Ames Housing

Data Modeling and Analysis

By
Gip, Nutt, Wow, Ment

Problem Statement

In 2011, the Ames Assessor's Office received several requests from the current homeowners asking for advices on remodelling their houses. Homeowners wanted to know

1. Which features of the houses should they focus on to make the highest return from sales?

2. What is the expected sale prices of those houses given certain housing features?

As an employee in the Ames Assessor's Office, I am tasked with creating regression models based on the Ames Housing Dataset, which contained the assessed values for individual residential properties sold in Ames, IA from 2006 to 2010. This model will predict the price of a house at sale and indicate what features have the most effect on the house price. I will use R2 and RMSE as evaluation metrics for my models.

Data set contains information from the Ames Assessor's Office used in computing assessed values for individual residential properties sold in Ames, IA from 2006 to 2010

Training Data , 80 features and 2051 observations

Id	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	...	Screen Porch	Pool Area	Pool QC	Fence	Misc Feature	Misc Val	Mo Sold	Yr Sold	Sale Type	SalePrice
109	533352170	60	RL	NaN	13517	Pave	NaN	IR1	Lvl	...	0	0	NaN	NaN	NaN	0	3	2010	WD	130500
544	531379050	60	RL	43.0	11492	Pave	NaN	IR1	Lvl	...	0	0	NaN	NaN	NaN	0	4	2009	WD	220000
153	535304180	20	RL	68.0	7922	Pave	NaN	Reg	Lvl	...	0	0	NaN	NaN	NaN	0	1	2010	WD	109000
318	916386060	60	RL	73.0	9802	Pave	NaN	Reg	Lvl	...	0	0	NaN	NaN	NaN	0	4	2010	WD	174000
255	906425045	50	RL	82.0	14235	Pave	NaN	IR1	Lvl	...	0	0	NaN	NaN	NaN	0	3	2010	WD	138500

Data dictionary to get type of cols (Numerical or Category)

http://jse.amstat.org/v19n3/decock/DataDocumentation.txt

```
BsmtFin SF 1 (Continuous): Type 1 finished square feet

BsmtFinType 2   (Ordinal): Rating of basement finished area (if multiple types)

       GLQ      Good Living Quarters
       ALQ      Average Living Quarters
       BLQ      Below Average Living Quarters
       Rec      Average Rec Room
       LwQ      Low Quality
       Unf      Unfinshed
       NA       No Basement
```

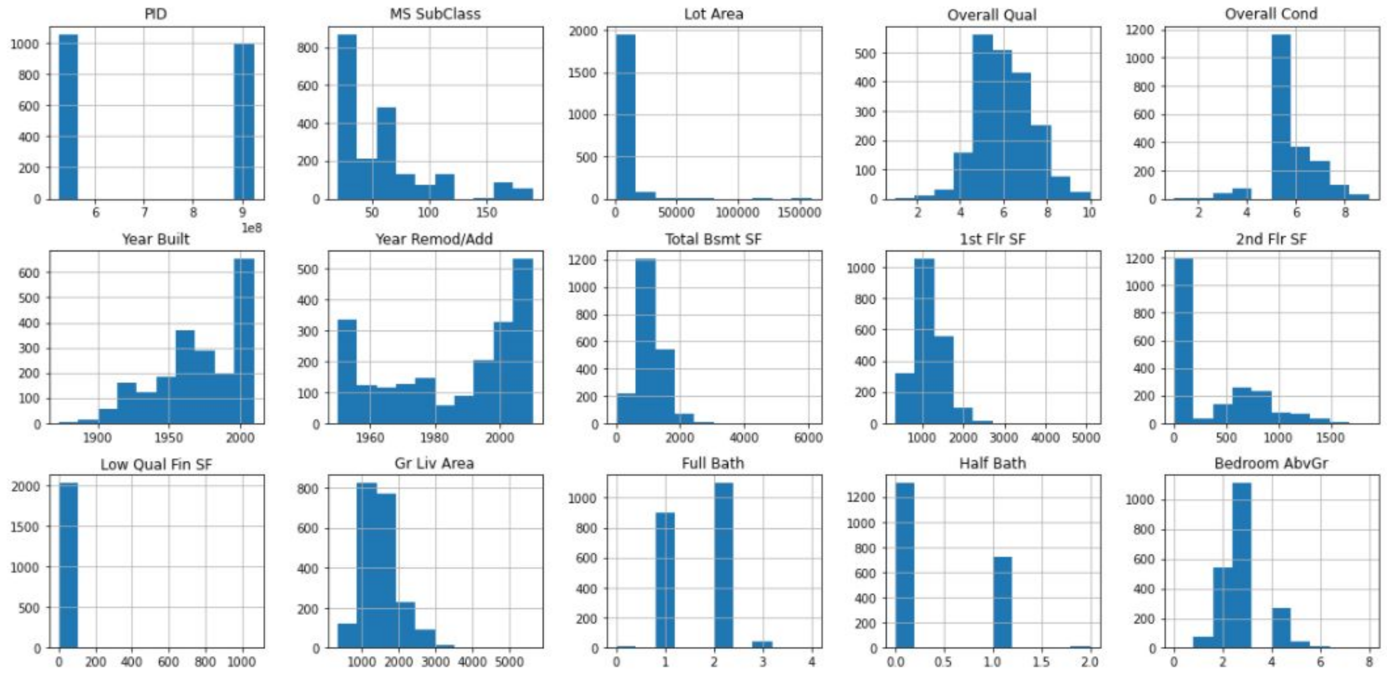
```
Misc Feature (Nominal): Miscellaneous feature not covered in other categories

       Elev     Elevator
       Gar2     2nd Garage (if not described in garage section)
       Othr     Other
       Shed     Shed (over 100 SF)
       TenC     Tennis Court
       NA       None
```

