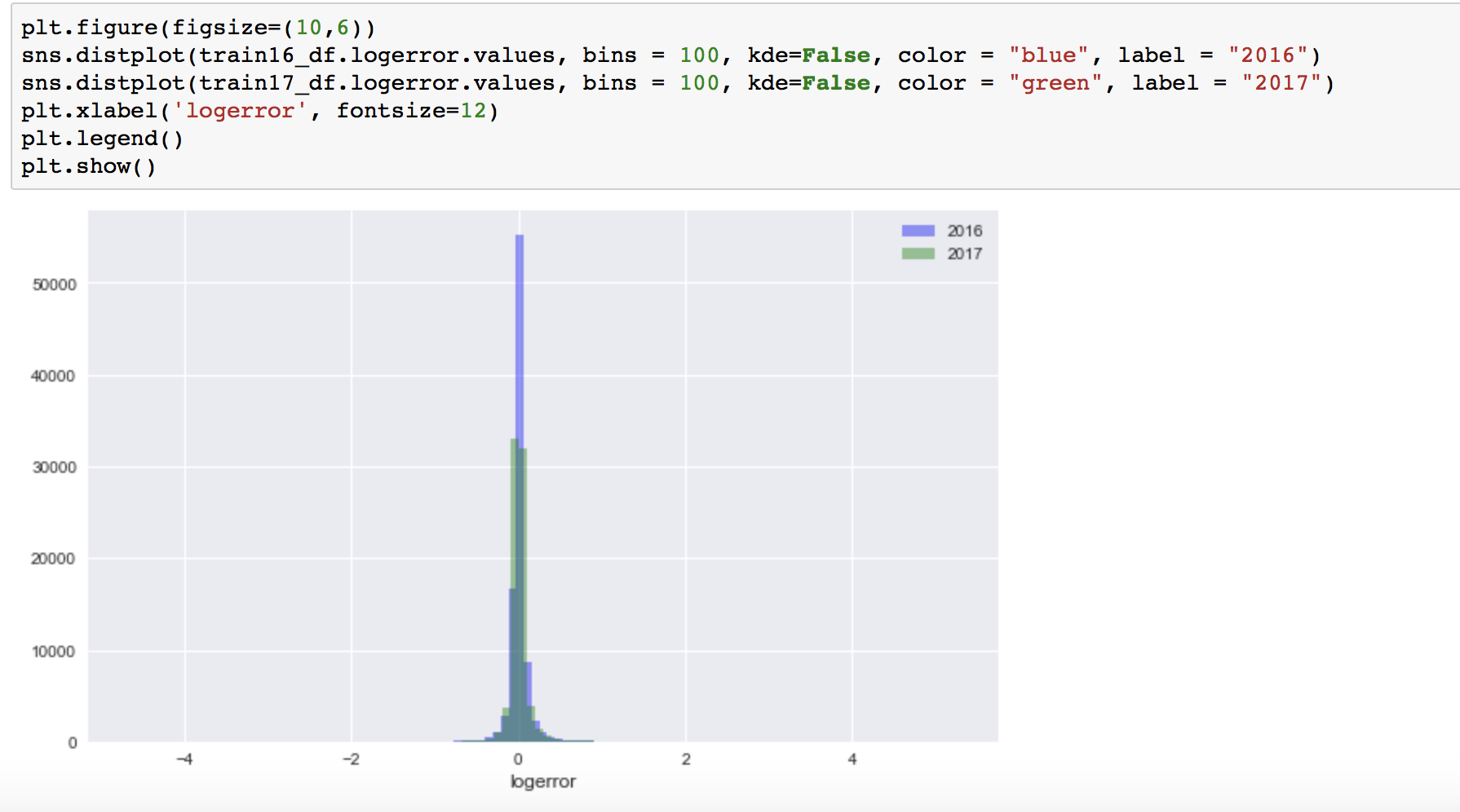
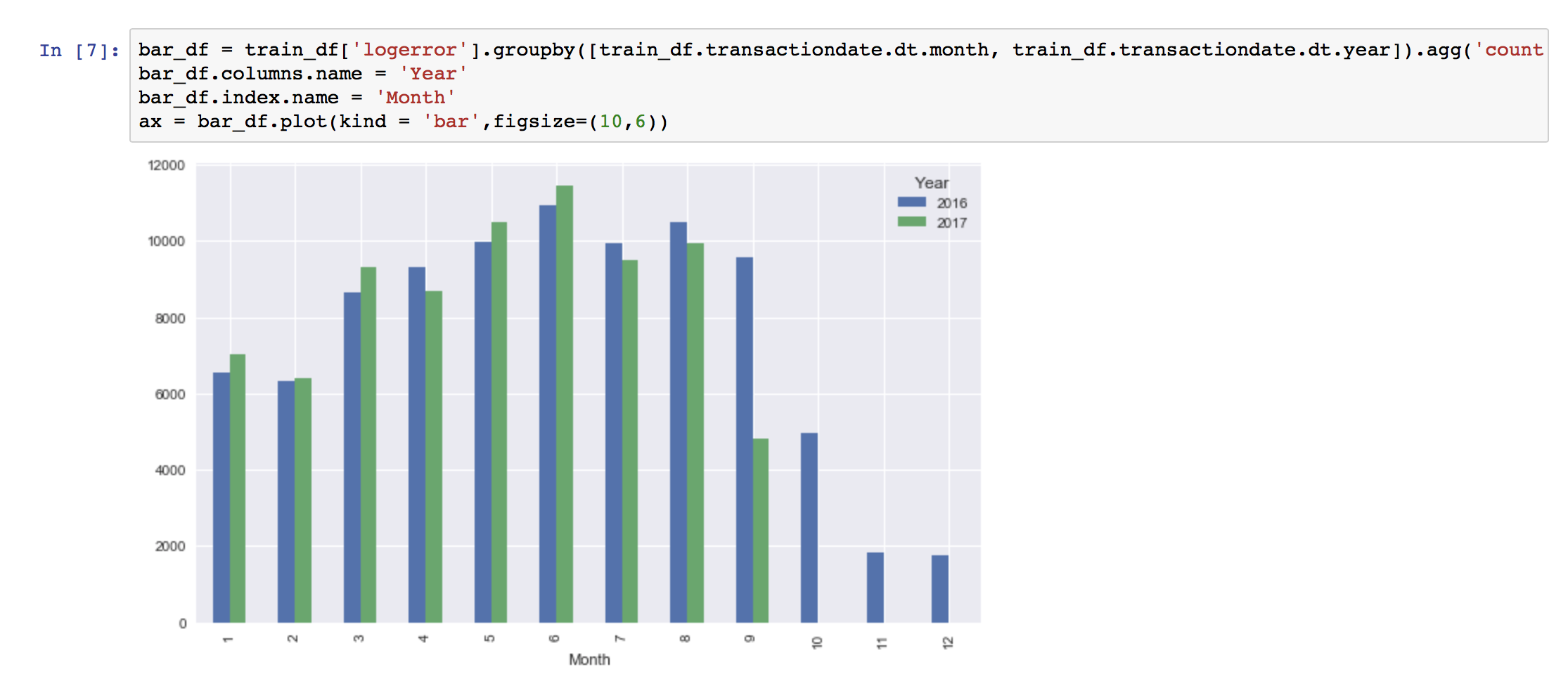
### Part 1: Data Ingestion, Wrangling, EDA

EDA performed:

1. Histogram – to see the distribution of log error in the dataset over the years. We see that we have a nice normal distribution



