Summer 2017
MIS 6V99 – Special Topics – Programming for Data Science
Programming Assignment #2
Healthcare Analytics – Recommender system for hospitals based on Medicare ratings and patient surveys

Kevin R. Crook

Scenario

The US government has two main single-payer, national social insurance programs. Medicare is for seniors who are age 65 and older who paid into the Medicare system when working (or their spouses paid into the system), and also for persons eligible for disability. Medicaid is for those who are indigent, and also for seniors who never worked and did not pay into the Medicare system (or did not have a spouse who paid into the system).

In the US, most hospitals are private, and private hospitals can be for profit or not for profit. Some hospitals are public, owned by a local county government, and often referred to as a "county hospital." Few counties have more than one hospital. Most large cities have several private hospitals, a mix of for profit and not for profit.

For most hospitals, Medicare / Medicaid is a major source of revenue, and for county hospitals it is usually the main source of revenue. Medicare/ Medicaid has standards that hospitals must meet. Some payments are withheld if standards are not met. Some payments are bonuses if standards are exceeded.

The main data sets are available for download at the official government website: data.medicare.gov. The data set of interest to us is the Hospital Compare data. This data set is updated several times per year. Private hospital owners and local county governments are always very much interested in this data set, as a major source of their revenue depends on meeting the criteria. Since hundreds of billions of dollars of government payouts are based on this dataset, it is widely studied, and there are numerous consultants and consulting businesses built around this dataset. Most data science shops associated with hospitals routinely analyze this data set.

Big Data is often described in terms of the 3 V's: Volume, Variety, and Velocity. In this assignment, we will focus on Variety. (One remaining assignment will focus on Volume, and the other will focus on Velocity). We will both read and write data in a variety of ways. All of these are common, everyday work for a Data Scientist.

You will need to write Python code to download the latest data set for Hospital Compare from data.medicare.gov and un-compress it into numerous files in csv format. You will need to write Python code to create an SQL based database, create a table to hold each of the files in the data set, and parse and load each of the data files into a table in the database. This is the first step in any analytical project of this magnitude. It will allow you to explore and cleanse the data using the convenience of SQL. You will also find this to be an extremely useful for re-use in future project whether at school or in the workplace.

We have a proprietary in-house system that creates our own ranking of hospitals and a list of focus states for analytics. The file produced will be an MS Excel workbook with 2 sheets. One sheet will have a ranking of hospitals and the other sheet will have a list of focus states. Your Python code will need to download this spreadsheet and read the data for use in further analytics.

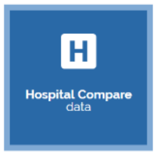
You will perform analytics using the data you loaded into SQL and produce 2 MS Excel Workbooks.

The first workbook will have hospital ranking information. It will have 1 sheet with the top 100 hospitals nationwide. For each of the states in the focus group, it will have the top 100 hospitals for that state.

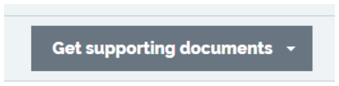
The second workbook will have a statistical analysis of the measures used to determine hospital ranking. It will have 1 sheet with each of the measures, along with the minimum, maximum, mean, and standard deviation for that measure for all hospitals. For each of the states in the focus group, it will have the same statistics for each measure, but only for hospitals in that state.

Downloading the Medicare Hospital Compare Data and Loading it into SQL

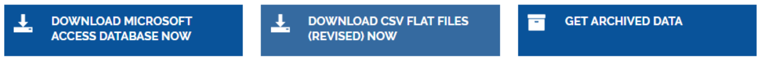
Visit the website, data.medicare.gov, click on the "Hospital Compare data" graphic.



In the upper right corner, you may want to look at the "Get supporting documents" dropdown, especially the "Downloadable Database Dictionary" which explains more about this data set.



You should also see 3 blue graphics side by side, the middle one is the download link for a zip file of csv files that your code will need to download. It is labeled "DOWNLOAD CSV FLAT FILES (REVISED) NOW". If you are using Chrome, you can right click on this and choose "Copy link address".


DOWNLOAD MICROSOFT ACCESS DATABASE NOW
DOWNLOAD CSV FLAT FILES (REVISED) NOW
GET ARCHIVED DATA

There is a chance that this link could get updated before the assignment is due, so please use the following link (it is all one line – word cannot fit it into 1 line):
https://data.medicare.gov/views/bg9k-emty/files/0a9879e0-3312-4719-a1db-39fd114890f1
?content_type=application%2Fzip%3B%20charset%3Dbinary&filename=Hospital_Revised_Flatfiles.zip

Your Python program will need to create a staging subdirectory called "staging". Use only relative path names and use the Python machine independent joining of directory names.

Your Python program will need to unzip the file into various csv files in the staging directory. The first line of each file will be the list of fields. Using the staging directory will make it a lot easier to look at the files for debugging purposes.

