spring 2020

Programming Assignment # 7

(35 points)

**Due Date: By 11:59 pm, Sunday, November 8, 2020**

**Caution:**

**The instructor has found many submissions with duplicate or similar code. The instructor expects individual effort on each and every submission. Assignments and exams assigned in this course are neither group projects nor any kind of group activity or collaboration are sanctioned. Such activities will be treated per the Academic Dishonesty policy as stated in the course syllabus. Each submission will be closely examined for plagiarism**. **A slightest hint of duplicate or similar code will be examined for academic integrity violation. Such submissions will automatically receive a grade of zero and reported to the Graduate College for further disciplinary action. To avoid such scrutiny, please do your own work. Please consult your instructor if you any questions or need clarification about assignments and examinations.**

1. **Requirements for Assignment 7:**
2. Name your Jupyter Notebook ‘YourName-Assignment7.ipynb’
3. Include your name and submission date as level 2 headings in the first cell of the notebook.
4. Insert a markdown cell with level 3 heading “Grading Comments – Totality of all points noted below resulted in a reduction of xx Points”
5. Insert a markdown cell with level 3 heading for the Program Name and its objective
6. The program design must be broken down into functions, with each function performing a specified task. The main driver program will also be coded as main() function. (see Hierarchy and IPO charts later)
7. Each function must include an appropriately written docstring.
8. Include functions defined for each program in their individual cells (in the same order as the IPOs described later).
9. Main() function should be the last one defined and in the last cell.
10. To execute the Main() function, include a call statement in its own cell below the Main() function cell.
11. The program will NOT use ANY global variables. All variables must be local to either the function or main program. Functions and main program can exchange data only through the use of arguments in function calls and function parameters and value returning functions. The same applies to libraries to be imported. They should be imported only in the functions where they are required or needed.
12. Please store the movies.sqlite database file in the following path: **“/Users/cis\_developer/CIS5357Fall2020/movies.sqlite”.** If you are not sure how to create this path on your computer (windows or macs), now is the time to learn it. The movie database file **MUST** be in this path. Do not replace your account name in place of cis-developer.
13. Upload your source code file (.ipynb) to the Assignments section of Canvas BEFORE 11:59 pm on Sunday, November 8, 2020. No other files need to be uploaded.
14. **NO LATE ASSIGNMENTS WILL BE ACCEPTED. ASSIGNMENTS SENT VIA EMAIL AS ATTACHMENTS WILL ALSO NOT BE ACCEPTED. YOU ARE ALLOWED A MAXIMUM OF THREE ATTEMPTS TO SUBMIT YOUR FILE BEFORE THE DUE DEADLINE. YOU MISS THE DEADLINE, YOU LOSE IT. So, please start early to have a chance at getting any problem resolved before the submission deadline.**

**Introduction:**

Programming Assignment 8 will require you to design a python program that will use functions to modularize the program as well as access data from an SQLite database. The data retrieved will be used to compute some basic statistics and generate some data visualization. Hierarchy and IPO charts are provided to help you design your program.

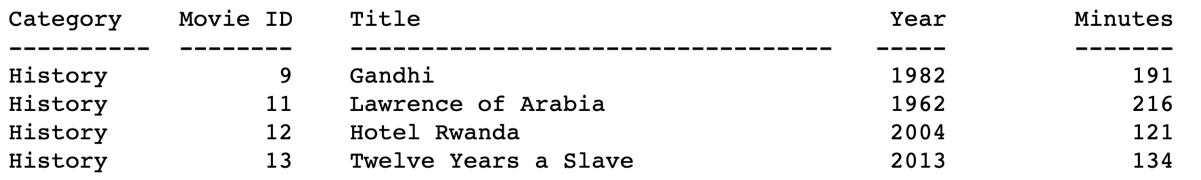
**Specifications:**

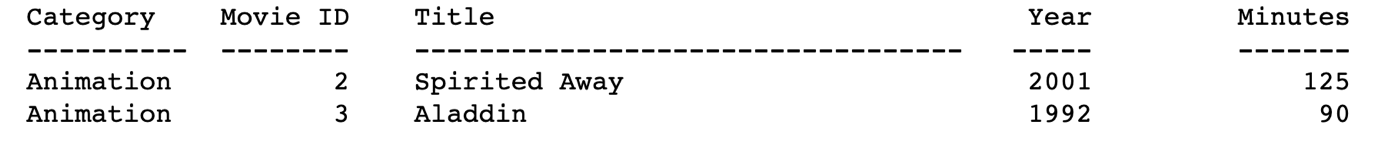
The python program you design will access SQLite database named movies.sqlite. The database may be downloaded from the ClassDemos folder on the course’s TRACS website. Please make sure that you access the database from the root directory of the USB drive e.g. D:/movies.sqlite.