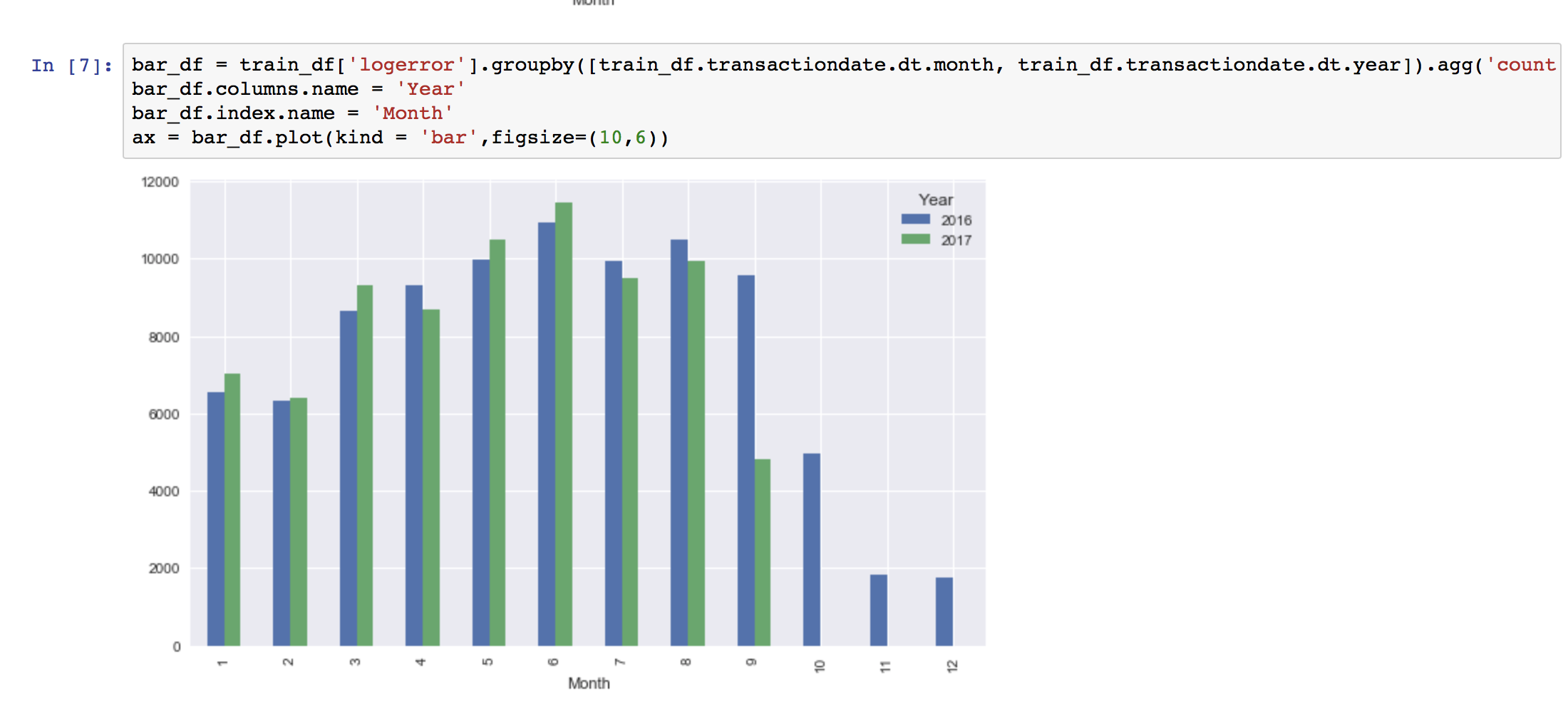
2) Sales per month bar graph to see the market conditions



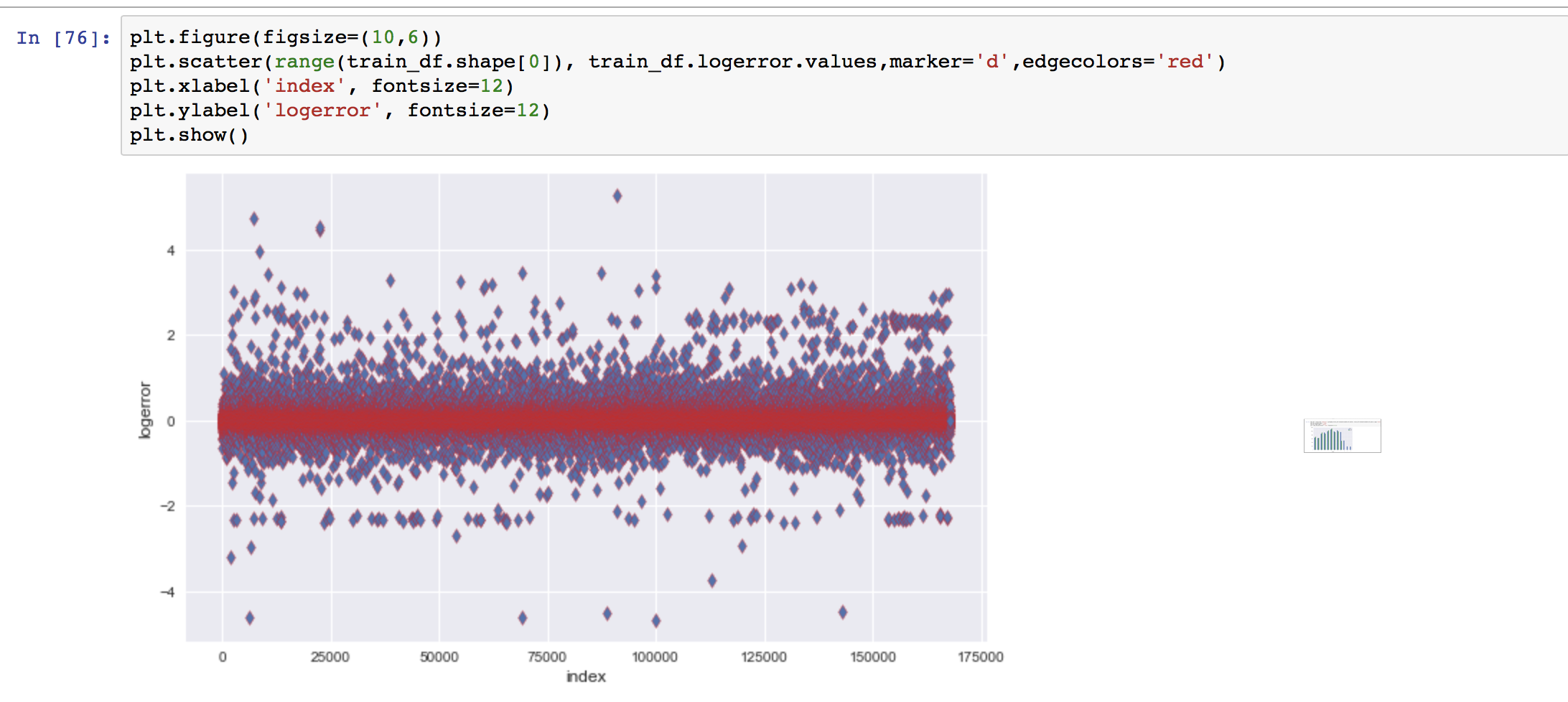
1. Trend of the logerror over the years per month to compare with the sales data. We see that the logerror reduces when sales are high. This shows that the zestimate is better for good market conditions.



