Regression Analysis of Short Term Rentals: A comparison of machine learning performance between OLS and other Regression Modelling techniques

Jawsem Al-Hashash

Western Governors University

Research Question

Airbnb is a platform that allows hosts to rent out properties to patrons looking to visit locations all over the world. This study will explore data on Airbnb rentals in New York city to predict the price of a rental. It will also utilize census data and other geospatial data sources for prediction. Do penalized linear and decision tree based regression methods perform better than Ordinary Least Squares regression? Our null and alternative hypotheses are listed below:

H0: **There is not a large difference(less than 10%) in machine learning performance between penalized linear and decision tree based regression methods and ordinary least squares regression.**

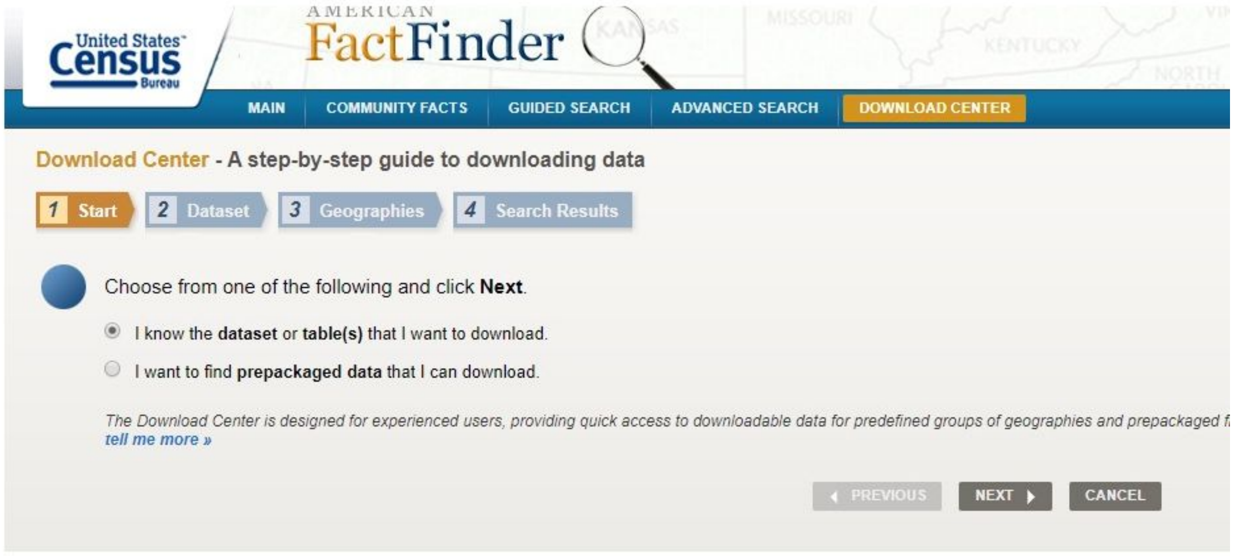
H1: **There is a large difference(greater than 10%) in machine learning performance between penalized linear and decision tree based regression methods and ordinary least squares regression.**

The mean squared error of four different regression techniques, 2 penalized and 2 decision tree based, will be compared with the mean squared error of OLS regression. If at least one of the four regression techniques has at least 10% less mean squared error than OLS regression, the null hypothesis will be rejected. There is merit to studying this topic in the context of Airbnb rentals because Airbnb rentals can have an effect on housing costs (Barron, Kung, & Proserpio, 2019) and studies in the past have shown that alternative regression techniques can perform better than OLS regression(Kan, Kharrazi, Chang, Bodycombe, Lemke & Weiner, 2019).

Data Collection

Data from three sources will be used for this study. The datasets will be merged into one final dataset for analysis. All three datasets are available for public use. The first dataset being used is available for download on the kaggle website. Kaggle is an online data science community that hosts data science and machine learning competitions. The New York City Airbnb data is located here. The second data source is Census data from the American Factfinder website. American Factfinder stores historical census data. The data this study will use is from the American Community Survey. Five tables from the 2017 ACS 5 Year estimates will be used. The tables are ACS_17_5YR_S0101, ACS_17_5YR_S1501, ACS_17_5YR_S1701, ACS_17_5YR_S1903, and ACS_17_5YR_S2301. The initial approach was to extract data from the American Factfinder website via an API, however this required an API key to be generated. Due to time constraints, this hurdle was overcome by exploring the American Factfinder website and finding the download interface. The interface was used to access the American Community Survey tables that were used for the study. The data was pulled for all Census Tracts in the state of New York.

The series of screenshots below describe how to download these datasets.



Download Center - A step-by-step guide to downloading data

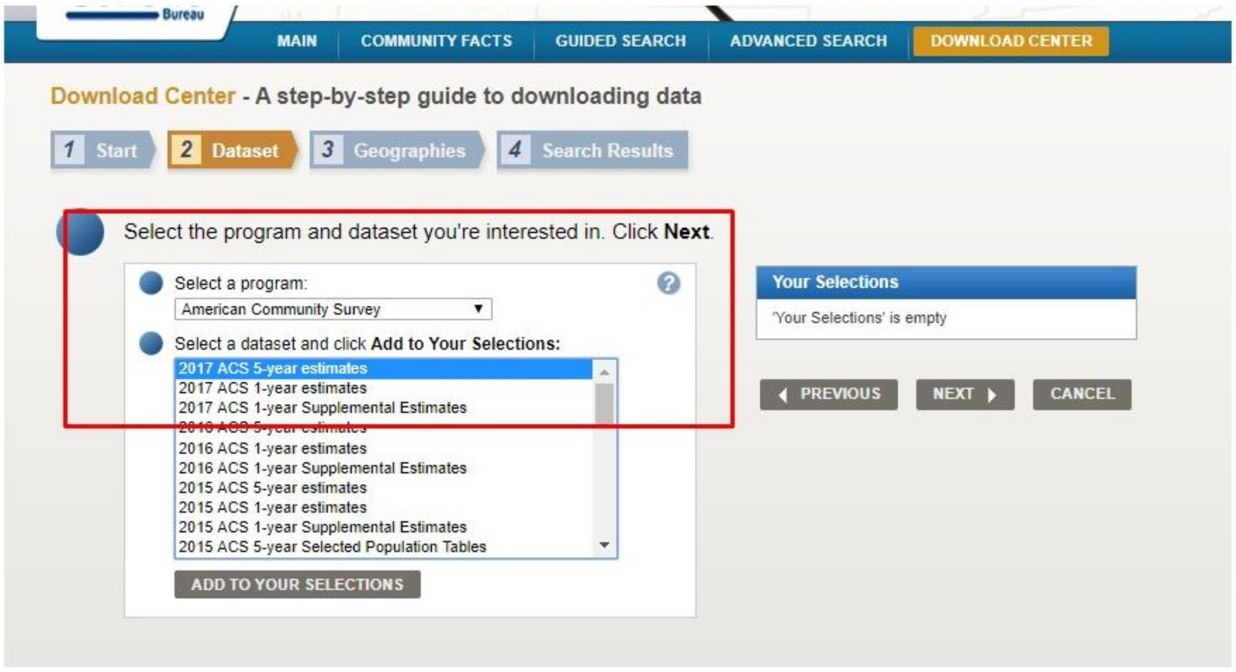
1 Start 2 Dataset 3 Geographies 4 Search Results

Choose from one of the following and click **Next**.

(●) I know the **dataset** or **table(s)** that I want to download.

( ) I want to find **prepackaged data** that I can download.

The Download Center is designed for experienced users, providing quick access to downloadable data for predefined groups of geographies and prepackaged f
tell me more »



Download Center - A step-by-step guide to downloading data

1 Start 2 Dataset 3 Geographies 4 Search Results

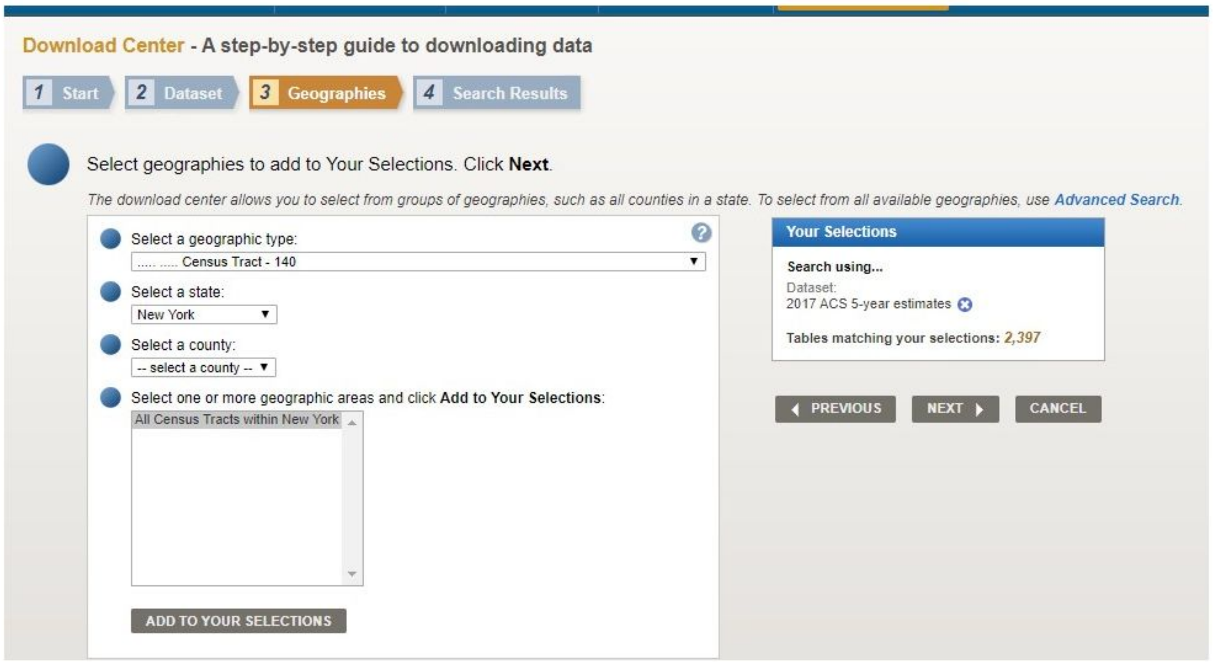
Select the program and dataset you're interested in. Click **Next**.

Select a program: American Community Survey

Select a dataset and click **Add to Your Selections:**

2017 ACS 5-year estimates
2017 ACS 1-year estimates
2017 ACS 1-year Supplemental Estimates
2016 ACS 5-year estimates
2016 ACS 1-year estimates
2016 ACS 1-year Supplemental Estimates
2015 ACS 5-year estimates
2015 ACS 1-year estimates
2015 ACS 1-year Supplemental Estimates
2015 ACS 5-year Selected Population Tables

Your Selections
'Your Selections' is empty



Download Center - A step-by-step guide to downloading data

Select geographies to add to Your Selections. Click **Next**.

The download center allows you to select from groups of geographies, such as all counties in a state. To select from all available geographies, use Advanced Search.

Select a geographic type: .... Census Tract - 140

Select a state: New York

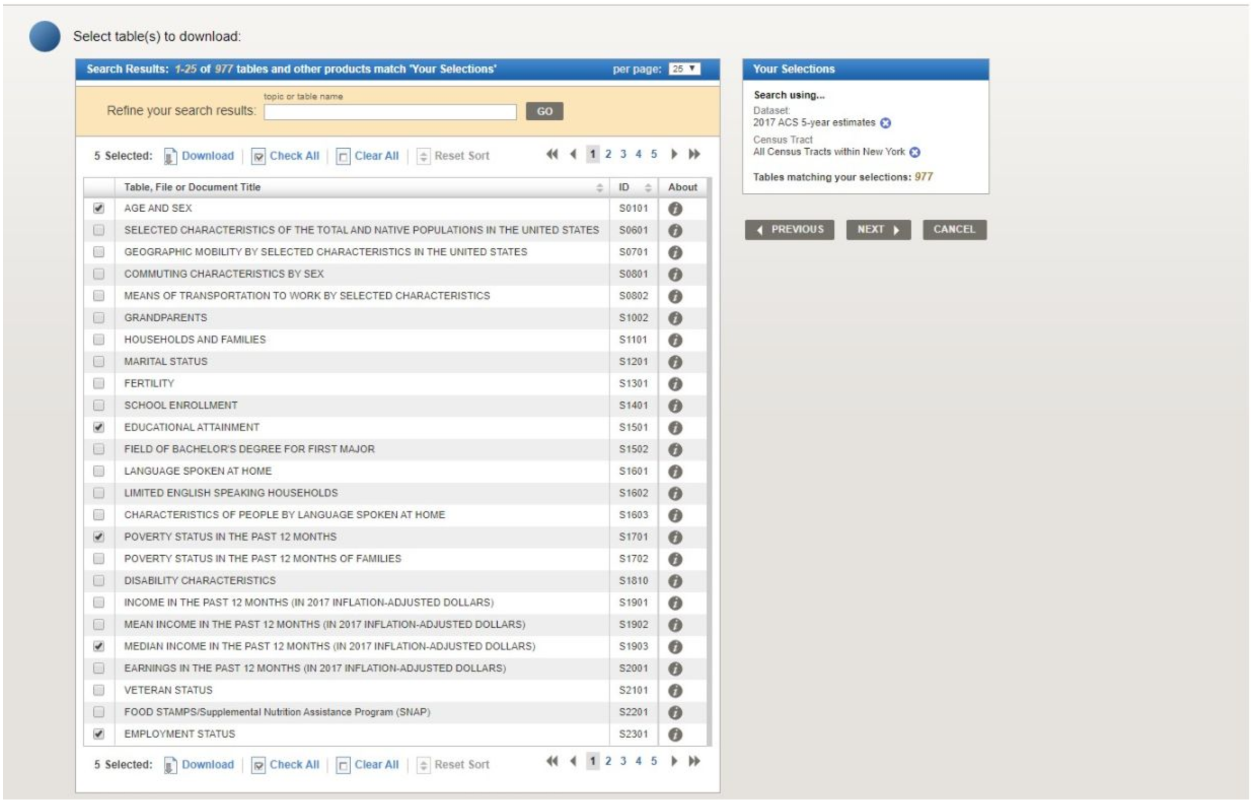
Select a county: -- select a county --

Select one or more geographic areas and click **Add to Your Selections**: All Census Tracts within New York

Your Selections

Search using...
Dataset:
2017 ACS 5-year estimates

Tables matching your selections: 2,397



Select table(s) to download:

Search Results: 1-25 of 977 tables and other products match 'Your Selections' per page: 25

Refine your search results: topic or table name

5 Selected: Download | Check All | Clear All | Reset Sort 1 2 3 4 5

	Table, File or Document Title	ID	About
✓	AGE AND SEX	S0101	
	SELECTED CHARACTERISTICS OF THE TOTAL AND NATIVE POPULATIONS IN THE UNITED STATES	S0601	
	GEOGRAPHIC MOBILITY BY SELECTED CHARACTERISTICS IN THE UNITED STATES	S0701	
	COMMUTING CHARACTERISTICS BY SEX	S0801	
	MEANS OF TRANSPORTATION TO WORK BY SELECTED CHARACTERISTICS	S0802	
	GRANDPARENTS	S1002	
	HOUSEHOLDS AND FAMILIES	S1101	
	MARITAL STATUS	S1201	
	FERTILITY	S1301	
	SCHOOL ENROLLMENT	S1401	
✓	EDUCATIONAL ATTAINMENT	S1501	
	FIELD OF BACHELOR'S DEGREE FOR FIRST MAJOR	S1502	
	LANGUAGE SPOKEN AT HOME	S1601	
	LIMITED ENGLISH SPEAKING HOUSEHOLDS	S1602	
	CHARACTERISTICS OF PEOPLE BY LANGUAGE SPOKEN AT HOME	S1603	
✓	POVERTY STATUS IN THE PAST 12 MONTHS	S1701	
	POVERTY STATUS IN THE PAST 12 MONTHS OF FAMILIES	S1702	
	DISABILITY CHARACTERISTICS	S1810	
	INCOME IN THE PAST 12 MONTHS (IN 2017 INFLATION-ADJUSTED DOLLARS)	S1901	
	MEAN INCOME IN THE PAST 12 MONTHS (IN 2017 INFLATION-ADJUSTED DOLLARS)	S1902	
✓	MEDIAN INCOME IN THE PAST 12 MONTHS (IN 2017 INFLATION-ADJUSTED DOLLARS)	S1903	
	EARNINGS IN THE PAST 12 MONTHS (IN 2017 INFLATION-ADJUSTED DOLLARS)	S2001	
	VETERAN STATUS	S2101	
	FOOD STAMPS/Supplemental Nutrition Assistance Program (SNAP)	S2201	
✓	EMPLOYMENT STATUS	S2301	

5 Selected: Download | Check All | Clear All | Reset Sort 1 2 3 4 5

Your Selections

Search using...
Dataset:
2017 ACS 5-year estimates
Census Tract
All Census Tracts within New York

Tables matching your selections: 977

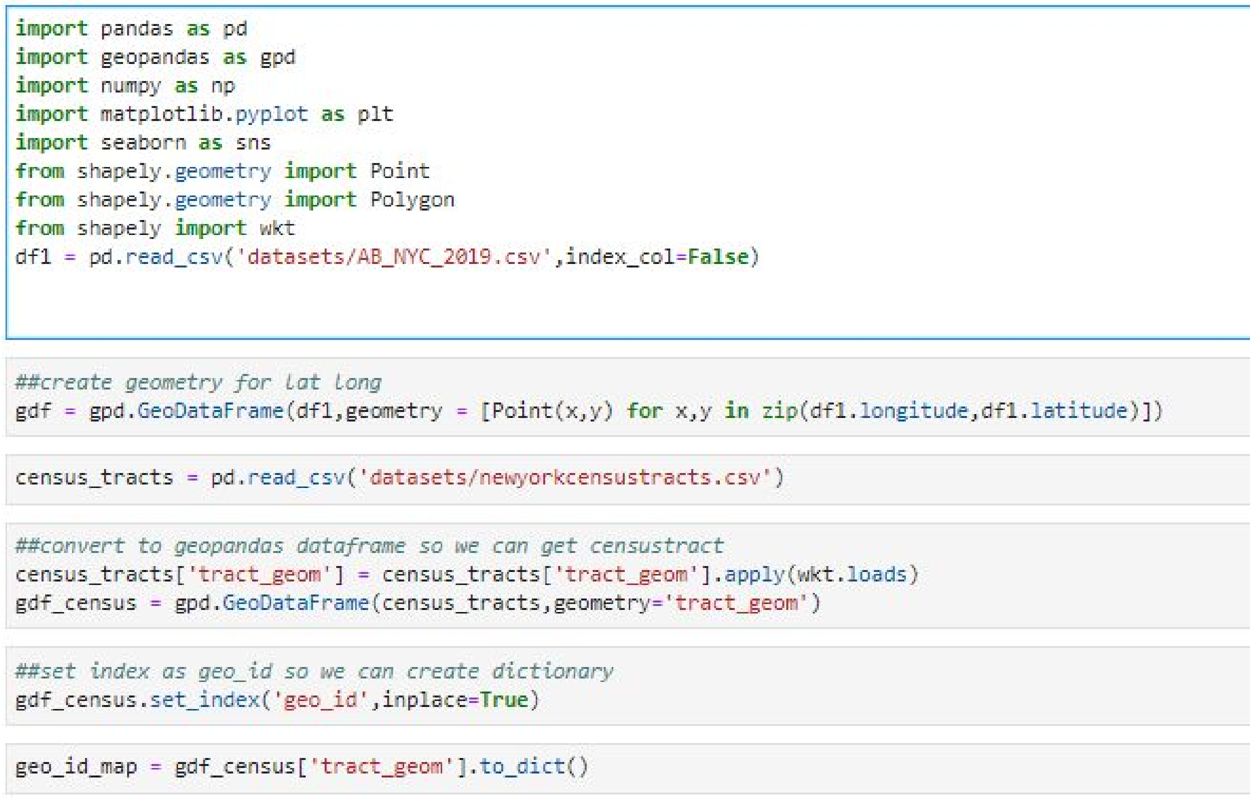
The third source of data is the Google Big Query public data repository. Public datasets from the New York tree census, subway entrance locations and census tracts were extracted. The full tables that were extracted from the Google Big Query interface are bigquery-public-data.new_york_subway.station_entrances, bigquery-publicdata.geo_census_tracts.census_tracts_new_york and bigquery-publicdata.new_york.tree_census_2015. The tables are extracted using ***select * from [TABLE]*** queries from the Google Big Query web UI. For easy access, all datasets from each of these sources are saved in at this link.

All of the datasets used for this analysis were extracted manually from a few different public websites. The advantages of using this methodology is that it does not require a significant technical effort to acquire the data. The data could also be acquired in a similar fashion by following the data collection steps above. It also does not require any API keys or API calls to be made. A disadvantage of this methodology is that it requires a human to manually go out to these websites and extract the data. It also requires some research to look into where to acquire data that would be useful to our analysis. A more sophisticated method would be to extract data via an API or via a web-scraping technique using python. This would remove a lot of the manual steps of going out to each website and downloading the data. It would also allow all the work to be done automatically via python code.