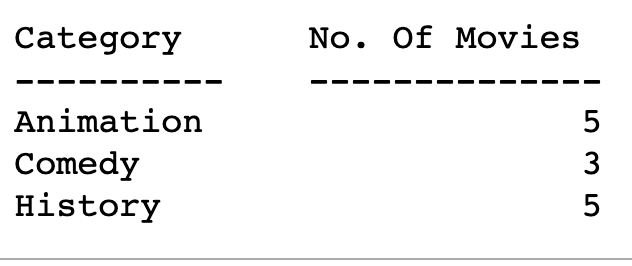
Your program will execute three SQL DML SELECT queries. Each query will be performed in its own function.

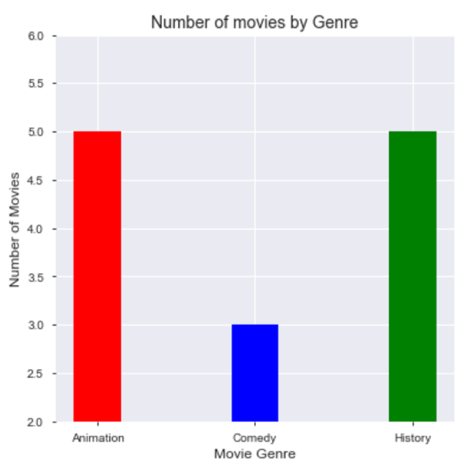
1. Display a list of all movies classified as a specified genre with a running time of **at leas**t the minimum time specified. Please refer to the IPO chart for this function as it requires that this query be written so that the we can get this information for any genre and for any running time. The values for genre and running time will be provided by the calling statement at runtime. This means that the query should be written with parameters whose values will be passed by the calling statement. For the purpose of testing, we will use “History’ as the genre and 2 hours as the minimum running time. For the second test, use ‘Animation’ as the movie genre and 1.5 hours as the minimum running time.

For each movie, display category name, movie ID, movie title, year released, and running time. The data must be displayed in this order and in a visually appealing tabulation. Note: nicely aligned columns that are appropriately spaced; numeric values and their column headers are right-aligned while category name and title are left-aligned.

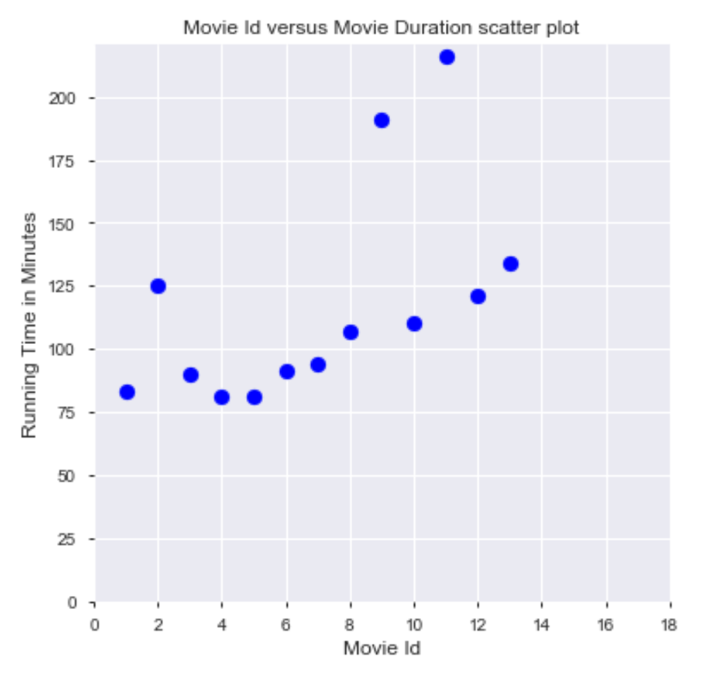




1. Retrieve a count of number of movies by category (genre). In other words, the data retrieved should show each category name and number of movies in that category. Again, display this in a tabulated format as illustrated below. In addition, visualize this data in a bar chart, where category names are displayed on the x-axis and count of movies in the category on the vertical axis. Use the ‘seaborn’ style; set font size for x-y labels to 12 and that for title to 14. Bar colors should be as depicted in the bar-chart below.



1. Retrieve running time of each movie and have the data displayed in a scatter plot. Y-axis shows the running time in minutes while the horizontal axis shows movie id in ascending sequence. Note: formatting – use ‘seaborn’ style, origin is set to [0,0], and scatter plot is blue in color. The title and labels use the same font size as in the previous graph.



