Final Project - House Price Predictor

Group members: Kiran Sapra, Kuldeep Garg

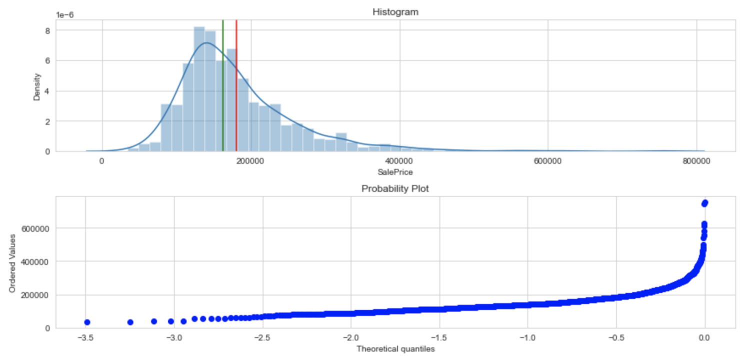
Although we found all project ideas interesting, we decided to work on the House Price Predictor because we both agreed this was very relatable to us since we have been discussing it before the project and being students, we move very often. We thought this would be a very helpful tool for new residents looking to buy a house.
Using the data set provided we predicted the price of houses based on information such as but not limited to House condition, sale price, number of rooms, size and area, etc.
Our submission includes a total of the following files. There's the notebook where we worked with the dataset, readme that has been updated weekly as per the deliverable dates and the Discussion writeup.
This project helped us refine our python skills. We started out by practicing our GIT CLI and importing the dataset provided to us.
We researched various Python libraries that we would need throughout the project and the following:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.gridspec as gridspec
from scipy import stats
import seaborn as sns
sns.set_style('whitegrid')

import warnings
warnings.simplefilter(action='ignore')