Data Extraction and Preparation

Data was extracted and prepared using the Python programming language. The first step of the process is to load the libraries needed for our data preparation. Pandas is loaded for data preparation and file processing. Geopandas, the Shapely Point and Polygon classes and the shapely function wkt are loaded for the handling of geospatial location data. Numpy is loaded for matrix and mathematical processing. Finally the seaborn and matplotlib libraries are loaded for data visualization.

The next step in our process is to load the initial Airbnb New York city data set into a pandas dataframe. Once loaded we convert the pandas dataframe into a geopandas dataframe, using the latitude and longitude as the geometry for the dataframe. The same loading and conversion process is done with the New York Census tract dataset, however we are using creating a Polygon as the geometry of the geopandas dataframe instead of a Point.



```
import pandas as pd
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from shapely.geometry import Point
from shapely.geometry import Polygon
from shapely import wkt
df1 = pd.read_csv('datasets/AB_NYC_2019.csv',index_col=False)
```

```
##create geometry for lat long
gdf = gpd.GeoDataFrame(df1,geometry = [Point(x,y) for x,y in zip(df1.longitude,df1.latitude)])
```

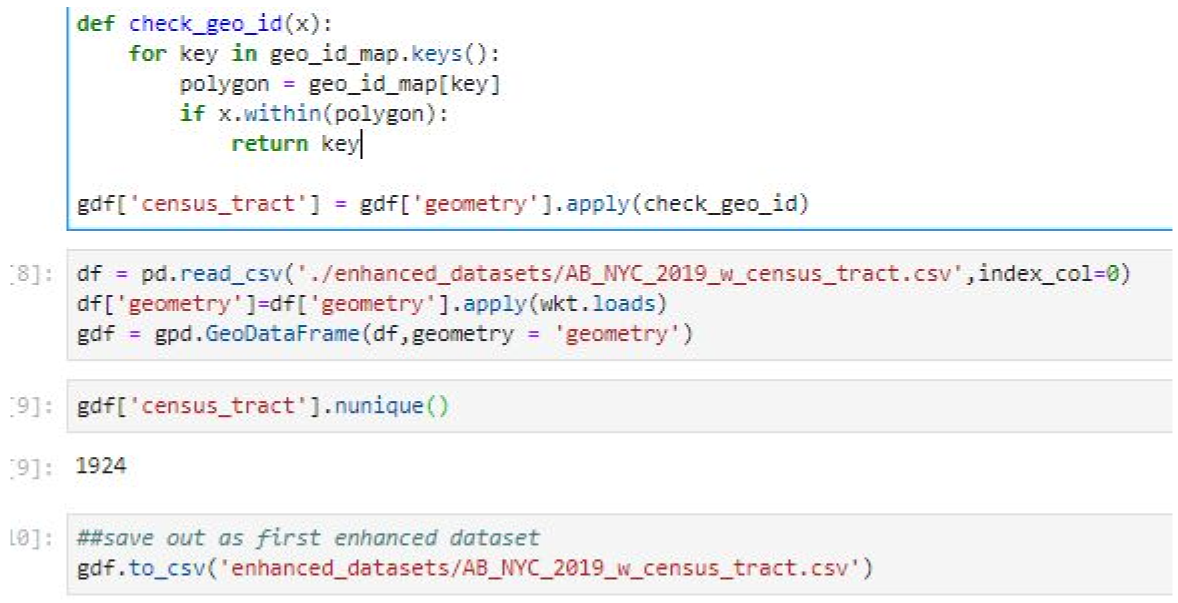
```
census_tracts = pd.read_csv('datasets/newyorkcensustracts.csv')
```

```
##convert to geopandas dataframe so we can get censustract
census_tracts['tract_geom'] = census_tracts['tract_geom'].apply(wkt.loads)
gdf_census = gpd.GeoDataFrame(census_tracts,geometry='tract_geom')
```

```
##set index as geo_id so we can create dictionary
gdf_census.set_index('geo_id',inplace=True)
```

```
geo_id_map = gdf_census['tract_geom'].to_dict()
```

The Point class in the context of the geopandas and shapely libraries is a constructor that takes a positional coordinate value or tuple parameter and creates a specific Point in a two dimensional space (The Shapely User Manual — Shapely 1.6 documentation, 2019). The Polygon class takes an ordered sequence of parameters and creates a two dimensional object connecting those points. The Point and Polygon classes can be used with latitudes and longitudes to perform various calculations. In the context of this analysis, the *within* method is used on a specific point to check if it is within a Polygon. This is done to acquire the specific census tract for each observation in our Airbnb dataset. The code that does this check is in the *check_geo_id* function below. This code takes a few minutes to run but once done we have a census tract for each observation from the Airbnb data. In total there are 1924 unique census tracts in our data. The data is saved out to a new enhanced dataset so the code does not need to be run again.



```
def check_geo_id(x):
    for key in geo_id_map.keys():
        polygon = geo_id_map[key]
        if x.within(polygon):
            return key

gdf['census_tract'] = gdf['geometry'].apply(check_geo_id)
```

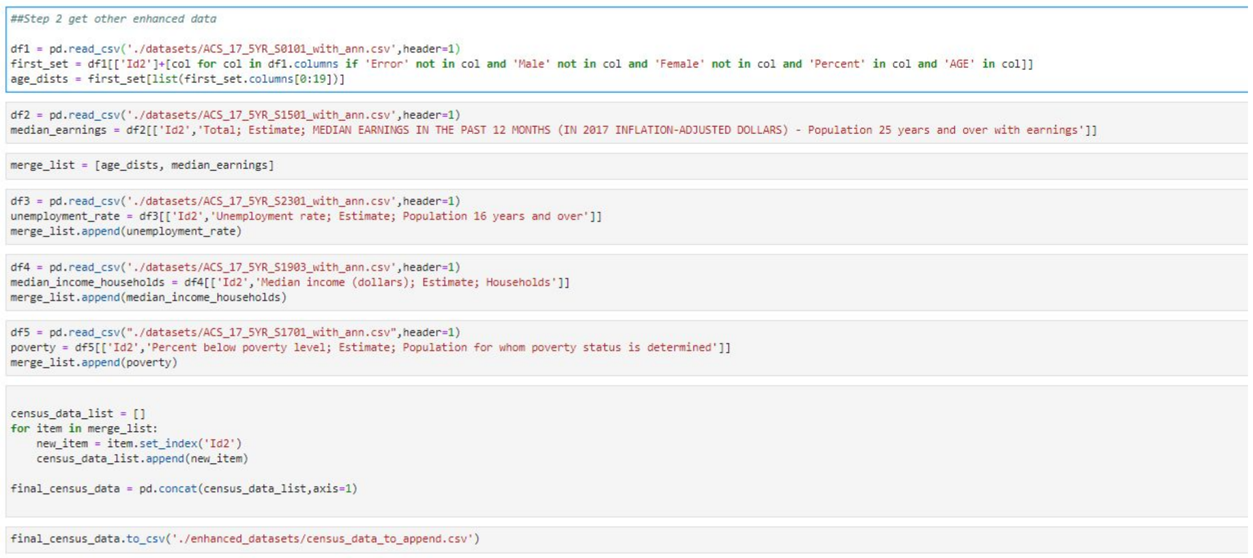
```
[8]: df = pd.read_csv('./enhanced_datasets/AB_NYC_2019_w_census_tract.csv',index_col=0)
     df['geometry']=df['geometry'].apply(wkt.loads)
     gdf = gpd.GeoDataFrame(df,geometry = 'geometry')
```

```
[9]: gdf['census_tract'].nunique()
```

```
[9]: 1924
```

```
10]: ##save out as first enhanced dataset
     gdf.to_csv('enhanced_datasets/AB_NYC_2019_w_census_tract.csv')
```

The next step in the data merging process is to merge together all five our census datasets. Each table is saved in a .csv file so we read our data into a dataframe and look at each set of data so we can manually choose what data we want to append. The S0101 data contains age distributions for each census tract. Eighteen columns are chosen from the dataset for age distributions from Under 5 years old to over 85 years old. Certain age distributions may have a correlation with the price of a rental so they may be useful features for prediction. At least one feature from each of our remaining census data sets. Median earnings metrics are pulled from the S1903 and S1502 datasets. Unemployment rate is pulled from the S2301 dataset and Percent below poverty level is pulled from the S1701 datasets. These features could also be correlated with Airbnb prices, so they may be useful to the models we build.



```
##Step 2 get other enhanced data

df1 = pd.read_csv('./datasets/ACS_17_5YR_S0101_with_ann.csv',header=1)
first_set = df1[['Id2']+[col for col in df1.columns if 'Error' not in col and 'Male' not in col and 'Female' not in col and 'Percent' in col and 'AGE' in col]]
age_dists = first_set[list(first_set.columns[0:19])]
```

```
df2 = pd.read_csv('./datasets/ACS_17_5YR_S1501_with_ann.csv',header=1)
median_earnings = df2[['Id2','Total; Estimate; MEDIAN EARNINGS IN THE PAST 12 MONTHS (IN 2017 INFLATION-ADJUSTED DOLLARS) - Population 25 years and over with earnings']]
```

```
merge_list = [age_dists, median_earnings]
```

```
df3 = pd.read_csv('./datasets/ACS_17_5YR_S2301_with_ann.csv',header=1)
unemployment_rate = df3[['Id2','Unemployment rate; Estimate; Population 16 years and over']]
merge_list.append(unemployment_rate)
```

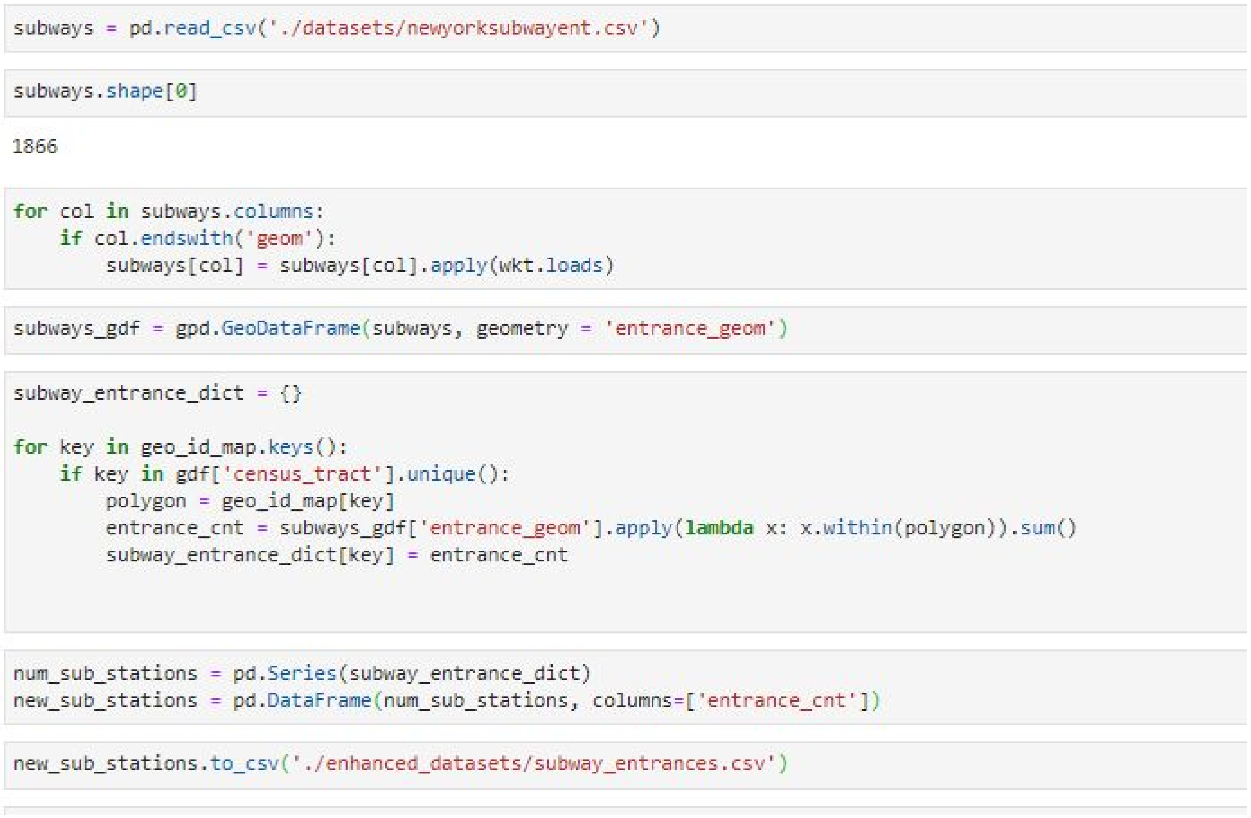
```
df4 = pd.read_csv('./datasets/ACS_17_5YR_S1903_with_ann.csv',header=1)
median_income_households = df4[['Id2','Median income (dollars); Estimate; Households']]
merge_list.append(median_income_households)
```

```
df5 = pd.read_csv("./datasets/ACS_17_5YR_S1701_with_ann.csv",header=1)
poverty = df5[['Id2','Percent below poverty level; Estimate; Population for whom poverty status is determined']]
merge_list.append(poverty)
```

```
census_data_list = []
for item in merge_list:
    new_item = item.set_index('Id2')
    census_data_list.append(new_item)

final_census_data = pd.concat(census_data_list,axis=1)
```

```
final_census_data.to_csv('./enhanced_datasets/census_data_to_append.csv')
```

The next two datasets that are going to be merged are the subway entrance and new york trees datasets. The number of subway entrances, alive trees, tree stumps and dead trees will be aggregated per census tract and used to merge with our Airbnb data. This is done in a similar fashion as merging the census tract with our Airbnb data. The *within* method will be used for each census tract polygon and then that will be summed up and added to a dictionary that contains the census tracts and sum. The dictionaries will be used to create a dataframe with the counts for each census tract. In total, there are 1866 subway entrances to count up and nearly 700,000 trees. The process that creates the dictionaries can take an hour or two.



```python
subways = pd.read_csv('./datasets/newyorksubwayent.csv')
```

```python
subways.shape[0]
```

```
1866
```

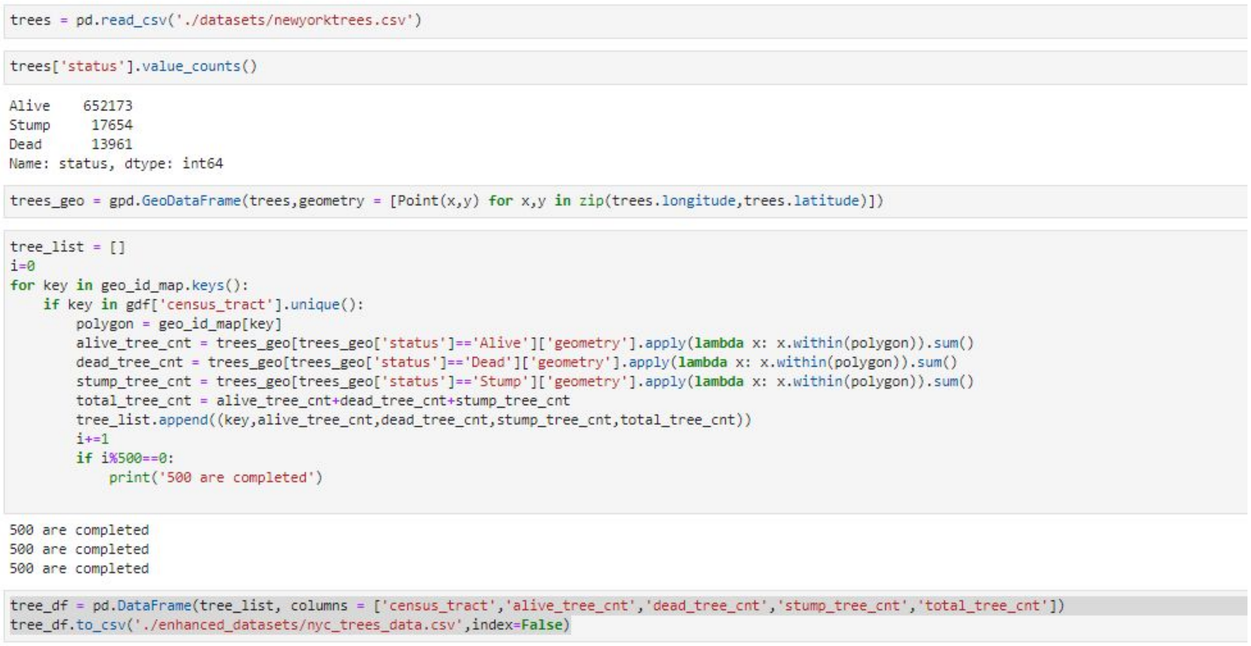
```python
for col in subways.columns:
    if col.endswith('geom'):
        subways[col] = subways[col].apply(wkt.loads)
```

```python
subways_gdf = gpd.GeoDataFrame(subways, geometry = 'entrance_geom')
```

```python
subway_entrance_dict = {}

for key in geo_id_map.keys():
    if key in gdf['census_tract'].unique():
        polygon = geo_id_map[key]
        entrance_cnt = subways_gdf['entrance_geom'].apply(lambda x: x.within(polygon)).sum()
        subway_entrance_dict[key] = entrance_cnt
```

```python
num_sub_stations = pd.Series(subway_entrance_dict)
new_sub_stations = pd.DataFrame(num_sub_stations, columns=['entrance_cnt'])
```

```python
new_sub_stations.to_csv('./enhanced_datasets/subway_entrances.csv')
```



```python
trees = pd.read_csv('./datasets/newyorktrees.csv')
```

```python
trees['status'].value_counts()
```

```
Alive    652173
Stump     17654
Dead      13961
Name: status, dtype: int64
```

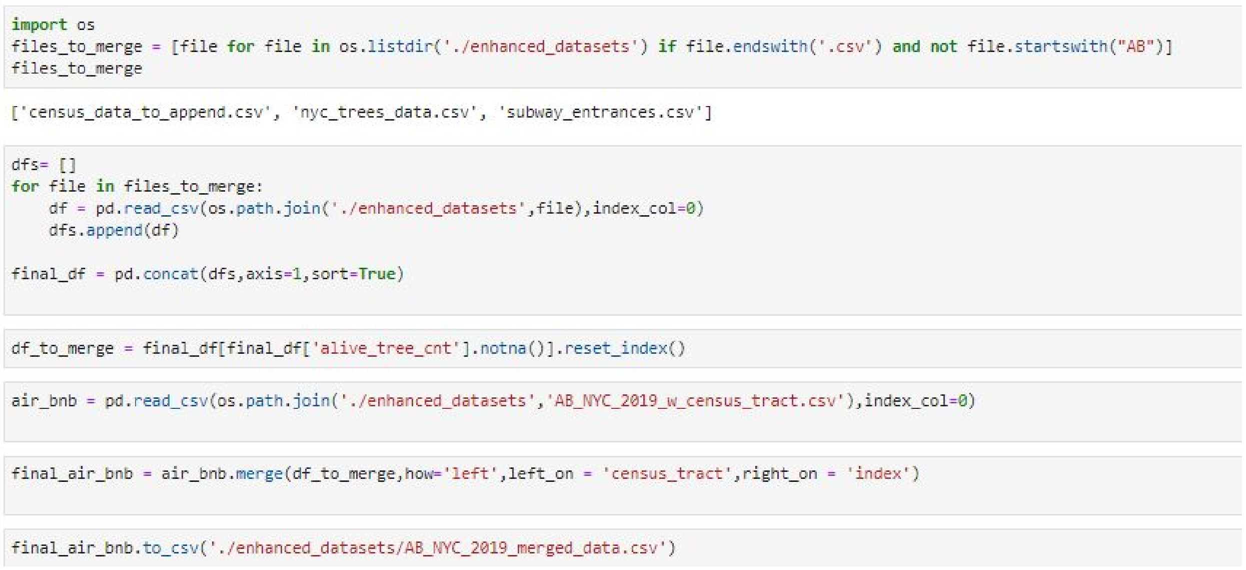
```python
trees_geo = gpd.GeoDataFrame(trees,geometry = [Point(x,y) for x,y in zip(trees.longitude,trees.latitude)])
```

```python
tree_list = []
i=0
for key in geo_id_map.keys():
    if key in gdf['census_tract'].unique():
        polygon = geo_id_map[key]
        alive_tree_cnt = trees_geo[trees_geo['status']=='Alive']['geometry'].apply(lambda x: x.within(polygon)).sum()
        dead_tree_cnt = trees_geo[trees_geo['status']=='Dead']['geometry'].apply(lambda x: x.within(polygon)).sum()
        stump_tree_cnt = trees_geo[trees_geo['status']=='Stump']['geometry'].apply(lambda x: x.within(polygon)).sum()
        total_tree_cnt = alive_tree_cnt+dead_tree_cnt+stump_tree_cnt
        tree_list.append((key,alive_tree_cnt,dead_tree_cnt,stump_tree_cnt,total_tree_cnt))
        i+=1
        if i%500==0:
            print('500 are completed')
```

```
500 are completed
500 are completed
500 are completed
```

```python
tree_df = pd.DataFrame(tree_list, columns = ['census_tract','alive_tree_cnt','dead_tree_cnt','stump_tree_cnt','total_tree_cnt'])
tree_df.to_csv('./enhanced_datasets/nyc_trees_data.csv',index=False)
```

