

Take Home Data Analysis

By: Christopher Leung



1. Data Overview
   1. Before we get into cleaning of data or analyzing of data, lets just look at the first rows of data.



A few notes about the data:

* Judging from the look of the data it looks like standard housing price data that is from Atlanta, some of the columns starting with features appear to have a nested list within each row. This is probably generated from the source data in which was being grabbed from. There is also one column (**xf\_attributes**) that has what looks like a dictionary within it but not quite.

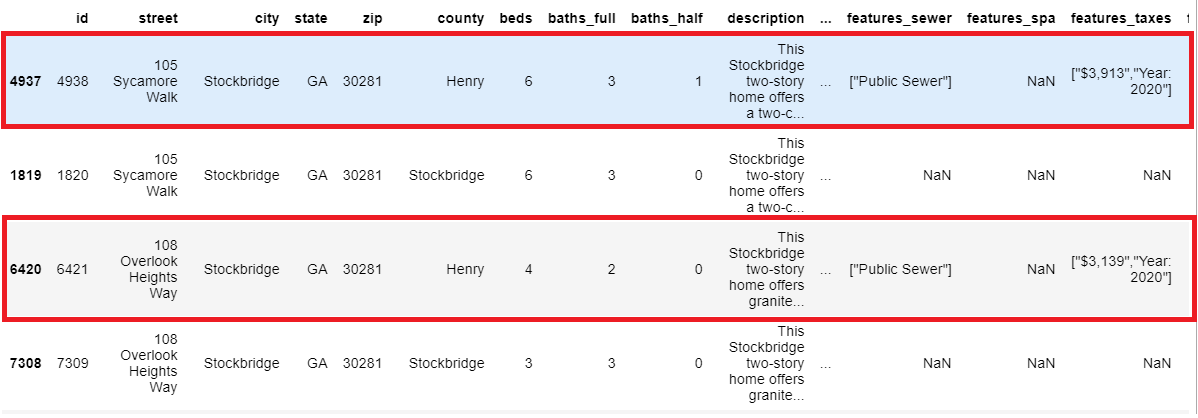
Python Code Used:

* **Import pandas as pd** – code allows the python library pandas to be called
* **Import numpy as np** – code allows the python library numpy to be called.
* **mls** variable to store the data as a dataframe and **mls.head()** displays the top few rows





1. Objective
   1. The goal of my analysis is to utilize machine learning models to predict the housing price data in Atlanta. As prerequisites before implementing a model, I would need to do some general data preprocessing and data housework to see which variables to pick along with removing any anomalies data. In addition, I will do small ad hoc analysis to get a feel for how the data distributions are and what models to implement for such a data set.
2. Data Preprocessing – Part A
   1. **Streets** – this column has some issues with duplications of name, to get further information I have pulled in the information below.



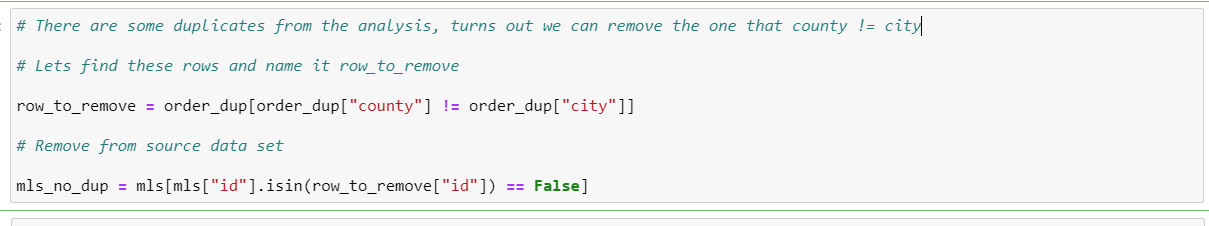
Observation:

* There appears to be a mix up between the counties and some counties are not aligned with the street and city. After doing a quick google check to see if the county exists in the city/state, I find out that the counties that is not the same as city name are the faulty rows and must be deleted before any analysis occurs.

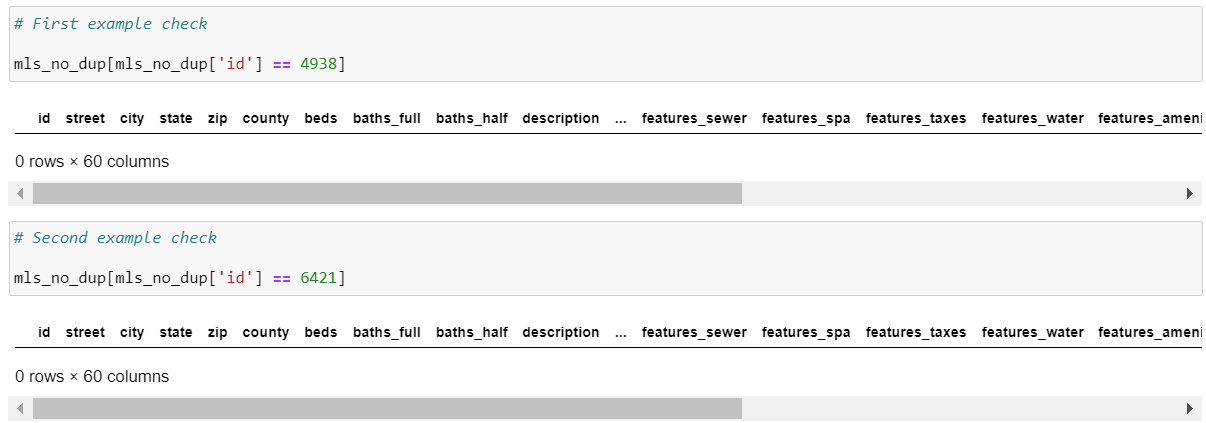


Python Code Used:

* The **row\_to\_remove** variable stores the rows with county that does not equal to city.
* **mls\_no\_dup** variable simply finds the rows of data that do not exist in the **row\_to\_remove**.