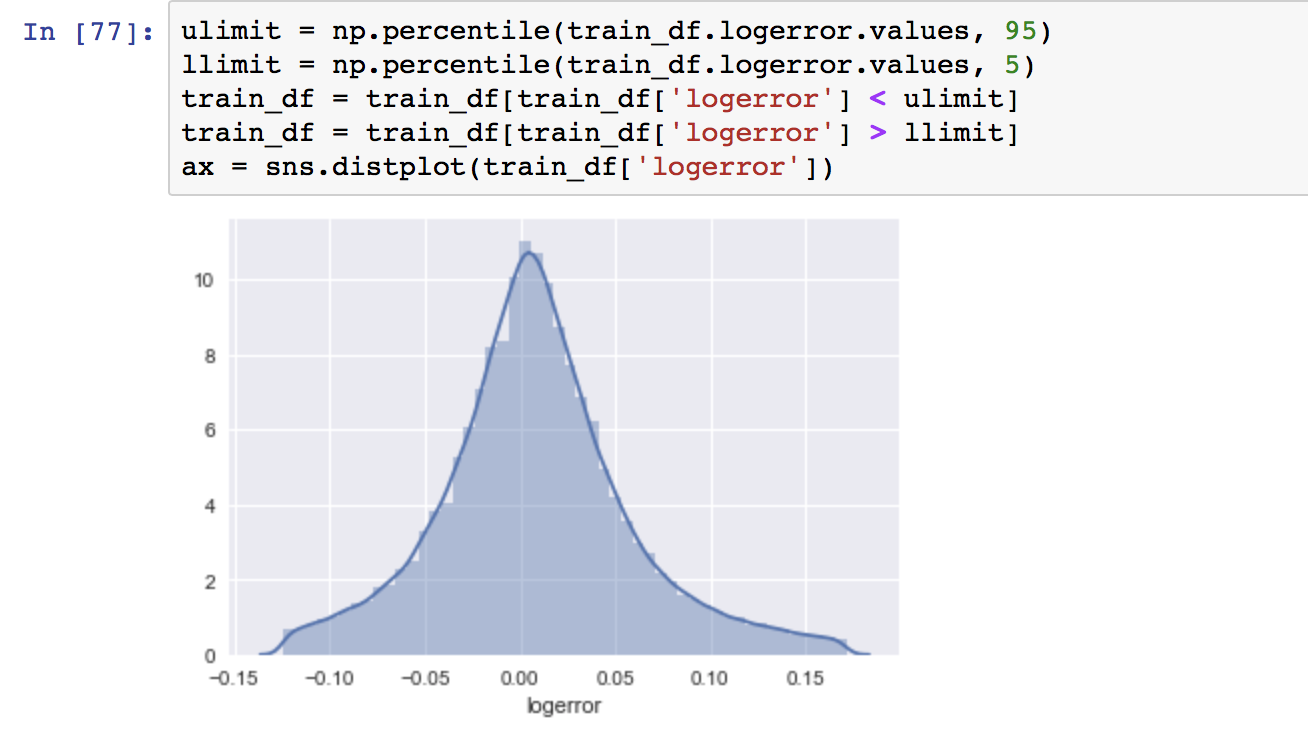
4)



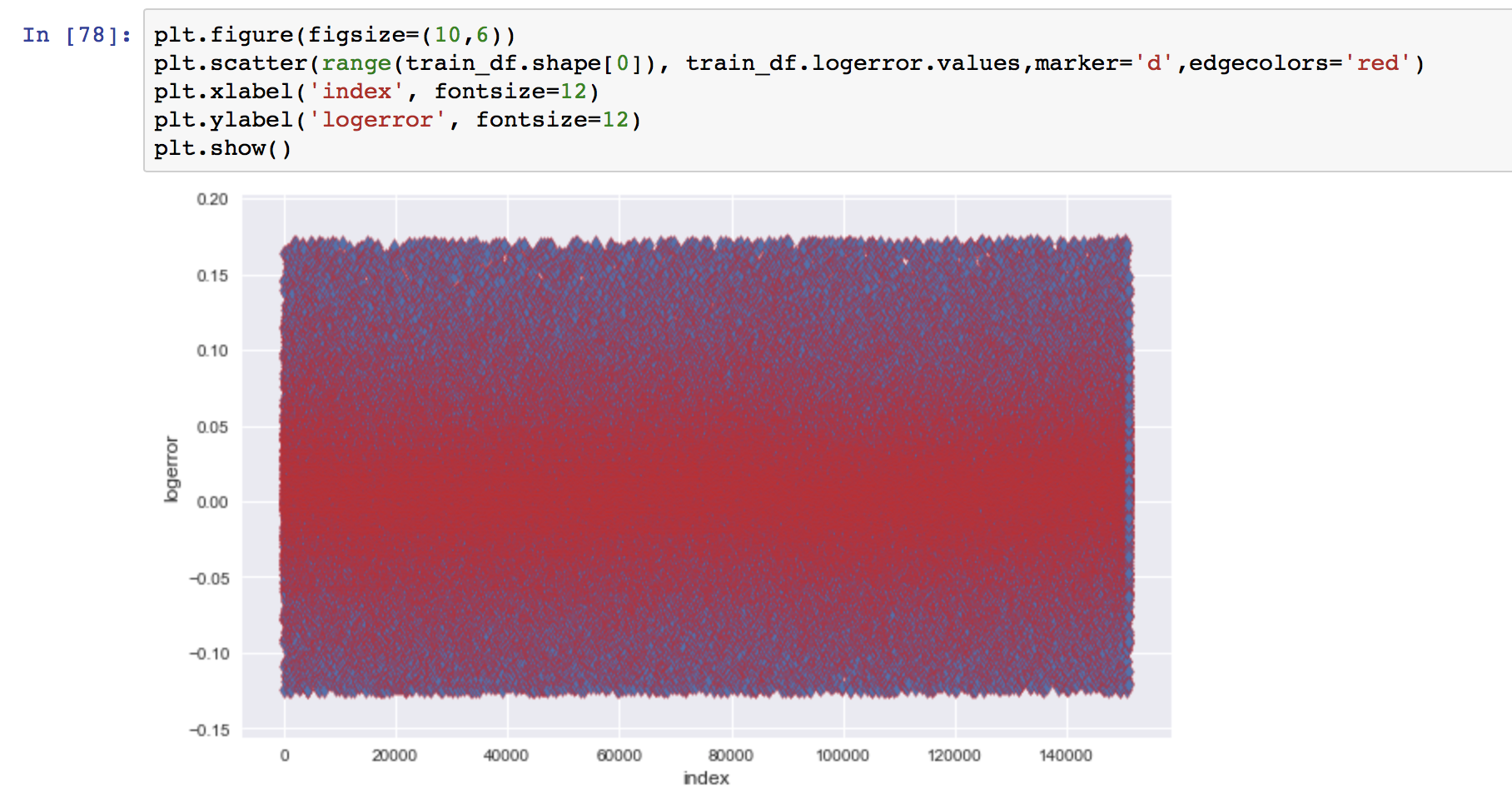
5)Scatter Plot for logerror shows that there are a few outliers. We need to get rid of them



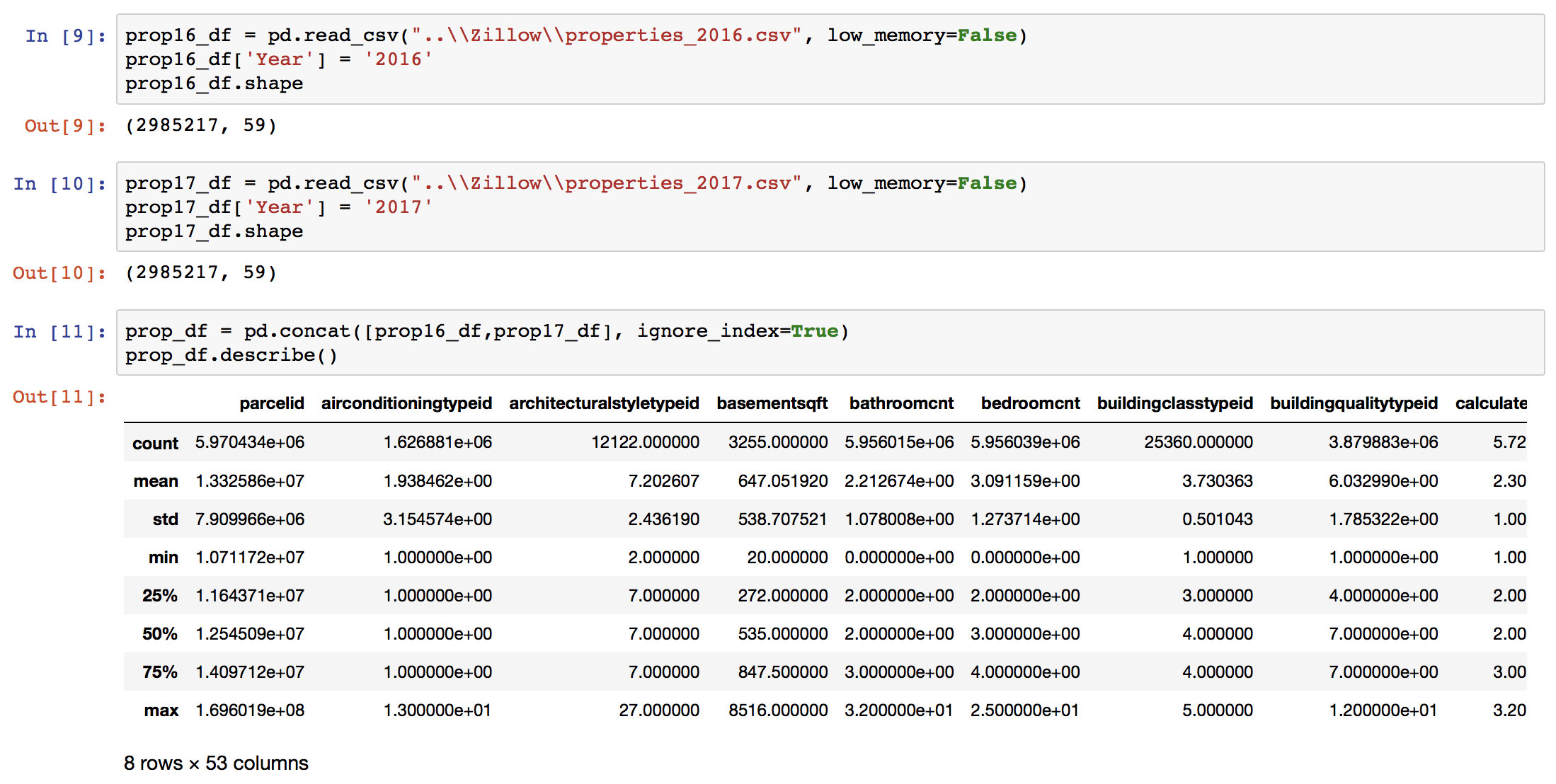
1. After getting rid of the outliers we have the data for the 95th percentile. This also a nice normal distribution.



