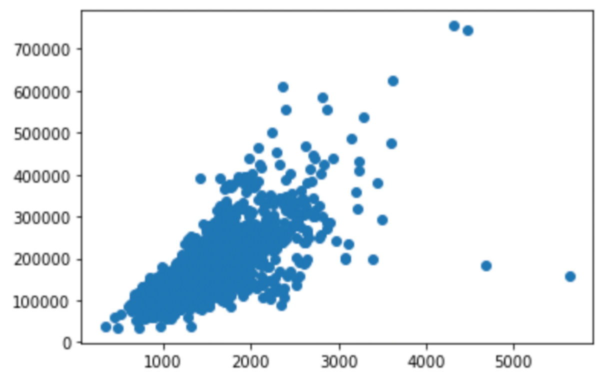
<u>House Price Predictor Tool</u>

By: Stephen Boorman & Weiran Zhang

One of the biggest financial decisions one can make would be to buy a property. Given the rate at which housing prices are rising in the U.S., it makes sense that the buyer would desire more tools to determine what “value” they are getting for their money. So the purpose of this project is to take a data set that incorporates historical sale price into a row of other information about the property. Other columns included, but not limited to: pool, fence, square footage, utilities, rooms above grade, garage, bathrooms, bedrooms, and overall quality. Given the abundance of information, it is important to parse out what is actually important and will have a big impact on sale price and what would be considered “noise.” If given the entire data set and we plotted square footage over the sale price, to assume that bigger = more expensive, we would end up with a scatter plot like this:

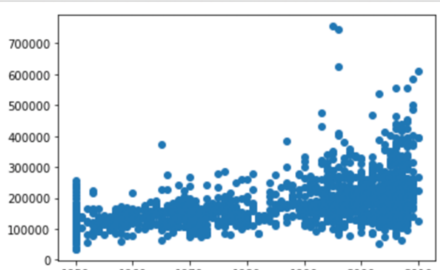
Figure 1:



While there is some correlation to be made, there are too many outliers that may skew the analysis. What can be learned from this is that a more sophisticated model is imperative.

Figure 2:

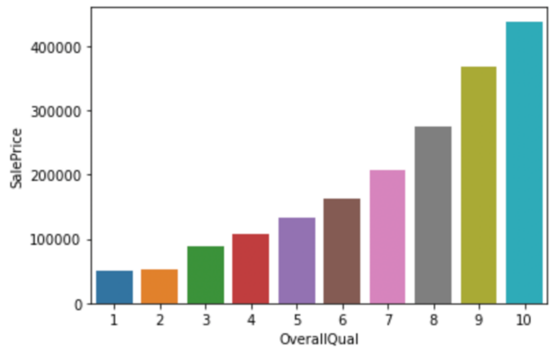
```
#YearRemodAdd has 0.5 correlation
#there is relation between them but it's getting weak compare with previous one
#we set 0.5 as threshold
plt.scatter(data['YearRemodAdd'], data["SalePrice"]);
```



In the above plot, we have the Year remodeled over the sale price. We chose the variable of year remodeled as there was a 0.5 threshold indicated a strong correlation of sale price to when the house was remodeled. Even in this plot, there are still apparent outlier data that has the potential to skew the results. However, this can be mitigated with overall amount of more “expected” data.

Next, we plotted the overall quality of the property correlated with the sale price. See Figure 3. Here we can see the strongest correlation yet between overall quality and sale price. Using intuition, this makes sense as a nicer house will see for more money. Figure 3 proves that.

Figure 3:



Finally, using linear regression, we were able to create a table that would, with high confidence, show us what columns would be most correlated with sale price. See Figure 4. With and emphasis on the column P>|t|, we can find values that are below 0.5 which will tell us with high confidence there is a strong correlation. These columns would include: OverallQual, YearBuilt, YearRemodAdd, 1stFlrSF, GRLivArea, TolRimsAbvGrd, and GarageCars. It was at this point we decided to remove GarageArea, FullBath, and TotRmsAbvGrd from our training data because the P>|t| was above 0.5 which is a low confidence.

Figure 4

Dep. Variable:	SalePrice	**R-squared:**	0.807
Model:	OLS	**Adj. R-squared:**	0.805
Method:	Least Squares	**F-statistic:**	361.2
Date:	Sat, 06 Aug 2022	**Prob (F-statistic):**	1.04e-300
Time:	11:18:11	**Log-Likelihood:**	285.23
No. Observations:	876	**AIC:**	-548.5
Df Residuals:	865	**BIC:**	-495.9
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.3728	0.762	3.116	0.002	0.878	3.868
OverallQual	0.0971	0.007	13.977	0.000	0.083	0.111
YearBuilt	0.0021	0.000	7.034	0.000	0.002	0.003
YearRemodAdd	0.0022	0.000	5.729	0.000	0.001	0.003
TotalBsmtSF	6.679e-05	2.39e-05	2.797	0.005	1.99e-05	0.000
1stFlrSF	6.506e-05	2.83e-05	2.296	0.022	9.44e-06	0.000
GrLivArea	0.0002	2.45e-05	7.091	0.000	0.000	0.000
FullBath	-0.0114	0.015	-0.742	0.458	-0.042	0.019
TotRmsAbvGrd	0.0132	0.007	1.990	0.047	0.000	0.026
GarageCars	0.0707	0.019	3.742	0.000	0.034	0.108
GarageArea	5.586e-05	6.73e-05	0.830	0.407	-7.62e-05	0.000

Omnibus:	600.704	**Durbin-Watson:**	2.014
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	20850.665
Skew:	-2.622	**Prob(JB):**	0.00

Given this information, we can now train our ML with accurate data to make accurate predictions of sale price given these factors.