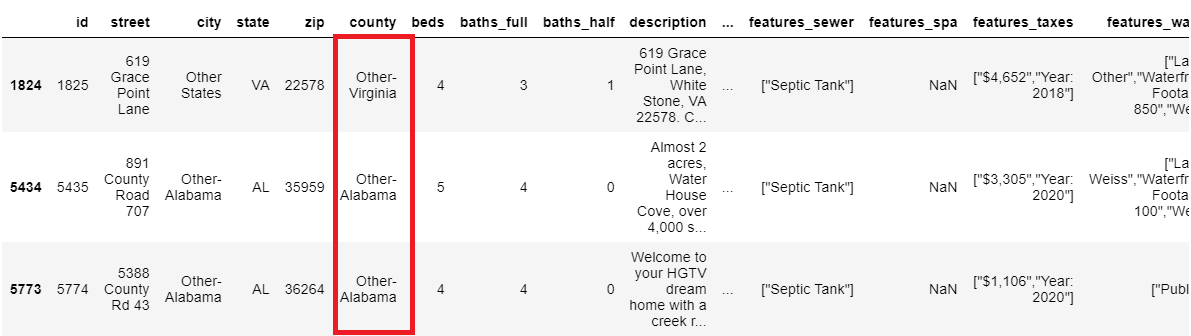


* The result of testing the first row and third row is shown below. Great! The data have been removed from analysis and we have called this variable **mls\_no\_dup**.



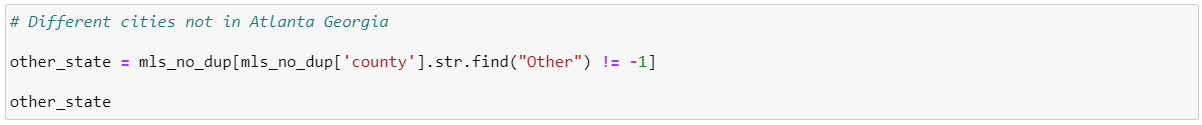


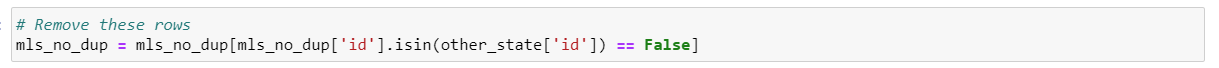
1. Data Preprocessing – Part C
   1. **County** – There are some rows that don’t pertain to our Atlanta, Georgia location for our analysis, lets remove them quickly.



Python Code:

* **other\_state** variable is to store the counties that do not have “Other” in the string. The **str.find(“Other”)** function simply finds any county with the text “Other” within it. And since the function displays a numerical value from it, we need to find the ones that are not numerically as **-1** since the default number it gives out when it doesn’t find a text is **-1.** In other words, the counties other than Atlanta are stored in it.
* **mls\_no\_dup** variable simply removes the addresses not from Atlanta.







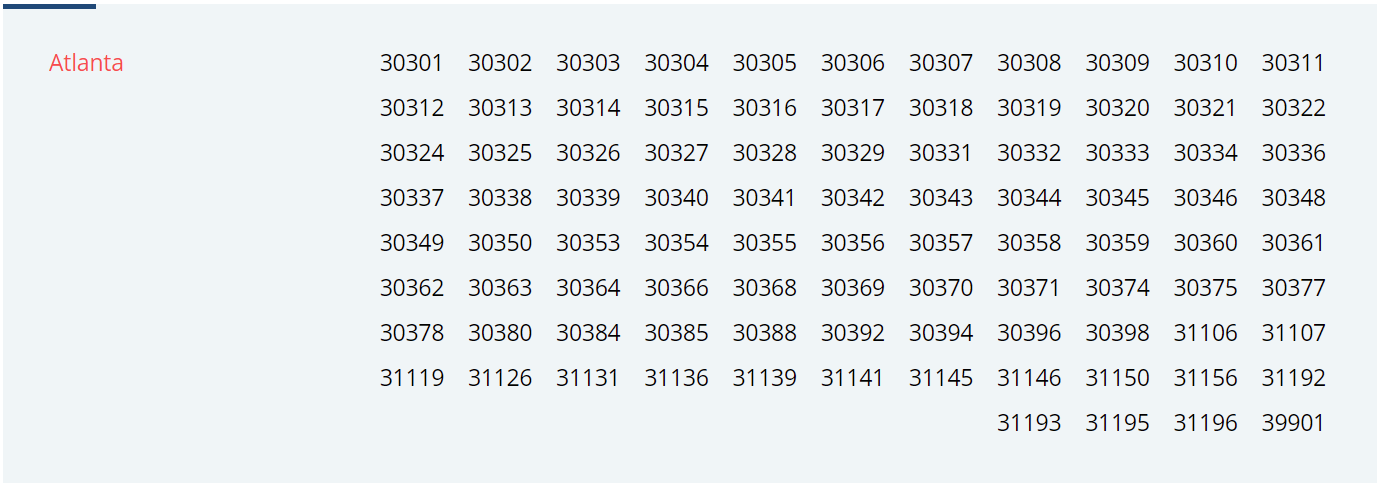
Our result:

* Nice! It’s all been removed properly



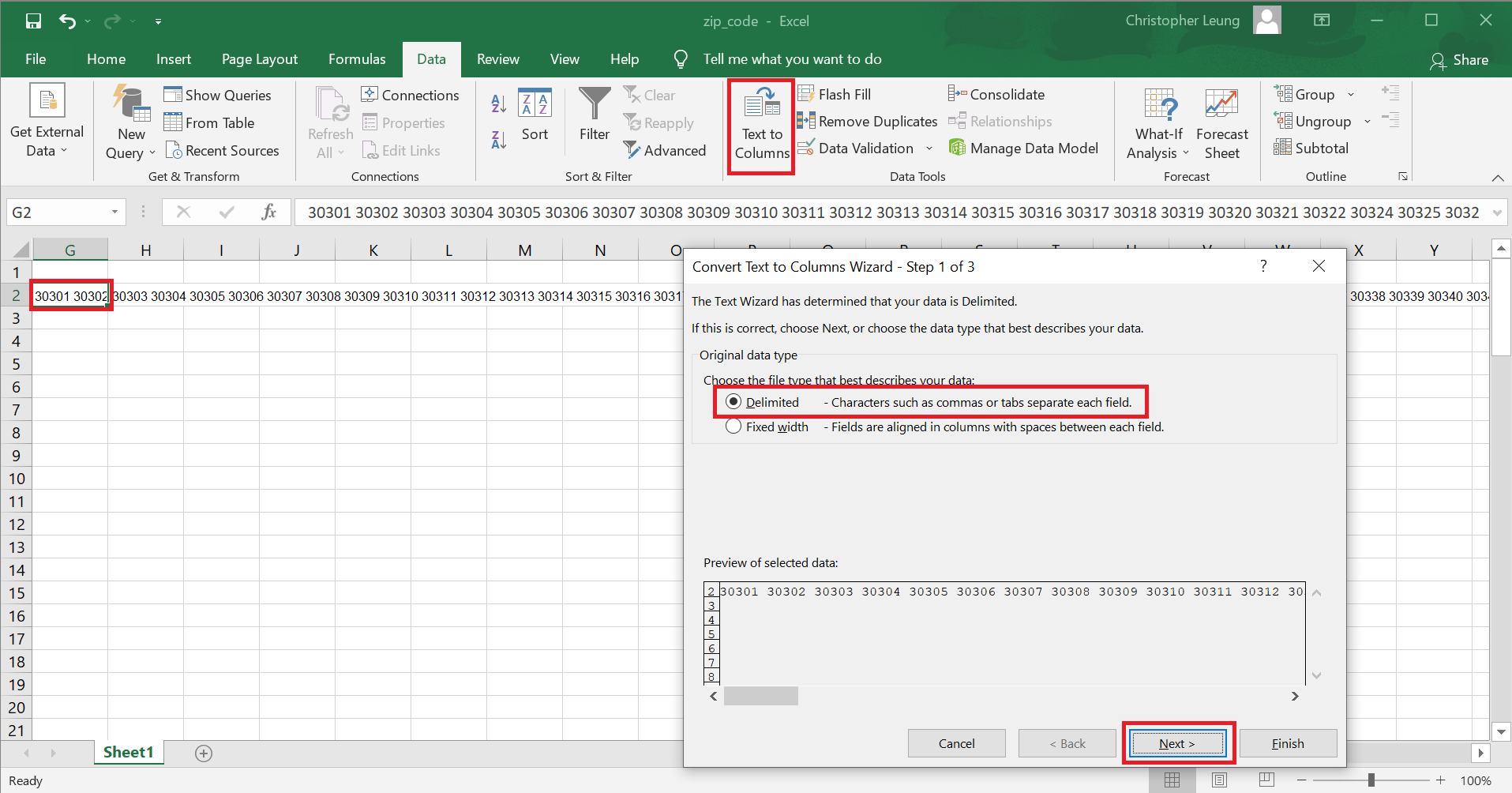


1. Data Preprocessing – Part D
   1. **Zip codes** – Lets remove the zip codes that are clearly not in the Atlanta area. I extracted a list of zip codes from ([**https://worldpostalcode.com/united-states/georgia/atlanta**](https://worldpostalcode.com/united-states/georgia/atlanta)). It should be a good accurate estimation of all zip codes in Atlanta. Now how do I get this into Python?

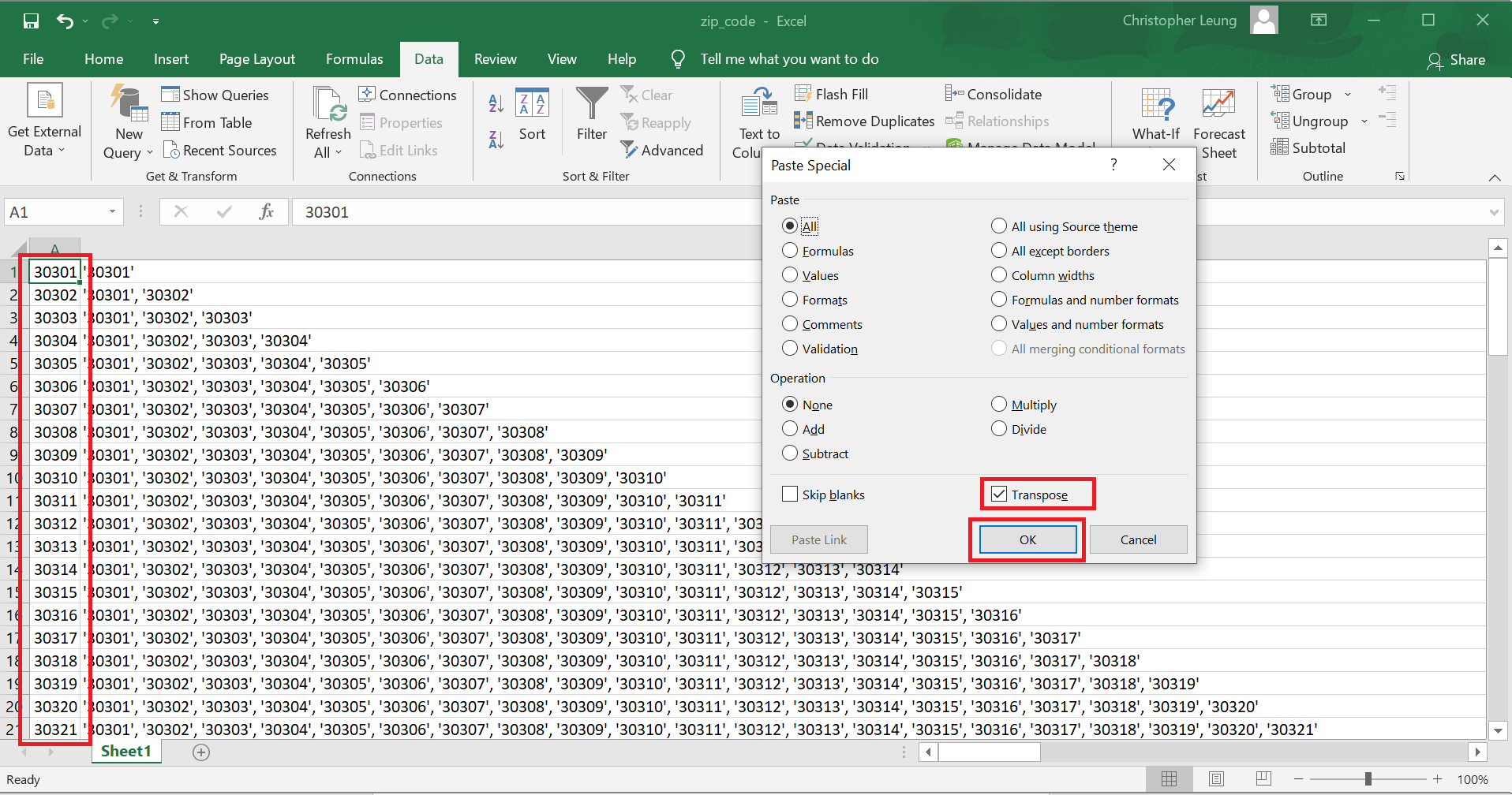




* Let’s just copy and paste the above data into Microsoft Excel. Next, we want to remove those spaces to separate the numbers into different columns per zip code.

