**MIDTERM REPORT**

**Advance in Data Sciences and Architecture**

**INFO 7390**

PROFESSOR:

**SRIKANTH KRISHNAMURTHY**

TEAM MEMBERS (Team 3):

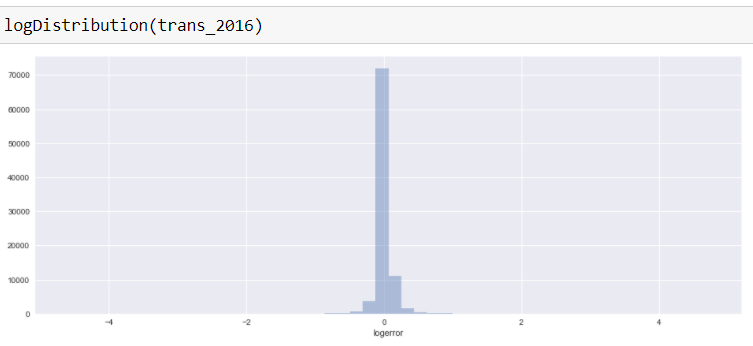
Sonali Chaudhari

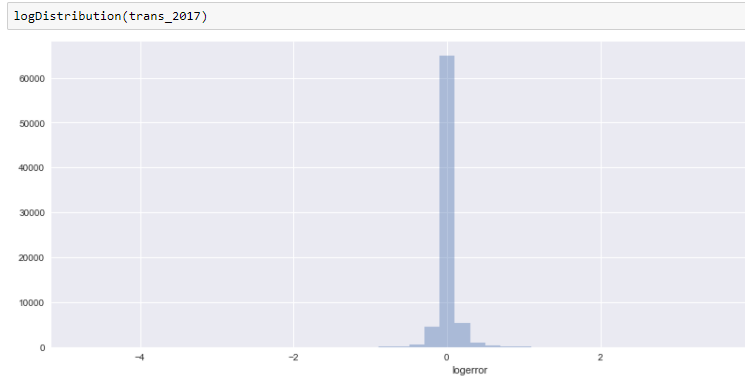
**Part 1:**

EDA

We first looked at different analysis of the EDA: For most of the EDA fraph we used the seaborn package because it had a lot of graph and plotting features to analyze the data properly.

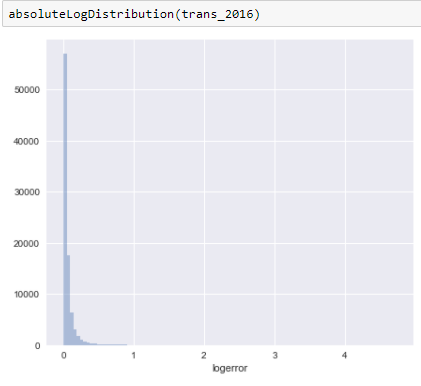
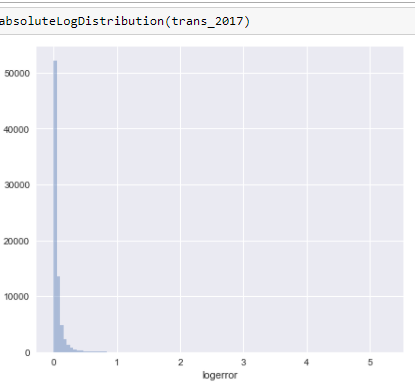
**We analyzed 2016 and 2017 data separately**. This is because we did not want the 2016 data to skew the 2017 data and vice versa. Furthermore, even for missing data, we dealt with the files separately as the amount of missing data varies, hence the way to deal with it might slightly change

1. We first dealt with the training csv that had the Transactiondate, and logError.
2. We separated the transaction date into year and month.
   1. This made it so there would be a column differentiating 2017 and 2016 data.
3. We looked at these Transaction Columns
   1. Log error = Log(Zestimate)-Log(Salesprice)



Above is the distribution of log error

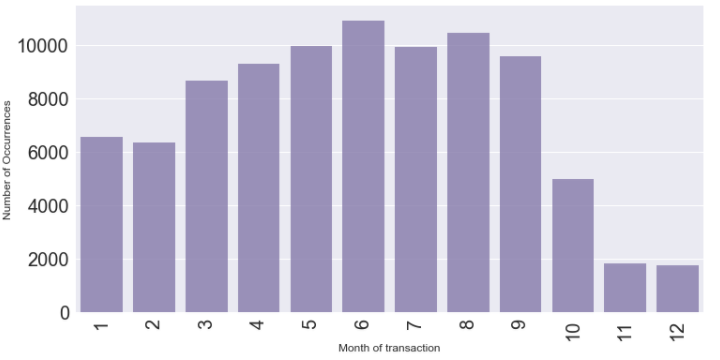
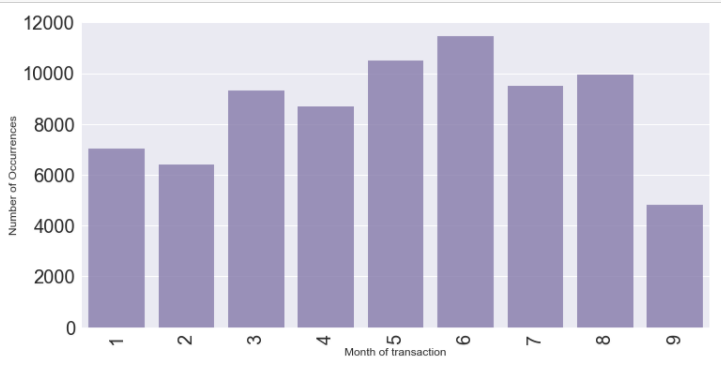
* 1. We also looked at distribution of the absolute log error



From both these graphs it can be see that the log error is in the middle/most values are in certain area.

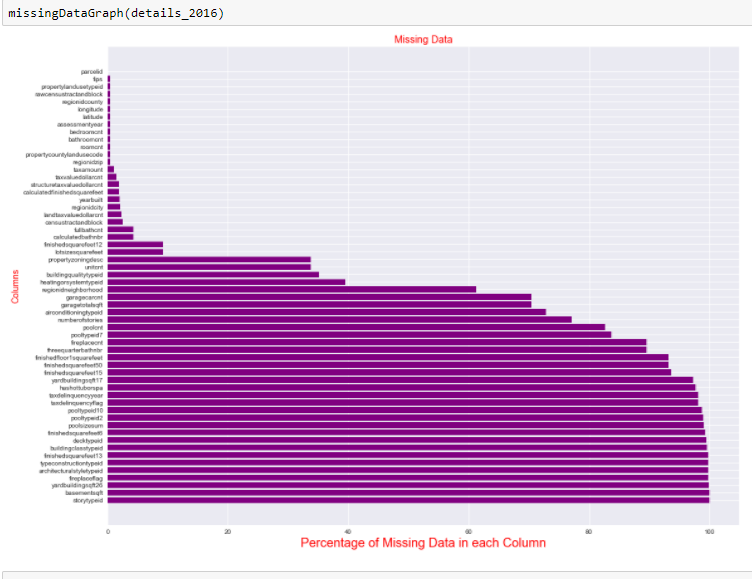
* 1. To see how the transaction changed over the 12 months and which month had the most transactions: we plotted a histogram.



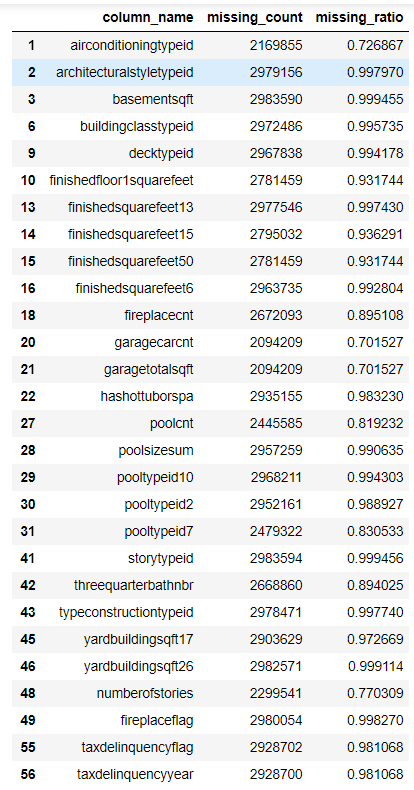


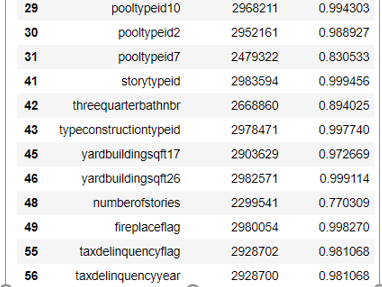
1. **Missing Data**: we now looked at all the columns and decided to look at a 70/30 split. So that means only data that has 30% missing data. This way we are not skewing the data too much and in the future we might ruin our plot

*Below is the graph of all the data and the amount of missing data each column has:*



We weeded out data by only taking on 30% missing and below is the list of columns with more than 30 percent missing.





* 1. We wanted to explore the data types for all columns to see what kind of missing data methods we would do. We noticed there was only 1 catagorical column of propertycountylandusecode, which we would deal with by giving it dummy variables.
  2. Also we looked at mean, count, median, and SD for each to get a better understanding of all the columns and to see what kind of data existed.

1. For some of the EDA, we needed to look at the numerical data analysis separate from the categorical, since right now categorical does not have the dummy variable replacements.
2. **Location Features**: Following are the features to consider: Latitude, Longitude, fips, regionZip, regionCity, censustractandblock
   1. By looking at the “fips” column and validating it against the <https://www.census.gov/geo/reference/codes/cou.html>, we were able to do a look up of the unique values in the column with this site (6037., 6059., 6111)
      1. The three county we got was:

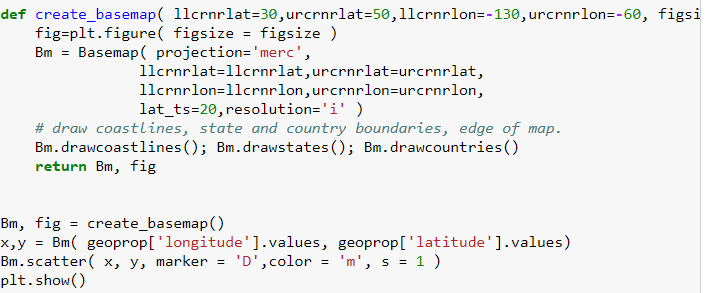
For 2016 and 2017 Data

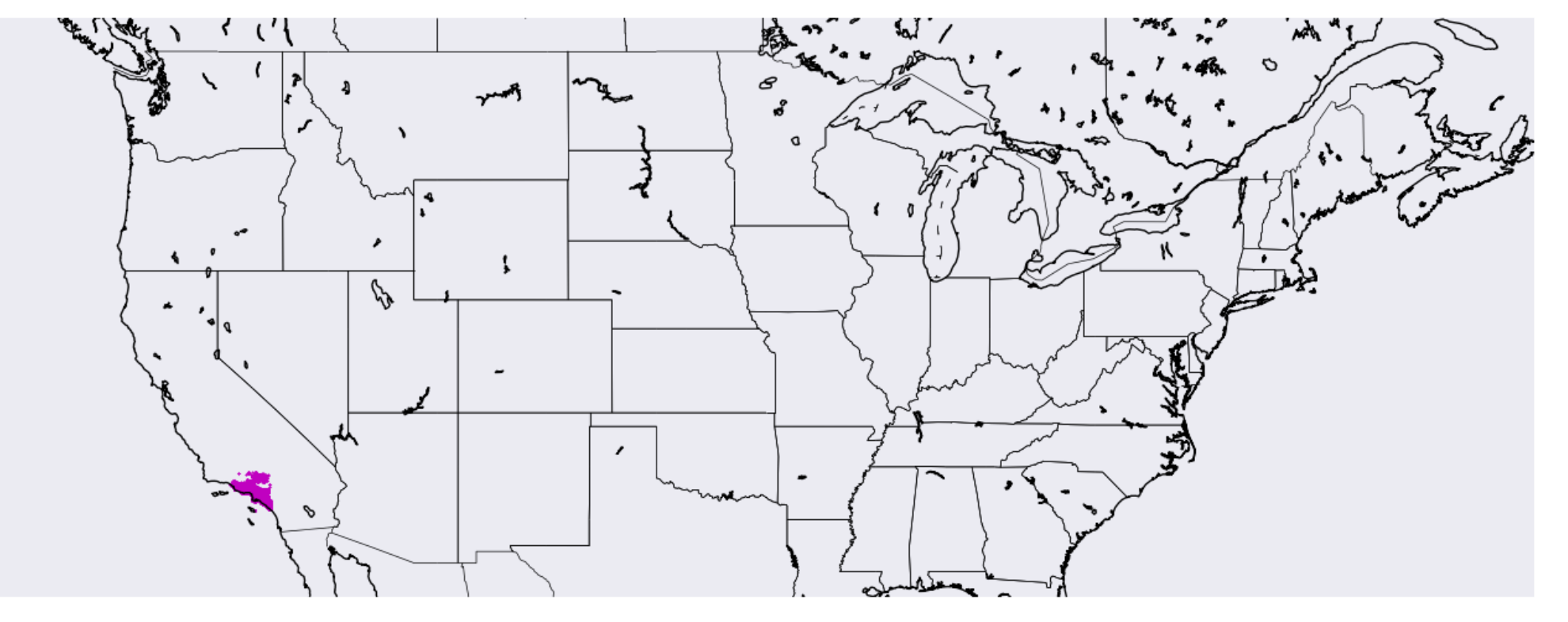
6037 Los Angeles County California

6059 Orange County California

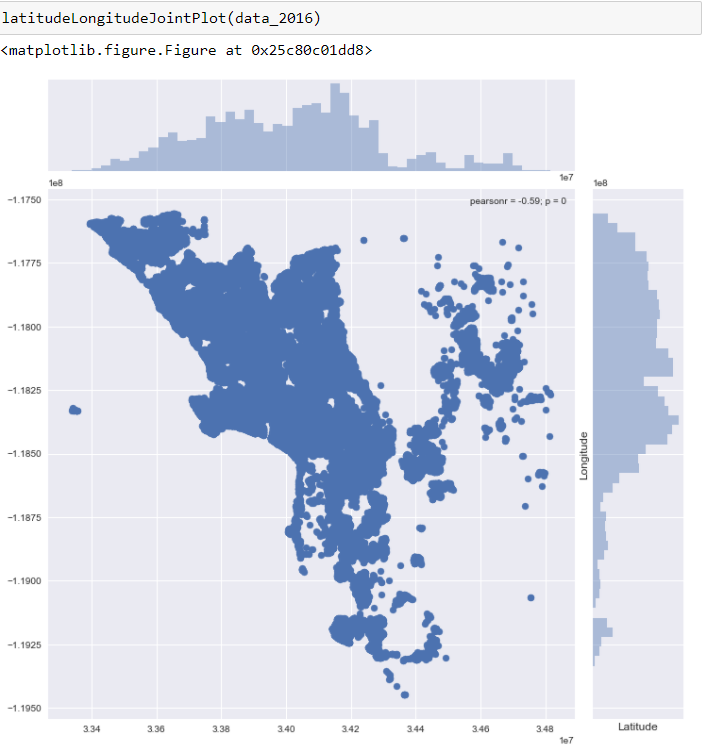
6111 Ventura County California

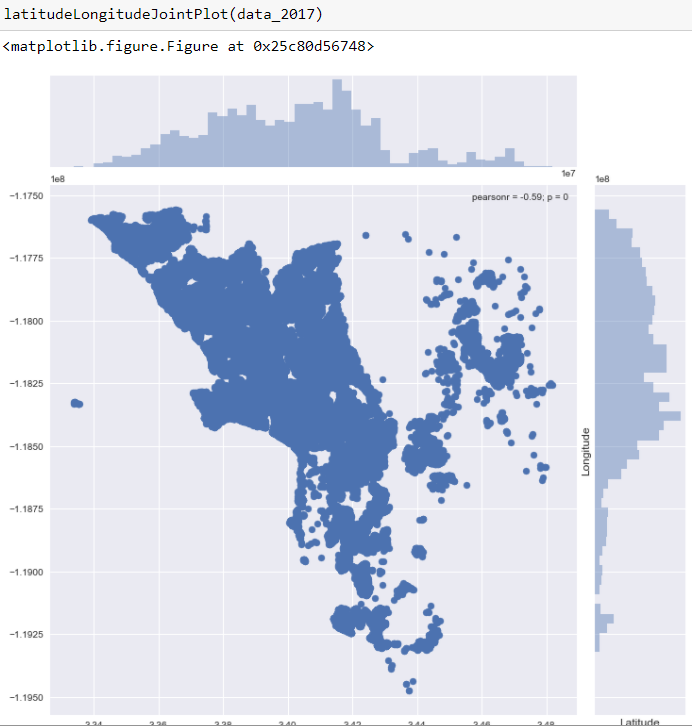
* 1. We the used base map with longitude to do a map plot and to see the locations on a united states map where all the Zillow data housing is from.



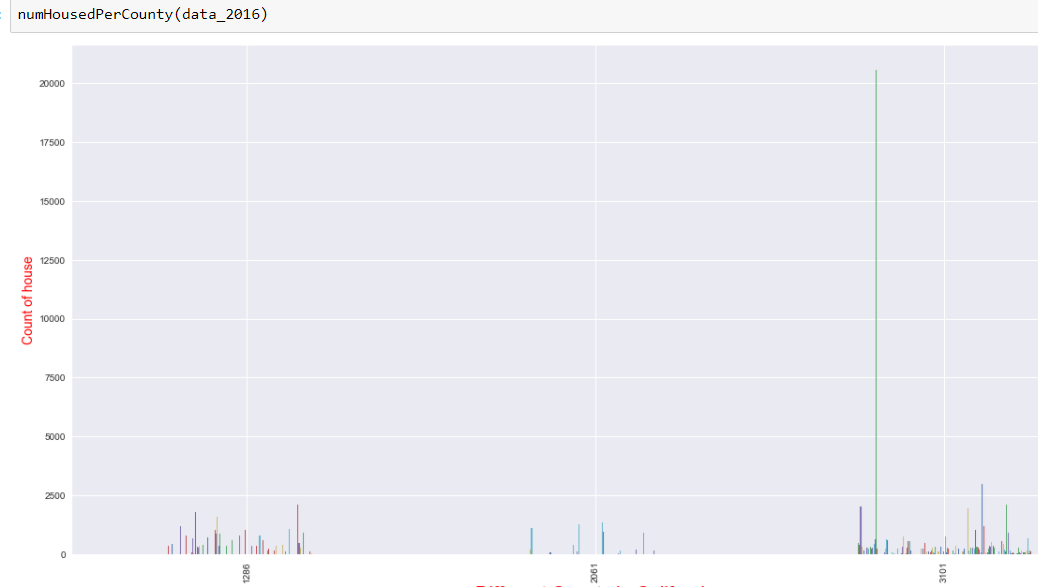


* 1. We also plotted the correlation between longitude and latitude. We used a JointPlot for this since it allows to individually view the data distribution of latitude and longitude and the correlation.

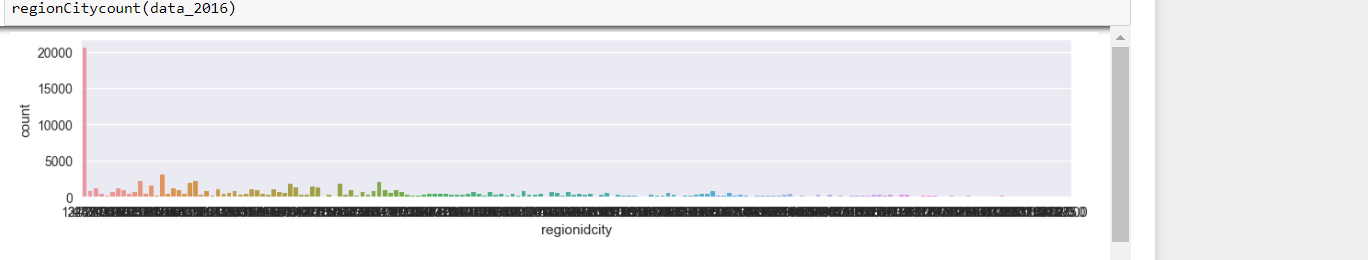




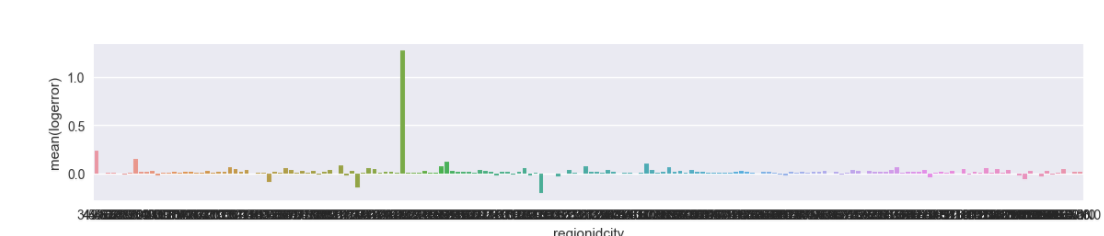
* + 1. It is clear from abow that the distribution atleast for latitude and longitude is same for both 2016 and 2017
  1. When looking more into the regionzip, we noticed that a few zips were not even from the USA and would be invalid data in that column.
     1. For example: "97319" is zipcode for Lithuania, which is in Europe.
  2. To see the distribution of housing across the counties, we plotted a bar graph and grouped by county.



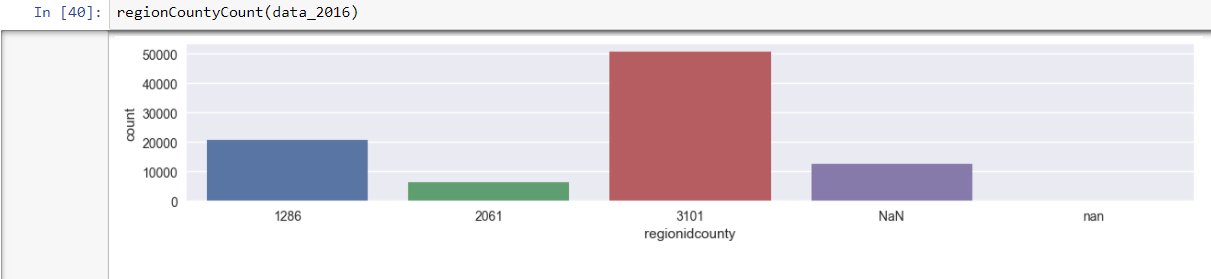
* + 1. Even though we did not also include a screenshot of the 2017 data, it is clear that both years has the same data distribution and it can clearly be seen that the data is skewed in the 3rd region.
  1. We wanted to see how this regionCounty looked against log error so we plotted the graph against logerror and absolute logerror.
     1. Logerror

