Part 1 EDA and Wrangling:

1. Missing value analysis: Please refer to “part1/[Wrangling.py](https://github.com/XunPeng715/ADS_homework2/blob/master/part1/Wrangling.py)” for the wrangling code. It is also in “rawDataEDA.ipynb”. We analyzed the missing values of the merged data. The merged data was obtained by merging the training data with a small property data (40MB). First we analyzed the percentages of every column’s missing values and we only handled those columns with reasonable amount of missing values. Then we divide these columns into numeric ones and enumerate ones. For numeric ones, we fill in the mean values; and for enumerate one, we fill in the value that appears the most times.
2. EDA: please refer to “rawDataEDA.ipynb” for details.
3. Docker image: please refer to <https://hub.docker.com/r/xunpeng715/ads_homework2/tags/>.

Part 2: Database:

We use Ubuntu on Amazon EC2 as our web service. Our public DNS name is ec2-35-167-59-55.us-west-2.compute.amazonaws.com. The public elastic IP address is 35.167.59.55.

1. We set up sqlite3 on Ubuntu.
2. In this homework, we write down a python script to import CSV file and then create database and tables in this database and then import data from CSV file to databases. (eg: We use locations.py to import data from properties\_2016\_test.csv to locations\_v2.db)

Part 3: Rest API using flask

1. Install apache webserver and mod-wsgi:

sudo apt-get install apache2

sudo apt-get install libapache2-mod-wsgi

1. Install flask using pip tool

sudo pip install flask

1. Create a director for our flask app.

mkdir ~/flaskapp

sudo ln –sT ~/flaskapp /var/www/html/flaskapp

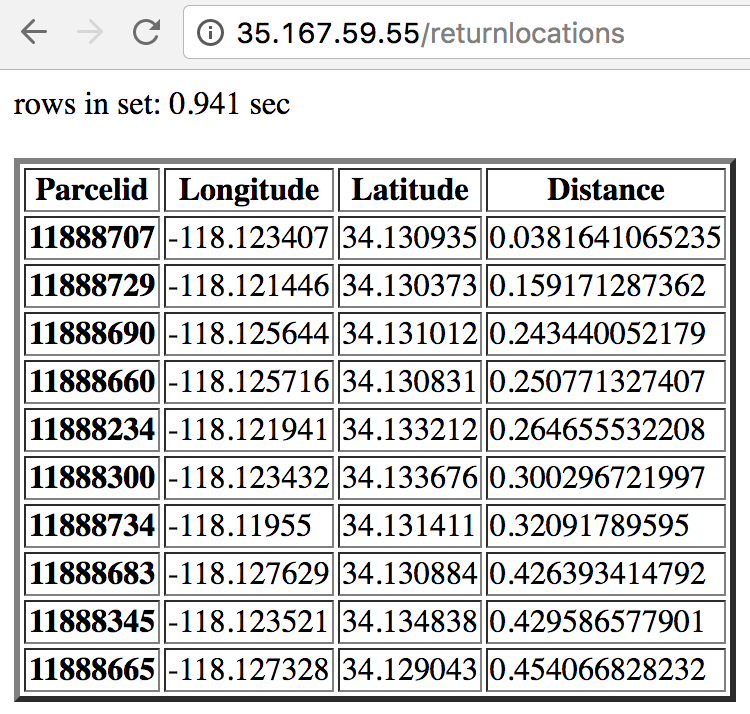
1. We can create app named flaskapp.py. This application will handle all the url that visited from client.
2. Visit <http://35.167.59.55/> you will get to the default index interface <http://35.167.59.55/location> url will return you form about longitude and latitude, and number of closest locations you want to return. And click submit button it will return a table with these number of closest locations and time the application spent on search. <http://35.167.59.55/d3> will return a EDA graph through d3.js

Part 4: Geospartial Search

Search algorithm was implemented in GeospartialSearch3.py

Idea of this algorithm, the time complexity of this algorithm is O(n):

1. We deal with those locations as objects(class), each object has 4 attributes (parchlid, longitude, latitude, distance).
2. I created a list with capacity 10. And we keep this list sorted over time.
3. We just add the location to list when its size was smaller than 10 and then sort list.
4. We need to throw the last location in list if the new location is closer then then last one and then add this new location to this list and then sort the list
5. We do not have to do anything if location is far then the last location in the list



part 5: D3

Please visit <http://35.167.59.55/d3> for a web report that visualizes the property build year density time series using D3. This part was done for presentation purpose. Please refer to “part5/buildYearDensity.html” for details.

