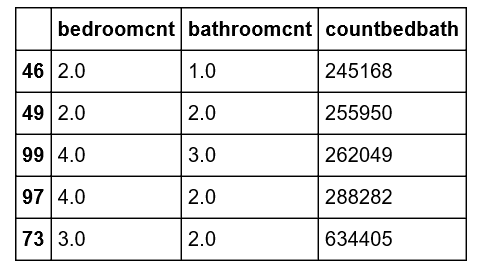
# Report

## Creating Data as a Service

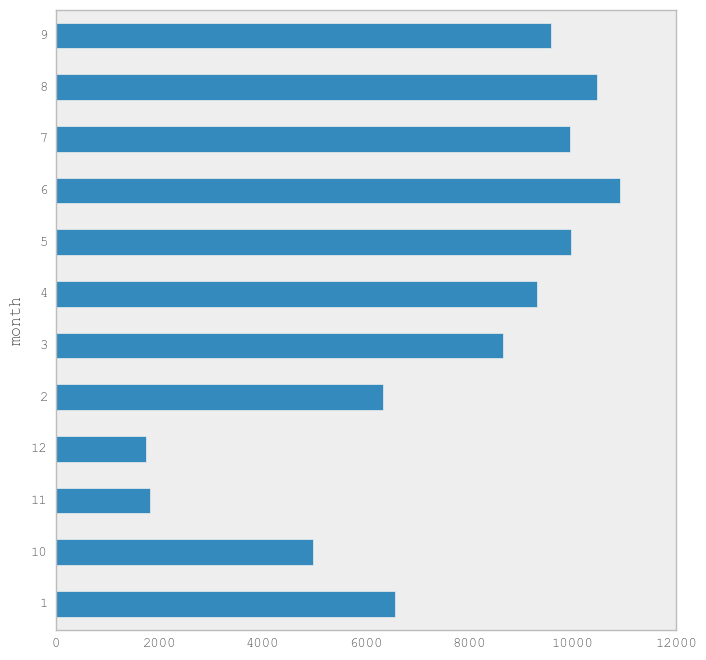
1. Data Ingestion, EDA, Wrangling:

* The first step is to download the Zillow dataset from Kaggle containing properties file, Training data and data dictionary.
* The insights of the data can be found from RawDataEDA.ipynb which is a ipython jupyter notebook.
* The insights from the data are as follows:
  + - * + The most common bed and bath combination found out in most of the Zillow properties is as follows:

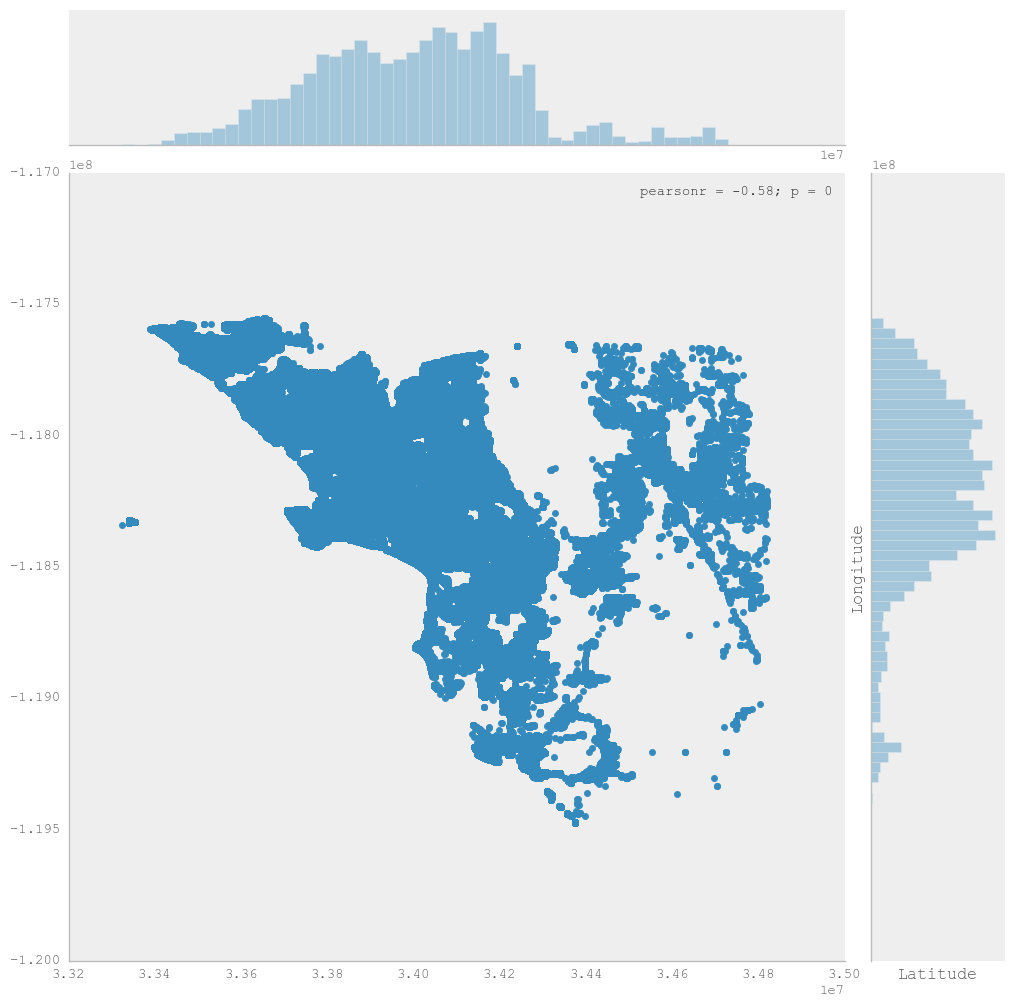


We can observe that the combination of 3 bed rooms and 2 bathrooms is the house type which is mostly recorded and followed by 4 bed and 2 bathrooms.