All the acquired data is now ready to be merged into one final dataframe for modelling and analysis. The python os library is used to loop through all the files in the enhanced_datasets folder and merge them together into one dataset. This dataset is then merged with our Airbnb dataset by census tract to create a final dataset for modelling and analysis.



```
import os
files_to_merge = [file for file in os.listdir('./enhanced_datasets') if file.endswith('.csv') and not file.startswith("AB")]
files_to_merge
```

```
['census_data_to_append.csv', 'nyc_trees_data.csv', 'subway_entrances.csv']
```

```
dfs= []
for file in files_to_merge:
    df = pd.read_csv(os.path.join('./enhanced_datasets',file),index_col=0)
    dfs.append(df)

final_df = pd.concat(dfs,axis=1,sort=True)
```

```
df_to_merge = final_df[final_df['alive_tree_cnt'].notna()].reset_index()
```

```
air_bnb = pd.read_csv(os.path.join('./enhanced_datasets','AB_NYC_2019_w_census_tract.csv'),index_col=0)
```

```
final_air_bnb = air_bnb.merge(df_to_merge,how='left',left_on = 'census_tract',right_on = 'index')
```

```
final_air_bnb.to_csv('./enhanced_datasets/AB_NYC_2019_merged_data.csv')
```

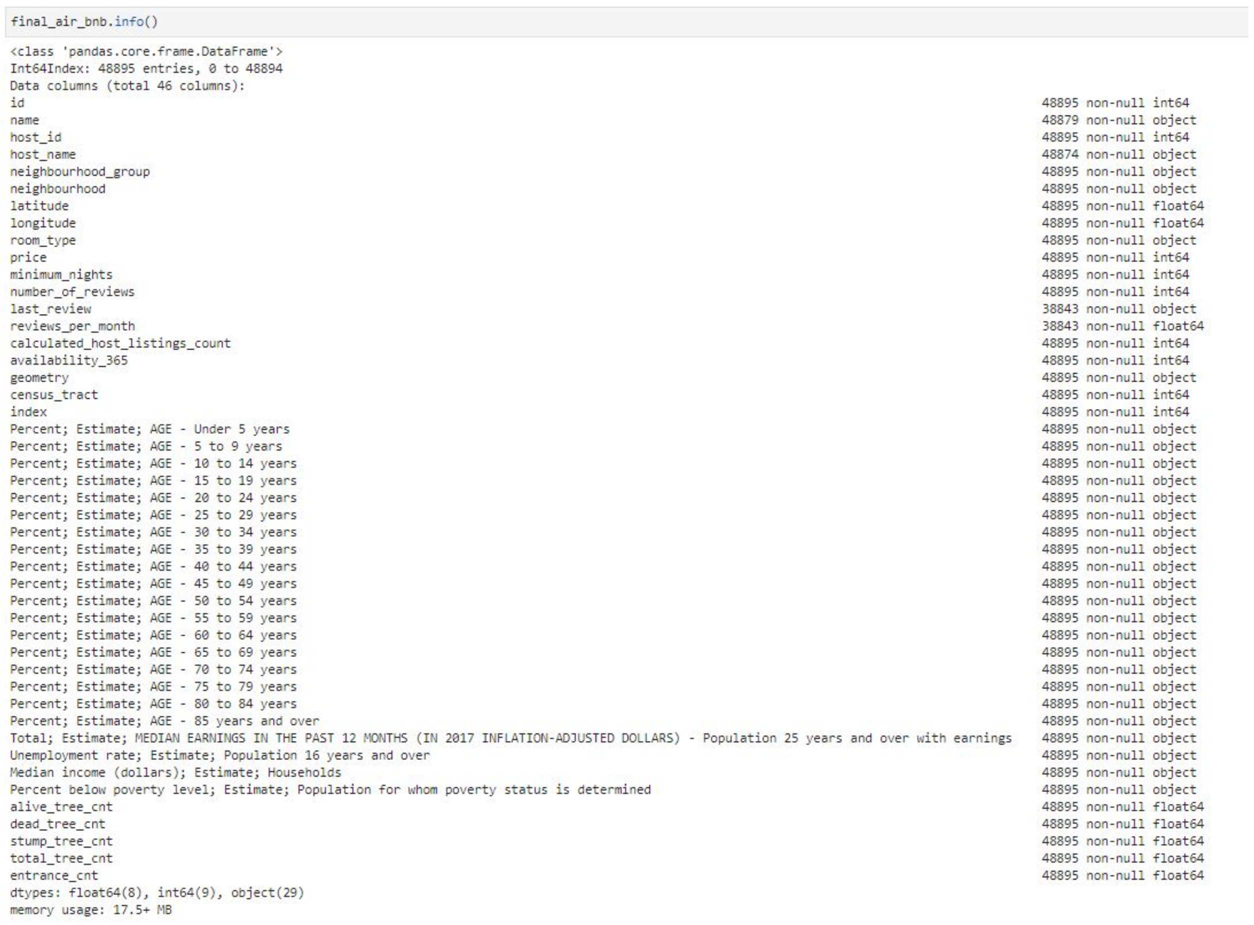
Python was used for the data preparation process. Python is a free, open source programming language that has a large development community. It has many useful packages for statistics, machine learning, geospatial analysis and data visualization. Python doesn’t require an expensive server, network or storage hardware that might be needed for a language like SPSS or SAS. Jupyter notebooks were used for the data preparation process. With a jupyter notebook, one can write code and explain it in the same interface. This makes it fairly straightforward to explain each of the steps that are taken in the data preparation process. Jupyter notebooks can also be saved in a variety of formats which makes them easy to share. For the data preparation process, the Shapely and geopandas libraries were used. The advantage of these libraries is that geospatial calculations with only a few lines of code.

Analysis

Analysis will be performed in 6 steps. First, exploratory data analysis will be performed to assess features, missing values and distributions. Second, a base model will be created using OLS (ordinary least squares) regression. The model will be assessed using 5-fold cross validation the average mean squared error across each fold will be used as the performance baseline. Steps 3-6 through, will be the model creation and assessment of the four regression techniques. The techniques are: Lasso, Ridge, Random Forest and XGBoost regression. Each model created will also be run through 5-fold cross validation and the average mean squared error will be used to compare with our baseline performance.

Exploratory Data Analysis

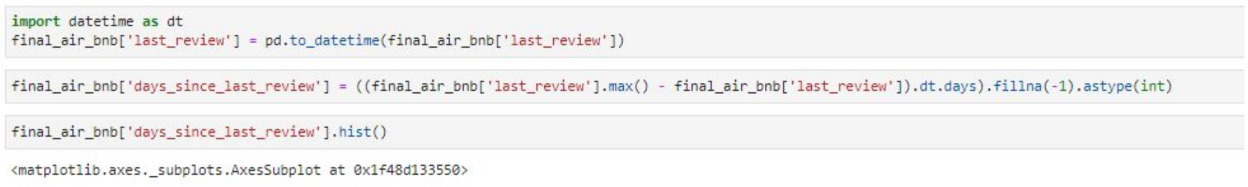
Exploratory data analysis will be performed on the dataset generated in our data extraction and preparation steps. Exploratory data analysis (EDA) is a type of data analysis the lends importance to keeping an open mind (Yu, 1977). The goals of this analysis is to understand the various distributions of our dataset, assess missing values and determine the features that will be most useful for our modelling. Various plots and charts will be created and feature engineering may also be performed if analysis shows it can be useful. The EDA code is below.



```
final_air_bnb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48895 entries, 0 to 48894
Data columns (total 46 columns):
id                                                                                                           48895 non-null int64
name                                                                                                         48879 non-null object
host_id                                                                                                      48895 non-null int64
host_name                                                                                                    48874 non-null object
neighbourhood_group                                                                                          48895 non-null object
neighbourhood                                                                                                48895 non-null object
latitude                                                                                                     48895 non-null float64
longitude                                                                                                    48895 non-null float64
room_type                                                                                                    48895 non-null object
price                                                                                                        48895 non-null int64
minimum_nights                                                                                               48895 non-null int64
number_of_reviews                                                                                            48895 non-null int64
last_review                                                                                                  38843 non-null object
reviews_per_month                                                                                            38843 non-null float64
calculated_host_listings_count                                                                               48895 non-null int64
availability_365                                                                                             48895 non-null int64
geometry                                                                                                     48895 non-null object
census_tract                                                                                                 48895 non-null int64
index                                                                                                        48895 non-null int64
Percent; Estimate; AGE - Under 5 years                                                                       48895 non-null object
Percent; Estimate; AGE - 5 to 9 years                                                                        48895 non-null object
Percent; Estimate; AGE - 10 to 14 years                                                                      48895 non-null object
Percent; Estimate; AGE - 15 to 19 years                                                                      48895 non-null object
Percent; Estimate; AGE - 20 to 24 years                                                                      48895 non-null object
Percent; Estimate; AGE - 25 to 29 years                                                                      48895 non-null object
Percent; Estimate; AGE - 30 to 34 years                                                                      48895 non-null object
Percent; Estimate; AGE - 35 to 39 years                                                                      48895 non-null object
Percent; Estimate; AGE - 40 to 44 years                                                                      48895 non-null object
Percent; Estimate; AGE - 45 to 49 years                                                                      48895 non-null object
Percent; Estimate; AGE - 50 to 54 years                                                                      48895 non-null object
Percent; Estimate; AGE - 55 to 59 years                                                                      48895 non-null object
Percent; Estimate; AGE - 60 to 64 years                                                                      48895 non-null object
Percent; Estimate; AGE - 65 to 69 years                                                                      48895 non-null object
Percent; Estimate; AGE - 70 to 74 years                                                                      48895 non-null object
Percent; Estimate; AGE - 75 to 79 years                                                                      48895 non-null object
Percent; Estimate; AGE - 80 to 84 years                                                                      48895 non-null object
Percent; Estimate; AGE - 85 years and over                                                                   48895 non-null object
Total; Estimate; MEDIAN EARNINGS IN THE PAST 12 MONTHS (IN 2017 INFLATION-ADJUSTED DOLLARS) - Population 25 years and over with earnings    48895 non-null object
Unemployment rate; Estimate; Population 16 years and over                                                    48895 non-null object
Median income (dollars); Estimate; Households                                                                48895 non-null object
Percent below poverty level; Estimate; Population for whom poverty status is determined                      48895 non-null object
alive_tree_cnt                                                                                               48895 non-null float64
dead_tree_cnt                                                                                                48895 non-null float64
stump_tree_cnt                                                                                               48895 non-null float64
total_tree_cnt                                                                                               48895 non-null float64
entrance_cnt                                                                                                 48895 non-null float64
dtypes: float64(8), int64(9), object(29)
memory usage: 17.5+ MB
```

A few items will need to be addressed with the object columns in order to prepare our dataset for modelling. One, the last review column is a date column with some null values. A date column would be hard to use for our model but we can use the column to create a feature which will be called days since last review. In order to generate this new feature, the last review column will need to be converted to a date. The maximum review date will be used as the date that will be subtracted from. If there are no reviews, the new feature will take the value of -1.

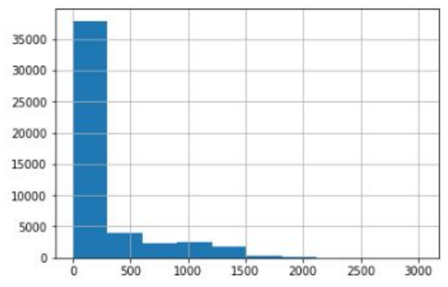


```
import datetime as dt
final_air_bnb['last_review'] = pd.to_datetime(final_air_bnb['last_review'])
```

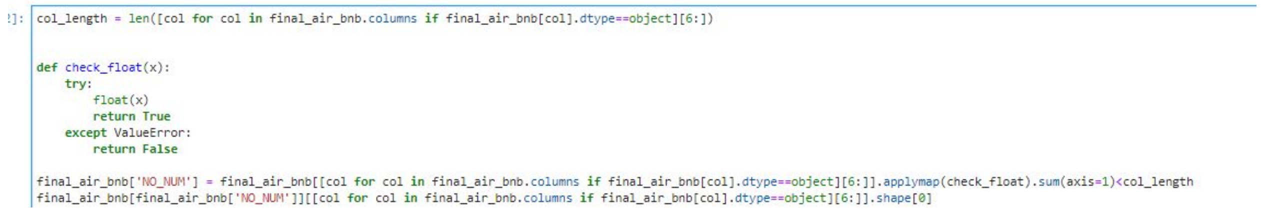
```
final_air_bnb['days_since_last_review'] = ((final_air_bnb['last_review'].max() - final_air_bnb['last_review']).dt.days).fillna(-1).astype(int)
```

```
final_air_bnb['days_since_last_review'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f48d133550>
```

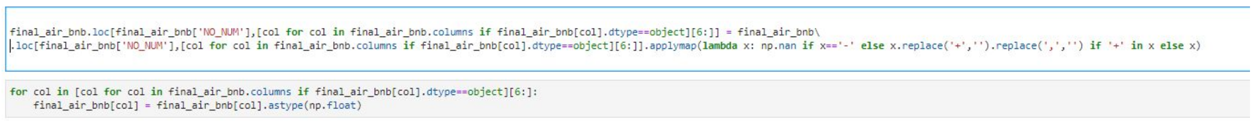


Second, the percent based columns have been read in as objects when we expect them to be numeric values. This data will be reviewed to determine how to do the numerical conversion.



```
!]: col_length = len([col for col in final_air_bnb.columns if final_air_bnb[col].dtype==object][6:])


    def check_float(x):
        try:
            float(x)
            return True
        except ValueError:
            return False

    final_air_bnb['NO_NUM'] = final_air_bnb[[col for col in final_air_bnb.columns if final_air_bnb[col].dtype==object][6:]].applymap(check_float).sum(axis=1)<col_length
    final_air_bnb[final_air_bnb['NO_NUM']][[col for col in final_air_bnb.columns if final_air_bnb[col].dtype==object][6:]].shape[0]
```

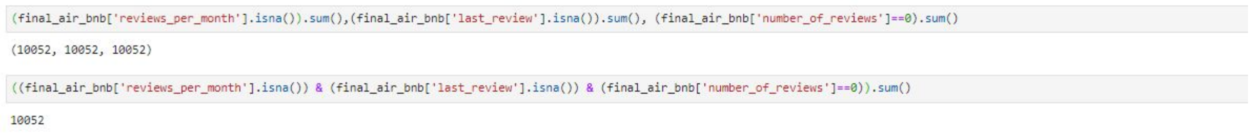
There are 283 records that have objects for what should be float columns. These are objects in two different ways. The first way is that there is a "-" filled in for the value. An assumption we can make with this value is that the data is not available and they can be filled with a null value. The second way is that there are figures that say "250,000+". For these cases, the "+" will be removed so we have a normal number. Since 283 records represent a small portion of our overall data set, the records for these figures can most likely be dropped before the modelling process begins.



```
final_air_bnb.loc[final_air_bnb['NO_NUM'],[col for col in final_air_bnb.columns if final_air_bnb[col].dtype==object][6:]] = final_air_bnb\
.loc[final_air_bnb['NO_NUM'],[col for col in final_air_bnb.columns if final_air_bnb[col].dtype==object][6:]].applymap(lambda x: np.nan if x=='-' else x.replace('+','').replace(',','') if '+' in x else x)
```

```
for col in [col for col in final_air_bnb.columns if final_air_bnb[col].dtype==object][6:]:
    final_air_bnb[col] = final_air_bnb[col].astype(np.float)
```

There are two columns with quite a few null values in them, last_review and reviews_per_month. The reason there may be null values is because there are 0 reviews for these properties. These will be compared with number_of_reviews to see if the theory is true.



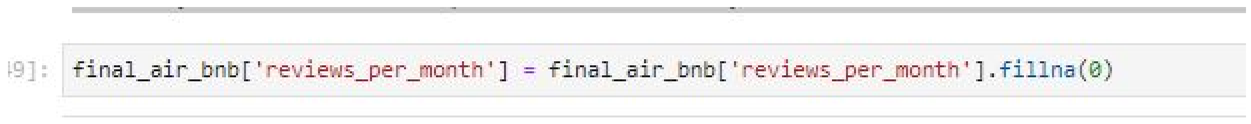
```
(final_air_bnb['reviews_per_month'].isna()).sum(),(final_air_bnb['last_review'].isna()).sum(), (final_air_bnb['number_of_reviews']==0).sum()
```

```
(10052, 10052, 10052)
```

```
((final_air_bnb['reviews_per_month'].isna()) & (final_air_bnb['last_review'].isna()) & (final_air_bnb['number_of_reviews']==0)).sum()
```

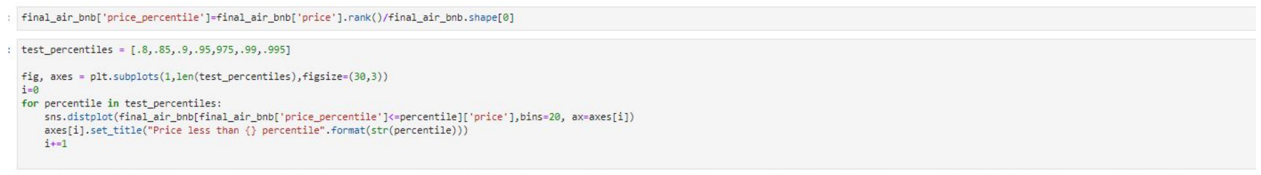
```
10052
```

The theory is correct. The reviews_per_month field can safely be filled with a 0 value. The last_review field does not need to be filled as we have already used it to create our feature for days_since_last_review.



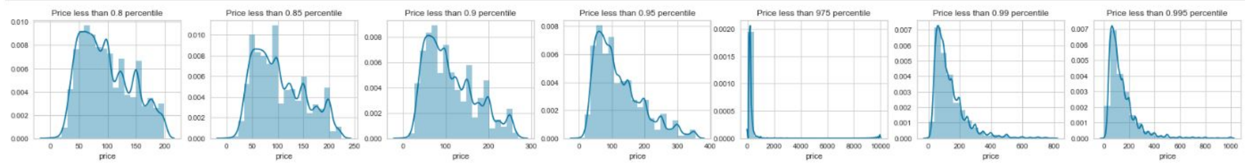
```
49]: final_air_bnb['reviews_per_month'] = final_air_bnb['reviews_per_month'].fillna(0)
```

In the next steps of the EDA process, various plots are created to examine the distribution of the target variable and its relationship to other variables.

