Overview

We chose to do our project on tax delinquency data from Philadelphia, and, knowing how city governments often work, wanted to focus on immediate benefit rather than sophisticated analysis. Our tool automatically creates a variety of visualizations following a data pull, and optionally creates inserts providing directions to the nearest service center for properties that are a given distance from a service center.

Our thought process was that the automatic creation of descriptive visualizations would allow more time for city employees to focus on prescriptive solutions themselves, given their assumed expertise. We, obviously, chose to use maps primarily for spatial visualizations to show the landscape of delinquent properties, while using traditional plots for general descriptive statistics for areas of interest.

The inserts are meant to lower a barrier to seeking out services. Our thought process is that a direct appeal to utilizing the service and providing all necessary information on the steps to solicit service eliminates a real barrier. We use an API to acquire directions and any associated costs of travel for any property that is both delinquent and a given distance from a provider.

Set out to accomplish VS Actually accomplished

It is a truism that government employees are often stretched thin with recurring tasks. So, we wanted to do was relieve the basic burden of acquiring descriptive visualizations. We also wanted to lower a barrier to taking advantage of public services. We noticed that Philadelphia offered a relatively robust dataset on tax delinquent properties in the city, and felt that we could accomplish both goals by:

(1) Automatically creating visualizations upon a data pull. The visualizations provide officials outputs for recurring presentations and can be used to help target areas for more analysis.

(2) Creating an insert to include in delinquency notices that provides direct appeals to service offerings. The appeal includes provider contact information, directions to the provider (driving & transit), and any associated costs/fares.

We accomplished what we sent out to accomplish, but we do see a variety of shortcomings. We would like to include other datasets to provide even more robust visualizations and find a better way to identify properties to receive an appeal. However, we feel that we have established a solid template to build on. Were this a real project in the city, there would be more iterations to fine-tune and improve our work, but we believe that we did our initial scoping relatively well, adapted to changes, and built tools that could be of use in the city today.

Software Structure

Our software is built using four main files that each cover a different set of tasks. We have a fifth file, which simply aggregates the four files and executes a command line input. First, we will provide a broad description of each of the four main files, then explain how they interact via our execution file.

First, we have a file, called 'collect_info.py', that handles standard data collection, cleaning, and manipulation tasks across our software. It fetches two city datasets from Philadelphia's Open Data Portal, cleans it, and provides additional functions to help us crosswalk the data from one file to another as needed.

Second and third, we have our visualization files. The first is a file called 'Data_Viz_Map.py' that creates a variety of interactive maps from our datasets and saves them as html files. The second is a file called 'analyze_data.py' that creates static visualizations focused not only on understanding the literal landscape of delinquencies and agencies, but on providing some descriptive statistics surround amounts due, payments received, and properties per agency. This file's functions will also save any visualization they create in a given folder.

Finally, we have our file for creating the inserts described in the overview. It takes a cleaned dataset and further parses it down to contain only data more than a given distance from a service provider. The file then queries an API to get two sets of directions and creates a word document of the insert, saving it in a given folder.

These four files are tied together in an execution file, 'execution_script.py', that takes command line inputs and executes them calling on our other files as needed. If, say, there is an upcoming meeting to update the Mayor's Office on the state of tax delinquency, we could run the execution file with arguments that would generate all our visualizations and save them in one place. The execution script would use 'collect_info' to fetch and clean the data, then feed that data to our visualization files to create and save the visualizations. If, on the other hand, delinquency notices are being prepared to be sent out, the execution file can be passed another set of arguments. The script would then use 'collect_info' to fetch and clean the data, then use the 'get_directions' file to create and store out inserts for properties that are a given distance from service options.