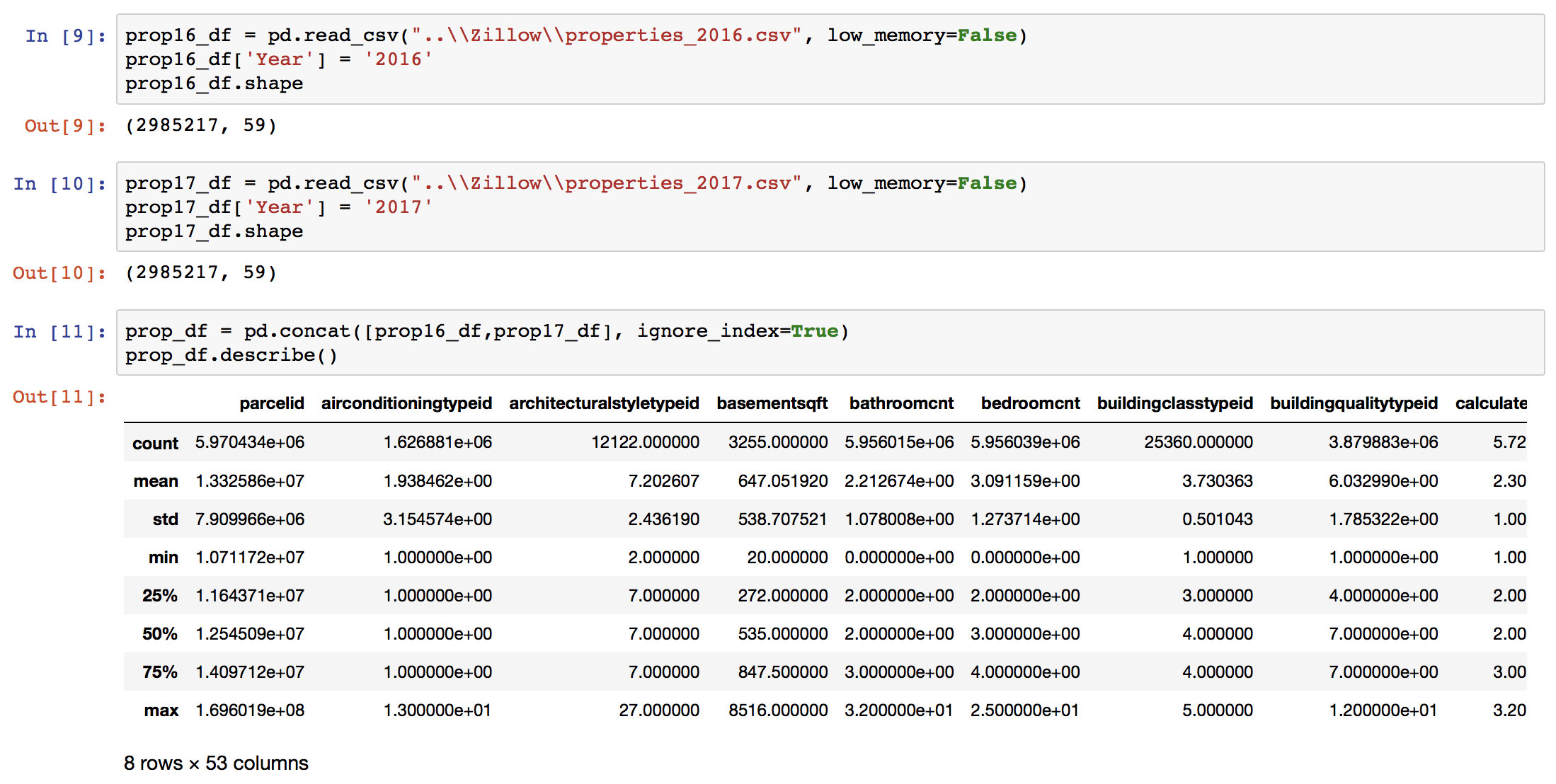
8) Scatter Plot after removing the data outliers shows that we now have a nice tight dataset. This will help with regression.



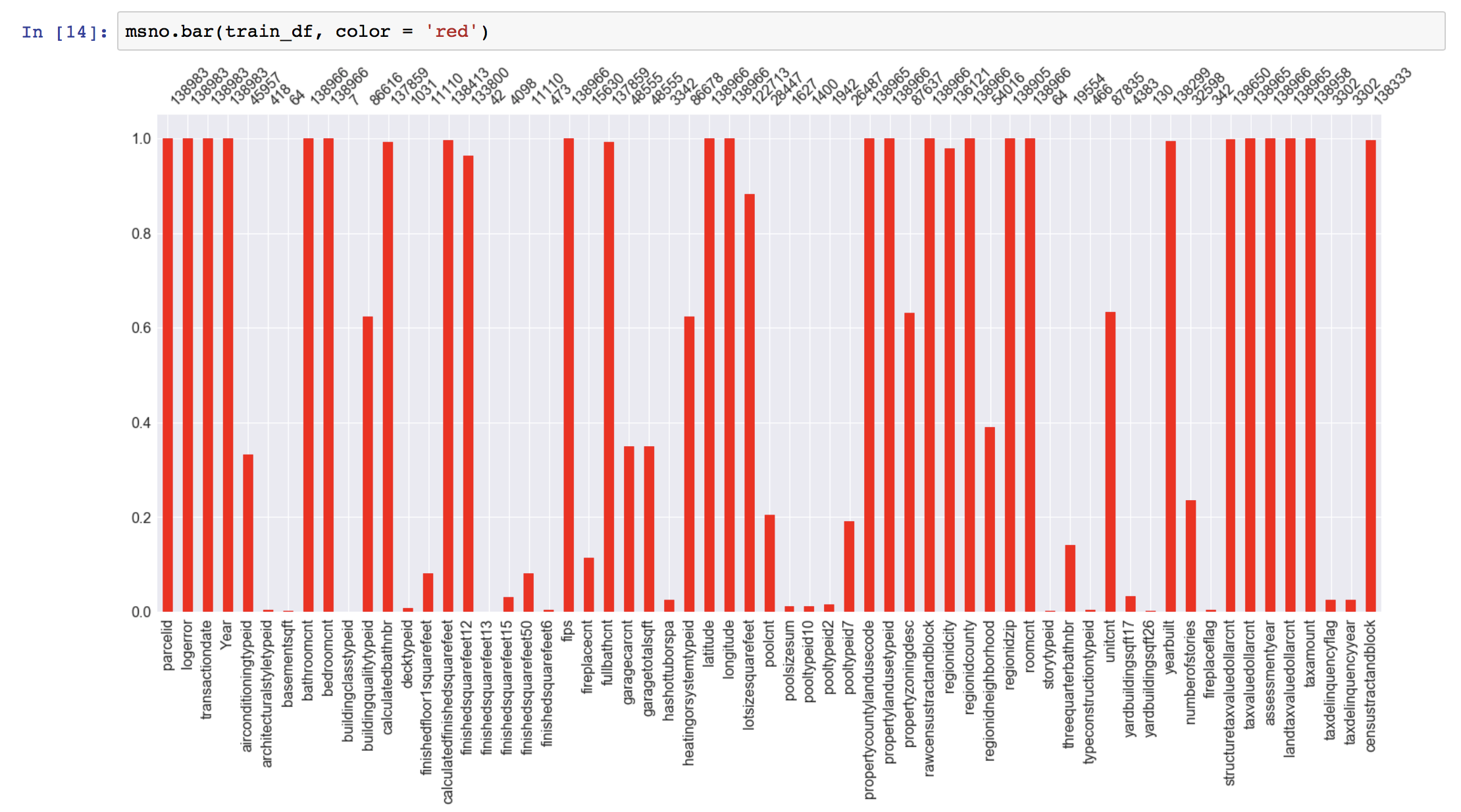
1. Concatenating data per requirement.