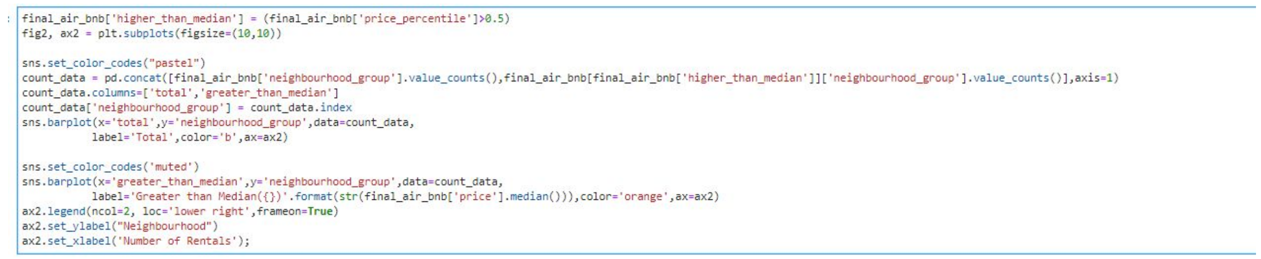


```
: final_air_bnb['price_percentile']=final_air_bnb['price'].rank()/final_air_bnb.shape[0]
```

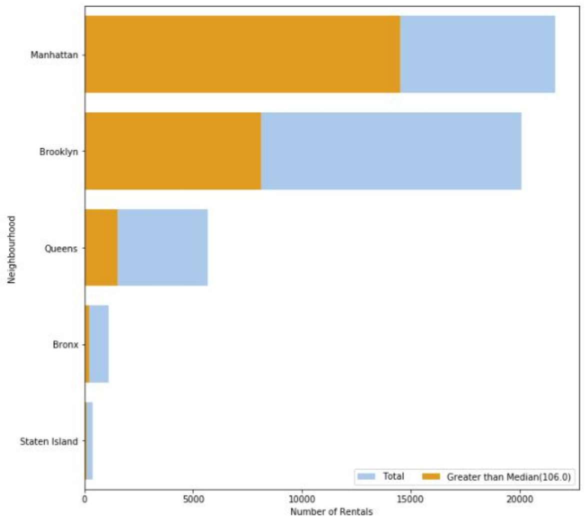
```
: test_percentiles = [.8,.85,.9,.95,975,.99,.995]

  fig, axes = plt.subplots(1,len(test_percentiles),figsize=(30,3))
  i=0
  for percentile in test_percentiles:
      sns.distplot(final_air_bnb[final_air_bnb['price_percentile']<=percentile]['price'],bins=20, ax=axes[i])
      axes[i].set_title("Price less than {} percentile".format(str(percentile)))
      i+=1
```

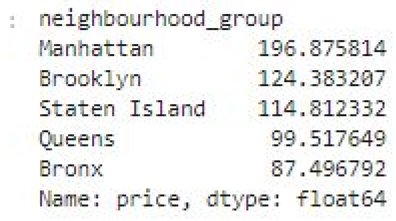


The target variable, price, is right skewed even when eliminating higher outliers. As higher percentile prices are eliminated the distribution starts to look more normal.



```
final_air_bnb['higher_than_median'] = (final_air_bnb['price_percentile']>0.5)
fig2, ax2 = plt.subplots(figsize=(10,10))

sns.set_color_codes("pastel")
count_data = pd.concat([final_air_bnb['neighbourhood_group'].value_counts(),final_air_bnb[final_air_bnb['higher_than_median']]['neighbourhood_group'].value_counts()],axis=1)
count_data.columns=['total','greater_than_median']
count_data['neighbourhood_group'] = count_data.index
sns.barplot(x='total',y='neighbourhood_group',data=count_data,
           label='Total',color='b',ax=ax2)

sns.set_color_codes('muted')
sns.barplot(x='greater_than_median',y='neighbourhood_group',data=count_data,
           label='Greater than Median({})'.format(str(final_air_bnb['price'].median())),color='orange',ax=ax2)
ax2.legend(ncol=2, loc='lower right',frameon=True)
ax2.set_ylabel("Neighbourhood")
ax2.set_xlabel('Number of Rentals');
```



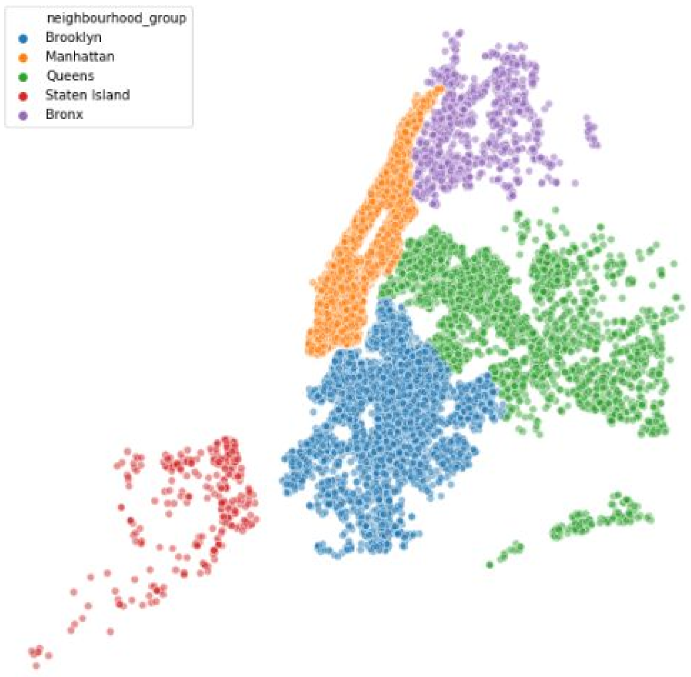


```
neighbourhood_group
Manhattan        196.875814
Brooklyn         124.383207
Staten Island    114.812332
Queens            99.517649
Bronx             87.496792
Name: price, dtype: float64
```

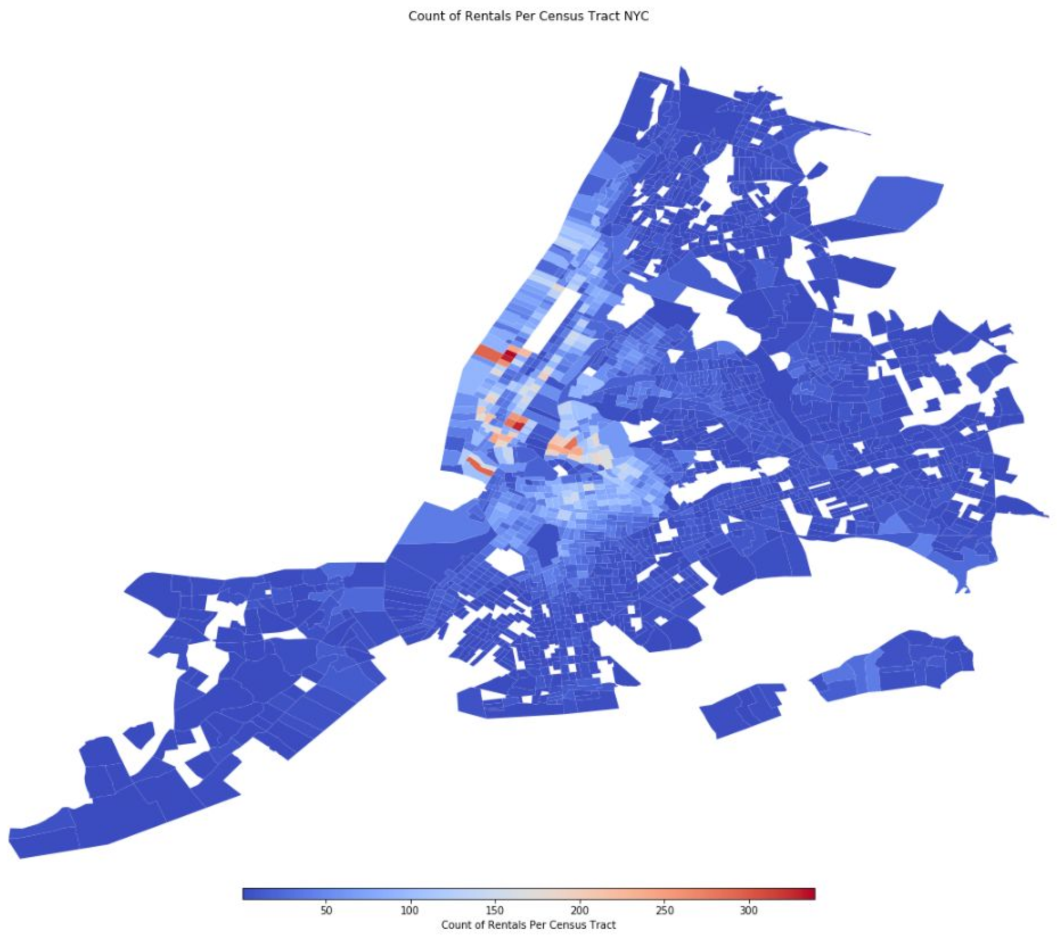
From the above plot, one can see that Manhattan and Brooklyn make the majority of our dataset. Manhattan has the highest proportion of rentals with a price greater than our median. The average price of a rental in Manhattan is more than 70 dollars more than the average price in the next highest, Brooklyn.

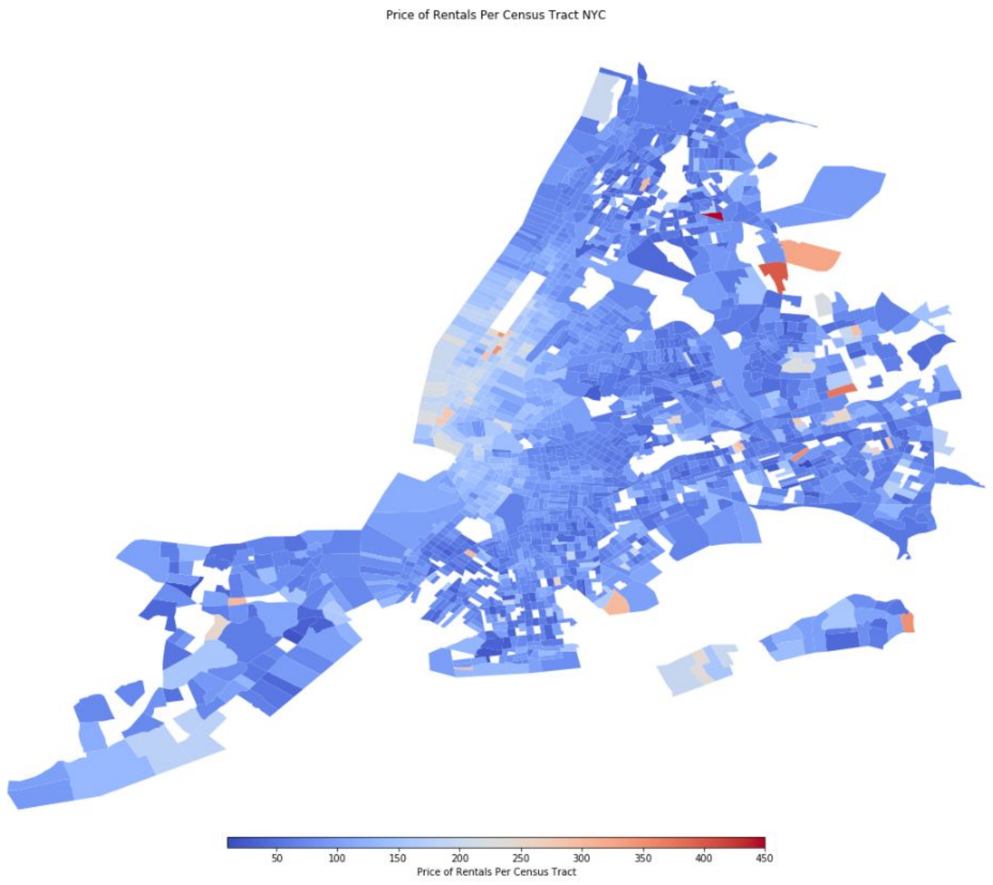
As a final step to the EDA process, geographical plots of various data points are plotted by census tract.

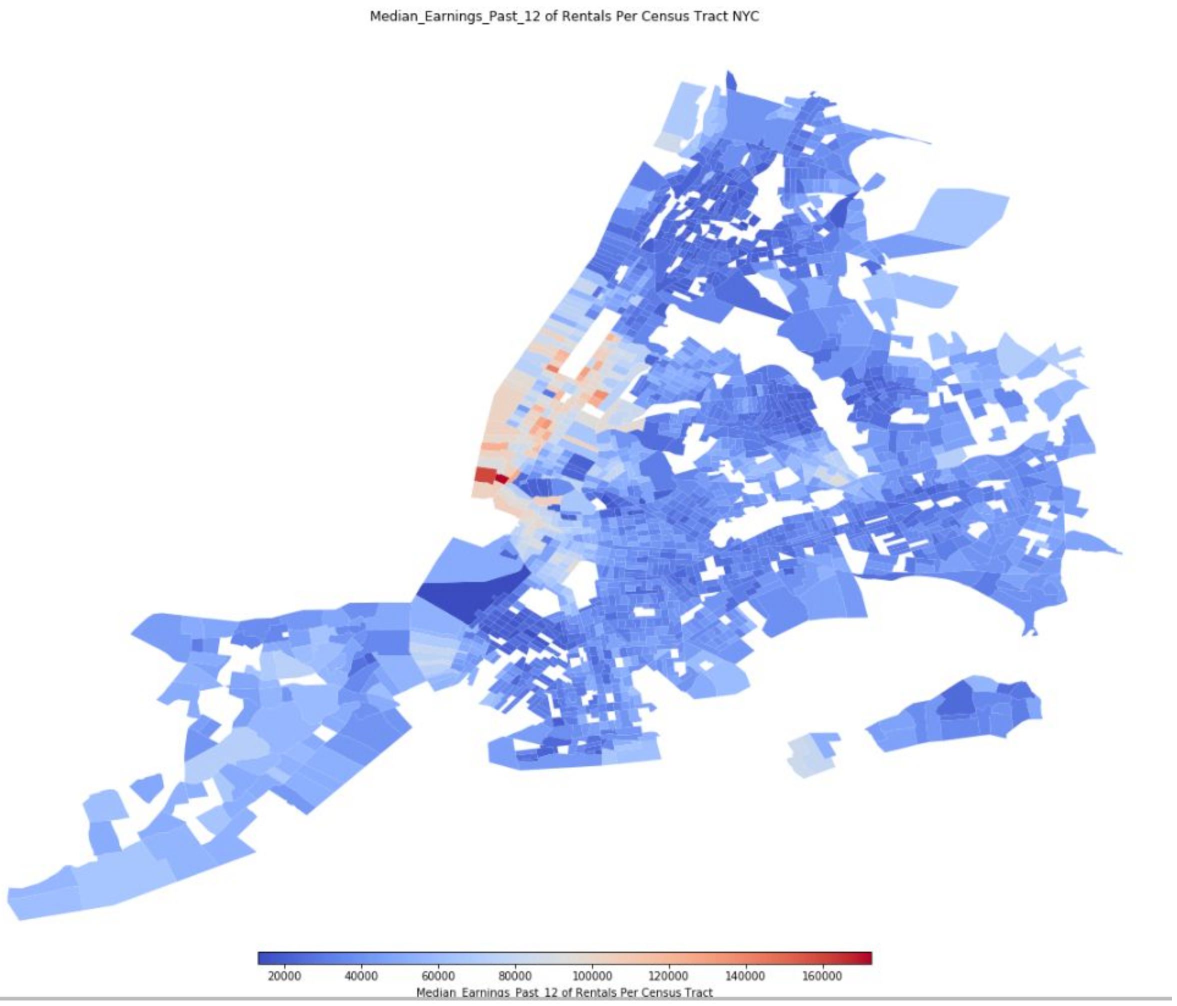
```
fig, ax = plt.subplots(1,1,figsize=(10,10))
sns.scatterplot(x=final_air_bnb['longitude'],y=final_air_bnb['latitude'],hue=final_air_bnb['neighbourhood_group'],ax=ax,alpha=0.5)
plt.axis('off');
```



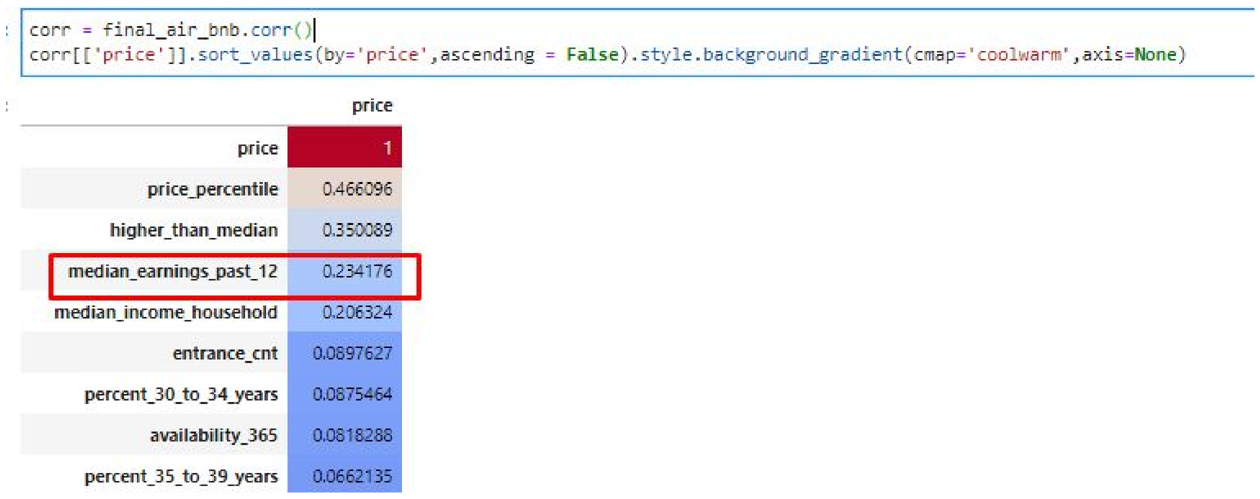
```
from matplotlib import cm
plot_list = ['count','price','median_earnings_past_12','poverty_rate']



for x in plot_list:
    fig, ax = plt.subplots(1,1,figsize=(20,20))
    mappable = cm.ScalarMappable(cmap='coolwarm')
    mappable.set_array(plot_data[plot_data[x].notna()][x])
    clrbr = plt.colorbar(mappable,orientation='horizontal', shrink=0.5,aspect=50, pad=-0.01)
    clrbr.set_label("{} of Rentals Per Census Tract".format(x.title()))
    plot_data[plot_data[x].notna()].plot(column=x,ax = ax,cmap='coolwarm')
    ax.set_title("{} of Rentals Per Census Tract NYC".format(x.title()))
    plt.axis('off')
```


Count of Rentals Per Census Tract NYC
50
100
150
200
250
300
Count of Rentals Per Census Tract


Price of Rentals Per Census Tract NYC
50
100
150
200
250
300
350
400
450
Price of Rentals Per Census Tract


Median_Earnings_Past_12 of Rentals Per Census Tract NYC
20000
40000
60000
80000
100000
120000
140000
160000
Median_Earnings_Past_12 of Rentals Per Census Tract

Correlation analysis and the plots above indicates that the numerical feature with the highest correlation to price is median earnings in the past 12 months.

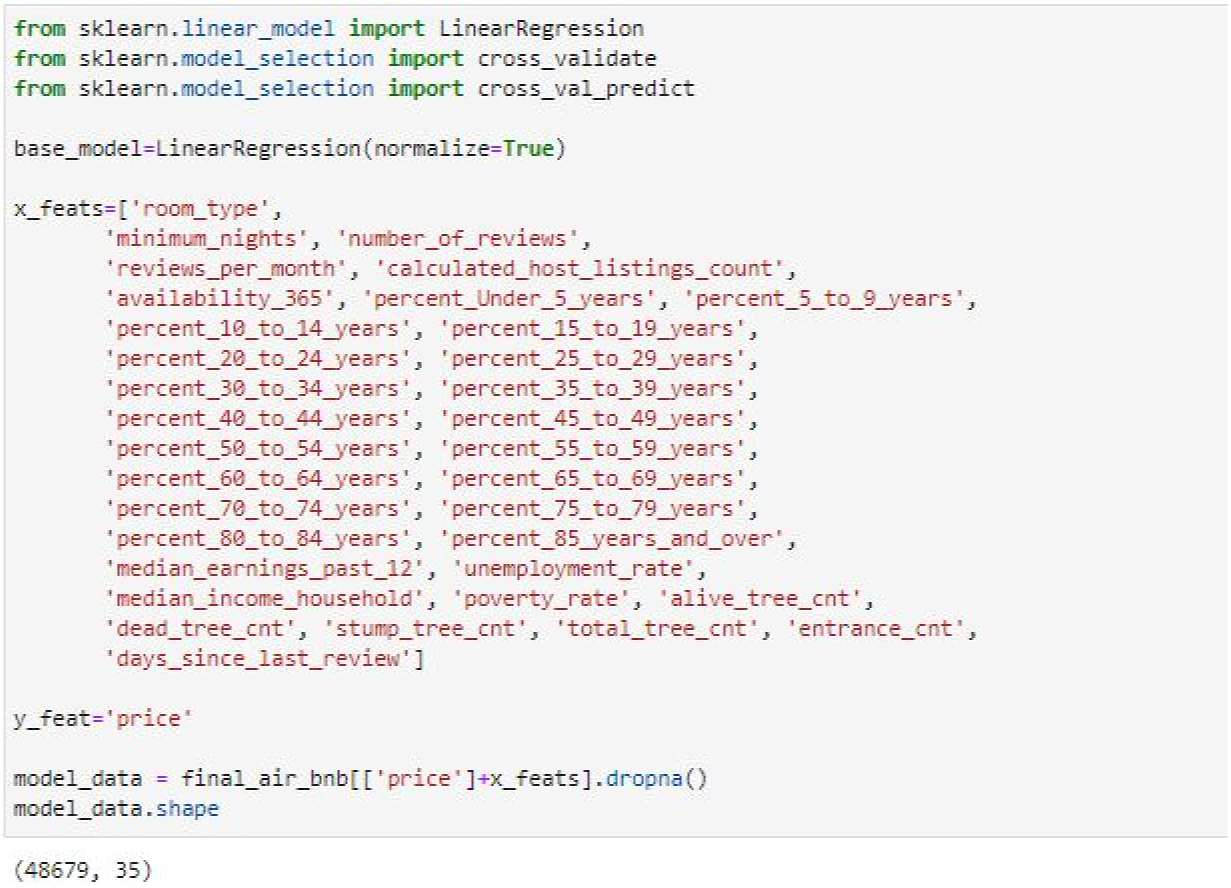


```
corr = final_air_bnb.corr()
corr[['price']].sort_values(by='price',ascending = False).style.background_gradient(cmap='coolwarm',axis=None)
```

	price
price	1
price_percentile	0.466096
higher_than_median	0.350089
median_earnings_past_12	0.234176
median_income_household	0.206324
entrance_cnt	0.0897627
percent_30_to_34_years	0.0875464
availability_365	0.0818288
percent_35_to_39_years	0.0662135

