**Code Flow for our D.O.M. and Gerry Project**

*By Dongling Xie, Kaidi Zhang, Kaurik Saxena, Phil Prosapio*

**Introduction**

Due to having four different team members working on this project and us altering our code and files as we learned more throughout the class, the code and sources are a bit scattered. This document’s purpose is to guide you through the exact steps that must be (and shortcuts that can be) taken to produce our final product. It also explains in details what is happening in each part of the code.

**Step 1: Raw Data Source**

The first step of our project is to collect the data that our product is based on from a handful of different sources and via three different means of data collection. The breakdown of our sources is as follows:

Web Scraping - 3 sources

CSV File - 1 source

Database file - 1 source

Excel File (xlsx) - 1 source

*Web-scraped data*

1) City Data Website:<http://www.city-data.com/city/Pennsylvania.html>

This data is scraped by using the code in *City\_Data\_Web\_Scrapping.ipynb*. Major note with this code, **it takes a very long time to run (4 to 6 hours) and will not be able to be completed in one run no matter what. The website will blacklist your IP address for the day after about 900 webpages scraped. I worked around this by running the code and saving off files every 150 webpages (named “*output###.py”*) and then when my connection was terminated, I VPN’d to a new IP address, restarted my code with the *rerun* tag and then once it finished, I manually combined the “*output####.py”* files together to make the final output file of “*all\_cities.py”.* Please let me (Phil) know if you have questions.**

2) Statistical Atlas Website: <https://statisticalatlas.com/state/Pennsylvania/Overview>

This data is scraped by using the code in *Statistical\_Atlas\_Web\_Scraping.ipynb*. It will give us 6 files as: *“files/AllEduAttainment.csv”, “files/AllDetailedEduAttainment.csv”, “files/AllEduAttainmentSexRatio.csv”, “files/ADEASexRatioFemale.csv”, “files/ADEASexRatioMale.csv”, “files/AllBachDegreeByAge.csv”*.

The whole program may take about 25 minutes to finish in order to scrape all information within Pennsylvania range at county level and even longer to get all information at zip-code level. But for our project, we are focusing on county and city level analysis, so zip-code level data will not be used but may be used in further development when the project goes further.

3) Zip Code Data: [https://statisticalatlas.com/state/Pennsylvania/Overview](https://www.zipcodestogo.com/Pennsylvania/)

This is data is scraped by using the code in *Zip Code.ipynb*. It will give us 2 csv files as: “*County\_Zip.csv*”, “*City\_Zip.csv*”. Considering that one county or one city may have several zip codes, I reduced the repetition in zip codes and make every county and city have one zip code in order to combine files in next step.

***Thus if you want to skip the long web-scraping code time, you can just run the next stage of our code’s process using the files defined above for each data source. All the codes are configured to read respective files and generate intermediates from and within the “files” folder of the project.***

*CSV files*

*2016\_PSSA\_School\_Level\_Perfomance\_Results.csv* which we pulled from Kaggle.com

*Databases*

The database file comes from: <https://www.kaggle.com/mchirico/pennsylvania-safe-schools-report#database.sqlite>

The data is processed by using *Data\_from\_Database.ipynb.* This dataset has too many columns whose value is 0 or missing, so we just take several columns that has the most meaningful value to evaluate our school safety condition. The output for this program is “files/*School\_Safety\_Data.csv”*

*Excel*The dataset is from: <https://raw.githubusercontent.com/plotly/datasets/master/minoritymajority.csv>

This dataset is only used for map visualization. It has FIPS information for each county which can be mapped into plotly API to generate the Pennsylvania map on county level.

***Step 2: Data Cleaning***

In order to clean the information pulled from the City-Data website, just run main from ***Updated\_Data\_Scrubber.ipynb*** while making sure to pass in “***all\_cities.py****”* files (the output from the City-Data web-scraping).

**if** \_\_name\_\_ == "\_\_main\_\_": ← Main tag from Updated\_Data\_Scrubber & what it should look like  
 main("files/all\_cities.py")

*Updated\_Data\_Scrubber* will produce three cleaned files that we use in later code:

* 'files/Fully\_Cleaned\_CD\_Data.csv'
* 'files/Fully\_Cleaned\_AEA\_Data.csv'
* 'files/Fully\_Cleaned\_SS\_Data.csv'

For those 6 files scraped from *Statistical Atlas,* the cleaning process is done within the scraping part. No need for another process to clean the data.

***Similar to before, you can skip this step and move on to Step 3 and just use the cleaned files mentioned above.***

***Step 3: Combine files***

In order to combine all of our files together, you need to run the main tag from the ***Data\_Combiner.ipynb*** while making sure that all for methods are being called with files passed in.

Code and files should exist and look like this when you run it.