from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score, GridSearchCV, KFold, RandomizedSearchCV, train_test_split
import math
import sklearn.model_selection as ms
import sklearn.metrics as sklm
import requests as rq
```

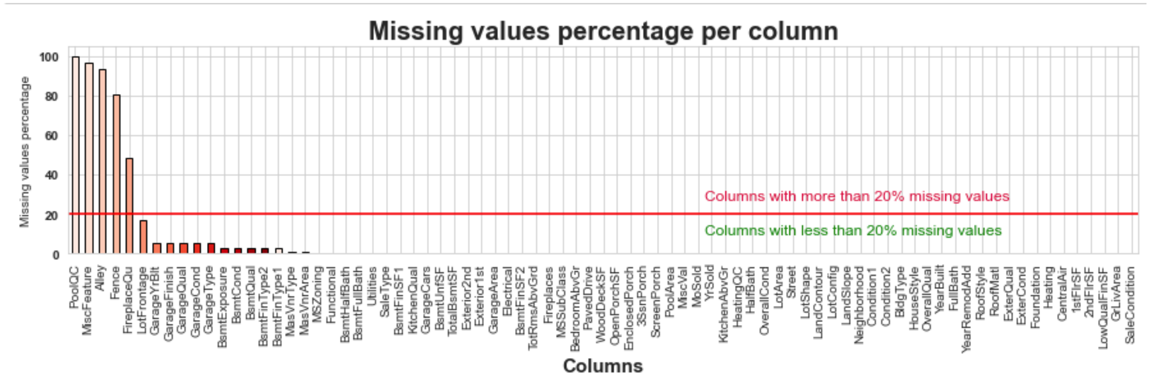
Using python's requests module we downloaded the house prediction dataset after which we loaded it into a csv file. From the Pandas module, we used the read_csv method and loaded it onto a dataframe. We segregated the dataset into train and test based on the fact that the sale price was available for half the records.
We used the dataframe corr method and noticed that the better the quality of the house is, the higher the price will be. Another factor was the living area.
We used matplotlib's scatter method to plot this data on a graph and this showed us some outliers.
We then used a matplotlib further to create a histogram and a probability plot.



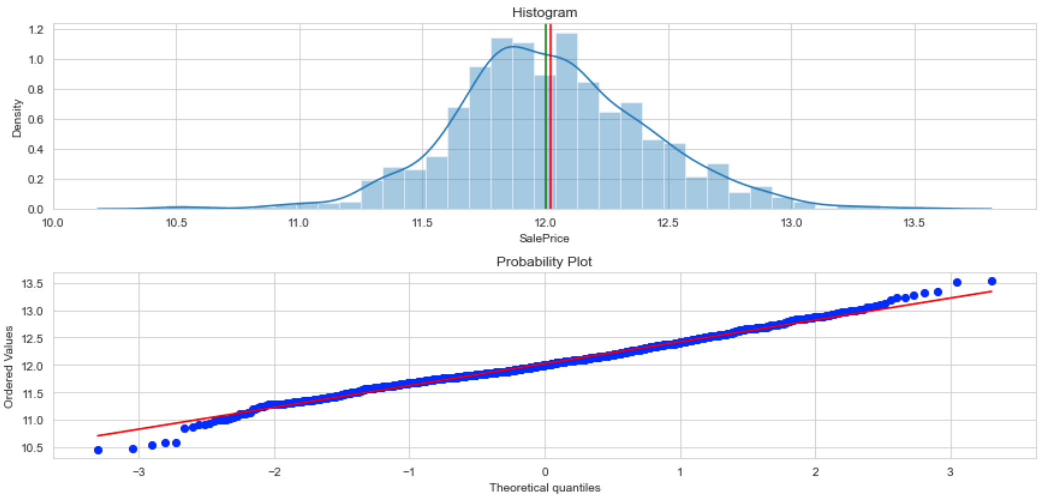
In the first diagram, we decided to use histogram because we thought it's the easiest to read. The red line in this histogram represents the mean and the black line represents the median. What we observed after this is that the sale price isn't normally distributed.

In accordance with the above probability plot, the red line would have been the points for y axis if the points were normally distributed. The distance between the red line and blue dots is minimal in the middle area but there are still some outliers (specially along the top right and bottom left areas).

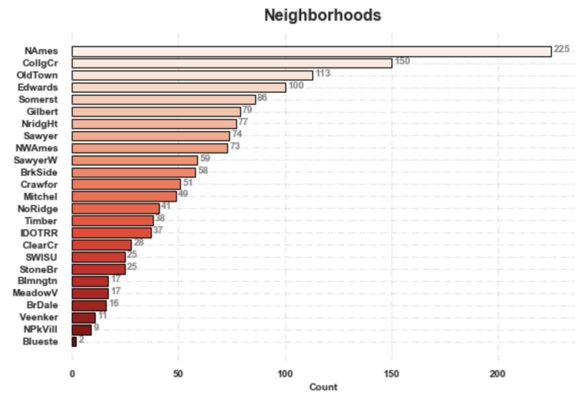
We also worked on null value analysis since some of the data were missing but not insignificant. We started with creating a function that would get all the columns, the number of null values in each column and their percentage.
Once we had this data, we wanted to see which of these null values should be considered for sale price prediction. So, we plotted a graph showing columns and percentage of missing values in these columns. We chose to ignore any columns with less than 20% missing values.

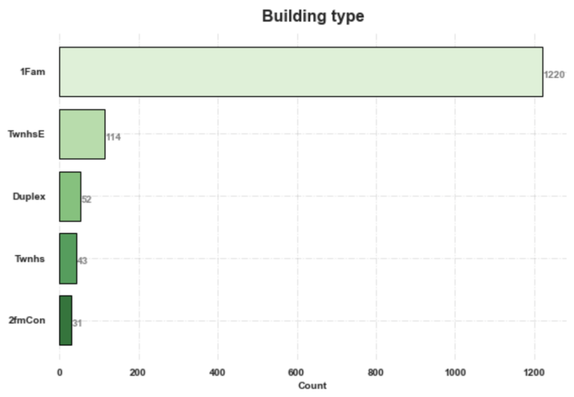


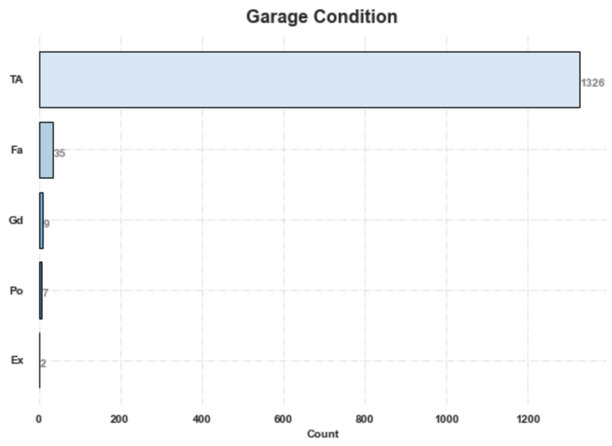
We had two categories to deal with, numerical categories (living area, number of bedrooms etc.) and non-numerical categories (overall quality, neighborhood etc.). We segregated this data and started working on numerical categories first. We created a heatmap for all columns with non-numerical values to identify how the factors were affecting the Sale Price. To ensure that the Sale Price in histogram and probability plot showed a normal distribution, we performed a log transformation and plotted those graphs again.



Next we worked on the columns with non-numerical values and plotted graphs for Neighborhood frequency, dwelling type and garage condition







We also did the same null analysis for these columns and ignored the columns where more than 80% of the values were missing considering that they would not be able to bring much value to the prediction. Next we started filling the NA values for the other columns. In the description for 'Functional Feature', it is given that Typ refers to the Typical Functionality, therefore replace null values in the 'Functional' feature with 'Typ'. Replace the null values for columns ('Electrical', 'KitchenQual', 'Exterior1st', 'Exterior2nd', 'SaleType') with their mode. We are replacing with mode because all of these features are non numerical variable, we can't take the mean nor the median. In order to fill the null values for MSZoing, we will take into account the MSSubClass feature. This is because the type of dwelling in a given area largely affects the zone of the area. In order to fill the null values for LotFrontage(Linear feet of street connected to property), we fill it with the median of values that is grouped by Neighborhood. For the other columns with 'NA' as value, we assumed that they represent the absence of that feature for a particular house, so we replaced those null values with None. None for categorical feature and 0 for numerical feature.

As a part of the extension, we created new features (TotalSF, Total_Bathrooms, Total_porch_sf and YearsSinceRemodel)