EDA histogram

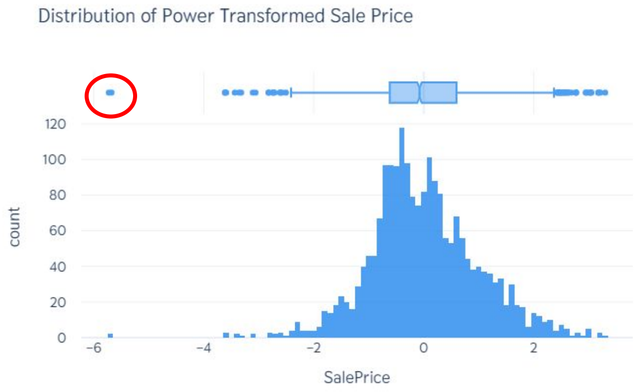
Data Visualization

```
#create histogram from the train dataset
train.hist(figsize=(20,20));
```



Data Preprocessing

Target Variable : SalePrice



Original

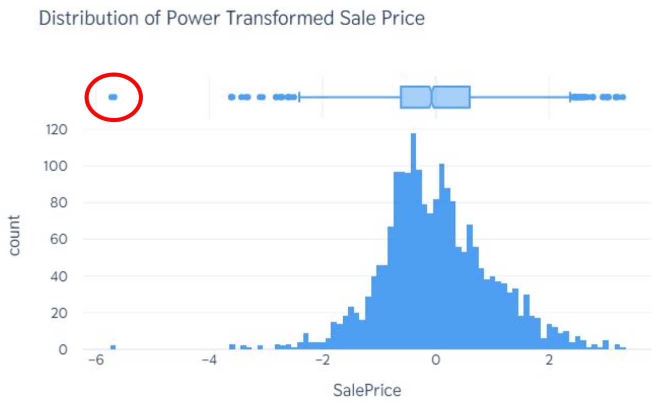
- Right skewed
- Straight line won't fit well

