AutoETL: Move address coordinates to an associated rooftop

Background

This project is a take home coding challenges for ready engineering candidates. The challenge is based on the Git issue linked here (https://github.com/ready/builders-challenge/issues/18). The goal of the project is to take a set of locations and their associated coordinates (lat and lon) and adjust them so that the coordinates are allocated to a structure (building rooftop). In addition to the list of locations provided, geojson parcel and building envelope data was also supplied. The final deliverable should be packaged within an Airflow DAG that allows for one-click ETL automation.

Definitions

Location: A location is an individual address that represents a service point. A location may have multiple buildings associated with it (e.g. an unattached garage).
Building: A building is a structure located on a parcel. A building may or may not have an address associated with it.
Parcel: A parccel is a subset of an overall landmass. A Parcel can have multiple buildings and locations associated with it
Building Centroid: This is a point that falls within the geometry of the building and is used as the updated geo-coordinate (identified as ed_lat / ed_lon in provided geojson data for building)

Datasets

ms_hinds_locations: A csv listing of locations and their associated attributes
ms_hinds_parcels.ndgeojson: A geojson file with the geometry of each parcel in Hinds county
ms_hinds_buildings.json: A json file with the geometry of each building in Hinds county
ms_hinds_buildings_join_table.csv: A join table used to match parcel to building

Created Dataset

parcels_join: A built dataset that lists all parcels containing buildings in Hinds county. It is built by:

1. Loading parcel geojson data and isolating the parcel_id (ll_uuid)
2. Joining this to the provided building join table with the parcel_id to get building and structure id (ed_str_uuid and ed_bld_uuid)
3. Isolating all parcels with a non-null building and structure id (ed_str_uuid and ed_bld_uuid)
4. Joining building data to parcel data by building and structure id to get building centroids

Methodology

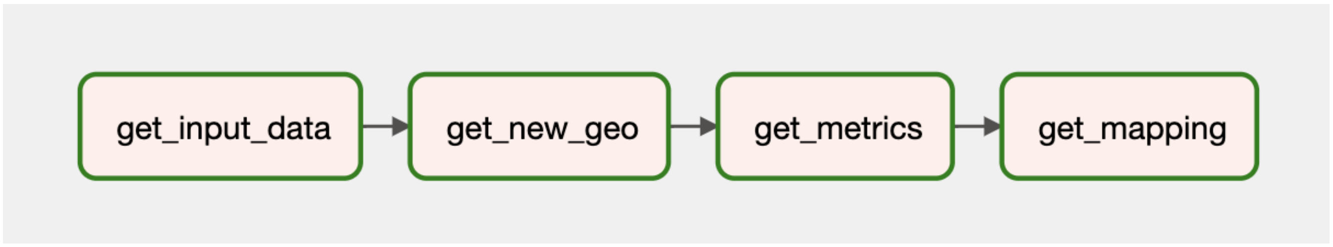
My proposed solution utilizes parcel, building, and location data to find the optimal lat/lon associated with a location. I created a function that defines the various methods to assigning an updated geo-location to each location. The DAG runs this function against every location provided to determine the optimal geo-location. The potential scenarios are as follows:

1. If a location has a parcel_id and only one building is on that parcel, the location will be assigned the building's lat/long as its updated lat/lon
2. If a location has a parcel_id and there are multiple buildings on that parcel, we calculate the Haversine distance between the location and all associated buildings, then select the smallest distance and assign that building's lat/long as the location's updated lat/lon
3. If a location has a parcel_id, but no point lat/long, we simply assign the first building id associated with the parcel as the location's updated lat/long
4. If a location has no parcel_id, look at nearest buildings and assign the closest building lat/long with the parcel. This is done iteratively by filtering all buildings within 111m of the lat/lon and choosing the closest. If no buildings fall within that area we expand to 1.11km and re-check for the closest. We keep expanding until our maximum of 111km. If no buildings fall within that range we do not create an updated lat/lon.
5. If location has no parcel_id or f_lat/f_long, we do not create an updated lat/lon.

This generates a file with each location and its associated lat/lon. We then use this file to create a metrics dataframe and associated map.

Airflow DAG

Airflow was used to build this ETL. A graphical representation of the DAG is shown below. This operates on a one-click basis. It reads from remote Google Drive and stores outputs locally.



Outputs and Metrics

Final locations with updated lat and long (updated_lat, updated_lon) located at:

"~/airflow/data/outputs/locations_updated.csv"

See below for the associated outputs and metrics. Overall metric table located at:

"~/airflow/data/outputs/metrics.csv"

