House Prices: Advanced Regression Techniques

Predict sales prices and practice feature engineering, RFs, and gradient boosting

December 2020
CSPB 3022 Final Project
Vidya Giri

Kaggle ID: vidyagi
Github Repository: https://github.com/vidgi/predictingHousePrices

Table of Contents

1. Problem Description

The Ames Housing dataset was compiled by Dean De Cock for use in data science education and was provided for use in this Kaggle prediction competition. There are 79 explanatory variables describing aspects of residential homes in Ames, Iowa.

The problem posed is to find an effective method to use the given explanatory variables to predict the final price of each home. To do so, we will need to understand the dataset given with the basic structure, conduct exploratory data analysis, build a predictive model that can be trained, and test our model's effectiveness and final results.

Importing libraries and train and test data from csv files

In [2]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm
from scipy import stats
%matplotlib inline
```

In [3]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

Obtaining dimension/structure of data

In [4]:
```
print('shapes:')
print(train.shape)
train.head()
```

```
shapes:
(1460, 81)
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl

5 rows × 81 columns

learning more about the data columns and their data types

In [5]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     1452 non-null   object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int64
```

```
 37  BsmtUnfSF      1460 non-null   int64  
 38  TotalBsmtSF    1460 non-null   int64  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1460 non-null   int64  
 44  2ndFlrSF       1460 non-null   int64  
 45  LowQualFinSF   1460 non-null   int64  
 46  GrLivArea      1460 non-null   int64  
 47  BsmtFullBath   1460 non-null   int64  
 48  BsmtHalfBath   1460 non-null   int64  
 49  FullBath       1460 non-null   int64  
 50  HalfBath       1460 non-null   int64  
 51  BedroomAbvGr   1460 non-null   int64  
 52  KitchenAbvGr   1460 non-null   int64  
 53  KitchenQual    1460 non-null   object 
 54  TotRmsAbvGrd   1460 non-null   int64  
 55  Functional     1460 non-null   object 
 56  Fireplaces     1460 non-null   int64  
 57  FireplaceQu    770 non-null    object 
 58  GarageType     1379 non-null   object 
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object 
 61  GarageCars     1460 non-null   int64  
 62  GarageArea     1460 non-null   int64  
 63  GarageQual     1379 non-null   object 
 64  GarageCond     1379 non-null   object 
 65  PavedDrive     1460 non-null   object 
 66  WoodDeckSF     1460 non-null   int64  
 67  OpenPorchSF    1460 non-null   int64  
 68  EnclosedPorch  1460 non-null   int64  
 69  3SsnPorch      1460 non-null   int64  
 70  ScreenPorch    1460 non-null   int64  
 71  PoolArea       1460 non-null   int64  
 72  PoolQC         7 non-null      object 
 73  Fence          281 non-null    object 
 74  MiscFeature    54 non-null     object 
 75  MiscVal        1460 non-null   int64  
 76  MoSold         1460 non-null   int64  
 77  YrSold         1460 non-null   int64  
 78  SaleType       1460 non-null   object 
 79  SaleCondition  1460 non-null   object 
 80  SalePrice      1460 non-null   int64  
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

Overall, from looking at these properties of the data set we can see that there are 1460 rows of train data. From inspecting the data, we can see that there might be some columns that may have missing/not applicable data that we may need to filter out in the cleaning process.

Additionaly, for this dataset there are 81 columns:

- House ID (ID)
- 36 are quantitative attributes (float64 and int64 data types)
- 43 are categorical attributes (object data types)
- Sale Price of the House (SalePrice)

Quantitative attributes: MSSubClass, LotFrontage, LotArea, OverallQual, OverallCond, YearBuilt, YearRemodAdd, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea, BsmtFullBath, BsmtHalfBath, FullBath, HalfBath, BedroomAbvGr, KitchenAbvGr, TotRmsAbvGrd, Fireplaces, GarageYrBlt, GarageCars, GarageArea, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, MiscVal, MoSold, YrSold

Qualitative attributes: MSZoning, Street, Alley, LotShape, LandContour, Utilities, LotConfig, LandSlope, Neighborhood, Condition1, Condition2, BldgType, HouseStyle, RoofStyle, RoofMatl, Exterior1st, Exterior2nd, MasVnrType, ExterQual, ExterCond, Foundation, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, Heating, HeatingQC, CentralAir, Electrical, KitchenQual, Functional, FireplaceQu, GarageType, GarageFinish, GarageQual, GarageCond, PavedDrive, PoolQC, Fence, MiscFeature, SaleType, SaleCondition

2. Exploratory Data Analysis

For exploratory data analysis, I will show the cleaning process, visualizations, histograms, and derive my plan of analysis.

