Travis Ormsby EX04: Python Script

Background

In the US, large transit providers are required to conduct a Service Equity Analysis (SEA) whenever they make a major change in service. An SEA determines if the costs and benefits of the service change are distributed roughly equally between both white people and people of color, as well as between people in poverty and people who aren't. The Federal Transit Administration (FTA) provides guidance to transit providers on how to conduct an SEA in a way that meets the federal regulatory requirement.

Unfortunately, the methods approved by the FTA for assessing the fairness of changes in the transit system suffer from a variety of shortcomings. I was interested in trying to correct three of these shortcomings with my Python script:

- SEA methods aggregate all people of color into a single group. This is a problem because it can mask inequities. For example, if a change in service hurts African Americans and helps Hispanics, the effects might wash out when aggregated.

- SEA methods do not account for cumulative service changes. Each major change in service is considered separately. It is possible that a series of changes that each had minor effects on equity could have a substantial effect when aggregated.

- SEA methods do not account for population change. It is possible that population change could render the transit system less equitable, even if the system itself never changed. For example, if there was a high degree of population growth among low income people in areas that were poorly served by transit, the system is now less equitable.

To solve these problems, transit providers would need to perform more sophisticated analyses more often. Packaging this solution as a Python script automates enough of the analysis that the better solution need not take more GIS resources than are currently devoted to SEA.

What the script does

The SEA process in the script is based on the process used by Metro Transit.

For each census block in the study area, count the total number of transit trips that pass within 0.25 miles of the block's center. These trips are deemed accessible to everybody living in the block. Calculate this value twice - once for the time period before the service level changes under consideration, and again for the time period after the change. The script accomplishes this by spatially joining GTFS transit data to points representing the block centers with a 0.25 mi search radius.

For each population subgroup, calculate the total effect of the service changes. Metro Transit calculates this value from the following formula:

$$\Delta service = \frac{\sum_{i=0}^{n} p_i \, \Delta t_i}{\sum_{i=0}^{n} p_i}$$

Where:

$n =$ *the number of census blocks*

$p_i =$ *subgroup population of the ith census block*

$\Delta t_i =$ *rate of change in trips in the ith census block*

Metro Transit defines four population subgroups for their analysis: Majority (non-Hispanic Whites), Minority (people of color), Poverty (people living below the federal poverty line), and Non-Poverty (people living above the poverty line).

The script solves the first problem of aggregating different groups of people of color by allowing the user to define any subgroups they wish. The script will evaluate the effects on each of those subgroups, as long as the user has a field for the count of that population for each census block

The script solves the problem of not analyzing cumulative service changes by defining the change in trips as the change between time instants, as opposed to the effects of a single major service change. The script takes GTFS data as the input for transit, so passing it GTFS data for November 2009 and November 2019 (for example) will aggregate all of the changes that occurred over the entire time period.

The script solves the problem of not accounting for population change by generalizing the formula used by Metro Transit:

$$\Delta service = \frac{\sum_{i=0}^{n} p_i \left(\frac{\Delta t_i + 1}{\Delta p_i + 1} - 1\right)}{\sum_{i=0}^{n} p_i}$$

Where

$n =$ *the number of census blocks in the study area*

$p_i =$ *the subgroup population of the ith census block at the ending time instant*

$\Delta t_i =$ *rate of change in trips in the ith census block*

$\Delta p_i =$ *rate of change in subgroup population in the ith census block*

The script can also be run in a few different scenarios to get more sophisticated results using this formula. By using the same transit data for both the start and end time (i.e. $\Delta t_i = 0$), the script can isolate the equity effects of population change alone. Similarly, by using the same demographic data for the start and end time (i.e. $\Delta p_i = 0$), the script can isolate the effects of service change alone, a scenario that matches the values returned by Metro Transit's formula.

The script has two outputs. The first is a polygon feature class of census blocks in the study area with attributes for the change in trips/person for each subgroup. The second is a .csv file showing the total population weighted change in service for each subgroup. For both outputs, the values are the rate of change over the time period. For example, a value of 0.2 would indicate an increase in trips/person of 20%, while a value of 0.05 would indicate a 5% decrease.

