Phase 1: Data Sources and Collection

In this research and development project, I primarily used data sources from the City of Chicago Data Portal (https://data.cityofchicago.org/). The datasets found within this website were made easily accessible by using The Socrata Open Data (SODA) API. The data retrieved from the SODA API were then inserted into relevant PostgreSQL data tables to create a data lake. The datasets collected from the City of Chicago Data Portal, ETL processes taken for each dataset, and PostgreSQL data tables were as follows:

COVID-19 Cases, Tests, and Deaths by ZIP Code

Database Identifier: yhhz-zm2v
Attributes:
- Zip_code
- Week_start
- Case_rate_weekly
- percent_tested_positive_weekly

PostgreSQL Data Table: covid_19_by_zip
ETL: All data pulled from the Socrata API were entered into a Pandas dataframe for further processing. For database yhhz-zm2v, all rows with any missing values were dropped. Additionally, any rows which contained zip_code == 'Unkown' were removed as well. After these rows were removed from the the dataframe, the dataframe index was reset. Lastly, a new feature titled 'zip_code_week_start" was engineered by concatenating the zip_code and week_start values. This feature allowed for a suitable primary key within the covid_19_by_zip table. Additionally, O'Hare specific data needed to be entered into the covid_19_by_zip dataframe. The case_weekly_rate for O'Hare was NA, so there was an additional SoQL call to pull O'Hare specific data and set all case_rate_weekly values to 0 prior to adding the data into covid_19_by_zip.

Taxi Trips
Database Identifier: wrvz-psew
Attributes:
- Trip_id
- Pickup_community_area
- Dropoff_community_area
- Trip_start_timestamp
- trip_end _timestamp
- Pickup_centroid_latitude
- Pickup_centroid_longitude
- Dropoff_centroid_latitude
- Dropoff_centroid_longitude
- trip_miles

PostgreSQL Data Table: taxi_trips

ETL: For this data pull, there was minimal data manipulation before inserting into the PostgreSQL data table. All NA rows were dropped and the pandas dataframe had the index reset. For the taxi trips data, the trip_id served as an acceptable natural primary key so no feature engineering was required.

Transportation Network Providers - Trips
Database Identifier: m6dm-c72p
Attributes:

- Trip_id
- Pickup_community_area
- Dropoff_community_area
- Trip_start_timestamp
- trip_end _timestamp
- Pickup_centroid_latitude
- Pickup_centroid_longitude
- Dropoff_centroid_latitude
- Dropoff_centroid_longitude
- trip_miles

PostgreSQL Data Table: tnp_trips
ETL: For this data pull, there was minimal data manipulation before inserting into the PostgreSQL data table. All NA rows were dropped and the pandas dataframe had the index reset. For the tnp trips data, the trip_id served as an acceptable natural primary key so no feature engineering was required.

Chicago COVID-19 Community Vulnerability Index (CCVI)

Database Identifier: xhc6-88s9
Attributes:

- Community_area_or_zip
- Ccvi_score
- ccvi_category

PostgreSQL Data Table: ccvi
ETL: For the SoQL call, two conditional calls were deployed. First, only ccvi_categories = 'HIGH' were selected. Second, only rows where community_area_name IS NOT NULL were selected. Once this data was pulled and read into a Pandas dataframe, all rows with NA values were dropped and the dataframe index was reset.

Building Permits

Database Identifier: ydr8-5enu
Attributes:

- Permit_
- Permit_type
- Community_area
- Issue_data
- reported_cost

PostgreSQL Data Table: building_permits
ETL: The only modifications to this data set was to drop all NA values and to reset the index before inserting into the PostgreSQL data table.

Public Health Statistics- Selected public health indicators by Chicago community area
Database Identifier: kn9c-c2s2
Attributes:

- Ca
- Community_area_name
- Percent_households_below_poverty
- Percent_aged_16_unemployed
- per_capita_income

PostgreSQL Data Table: socio_economic_data
ETL: The only modifications to this data set was to drop all NA values and to reset the index before inserting into the PostgreSQL data table.

Additional Tables and Feature Engineering:

PostgreSQL Data Table: community_areas
Data Source: CommAreas.csv; Downloaded from the City of Chicago Data Portal.
ETL: The community area number and associated community names were pulled from this .csv file. Then the dataframe was sorted by community area numbers and all NA values were dropped before the index was reset. This dataset was then inserted into the community_areas PostgreSQL data table.