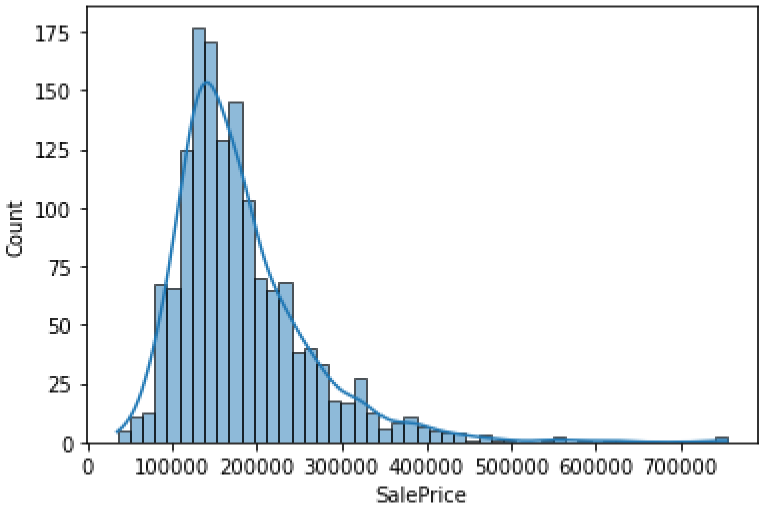
Starting off, I wanted to plot the overall distribution of sales prices to see the general range of prices in our dataset. This does not appear to be a normal distribution and we will need to take into account our various features and attributes to undertstand their effect on the prices

In [6]:

```
sns.histplot(train['SalePrice'],kde=True);
train['SalePrice'].describe()
```

Out[6]:

```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```



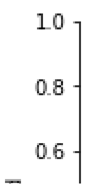
To visualize the quantitative variables that might be significant further, I pairplotted the variables against SalePrice

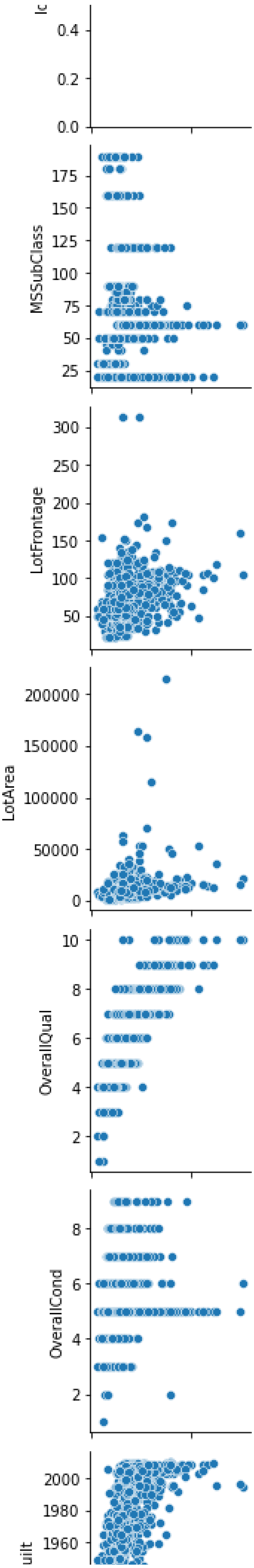
In [7]:

```
sns.pairplot(train,dropna=True, x_vars=['SalePrice'])
```

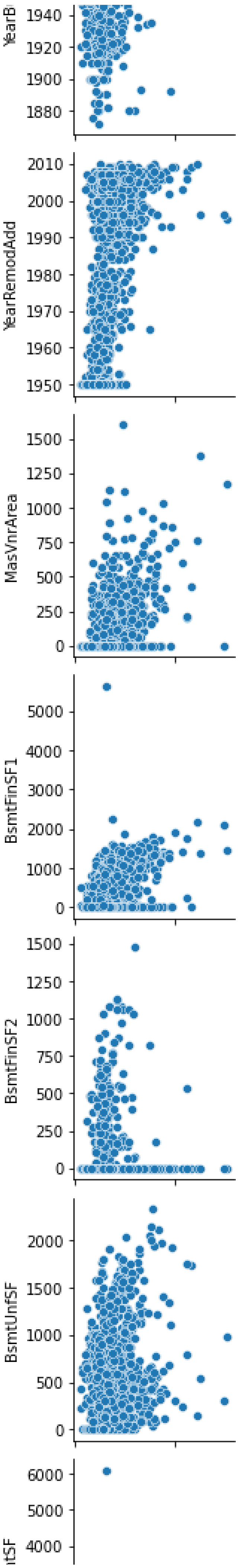
Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x7ff805526520>
```

