**Preparing the dataset - Data Wrangling**

1. **Checking for errors on categorical columns**

The document “data\_description.txt” shows all the allowed values for each categorical data. Let’s list all unique values on each categorical column to look for mistyped data. IMPORTANT: there are 3 categorical columns that are represented by numbers ('MSSubClass','OverallQual','OverallCond'). Please note the highlighted line where we added those 3 columns.

**import pandas as pd**

**df = pd.read\_csv('train.csv', header=0, index\_col='Id')**

**dfnumbers = df.\_get\_numeric\_data()**

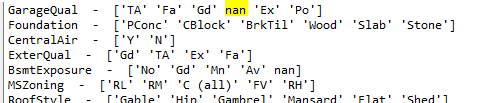
**catcols = set(df.columns) - set(dfnumbers.columns)**

**catcols = catcols.union(set(['MSSubClass','OverallQual','OverallCond']))**

**for col in catcols:**

**print(col, ' - ',df[col].unique())**

Here is a part of the output



With a quick inspection, we notice some NaNs that are not “missing data”. The highlighted nan, for example, means “no garage”, but it could also means “missing data” in other columns. **We’ll look into that later**.

Comparing the output with the data description, we noticed small errors that are very easy to miss if we inspect manually. So we created a .csv file called “AllPossibleValuesForCatCols.csv”, using the data\_description.txt as the source, to list all possible values those categorical columns may have. We’ll import this file as a Dataframe and match the actual values found on the dataset with the possible values for that column. This way we can find typos easier. This is the code to find all typos:

**df = pd.read\_csv('train.csv', header=0, index\_col='Id')**

**dfAll = pd.read\_csv('AllPossibleValuesForCatCols.csv', header=0)**

**dfAll = dfAll.applymap(lambda x: x.strip() if isinstance(x, str) else x)**

**diffs = {}**

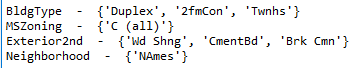
**for col in dfAll.columns:**

**diffs[col] = set(df[col]) - set(dfAll[col])**

**for key, value in diffs.items():**

**if(len(value)>0): print(key, ' - ',value)**

Here is the output:



wechanged the following:

* Column **BldgType**: “Duplex” should be “Duplx” and “2fmCon” should be “2FmCon”. “Twnhs” should be “TwnhsI”, because we see “TwnhsE” on the data but no TwnhsI:

**df.BldgType = df.BldgType.replace('Twnhs', 'TwnhsI')**

**df.BldgType = df.BldgType.replace('Duplex', 'Duplx')**

**df.BldgType = df.BldgType.replace('2fmCon', '2FmCon')**

* Column **MSZoning:** “C (all)” should be “C”

**df.MSZoning = df.MSZoning.replace('C (all)', 'C')**

* Column **Exterior2nd:** “Wd Shng” should be “WdShing” because wecan see “Wd Shng” on the data. “CmentBd” should be “CemntBd” and “Brk Cmn” should be “BrkComm”

**df.Exterior2nd = df.Exterior2nd.replace('Wd Shng', 'WdShing')**

**df.Exterior2nd = df.Exterior2nd.replace('CmentBd', 'CemntBd')**

**df.Exterior2nd = df.Exterior2nd.replace('Brk Cmn', 'BrkComm')**

* Column **Neighborhood**: “NAmes” should be “Names” (meaning North Ames)

**df.Neighborhood = df.Neighborhood.replace('NAmes', 'Names')**

If we run the same code again to check typos, the output is empty!

Back to the unique values from all columns, a few other things that caught my attention:

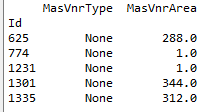
* Columns **MasVnrType** and **MasVnrArea:** there are “None” values and NaN on the MasVnrType column. As there are only 8 values in this situation, we used the following code to replace NaN with ‘None’ on column MasVnrType and NaN with 0 on the MasVnrArea column:

**df = df.replace({'MasVnrType':{np.nan : 'None'}, 'MasVnrArea': {np.nan : 0}}, value = None)**

* Still working on columns **MasVnrType** and **MasVnrArea,** when the MasVnrType is None, the MasVnrArea should be zero. The following code verifies this statement:

**print(df[['MasVnrType', 'MasVnrArea']][(df.MasVnrType == 'None') & (df.MasVnrArea > 0)])**

Output:

Which is not what we expected. To deal with that, the two records with Areas = 1.0 I’ll replace with Area = 0.0, and the other three values I’ll replace the MasVnrType with the most common Masonry veneer type. 