PostgreSQL Data Table: all_trip_info
Data Source: Data from taxi_trips, tnp_trips, and Boundaries - ZIP Codes.geojson. The geojson file was pulled from the City of Chicago Data Portal.
ETL: The purpose of this data table was to combine taxi and tnp trip data and to include pickup and dropoff community names and zip codes for each trip. In order to achieve this, all taxi and tnp trip data were read into pandas dataframes. Then the data sets were stacked and re-sorted by trip_start_timestamps. The dataframe's index was then reset and a dictionary created from the community_area table data was used to map each pickup and dropoff community number to a community name. This information is all sent to a function titled add_zip_insert in order to add the respective pickup and dropoff zip codes for each ride. For each ride, the pickup and dropoff coordinates were transformed into shapely.geometry Point objects. Then loading in zip code boundaries from the 'Boundaries - ZIP Codes.geojson' file, each Zip code are was turned into a shapely.geometry shape object. Once this was set up, each trip was iterated through to check which zip code the pickup and dropoff location belonged to based on the coordinate data. This data was then loaded into the dataframe and added to the all_trip_info datatable.

PostgreSQL Data Table: taxi_trips_info
Data Source: Data from taxi_trips and Boundaries - ZIP Codes.geojson. The geojson file was pulled from the City of Chicago Data Portal.
ETL: The purpose of this data table was to combine taxi include pickup and dropoff community names and zip codes for each taxi trip. In order to achieve this, all taxi data were read into pandas dataframes.

Then the data was re-sorted by trip_start_timestamps. The dataframe's index was then reset and a dictionary created from the community_area table data was used to map each pickup and dropoff community number to a community name. This information is all sent to a function titled write_taxi_trips_info in order to add the respective pickup and dropoff zip codes for each ride. For each ride, the pickup and dropoff coordinates were transformed into shapely.geometry Point objects. Then loading in zip code boundaries from the 'Boundaries - ZIP Codes.geojson' file, each Zip code area was turned into a shapely.geometry shape object. Once this was set up, each trip was iterated through to check which zip code the pickup and dropoff location belonged to based on the coordinate data. This data was then loaded into the dataframe and added to the taxi_trips_info datatable.

Flat Files:

CommAreas.csv
Boundaries - ZIP Codes.geojson
Chicago_geojson.json

Each of these files were downloaded from the City of Chicago Data Portal and stored within the app working directory. They were called as needed for the various microservices.

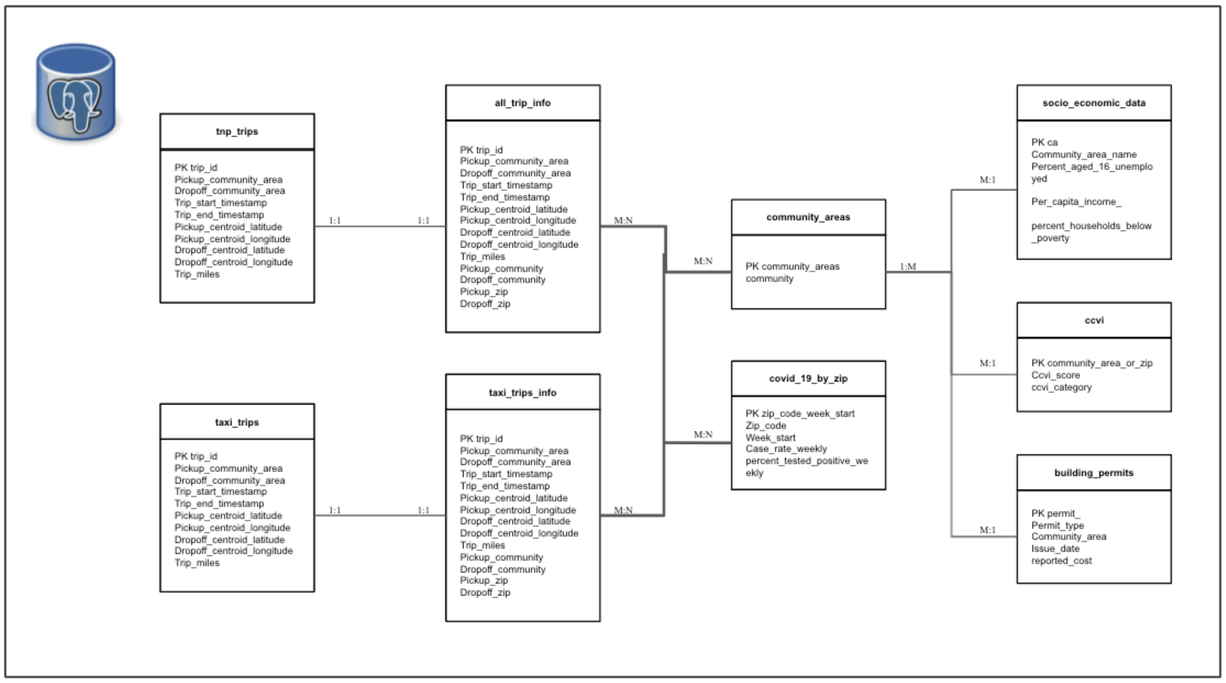
- Talk about batch processing the data, how frequently to update, etc.
- First set up the PostgreSQL Database