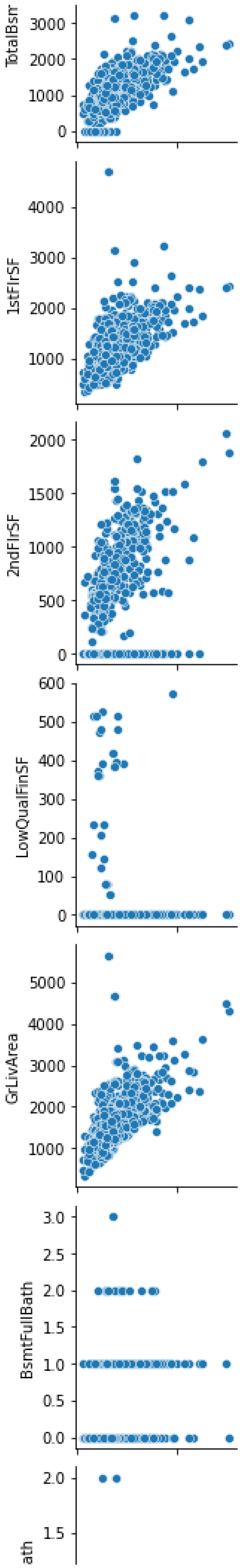


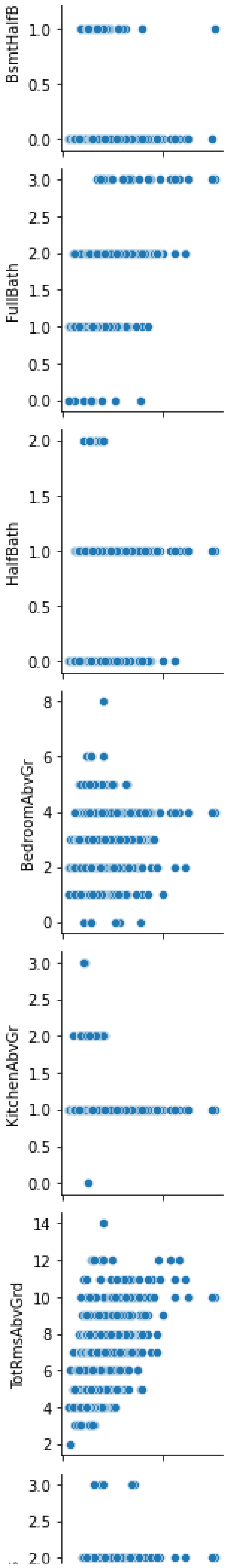


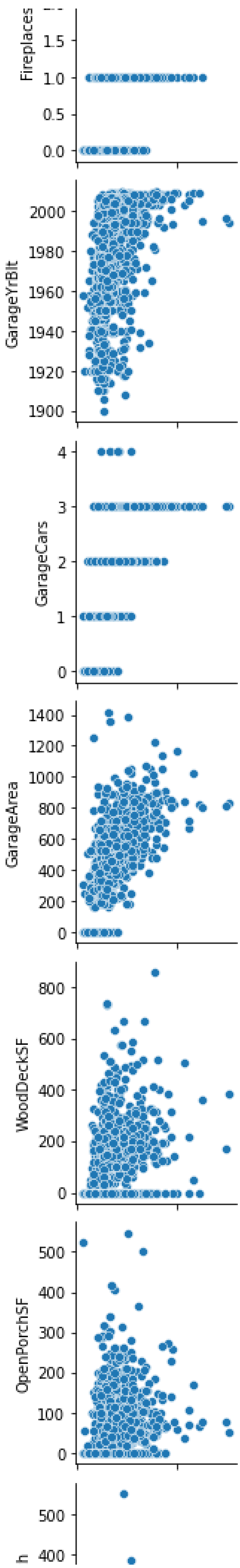
0.4
0.2
0.0
MSSubClass
175
150
125
100
75
50
25
LotFrontage
300
250
200
150
100
50
LotArea
200000
150000
100000
50000
0
OverallQual
10
8
6
4
2
OverallCond
8
6
4
2
2000
1980
1960



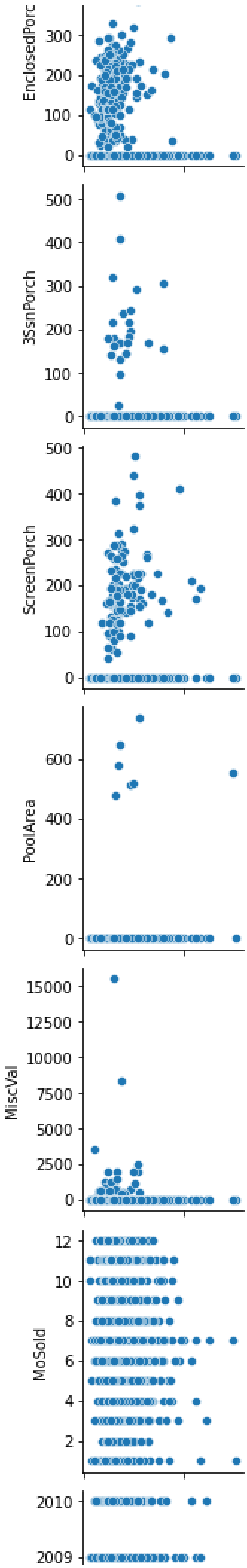
YearB
1940
1920
1900
1880
YearRemodAdd
2010
2000
1990
1980
1970
1960
1950
MasVnrArea
1500
1250
1000
750
500
250
0
BsmtFinSF1
5000
4000
3000
2000
1000
0
BsmtFinSF2
1500
1250
1000
750
500
250
0
BsmtUnfSF
2000
1500
1000
500
0
tSF
6000
5000
4000


TotalBsm
0
1000
2000
3000
1stFlrSF
1000
2000
3000
4000
2ndFlrSF
0
500
1000
1500
2000
LowQualFinSF
0
100
200
300
400
500
600
GrLivArea
1000
2000
3000
4000
5000
BsmtFullBath
0.0
0.5
1.0
1.5
2.0
2.5
3.0
ath
1.5
2.0

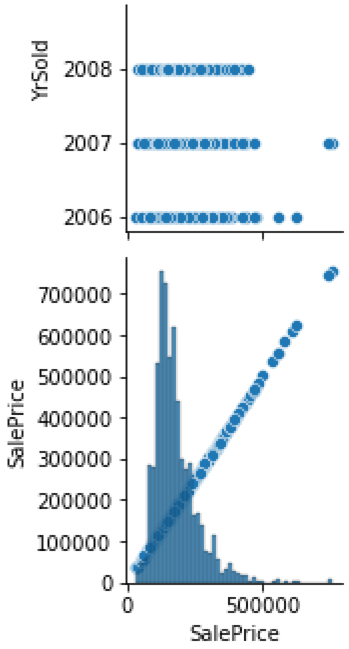

BsmtHalfB
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd



Fireplaces
1.5
1.0
0.5
0.0
GarageYrBlt
2000
1980
1960
1940
1920
1900
GarageCars
4
3
2
1
0
GarageArea
1400
1200
1000
800
600
400
200
0
WoodDeckSF
800
600
400
200
0
OpenPorchSF
500
400
300
200
100
0
500
400