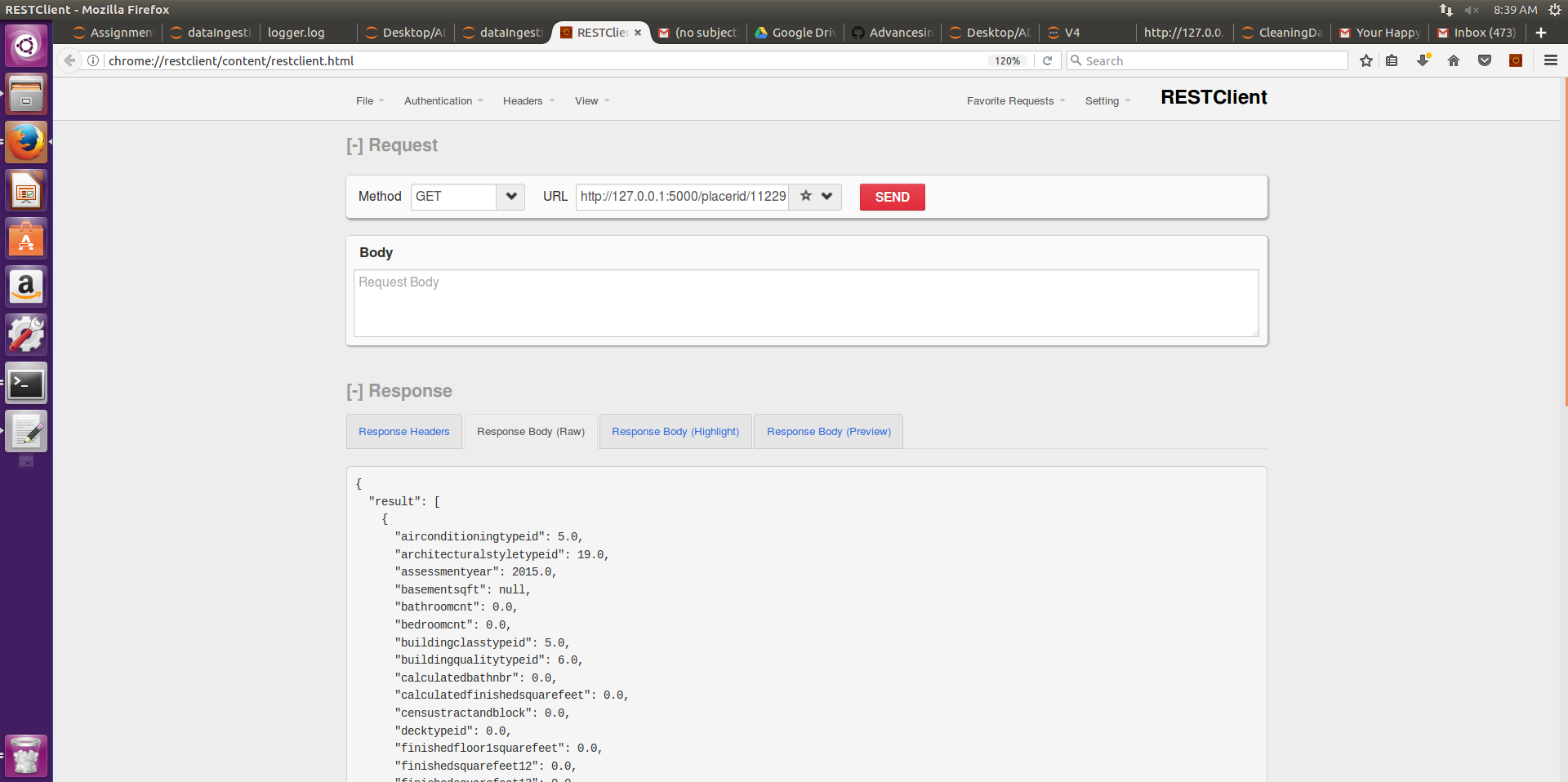
* + - * + When we merge the files of training and properties we can get that most number of sales happened in 2016 was in June month whereas minimum was in December month. We need to apply a left join so that we can get all the values of training data and replacing values of properties data with Nan where the values of properties data from training data is not available.



* + - * + The tax paid by consumer was roughly 0.72 dollars per Sq. ft. in 2015 can be inferred from this
        + There are total 11,564 number of properties which has all three amenities i.e. Fireplace, garage and hot tub/pool.
        + The most used AC type is central whereas least used is Evaporative cooler which is a very old technology.
        + We can see where the maximum longitude and latitude at which properties are located in a rough map by using Seaborn librarary



1. Create a DBaas(Database as a Service):
   * + We were assigned the Google BigQuery database for storing the clean data that we created using the CleaningData.ipynb jupyter notebook.
     + Using command line we executed the task of uploading data to the Google BigQuery Database.
2. Create a Rest API to serve the data:
   * + The next task was to create a Web services REST API to serve the data from the BigQuery database.
     + We used Flask to execute this task and developed our own API.



* + - * The above shown screen shot displays the GET method we designed where in we can enter a parceled with the API key of which we need the details and it will return a JSON response with every details of the particular parcelid.
      * This is how the JSON response looks like:

<http://127.0.0.1:5000/placerid/11229347>

method - GET

output

{

"result": [

{

"airconditioningtypeid": 5.0,

"architecturalstyletypeid": 19.0,

"assessmentyear": 2015.0,

"basementsqft": null,

"bathroomcnt": 0.0,

"bedroomcnt": 0.0,

"buildingclasstypeid": 5.0,

"buildingqualitytypeid": 6.0,

"calculatedbathnbr": 0.0,

"calculatedfinishedsquarefeet": 0.0,

"censustractandblock": 0.0,

"decktypeid": 0.0,

"finishedfloor1squarefeet": 0.0,

"finishedsquarefeet12": 0.0,

"finishedsquarefeet13": 0.0,

"finishedsquarefeet15": 0.0,

"finishedsquarefeet50": 0.0,

"finishedsquarefeet6": 0.0,

"fips": 6037.0,

"fireplaceflag": null,

"garagecarcnt": 0.0,

"garagetotalsqft": 0.0,

"heatingorsystemtypeid": 14.0,

"landtaxvaluedollarcnt": 2077.0,

"latitude": 34526913.0,

"longitude": -118050581.0,

"lotsizesquarefeet": 3817.0,

"numberofstories": 1.0,

"parcelid": 11229347.0,

"poolcnt": 0.0,

"pooltypeid10": null,

"propertylandusetypeid": 261.0,

"rawcensustractandblock": 60379107.091112,

"regionidcity": 40227.0,

"regionidcounty": 3101.0,

"regionidneighborhood": 0.0,

"regionidzip": 97330.0,

"roomcnt": 0.0,

"storytypeid": 0.0,

"structuretaxvaluedollarcnt": 0.0,

"taxamount": 174.21,

"taxdelinquencyyear": 0.0,

"taxvaluedollarcnt": 2077.0,

"typeconstructiontypeid": 14.0,

"unitcnt": 1.0,

"yardbuildingsqft17": 0.0,

"yardbuildingsqft26": 0.0,

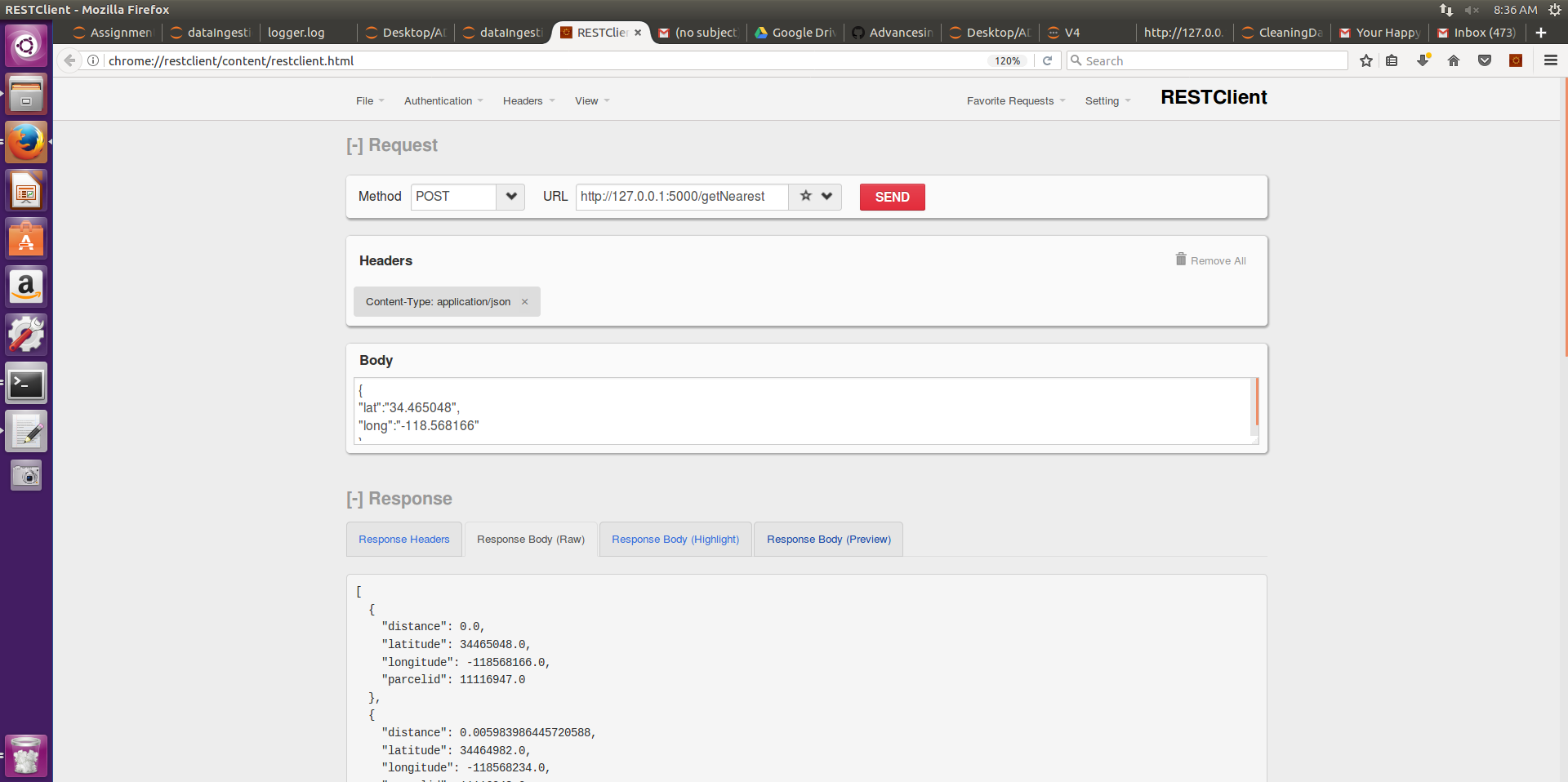
"yearbuilt": 0.0

}

]

}

1. Enhancing your REST API: Geospatial search:
   * + The fourth step is to enhance the REST API using the Geospatial Search which will return the 10 closet distances and the places with its latitude and longitude.
     + We designed a post method for the same as below where we send a json with Latitude and Longitude and get the nearest results.