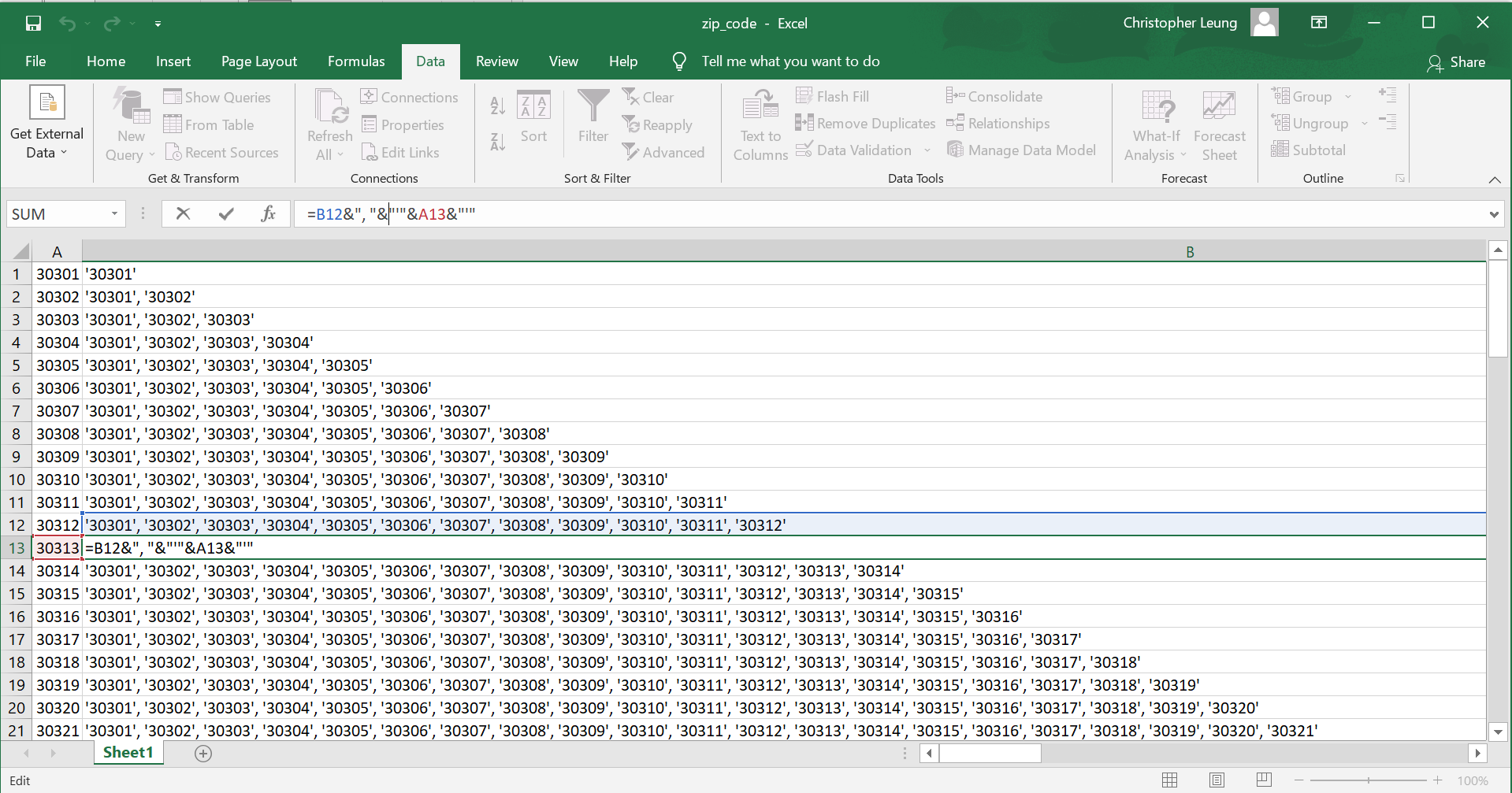


* Once that has been separated cleanly, we can simply copy and paste transposed data into the first column.





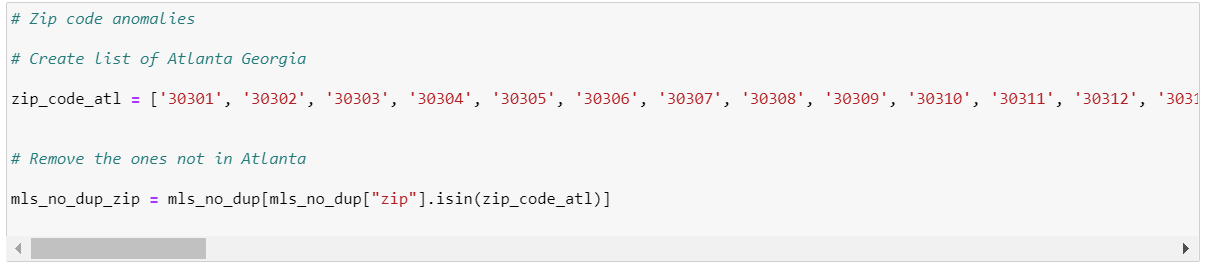
* Let’s do some data magic! We eventually want to create a list in python that allows us to grab only zip codes from Atlanta from our data set. We make this easier by starting a base for the python list code. The Excel formula highlights the cell above and to the left of the cell, what it does is concatenate the previous zip codes and the new zip code using the ‘ symbol and commas.





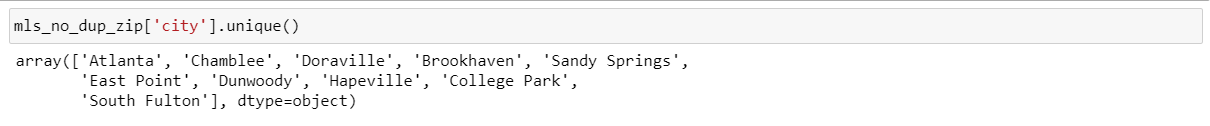
Python Code:

* **zip\_code\_atl** – copy all the zip codes from the convenience of Microsoft Excel and paste it into the variable. We make the zip\_code\_atl variable a list of all Atlanta zip codes.
* **mls\_no\_dup\_zip** – variable that finds all the zip codes that are contained from the **zip\_code\_atl** variable.