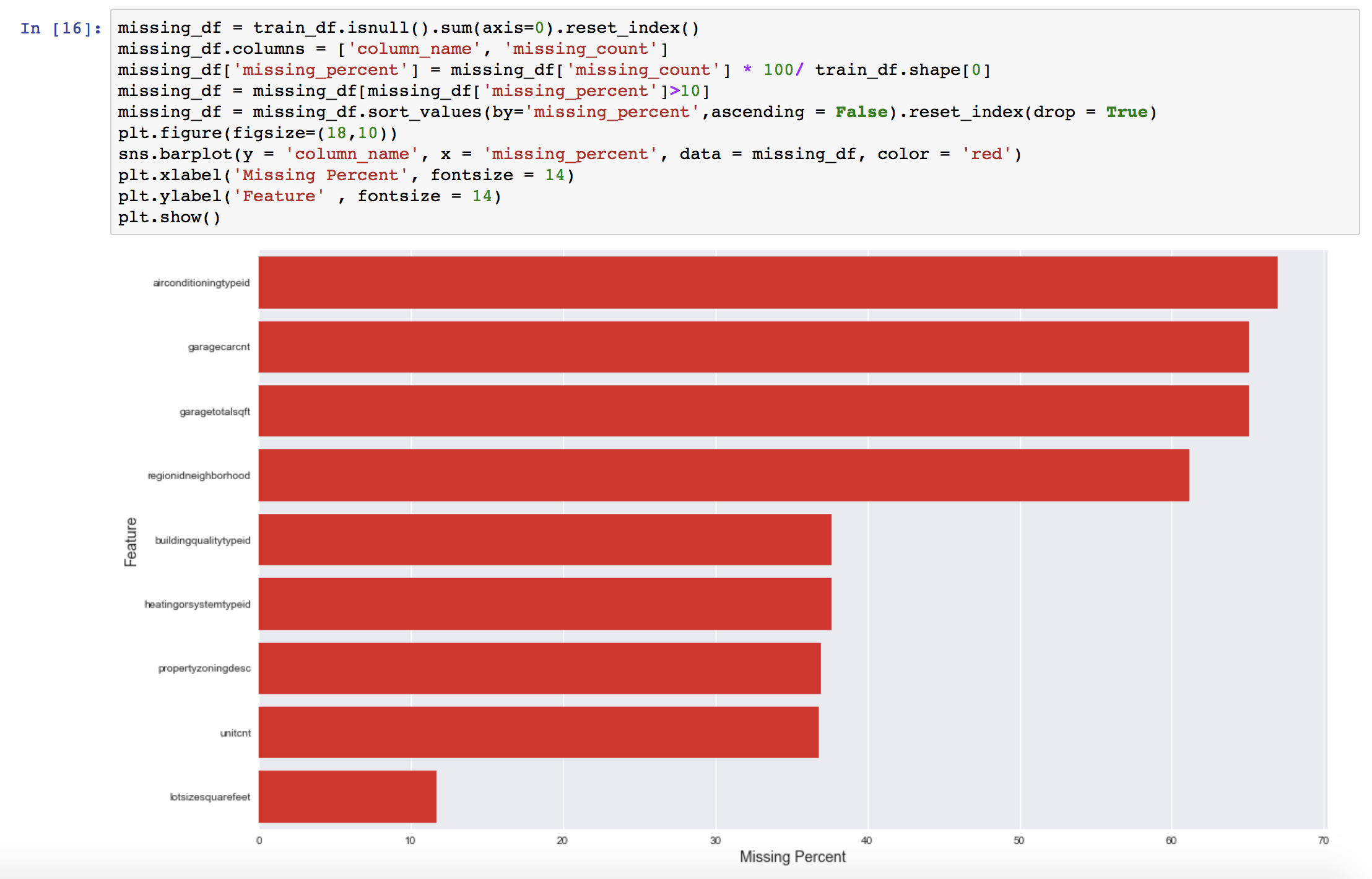
9)



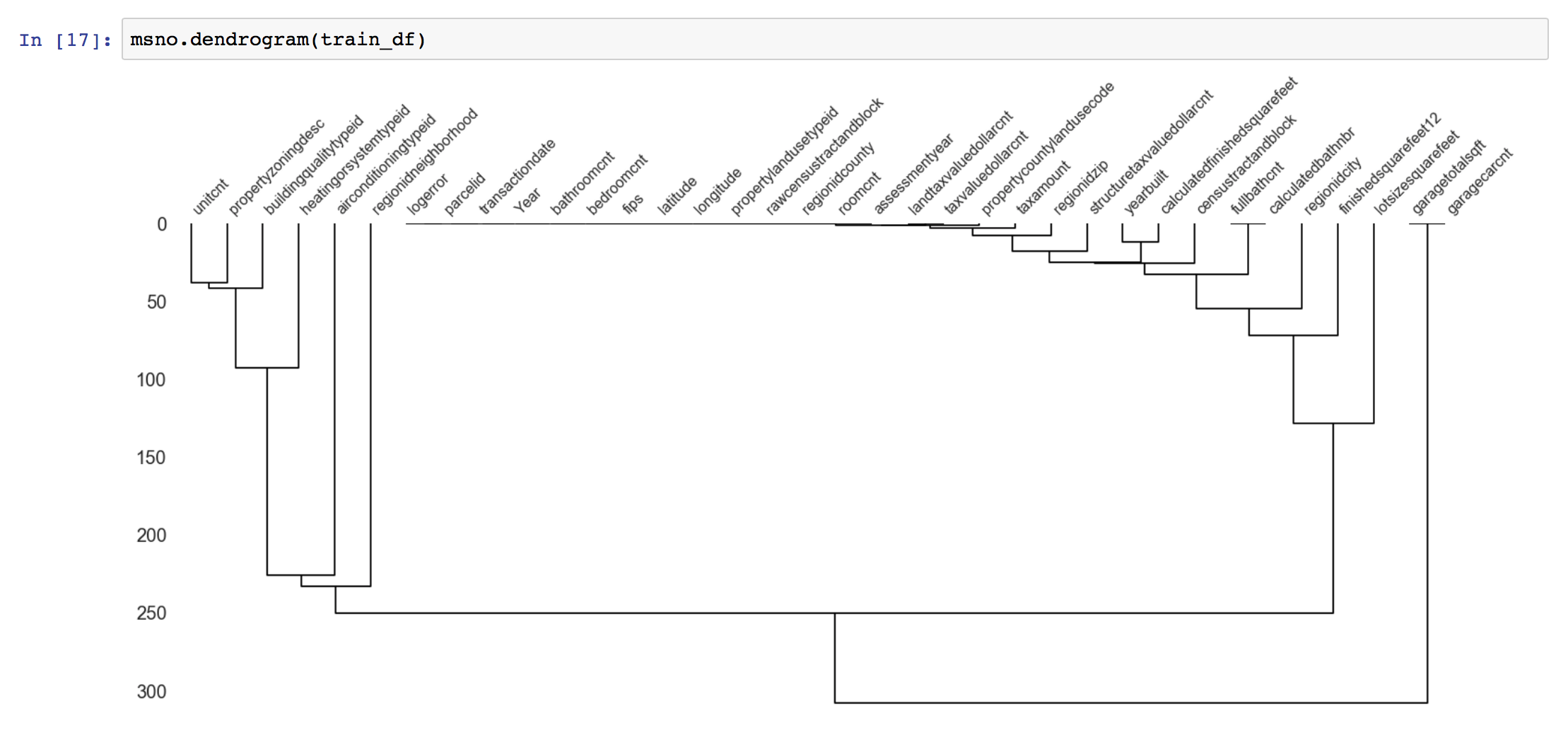
10) Getting the data density. This graph shows how densely are each of the columns packed in the dataset.



