Rideshare Clustering

Austin Taylor, Trevor Clack

Software packages:

Originally, this project used a library called "adjustText" that makes one of our plots more legible. But the library shared directories with numpy and its installation on the raspberry pi seems to alter where essential parts of numpy are stored. Consequently, the installation of adjustText causes python to be unable to import numpy, and it's for this reason that we commented out the parts of our code that rely on adjustText. We would like to note that this problem seems to be related to the filepaths that are followed to find imported directories. It is not a problem that we've experienced on other machines and troubleshooting online has indicated that this is the case.

We do NOT advise doing this on the raspberry pi.

However, the package can be installed on a debian derived linux distro using "sudo apt install adjustText"

Project premise:

Inspired by the stress of coordinating riders and drivers for a music club. This club has around 100 musicians, each of whom have changing availabilities. Each weekend a subset of this (20 or so) go to perform for a local retirement center or homeless shelter. A "gig coordinator" is assigned each week to get in contact with the drivers and riders and it has turned out to be quite a headache. This code is intended to automate it.

The structure of our project is fairly simple, there are five files that are used by default.

main.py
clustfunc.py
Masterlist.csv
going.csv
Locations.csv

Additionally, we've experimented with randomly generated data in the form of DatGen.py which generates two csv files: SampleDatMasterList.csv and SampleDatLocations.csv to test the algorithm with randomly generated data. We run these through the file demo.py, rather than main.py for ease of use, in addition to constraints we've used for testing. Ultimately, we built demo.py for testing and showcasing purposes.

For the five main files listed above, the three .csv files are simply the data that will be read in. The three .csv files are only used to hold data in the following form:

MasterList.csv – date/time,email,name,latitude,longitude,car status,number of seats
going.csv – date/time,email,attendance status(yes/no),car status,number of seats
Locations.csv – locations,latitude,longitude

(Note: car status and number of seats are vestiges of a feature we desired to have but ultimately did not implement. Ideally it would have made clustering even more dynamic, allowing for group sizes to be limited by the space available to each driver but with the approach we made to clustering the groups would have changed format completely).

Of the other two files, main.py is where the code is stored and is the file that runs the program. clustfunc.py contains most of the large functions being called in main.py. We used filename variables at the top of the file so that the administrator can quickly change them without digging through the code. Despite this construction, the sample data generated by DatGen.py cannot be run through main.py because it lacks an event list (e.g. going.csv). We built main.py with two major steps; the first step is focused on constructing dictionaries with the rider, driver, and location information, and the second clusters riders to a location and driver. Once everything has been clustered, we generate two plots, each of which emphasize different aspects of the clustering.

The bulk of the work is done in clustfunc.py where our functions are stored. There 6 functions that we use:

adjaceancy, cluster, LocGrpMatch, DriverToGrp, clusterdraw, LocMap.

Adjaceancy, is used to build an adjaceancy matrix with all the distances between riders making up the elements of the matrix. Adjaceancy feeds into cluster which groups riders together by shortest distance and it uses "complete linkage clustering" to determine the distance between clusters. It is important to note that complete linkage determines the distance bewteen clusters with the largest distance between one of its constituents to the target.

LocGrpMatch matches each cluster to a pickup location. It does so by calculating the epicenter of the riders and matching it to the closest pickup location. Similarly, DriverToGrp matches the chosen locations to the nearest driver, but it also outputs a list of the drivers that weren't matched with a cluster.

The last two functions, clusterdraw and LocMap, are used to generate our plots. Clusterdraw generates a colorcoded plot that shows the connections between locations and the matching riders and drivers. As for LocMap, a plot is generated where each location has a list of the people who were clustered to it.

Outside of the main files we have google forms setup to acquire data from riders and drivers. The forms can found at:

MASTERLIST:
https://docs.google.com/forms/d/e/1FAIpQLSdnKRFPaNHGaOvuUKCrbap0QG8a
Rm6CC2NxCuUMI8nu0VgDjA/viewform

GOING:
https://docs.google.com/forms/d/e/1FAIpQLSfFCQh1hVi4kufxqsU-8ZwCI_sIRgPMi bMp3JMsRj_oeO9K

The master list keeps track of all the people that are in the group and allowed to ride while the event list is used to determine who should be clustered together for that event. The idea is the administrator will maintain the master list, as well as posting new forms for the event lists.