* + - * We incorporated a SQL query in the Google Bigquey to enhance the search and display the results in 0.03 seconds of response time.

SELECT parcelid,latitude, longitude, SQRT( POW(69.1 \* (ifNULL(latitude,0)/1000000 - 33.975549), 2) +POW(69.1 \* (-118.398761- ifNULL(longitude,0)/1000000) \* COS(ifNULL(latitude,0)/ 57300000), 2)) AS distanceFROM latlong.zillow ORDER BY distance limit 10;

<http://127.0.0.1:5000/getNearest>

Method type - POst

Header -application/json

{  
"lat":"34.465048",      
"long":"-118.568166"  
}  
  
  
The JSON response will look like this:

[

{

"distance": 0.0,

"latitude": 34465048.0,

"longitude": -118568166.0,

"parcelid": 11116947.0

},

{

"distance": 0.005983986445720588,

"latitude": 34464982.0,

"longitude": -118568234.0,

"parcelid": 11116948.0

},

{

"distance": 0.010930769323787398,

"latitude": 34464922.0,

"longitude": -118568282.0,

"parcelid": 11116949.0

},

{

"distance": 0.022628290171355,

"latitude": 34465358.0,

"longitude": -118568294.0,

"parcelid": 11116910.0

},

{

"distance": 0.023675325368298725,

"latitude": 34465370.0,

"longitude": -118568308.0,

"parcelid": 11116911.0

},

{

"distance": 0.024731560072508964,

"latitude": 34465382.0,

"longitude": -118568322.0,

"parcelid": 11116912.0

},

{

"distance": 0.025795864242328465,

"latitude": 34465394.0,

"longitude": -118568336.0,

"parcelid": 11116913.0

},

{

"distance": 0.026867278896212297,

"latitude": 34465406.0,

"longitude": -118568350.0,

"parcelid": 11116914.0

},

{

"distance": 0.03490495438499597,

"latitude": 34465544.0,

"longitude": -118568050.0,

"parcelid": 11116919.0

},

{

"distance": 0.035146944448325425,

"latitude": 34465386.0,

"longitude": -118567705.0,

"parcelid": 11116946.0

}

]