Our result:

* + We quickly look at the cities to see if they are from Atlanta. It looks pretty good. The **.unique()** function displays the distinct values in the column city.





1. Data Preprocessing – Part E:
   * **Beds** – There are some rows with beds that are 0. I need to find a number to replace the zero.



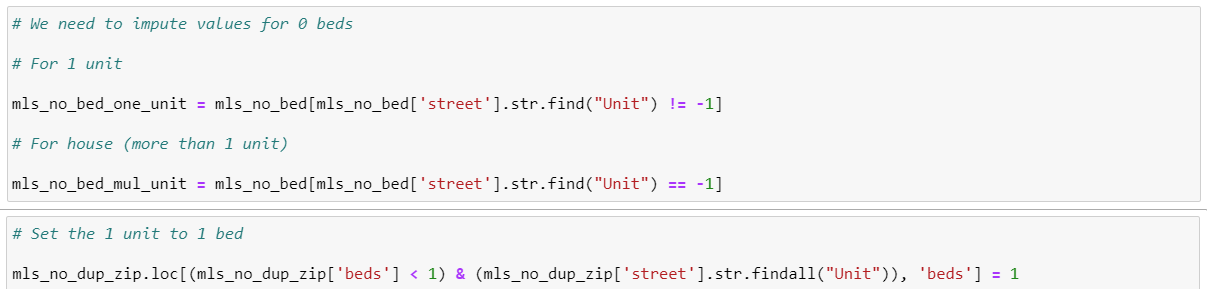
Things to note:

* + The streets indicated as “Unit” appear to be one unit, like an apartment. For these rows I will assume it is a 1 bedroom. For the other multi room bedrooms, I will take a mode of the most popular number of bedrooms from the data set.

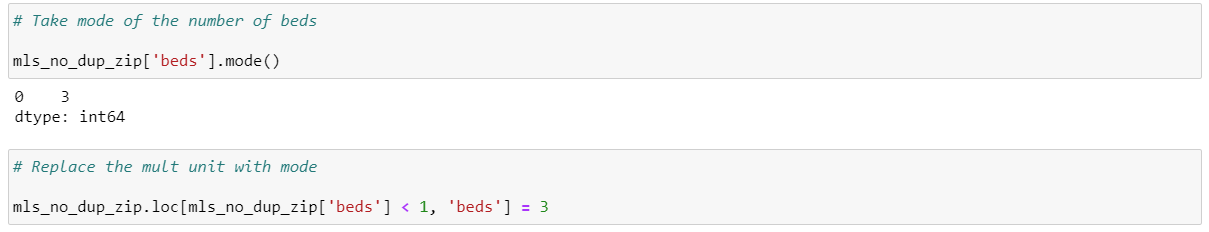


Python Code:

* + We declared variable **mls\_no\_bed\_one\_unit** as the filter to find the ones with the string “Unit” in it. The other variable **mls\_no\_bed\_mul\_unit** is a filter to find the ones without the string “Unit”.
  + The last line of code in python simply finds the one unit from the original dataset with 0 bed and we now input 1 in it. As a reminder, **mls\_no\_dup\_zip** is the original dataset and the other two variables (**mls\_no\_bed\_one\_unit, mls\_no\_bed\_mul\_unit**) are splits from it, it doesn’t contain all the data we need. The ampersand symbol is to join the two criterias together, the **0** beds and the street names that contain **Unit** within it.

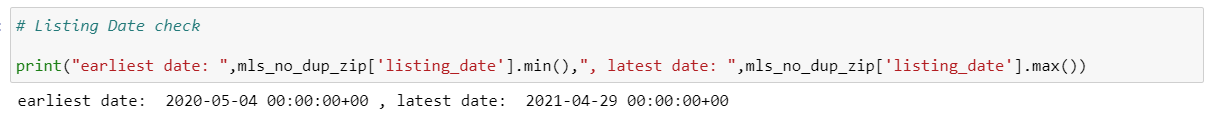


* Next step is to fix the non-unit addresses with **0** beds. The first code {**mls\_no\_dup\_zip['beds'].mode()**} takes the statistic mode of beds which is 3. The next line of code {**mls\_no\_dup\_zip.loc[mls\_no\_dup\_zip['beds'] < 1, 'beds'] = 3**} inputs the number 3 into the criteria of **0** beds and since the remaining 0 beds are the ones non-unit, we can simply inject **3** into these rows.

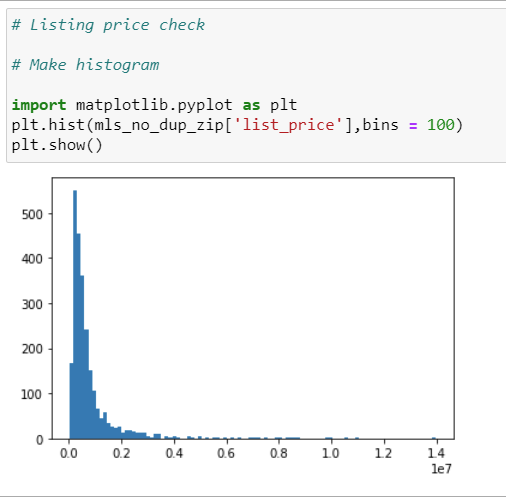




1. Data Preprocessing – Part F:
   1. **Listing Dates -** Let’s check the next column to ensure the dates are aligned with the times specified. Active till 4/29/2021. Perfect!
   2. The **.min()** and **.max()** finds the minimum and maximum dates for the column **listing\_date**