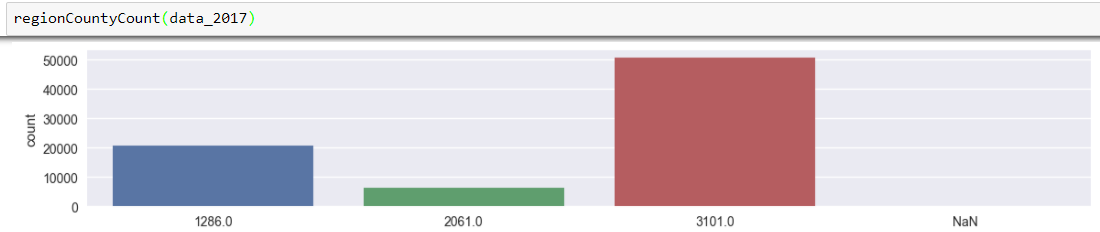


* + 1. Absolute log error

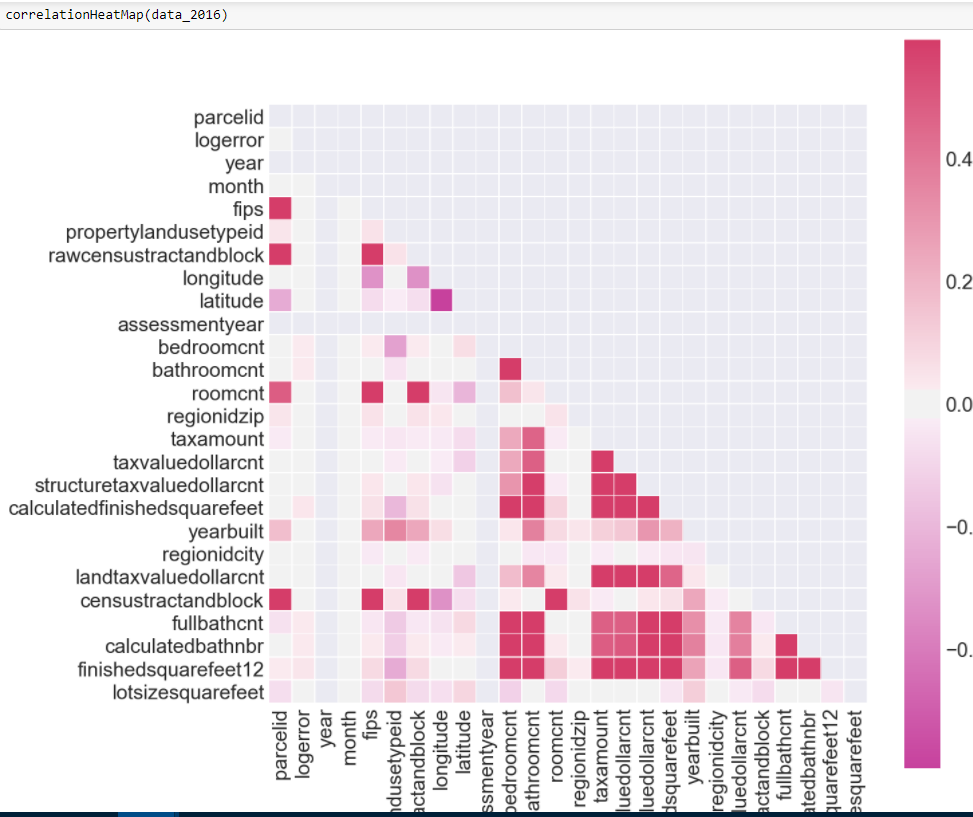
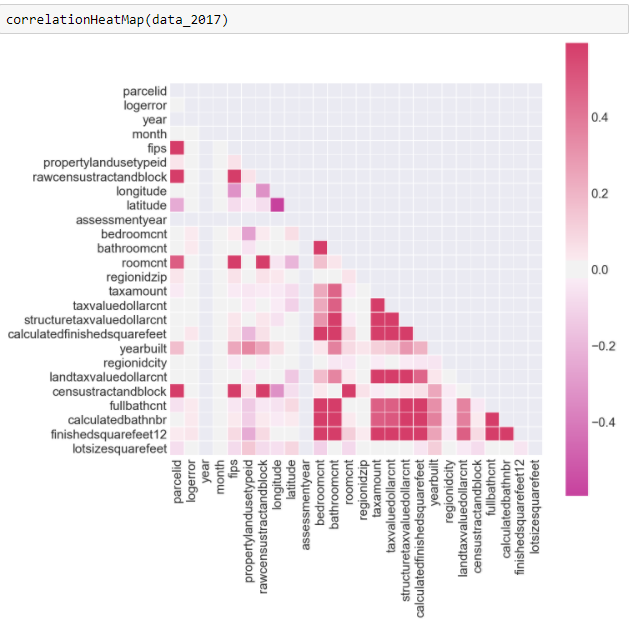


* 1. We also grouped by the region county to view how much of each of the 3 counties are there and how much nan was there.

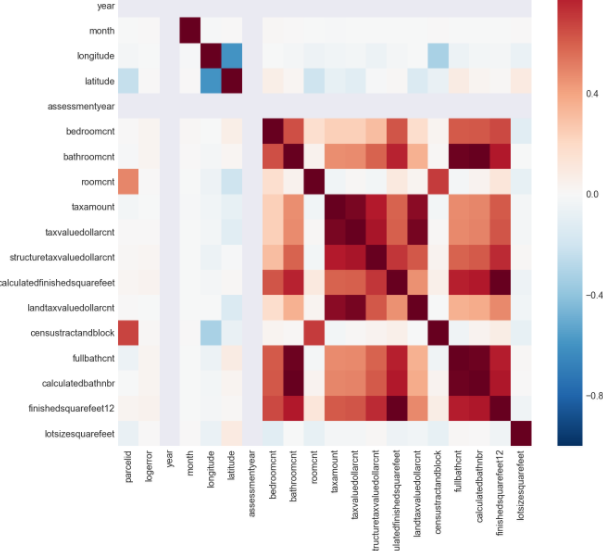


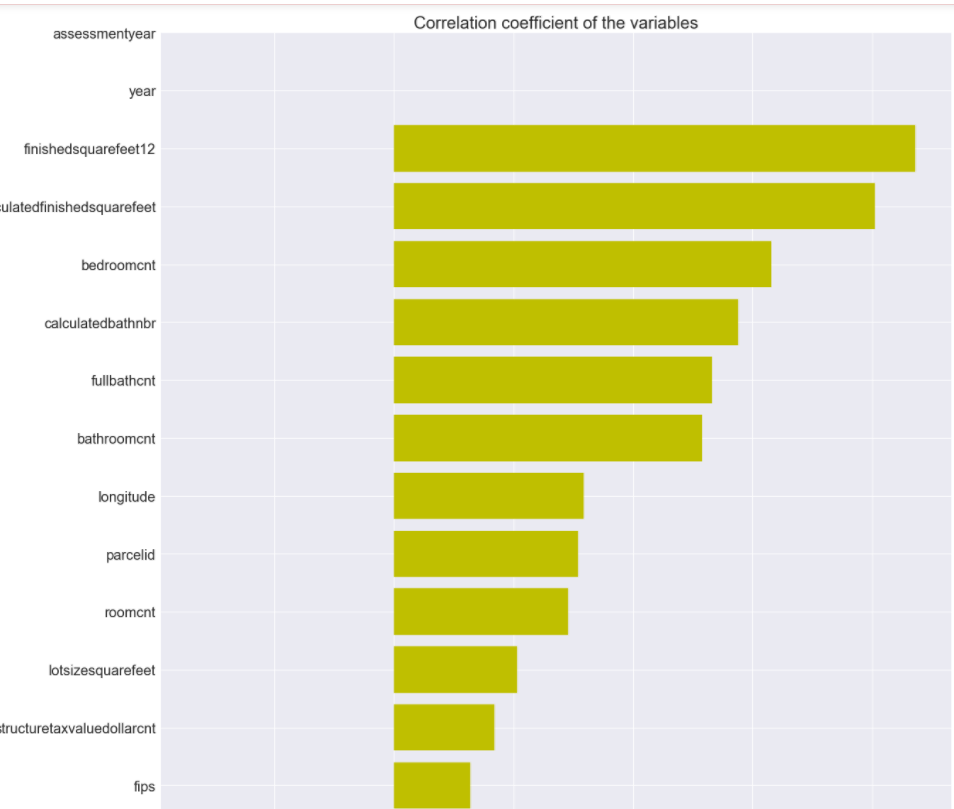


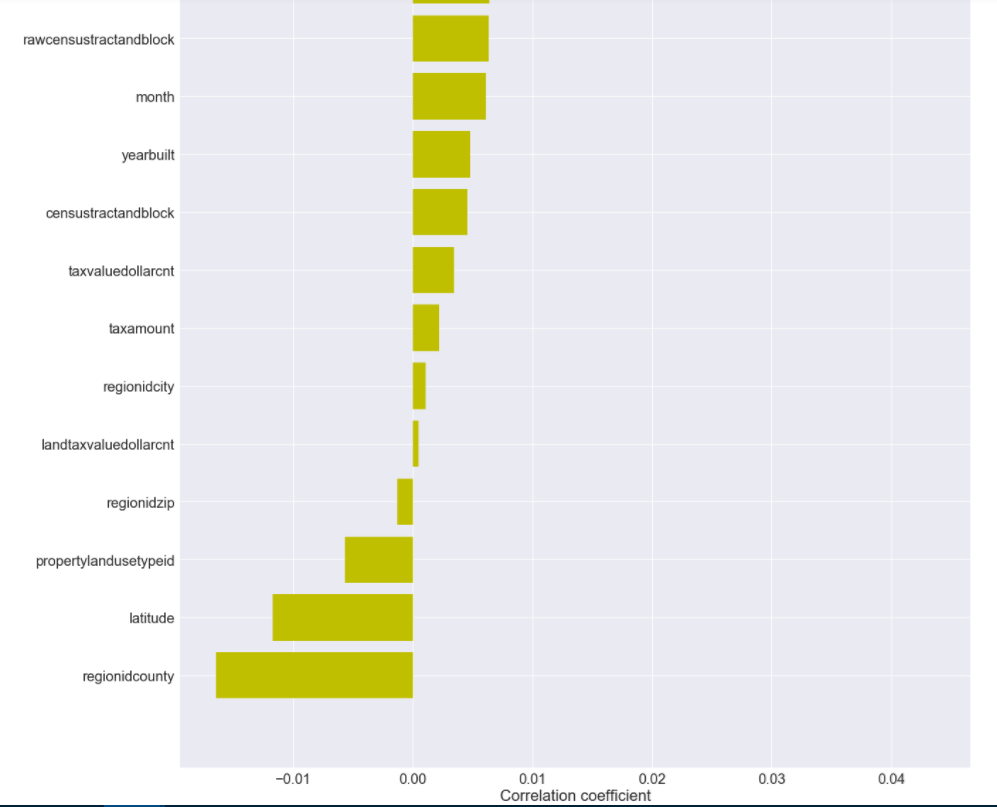
1. **Correlation Graphs:** we used different correlation methods to view the association between the columns as well as the correlation of each to the log error
   1. Heat Map correlation graph



* 1. We also looked at another correlation map, similar to above:



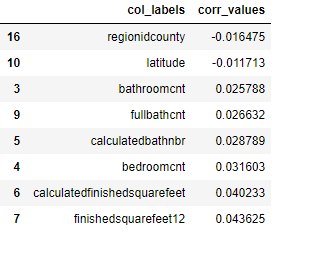
* 1. We looked at correlation between the numerical columns by calculating the correlation with mean value and plotting it.



