**Data Science: Dealing with Null Values in Python (NaN’s)**

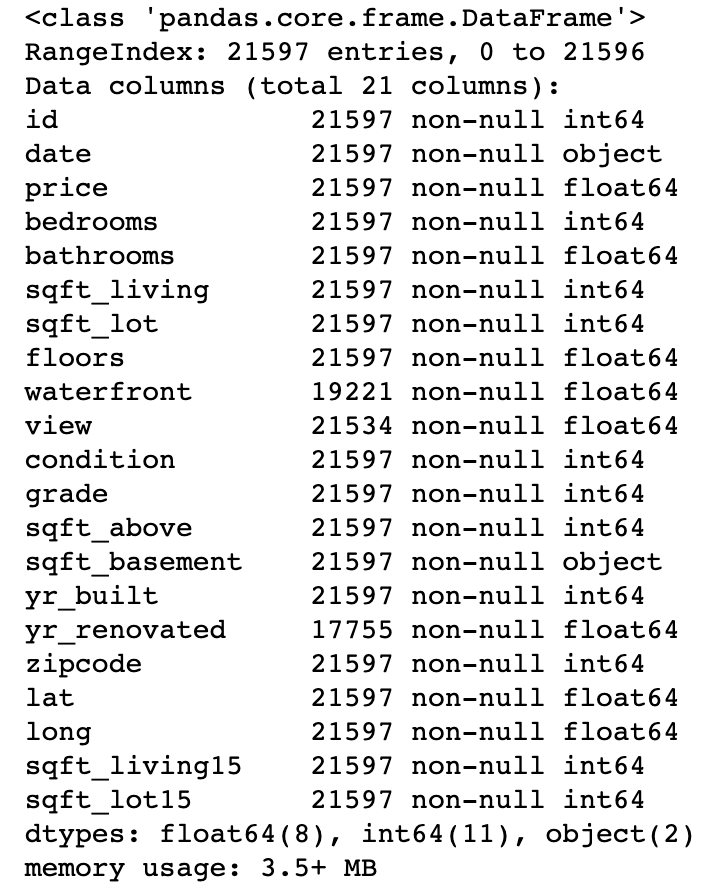
I’ve just completed my first data science project and have decided to write about one of the first tasks I completed: dealing with null values!

Please note, for the purpose of this blog I will using the “House Sales in King County, USA” dataset to assist with my examples. This dataset can be found on kaggle.com if you want to code along1!

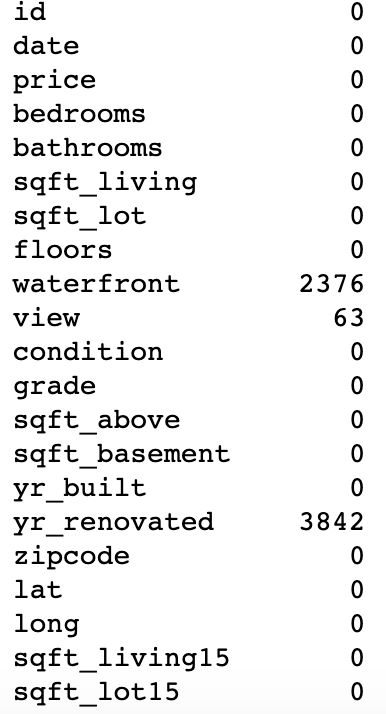
After loading in my dataset, and initially naming my dataset = “data”, I immediately started looking at what my dataset contained. Typing into Python data.info() I could view a list of each column within the data set, the total number of non-null values within each column and the data type of each column. *Please note, for the purposes of this blog each column we will be looking at will be continuous.*  While the information I get from using data.info() is extremely helpful, when dealing particularly with null values, I’ve found it an easier to view which columns contain null values by using data.isna().sum() as well. Using data.isna().sum() I am able to see the total number of null values for each column, rather then having to look through and see which columns less non-null values has then the total count, which can be quite overwhelming on my eyes.

Please look at an example of each output below and you’ll be able to see what I’m talking about.

data.info():



data.isna().sum():



Don’t you agree that It is much easier to see that waterfront, view and yr\_renovated all contain null values in the second output compared to the first output? Now on to dealing with these null values!

There are three different methods I can use when dealing with my null values: Keep, Replace or Remove. In order to decide the best method to use it is necessary that I look into each individual column that contains null values further.

