The model contained within these python files will depend upon a number of external data files in order to efficiently obtain its desired output. This repository will include a **data** folder which will include the necessary files needed to operate each code file. Below are the files in the order they should be run, their basic functioning, and the files that will be needed for them to properly work.

A note is that this file is complementary to the comments on the files themselves and the report that was made for this phase of the project.

Also, it was not possible to upload many of the heavier files into the repository. Due to this, it will not be possible to take the already set up OSRM environment I talked about previously and the instructions to set it up will have to be followed. Also, the links to download some of the more important data files will be added here:

LODES data: <https://lehd.ces.census.gov/data>

EMFAC data: <https://ww2.arb.ca.gov/our-work/programs/msei/on-road-emfac>

ZIP code tabulation area TIGER files download: <https://www2.census.gov/geo/tiger/TIGER2020/ZCTA520/tl_2020_us_zcta520.zip>

Counties – 2020 – TIGER files download:

<https://www2.census.gov/geo/tiger/TIGER2020/COUNTY/tl_2020_us_county.zip>

Note on GEOIDS:

Many of the data files contained in this document and in the model in general will have GEOID as geographical identifiers. The first 2 digits of each GEOID represent state and the following 3 represent the county. This might be somewhat useful if any modifications to the code done up to this point are necessary. The link below contains some more detailed information on GEOIDs that might end up being useful:

<https://www.census.gov/programs-surveys/geography/guidance/geo-identifiers.html>

Finally, even if it is not part of the model itself, I have also added a file to the repository called Census\_ZIPcode\_filtering.py. Since the data downloaded from the census database can only be downloaded for all ZIP codes in the US for a specific category, this file serves to filter it so that it only includes ZIP codes of the desired county. That being said, there might be parts of the code and specific names of variables that might need to be modified in order to work for other formats of datasets. The link to download census socioeconomical data is as follows:

<https://data.census.gov>

GEOIDtocoord.py: The most basic and first file of the model, its function is to take the LODES data and the tiger shapefiles for California and output a csv file containing the latitude and longitude coordinates for the centroids of each GEOID in California.

Needed files:

tl\_2020\_06\_tabblock20.shp

Tiger shapefile

ca\_od\_main\_JT00\_2022.csv.gz

California LODES data for job type 00

GEOID\_filter\_santa\_clara.py: This file will tale the .csv that was previously generated with coordinates for the centroids of each GEOID in California and filter it for the Santa Clara or any other desired county.

Needed files:

GEOID\_to\_Centroid.csv

Output of GEOIDtocoord.py

fleetdatabase\_santaclara.py: This file will read the EMFAC fleet database for the county of Santa Clara and will edit it so that it can later be joined with the EMFAC emissions dataset.

Needed files:

FleetDB-County-SANTACLARA-2022-P\_T1-GVWR-All-All-Agg-Selected\_Model\_Years-All-ByCensusBlockGroupCode.csv

Fleet database for the desired county

Emissions\_toymodel\_SantaClara: This file will take both the EMFAC fleet and emissions databases and run a series of operations in the two of them in order to obtain an emissions profile for each GEOID in the Santa Clara County. This file can be altered for any   
County other than Santa Clara and it has this file name for clarity.

Needed files:

EMFAC.csv

Emfac emissions data

cleaned\_fleet\_data.csv

Output of previous code containing the processed fleet data

OSMR\_execution\_SantaClara: This file will execute the final model, determining the emissions generated by each commute and establishing the emissions that were done over the ZIP codes their routes passed through. It outputs three .csv files with data on the aggregate emissions for each ZIP code, emissions caused by each ZIP code when that ZIP code is the start of commute routes and emissions caused when a ZIP code is the destination of commute routes. The code is reading some inputs as .parquet files, which are more efficient than .csv that being said, there is no problem working with .csv files instead of .parquet.

Needed files:

tl\_2020\_us\_zcta520.shp

US ZIP code tiger shapefile

santa\_clara\_geoids.parquet

List of Santa Clara geoids

avg\_emissions\_per\_geoid\_SantaClara.parquet

Average emissions per geoid file

Note on data files:

Many of the data files included in this repository are specific to the Santa Clara County, meaning that if analysis were to be done for another county or any other geographical region, it would be necessary to change some of the inputs, such as specifying filtering options for other counties in the GEOID\_filter\_santa\_clara.py script. That being said, most of the code is organized in a way where only very limited parts of it should be changed. It is also important to note that the scripts that were added to this repository have the original file paths for inputs and outputs to which were used when all of the code was originally run during the summer of 2025. These portions of the code MUST be changed, or they will cause errors or organizational problems with files down the line. It should also be noted that due to the way the code is organized, it would be ideal for the user to read the information contained in this file and the comments on the code, so they understand the necessary modifications needed to have the code running in the way they need it. That being said, the principal working of most scripts should be fairly easy to understand and have simple logic behind them. The more complicated parts of the code should not be changed unless one is to alter the model and execution of the model itself, and not only the area it is analyzing.

Note on tiger shapefiles:

In order to utilize the tiger shapefiles, which are required for some of the scripts given here, it is necessary to have a group of different files installed in the same folder, even if only one of them will be accessed at a given time. These files will be included in a folder in the repository to make it easier to get them ready for running. They can also be downloaded externally and will work exactly the same.

Notes on EMFAC data:

If there is any need to alter EMFAC data or download it again using the same settings but for a different county of California, the exact configurations that were used for the data are contained in the report that was done for this phase of the project’s development. It is contained in the appendix for both the emissions and fleet datasets and should be fairly easy to replicate or change if needed.