First, replace the two records where Area = 1.0 with 0.0 (there are only those two records)

**df.MasVnrArea = df.MasVnrArea.replace(1.0,0.0)**

For the other 3 values, let’s perform a frequency count on the MasVnrType column where the Area is >0 to find the most common value with the following code:

**print(df[['MasVnrType', 'MasVnrArea']][df.MasVnrArea > 0].MasVnrType.value\_counts())**

Which outputs the following:

BrkFace 444 Stone 127 BrkCmn 15 None 5

So, for the three other records, I’ll replace the type = ‘None’ with type = ‘BrkFace’:

**for index, row in df.iterrows():**

**if((row['MasVnrType'] == 'None') & (row['MasVnrArea'] > 0)):**

**df.loc[index, 'MasVnrType'] = 'BrkFace'**

Now, all Types = None have the Area = 0, as it should

This type of consistency will be checked for every column later (for example, if the house has no garage, all columns related to “garage” should show NA).

1. **Cleaning NaN from categorical columns**

We already know that missing values are stored as NaN. The following code lists the categorical columns with their unique values.

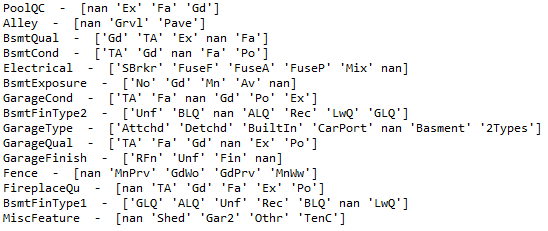
**nulls = df.isnull().sum()**

**catcolswithnan = set(dfAll.columns) & set(nulls[nulls>0].index)**

**for col in catcolswithnan:**

**print(col, ' - ',df[col].unique())**

Which produces the following output



For all columns, it makes sense to replace nan with “No Item”, except for the column “Electrical”. **It doesn’t make sense not to have an electrical system**. Let’s first deal with replacing all nan from the above columns with “No Item” and deal with the “Electrical” column later.

**for col in catcolswithnan:**

**df[col] = df[col].replace(np.nan, 'No Item')**

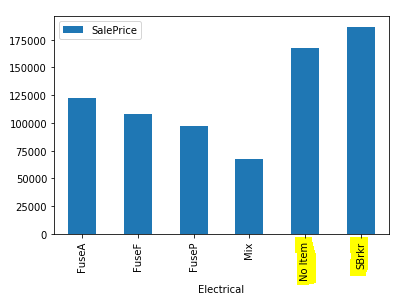
Now let’s check the Electrical colum. How many “No Item” does this column have?

**print(df[df.Electrical == 'No Item']['Electrical'].count())**

The output is 1 (row Id = 1380).

Let’s see how the price of this house stands against the other types of electrical systems

**df[['Electrical', 'SalePrice']].groupby('Electrical').mean().plot(kind = 'bar')**

**plt.show()**

We can easily notice that, according to the price of the house, the electrical system is likely to be the SBrkr, so let’s replace “No Item” by “SBrkr”:

**df.Electrical = df.Electrical.replace('No Item', 'SBrkr')**

1. **Cleaning NaN from numerical columns**

With the following code, we create a DataFrame with only the numeric columns:

**dfNumericCols = set(df.columns) - set(dfAll.columns)**

**dfNumeric = df[list(dfNumericCols)]**

Now let’s inspect the NaNs with the following code:

**print(dfNumeric.isnull().sum()[dfNumeric.isnull().sum()>0])**

Which yields the output:

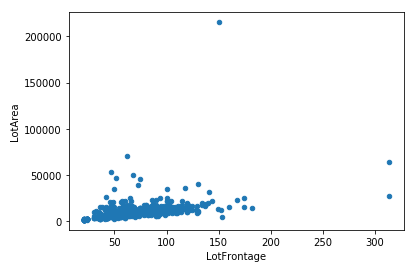


* Column **GarageYrBlt**: Hopefully, those NaN are for houses whithout garage. Let’s check the unique values of the colum GarageType when GarageYrBlt is NaN:

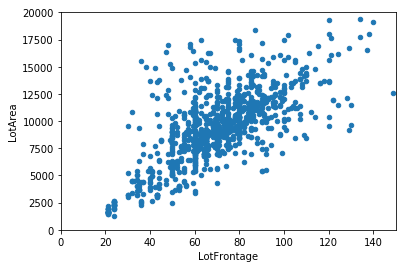
**print(df[['GarageType', 'GarageYrBlt']][df.GarageYrBlt.isnull()].GarageType.unique())**

Output is ['No Item'], so no action is needed.