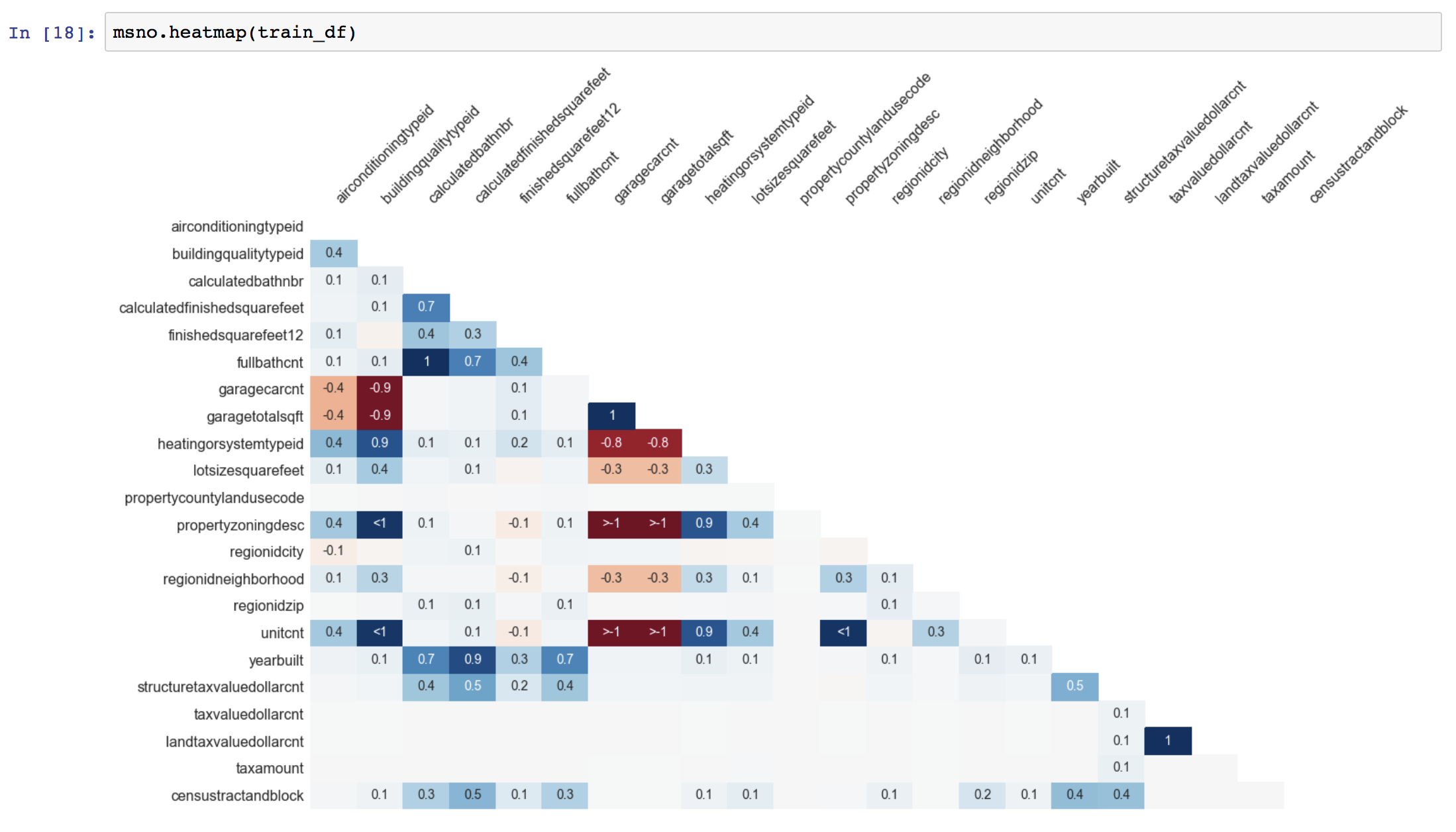
11) Getting the percent missing values in the columns after removing the columns that have missing values more than 75%



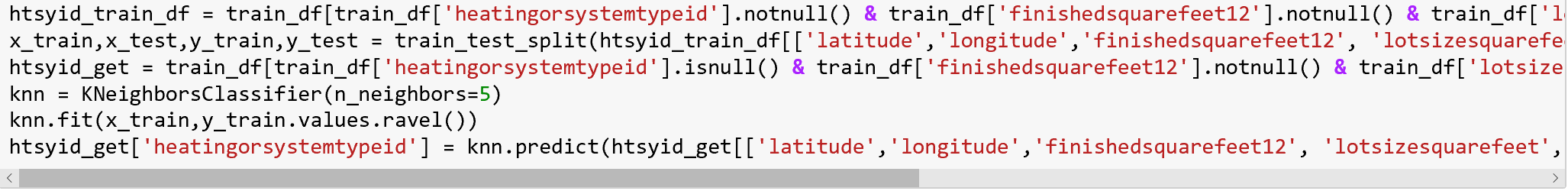
12) Exploring the co-relation between the features with the help of a dendogram.



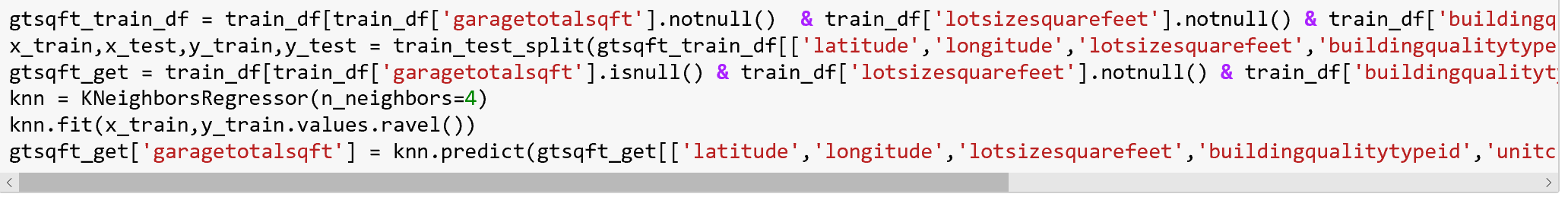
13) Plotting the heatmap of the co-relation between the features to get a better idea of how tightly are they co-related with each other