Box-Cox Transformed

- Almost normally distributed
- 2 outliers on the left

Data Preprocessing

Target Variable : SalePrice



Box-Cox Transformed

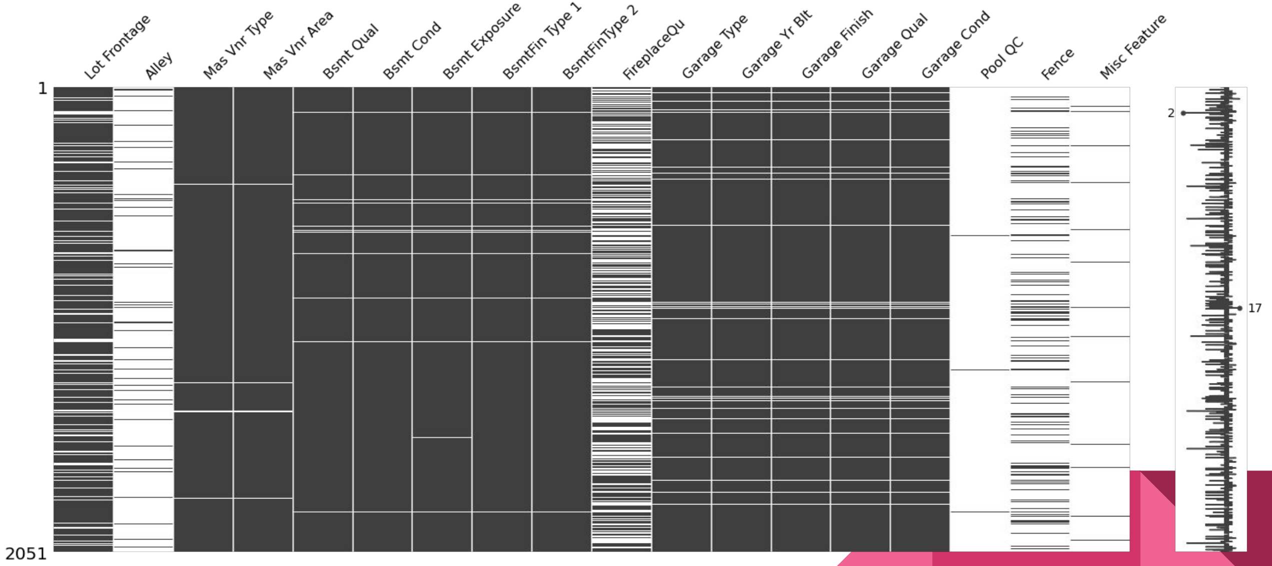
- Almost normally distributed
- 2 outliers on the left

Outlier Removed

- Normally distributed
- Straight line can fit

Data missing

```
Pool QC          2042
Misc Feature     1986
Alley            1911
Fence            1651
FireplaceQu      1000
Lot Frontage      330
Garage Finish     114
Garage Cond       114
Garage Qual       114
Garage Yr Blt     114
Garage Type       113
Bsmt Exposure      58
BsmtFinType 2      56
BsmtFin Type 1     55
Bsmt Cond          55
Bsmt Qual          55
Mas Vnr Type       22
Mas Vnr Area       22
Bsmt Half Bath      2
Bsmt Full Bath      2
Garage Cars         1
Garage Area         1
Bsmt Unf SF         1
BsmtFin SF 2        1
Total Bsmt SF       1
BsmtFin SF 1        1
dtype: int64
```



Filling Missing Value Strategy

- **Numerical** : Replace with **0**.
  - Rationale :
    - Areas or numbers of absent house attributes are equivalent to 0.
- **Nominal** : Replace with string **"None"**.
  - Rationale :
    - Types of absent house attributes are equivalent to a new special type "None".
- **Ordinal** : Replace with **mode**, the most frequent existing value.
  - Rationale :
    - The most frequent existing value or mode is the easiest to implement.
    - Good statistical representation of the feature.

Dropping Outliers

The **Special Notes section in data dictionary**

,we are **advised to drop "any houses with more than 4000 square feet from the dataset."**

```
#check the dataset shape before dropping outliers
train.shape
```