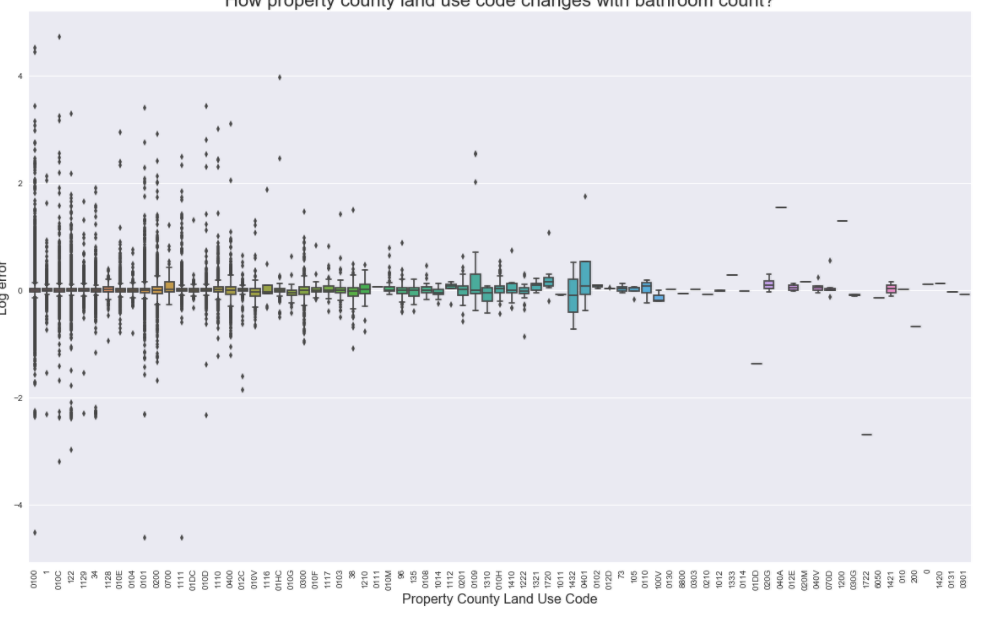
* + 1. We just picked the most correlated columns with the below code



We found that the same columns were highly correlated in 2016 and 2017



* + 1. We then looked at the correlation between each of the highly correlated columns and the log error. We also looked at the categorical data at same time. We grouped each of these correlations by the region county with fips to see the variance in each.
       1. Below is box plot chart to see the range of the values and the median of each value

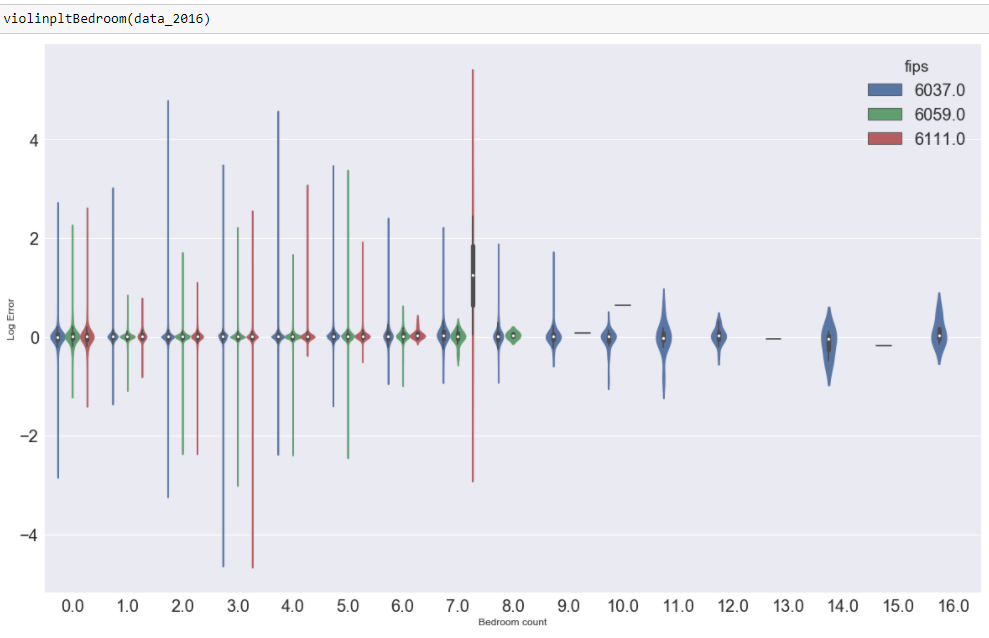


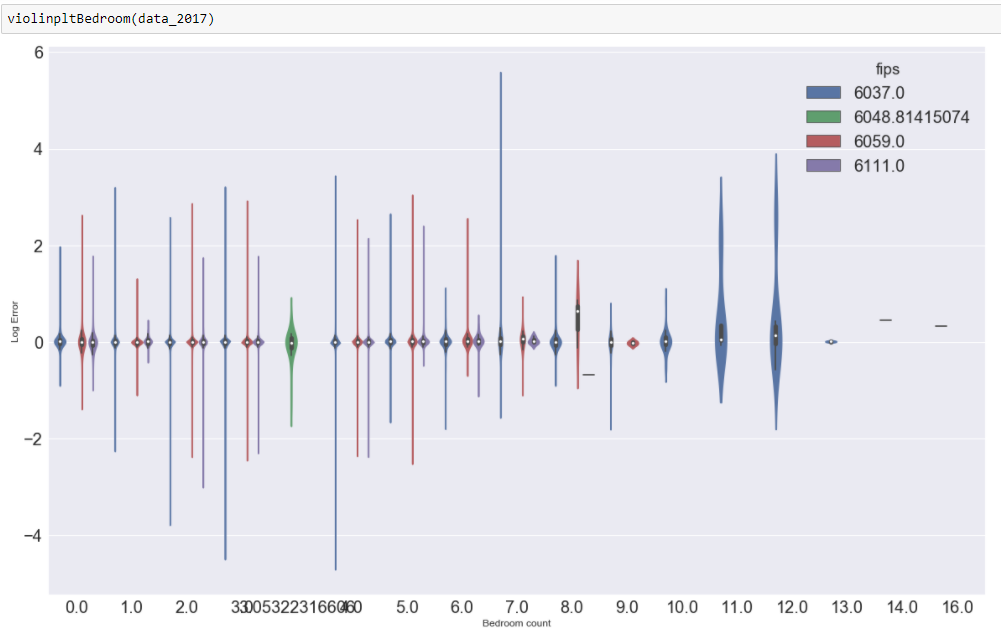
* + - 1. We also graphed a violin chart to view not just what can been seen by a box plot, but also that it shows the quantity of wach value.



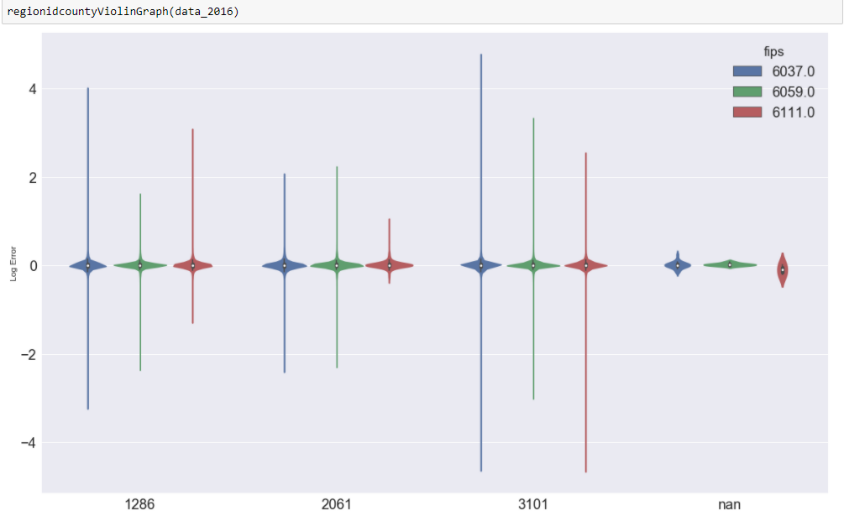
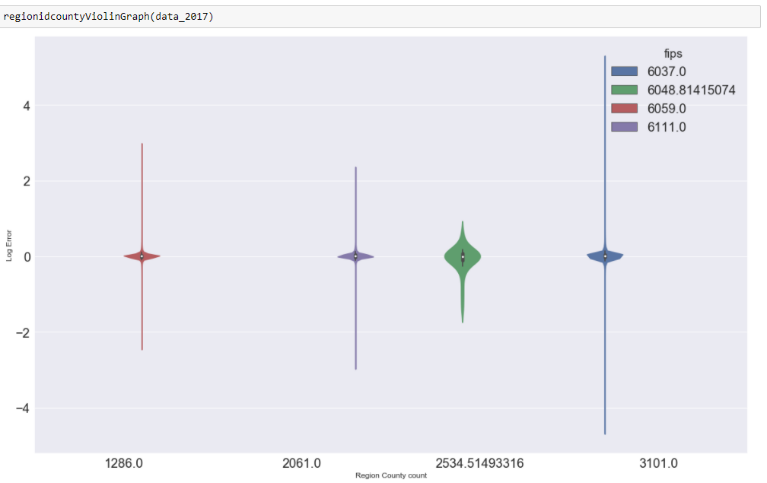


* + - 1. Correlation between log error and full bathroom count

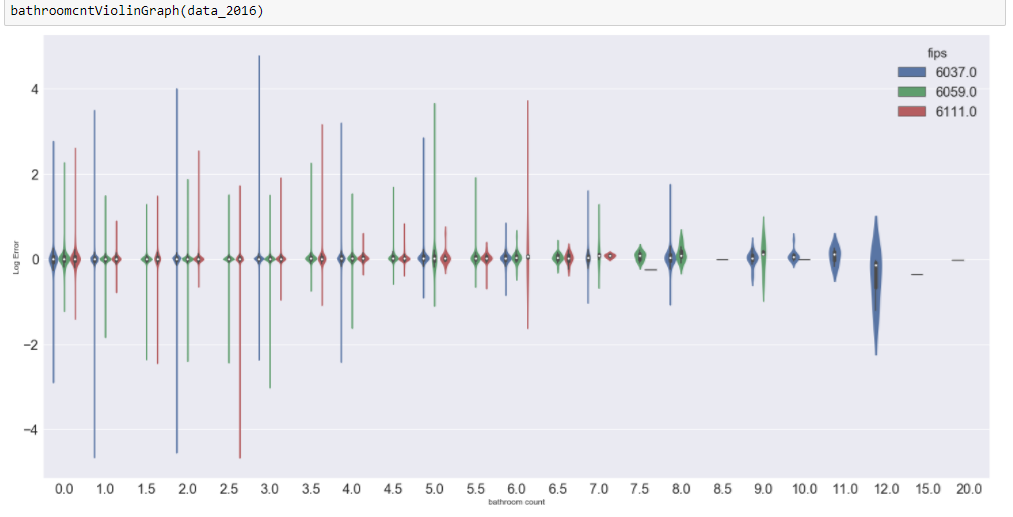


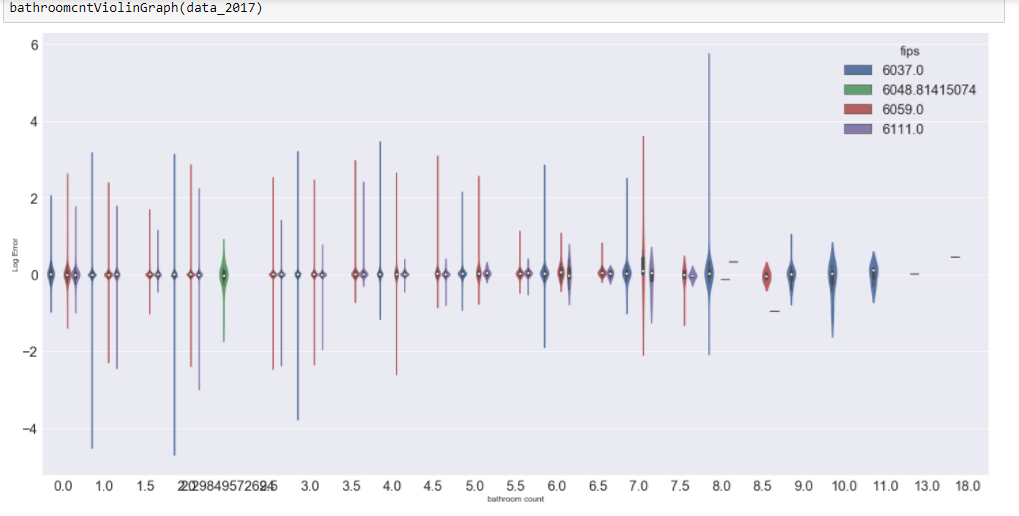


* + - 1. Correlation between region county and log error.