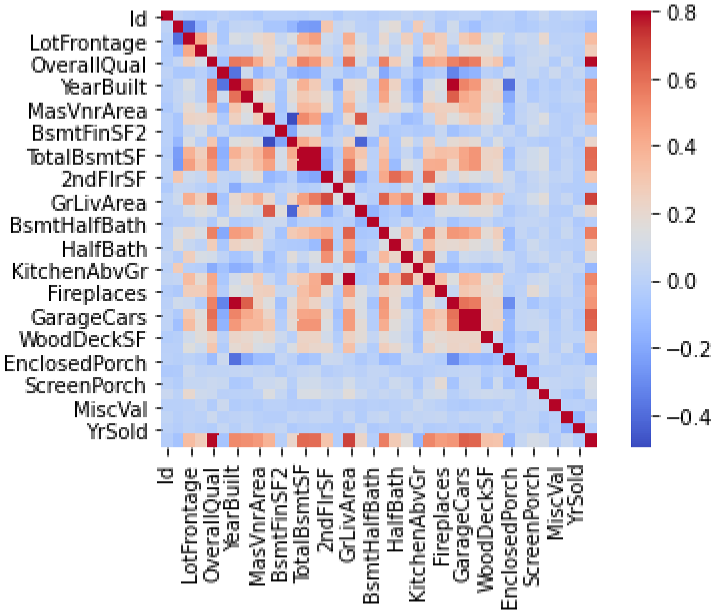
EnclosedPorc
3SsnPorch
ScreenPorch
PoolArea
MiscVal
MoSold



I also drew a heatmap to see any other variables that might show a significant relationship.

In [8]:
```
corrmat = train.corr()
sns.heatmap(corrmat,cmap = 'coolwarm', vmax=.8, square=True)
```

Out[8]: <AxesSubplot:>



We can also check the correlation of the variables

In [9]:
```
x= train.corr()
x = x[x['SalePrice']>0.5]
x['SalePrice']
```

Out[9]:
```
OverallQual      0.790982
YearBuilt        0.522897
YearRemodAdd     0.507101
TotalBsmtSF      0.613581
1stFlrSF         0.605852
GrLivArea        0.708624
FullBath         0.560664
TotRmsAbvGrd     0.533723
GarageCars       0.640409
GarageArea       0.623431
```

```
SalePrice        1.000000
Name: SalePrice, dtype: float64
```

Some of the variables above appear to be correlated with SalesPrice as seen above where the plots generally seem to be positively correlated. The variables also appear to colinear since the variables are highly correlated, so we can use multi-linear regression. This can also be seen from the correlation table above.

Overall from this EDA we can start to narrow down some key variables that can come in handy for our model for now:

- OverallQual
- YearBuilt
- YearRemodAdd
- TotalBsmtSF
- GrLivArea
- FullBath
- TotsRmsAbvGrd
- GarageCars
- GarageArea

3. Model Architecture

From the EDA, I can start to create a multilinear model using the variables I have derived above. First, I will construct a mulit-linear model without interaction terms and predict the SalesPrice column on the other columns and print out the summary table.

Then, based on this, I will then construct a multi-linear model with interactions that are statistically significant at the $p = 0.05$ level. I will start with full interactions and then eliminate interactions are do not meet the $p = 0.05$ threshold to refine the model.

First, I will start off with all the variables with: `SalePrice~OverallQual*GrLivArea*GarageCars*TotalBsmtSF*FullBath*YearBuilt` This will enable me to check all of the interactions and narrow down the variables as needed.

In [20]:
```
import statsmodels.formula.api as smf
import statsmodels.api as sm

model_trn = smf.ols(formula='SalePrice~OverallQual*GrLivArea*GarageCars*Total
# model_trn.summary() output omitted for report readability
```

This is a good start and I will now narrow down the variables even further by checking the individual probabilities to eliminate variables.