Ordinary Least Squares Regression

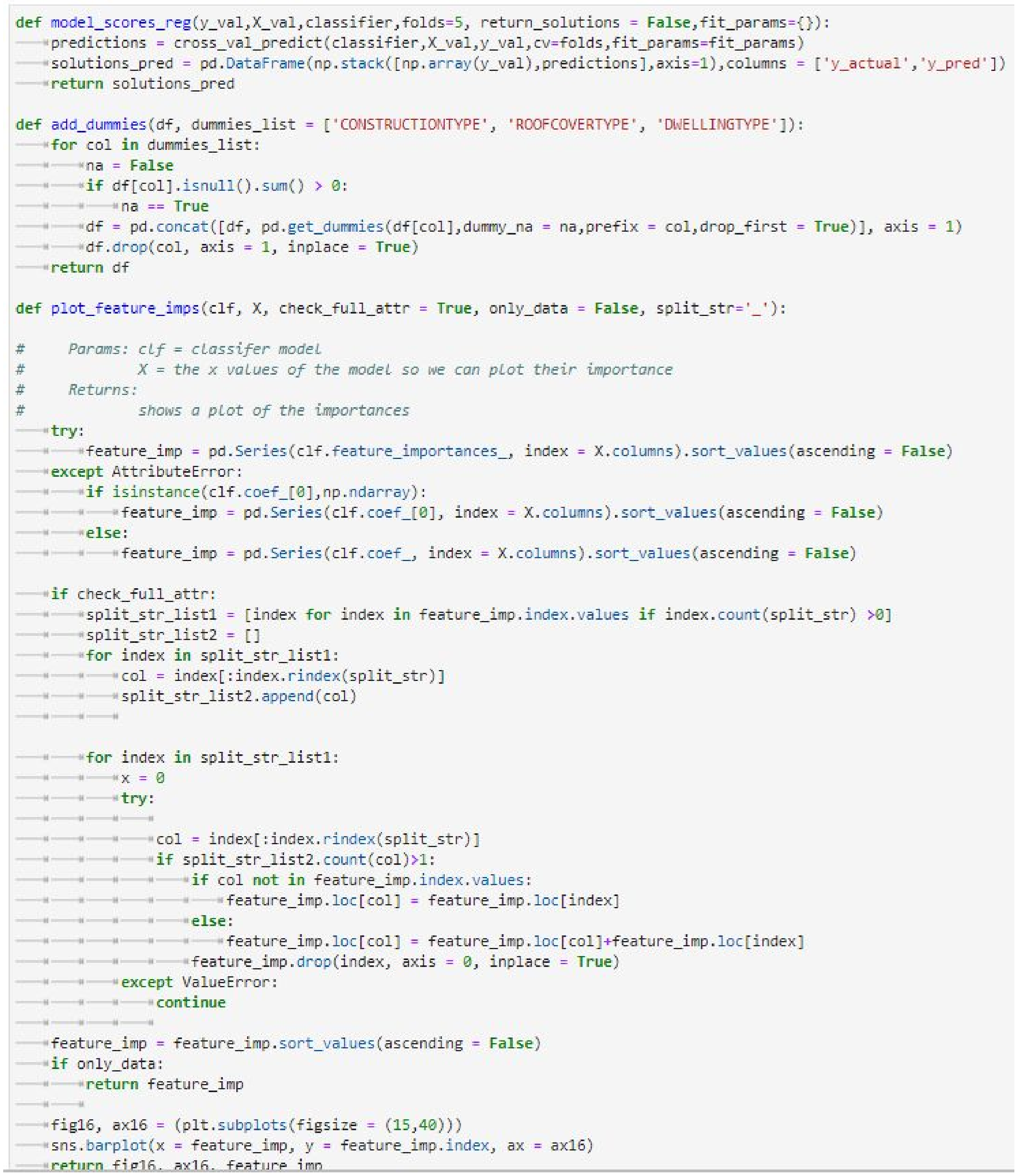
The first step in the modelling phase of this analysis is to establish a baseline performance with OLS regression. The main goal of the base model is minimizing the mean squared error while being aware of the assumptions of this regression method. The assumptions of OLS Regression are homoscedasticity of variance of residuals, error of residuals are linearly independent(random), errors are normally distributed and little no multicollinearity among predictor variables(Tuffrey, 2011). The sci-kit learn library will be used to create each model. The columns neighbourhood_group and neighbourhood are not used for the model. The census data has census tract level data for each observation so these fields are not necessary. It would also be easier for the model to generalize to other cities without specific neighborhoods and groups in our predictor variables. The neighbourhood field also contains many unique values which, when one hot encoded would add a lot of extra variables creating a higher dimensional dataset that would make our models more difficult to interpret.



```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_predict

base_model=LinearRegression(normalize=True)

x_feats=['room_type',
        'minimum_nights', 'number_of_reviews',
        'reviews_per_month', 'calculated_host_listings_count',
        'availability_365', 'percent_Under_5_years', 'percent_5_to_9_years',
        'percent_10_to_14_years', 'percent_15_to_19_years',
        'percent_20_to_24_years', 'percent_25_to_29_years',
        'percent_30_to_34_years', 'percent_35_to_39_years',
        'percent_40_to_44_years', 'percent_45_to_49_years',
        'percent_50_to_54_years', 'percent_55_to_59_years',
        'percent_60_to_64_years', 'percent_65_to_69_years',
        'percent_70_to_74_years', 'percent_75_to_79_years',
        'percent_80_to_84_years', 'percent_85_years_and_over',
        'median_earnings_past_12', 'unemployment_rate',
        'median_income_household', 'poverty_rate', 'alive_tree_cnt',
        'dead_tree_cnt', 'stump_tree_cnt', 'total_tree_cnt', 'entrance_cnt',
        'days_since_last_review']

y_feat='price'

model_data = final_air_bnb[['price']+x_feats].dropna()
model_data.shape
```

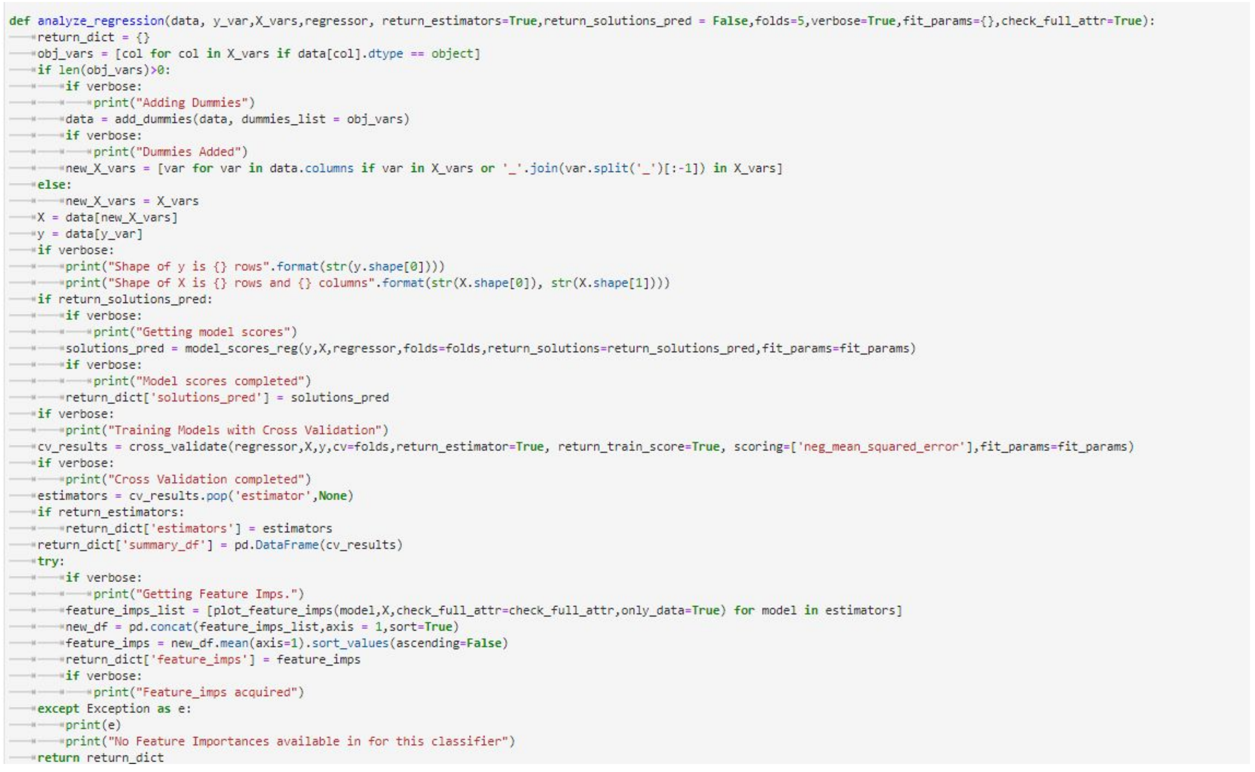
```
(48679, 35)
```

There are a total of 35 variables. There are 34 predictor variables and 1 target variable(price). 5-fold cross validation will be used to assess performance. Cross validation is a technique that uses the entire data set for training and for validation. It is one of the most widely used methods for estimating prediction error(Hastie, Tibshirani & Friedman, 2009). The benefits of this technique is that the whole dataset can be utilized for the model. The disadvantages of this technique is that it can be resource intensive with large datasets. There are few functions that have been created to handle the cross validation and model analysis steps.



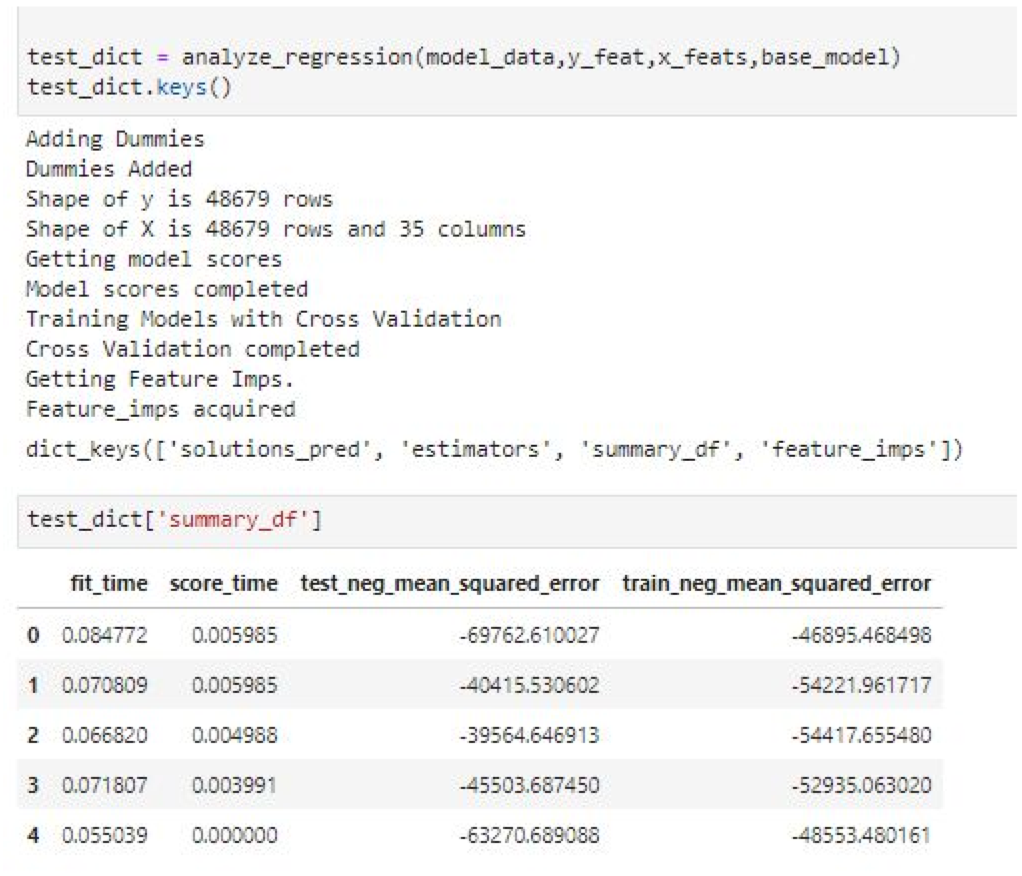
```python
def model_scores_reg(y_val,X_val,classifier,folds=5, return_solutions = False,fit_params={}):
    predictions = cross_val_predict(classifier,X_val,y_val,cv=folds,fit_params=fit_params)
    solutions_pred = pd.DataFrame(np.stack([np.array(y_val),predictions],axis=1),columns = ['y_actual','y_pred'])
    return solutions_pred

def add_dummies(df, dummies_list = ['CONSTRUCTIONTYPE', 'ROOFCOVERTYPE', 'DWELLINGTYPE']):
    for col in dummies_list:
        na = False
        if df[col].isnull().sum() > 0:
            na == True
        df = pd.concat([df, pd.get_dummies(df[col],dummy_na = na,prefix = col,drop_first = True)], axis = 1)
        df.drop(col, axis = 1, inplace = True)
    return df

def plot_feature_imps(clf, X, check_full_attr = True, only_data = False, split_str='_'):

#     Params: clf = classifer model
#             X = the x values of the model so we can plot their importance
#     Returns:
#             shows a plot of the importances
    try:
        feature_imp = pd.Series(clf.feature_importances_, index = X.columns).sort_values(ascending = False)
    except AttributeError:
        if isinstance(clf.coef_[0],np.ndarray):
            feature_imp = pd.Series(clf.coef_[0], index = X.columns).sort_values(ascending = False)
        else:
            feature_imp = pd.Series(clf.coef_, index = X.columns).sort_values(ascending = False)

    if check_full_attr:
        split_str_list1 = [index for index in feature_imp.index.values if index.count(split_str) >0]
        split_str_list2 = []
        for index in split_str_list1:
            col = index[:index.rindex(split_str)]
            split_str_list2.append(col)


        for index in split_str_list1:
            x = 0
            try:

                col = index[:index.rindex(split_str)]
                if split_str_list2.count(col)>1:
                    if col not in feature_imp.index.values:
                        feature_imp.loc[col] = feature_imp.loc[index]
                    else:
                        feature_imp.loc[col] = feature_imp.loc[col]+feature_imp.loc[index]
                    feature_imp.drop(index, axis = 0, inplace = True)
            except ValueError:
                continue

    feature_imp = feature_imp.sort_values(ascending = False)
    if only_data:
        return feature_imp

    fig16, ax16 = (plt.subplots(figsize = (15,40)))
    sns.barplot(x = feature_imp, y = feature_imp.index, ax = ax16)
    return fig16, ax16, feature_imp
```

The model_scores_reg function performs cross validation and performs a prediction on every observation. It returns the predicted value and the actual value for each observation ran through the model. The add_dummies function adds one-hot encoded dummy variables to a dataframe and drops the original variable using the pandas get_dummies method. The plot_feature_imps will either create a bar plot of the feature importances or return the feature importances in a dataframe. The plot_feature_imps will either use the feature importances from a decision tree based model or the coefficients created in a linear model. If neither are available it will not work. The last function, analyze_regression is below. This function combines the first three functions to perform cross validation on a dataset given the data, the y values, the x values and the sci-kit learn regressor. When passed those four pieces of information it will return the models created by cross validation, the feature importances, the predictions and original y values of the model and a summary of the scores.



```python
def analyze_regression(data, y_var,X_vars,regressor, return_estimators=True,return_solutions_pred = False,folds=5,verbose=True,fit_params={},check_full_attr=True):
    return_dict = {}
    obj_vars = [col for col in X_vars if data[col].dtype == object]
    if len(obj_vars)>0:
        if verbose:
            print("Adding Dummies")
        data = add_dummies(data, dummies_list = obj_vars)
        if verbose:
            print("Dummies Added")
        new_X_vars = [var for var in data.columns if var in X_vars or '_'.join(var.split('_')[:-1]) in X_vars]
    else:
        new_X_vars = X_vars
    X = data[new_X_vars]
    y = data[y_var]
    if verbose:
        print("Shape of y is {} rows".format(str(y.shape[0])))
        print("Shape of X is {} rows and {} columns".format(str(X.shape[0]), str(X.shape[1])))
    if return_solutions_pred:
        if verbose:
            print("Getting model scores")
        solutions_pred = model_scores_reg(y,X,regressor,folds=folds,return_solutions=return_solutions_pred,fit_params=fit_params)
        if verbose:
            print("Model scores completed")
        return_dict['solutions_pred'] = solutions_pred
    if verbose:
        print("Training Models with Cross Validation")
    cv_results = cross_validate(regressor,X,y,cv=folds,return_estimator=True, return_train_score=True, scoring=['neg_mean_squared_error'],fit_params=fit_params)
    if verbose:
        print("Cross Validation completed")
    estimators = cv_results.pop('estimator',None)
    if return_estimators:
        return_dict['estimators'] = estimators
    return_dict['summary_df'] = pd.DataFrame(cv_results)
    try:
        if verbose:
            print("Getting Feature Imps.")
        feature_imps_list = [plot_feature_imps(model,X,check_full_attr=check_full_attr,only_data=True) for model in estimators]
        new_df = pd.concat(feature_imps_list,axis = 1,sort=True)
        feature_imps = new_df.mean(axis=1).sort_values(ascending=False)
        return_dict['feature_imps'] = feature_imps
        if verbose:
            print("Feature_imps acquired")
    except Exception as e:
        print(e)
        print("No Feature Importances available in for this classifier")
    return return_dict
```

The first iteration of cross validation is below:



```
test_dict = analyze_regression(model_data,y_feat,x_feats,base_model)
test_dict.keys()
```

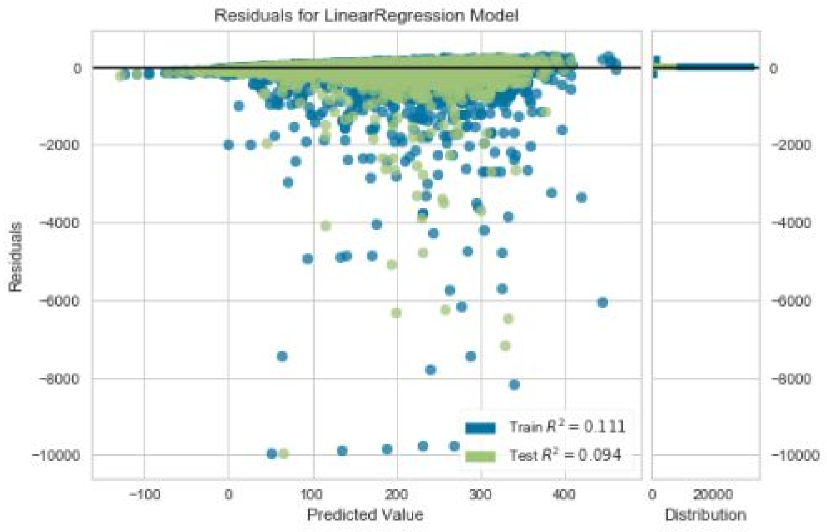
```
Adding Dummies
Dummies Added
Shape of y is 48679 rows
Shape of X is 48679 rows and 35 columns
Getting model scores
Model scores completed
Training Models with Cross Validation
Cross Validation completed
Getting Feature Imps.
Feature_imps acquired
dict_keys(['solutions_pred', 'estimators', 'summary_df', 'feature_imps'])
```

```
test_dict['summary_df']
```

	fit_time	score_time	test_neg_mean_squared_error	train_neg_mean_squared_error
0	0.084772	0.005985	-69762.610027	-46895.468498
1	0.070809	0.005985	-40415.530602	-54221.961717
2	0.066820	0.004988	-39564.646913	-54417.655480
3	0.071807	0.003991	-45503.687450	-52935.063020
4	0.055039	0.000000	-63270.689088	-48553.480161

There is high variance in our mean squared error. The model appears to be unstable. Assumptions will be checked and the model will be tweaked so it performs more consistently across each fold. The Python library, yellowbrick, is used to examine the residual plots to determine next steps.