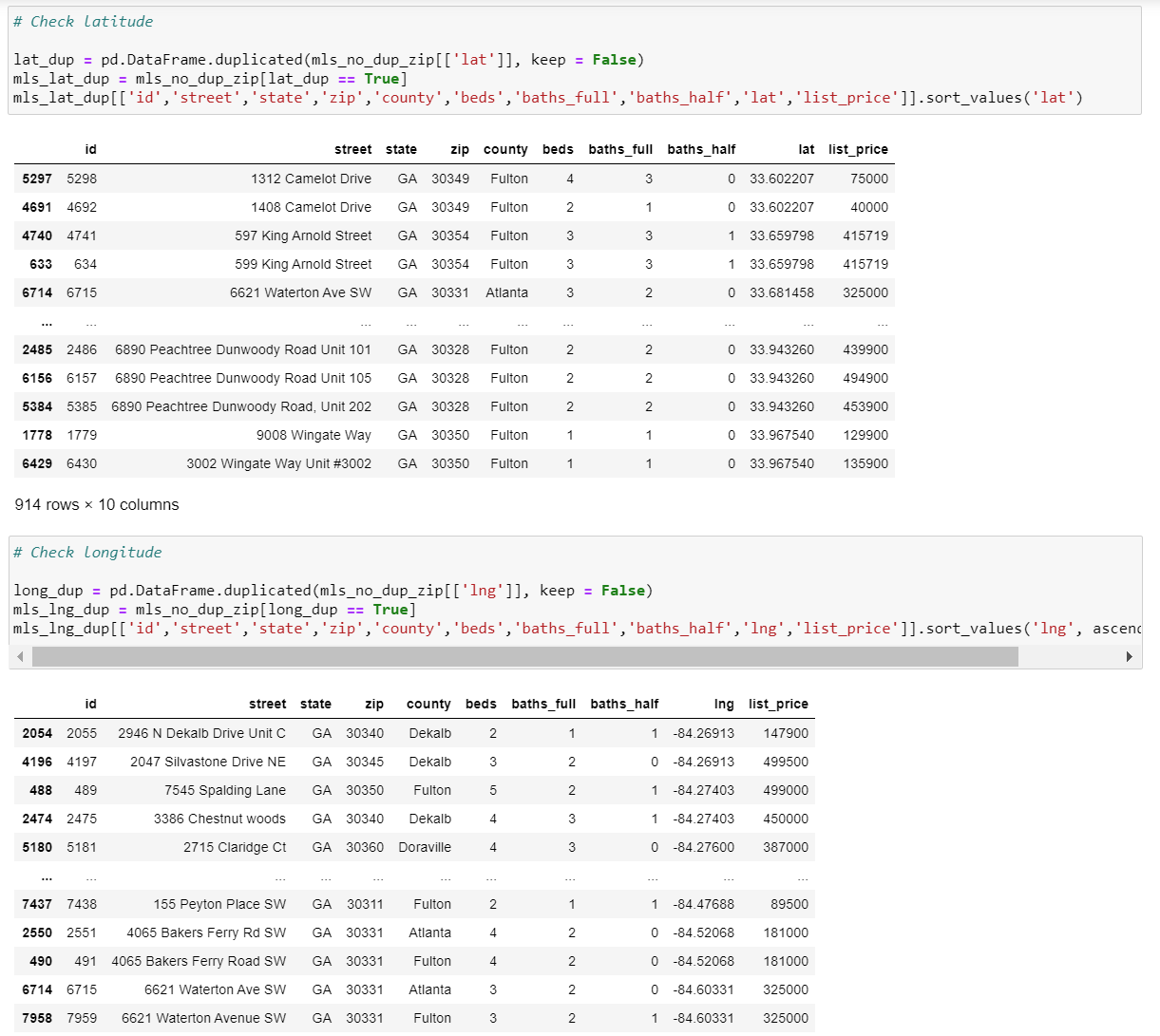


1. Data Preprocessing – Part G:
   1. **List Price –** We need to check list price for any abnormalities since this is a very important variable. I created a histogram below to see the distribution of the prices. There are some outliers but I don’t feel I should significantly remove them as they are a part of how the housing markets are. Sometimes I will remove outliers but, in this case, we have the majority of data in the lower end of the spectrum in terms of list price. The y axis is the count of houses with a list price in x axis but this x axis is the bin size broken up in 100 segments of all the list prices. This is simply a small analysis, not a perfect way to visualize our data.





1. Data Preprocessing – Part H:
   1. **Latitude and Longitude** – There has been some latitude numbers and longitude numbers that are duplicate, with my analysis I did break them up and looked at each individual one as it was easier for me to see. It is not the best method to look at but I see some addresses are very close to each other, maybe not enough to cause a huge difference in the coordinate system. I deemed it ok to use as the data appears pretty reliable.

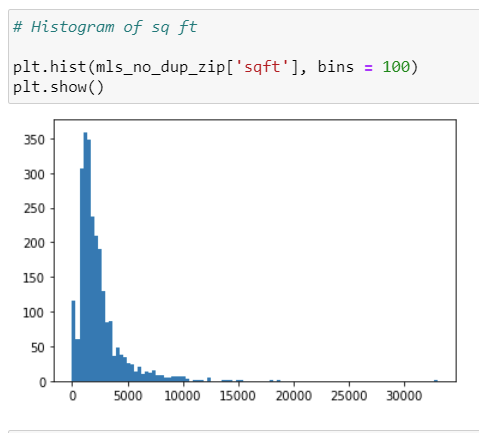


Python Code:

* **lat\_dup and long\_dup** variables do the same thing as they find the duplicate latitude and longitude from the most updated filtered data set (**mls\_no\_dup\_zip**). And the last line of code for long and lat are just to sort the data ascending with less columns.



1. Data Preprocessing – Part I:
   1. **Square feet** – I wanted to plot the histogram to see for any anomalies in this variable. It turns out there are a few houses with 0 square feet and its obviously incorrect. I tried looking for another number in the other columns that might’ve had it but I was quite unlucky. Since it was a small portion of the data (116 records), I felt it was ok to remove them from analysis.



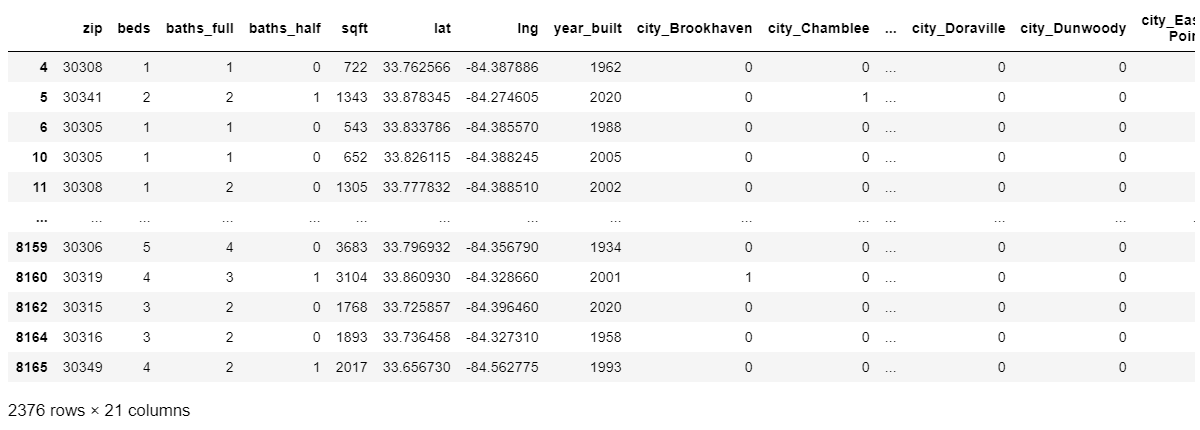
Python Code:

* **mls\_no\_dup\_zip\_sqft –** variable that only takes square feet greater than 0 from the most recent dataset (**mls\_no\_dup\_zip**).