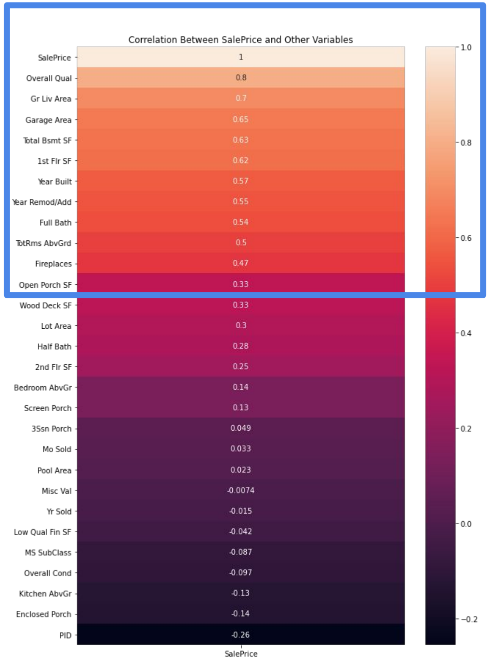
```
(2051, 80)
```

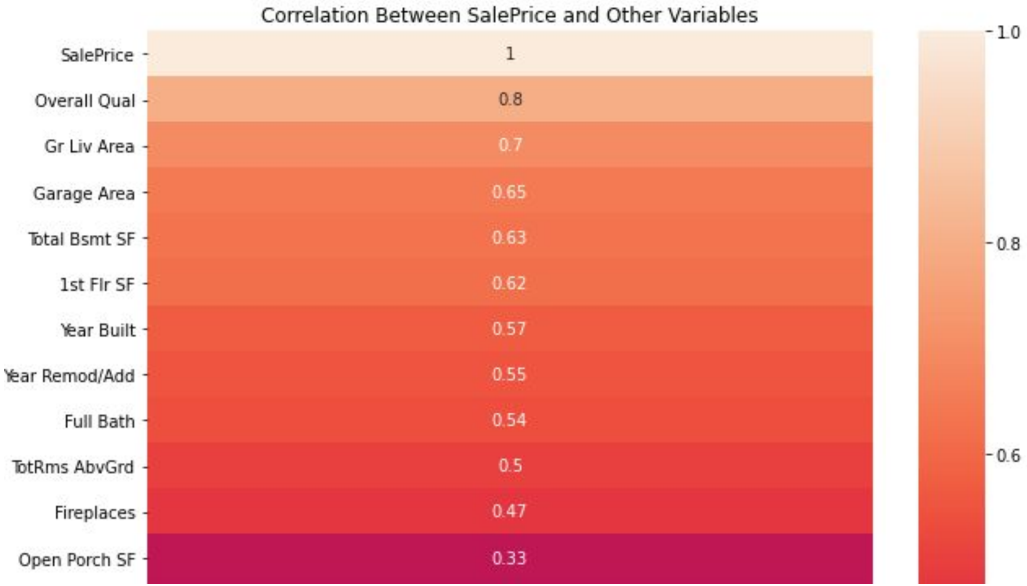
```
#drop the outliers
train = train[train['Gr Liv Area'] < 4000]
```

```
#check the dataset shape after dropping outliers
train.shape
```