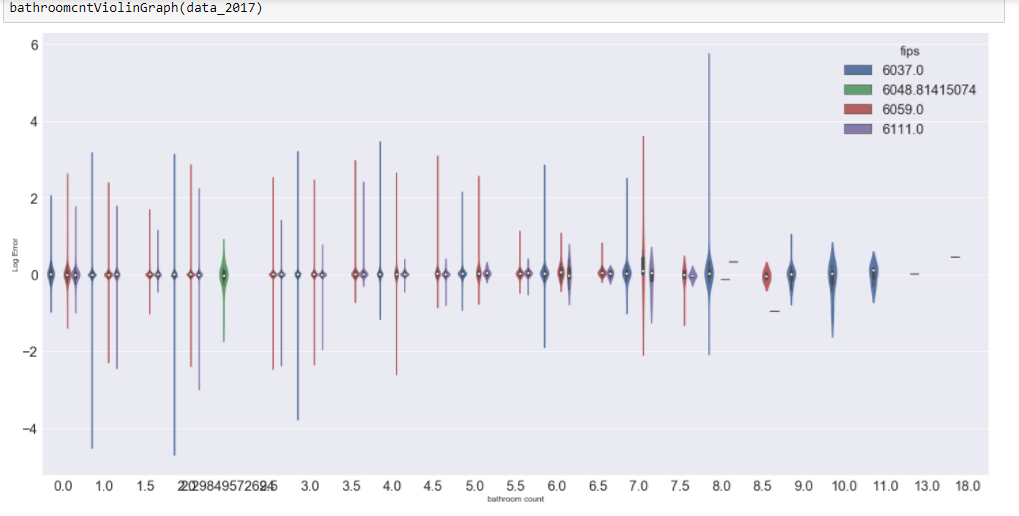
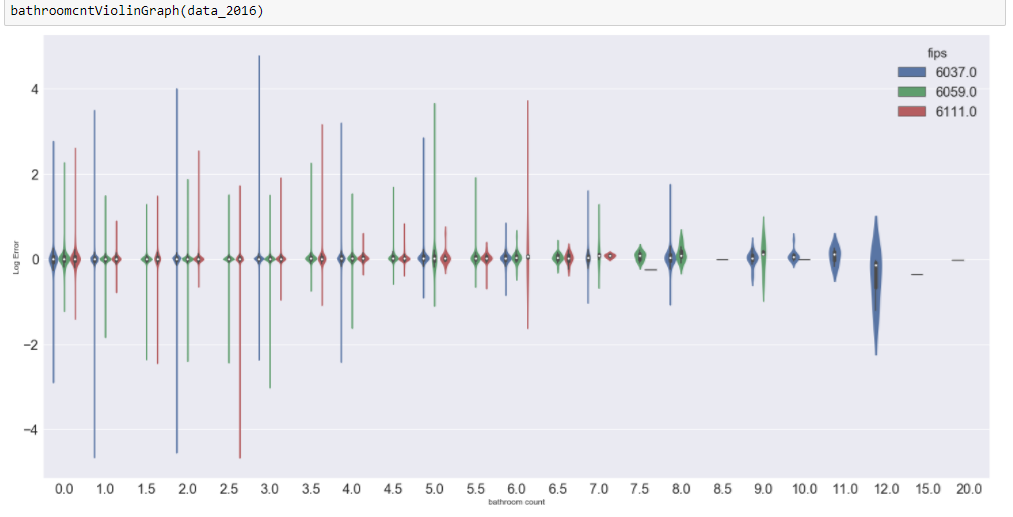


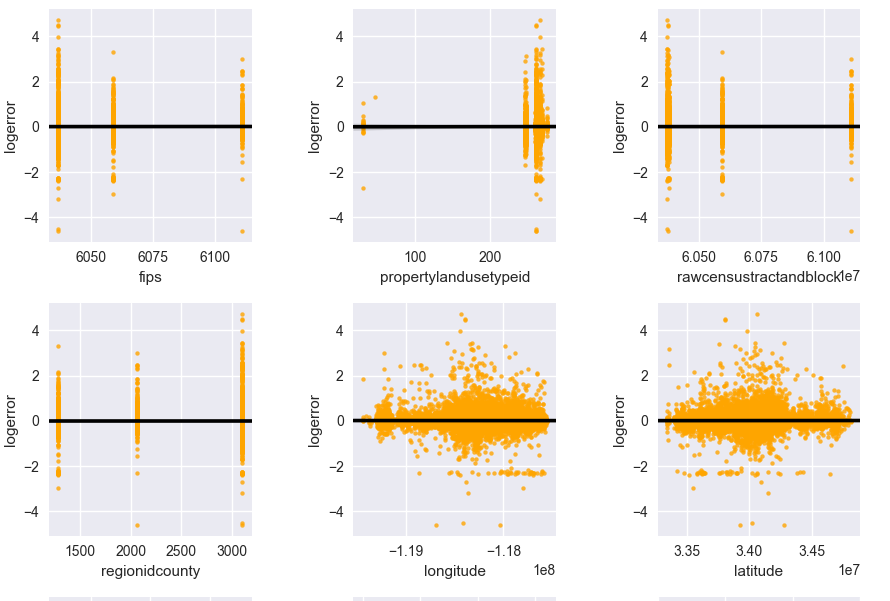
* + - 1. Correlation between log error and bathroom count

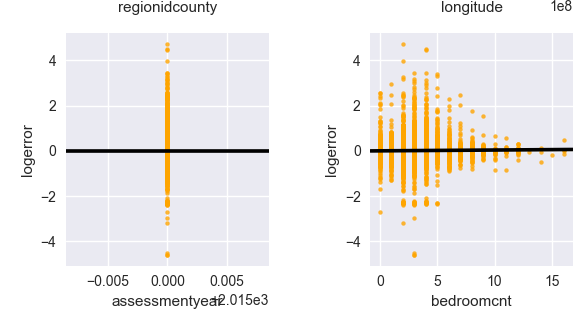




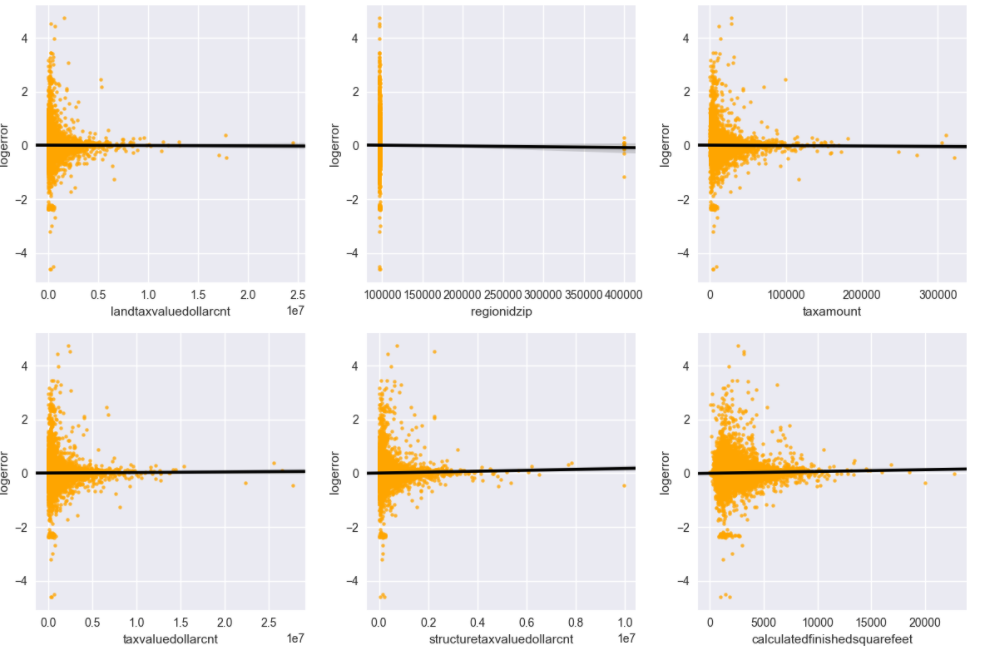
* + - 1. Correlation between full bathroom count and log error. When plotting this, it was clear that it had the same data as bathroom count, hence it is a redundant column.

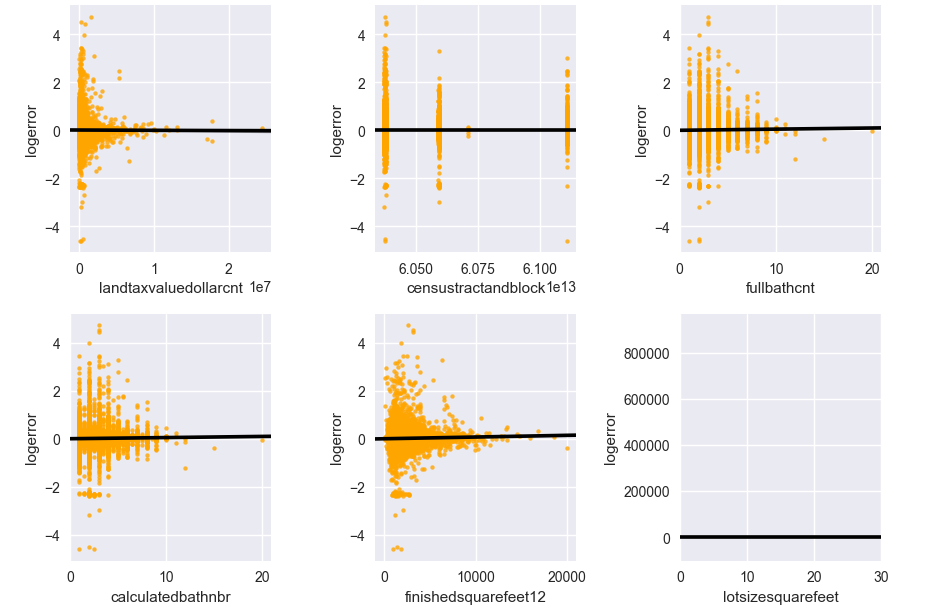


* 1. Next we plotted, with regplot, the correlation graph of all the numerical columns with the log error. It shows the distribution of the values.





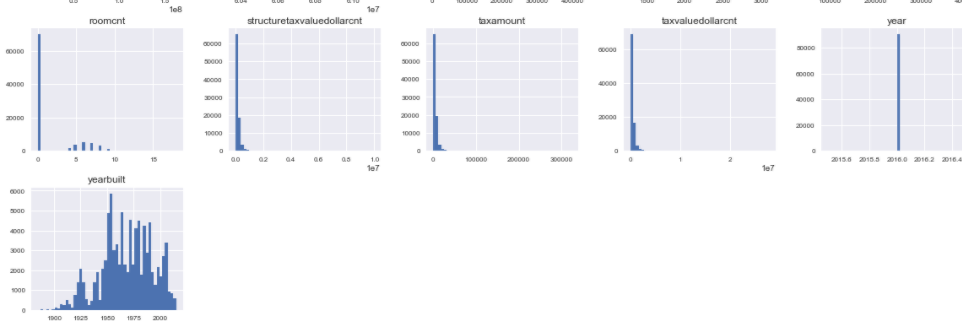




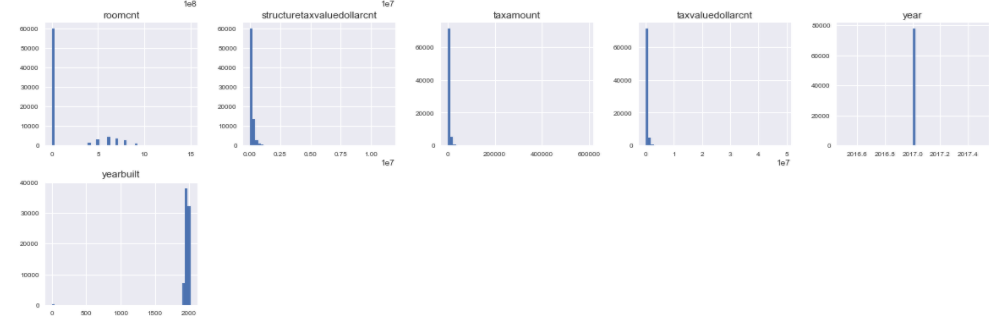
* + 1. The 2017 graph for this was aso plotted though it is clearly seen there is a few differences between the correlation and there is more missing value.

1. We also plotted all joint plots for all the values to view both correlation between log error and the column, and also the distribution of each
2. To view the distribution of all the values, we did a histogram plot. This would be useful to view data before and after dealing with missing data to view if we changed a lot of data.