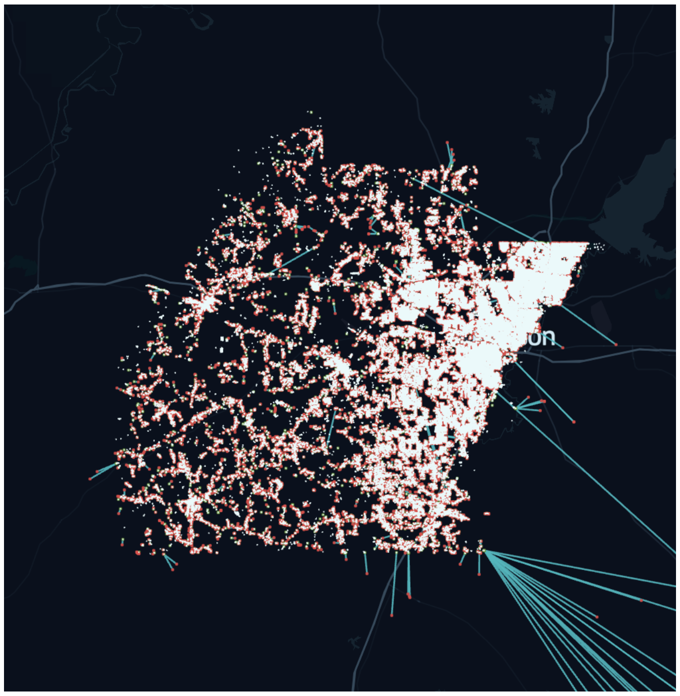
In [5]:
```python
import pandas as pd
import os

cwd = os.getcwd()
metrics = pd.read_csv(cwd + '/data/outputs/metrics.csv')

print('Average distance (km) points moved: ' + str(metrics['avg_distance_moved'][0]))
print('Smallest distance (km) moved: ' + str(metrics['min_distance_moved'][0]))
print('Largest distance (km) moved: ' + str(metrics['max_distance_moved'][0]))
print('Number of Locations without an updated geolocation: ' + str(metrics['no_change_locations'][0]))
```

```
Average distance (km) points moved: 0.035771907500187
Smallest distance (km) moved: 0.0
Largest distance (km) moved: 52.022635642639926
Number of Locations without an updated geolocation: 3724
```

KeplerGL map is best viewed by opening the html file from the associated repo. The file is located at:

"~/airflow/data/outputs/newgeo.html"

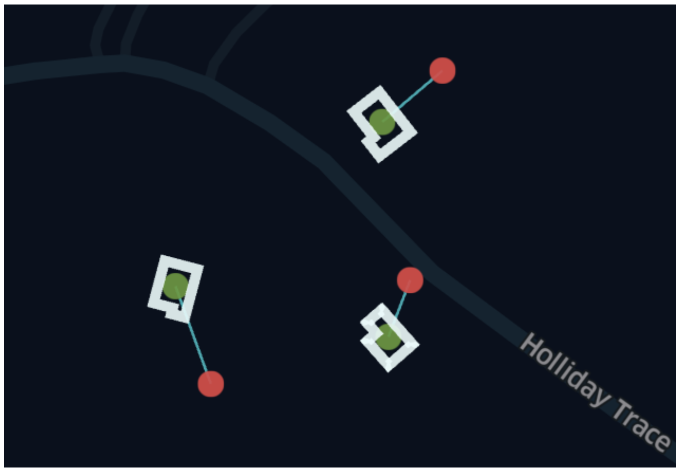
An overall view of the map is:



The map has 4 sets of data:

1. Buildings: represented on the base map as wireframes in white
2. Original Locations: represented as red dots
3. Updated Locations: represented as green dots
4. Lines: Blue lines showing where an original location was moved to

Corrected geo-location movement can be seen here:



Note on Data:

In order to provide a smaller repo, I have removed all input data generated during the Airflow run. In particular, the parcels data is almost 0.5GB. I have left all outputs in the repo for viewing though.

Improvements

There are a number of improvements that could be made to this process:

1. This function could be improved to increase performance. Likely parallelizing or using a technology like PySpark would improve performance. (Takes approx 30 mins to run currently)
2. Cases such as apartment buildings (see 51 NORTHTOWN DR APT with 27+ units) all have same f_lat/f_lon, ll_uuid (f5a2446a-6773-4c64-9f46-0dee770bebe4), and are therefore corrected to the same updated lat/lon. We could improve these multi-address locations by incorporating more data on apartment layouts or attempting to split these locations across the buildings instead of assigning them all to one building.
3. There is zero distance movement for those locations that do not have an existing lat/long to compare to which will cause a slight error in metrics. Positively, these locations are being assigned a lat/lon though which is an improvement.
4. We should identify clear outliers that should not be assigned a new value. Some locations are in completely different areas of the state and should not be assigned a new lat/lon based on our metrics. This will create inaccuracies and should be addressed.
5. We should check whether an original location is already within a buildings geometry, and not move it if it is. This will help cut down on erroneous movement and prevent multiple locations from being assigned the same building.
6. We could count the number of structures within a parcel and compare to corrected number of lat/longs now associated with that parcel to flag potential errors

Resources

Docker setup (https://towardsdatascience.com/setting-up-apache-airflow-with-docker-compose-in-5-minutes-56a1110f4122)
Airflow Packages setup (https://stackoverflow.com/questions/67887138/how-to-install-packages-in-airflow-docker-compose)
Geopandas setup (https://medium.com/analytics-vidhya/fastest-way-to-install-geopandas-in-jupyter-notebook-on-windows-8f734e11fa2b)
Parcel Mapping Background Research (https://www.sco.wisc.edu/wp-content/uploads/2017/07/APPMP_Report_Web_September2014.pdf)
Haversine Distance (https://medium.com/analytics-vidhya/finding-nearest-pair-of-latitude-and-longitude-match-using-python-ce50d62af546)

Troubleshooting Notes on Docker. Make sure to:

Create a dockerfile with pip installs
Update docker-compose.yml to build .
Run docker build . --tag pyrequire_airflow:2.3.2
Run docker-compose up

```
In [ ]:
```