1. Hierarchy Chart:
2. Input-Process-Output (IPO) charts:

|  |  |  |  |
| --- | --- | --- | --- |
| The get\_connection() Function | | | |
| Input | | Process | Output |
| None | * If needed, Import any library or libraries needed for this function’s task(s). * Initialize any local constants or variables if needed. * Open connection to the movies.sqlite database. * Return open connection object | | Open connection object |

|  |  |  |
| --- | --- | --- |
| The close\_connection() Function | | |
| Input | Process | Output |
| Open connection object | * If needed, Import any library or libraries needed for this function’s task(s). * Initialize any local constants or variables if needed. * Check if connection open * If the connection is open, close it. | None |

|  |  |  |
| --- | --- | --- |
| The display\_movies\_by\_category() Function | | |
| Input | Process | Output |
| Open connection object, movie\_category, minimum\_running\_time | * If needed, Import any library or libraries needed for this function’s task(s). * Initialize any local constants or variables if needed. * Define a SQL SELECT query to retrieve data for category name, movie ID, movie title, year released, and running time for all movies classified as the specified genre and having a specified minimum running time. The query should be written to accept any movie genre and any running length specified. * Execute the query against the open database connection received. * Once data are retrieved, display the data in tabular format (nicely, properly aligned and labeled columns) in the console. | None |

|  |  |  |
| --- | --- | --- |
| display\_movie\_length() function | | |
| Input | Process | Output |
| Open database connection object | * If needed, Import any library or libraries needed for this function’s task(s). * Initialize any local constants or variables if needed. * Define a SQL SELECT query to retrieve movie ID and running length for each movie in the database. Data must be retrieved in increasing order by MovieID. * Execute the query against the open database connection received. * Once the data are retrieved, store the data in appropriate python data structure(s) to get them ready for data visualization. * Call the display\_scatter\_plot() function to display the running times in a scatter plot, passing in the Python data structures containing values for x-axis and y-axis. | None |

|  |  |  |
| --- | --- | --- |
| display\_scatter\_plot() Function | | |
| Input | Process | Output |
| Python data structures containing values for X- and Y-axes | * If needed, Import any library or libraries required for this function’s task(s). * Initialize any local constants or variables if needed. * Set up the design for plotting a scatter plot * Y-axis will represent movie duration in minutes while x-axis will represent movie id * The scatter plot must include a meaningful heading, x- and y-axes labels in readable format (see output to match)if needed | None |

|  |  |  |
| --- | --- | --- |
| display\_movie\_count\_by\_category() Function | | |
| Input | Process | Output |
| Open database connection object | * If needed, Import any library or libraries required for this function’s task(s). * Initialize any local constants or variables if needed. * Define a SELECT query to retrieve a count of movies by category. The query should retrieve category name and a count of movies that are classified in that category. * Execute the query to retrieve the results. * Store the results in python data structures for visualization purposes. * Call the display\_bar\_plot() function to visualize the data in a bar chart form. Pass in the Python structure(s) that contain values to be plotted on x- and y-axis. | None |

|  |  |  |
| --- | --- | --- |
| The display\_bar Function (Choice # 5) | | |
| Input | Process | Output |
| Python data structure(s) containing values for X- and Y-axes. | * If needed, Import any library or libraries required for this function’s task(s). * Initialize any local constants or variables if needed. * Set up the design for plotting a bar chart. * Y-axis will represent count of movies while x-axis will represent category names. * The bar chart must include a meaningful heading, x- and y-axes labels in readable format (see output to match) | None |

|  |  |  |
| --- | --- | --- |
| The main() Function | | |
| Input | Process | Output |
| None | * If needed, Import any library or libraries required to perform this function’s task(s). * Initialize all local constants and variables to be used in the main function if needed. * Call get\_connection() function to receive an open connection to the database. * Call display\_movies\_by\_category, passing in the open connection, ‘History’ as the movie category/genre name, and 2 hours as minimum running time in minutes. * Call display\_movies\_by\_category() a second time, passing in the open connection, ‘Animation’ as the movie category/genre name, and 1 hour as minimum running time in minutes. * Call display\_movie\_count\_by\_category, passing in the open connection. * Call display\_movie\_length, passing in the open connection. * Call close\_connection() function to close the connection to the database. | None |

HINT:

To group counts by a data field so that it shows a count of items for each value of a data field, we use the following SELECT statement

SELECT *fld-1, count(fld-2)………*

FROM *tables to be used to retrieve data*

Group By *fld-1*

Order By *count(fld-2) asc*

Any field that is listed in SELECT clause but is not a part of an aggregate function (such as fld-1 above) must be included in the GROUP BY clause. ORDER BY is optional but allows to arrange the counts in a specified order. Use the HAVING clause instead of WHERE If the data are to be grouped and the groups formed need to meet a selection criterion,.

SELECT *fld-1, count(fld-2)…………*

FROM *tables to be used to retrieve data*

Group By *fld-1*

HAVING *selection-conditions* (these conditions would be similar to the conditions in where clause)

Order By *count(fld-2) asc*