1. Data Preprocessing – Part J:
   1. Now that most of our variables are cleaned up, before pushing this information into an algorithm, I would need to make sure all categorical (non-numerical) data is numerical as mathematics cannot compute categories. A way to solve this is to create more columns that are binary (0 or 1) for the data sets that have that particular attribute. This is a called **dummy variable** in python. The columns city\_Brookhaven, city\_Chamblee are created dummy variables by the code and marks 1 or 0 depending if that address is from there.



* The variables I picked which I felt would provide the least amount of bias error (bias error is the error in which we have not picked or assumed the correct variables that would affect the target variable (price listed)). They are **city, zip, beds, baths\_full, baths\_half, sqft, property\_type, lat , lng , and year\_built**.

Python Code:

* Using package **get\_dummies** from the **pandas** library, we can injest the categorical information into variable **X**.
* **Y** variable is just the target variable that we want to predict.





1. **Model Development – Linear Regression:**
   1. To start off, I wanted to use a simple model linear regression to see if this model will give us a good baseline result. This math model has been around for many, many years and can still be reliable. I decided to split the data set randomly training dataset will be 70% of the entire data and 30% will be test data set. Training data set means the model will read the information and see the target variable list price and basically learn from it.
   2. A linear regression is a model that fits a line in your multi-dimensional dataset, formula being:

* (Y = m\*x + b) and in this case we have a lot of x variables, it would be more like
* Y = m\*((city), ( zip), (beds), (baths\_full), etc.) + b
* m is the slope or coefficient of the variable; this is determining the fit of the model.
* B is the intercept; it is just a point the line starts from. Y is the target variable (list\_price).
  1. **R-squared** is a metric used to measure performance of an algorithm. Its range is from 0 to 1, looks like an exam score but it’s not quite an exam score for a model, although it’s a good way to evaluate a model. It’s a way of looking at the accuracy of the model but nothing is perfect.
     1. The exact definition of **R-squared** is it is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. Let’s just say it’s a measurement for performance accuracy and we will use this as a grading metric.

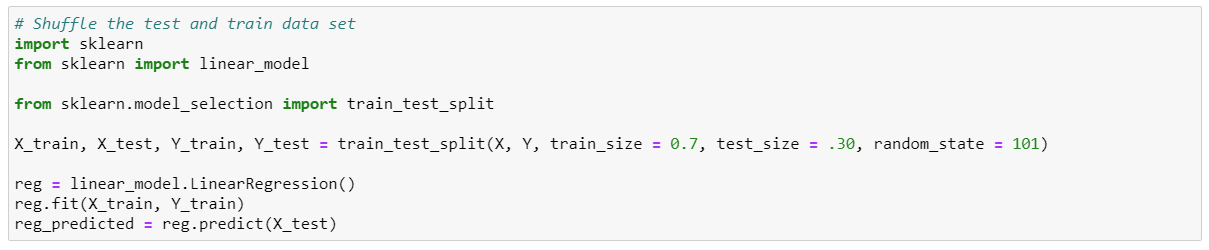


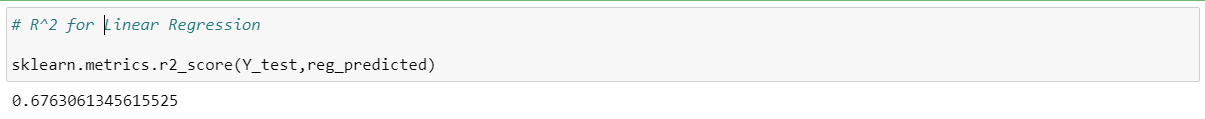
Python Code:

* We import **sklearn** as the main package to any data science modeling tool and import the **linear\_model**.
* **X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, train\_size = 0.7, test\_size = .30, random\_state = 101)**

Splits the data set into:

* + 70% goes to (X\_train & Y\_train) and 30% goes to (X\_test & Y\_test)
  + X\_train or X\_test are theindependentvariables from above (city, zip, beds, baths\_full, baths\_half, sqft, property\_type, lat , lng , and year\_built).
  + Y\_train and Y\_test is the list\_price our target variable.
  + **Random\_state** randomizes these data points before they are split in (70%/30%), you can pick any number really.
* **Reg = linear\_model.LinearRegression()** – opens the linear regression model package
* **Reg.fit(X\_train, Y\_train)** – fits the model so it can be trained to predict
* **reg\_predicted = reg.predict(X\_test)** – predicts the test data set
* **sklearn.metrics.r2\_score(Y\_test,reg\_predicted)** – creates the R-squared percentage metric