* Column **LotFrontage**: those are legitimate NaNs, too many to discard. Inspecting the columns, we found a **LotArea.** Maybe a scatter plot will show a relationship between those two variables:

**df[['LotFrontage', 'LotArea']].plot(x = 'LotFrontage', y = 'LotArea', kind = 'scatter')**

We can easily identify the presence of outliers on both dimensions. To better identify the relationship, let’s “zoom in” in the data on the graphic. (we’ll deal with outliers later)

**df[['LotFrontage', 'LotArea']].plot(x = 'LotFrontage', y = 'LotArea',xlim=(0,150), ylim=(0,20000), kind = 'scatter')**

As suspected, there is a big correlation between those two variables. Filling those 259 NaN from LotFrontage with values that are proportional to the column LotArea makes sense. To do that, I’ll find the equation that best represents the linear regression between these two columns and apply it for the NaNs. As I still don’t know how to do it in Python, I’ll find the equation in excel

**y = 0.0046x + 27.113**

Based on this equation, the NaN from LotFrontage will be filled as follows:

**for index, col in df.iterrows():**

**if(np.isnan(col['LotFrontage'])):**

**df.loc[index, 'LotFrontage'] = 0.0046\*df.loc[index, 'LotArea'] + 27.113**

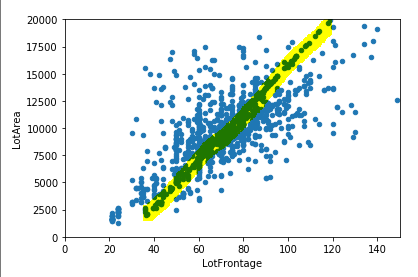
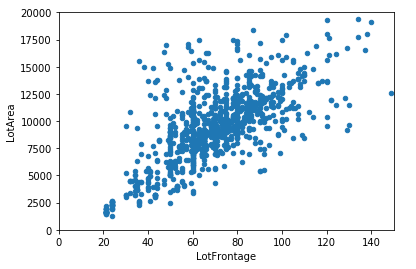
If we run the code to find null values on df:

**print(df.isnull().sum()[df.isnull().sum()>0])**

We have this as output

****

Which means we no longer have NaN on the column LotFrontage. Plotting the scatter plot before and after this operation, we can see the values added:

**BEFORE AFTER**

Good, but not ideal. Hopefully, I’ll improve this skill along the course.

1. **Consistency between columns**

If a house has no garage, all columns related to “garage” should reflect this fact. We’ve already done this for MasVnrType column on item 1 of this report, now let’s take the same approach for garage, Basement, fireplace, pool and miscellaneous features.

* 1. Garage

Columns to analyze: GarageType, GarageYrBlt, GarageFinish, GarageCars, GarageArea, GarageQual, GarageCond

Code:

**collist = ['GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual', 'GarageCond']**

**df.GarageType== 'No Item'**

**myDict = {}**

**for col in collist:**

**myDict[col] = []**

**for index, row in df.iterrows():**

**if(df.loc[index, 'GarageType'] == 'No Item'):**

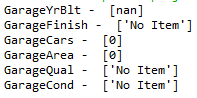
**for col in collist:**

**myDict[col].append(row[col])**

**dfGarage = pd.DataFrame(myDict)**

**for col in dfGarage.columns:**

**print(col, '- ', dfGarage[col].unique())**



Output:

* 1. Basement

Columns to analyze: BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinSF1, BsmtFinType2, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, BsmtFullBath, BsmtHalfBath

Code:

**collist = ['BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2',**

**'BsmtUnfSF', 'TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath']**

**df.BsmtQual == 'No Item'**

**myDict = {}**

**for col in collist:**

**myDict[col] = []**

**for index, row in df.iterrows():**

**if(df.loc[index, 'BsmtQual'] == 'No Item'):**

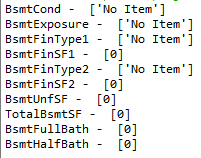
**for col in collist:**

**myDict[col].append(row[col])**

**dfBsmt = pd.DataFrame(myDict)**

**for col in dfBsmt.columns:**

**print(col, '- ', dfBsmt[col].unique())**

Output: 

* 1. Fireplace

Columns to analyze: Fireplaces, FireplaceQu

Code: **print(df[['Fireplaces', 'FireplaceQu']][(df.FireplaceQu == 'No Item') & (df.Fireplaces != 0 )])**

Output: Empty DataFrame

* 1. Pool

Columns to analyze: PoolArea, PoolQC

Code: **print(df[['PoolArea', 'PoolQC']][(df.PoolQC == 'No Item') & (df.PoolArea != 0 )])**

Output: Empty DataFrame

* 1. Miscellaneous

Columns to analyze: MiscFeature, MiscVal

Code: **print(df[['MiscFeature', 'MiscVal']][(df.MiscFeature == 'No Item') & (df.MiscVal != 0 )])**

Output: Empty DataFrame

For all groups of columns, no inconsistencies were found.