Your Python program will need to create an SQLite database in the local directory (not in the staging directory – don't use any path names) named "**medicare_hospital_compare.db**".

Your Python program will need to create a table for each csv file in the staging directory. You should ignore files other than csv files. Note: in this data set, there is one file that is corrupt, which you may ignore: "**FY2015_Percent_Change_in_Medicare_Payments.csv**"

Table names should be the same as the file name without the file extension (.csv), with the transformation detailed below applied.

Column names should be the same as the field name in the first line of the csv file, with the transformation detailed below applied.

Table names and column names should have the following transformations to make them acceptable for SQLite (in this order):

1. Convert all letters to lower case
2. Replace each blank " " with an underscore "_"
3. Replace each dash or hyphen "-" with an underscore "_"
4. Replace each percent sign "%" with the string "pct"
5. Replace each forward slash "/" with an underscore "_"
6. Multiple underscores in a row are ok – actually needed in some cases to prevent duplicate names
7. If a table name starts with anything other than a letter "a" through "z" then prepend "t_" to the front of the table name
8. If a column name starts with anything other than a letter "a" through "z" then prepend "c_" to the front of the column name

Since we are loading these as staging tables, use "text" for the data type for every column, even if you think the column might be numeric, a date, etc. This will allow anything to be loaded into every column, otherwise bad data would cause load errors.

All data from each file should be loaded into the corresponding table, except the first line containing the field list.

Suggestion: Query out a few records and manually compare them to the first few lines of the file. Query out the row count for each table and compare it to the number of rows in the file. Writing a separate Python program to test this would save a lot of time.

Hint: this data set was created on windows with windows encoding (cp1252). It also looks like these files were created with concatenation of multiple files on windows, which can create nulls in the file. You will need to read the files with an encoding of cp1252, remove any nulls, and write them out in utf-8 encoding before any of the Python csv modules will be able to read it.

Download MS Excel Workbook of In House Proprietary Hospital Rankings and Focus List of States

Our in house system has produced a ranking of all hospitals in the US and a focus list of states that we need to use in our analysis.

Your Python program should download the MS Excel Workbook from the following link:
http://kevincrook.com/utd/hospital_ranking_focus_states.xlsx

Your Python program should read this workbook. The first sheet is "**Hospitals National Ranking**" and contains a ranking list of all hospitals in the US, with columns "**Provider ID**" and "**Ranking**". The second sheet is "**Focus States**" and contains a list of the focus states, with columns "**State Name**" and "**State Abbreviation**".

Create the Hospital Ranking MS Excel Workbook

Your Python program should create a hospital ranking MS Excel Workbook named "**hospital_ranking.xlsx**" in the local directory without using any path names.

It should have a first sheet named "**Nationwide**". It should have the following column headers "**Provider ID**", "**Hospital Name**", "**City**", "**State**", and "**County**". Follow this header row with the top 100 hospitals as ranked by the in house proprietary system, ordered by rank. For the state column, the data should use the 2 letter state abbreviation.

For each of the states in the focus list, it should have a separate sheet for each state. The sheet name should be the state name spelled out, not an abbreviation. The sheets should be in alphabetic order by the state name spelled out. Each sheet should have the same columns and data as the first sheet, except the data should be the top 100 hospitals located in that state, ordered by rank.

Hint: you will probably want to query data out of the **hospital_general_information** table to match up with the provider id's.

Create the Measures Statistical Analysis MS Excel Workbook

Your Python program should create a hospital ranking MS Excel Workbook named "**measures_statistics.xlsx**" in the local directory without using any path names.

From the table **timely_and_effective_care___hospital** query out the **state**, **measure_id**, **measure_name**, and **score**. Some of the scores have non-numeric data, some have a mix of numeric and non-numeric data. If all scores for a measure are non-numeric, ignore that measure. If a score has a mix of numeric and non-numeric data, ignore the non-numeric data and just find statistics on the numeric data.

It should have a first sheet named "**Nationwide**". It should have the following column headers "**Measure ID**", "**Measure Name**", "**Minimum**", "**Maximum**", "**Average**", and "**Standard Deviation**". Follow this with 1 row per measure. Sort by measure_id. Calculate the minimum, maximum, average, and standard deviation for that measure for all hospitals nationwide.

For each of the states in the focus list, it should have a separate sheet for each state. The sheet name should be the state name spelled out, not an abbreviation. The sheets should be in alphabetic order by the state name spelled out. Each sheet should have the same columns and data as the first sheet, except the data should be statistics for that measure only for hospitals located in that state.

Individual Assignment

This assignment in an individual assignment. You may consult with other student about general approaches to solving the problem and for asking for help to resolve stack traces, but all coding must be your own work. An electronic comparison for similarities in submissions will be made. Any similarities greater than 70% will be investigated by the instructor, with possible referrals for academic dishonesty.