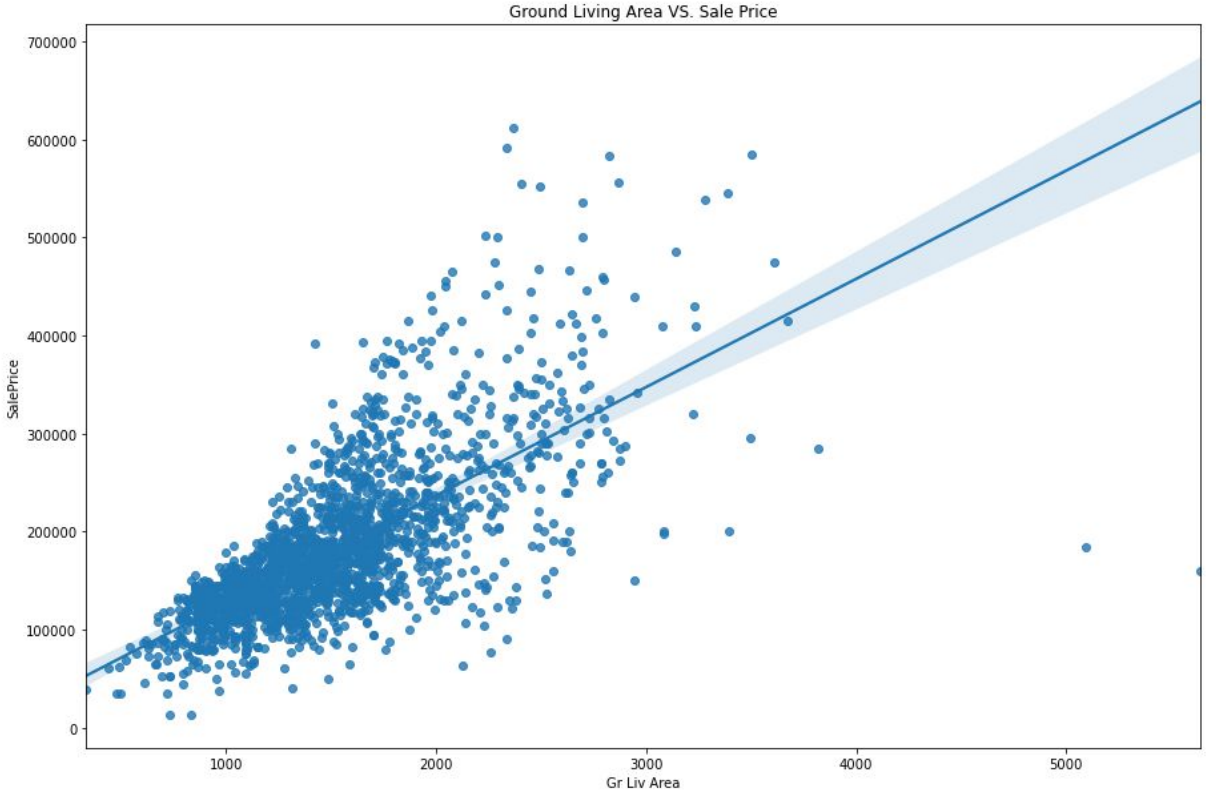
```
(2049, 80)
```

High correlation features

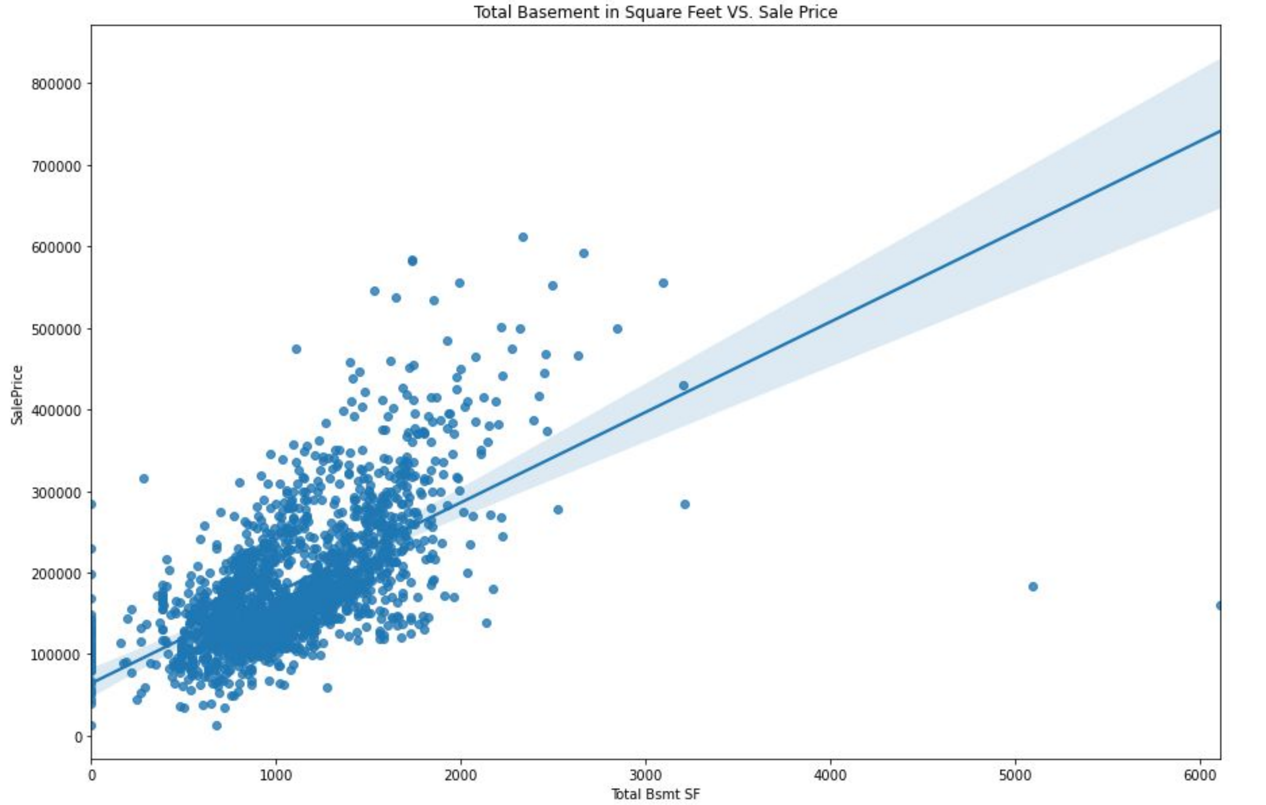




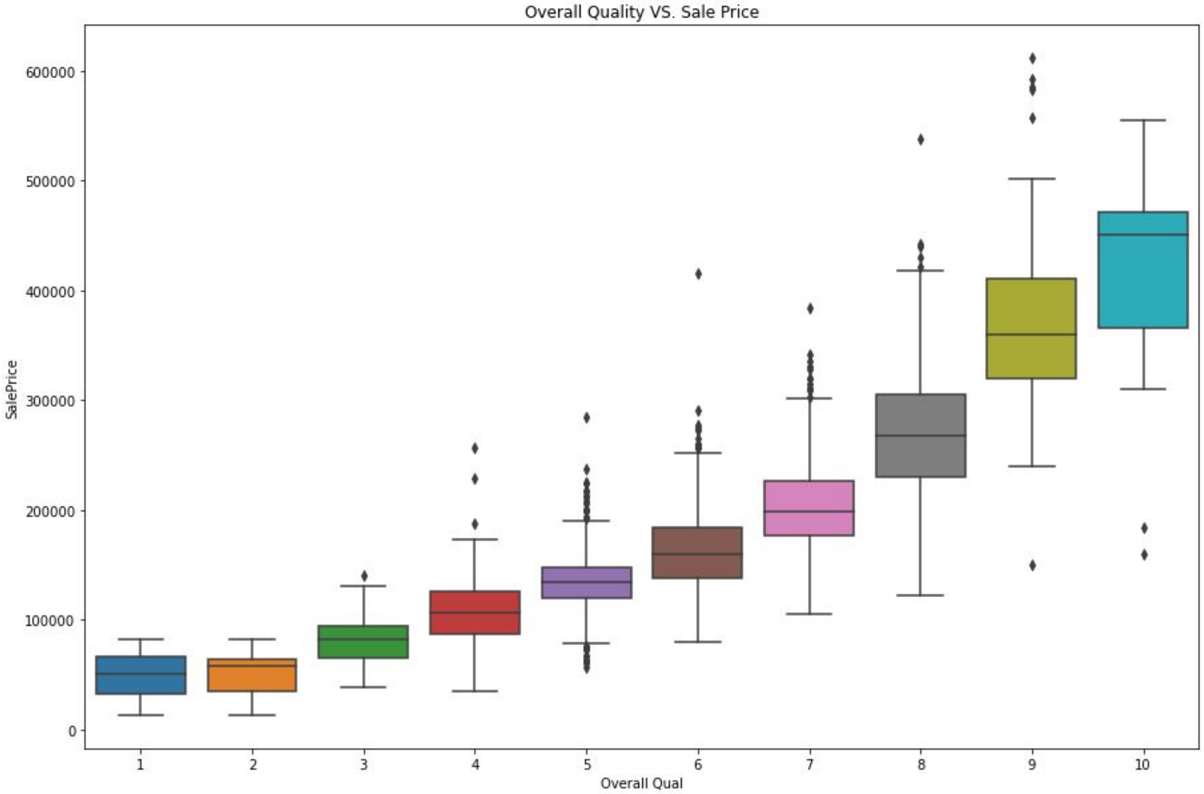
Scatter plot on High correlation features[numerical] (samples)



Scatter plot on High correlation features[numerical]
(samples)



Boxplot on High correlation features [Category] (samples)



Numerical -- ready for training in model

Ordinal -- need to encode to be Numerical

Nominal -- need to convert to Numerical (get dummies)

Ordinal columns -- Map to discrete values by order

```
#Replace Lot Shape data in both datasets
train['Lot Shape'].replace({'Reg': 4, 'IR1': 3, 'IR2': 2, 'IR3': 1}, inplace = True)
test['Lot Shape'].replace({'Reg': 4, 'IR1': 3, 'IR2': 2, 'IR3': 1}, inplace = True)
```

```
#Replace Utilities data in both datasets
train['Utilities'].replace({'AllPub': 4, 'NoSewr': 3, 'NoSeWa': 2}, inplace = True)
test['Utilities'].replace({'AllPub': 4, 'NoSewr': 3}, inplace = True)
```

```
#Replace Land Slope data in both datasets
train['Land Slope'].replace({'Gtl': 3, 'Mod': 3, 'Sev': 2}, inplace = True)
test['Land Slope'].replace({'Gtl': 3, 'Mod': 3, 'Sev': 2}, inplace = True)
```

```
#Replace Exter Qual data in both datasets
train['Exter Qual'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2}, inplace = True)