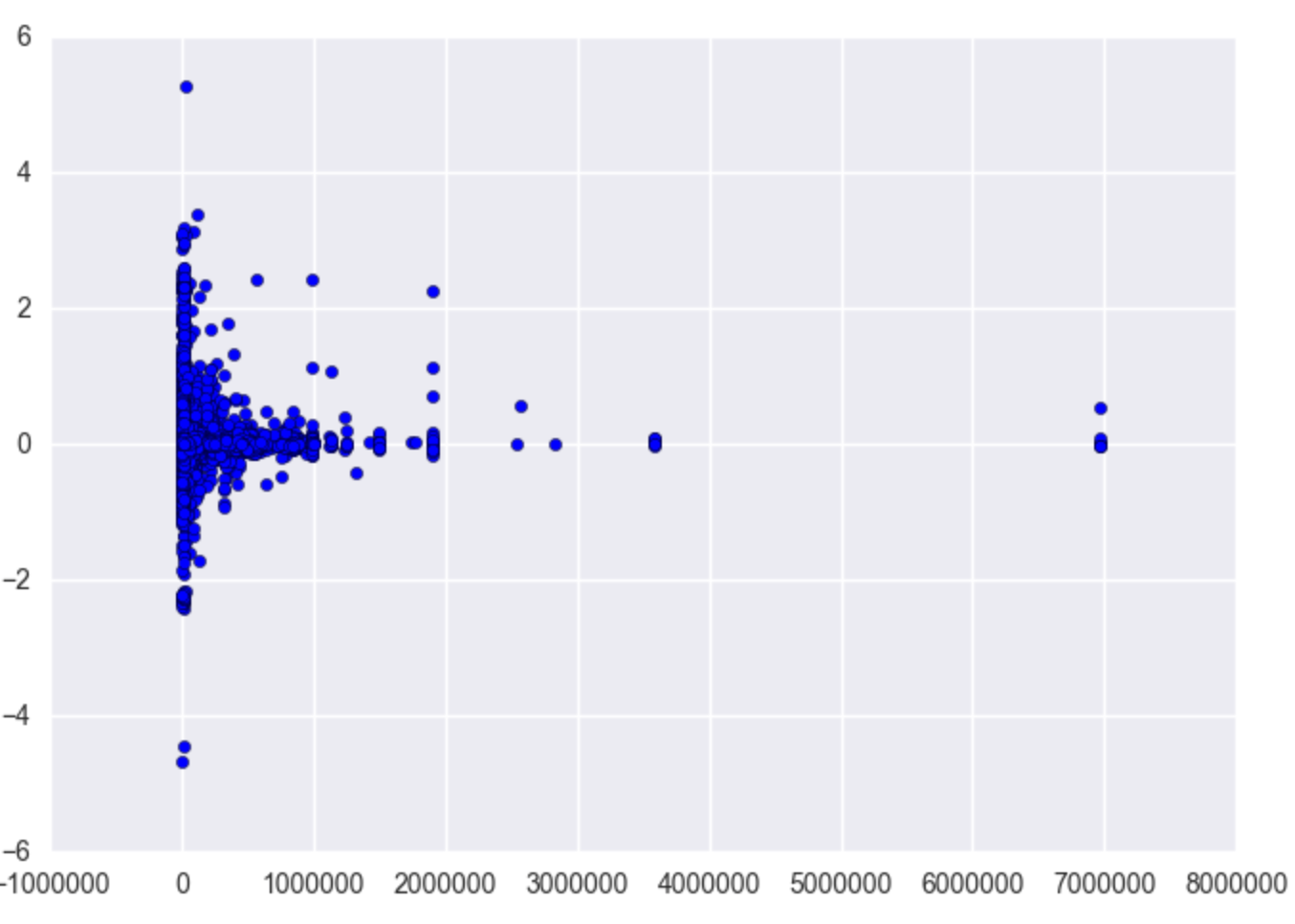
1. We so found and interesting plot for EDA called *panas\_profiling.ProfilingReport*
   1. Displays the report of:
   2. Amount of missing data in each column
   3. count of different data types
   4. Amount of 0s in each column
   5. Description of each (mean, min, max, distinct counts)
   6. The histogram
   7. Common Values
   8. Extreme/outlier values

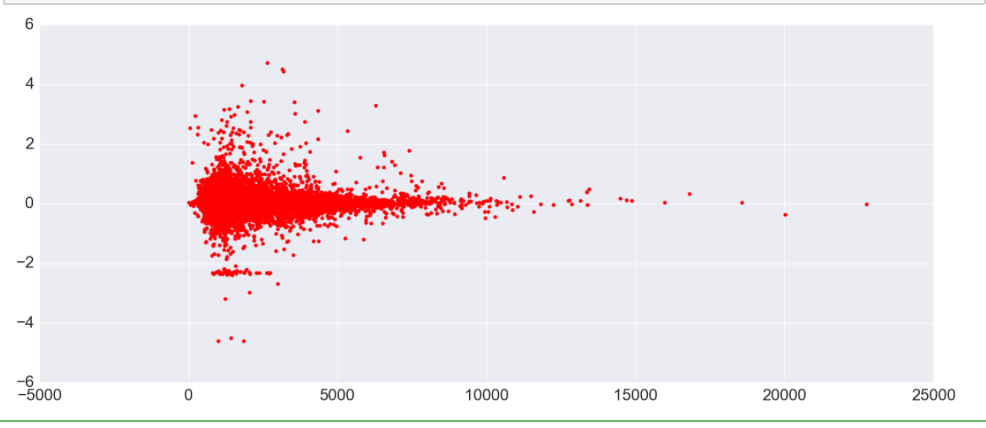
After doing all the EDA analysis we dealt with missing data and once missing data was dealt with, we actually created another .py file that reads in the cleaned 2016 and 2017 csv and does the same EDA on it and we compared the before and after to view the differences.

Following are the imports needed:

1. import csv, sys
2. from pandas import Series, DataFrame
3. import pandas as pd
4. import matplotlib.pyplot as plt
5. import seaborn as sns
6. import numpy as np
7. import pandas\_profiling
8. color = sns.color\_palette()

**Outlier detection**

1. **lotsizesquarefeet , logerror**
2. **calculatedfinishedsquarefe**

**Part 1 : Handling Missing Data**

**Steps**

1. Imported the properties and a=transaction CSVs for both years
2. For both the properties files for 2016 and 2017, we check for percentage of missing data, got rid of columns with more than 70% missing data.
3. For the transaction data frame separate the year a month value and store it in new columns from ‘transactiondate’ column.

trans['transactiondate'] =pd.to\_datetime(trans[‘transactiondate'])

trans['year'] = trans[‘transactiondate'].dt.year

trans['month'] = trans[‘transactiondate’].dt.month

1. Merge the properties and transaction data frame into a single data frame based on the parceled, and left join with respect to transaction data frame.
2. **Redundant Columns :**There few redundant columns which have the same meaning and data entered but different 5 missing data.
   1. rawcensustractandblock and censustractandblock are same meaning column-rawcensustractandblockcontains has no missing data but censustractandblock columns contains missising data.

del df[‘censustractandblock’]

* 1. 'bathroomcnt','fullbathcnt','calculatedbathnbr' are same meaning column->bathroomcnt has no missing data but fullbathcnt,calculatedbathnbr columns contains missising data.

del df['fullbathcnt']

del df['calculatedbathnbr']

del df['finishedsquarefeet12']

* 1. ‘calculatedfinishedsquarefeet','finishedsquarefeet12' are same meaning column->calculatedfinishedsquarefeet had less missing data than finishedsquarefeet12 -> deleting the column

1. **Handling rehionidzip:** The Zipcode in USA id is five digit code. But after analysis we found the regionidzipcode contains values have more than 5 digits.
   1. Replace the false zip code with mode of zip code belonging to a particular regioncountyid.
   2. For missing zip code values: Using the ZipcodeSearchEngine library retrieve the closest zip code given the latitude and longitude and fill in the missing data.
2. **Handling ‘propertycountylandusecode’** : Use LabelEncoder()
3. **‘taxvaluedollarcnt’ ,’landtaxvaluedollarcnt’ and ’taxamount’:** We have replaced the NaN values with median of each column.
4. **Handling ‘yearbulit’:** To fill in missing year built we have used KNN. SO we train the model for given latitude and longitude predict the year built. We have assumed that generally the house are built during the same time during a same time period.
5. **Adding ‘age’ column:**  Once we have no NaN in year built we can add new column called age - denoting age of the house by subtracting the the year built from current year i.e 2017
6. **‘calculatedfinishedsquarefeet’ :** Contains outlier for values less than 120. There are values like 2,44,60 which are insignificant for the column meaning. So removing the rows have the column value less than 120
7. **‘structuretaxvaluedollarcnt’:** We have replaced the NaN values with median of each value.
8. **‘lotsizesquarefeet’:** There are outliers for value greater than 2000000 for 2017 data and 3000000 for 2016 data. Replacing the outliers and NaN with the median value.
9. ‘**Regionidcity’ :** Replace the NaN with the maximum occurring cited with a regionidcounty.
10. Concatenating the two data frame after handling missing values into one common data frame, getting rid of false index column and Unnamed columns and writing it to a CSV called Clean\_Combined.csv
11. Zipping the file to a zip folder ziollowdata and uploading it to AWS using AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY.

**Part 2 Build a prediction model**

We built 3 different models namely :

* Multiple linear regression
* Random forests
* Neural networks

General Steps”

1. We calculated rms and mae values for each models and compared all the three them.
2. We split the data in the set of 70-30, 70 - train , 30 - test using cross validation. There is a library hat splits the data.

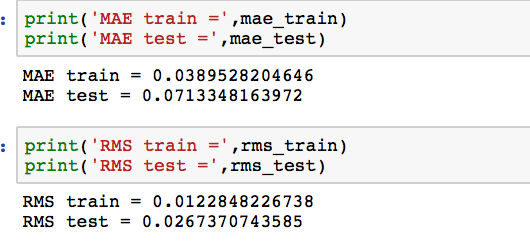
from sklearn.cross\_validation import train\_test\_split

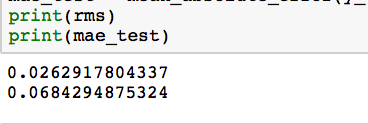
**Random Forest:**

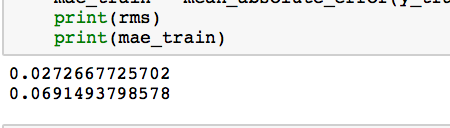
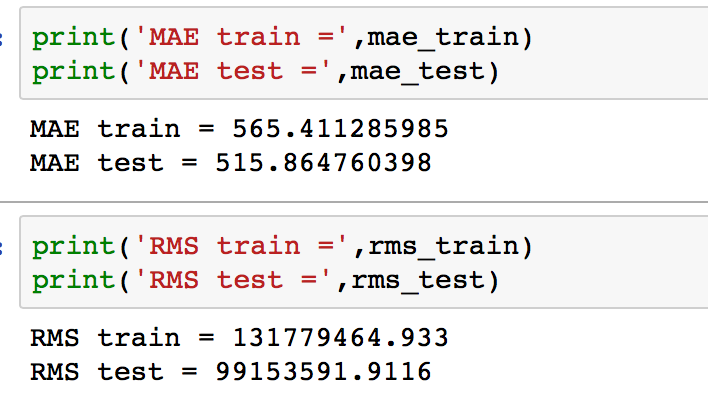
rf = RandomForestRegressor(n\_estimators=300, min\_samples\_leaf=3)

rf.fit(X\_train, y\_train)

y\_train\_predicted = rf.predict(X\_train)

y\_test\_predicted = rf.predict(X\_test)