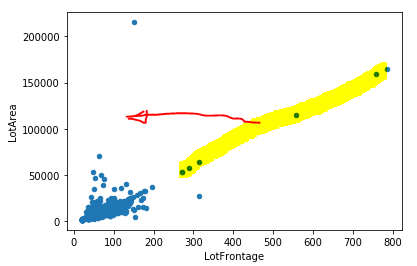
1. **Checking outliers**

As shown on the scatter plot previously presented, both LotFrontage and LotArea has outliers. In this section, we are going through a few relevant numerical columns to inspect outliers and decide whether to keep or discard them.

* LotArea and LotFrontage

Let’s print that scatter plot again but without the X and Y limits

**df[['LotFrontage', 'LotArea']].plot(x = 'LotFrontage', y = 'LotArea', kind = 'scatter')**

**#plt.show()**

We notice that we introduced outliers on our attempt to fill in the NaNs. As lotFrontage almost never surpasses the 200 units, we will limit this dimension to 200 regardless of the lot Area. It’s like we are using one regression equation for lotFrontage until 200 and another one for greater than 200.

Using Excel again to find the equation:

y = 0.0006x + 19.657

Applying this equation for LotFrontage > 200:

**aux = 0**

**for index, col in df.iterrows():**

**if(np.isnan(col['LotFrontage'])):**

**aux = 0.0046\*df.loc[index, 'LotArea'] + 27.113**

**if(aux>200):**

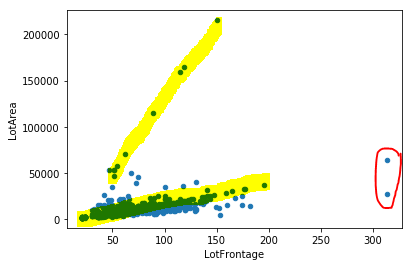
**df.loc[index, 'LotFrontage'] = 0.0006\*df.loc[index, 'LotArea'] + 19.657**

**else:**

**df.loc[index, 'LotFrontage'] = aux**

**df[['LotFrontage', 'LotArea']].plot(x = 'LotFrontage', y = 'LotArea', kind = 'scatter')**

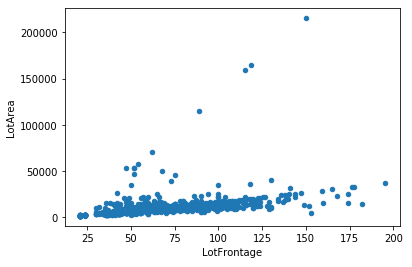
**plt.show()**

Output:

We can see highlighted in yellow the two equations used to get rid of NaNs. Even though it’s tempting to just discard the values where LotArea is greater than 50,000, the price of the properties has a strong correlation to the Lot Area, so we need those numbers. The two values inside the red circle, however, they do not bear relation to the house selling price, meaning they can undermine the precision of our future model. We’ll drop those two values.

**df= df[df.LotFrontage < 200]**

Output:



* Violin plots for the other variables

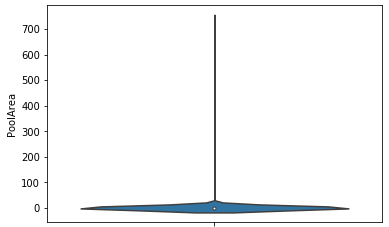
We used the following code to plot violin plots for all other variables.

**for col in dfNumericCols:**

**ax = sns.violinplot(y=col, data=df)**

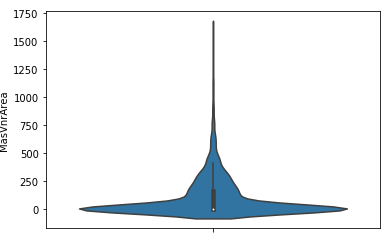
**plt.show()**

Inspecting each one of the curves, the following called our attention:

* PoolArea

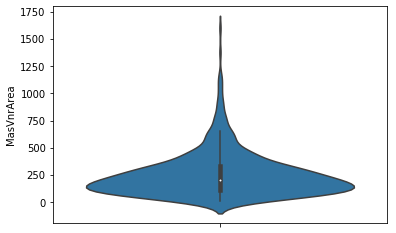
Looking back into the data, this behavior is expected as we only have 7 values different than 0. Maybe this data can help predict house selling prices for specif cases, so we’ll keep this column as-is.

The same argument goes to columns 3SsdPorch, BsmtFinSF2, LowQualFinSF, MiscVal, ScreenPorch and BsmtHalfBath.

* MasVnrArea

This variable has more than 500 samples different than zero, so let’s see if we can find outiers in a violin distribution that ignores the zeros.

Code: **ax = sns.violinplot(y='MasVnrArea', data = df[df.MasVnrArea > 0])**

Output:

Now this distribution has only a few outliers that can be useful for some cases and may have a combining effect with other variables, so we’ll keep it as is.

1. **Conclusion**

Based on my experience, this dataset is rather clean, allowing us to keep almost all records. We were able to spot typos and filled some NA values in a way that will foster the predicting power or our model. This dataset is now ready for the next steps.