test['Exter Qual'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2}, inplace = True)
```

```
#Replace Exter Cond data in both datasets
train['Exter Cond'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1}, inplace = True)
test['Exter Cond'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1}, inplace = True)
```

```
#Replace Bsmt Qual data in both datasets
train['Bsmt Qual'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0}, inplace = True)
test['Bsmt Qual'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0}, inplace = True)
```

```
#Replace Bsmt Cond data in both datasets
train['Bsmt Cond'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0}, inplace = True)
test['Bsmt Cond'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0}, inplace = True)
```

```
#Replace Bsmt Exposure data in both datasets
train['Bsmt Exposure'].replace({'Gd': 4, 'Av': 3, 'Mn': 2, 'No': 1, 'NA': 0}, inplace = True)
test['Bsmt Exposure'].replace({'Gd': 4, 'Av': 3, 'Mn': 2, 'No': 1, 'NA': 0}, inplace = True)
```

```
#Replace BsmtFin Type 1 data in both datasets
train['BsmtFin Type 1'].replace({'GLQ': 6, 'ALQ': 5, 'BLQ': 4, 'Rec': 3, 'LwQ': 2, 'Unf': 1, 'NA': 0}, inplace = True)
test['BsmtFin Type 1'].replace({'GLQ': 6, 'ALQ': 5, 'BLQ': 4, 'Rec': 3, 'LwQ': 2, 'Unf': 1, 'NA': 0}, inplace = True)
```