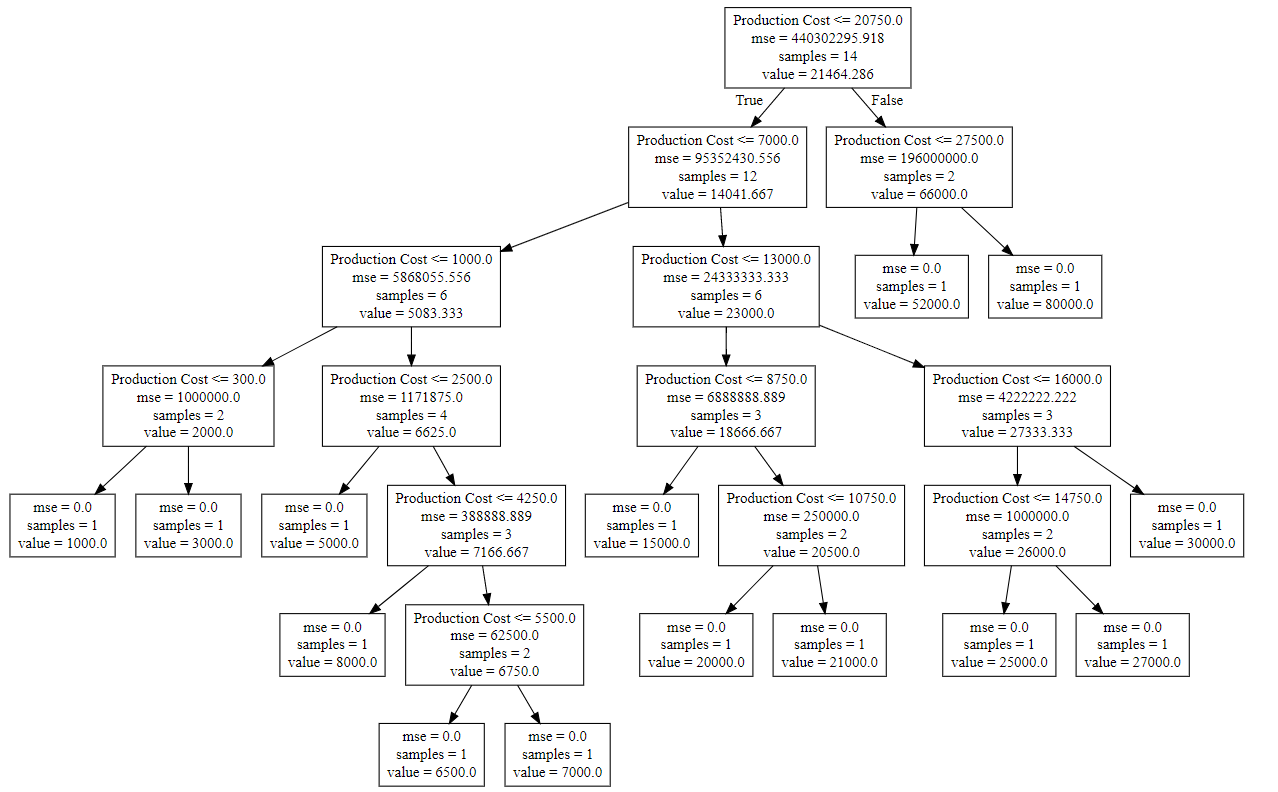


Thoughts:

* The model isn’t too accurate but not terrible on first try at 67.6%.

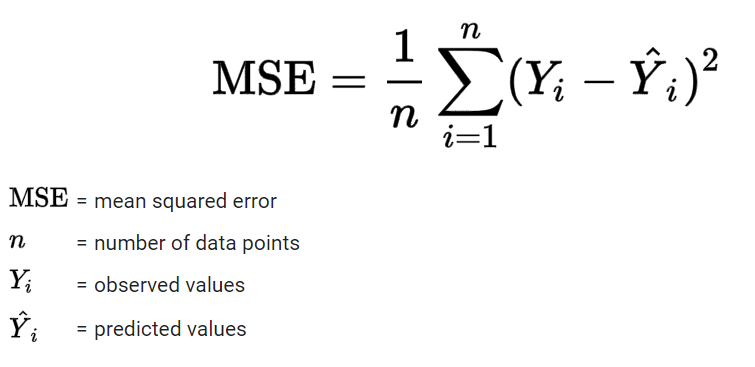


1. **Model Development – Random Forest:**
   1. For my second model I decided to pick Random Forest as it is a simple model to deploy and requires less computation power in relations to a gradient boosted tree or a neural network, in which are very complex models that require parameter tuning.
   2. A Random Forest is a model that is comprised of many Decision Trees (however many you want). Each Decision Tree takes a random number of samples with replacement from the data set, so a Decision Tree in a Random Forest may have the opportunity to have the same address twice in the Decision Tree. Once we have calculated all the Decision Trees, we aggregate all of them and take an average if we are predicting a number or taking a majority vote if we are predicting a category (such as Yes buyers or No buyers).
   3. A Decision Tree divides your data by Information Gain and Entropy helps calculate it. The idea is to continually split your tree until you get to the purer result. An example of one Decision is below. We try to split the Production Cost into more trees by its numerical value and the best decision for the lowest Mean Squared Error. What is Mean Squared Error (MSE)?





* 1. The math behind one Decision Tree is as follows, it utilizes Mean Squared Error (MSE) to get the average error per tree. The idea is averaging the squared difference between your predicted and actual values. This measurement is impurity, when a data is impure it has high volatility within it, meaning there is a lot of noise and not enough patterns to split the data.



Python Code:

* **from sklearn.ensemble import RandomForestRegressor** – enables the python package from library
* **rf = RandomForestRegressor(n\_estimators=1000, max\_features=0.4)** – opens the Random Forest model package.
  + **N\_estimators** is how many trees you want to use, generally the more trees you have the more accurate your predictor is but it will require more computational power to deliver the calculations for 1000 trees.
  + **Max\_features** is when to split the node, general rule of thumb is square root but in this case using 0.4 gave me the best results
* **rf.fit(X\_train, Y\_train)**– fits the model so it can be trained to predict
* **rf\_predicted = rf.predict(X\_test)** – predicts the test data set
* **sklearn.metrics.r2\_score(Y\_test,reg\_predicted)** – creates the R-squared percentage metric





Results:

* The model accuracy improved from the last model which was the linear regression. The R^2 improved to 79%, which to me is a much better model than linear regression model.

1. **Things I hadn’t been able to get to:**
   1. If there is additional time, I would love to look into each of the feature\_ variables such as features\_appliances\_and\_equipment, features\_bathrooms, etc.
   2. Created a **feature importance** calculation to figure out which features to actually pick to lower bias error.
   3. Utilized more complex models such as **gradient boosted trees** or **neural networks** to get a much more accurate prediction.
   4. Looked into **Mean Absolute Error (MAE)** and **MSE (Mean Squared Error)** as a way to evaluate the model instead of just relying on **R-squared**.
   5. Grabbed more data to run analysis as after the data preprocessing step, I had to remove a lot of erroneous data and the amount of data shrunk which can cause accuracy issues with model.
   6. Utilized **K-Fold Cross Validation** to allow the model to train on k segments of the training data, this would lead to increase in accuracy for the model if this function was used.