All dataset, besides the *_info trip datasets, will be updated via batch processing. Based on interactions with the SODA API, it appears that datasets are updated arbitrarily either on a daily or weekly basis. Therefore, I recommend calling the python file to check for new data entries and update the PostgreSQL database once every 12 hours. However, for production, the batch processing was set to 120 seconds for unit testing purposes.

Due to the computational intensive nature of adding each zip code for each trip, these functions will be called once a day and preferably overnight when it is estimated that app usage rates will be low.

Another important, and tangentially related concept, to Data Sources and Collection is where this data is stored. As mentioned above, a PostgreSQL engine was used to create a CBI Database and store the respective relational data tables. The architecture of that database is shown below:

CBI Database



Appendix:

Data Table: **covid_19_by_zip**

Fields:

- Zip_code_week_start (varchar[50]) PK
- Zip_code (integer)
- week_start (date)
- Case_rate_weekly (numeric)
- Percent_tested_positive_weekly (numeric)

Data Table: **taxi_trips**

Fields:

- Trip_id (varchar[50]) PK
- Pickup_community_area (smallint)
- Dropoff_community_area (smallint)
- Trip_start_timestamp (timestamp with time zone)
- Trip_end_timestamp (timestamp with time zone)
- Pickup_centroid_latitude (numeric)
- Pickup_centroid_longitude (numeric)

- Dropoff_centroid_latitude (numeric)
- Dropoff_centroid_longitude (numeric)
- Trip_miles (numeric)

Data Table: **tnp_trips**

Fields:
- Trip_id (varchar[50]) PK
- Pickup_community_area (smallint)
- Dropoff_community_area (smallint)
- Trip_start_timestamp (timestamp with time zone)
- Trip_end_timestamp (timestamp with time zone)
- Pickup_centroid_latitude (numeric)
- Pickup_centroid_longitude (numeric)
- Dropoff_centroid_latitude (numeric)
- Dropoff_centroid_longitude (numeric)
- Trip_miles (numeric)

Data Table: **community_areas**

Fields:
- Community_area_number (smallint) PK
- Community (varchar[100])

Data Table: **ccvi**

Fields:
- Community_area_or_zip (smallint) PK
- Ccvi_score (numeric)
- Ccvi_category (varchar[10])

Data Table: **building_permits**

Fields:
- Permit_ (integer) PK
- Permit_type (varchar[50])
- Community_area (smallint)
- Issue_date (date)
- Reported_cost (numeric)

Data Table: **socio_economic_data**

Fields:
- Ca (smallint) PK

- Community_area_name (varchar[50])
- Percent_aged_16_unemployed (numeric)
- Per_capita_income_ (integer)
- Percent_households_below_poverty (numeric)

Data Table: **all_trip_info**

Fields:
- Trip_id (varchar[50]) PK
- Pickup_community_area (smallint)
- Dropoff_community_area (smallint)
- Trip_start_timestamp (timestamp with time zone)
- Trip_end_timestamp (timestamp with time zone)
- Pickup_centroid_latitude (numeric)
- Pickup_centroid_longitude (numeric)
- Dropoff_centroid_latitude (numeric)
- Dropoff_centroid_longitude (numeric)
- Trip_miles (numeric)
- Pickup_community (varchar[50])
- dropoff_community(varchar[50])
- Pickup_zip (integer)
- Dropoff_zip (integer)

Data Table: **taxi_trips_info**

Fields:
- Trip_id (varchar[50]) PK
- Pickup_community_area (smallint)
- Dropoff_community_area (smallint)
- Trip_start_timestamp (timestamp with time zone)
- Trip_end_timestamp (timestamp with time zone)
- Pickup_centroid_latitude (numeric)
- Pickup_centroid_longitude (numeric)
- Dropoff_centroid_latitude (numeric)
- Dropoff_centroid_longitude (numeric)
- Trip_miles (numeric)
- Pickup_community (varchar[50])
- dropoff_community(varchar[50])
- Pickup_zip (integer)
- Dropoff_zip (integer)