Ordinal columns -- Map to discrete values by order

In [51]:
```
#Replace Exter Qual data in both datasets
train['Exter Qual'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2}, inplace = True)
test['Exter Qual'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2}, inplace = True)
```

In [52]:
```
#Replace Exter Cond data in both datasets
train['Exter Cond'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1}, inplace =
True)
test['Exter Cond'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1}, inplace = T
rue)
```

In [53]:
```
#Replace Bsmt Qual data in both datasets
train['Bsmt Qual'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0}, in
place = True)
test['Bsmt Qual'].replace({'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0}, inp
lace = True)
```

Missing columns

➡ Before we run our model, we need to check if both train and test data have the same number of columns.

```
In [72]: #check train dataset shape
         train.shape

Out[72]: (2049, 233)

In [73]: #check test dataset shape
         test.shape

Out[73]: (879, 224)
```

Missing columns

➡ Since there are more columns in train dataset, we will find out which columns are different in each datasets and create those columns in each datasets accordingly.

In [74]:
```
#find the missing columns in train dataset
missing_cols_train = list(set(test.columns) - set(train.columns))
```

In [76]:
```
#find the missing columns in test dataset
missing_cols_test = list(set(train.columns) - set(test.columns))
```

Missing columns

- So we fix the missing columns by create a new column with a value of 0.

In [78]:
```
#input missing columns in train dataset
for column in missing_cols_train:
    train[column] = 0
```

In [81]:
```
#input missing columns in test dataset
for column in missing_cols_test:
    test[column] = 0
```

Nominal columns -- use get_dummies to convert to Numerical

➡ Drop first dummy columns in both dataset

```
In [84]: #drop the first dummy columns in both datasets
         for dummy in dummy_columns:
             dummy_list = [col for col in train.columns if dummy in col]
             train.drop(columns = dummy_list[0], inplace = True)
             test.drop(columns = dummy_list[0], inplace = True)
```

```
In [85]: #check shape of train dataset
         train.shape

Out[85]: (2049, 219)
```

```
In [86]: #check shape of test dataset
         test.shape

Out[86]: (879, 219)
```

Models

Create predictor and target variable. Standardize the predictors

```
In [87]: #create X and y variables
         X = train.drop(columns = ['SalePrice'], axis = 1)
         features = list(X.columns)
         y = train['SalePrice']
```

```
In [88]: #evaluate train/test split
         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

```
In [89]: #scale and fit the data
         ss = StandardScaler()
         ss.fit(X_train)

Out[89]: StandardScaler()
```

```
In [90]: X_scaled_train = ss.fit_transform(X_train)
         X_scaled_test = ss.transform(X_test)
```

Baseline Model

```
In [92]: #calculate the mean of target variable
         y_avg = y_test.mean()
         y_avg

Out[92]: 181457.50877192983

In [93]: #create baseline prediction
         baseline_preds = [y_avg for i in y]

In [94]: #calculate baseline R2 scores
         r2_score(y,baseline_preds)

Out[94]: -7.361433107533344e-08

In [154]: #calculate baseline RMSE
          np.sqrt(mean_squared_error(y, baseline_preds))

Out[154]: 79276.56390558237
```

Linear Regression Model

In [86]:
```
#instantiate and fit linear model
lr = LinearRegression()
lr.fit(X_scaled_train,y_train)
```

Out[86]: LinearRegression()

In [113]:
```
lr.score(X_scaled_train, y_train)
```

Out[113]: 0.9305945750188552

Cross-validate the R^2 of an ordinary linear regression model with 5 cross-validation folds.

In [87]:
```
#R2 on unseen data from linear regression
lr_cv_scores = cross_val_score(lr, X_scaled_train, y_train,cv=5)
lr_cv_scores.mean()
```

Out[87]: -6.203213758152376e+20

In [88]:
```
#RMSE on unseen data from linear model
lr_rmse_cv_scores = np.abs(cross_val_score(lr, X_scaled_train, y_train,cv= 5, scoring='neg_root_mean_squared_error'))
lr_rmse_cv_scores.mean()
```

Out[88]: 1069556644863256.8

Ridge Regression Model

In [157]:
```
#find out which alpha to choose
ridge_alphas = np.logspace(0, 5, 100)
optimal_ridge = RidgeCV(alphas=ridge_alphas, cv=5)
optimal_ridge.fit(X_scaled_train, y_train)
print(optimal_ridge.alpha_)
```