In [11]:
```
model_trn = smf.ols(formula='SalePrice~OverallQual+GrLivArea+GarageCars+Total
model_trn.summary()
```

Out[11]:

OLS Regression Results

Dep. Variable:	SalePrice	**R-squared:**	0.772
Model:	OLS	**Adj. R-squared:**	0.771
Method:	Least Squares	**F-statistic:**	546.8
Date:	Tue, 08 Dec 2020	**Prob (F-statistic):**	0.00
Time:	00:40:52	**Log-Likelihood:**	-17463.

No. Observations:	1460	**AIC:**	3.495e+04
Df Residuals:	1450	**BIC:**	3.500e+04
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.159e+06	1.29e+05	-8.978	0.000	-1.41e+06	-9.06e+05
OverallQual	1.923e+04	1186.103	16.216	0.000	1.69e+04	2.16e+04
GrLivArea	53.7230	4.158	12.920	0.000	45.567	61.879
GarageCars	1.053e+04	3051.640	3.451	0.001	4543.945	1.65e+04
TotalBsmtSF	28.9907	2.902	9.992	0.000	23.299	34.682
FullBath	-6408.3114	2685.771	-2.386	0.017	-1.17e+04	-1139.899
YearBuilt	258.0525	50.347	5.125	0.000	159.292	356.813
YearRemodAdd	294.6839	63.791	4.620	0.000	169.552	419.816
GarageArea	16.9253	10.314	1.641	0.101	-3.307	37.158
TotRmsAbvGrd	41.7519	1121.858	0.037	0.970	-2158.886	2242.390

Omnibus:	485.052	**Durbin-Watson:**	1.980
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	60868.329
Skew:	-0.439	**Prob(JB):**	0.00
Kurtosis:	34.620	**Cond. No.**	4.42e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.42e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

From this, I can see that I will modify to the formula of `SalePrice~OverallQual*GrLivArea*GarageCars+TotalBsmtSF+FullBath+YearBuilt` since I want to reduce the number of interactions to less than 0.05

In [12]:
```
model_trn = smf.ols(formula='SalePrice~OverallQual*GrLivArea*GarageCars+Total
model_trn.summary()
```

Out[12]:

OLS Regression Results

Dep. Variable:	SalePrice	**R-squared:**	0.818
Model:	OLS	**Adj. R-squared:**	0.816
Method:	Least Squares	**F-statistic:**	590.9
Date:	Tue, 08 Dec 2020	**Prob (F-statistic):**	0.00
Time:	00:40:52	**Log-Likelihood:**	-17301.
No. Observations:	1460	**AIC:**	3.463e+04
Df Residuals:	1448	**BIC:**	3.469e+04
Df Model:	11		

Covariance Type:	nonrobust

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.075e+06	1.2e+05	-8.979	0.000	-1.31e+06	-8.4e+05
OverallQual	2.54e+04	4318.728	5.881	0.000	1.69e+04	3.39e+04
GrLivArea	134.0080	17.511	7.653	0.000	99.659	168.357
OverallQual:GrLivArea	-18.7141	2.895	-6.464	0.000	-24.393	-13.035
GarageCars	1.61e+04	1.12e+04	1.433	0.152	-5932.181	3.81e+04
OverallQual:GarageCars	-2848.4830	1917.586	-1.485	0.138	-6610.028	913.062
GrLivArea:GarageCars	-41.7223	7.810	-5.342	0.000	-57.043	-26.402
OverallQual:GrLivArea:GarageCars	8.8774	1.173	7.570	0.000	6.577	11.178
TotalBsmtSF	23.9166	2.598	9.206	0.000	18.821	29.013
FullBath	-2980.1027	2414.626	-1.234	0.217	-7716.641	1756.436
YearBuilt	233.1520	45.135	5.166	0.000	144.614	321.690
YearRemodAdd	277.7551	57.379	4.841	0.000	165.200	390.310

Omnibus:	886.909	**Durbin-Watson:**	2.000
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	90081.683
Skew:	-1.920	**Prob(JB):**	0.00
Kurtosis:	41.289	**Cond. No.**	3.74e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.74e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

Finally, I will reduce the variables to OverallQual, GrLivArea, and YearBuilt.