Notes on OSRM file:

When setting up OSRM, it is also necessary to have osm.pbf files, which are compressed extracts of OpenStreetMaps (OSM) data. They will be chosen based on the area that is to be analyzed by the routing method. The files for both north and south California will be added to the repository, with north California being the one needed to analyze the Santa Clara County. That being said, in case it is necessary to obtain these files for other regions, they can be obtained through the following link:

<https://download.geofabrik.de>

Running the code:

My recommendation for running the different parts of the code, especially the initial scripts, would be to simply download them from the repository and have them run in a local machine using an IDE such as VS Code that can execute python code. For the final script, the one that actually runs the full data analysis for the county, two approaches can be taken:

* Running the google maps API version, which is much simpler considering that it will not depend upon setting up the OSRM server, but will need previous approval and funding in order to be done. In case it is to be run, it is of great importance to first test the script for a limited number of OD pairs, such as 100 OD pairs, so that it is confirmed that outputs are the way that they should be and the code is running efficiently. If and only if that is the case, it would then be advised to proceed with running the entirety of the code. This should be done this way due to the fact that running the model using the Google Maps API would cost around $2200, so it is necessary to be sure that it is indeed working well for the intended purposes before getting to run it.
* Running the OSRM version, which will require setting up OSRM in the cluster environment, something that might ending up being slightly tricky to do.

In both cases, it is necessary to have some understanding of running files in a Linux environment, which might be a bit tricky if one does not have prior experience with it but is manageable and can be obtained by looking at some tutorials online. A note about running python files in such an environment is that a local python environment has to be set up and initialized whenever it is needed to run code there. There are some issues where some python environment setups might not have the most updated libraries, and this might cause some issues when trying to run the code. This was especially an issue with the GEOPANDAS library, which should be in its most current version (1.1.1 at the time of the writing of these instructions) in order for the model to run without any issues. That being said, please check the version of GEOPANDAS before thinking the error might lie in the code itself, as an older version might be the sole cause of any issues that are occurring! In my case, setting up a Conda environment was what worked the best for me and allowed me to get the most updated versions of all libraries. It is also necessary to reactivate the python environment whenever another session is opened, or another prompt is being used.

Now let us move onto the OSRM setup. I will try to go through the steps of setting it up here, but it must be noted that it is very probable that some bugs might arise in any step of this execution, and in the end some of it might be tailored towards fixing these errors. Usually these are linked to a specific file missing from an installation, which can then be located and downloaded (most times) from the actual OSRM github repository, which can be found in this link:

<https://github.com/Project-OSRM>

Moving on to the actual installation, I found this tutorial when I was working with this and it could be of some help, even though it utilizes Sudo, which is not available in the cluster. The link for it is as follows: <https://www.linuxbabe.com/ubuntu/install-osrm-ubuntu-22-04-open-source-routing-machine>

It should be possible to get assistance from the people that work with technical assistance for the cluster in order to make some adaptations to the steps followed.

Below I will go over the steps to set up OSRM and will base them on what I originally did for the cluster. I have tested it recently and it resolved without any issues in the virtual machine, so it should be safe to follow these instructions without any worries (everything after a # is simply a comment):

mkdir -p ~/osrm/{data,profiles,logs,scripts,tmp} # Create workspace for OSRM in the cluster

ls -la ~/osrm # Sanity check to make sure that the folder was created so we can move on

# Creates a place for container images

mkdir -p ~/osrm/images

cd ~/osrm/images

singularity pull osrm-backend.sif docker://osrm/osrm-backend # might be necessary to module load singularity

ls -lh ~/osrm/images/osrm-backend.sif # Checks if image file was created successfully

After this step, it will be necessary to get a .osm.pbf data file and add it into ~/osrm/data, which is the folder for data we created previously. That can be done through the file manager for the cluster, which allows for external files to be uploaded. With this done, we will move on to the next steps. There might be some warnings showing up when running these steps but they should not be of any concern.

# The following lines will run OSRM preprocessing steps. It must also be noted that the these lines have norcal-latest.osm.pbf as an example, but should be changed accordingly to the used .osm.pbf file.

singularity exec -B ~/osrm:/data ~/osrm/images/osrm-backend.sif \

osrm-extract -p /opt/car.lua /data/data/norcal-latest.osm.pbf

singularity exec -B ~/osrm:/data ~/osrm/images/osrm-backend.sif \

osrm-partition /data/data/norcal-latest.osrm

singularity exec -B ~/osrm:/data ~/osrm/images/osrm-backend.sif \

osrm-customize /data/data/norcal-latest.osrm

# The setup should be complete with the above steps, and then all that is left is running the lines of code below to initialize the server whenever needed. Once again, if osm.pbf file is not norcal, please have its name changed.

cd ~/osrm

singularity exec -B ~/osrm:/data ~/osrm/images/osrm-backend.sif \

osrm-routed --algorithm mld --port 5000 /data/data/norcal-latest.osrm

Doing this should allow for OSRM to be set up and for requests to be sent to it so that the model can be run. Using singularity makes the process much simpler than attempting to clone files directly from github and using cmake later. Still, I have ran through many annoying errors when trying to get the environment up and running, so please have patience with the debugging if that’s the case. If it runs smoothly then the model should be ready and will output the correct outputs!

Also, try to keep the actual python scripts and data files outside the osrm folders we created previously.