**Multiple Linear Regression:**

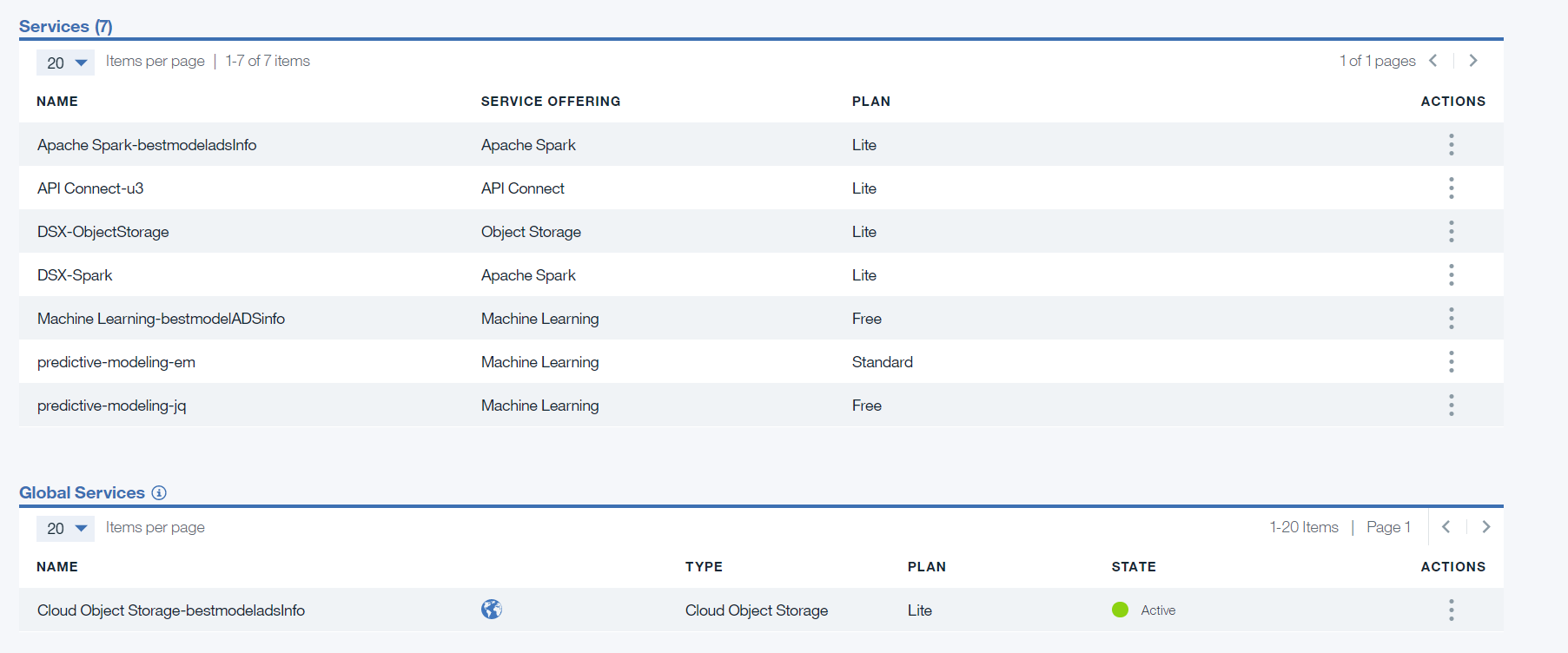
**Neural** **Networks:**

**Part 3:**

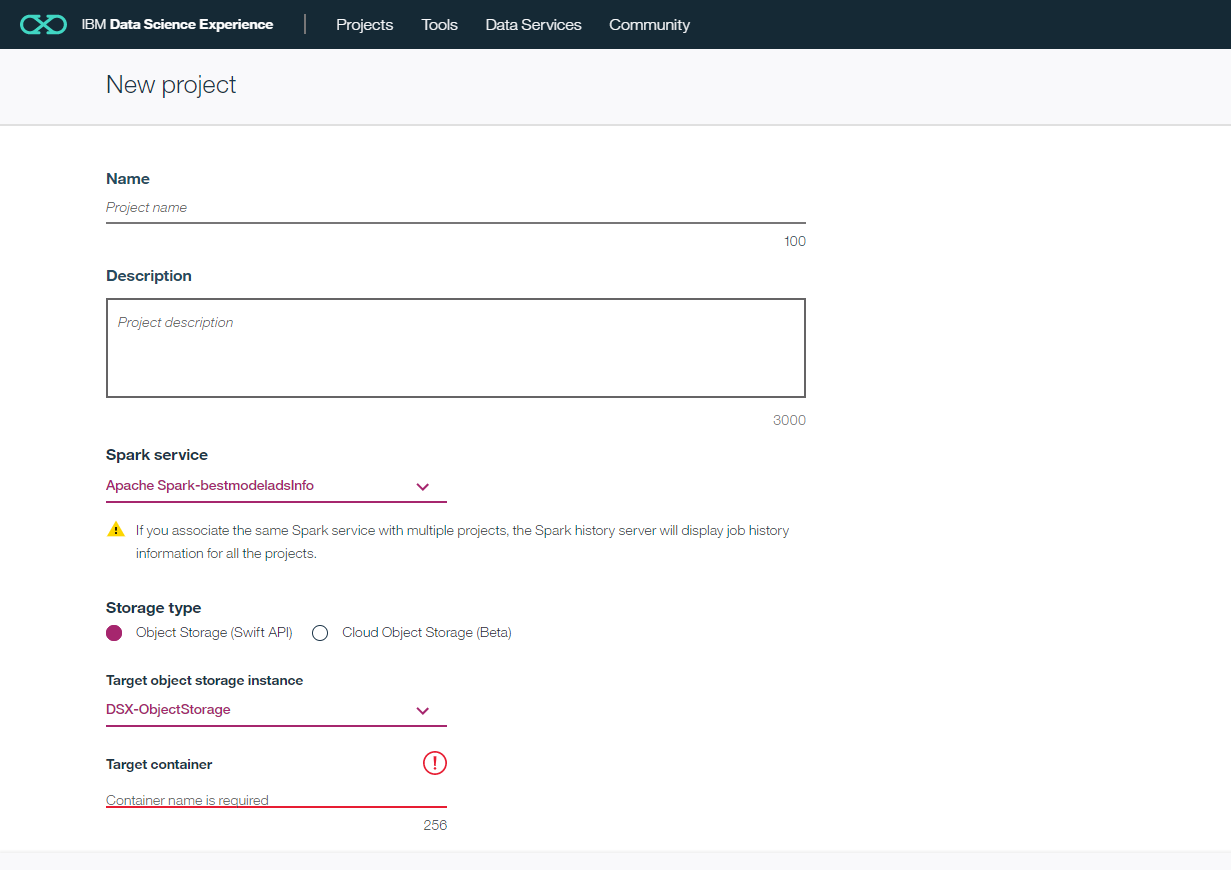
The tool that we had to use was the IBM Data Science Experience to deploy our model and get the API.

1. We first Had to set up the environment in our IBM Cloud <https://datascience.ibm.com/docs/content/analyze-data/ml-setup.html>

And when we did we got

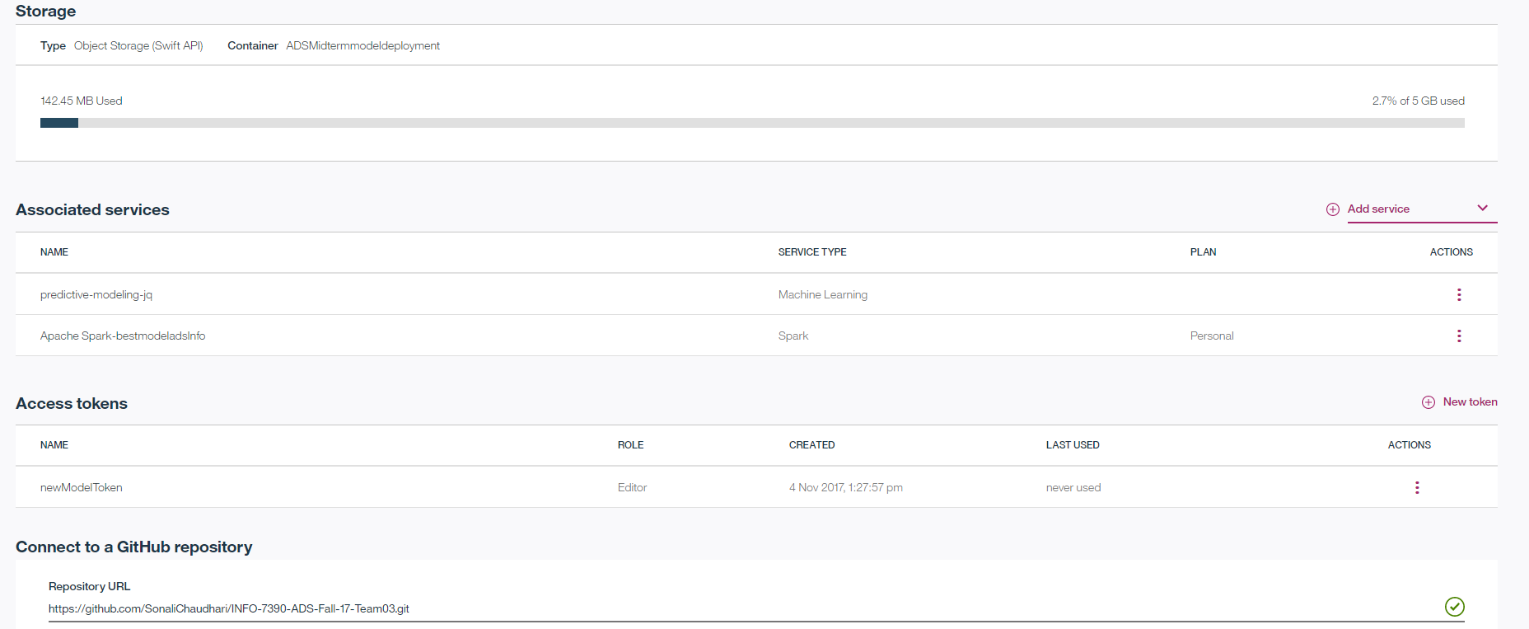


1. Then go to IBM Watson Machine Learning and create a new project

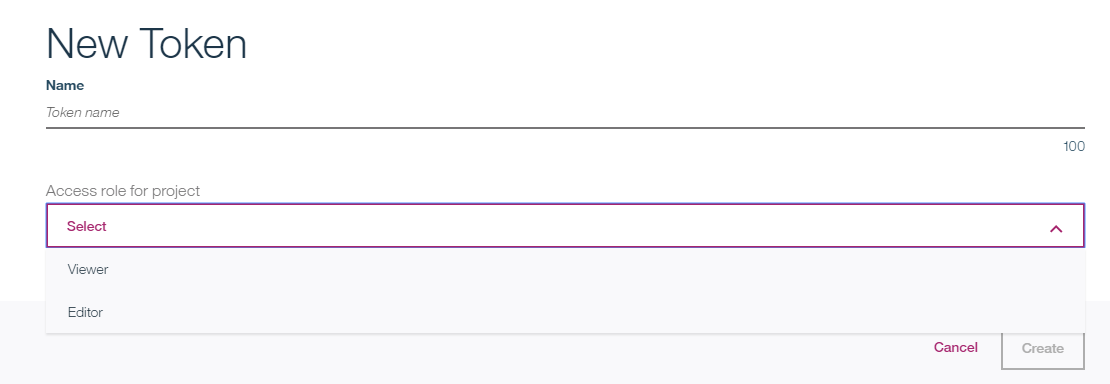


* 1. The spark service will be the one that you have set up for the environment and the target object is the storage that was set up.
  2. The target container will automatically be filled when you give a name for the project
  3. The storage type you can either choose the SWIFT API or the one that is AWS one which is the cloud object storage (in that case no need for target object storage)

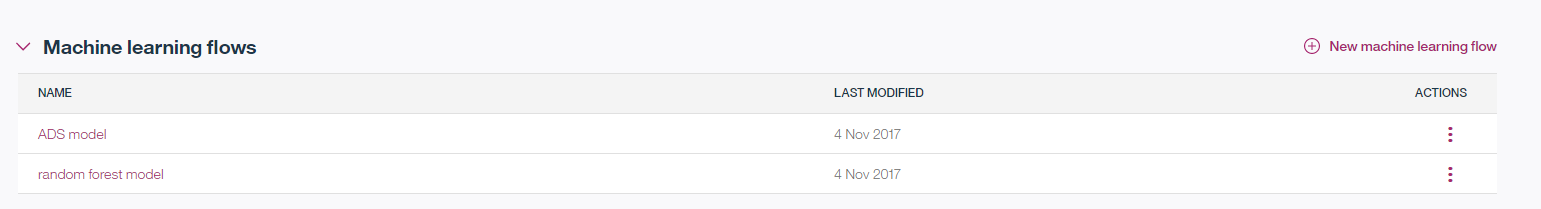
1. Once project is created : you will either be taken to it or go to the dashboard and choose your project. In the settings tab in project to the following .
   1. Create a token and also you can set up a link to your git.