Figure 5

Dep. Variable:	SalePrice	**R-squared:**	0.806
Model:	OLS	**Adj. R-squared:**	0.804
Method:	Least Squares	**F-statistic:**	514.3
Date:	Sat, 06 Aug 2022	**Prob (F-statistic):**	1.00e-303
Time:	11:31:24	**Log-Likelihood:**	282.88
No. Observations:	876	**AIC:**	-549.8
Df Residuals:	868	**BIC:**	-511.6
Df Model:	7		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	2.6322	0.722	3.644	0.000	1.214	4.050
OverallQual	0.0958	0.007	13.856	0.000	0.082	0.109
YearBuilt	0.0020	0.000	7.050	0.000	0.001	0.003
YearRemodAdd	0.0021	0.000	5.672	0.000	0.001	0.003
TotalBsmtSF	6.542e-05	2.35e-05	2.788	0.005	1.94e-05	0.000
1stFlrSF	6.742e-05	2.82e-05	2.395	0.017	1.22e-05	0.000
GrLivArea	0.0002	1.58e-05	12.860	0.000	0.000	0.000
GarageCars	0.0835	0.011	7.838	0.000	0.063	0.104

Omnibus:	607.488	**Durbin-Watson:**	2.010
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	21163.383
Skew:	-2.666	**Prob(JB):**	0.00
Kurtosis:	26.482	**Cond. No.**	4.36e+05

Here in figure 5 we can see that with a smaller number of rows (data fields) we can with a higher correlation train our data to be more accurate. This is what we ultimately used as training data and eventually as test data set as well. In figure 6, we executed on the columns previously mentioned to the sale price of a property on the train data and then we were able to get a score of how accurate it was, which was a score of about 0.8057.

We used our training data set to create a model and then used on the test data set, which led to the results of 0.8376 with predicting housing price. Now that our model has been trained and we know we have a degree of accuracy (~83%), we created some fake data to determine how much our fake house would be worth. These are somewhat realistic values, such as a built in 1971, 1800sqft, 2 car garage, overall quality of 8, etc. With these parameters, our model determined our fake house to be worth $220,727. When looking back at our dataset, this result seems to fall in line with what we would "expect."

Next, we took our trained model onto a new housing data set that we acquired through another github user's repository. We ran into a few issues. First, our trained model had certain parameters the new dataset did not include. This would be GarageCars. Also, our dataset used in the training was within a certain expected range, this was for sqfootage and overallcondidtion. Once the trained data came across data that was out of range, it made it unreliable and gave inaccurate results. See below. With more time we could probably clean the dataset to be a perfect fit with expected data with the columns that were not initially included.

Also, this means that our initial training data could use more data beyond the ranges initially input to deal with a larger range of data.

In the below figure, the top data set is the test data and the bottom table is the training data set. This show that they are not congruent by looking at the column names.

Out[33]:

	condition	yr_built	yr_built	sqft_basement	sqft_above	sqft_living	GarageCars
count	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.0
mean	6.818859	1971.005136	1971.005136	291.509045	1788.390691	2079.899736	0.0
std	1.301486	29.373411	29.373411	442.575043	828.090978	918.440897	0.0
min	2.000000	1900.000000	1900.000000	0.000000	290.000000	290.000000	0.0
25%	6.000000	1951.000000	1951.000000	0.000000	1190.000000	1427.000000	0.0
50%	6.000000	1975.000000	1975.000000	0.000000	1560.000000	1910.000000	0.0
75%	8.000000	1997.000000	1997.000000	560.000000	2210.000000	2550.000000	0.0
max	10.000000	2015.000000	2015.000000	4820.000000	9410.000000	13540.000000	0.0

In [36]:

```
x2_train.describe()
```

Out[36]:

	const	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	GarageCars
count	876.0	876.000000	876.000000	876.000000	876.000000	876.000000	876.000000	876.000000
mean	1.0	6.111872	1971.731735	1985.300228	1068.331050	1178.423516	1528.402968	1.772831
std	0.0	1.403276	30.062258	20.548035	466.678829	403.301209	540.803065	0.742038
min	1.0	1.000000	1872.000000	1950.000000	0.000000	372.000000	438.000000	0.000000
25%	1.0	5.000000	1954.000000	1968.000000	793.000000	884.750000	1145.500000	1.000000
50%	1.0	6.000000	1973.000000	1994.000000	1007.500000	1096.500000	1473.000000	2.000000
75%	1.0	7.000000	2001.000000	2004.000000	1325.000000	1429.250000	1780.250000	2.000000
max	1.0	10.000000	2010.000000	2010.000000	6110.000000	4692.000000	5642.000000	4.000000

In the end, we ended up with a score of:

```
-3.4378697928392024
```

This tells use that as predicted, with data that does not perfectly sync up with the training data, the predictor tool is inaccurate. Some ways we could mitigate it would be to retrain the tool with data that is similar to the new test data or found a way to better clean the new data. These are great directions for the next iteration of the housing price predictor tool.