The first method I use to help me do this is to determine the percentage of null values for each column:

yr\_ren\_null\_percentage = str(round(len(data[data.yr\_renovated.isna()])

/ len(data),3)\*100)

wf\_null\_percentage = str(round(len(data[data.waterfront.isna()])

/ len(data),3)\*100)

view\_null\_percentage = str(round(len(data[data.view.isna()])

/len(data),3)\*100)

print('yr\_ren percent null: ', yr\_ren\_null\_percentage + '%')

print('waterfront percent null: ', wf\_null\_percentage + '%')

print('view percent null: ', view\_null\_percentage + '%')

The code above outputs the three very helpful lines below for my analysis:

yr\_ren percent null: 17.8%

waterfront percent null: 11.0%

view percent null: 0.3%

These lines of code tell me the percentage of null values that exist in total based on each column of data. While I need to look further into how to deal with each column, I do know that removing the rows with null values for the columns like yr\_renovated and waterfront, that contain 10%-20% of null values, would be removing a large portion of my data and could cause a skew in my results. On the other hand, if there was a larger percentage of null values in each column then I would consider removing the column altogether. However, less than 20% is not too large of a percentage of null values so this is not the case for my dataset (and as a data scientist I know it can be risky for my model to delete out too much data).

Going back to my results, I see that view has only 0.3% of null values and at this point believe I will be deleting out the rows that contain null values from my dataset for this column since it is such a tiny percentage should not skew my data. However, before doing so I will still look into the unique values that view provides and the description/meaning for these values to make sure I am comfortable removing the null values and my data will not be badly skewed.

To further analyze the data in these columns, I’ll move from the largest to smallest percentage of null values for each column and perform the following steps (please see my example for yr\_renovated below):

STEP1: Look at how many unique values exist for the column

yr\_ren\_unique = data['yr\_renovated'].nunique()

print('unique yr\_renovated values =', yr\_ren\_unique)

STEP2: Look at a histogram plot for the column to determine the range of unique values

data.yr\_renovated.plot(kind = 'hist')

STEP3: Look at the counts of each unique value for the column to determine the largest to smallest

data.yr\_renovated.value\_counts()

STEP4: Determine the percentage of the first and second largest unique values for the column (for the example below yr\_renovated largest value was 0.0 and second largest was 2014)

yr\_ren\_zero = str(round((data[data['yr\_renovated']

== 0.0].count()['yr\_renovated']

/ len(data)),2) \* 100)

print('percentage of yr\_renovated 0.0 values =', yr\_ren\_zero + '%')

yr\_ren\_2014 = str(round((data[data['yr\_renovated']

== 2014].count()['yr\_renovated']

/ len(data)),3) \* 100)

print('percentage of yr\_renovated 2014 values =', yr\_ren\_2014 + '%')

For this particular dataset I am able to see that most of my values are 0.0 for yr\_renovated at 79.0% based on the code above. Analyzing this a bit further, this means that 79% of the homes I’m looking at in my dataset have not been renovated. 17.8% of the data in this column are null values leaving only 3.2% of the data with yr\_renovated years populated. I believe it’s safe to assume that the homes with null values were also not renovated based on how many were not renovated in my initial data set, so I have decided to replace the null values with 0.0.

Please note, before making any changes I will create a new dataframe since I am starting to manually make adjustments to my data. I will name my new dataset “data2”: data2 = data. Now that that’s taken care of, I will replace my null values with 0.0 for yr\_renovated.

data2['yr\_renovated'] = data2['yr\_renovated'].fillna(value = 0.0)

And now yr\_renovated will no longer contain those pesky null values!

If I instead decided to keep the null values for yr\_renovated I would update my null values (NaN) to become a string (‘NaN’).

data2['yr\_renovated'] = data2['yr\_renovated'].fillna(value = ‘NaN’)

Lastly, if I decided to remove the rows with null values, I would be able to do so using the formula below:

data2 = data2.dropna(subset=[‘yr\_renovated’])

Moving forward, I would now perform the same analysis for my other columns containing null values. By the end of this exercise I will no longer have any null values in my data and be able to move forward cleaning, exploring and predicting a proper model!

1“kc\_house\_data.csv” dataset containing 19 house features plus the price and the id columns, along with 21,613 observations between May 2014 and May 2015.