```
236.4489412645407
```

Cross-validate the R^2 of a ridge regression model with 5 cross-validation folds.

In [100]:
```
#Ridge R2 score on train data
ridge = Ridge(alpha=optimal_ridge.alpha_)
ridge_cv_scores = cross_val_score(ridge, X_scaled_train, y_train, cv=5)
ridge_cv_scores.mean()
```

Out[100]:
```
0.8985322226538128
```

In [101]:
```
#Ridge RMSE on train data
ridge_rmse_cv_scores = np.abs(cross_val_score(ridge, X_scaled_train, y_train,cv= 5, scoring='neg_root_mean_squared_error'))
ridge_rmse_cv_scores.mean()
```

Out[101]:
```
25097.438480598066
```

Lasso Regression Model

```
In [102]: #find out which alpha to choose
optimal_lasso = LassoCV(n_alphas=100, cv=5)
optimal_lasso.fit(X_scaled_train, y_train)
print(optimal_lasso.alpha_)
```

```
551.5890972965998
```

Cross-validate the R^2 of lasso regression model with 5 cross-validation folds.

```
In [161]: #Lasso R2 score on train data
lasso = Lasso(alpha=optimal_lasso.alpha_)
lasso_cv_scores = cross_val_score(lasso, X_scaled_train, y_train, cv=5)
lasso_cv_scores.mean()
```

```
Out[161]: 0.9022523108724902
```

```
In [162]: #Lasso RMSE score on train data
lasso_rmse_cv_scored = np.abs(cross_val_score(lasso, X_scaled_train, y_train,cv= 5, scoring='neg_root_mean_squared_error'))
lasso_rmse_cv_scored.mean()
```

```
Out[162]: 24641.47957217064
```

Elastic Net Model

n [95]:

```python
#find out the optimal alpha and l1 ratio
l1_ratios = np.linspace(0.01, 1.0, 25)
optimal_enet = ElasticNetCV(l1_ratio=l1_ratios, n_alphas=100, cv=5, verbose=1)
optimal_enet.fit(X_scaled_train, y_train)
print(optimal_enet.alpha_)
print(optimal_enet.l1_ratio_)
```

```
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
.....................
```

```
551.5890972965998
1.0
```

Model Fitting and Evaluation

```
In [96]: #lasso model fitting
         lasso.fit(X_scaled_train,y_train)

Out[96]: Lasso(alpha=551.5890972965998)
```

- Training R2 (Lasso) 0.9227
- Training CV R2 (Lasso) 0.9022
- Testing R2 (Lasso) 0.9174

- Training RMSE (Lasso) 22016.3
- Training CV RMSE (Lasso) 24641.4
- Testing RMSE (Lasso) 22841.0

Top 10 features of a house that affect the sale price

```
#create a dataframe showing features with the highest absolute coefficient; best features of house
lasso_coef_df = pd.DataFrame({'column': features, 'coef' : lasso.coef_, 'abs_coef' : np.abs(lasso.coef_)})
lasso_coef_df[lasso_coef_df['abs_coef']>0].sort_values(by ='abs_coef', ascending = False).head(10)
```

	column	coef	abs_coef
26	Gr Liv Area	24403.522928	24403.522928
6	Overall Qual	12044.692404	12044.692404
17	BsmtFin SF 1	9514.874598	9514.874598
21	Total Bsmt SF	7456.266607	7456.266607
8	Year Built	7261.307266	7261.307266
105	Neighborhood_NridgHt	7260.168983	7260.168983
11	Exter Qual	6339.364107	6339.364107
111	Neighborhood_StoneBr	5735.470491	5735.470491
207	Sale Type_New	5709.213039	5709.213039
33	Kitchen Qual	5195.567923	5195.567923

Interpretation:

For a square feet increase in **ground living area**, we expect the house sale price to increase by $22,403

For a unit increase in **overall material quality**, we expect the house sale price to increase by $12,044

For a square feet increase in **finished basement area type 1**, we expect the house sale price to increase by $9,514

Conclusion & Recommendation

- Lasso regression is the best model among the three because it is best at predicting unseen data and giving the least estimated error
- Ground Living area, Overall quality, Basement size and type are the most important determination of sale price of the house in Ames city
- Two good neighborhood that might be a good investment are Northridge Heights and Stone Brook
- The model can be used on other city housing data as well since top features in the model should be similar to other cities
- We can improve the model by adding interaction terms and removing more outliers.