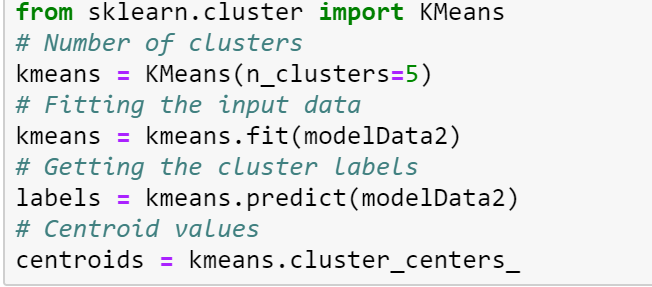
**if** \_\_name\_\_ == '\_\_main\_\_':  
 aeaMerge('files/Fully\_Cleaned\_CD\_Data.csv', 'files/Fully\_Cleaned\_AEA\_Data.csv')  
 combinezips('files/City\_Zip.csv', 'files/City\_Zip\_Lat\_Long.csv')  
 extract\_ethnicity()  
 male\_female\_age\_edu\_combiner()

***Step 4: Take action on files***

*Visualization*

**Using “*Visualization.ipynb”*** we can generate some descriptive graphs based on *“files/Master\_Data\_Files.csv”.* Using the same csv file, we also have more charts created in Tableau. We have basic pie charts, bar charts based on the general data, also a map view which shows the age breakdown in different level as part of the visualizations within the notebook.

*Clustering & Recommendation Preparations*



In this section we will run the Clustering algorithms (k-means) to predict the cluster centres for schools based out of features on safety and educational levels. To do this **simply run the ‘*Clustering.ipynb*’**. This will automatically fetch in files from the resource location and create new files for clusters and centroids within the ‘files’ folder. The code uses the unsupervised k-means clustering from **sklearn** machine learning package.

The notebook produces visualizations for optimal clusters and clustermap along with an **interactive 3D cluster graph** that users can use to examine each school point within different clusters based on its features. **Plotly** package **needs to be installed** for this as specified in requirements. It takes about a minute to load the final 3D visualizations of the clusters.

*Analysis through GUI*

Using **“GUI.py”** we can execute the interface for our product. There are three functionalities that we provide.

The first one is to select city level or county level and input the name in the entry field and then the user will receive relevant information about that city/county. This will output all of the socioeconomic data that we have compiled in our database for that location.

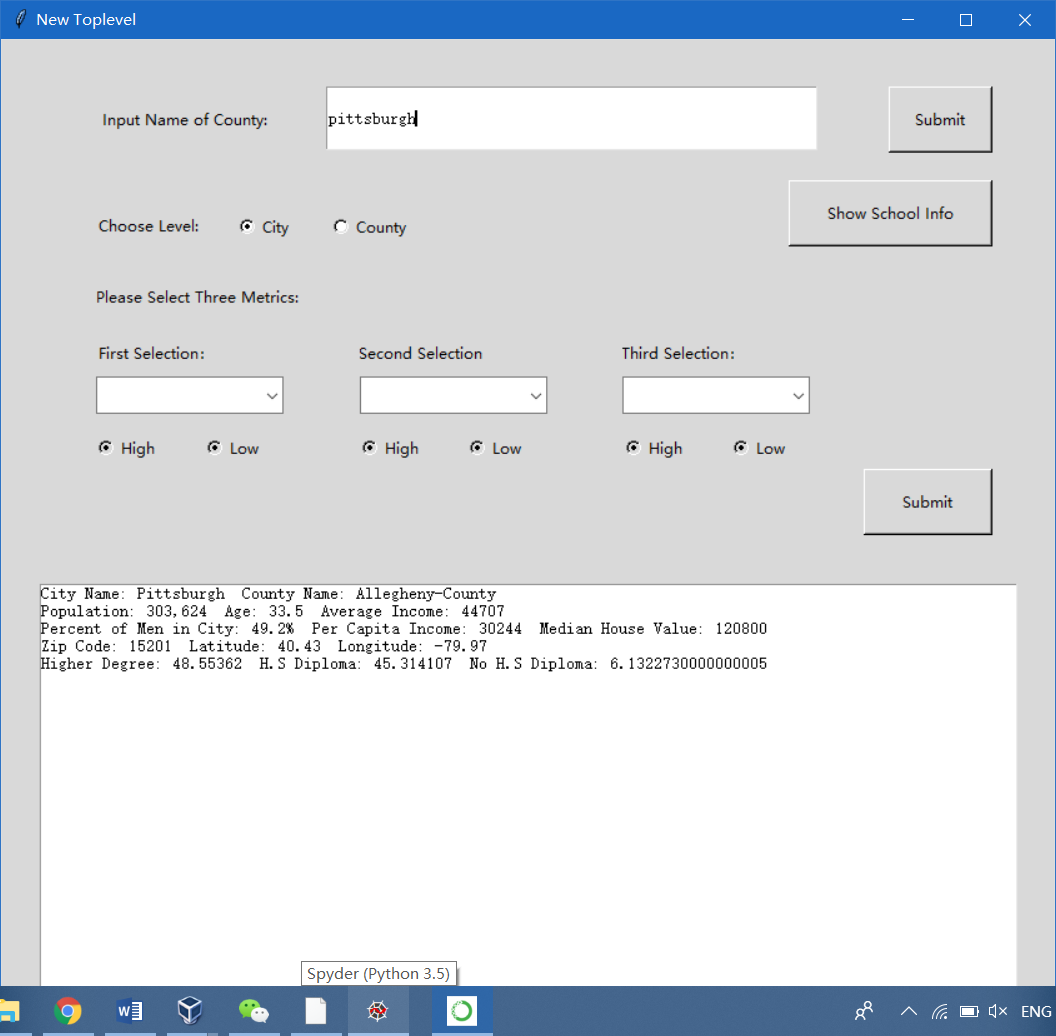
The second is to select three metrics, as well as sorting high-to-low or low-to-high, (columns from our data files) and and these metrics will be sent to a function named “ranking\_data” to generate rankings. The GUI will then output the top 5 cities for each of the three metrics as well as the top 5 cities for the combination of the three metrics you are interested in.