Altogether the data is collected from the forms, which is then pulled into main to create dictionaries. The dictionaries are used to make an adjaceancy matrix which in turn is used to cluster the riders together. The clusters are then matched to a pickup location and a driver, and finally the clusters are plotted better show how people were clustered. This document was created using "pdflatex" command operating on the .tex file

Results:

The program is run simply by running 'main.py'.

As you can see in the figures (final page), Clustering was successful. The data shown was taken from inputs in a user-friendly GoogleDoc survey. Two surveys had to be filled:

-One MasterList which contained the majority of the information as to where the riders and drivers live.

-One "Are you going this week" sort of survey.

Structuring it this way makes each week very easy since the members only need to fill out the "hard data" just once per year at most; each subsequent submission takes less than a minute. In order to make this algorithm less "one-and-done", we've created a DatGen.py and demo.py function which generate random data to illustrate the code more dynamically.

Credits:

There was mauch cross over but the following list reflects the majority work

Austin Taylor:

-adjaceancy
-cluster *****
-clusterdraw
-Latex compilation
-Iterating and finalizing main.py

Trevor Clack:

-LocGrpMatch
-DriverToGrp
-LocMap
-Front end data acquisition (google doc forms)
-Beggining of main.py (Reading into files and creating dictionaries)
-DatGen

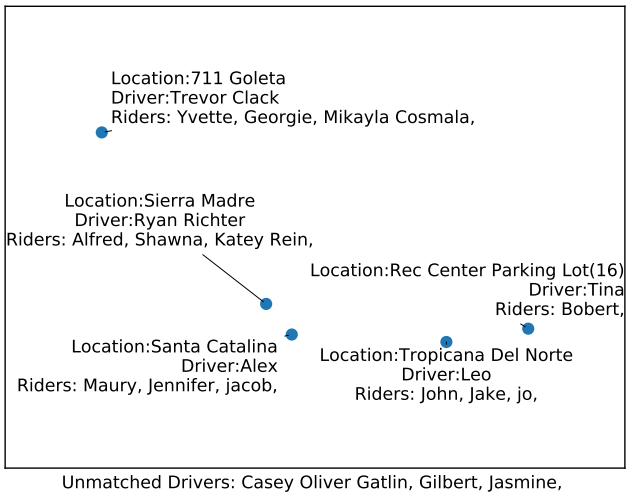

Figure 1: This plot concisely communicates where all drivers and riders will meet

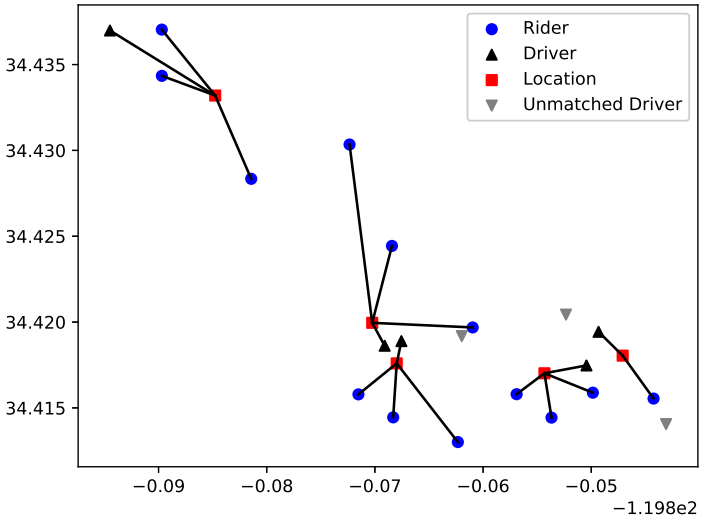

Figure 2: First plot generated emphasizes connections between riders and drivers to their pickup location