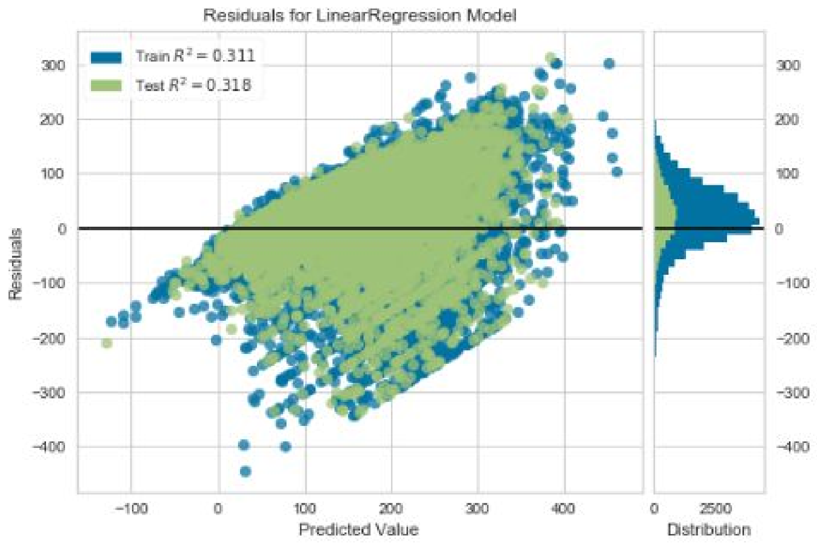
```
from sklearn.model_selection import train_test_split
from yellowbrick.regressor import ResidualsPlot

model_data = add_dummies(model_data,['room_type'])
X=model_data[model_data.columns[1:]]
y=model_data['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

visualizer = ResidualsPlot(base_model)
visualizer.fit(X_train,y_train)
visualizer.score(X_test,y_test)
visualizer.show()
```



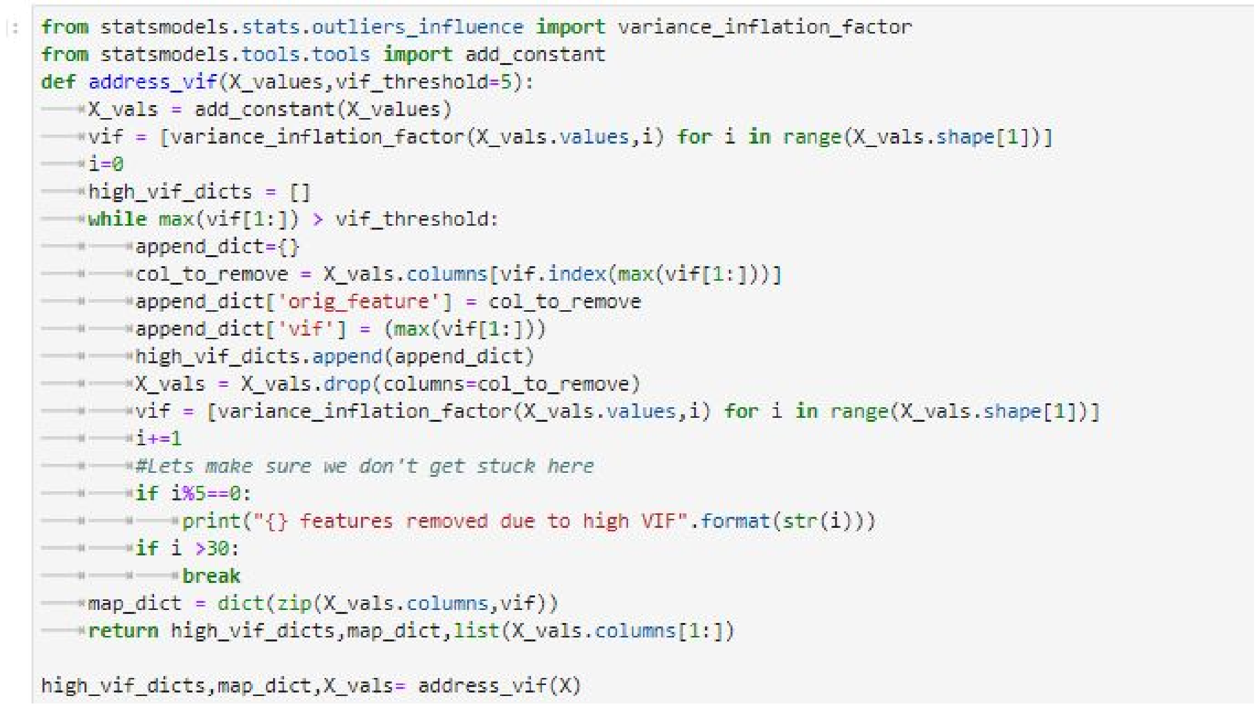
There are outliers in our model that could be preventing the model from making accurate predictions. The assumption of equal variance among the residuals is not holding. Outliers on price are removed.

```
model_data['price_percentile'] = model_data['price'].rank()/model_data.shape[0]
new_model_data = model_data[model_data['price_percentile']<=.975].reset_index(drop=True)
```

```
X=new_model_data[new_model_data.columns[1:-1]]
y=new_model_data['price']
def plot_residuals(X,y,model):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    visualizer = ResidualsPlot(model)
    visualizer.fit(X_train,y_train)
    visualizer.score(X_test,y_test)
    visualizer.show()

plot_residuals(X,y,base_model)
```

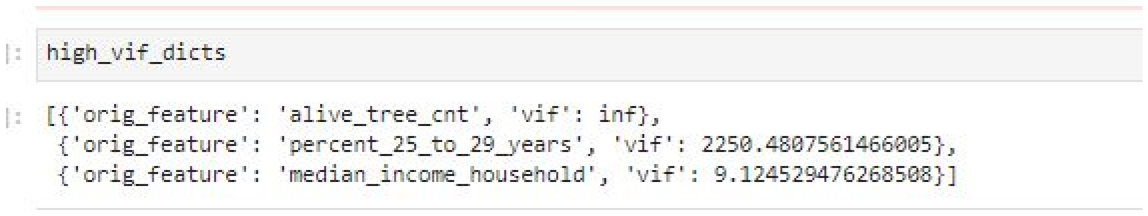


This residual plot looks more reasonable. The normality of the residuals assumption looks like it is holding, however variance does not look constant and the errors do not seem random. The following will be done to attempt to address the violation those two assumptions. Review and address multicollinearity in the predictors using Variance Inflation Factors. Attempt transformations on the target variable. The created function address_vif will be used to remove features that have a VIF>5.



```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
def address_vif(X_values,vif_threshold=5):
    X_vals = add_constant(X_values)
    vif = [variance_inflation_factor(X_vals.values,i) for i in range(X_vals.shape[1])]
    i=0
    high_vif_dicts = []
    while max(vif[1:]) > vif_threshold:
        append_dict={}
        col_to_remove = X_vals.columns[vif.index(max(vif[1:]))]
        append_dict['orig_feature'] = col_to_remove
        append_dict['vif'] = (max(vif[1:]))
        high_vif_dicts.append(append_dict)
        X_vals = X_vals.drop(columns=col_to_remove)
        vif = [variance_inflation_factor(X_vals.values,i) for i in range(X_vals.shape[1])]
        i+=1
        #Lets make sure we don't get stuck here
        if i%5==0:
            print("{} features removed due to high VIF".format(str(i)))
        if i >30:
            break
    map_dict = dict(zip(X_vals.columns,vif))
    return high_vif_dicts,map_dict,list(X_vals.columns[1:])

high_vif_dicts,map_dict,X_vals= address_vif(X)
```

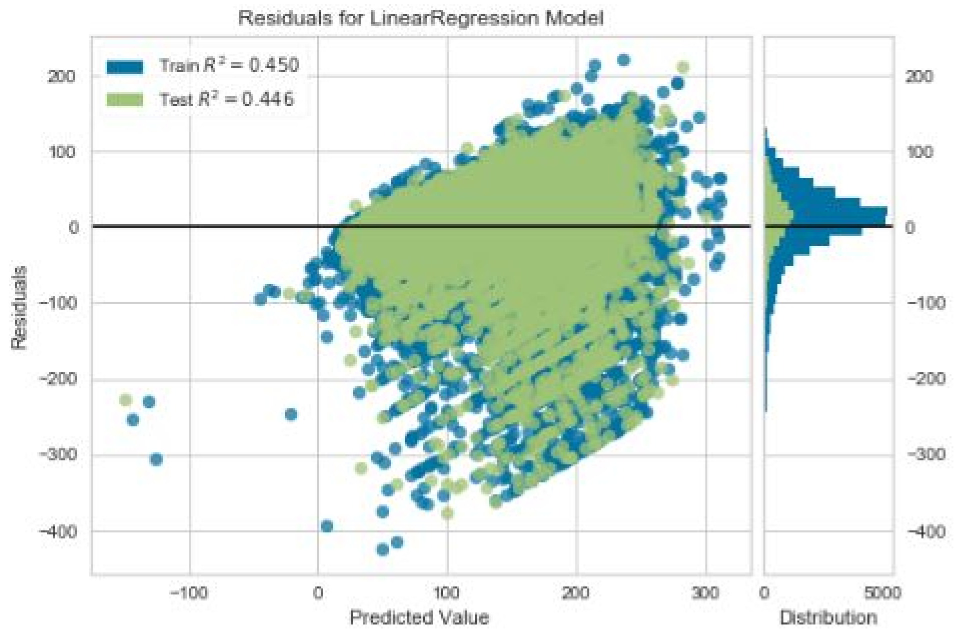
Three features were removed due to high VIF. VIFs for remaining features are all less than 5 so we have now addressed the multicollinearity assumption. The residuals plots are reviewed again to determine fit, with and without the square root transformation of price.



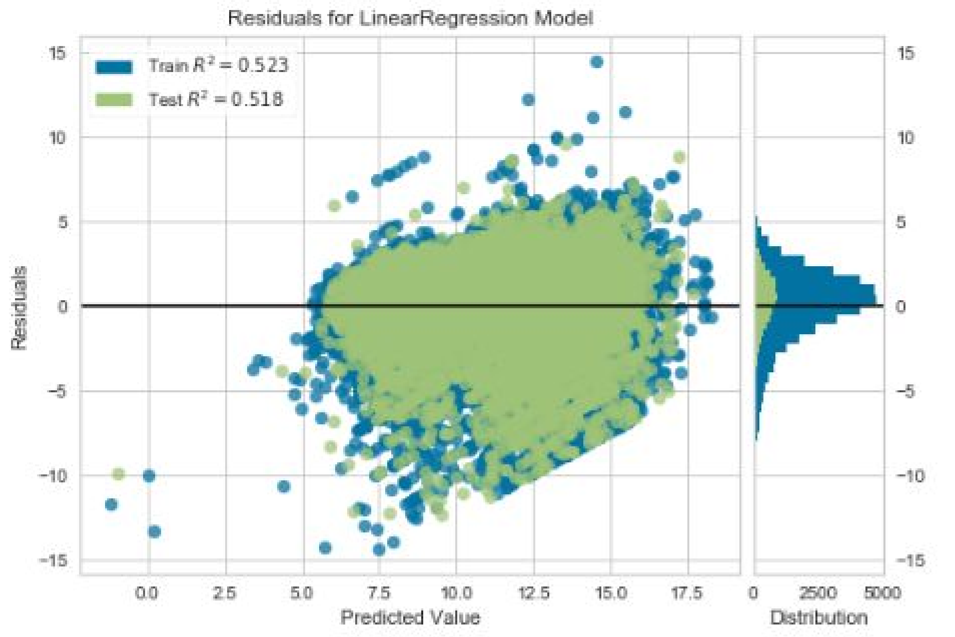
```
high_vif_dicts
```

```
[{'orig_feature': 'alive_tree_cnt', 'vif': inf},
 {'orig_feature': 'percent_25_to_29_years', 'vif': 2250.4807561466005},
 {'orig_feature': 'median_income_household', 'vif': 9.124529476268508}]
```

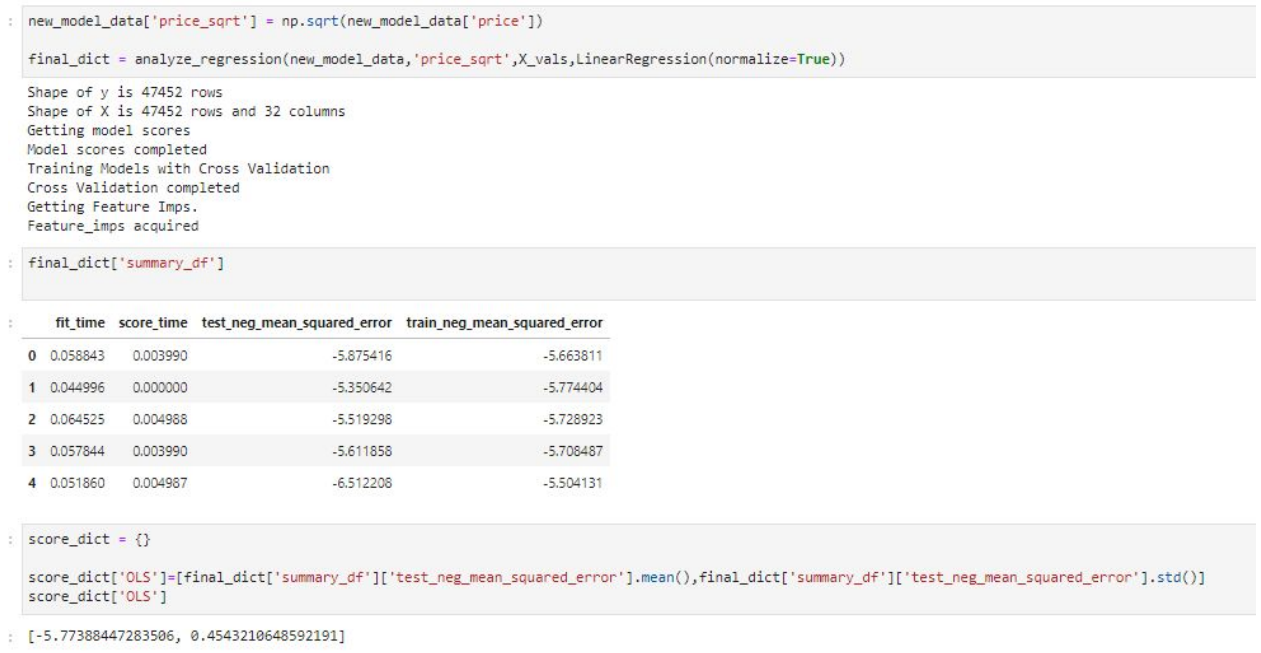
```
X=new_model_data[X_vals]
y=new_model_data['price']
plot_residuals(X,y,LinearRegression(normalize=True))
```



```
X=new_model_data[X_vals]
y=np.sqrt(new_model_data['price'])
plot_residuals(X,y,LinearRegression(normalize=True))
```



Our new residual plot still doesn't fit the assumptions perfectly but it is much closer than previously. The R Squared values are also much higher. In summary, the following was done to address assumptions and create a more robust OLS model. Outliers were removed by removing observations that had prices greater than the 97.5 percentile. Three highly intercorrelated features removed: alive_tree_count, median_income_houshold and percent_25_to_29_years. A square root transformation was done on the target variable to help handle non-linearity. The dataset is ready to be run again through cross validation to establish our base score to compare with the other models.



```
new_model_data['price_sqrt'] = np.sqrt(new_model_data['price'])

final_dict = analyze_regression(new_model_data,'price_sqrt',X_vals,LinearRegression(normalize=True))
```

```
Shape of y is 47452 rows
Shape of X is 47452 rows and 32 columns
Getting model scores
Model scores completed
Training Models with Cross Validation
Cross Validation completed
Getting Feature Imps.
Feature_imps acquired
```

```
final_dict['summary_df']
```

	fit_time	score_time	test_neg_mean_squared_error	train_neg_mean_squared_error
0	0.058843	0.003990	-5.875416	-5.663811
1	0.044996	0.000000	-5.350642	-5.774404
2	0.064525	0.004988	-5.519298	-5.728923
3	0.057844	0.003990	-5.611858	-5.708487
4	0.051860	0.004987	-6.512208	-5.504131

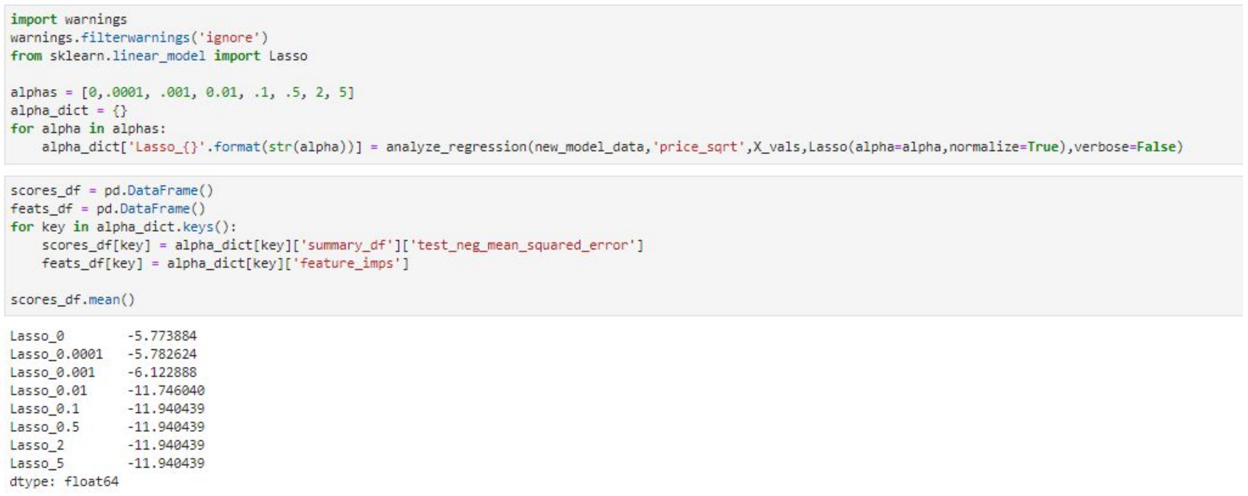
```
score_dict = {}

score_dict['OLS']=[final_dict['summary_df']['test_neg_mean_squared_error'].mean(),final_dict['summary_df']['test_neg_mean_squared_error'].std()]
score_dict['OLS']
```

```
[-5.77388447283506, 0.4543210648592191]
```

The variance of the test terms is lower and the model seems much more stable. The base model’s mean squared error is 5.77.