In [13]:
```
model_trn = smf.ols(formula='SalePrice~OverallQual*GrLivArea*YearBuilt', data=
model_trn.summary()
```

Out[13]:

OLS Regression Results

Dep. Variable:	SalePrice	**R-squared:**	0.768
Model:	OLS	**Adj. R-squared:**	0.767
Method:	Least Squares	**F-statistic:**	688.0
Date:	Tue, 08 Dec 2020	**Prob (F-statistic):**	0.00
Time:	00:40:53	**Log-Likelihood:**	-17476.
No. Observations:	1460	**AIC:**	3.497e+04
Df Residuals:	1452	**BIC:**	3.501e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]

Intercept	4.031e+06	7.71e+05	5.231	0.000	2.52e+06	5.54e+06
OverallQual	-8.337e+05	1.19e+05	-7.004	0.000	-1.07e+06	-6e+05
GrLivArea	-1639.8361	471.256	-3.480	0.001	-2564.252	-715.421
OverallQual:GrLivArea	283.7362	64.683	4.387	0.000	156.855	410.618
YearBuilt	-2029.3788	392.755	-5.167	0.000	-2799.806	-1258.951
OverallQual:YearBuilt	427.4303	60.382	7.079	0.000	308.985	545.875
GrLivArea:YearBuilt	0.8282	0.240	3.447	0.001	0.357	1.300
OverallQual:GrLivArea:YearBuilt	-0.1386	0.033	-4.220	0.000	-0.203	-0.074

Omnibus:	609.126	**Durbin-Watson:**	1.966
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	48838.306
Skew:	-1.034	**Prob(JB):**	0.00
Kurtosis:	31.258	**Cond. No.**	1.71e+10

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.71e+10. This might indicate that there are
strong multicollinearity or other numerical problems.

4. Result and Analysis

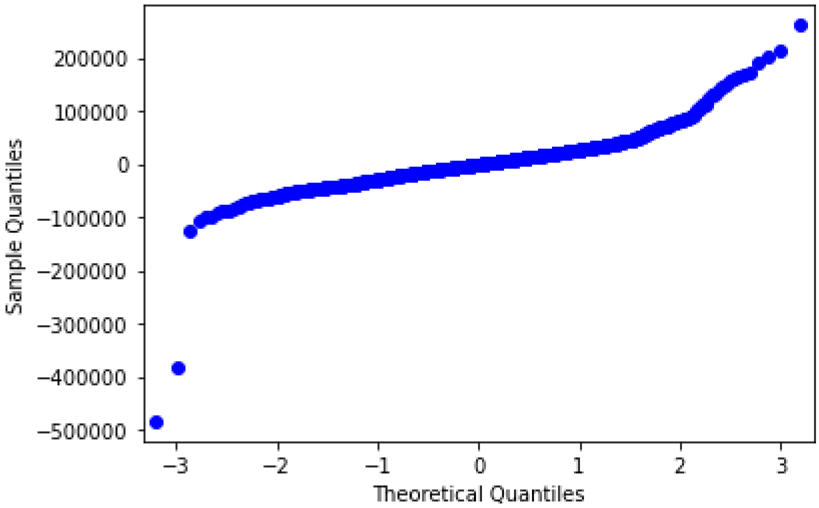
To check my results, I have created and errorcheck function and plotted the residuls to spot any outliers that we might want to remove to get a more accurate fit.