How to use the script

Because the script runs in an embedded Jupyter Notebook, you will need ArcGIS Pro 2.5 or later. Running the script in an external notebook or rewriting it to a .py file to run at the command line should be possible, but these methods are untested and unsupported. Step-by-step instructions for using the script are listed below.

1) Collect input demographic data. You will need a polygon feature class of census block geometry for the study area. This geometry must contain population counts for each of the subgroups you are interested in, for both the start and the end time instant of the study period. The data can be in a single feature class, or in two feature classes (one for each time instant). If using two feature classes, they must have a field with common values that can be used to join the tables together. The demographic data must be in a geodatabase, not in a shapefile.

2) Collect input transit data. You will need GTFS data for both the start and end time instant of the study period. The GTFS data must include the following files: `trips.txt`, `routes.txt`, `shapes.text`, and either `calender.txt` or `calendar_dates.txt`.

3) Open a new ArcGIS Pro project with a blank map. Add the demographic feature classes to the map.

4) Open the Jupyter Notebook containing the script.

5) In the Notebook, run each of the cells until you reach the section labeled Part 2: Establish parameters and run script. In the cell below that label, you will write all the necessary parameters for the script to run.

6) Set the **`workspace`** variable to the geodatabase where you want the output to be written. The value must be a string. If you are running this script with the example data, you can skip to step 10.

7) The **`start`** and **`end`** variables are dictionaries containing information on the input demographic and transit data for the starting and ending time instances, respectively.

    a) The **`demo`** key is for the name of the feature class holding the relevant demographic data. The value must be a string

    b) The **`gtfs`** key is for the file path of the directory holding the relevant GTFS files. The value must be a string.

    c) The **`agency_ids`** key holds values for the selected agency_id values. These values should come from the `agency.txt` file in the GTFS data. The value must be a list, even if it has only one element. Choosing the correct values requires that you understand which agencies in the GTFS data are relevant to your analysis.

    d) The **`service_ids`** key holds values for all the services that need to be analyzed. These values should come from either the `calendar.txt` or `calendar_dates.txt` files in the GTFS data. The value must be a list, even if it has only one element. Choosing the correct values requires that you understand which services are the correct ones to analyze, and which sets of services are comparable between the start and end time instances.

    e) The **`join_on`** key is for the field in the demographic data feature class that can be used to join the two sets of demographic data together. The value must be a string

8) The **`populations`** variable is a dictionary that holds information on which population groups are being analyzed. It must have a separate key for each group. The value for each key must be a list of two strings. The first string should reference the field name that holds data for the group's population at the start time instance. The second string should reference the field that holds the population count at the end time instance..

9) The **`out_fc`** variable is the name of the output feature class. The value must be a string.

10) The **`out_csv`** variable is the filepath of the output .csv file. The value must be a string

How to interpret the output

The .csv file is the most important piece of output, because it can be directly compared to similar tabular output produced for an SEA by a transit agency. If the table shows marginalized groups experiencing a greater benefit (or lower costs) than high-status groups, the changes in the transit system are acting to decrease the gap in transportation access. If the benefits are about the same, then the transit system is locking in the existing inequities. If the benefits for marginalized groups are lower, then changes in the transit system are actually making transportation access equity worse than it was before.

The output feature class may be useful if it shows any clear spatial patterns. If a particular group is seeing large service decreases in a concentrated area, transit providers can use the map to determine which neighborhoods should be considered for service expansion.

One caveat for using the output feature class is that it will contain non-zero values for blocks where the population of a subgroup is zero. For example, if the number of trips increased 20% in a block group that had no population at either the start or end time instance, the change in service per person would be recorded as 20%. That is a misleading value since no actual people were served.

Time spent

Not including the write-up, I spent approximately 50 hours on this project. About half of that time was spent trying to understand how to model the problem, the other half on writing/debugging the script.