Lasso Regression

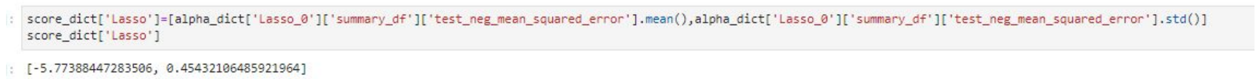
The first model we will compare our base OLS model to is Lasso regression. Lasso regression is a regression technique that applies penalty based on a constant and the beta values for the predictor variables(Tibshirani, 1996). Sci-kit learn has the lasso regression model available in the same manner it has linear regression. Lasso regression employs an alpha parameter that is used for L1 regularization. An alpha value of zero is the same model as our OLS model above. As alpha increases, coefficients will trend toward 0 and bias in the model will increase. Lasso regression is useful in dealing with overfitting.



```
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import Lasso

alphas = [0,.0001, .001, 0.01, .1, .5, 2, 5]
alpha_dict = {}
for alpha in alphas:
    alpha_dict['Lasso_{}'.format(str(alpha))] = analyze_regression(new_model_data,'price_sqrt',X_vals,Lasso(alpha=alpha,normalize=True),verbose=False)
```

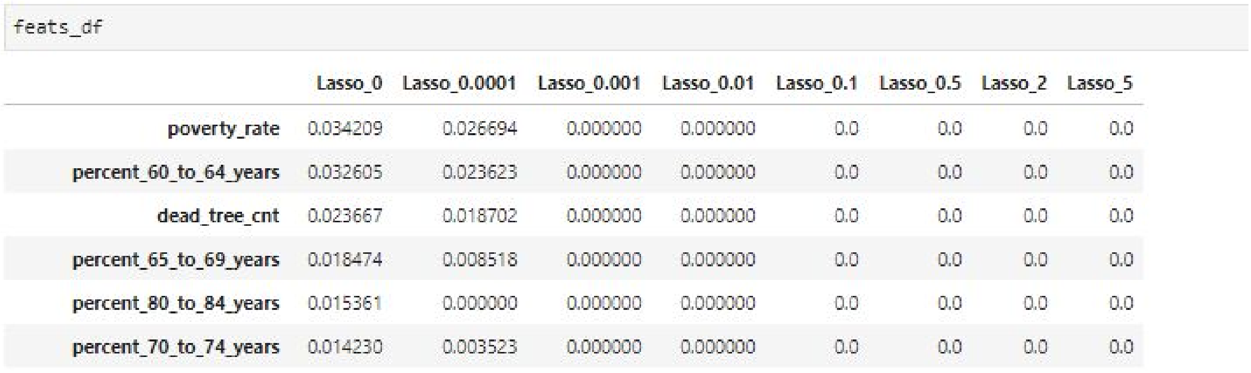
```
scores_df = pd.DataFrame()
feats_df = pd.DataFrame()
for key in alpha_dict.keys():
    scores_df[key] = alpha_dict[key]['summary_df']['test_neg_mean_squared_error']
    feats_df[key] = alpha_dict[key]['feature_imps']

scores_df.mean()
```

```
Lasso_0        -5.773884
Lasso_0.0001   -5.782624
Lasso_0.001    -6.122888
Lasso_0.01    -11.746040
Lasso_0.1     -11.940439
Lasso_0.5     -11.940439
Lasso_2       -11.940439
Lasso_5       -11.940439
dtype: float64
```



```
score_dict['Lasso']=[alpha_dict['Lasso_0']['summary_df']['test_neg_mean_squared_error'].mean(),alpha_dict['Lasso_0']['summary_df']['test_neg_mean_squared_error'].std()]
score_dict['Lasso']
```

```
[-5.77388447283506, 0.45432106485921964]
```



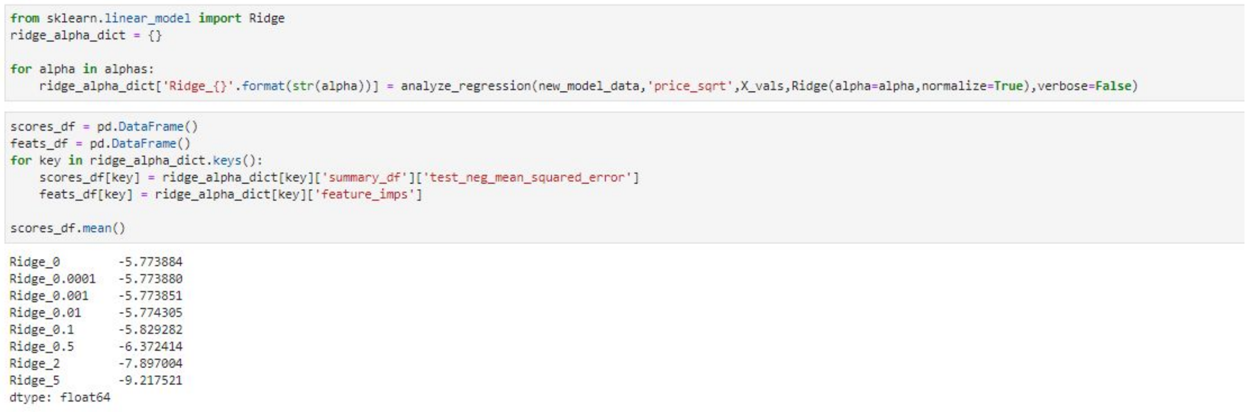
```
feats_df
```

	Lasso_0	Lasso_0.0001	Lasso_0.001	Lasso_0.01	Lasso_0.1	Lasso_0.5	Lasso_2	Lasso_5
poverty_rate	0.034209	0.026694	0.000000	0.000000	0.0	0.0	0.0	0.0
percent_60_to_64_years	0.032605	0.023623	0.000000	0.000000	0.0	0.0	0.0	0.0
dead_tree_cnt	0.023667	0.018702	0.000000	0.000000	0.0	0.0	0.0	0.0
percent_65_to_69_years	0.018474	0.008518	0.000000	0.000000	0.0	0.0	0.0	0.0
percent_80_to_84_years	0.015361	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0
percent_70_to_74_years	0.014230	0.003523	0.000000	0.000000	0.0	0.0	0.0	0.0

With lasso regression, the null hypothesis cannot be rejected. There is no improvement from Lasso regression. In fact, the higher the alpha parameter the worse the performance. From the feats_df, one can see at higher levels of alpha our coefficients trend toward 0. It also makes sense that there is little improvement as we can see from our original performance in Linear regression that train mean squared error was close to the test mean squared error. This means the model is exhibiting a high bias problem instead of a high variance problem.

Ridge Regression

Ridge regression is another technique that helps correct for multicollinearity and high variance by adding a bias parameter to improve performance(Hoerl & Kennard, 2000). This regression method serves a similar purpose as the Lasso regression method as it helps correct for overfitting. Based on the initial Linear Regression scores it does not appear that our model is overfitting and Ridge regression will not be useful. While Lasso regression adds a penalty related to a constant times the absolute value of the beta term, ridge adds a penalty related to a constant times the beta term squared.

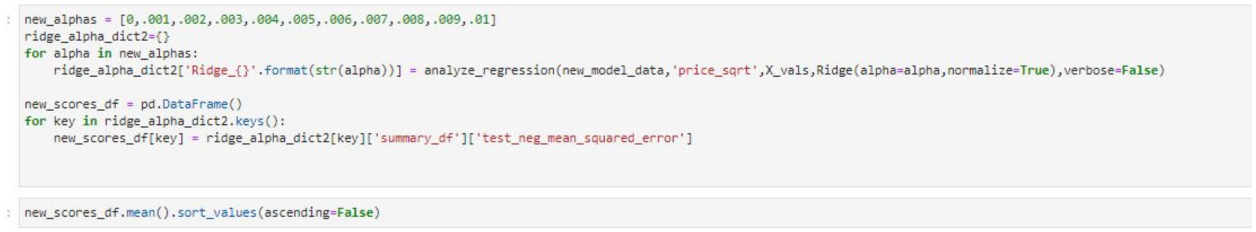


```
from sklearn.linear_model import Ridge
ridge_alpha_dict = {}

for alpha in alphas:
    ridge_alpha_dict['Ridge_{}'.format(str(alpha))] = analyze_regression(new_model_data,'price_sqrt',X_vals,Ridge(alpha=alpha,normalize=True),verbose=False)
```

```
scores_df = pd.DataFrame()
feats_df = pd.DataFrame()
for key in ridge_alpha_dict.keys():
    scores_df[key] = ridge_alpha_dict[key]['summary_df']['test_neg_mean_squared_error']
    feats_df[key] = ridge_alpha_dict[key]['feature_imps']

scores_df.mean()
```

```
Ridge_0         -5.773884
Ridge_0.0001    -5.773880
Ridge_0.001     -5.773851
Ridge_0.01      -5.774305
Ridge_0.1       -5.829282
Ridge_0.5       -6.372414
Ridge_2         -7.897004
Ridge_5         -9.217521
dtype: float64
```

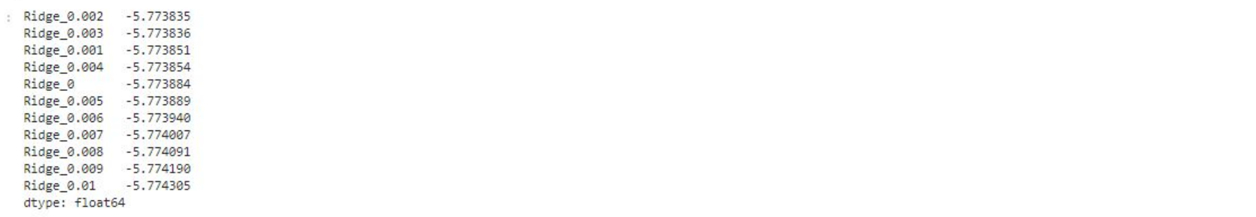
There is a slight improvement between 0 alpha and .001 alpha.

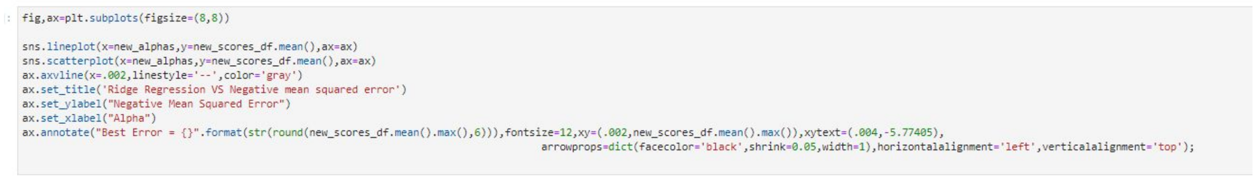


```
new_alphas = [0,.001,.002,.003,.004,.005,.006,.007,.008,.009,.01]
ridge_alpha_dict2={}
for alpha in new_alphas:
    ridge_alpha_dict2['Ridge_{}'.format(str(alpha))] = analyze_regression(new_model_data,'price_sqrt',X_vals,Ridge(alpha=alpha,normalize=True),verbose=False)

new_scores_df = pd.DataFrame()
for key in ridge_alpha_dict2.keys():
    new_scores_df[key] = ridge_alpha_dict2[key]['summary_df']['test_neg_mean_squared_error']
```

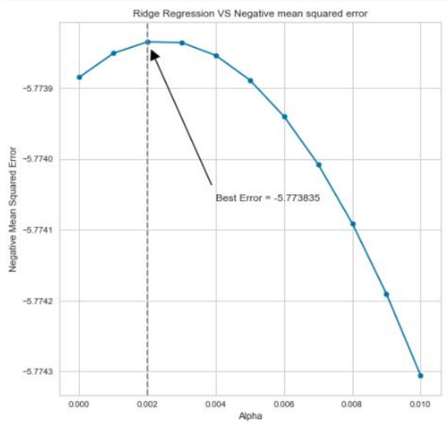
```
new_scores_df.mean().sort_values(ascending=False)
```

```
Ridge_0.002   -5.773835
Ridge_0.003   -5.773836
Ridge_0.001   -5.773851
Ridge_0.004   -5.773854
Ridge_0       -5.773884
Ridge_0.005   -5.773889
Ridge_0.006   -5.773940
Ridge_0.007   -5.774007
Ridge_0.008   -5.774091
Ridge_0.009   -5.774190
Ridge_0.01    -5.774305
dtype: float64
```





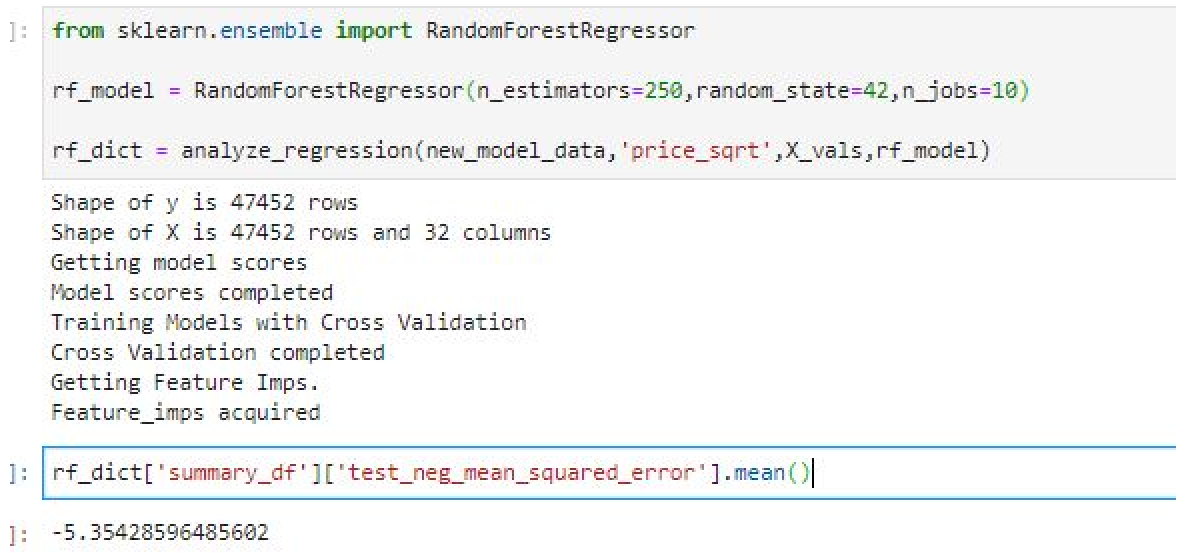
```
fig,ax=plt.subplots(figsize=(8,8))

sns.lineplot(x=new_alphas,y=new_scores_df.mean(),ax=ax)
sns.scatterplot(x=new_alphas,y=new_scores_df.mean(),ax=ax)
ax.axvline(x=.002,linestyle='--',color='gray')
ax.set_title('Ridge Regression VS Negative mean squared error')
ax.set_ylabel("Negative Mean Squared Error")
ax.set_xlabel("Alpha")
ax.annotate("Best Error = {}".format(str(round(new_scores_df.mean().max(),6))),fontsize=12,xy=(.002,new_scores_df.mean().max()),xytext=(.004,-5.77405),
                                                                                        arrowprops=dict(facecolor='black',shrink=0.05,width=1),horizontalalignment='left',verticalalignment='top');
```



With Ridge and an alpha of .002, there is a slight improvement in performance. The improvement is less than 1/1000th of a percent. The null hypothesis cannot be rejected for Ridge Regression either. This makes sense as Ridge helps control for overfitting and based on the previous metrics with Linear regression. The model does not appear to have an overfitting problem.

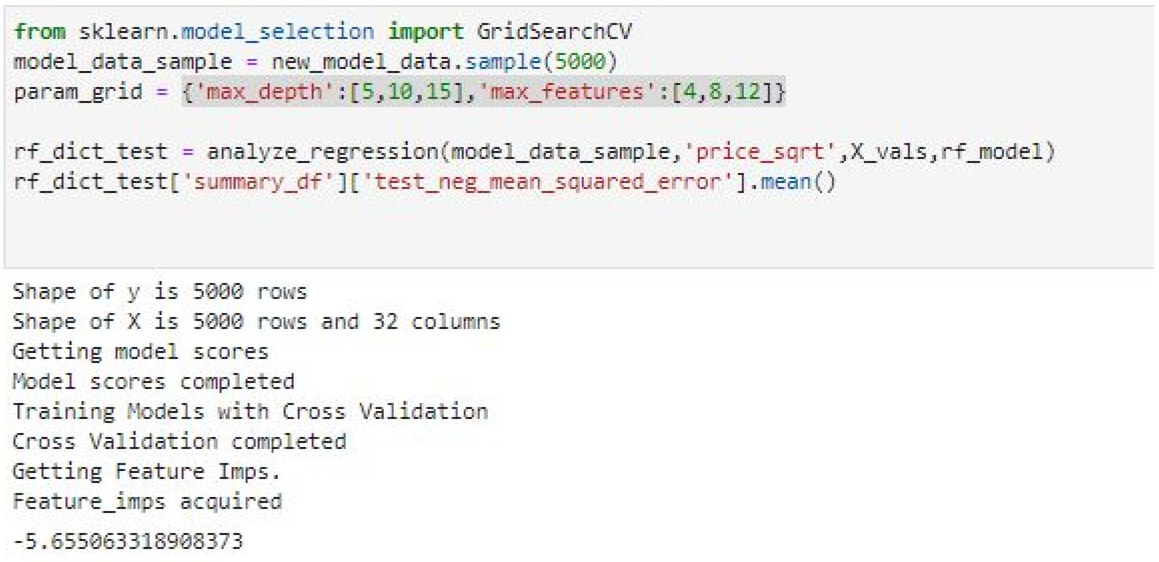
Random Forest Regression

Random forest methods have been used to generate predictions with good performance in relation to prediction error for many benchmark datasets(Segal,2004). Random forests can also learn a lot of complex non linear relationships between data without the need for complex data transformations. A random forest is an ensemble machine learning method that uses a technique called bagging. An ensemble machine learning method is one that creates a single model from multiple base models to improve performance. Bagging or Bootstrap Aggregating is a technique that generates an ensemble of models in which each model is trained on a randomly sampled subset of the training data with replacement(Sagi & Rokach 2018). The random forest algorithm is an ensemble of many decision trees using this bagging method.



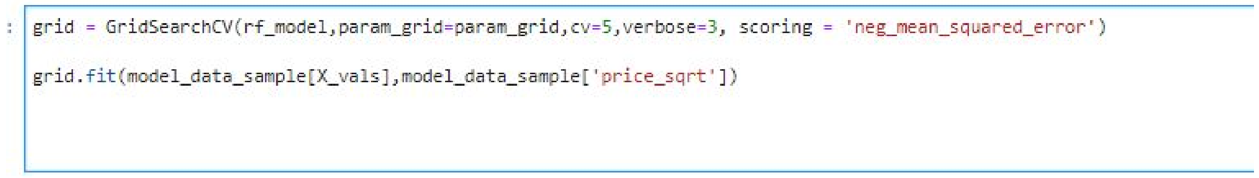
```
]: from sklearn.ensemble import RandomForestRegressor

   rf_model = RandomForestRegressor(n_estimators=250,random_state=42,n_jobs=10)

   rf_dict = analyze_regression(new_model_data,'price_sqrt',X_vals,rf_model)

   Shape of y is 47452 rows
   Shape of X is 47452 rows and 32 columns
   Getting model scores
   Model scores completed
   Training Models with Cross Validation
   Cross Validation completed
   Getting Feature Imps.
   Feature_imps acquired

]: rf_dict['summary_df']['test_neg_mean_squared_error'].mean()

]: -5.35428596485602
```

Performance looks promising but it is not quite at the 10% level. Grid search will be utilized to pick the best parameters. Grid search is a technique that is used to search through all combinations of parameters specified. It is useful in optimization algorithms to have the best parameters, however it can be resource intensive and the parameters to search through have to be manually chosen. In order to improve the speed of the grid search, a subset of the original dataset is used instead of the full set of records. The performance of our model on the subset of data is 5.655. This number will be used as a baseline for the performance of the models in the grid search.



```
from sklearn.model_selection import GridSearchCV
model_data_sample = new_model_data.sample(5000)
param_grid = {'max_depth':[5,10,15],'max_features':[4,8,12]}

rf_dict_test = analyze_regression(model_data_sample,'price_sqrt',X_vals,rf_model)
rf_dict_test['summary_df']['test_neg_mean_squared_error'].mean()
```

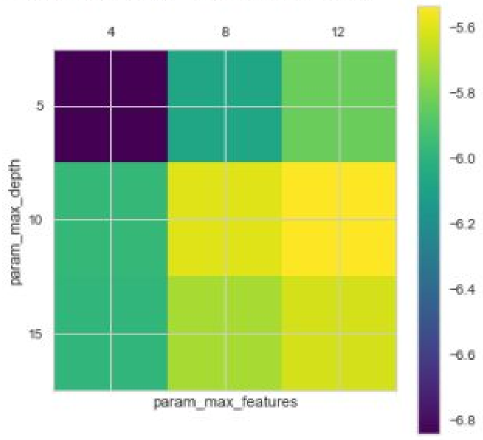
```
Shape of y is 5000 rows
Shape of X is 5000 rows and 32 columns
Getting model scores
Model scores completed
Training Models with Cross Validation
Cross Validation completed
Getting Feature Imps.
Feature_imps acquired
-5.655063318908373
```



```
grid = GridSearchCV(rf_model,param_grid=param_grid,cv=5,verbose=3, scoring = 'neg_mean_squared_error')

grid.fit(model_data_sample[X_vals],model_data_sample['price_sqrt'])
```

```
def plot_grid(data, values='mean_test_score',index = 'param_max_depth',columns='param_max_features'):
    results = pd.DataFrame(data)
    res_piv = pd.pivot_table(
        results, values=values, index=index,
        columns=columns)

    plt.matshow(res_piv.values,cmap='viridis')
    plt.xlabel(res_piv.columns.name)
    plt.xticks(range(res_piv.shape[1]), res_piv.columns)
    plt.ylabel(res_piv.index.name)
    plt.yticks(range(res_piv.shape[0]), res_piv.index);
    plt.colorbar()

plot_grid(grid.cv_results_)
print(grid.best_params_)
```

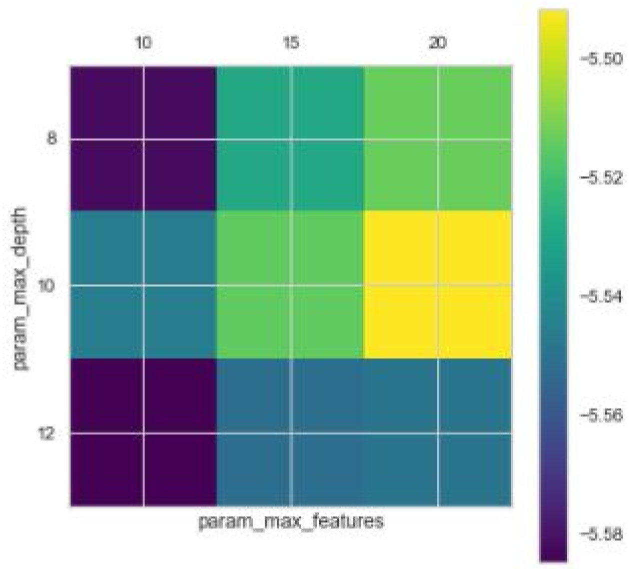
```
{'max_depth': 10, 'max_features': 12}
```



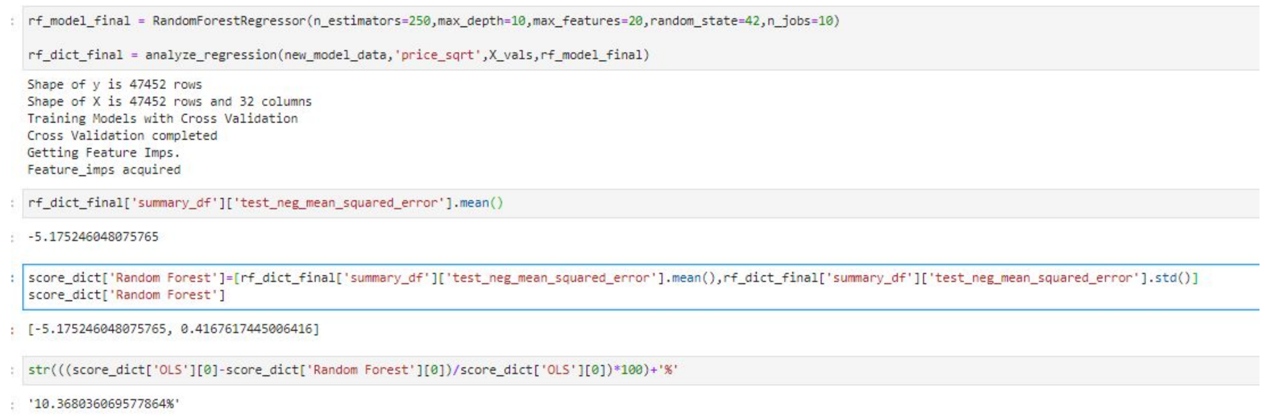
At a max_depth of 10, the model performs better as max features increase. Another grid search will be attempted closer the max depth of 10.

```
param_grid_new = {'max_depth':[8,10,12],'max_features':[10,15,20]}

grid2 = GridSearchCV(rf_model,param_grid=param_grid_new,cv=5,verbose=3, scoring = 'neg_mean_squared_error')

grid2.fit(model_data_sample[X_vals],model_data_sample['price_sqrt'])
```

```
34]:  plot_grid(grid2.cv_results_)
      print(grid2.best_params_, grid2.best_score_)

      {'max_depth': 10, 'max_features': 20} -5.491667811235959
```



The best parameters for the second grid search are max_depth of 10 and a max_features of 20. This represents a 3% improvement to the base model created earlier. This should be enough to run on the original random forest model.