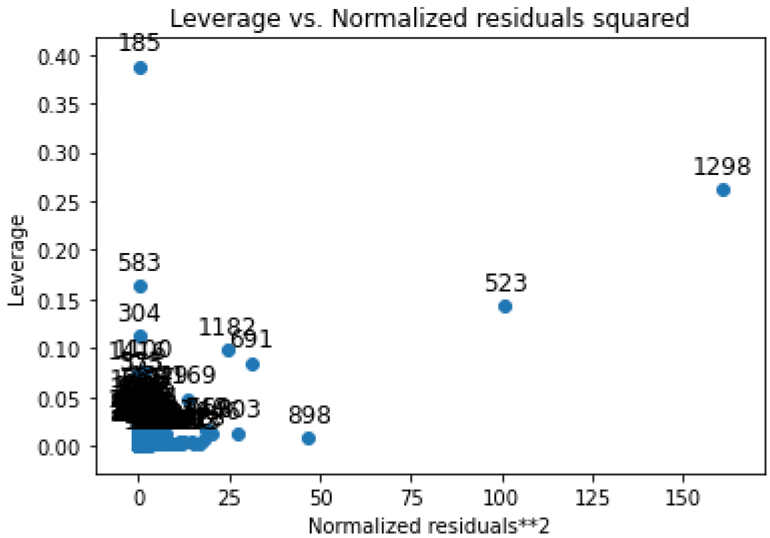
In [14]:
```
def errorcheck(model):
    predicted = model_trn.predict(train)
    actual= train['SalePrice']
    actual = np.log(actual)
    predicted = np.log(predicted)
    error=np.sqrt(np.sum(np.square(actual-predicted))/len(actual))
    return error

errorcheck(model_trn)
```

Out[14]:
```
0.191912740182196
```

Right now, we have an error of 0.178 and this can be reduced by removing some strategic points.

In [15]:
```
sm.qqplot(model_trn.resid);
sm.graphics.plot_leverage_resid2(model_trn, alpha=0.05);
```





From this graph, we can see that outliers 185, 523, and 1298 might have high leverage over our model and can be removed as outliers. Let's inspect the data to ensure.

In [16]:
```
train.iloc[[185,523,1298]]
```

Out[16]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCon
185	186	75	RM	90.0	22950	Pave	NaN	IR2	
523	524	60	RL	130.0	40094	Pave	NaN	IR1	
1298	1299	60	RL	313.0	63887	Pave	NaN	IR3	

3 rows × 81 columns

I decided to remove the data points and check if this improved my errorcheck result

In [17]:
```
train.drop(1298, inplace=True)
train.drop(523, inplace=True)
train.drop(186, inplace=True)
```

So, I fit the model to the updated data frame and checked the error.

In [18]:
```
model_trn = smf.ols(formula='SalePrice~OverallQual*GrLivArea*YearBuilt', data=
model_trn.summary()
errorcheck(model_trn)
```

Out[18]: 0.17816112832468034

And it showed an improvement from 0.178 to 0.166! Now we can use this updated model to create predictions on test.csv. I added some simple error checking to ensure that results were being generated for each data point:

In [19]:
```
ypred = model_trn.predict(test)
print(ypred)

df = pd.read_csv('test.csv')
output = pd.DataFrame({'Id': df.Id,
                       'SalePrice': ypred})
output.to_csv('submission.csv', index=False)

df = pd.read_csv('submission.csv')
df = df.dropna()
if(len(df)!=len(ypred)):
    print('Error with model prediction, submission.csv generated with errors'
else:
    print('submission.csv generated')
```

```
0       118635.166827
1       150251.984093
2       154839.672112
3       183756.773356
4       203848.387351
            ...
1454    115610.353089
1455    115610.353089
1456    129568.974067
1457    135841.693880
1458    241077.112494
Length: 1459, dtype: float64
submission.csv generated
```

This generated the csv that I was able to submit to the Kaggle competition and got a score of 0.18716. This was not as high as some of the more complex models on Kaggle but, it was interesting to read more about what other people did to solve this problem and analyze the given data.

5. Final Conclusions

Overall, it was very interesting working through this problem and using skills from this class to generate a model that can be used for practical purposes. I am sure well-known websites like Zillow also use similar datasets to provide estimates of houses as well as other items such as car insurance, other types of insurance, and other price estimators and it is interesting to think about how you can take more features into account to build a more accurate model. Although my model did not quite reach the top of the Kaggle leaderboard, I am interested to learn more in the future and use other forms of modelling, tools, and frameworks to build better estimators and more robust models.