Auto Grader

All directory, file, and other names must be spelled exactly as given, case sensitive. All outputs must be in the correct format, also case sensitive. The reason for this is that the auto grader will look for directories, files, etc. based on an exact spelling, case sensitive, and if it is not found it will not be run nor graded.

If the final source does not run to completion without a stack trace, no credit for the assignment will be given. The source code will be run in an Anaconda Python 3 sandbox using the release available on the first day of class. Unless explicitly specified otherwise, all files should be created in the local directory without any device names nor path names. If a subdirectory is specified, it must be created using relative pathnames and using the machine independent functions of Python to join directory names.

Only source code properly checked into GitHub will be run for grading. Only output files from the auto grader run of the program will be considered for grading. Student submitted output files will not be considered.

GitHub Repository ("repo")

You must create a GitHub repository called **mis_6v99_2017_summer** as a private repository and grant access to the instructor account **kevin-crook-ucb**. You must create a directory in the repository called **assignment_02**, with 1 and only 1 read me file (either **README.txt** or **README.md**, but not both) with at least 1 line of meaningful comment, and place the **analyze_medicare_data.py** file in that directory.

Only source code properly checked into GitHub will be considered for grading. The time of check into GitHub of source code will be the determining factor where time limits are considered.

Python Program

You will write a single file Python program, **analyze_medicare_data.py**, to accomplish them. The program must download and read all files correctly. The program must run without stack trace. The program must create the specified output files. Only output files created from the instructor's run of your source code can be considered for grading.

Documentation Strings and Ratio of Source Code to Comments

In your Python code all functions, classes, and methods should have a documentation string with at least 1 line of meaningful documentation. The ratio of non-empty source code lines to comments should be no more than 5 to 1.

(next page)

Grading Rubrics

Basic Criteria	Points
GitHub private repository was created correctly, instructor's account was given permissions, directory & files created correctly with exact names given, program runs without stack trace	5
All functions, classes, and methods have a documentation string with at least 1 line of meaningful comment. The ratio of non-empty source code lines to comments should be no more than 5 to 1.	5
sqlite3 database created in the local directory with the correct name	5
All csv files in the Medicare hospital compare data set have a staging table created in the database with the correct name	5
All staging tables have all columns in the corresponding csv file in the correct order, using the correct name, and using the text data type	5
All data from all csv files correctly loaded into the staging tables	10
Hospital ranking MS Excel workbook created as specified with the correct name, all sheets present, ordered, and named correctly, all headers present, ordered, and named correctly, and all data present in the correct cells in the correct format	10
Measures statistical analysis MS Excel workbook created as specified with the correct name, all sheets present, ordered, and named correctly, all headers present, ordered, and named correctly, and all data present in the correct cells in the correct format	10

Programming Productivity Points To be eligible for programming productivity points, your submission of source code in GitHub must meet all of the following criteria: • Final submission must be on time. Late submissions are not eligible. • Final submission must meet all of the basic criteria. • First submission of source code must have been made within a couple of days of the date the assignment was given. • Updates to source code should be checked frequently, every couple of days, until the final submission is made. • A cumulative total of points earned for all of the daily runs of the auto grader will be used to determine the productivity ranking points. Auto grader will be run once per day in the early morning hours.	Points
Top 10%	10
Next 10%	9
Next 10%	8
Next 10%	7
Next 10%	6
Next 10%	5
Next 10%	4
Next 10%	3
Next 10%	2
Next 10%	1

Timing of Submission for Rank Grading

The submission time will be considered in the tie breaker for rank grading. Completing the assignment sooner may give you a high rank for the semester.

Late Penalty

25% per day or fraction of a day. 1 second counts as a fraction of a day.

Technical Difficulties with GitHub Submission

If you encounter any technical difficulties with the GitHub submission, in order to preserve your submission date and time, you must have done all of the following:

- GitHub has a 99.9% uptime.   Any claims of technical difficulties should be rare.
- You must demonstrate a substantial work history – you must demonstrate that you did not wait until a few days before the due date to get started:
    - You must have checked your first version of the source code into GitHub within a couple of days of the day the assignment is given
    - You must have checked in new source code at least every couple of days
    - Your most recent check in of code should have rufn without stack trace, and demonstrate at least 80% of the minimum required functionality
- You must check in source code as soon as possible after the submission difficulty.
- Remember that your local GitHub tracks all updates to your file and stored these in GitHub, which the instructor as collaborator can access. The instructor will look at the local GitHub history of the file. If it shows any work was done after the claim of technical difficulty, the matter will be considered a violation of academic dishonesty.
- Immediately (within 1 minute) take a screen print which clearly shows the error you received on submission and also shows the date and time from your desktop clock.
- Email your instructor within 5 minutes of the error with the screen print attached.   You must also attach source code (do not attach output files).
- If any of these conditions are not met, it will not be considered.