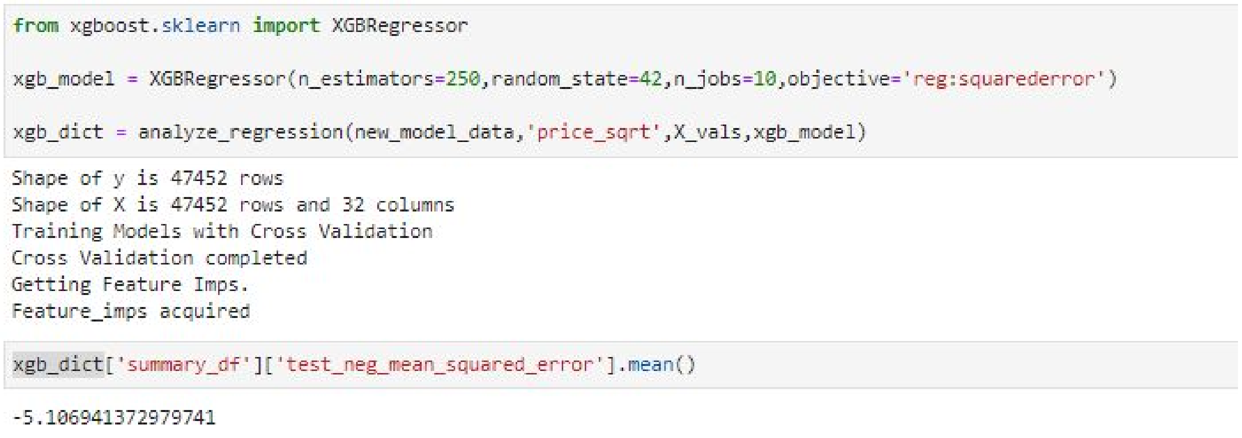


```
: rf_model_final = RandomForestRegressor(n_estimators=250,max_depth=10,max_features=20,random_state=42,n_jobs=10)

  rf_dict_final = analyze_regression(new_model_data,'price_sqrt',X_vals,rf_model_final)

  Shape of y is 47452 rows
  Shape of X is 47452 rows and 32 columns
  Training Models with Cross Validation
  Cross Validation completed
  Getting Feature Imps.
  Feature_imps acquired

: rf_dict_final['summary_df']['test_neg_mean_squared_error'].mean()

: -5.175246048075765

: score_dict['Random Forest']=[rf_dict_final['summary_df']['test_neg_mean_squared_error'].mean(),rf_dict_final['summary_df']['test_neg_mean_squared_error'].std()]
  score_dict['Random Forest']

: [-5.175246048075765, 0.4167617445006416]

: str(((score_dict['OLS'][0]-score_dict['Random Forest'][0])/score_dict['OLS'][0])*100)+'%'

: '10.368036069577864%'
```

With the Random Forest model, there is a 10.4% increase in performance over the OLS model. In order to achieve this level performance, two grid searches over the max depth and max features parameters were completed to choose optimal parameters.

XGBoost Regression

The XGboost variant of gradient boosted trees regression has also performed well on benchmark datasets(Chen & Rokach, 2016). Gradient boosted trees are decision tree based models that are optimized using the gradient descent algorithm. Due to the base model being decision tree based it can also learn complex non linear relationships in the data. XGBoost is also an ensemble method but instead of bagging it employs a technique called boosting.



```
from xgboost.sklearn import XGBRegressor

xgb_model = XGBRegressor(n_estimators=250,random_state=42,n_jobs=10,objective='reg:squarederror')

xgb_dict = analyze_regression(new_model_data,'price_sqrt',X_vals,xgb_model)
```

```
Shape of y is 47452 rows
Shape of X is 47452 rows and 32 columns
Training Models with Cross Validation
Cross Validation completed
Getting Feature Imps.
Feature_imps acquired
```

```
xgb_dict['summary_df']['test_neg_mean_squared_error'].mean()
```

```
-5.106941372979741
```

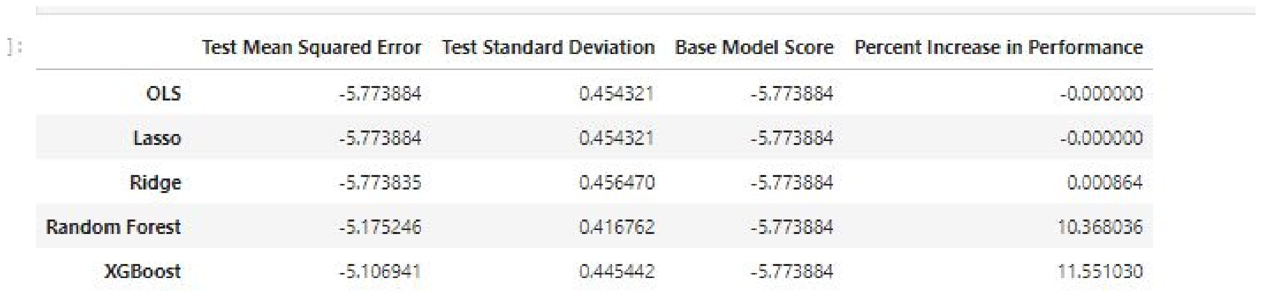
Using 250 estimators for the XGBRegressor results in an even better mean squared error than our grid searched random forest. The null hypothesis can be rejected for the XGBoost model.

Analysis: **Final Thoughts**

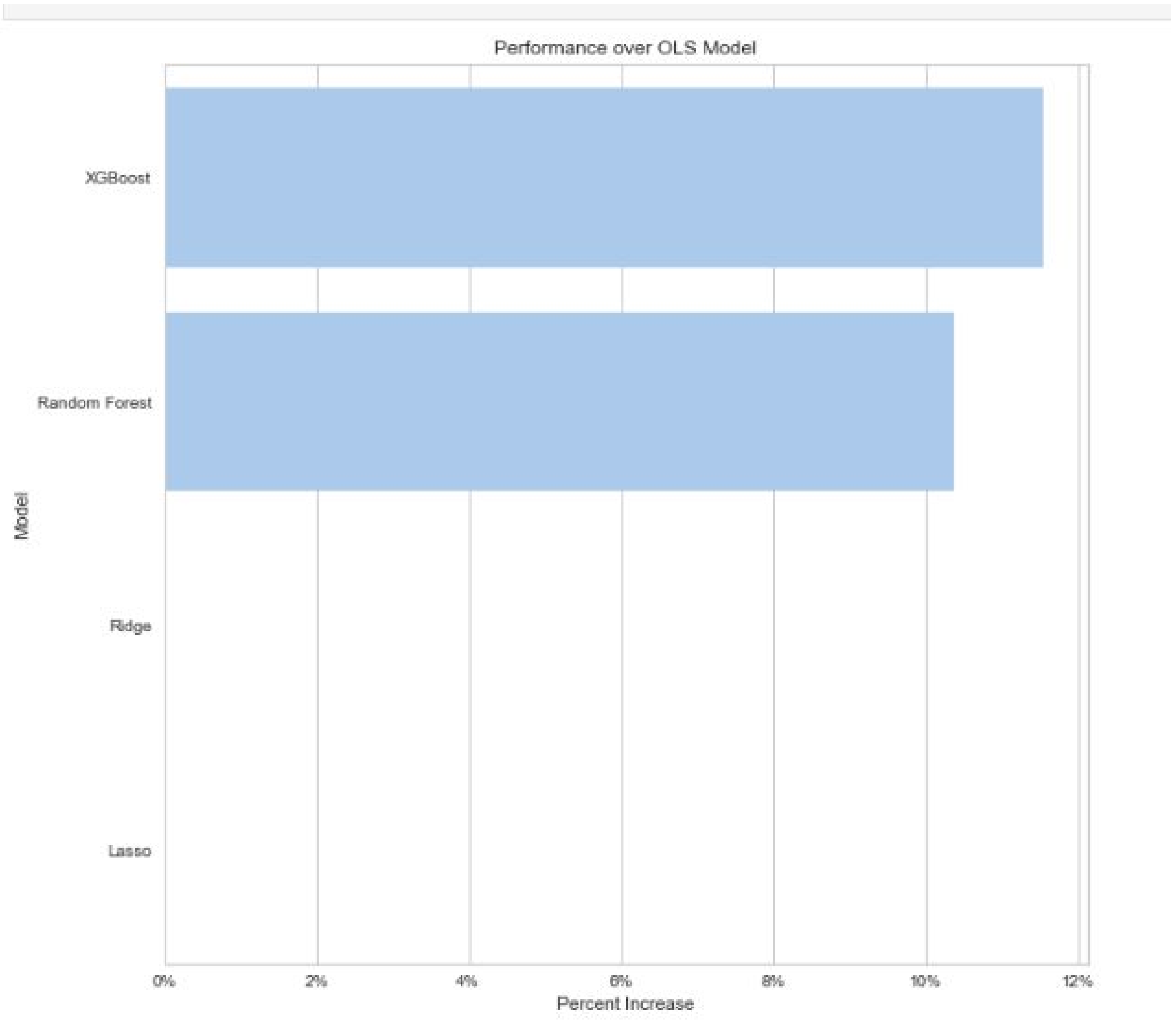
In this analysis, there were several modelling techniques that were employed. To establish the base Ordinary least squares model, outliers were removed and residual plots were analyzed to determine model fit. These are common methods of analyzing the fit of a linear model and when those methods were used in the analysis the Rsquared values increased and the model fit the data better. The disadvantage of these methods is they require manual effort and subjective judgement calls to assess model fit. For the Lasso and Ridge regression methods, the model was run through several levels of the bias parameter associated with the modelling technique. At each level of the bias parameter, performance was analyzed until the best bias parameter was chosen. The disadvantage of this technique is that the levels of the bias parameter are manually chosen. The grid search technique was employed for the Random Forest model to help choose the best parameters. Grid search is a technique that will try every combination of parameters that it is given. It does this by doing cross validation over K folds and averaging out the score. Heat maps were employed to examine the performance at each parameter until optimal parameters were found. The disadvantage of the grid search technique is that the search parameters are manually chosen and it can take quite some time to run through every combination. With XGBoost, the model was simply run with 250 estimators and all the default parameters. Using these parameters, the performance was already better than the optimized Random Forest model so nothing further was done to improve the model.

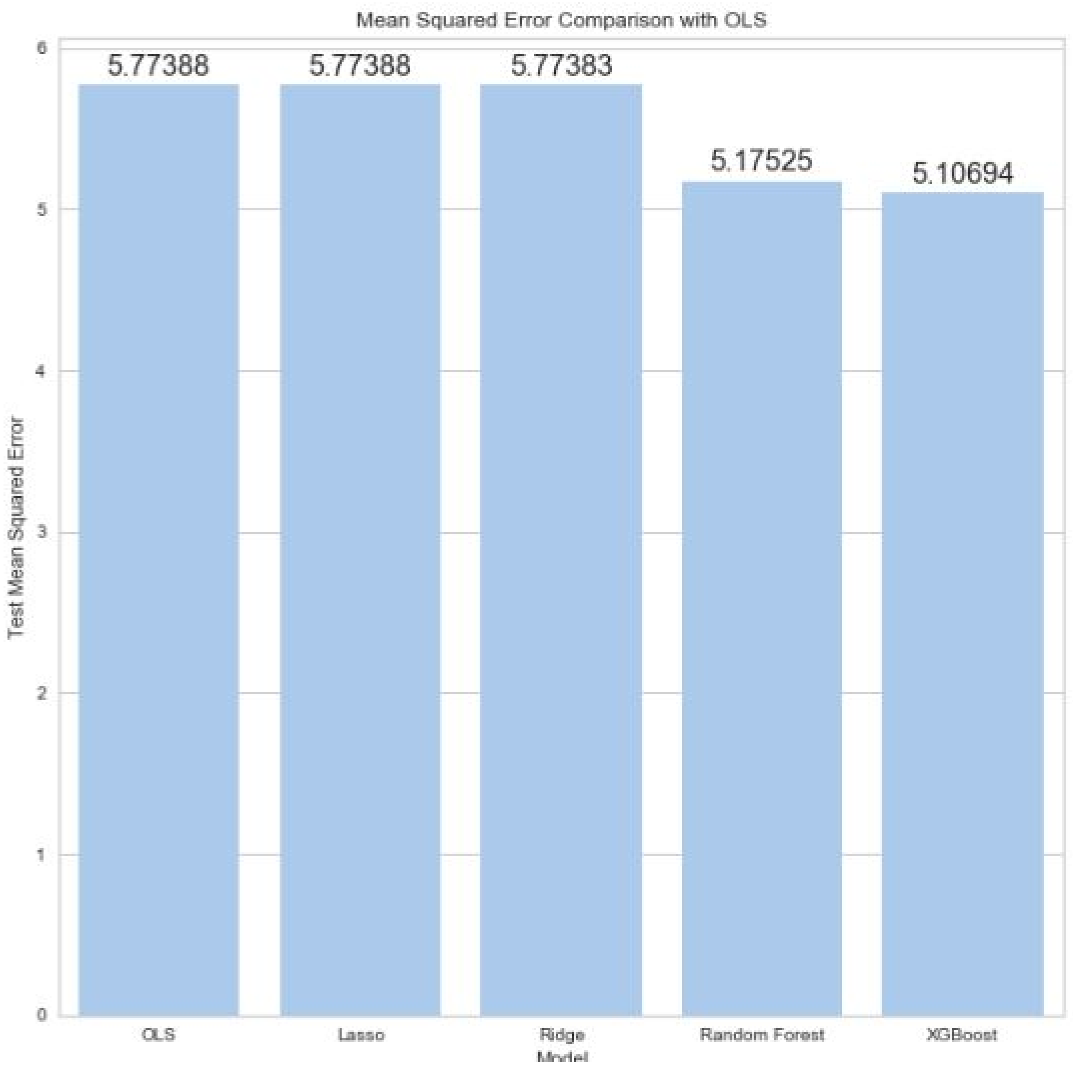
Data Summary and Implications

Overall the null hypothesis that there is not a large difference(less than 10%) in machine learning performance between penalized linear and decision tree based regression methods and ordinary least squares regression can be rejected. Both the Random Forest and XGBoost models, scored had lower mean squared error by more than 10% (10.4% and 11.6% respectively). One limitation of this result is that no interaction or higher order terms were added to the original dataset. If higher order terms and/or interactions were added and re-assessed at the linear modelling stage, the results from OLS may have performed better. Additional observations are related to the amount of effort it took to tweak each model. The linear models required the most effort as data had to be removed and manipulated to fit with assumptions and improve performance. Once that was established for OLS, it was quite simple to apply the same data to the Lasso and Ridge models. The forest based models required little data manipulation and the most that had to be done was tweak some parameters using grid search. In terms of the time it takes to train each model. The linear models trained and scored much faster than the forest based models.



	Test Mean Squared Error	Test Standard Deviation	Base Model Score	Percent Increase in Performance
OLS	-5.773884	0.454321	-5.773884	-0.000000
Lasso	-5.773884	0.454321	-5.773884	-0.000000
Ridge	-5.773835	0.456470	-5.773884	0.000864
Random Forest	-5.175246	0.416762	-5.773884	10.368036
XGBoost	-5.106941	0.445442	-5.773884	11.551030





In regards to the dataset in general, there are few courses of action that could be taken to analyze the data further. One course of action would be to acquire more census data to help better predict price. Another avenue to explore in relation to the dataset, would be to look at revenue earned per time interval instead of price. One theory is that many of the predictor variables for example, number of reviews, maybetter correlate to revenue earned versus the price of the rental. Also, the goal of renting a property is to earn revenue, not to price the rental the highest. There may be rentals in this list that are priced lower but earn substantially more revenue than higher priced rentals.

In relation to the topic studied, there are also a few more courses of action to explore. One course of action that can be recommended based on the results of the analysis is that if one is not going to add higher order terms or interactions into a linear model that they should utilize ensemble methods like Random forests or XGBoost for prediction. In review of how the linear models worked, it was noted that the Lasso and Ridge regression algorithms already handle for multicollinearity with their bias parameters. One direction of further study is adding higher order terms and interactions to the dataset and then using Lasso or Ridge regression on all the terms. The terms that stay in the model (and don't zero out) can be used in our Linear model. One comparison we can make with this direction is the effectiveness of using Lasso and Ridge regression for parameter selection versus using manual techniques like analyzing outliers, residual plots and VIF. A second approach for further study is a comparison of the performance of Linear regression models (OLS, Lasso/Ridge) and ensemble methods (Random Forest and XGBoost). The further study could compare the performance of the Linear model using higher order terms and interactions with the ensemble methods without the higher order terms. The Linear model can use the Lasso and Ridge models as tools for feature selection and dealing with multicollinearity.

Overall, there are a variety of regression modelling methods that one can explore. Ensemble methods like Random Forest and XGBoost handle non-linearity by default while linear methods require more data manipulation. Linear methods train and score much faster and can often be more interpretable. For any regression analysis, there is merit in exploring many modelling techniques to help determine the model that fits best with the business use case.

References

Barron, K., Kung, E., & Proserpio, D. (2019). Research: When Airbnb Listings in a City Increase, So Do Rent Prices. Harvard Business Review Digital Articles, 2–4. Retrieved from http://search.ebscohost.com.wgu.idm.oclc.org/login.aspx?direct=true&db=heh&AN=136410699&site=eds-live&scope=site

Kan, H. J., Kharrazi, H., Chang, H.-Y., Bodycombe, D., Lemke, K., & Weiner, J. P. (2019). Exploring the use of machine learning for risk adjustment: A comparison of standard and penalized linear regression models in predicting health care costs in older adults. Plos One, 14(3), e0213258. https://doiorg.wgu.idm.oclc.org/10.1371/journal.pone.0213258

The Shapely User Manual — Shapely 1.6 documentation. (2019). Shapely.readthedocs.io. Retrieved 25 October 2019, from https://shapely.readthedocs.io/en/stable/manual.html

Yu, C. H. (1977). Exploratory data analysis. Methods, 2, 131-160.

Tuffery, S. (2011). Data mining and statistics for decision making. Wiley. Retrieved from http://search.ebscohost.com.wgu.idm.oclc.org/login.aspx?direct=true&db=cat07141a&AN=ebc.EBC792450&site=eds-live&scope=site

Hastie, T., Tibshirani, R., Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction 2nd edition. Retrieved from https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12.pdf

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society: Series B (Methodological), 58(1), 267-288.

Hoerl, A. E., & Kennard, R. W. (2000). Ridge Regression: Biased Estimation for Nonorthogonal Problems. Technometrics, 42(1), 80. https://doi-org.wgu.idm.oclc.org/10.1080/00401706.2000.10485983

Segal, M. R. (2004). Machine learning benchmarks and random forest regression.

Sagi, O., Rokach, L. (2018). Ensemble learning: A survey. WIREs: Data Mining & Knowledge Discovery, 8(4), 1. https://doi-org.wgu.idm.oclc.org/10.1002/widm.1249

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794). ACM.