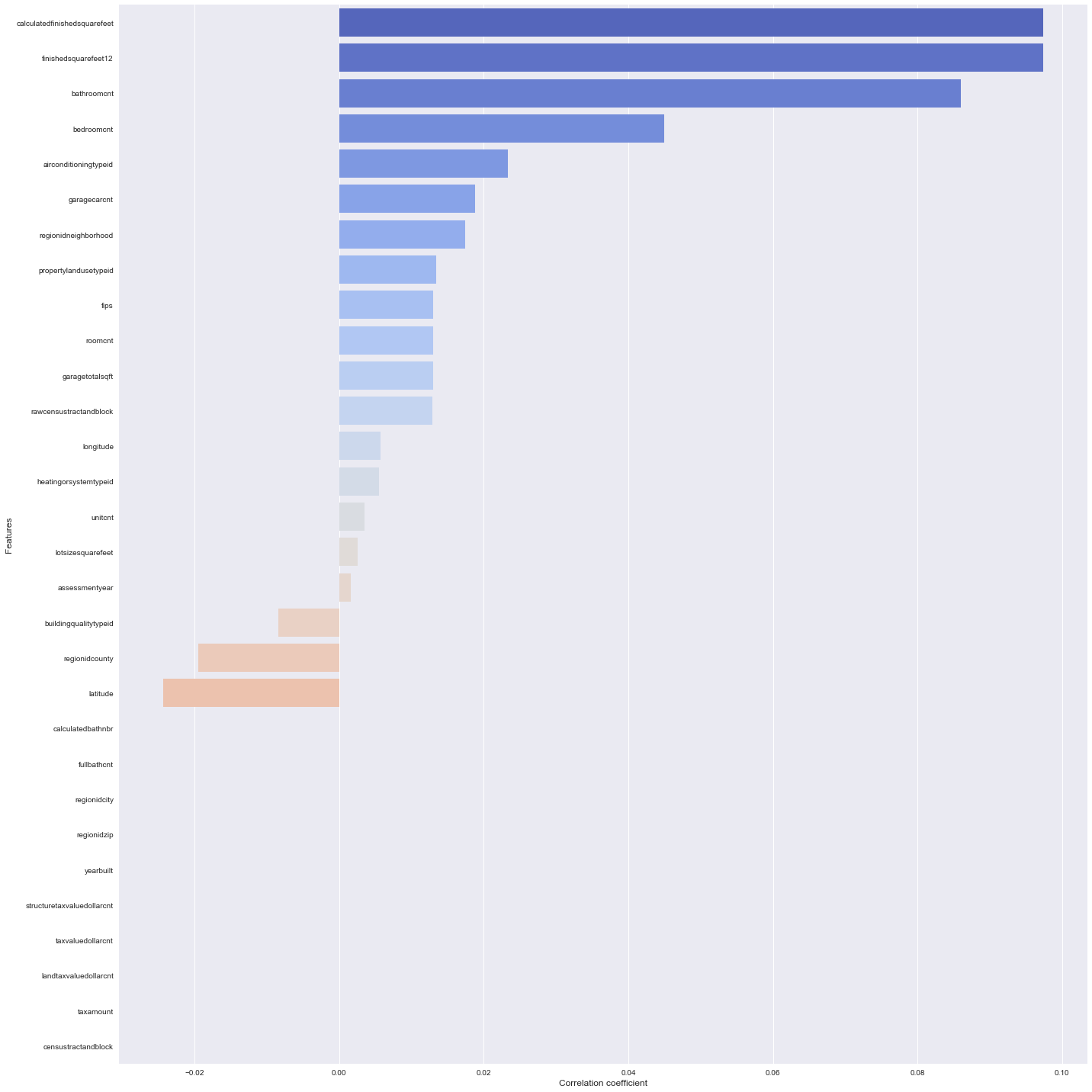
14) Using KNN classifier to fill the missing values of Categorical features



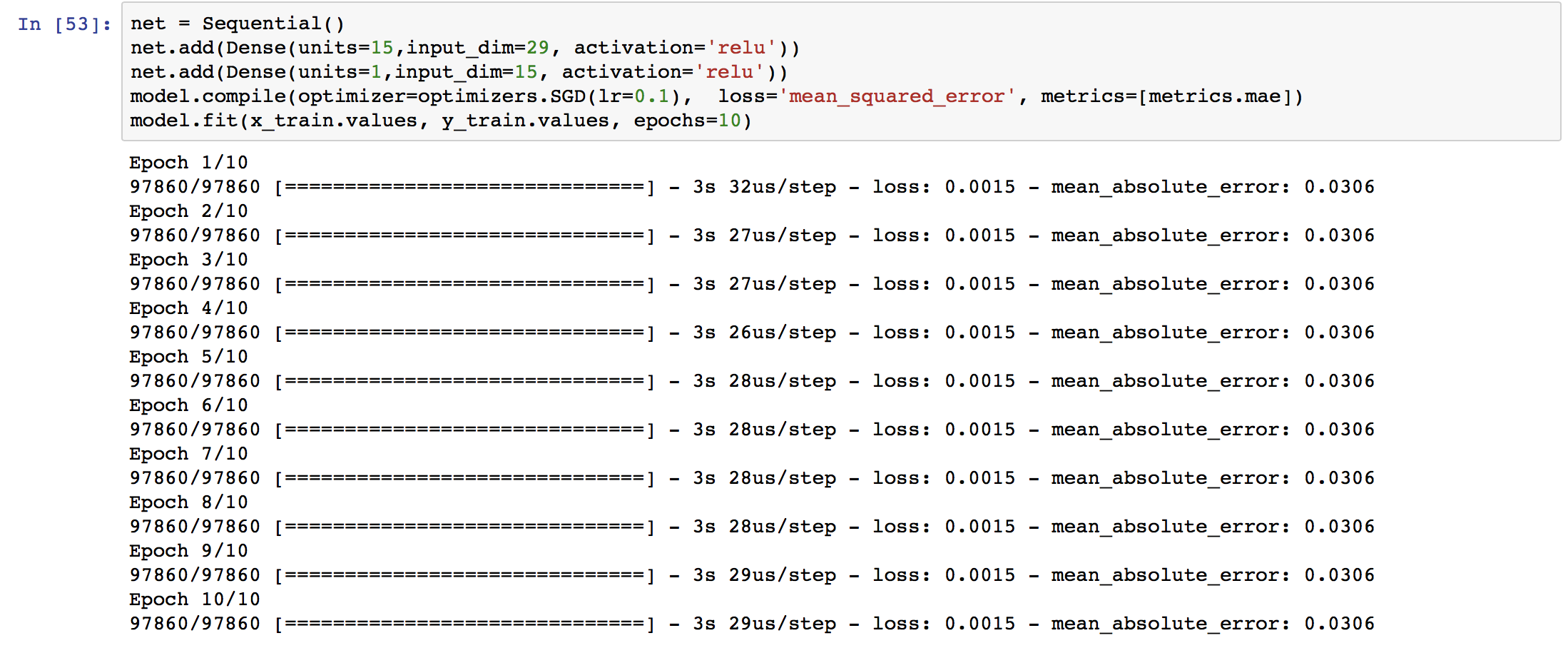
15) Using KNN regressor to fill the missing values of Continuous Features



16) Select the most relevant features (Feature Engineering)



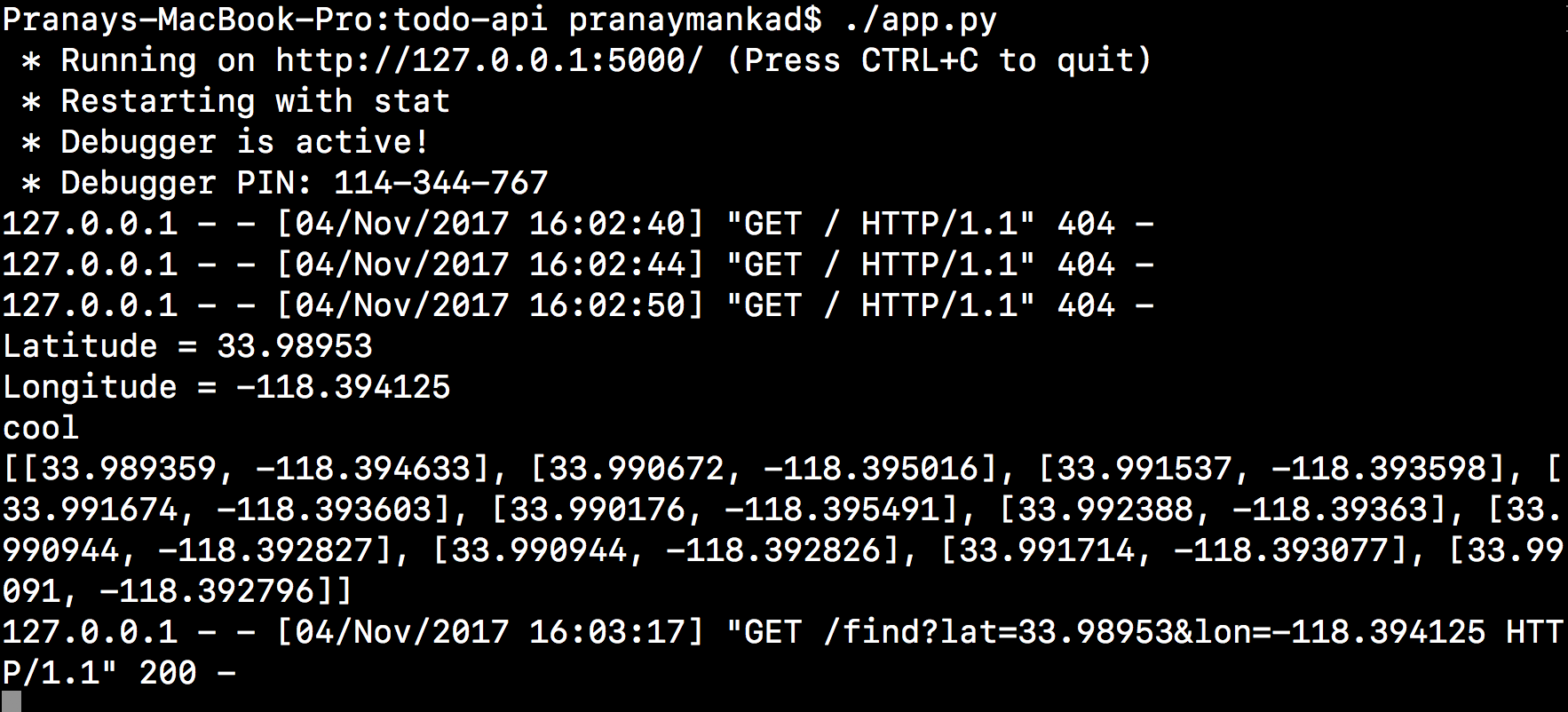
18)



## Part 4: Creating a Rest API, getting 10 closest points of interest

We created a Flask virtual environment and hosted our application on a local instance.

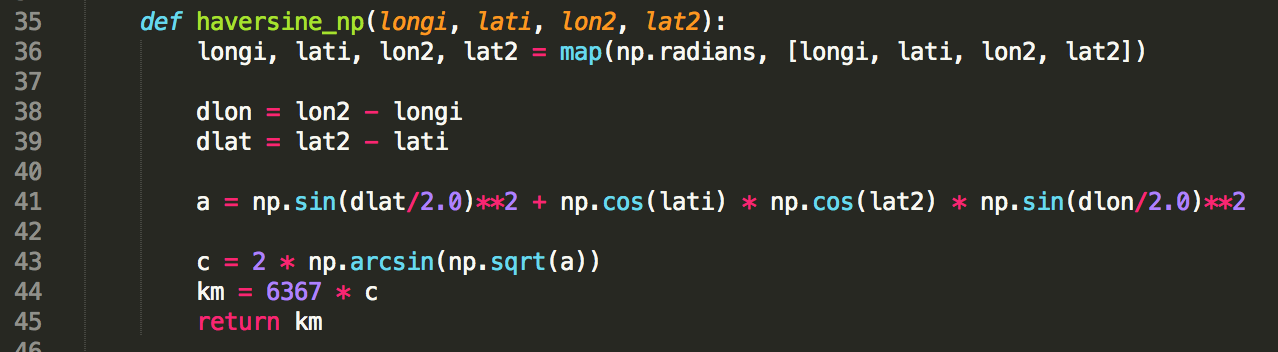
* Flask has a lightweight and modular design.
* Accepts command line arguments.
* Throws back JSON.
* It is Unicode based.
* Supports a RESTful architecture.
* Can use it with any ORM.
* Server is hosted at http://127.0.0.1:5000 and parameters are passed as http://127.0.0.1:5000/find?lat=abcdefg&lon= abcdefg
* Flask creates a virtual environment within which all the libraries are installed.
* Restful service can be accessed through CURL as well.



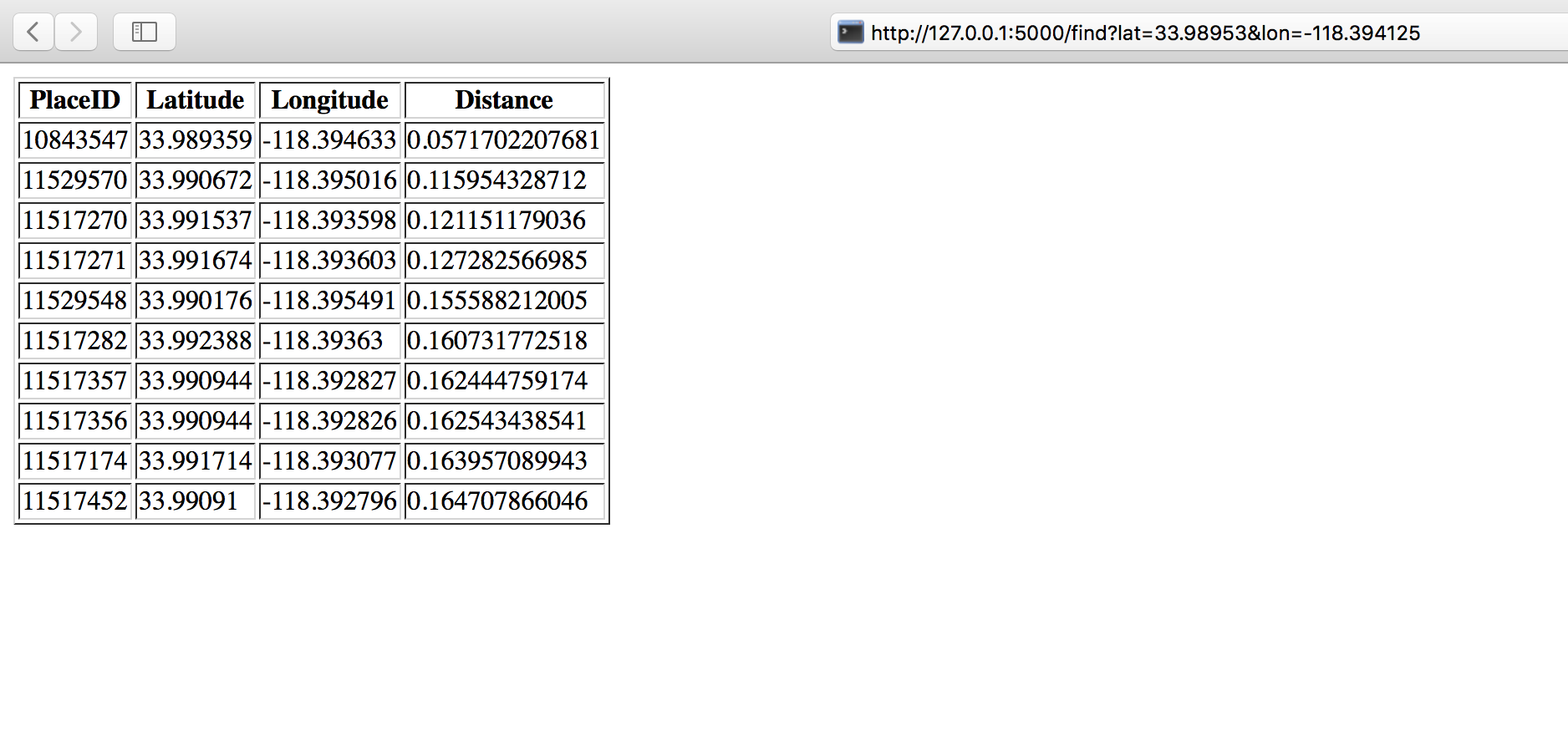
* Used a library JSON2HTML (<http://json2html.varunmalhotra.xyz)> to translate RAW JSON data pictured above to an HTML table as in the following screenshot.
* Process of getting an output is as below:

1. Combine 2016, 2017 Properties as CSV,
2. Form a dictionary and fetch latitude/ longitude from arguments
3. Used a Haversine function to calculate distance from points for all points in the dictionary
4. Sorted points are stored in a list which gives 10 closest properties to POI.
5. A new dictionary is formed and dumped into a JSON data format.
6. Using JSON2HTML, a table is rendered on the web address.

* Haversine function used:



* Test using http://127.0.0.1:5000/find?lat=33.98953&lon=-118.394125



* We are plotting the points on a map using Folium. But since we did not create an entire flask application, we used a Jupyter Notebook to demonstrate the same. Screenshot for the output is below.