```
def new_features(features):

    features['HasPool'] = features['PoolArea'].apply(lambda x: 1 if x > 0 else 0)
    features['Has2ndFloor'] = features['2ndFlrSF'].apply(lambda x : 1 if x > 0 else 0)
    features['HasGarage'] = features['GarageArea'].apply(lambda x: 1 if x > 0 else 0)
    features['HasBsmt'] = features['TotalBsmtSF'].apply(lambda x: 1 if x > 0 else 0)
    features['HasFireplace'] = features['Fireplaces'].apply(lambda x: 1 if x > 0 else 0)


    features['TotalSF'] = features['TotalBsmtSF'] + features['1stFlrSF'] + features['2ndFlrSF']

    features['Total_Bathrooms'] = (features['FullBath'] + (0.5 * features['HalfBath']) +
                                   features['BsmtFullBath'] + (0.5 * features['BsmtHalfBath']))

    features['Total_porch_sf'] = (features['OpenPorchSF'] + features['3SsnPorch'] +
                                  features['EnclosedPorch'] + features['ScreenPorch'])

    # Add years since remodel
    features['YearsSinceRemodel'] = features['YrSold'].astype(int) - features['YearRemodAdd'].astype(int)

    return features
```

For our final steps, we decided to train and test using different regression models.
Since the dataset contains categorical variables, we created dummies of the categorical features for an ease in modelling using pandas.get_dummies() function.
We use the predict function to predict the labels of the testing dataset.
Further, split the dataset into training and testing using the train_tests_split() function.
We import mean_squared_error() to measure the average of error squares and mean absolute error for the sum of absolute errors over the length of observations / predictions
We then create an instance of the lasso regression implementation. The regulation parameter is 0.1, we fit the lasso model and create the score.
We create an instance of ridge regression so that the model does not overfit the data. The regulation parameter here is 0.1 as well for the purposes of comparing.

```
Train Result:
================================================
Root Mean Squared Error: 0.16500929690462973
_______________________________________________
Mean Squared Error: 0.027228068064960247
_______________________________________________
Mean Absolute Error:
0.11185440762252903
_______________________________________________

Test Result:
================================================
Root Mean Squared Error: 0.23135552818701113
_______________________________________________
Mean Squared Error: 0.05352538042269091
_______________________________________________
Mean Absolute Error:
0.11392704143460534
_______________________________________________
```

```
Train Result:
================================================
Root Mean Squared Error: 0.08757192777593702
_______________________________________________
Mean Squared Error: 0.007668842534393929
_______________________________________________
Mean Absolute Error:
0.06251920602232108
_______________________________________________

Test Result:
================================================
Root Mean Squared Error: 0.21215370319613475
_______________________________________________
Mean Squared Error: 0.04500919377983363
_______________________________________________
Mean Absolute Error:
0.09884469589691164
_______________________________________________
```

Conclusion

This project definitely challenged in ways we didn’t think. We used various online sources (websites) along with the textbook to help us understand a few concepts. We think that this project has really helped us understand the way Python is used to analyze data. Regression analysis was the most complex and challenging according to us but we were able to work it through. The way the class was delivered is what we appreciate the most and think is a good starting point for implementing python in our careers.