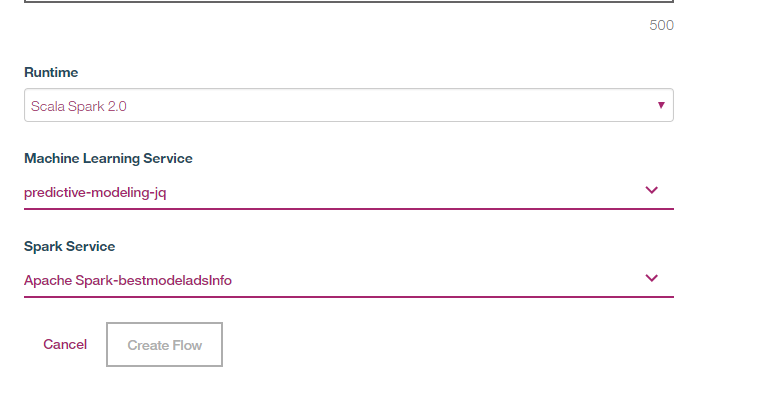
* 1. For access token just click on new token and close viewer or editor and name it.



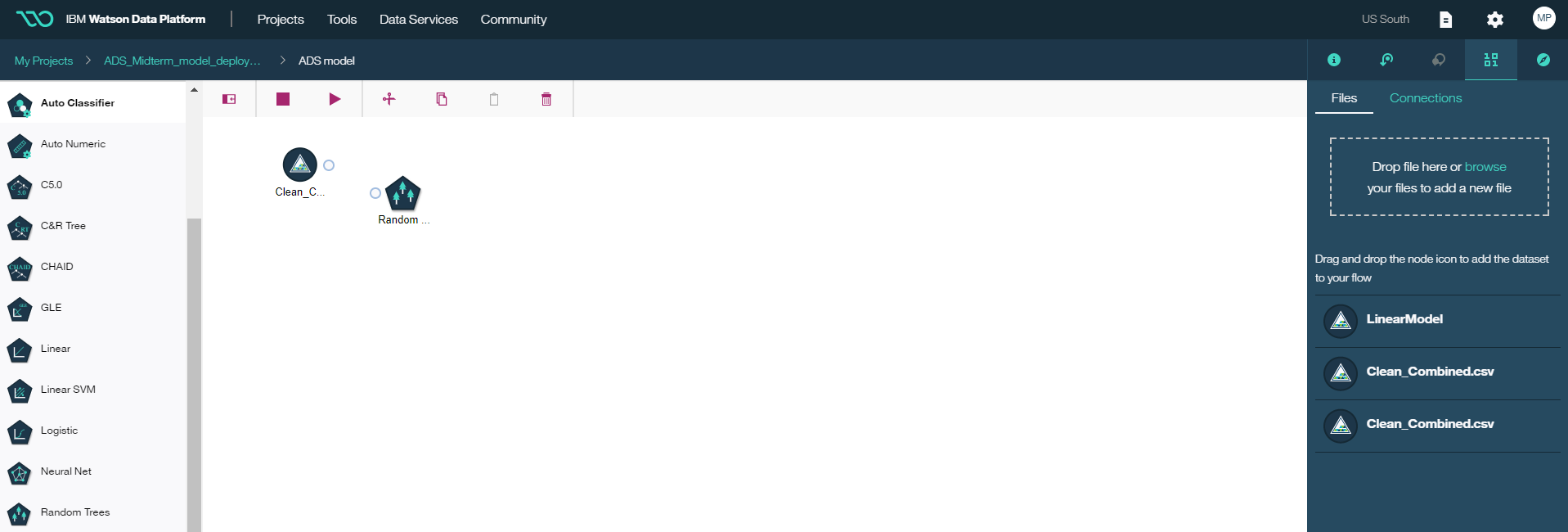
1. Once this is done you can go to the Asset tab and create a model. There are two ways to create a model.
   1. One is the similar way as what we did in Microsoft Azur, it is through the Flow area



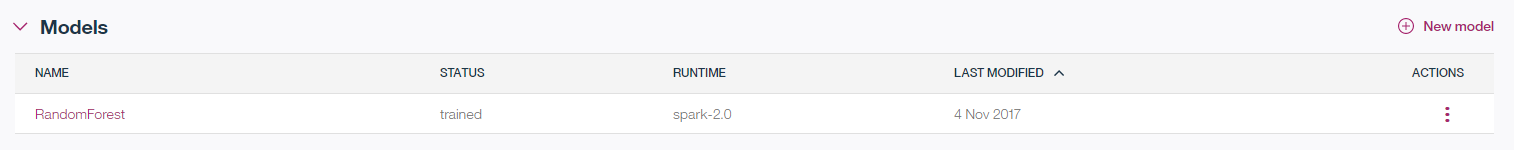
* + 1. When creating a new one you can create a SASS one or through spark services.
       1. Choose the spark services since it is what is easy and straight forward for deployment.



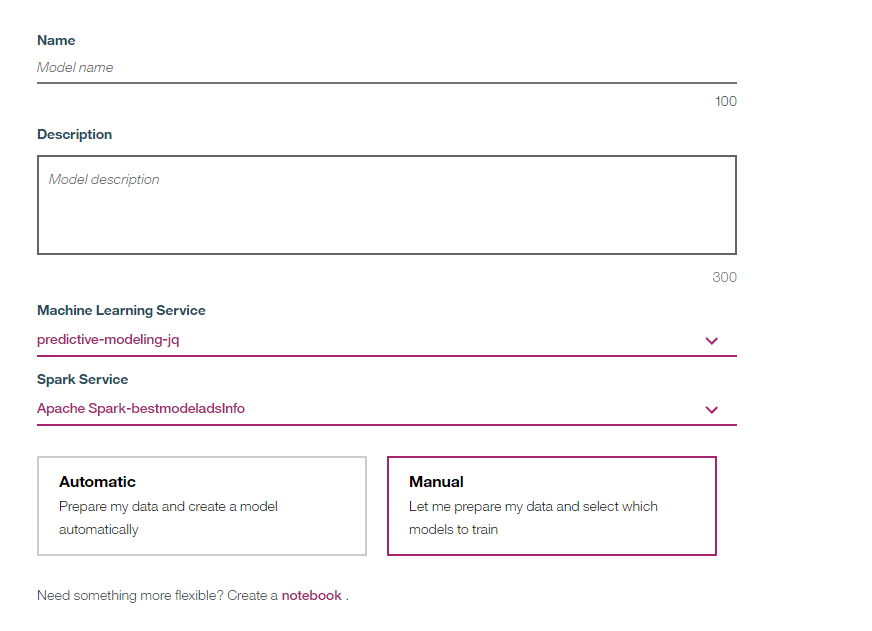
* + - 1. When you are creating for the first time the Machine learning service will not be there, click and create one. Just choose the free and name it.
    1. We pulled in our cleaned csv to here and attached the model we wanted to run from here.

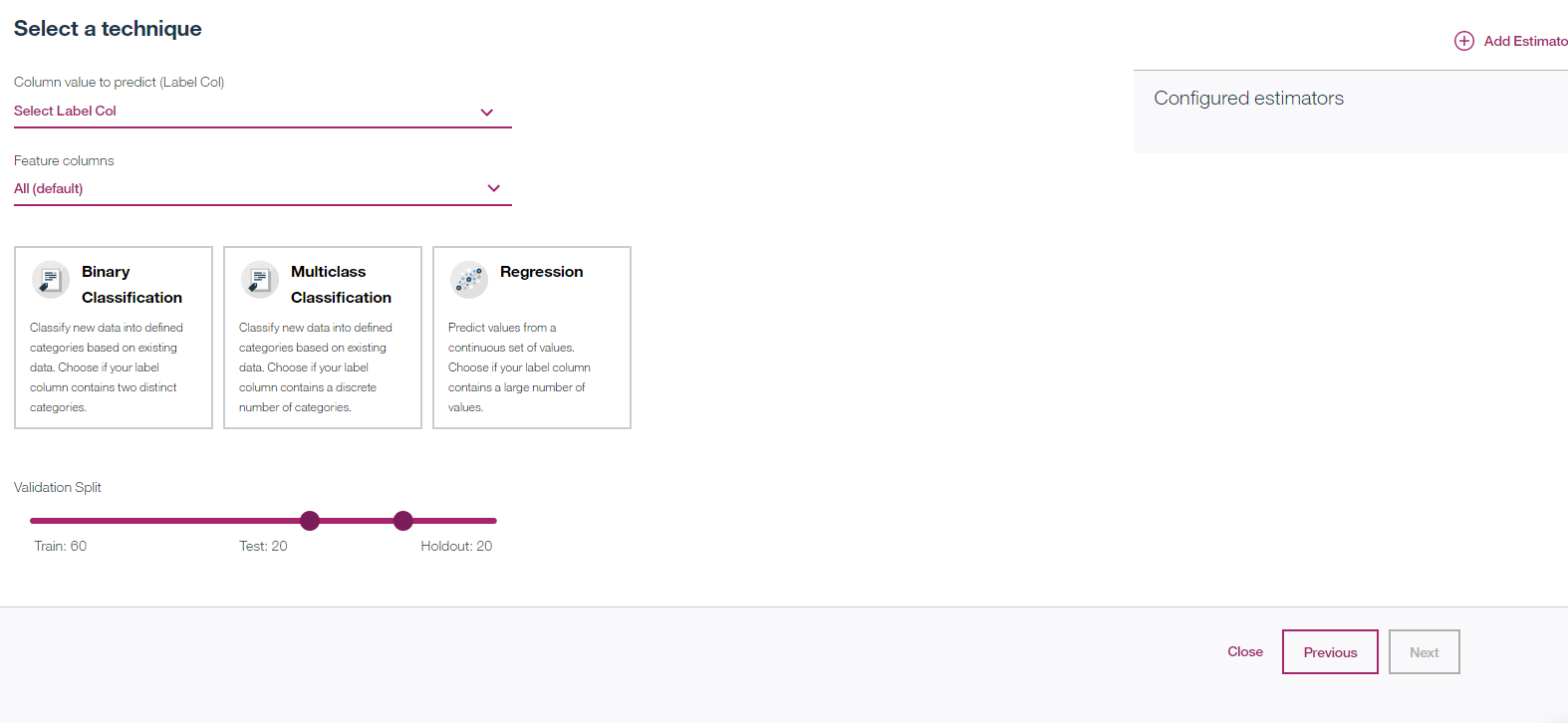


* 1. Another way that is different is that you can from the project Asset tab click on create model.



* 1. You will then be taken to a page to add CSV or choose existing one.
  2. Then on next page you can choose type of model and what the target and feature columns will be as well as how much testing vs training data will e used towards.





1. Then once model is trained and created we can deploy it

Deploy a model from the model builder

When you create a model by using the model builder, you can deploy the model directly from the model builder after you train it. This is also a good time to do a check on the data and the results.

Model builder deployment steps

On the model page, click Add Deployment.

On the Deploy model page, select the Online deployment type and type a deployment name.

Click Deploy.

When model deployment is complete, from the Actions menu, click View. The Deployment Details window appears. Note the scoring end point for future reference. You can only have one deployment per user.

Now is a good time to test the model prediction.

Go to the Test API tab.

The input data is populated with a sample record from the data set.

To test the model, change the values and click Predict.

1. Once that is done we can deploy it and follow this link to write jupyter notebook
   1. [https://console.bluemix.net/docs/services/PredictiveModeling/pm\_service\_api\_spark\_scoring.html#python](https://console.bluemix.net/docs/services/PredictiveModeling/pm_service_api_spark_scoring.html%23python)
      1. Choose the python tab in link