The third functionality is to click the “Show School Info” button and it triggers a new GUI on “school search”.

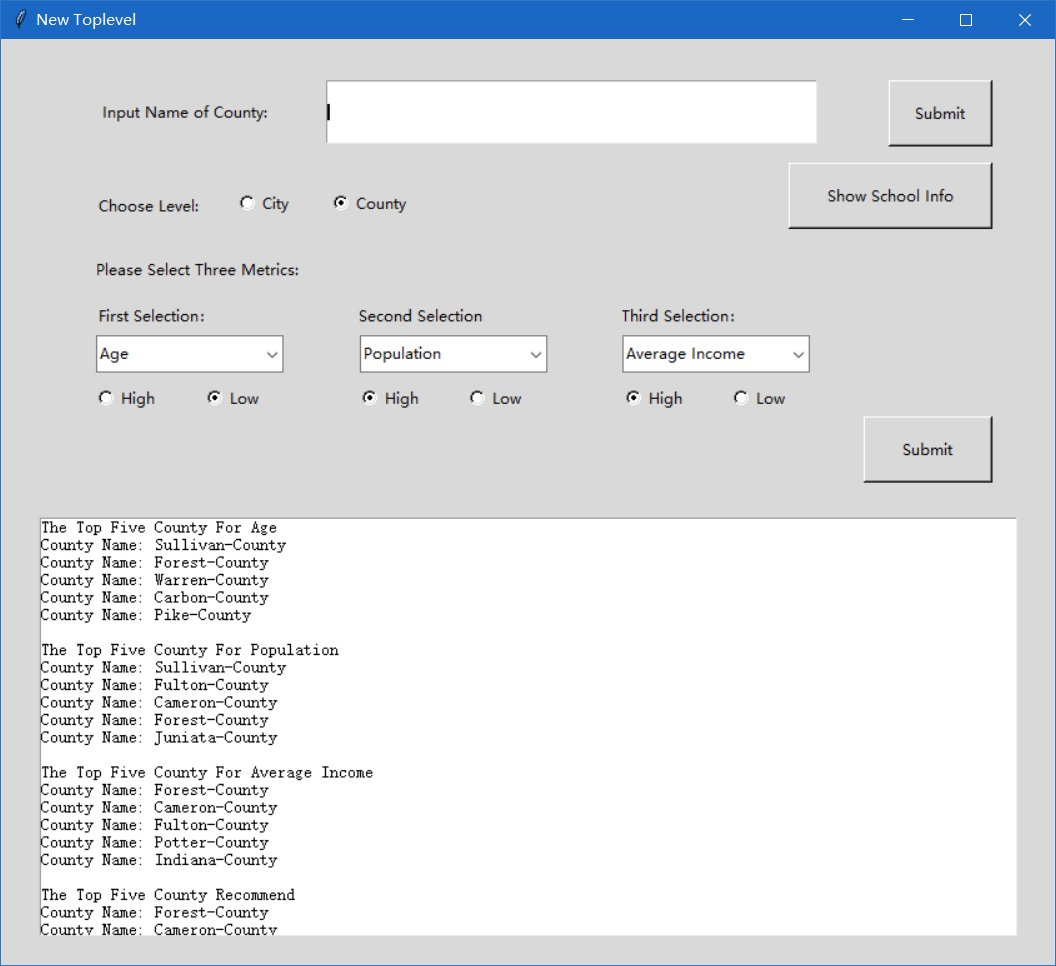
In the new GUI the user can get all information on cluster orientations based on different features by retrieving the cluster centers. User need to press the Run Cluster button to view cluster centers.

Once the user has decided on the perfect cluster fit. He can select it from the drop down menu and it should display all the school recommendations in the GUI. Users also have a choice to search best schools in their county of preference.

Functionality 1:



Functionality 2:



Functionality 3:

