Building a Regression Model to Predict House Prices

Newton Tran

INTRODUCTION

Regression analysis is a type of supervised machine-learning technique used to estimate the relationships between the dependent variable (also known as the response, outcome, or label) and one or more independent variables (also known as the regressors, predictors, or explanatory variables/features). One of the most common forms of regression is linear regression, which assumes that the independent and dependent variables are linearly related, meaning the latter changes at a constant rate as the former changes. Moreover, in linear regression, the best-fit line is the straight line (or hyperplane when dealing with two or more independent variables) that most accurately represents the relationship between the independent and dependent variables. Specifically, the goal is to find such a line that minimizes the error (or the difference) between the observed data points and the predicted values, whereby this line helps us predict the dependent variable for new, unseen data. Linear regression is important because:

- It is simple and interpretable
- It helps predict future outcomes based on past data
- It serves as a basis for other models like logistic regression and neural networks
- It is computationally efficient
- It is widely used
- It allows us to analyze the relationships between variables and how they influence each other

Another important aspect is correlation, which describes the statistical relationship between between two or more variables. It measures the degree to which changes in one variable are associated with changes in another variable. Most importantly though, correlation does not imply causation, meaning just because two variables are correlated does not necessarily mean that changes in one variable cause changes in the other. There are three cases for correlation, which include:

- Positive: as one variable increases, the other tends to increase as well
- Negative: as one variable decreases, the other tends to decrease as well
- Zero correlation: there is no consistent relationship between the variables

There are several ways to measure the correlation coefficient. One of the most common ways is through Pearson's correlation, which measures the linear relationship between two sets of data. When dealing with ordinal data, Spearman's rank correlation and Kendall's rank correlation are useful. They measure the strength and direction of the monotonic (strictly increasing or strictly decreasing) relationship between two ranked (ordinal) variables. Numerically, 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation.

Some examples of regression include:

- Predicting exam scores based on time spent on studying
- Explaining the elevated levels of lead in children based on a set of related variables such as income, accessibility to safe drinking water, and the presence of lead-based paint in a household, which corresponds to its age
- In risk management, investors can quantify how a stock's or asset's return is associated with several variables including market indices, interest rates, or macro indicators
- Forecasting sales prices of a product based on historical sales data

PROBLEM STATEMENT

The goal is to build a model that best predicts a house's price with 79 explanatory variables from lot frontage, to the year the house was built, to the finished and unfinished basement square footage, the number of full and half baths, and so forth.

THE DATA

The dataset is widely known as the Ames housing dataset, which consists of houses from Ames, Iowa. It consists of four components:

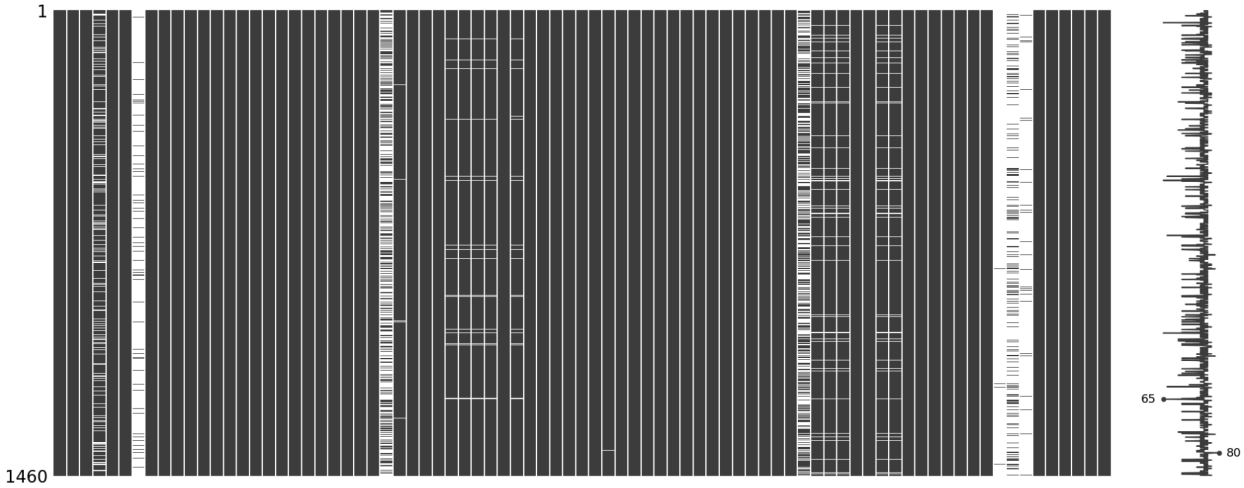
- `train.csv`, the training data that contains 1460 observations with 80 features including the target `SalePrice`
- `test.csv`, the test data that contains 1459 observations with 79 features without the target
- `data_description.txt`, a text file that briefly describes each feature
- `sample_submission.csv`, shows a sample submission for the Kaggle leaderboard

EXPLORATORY DATA ANALYSIS

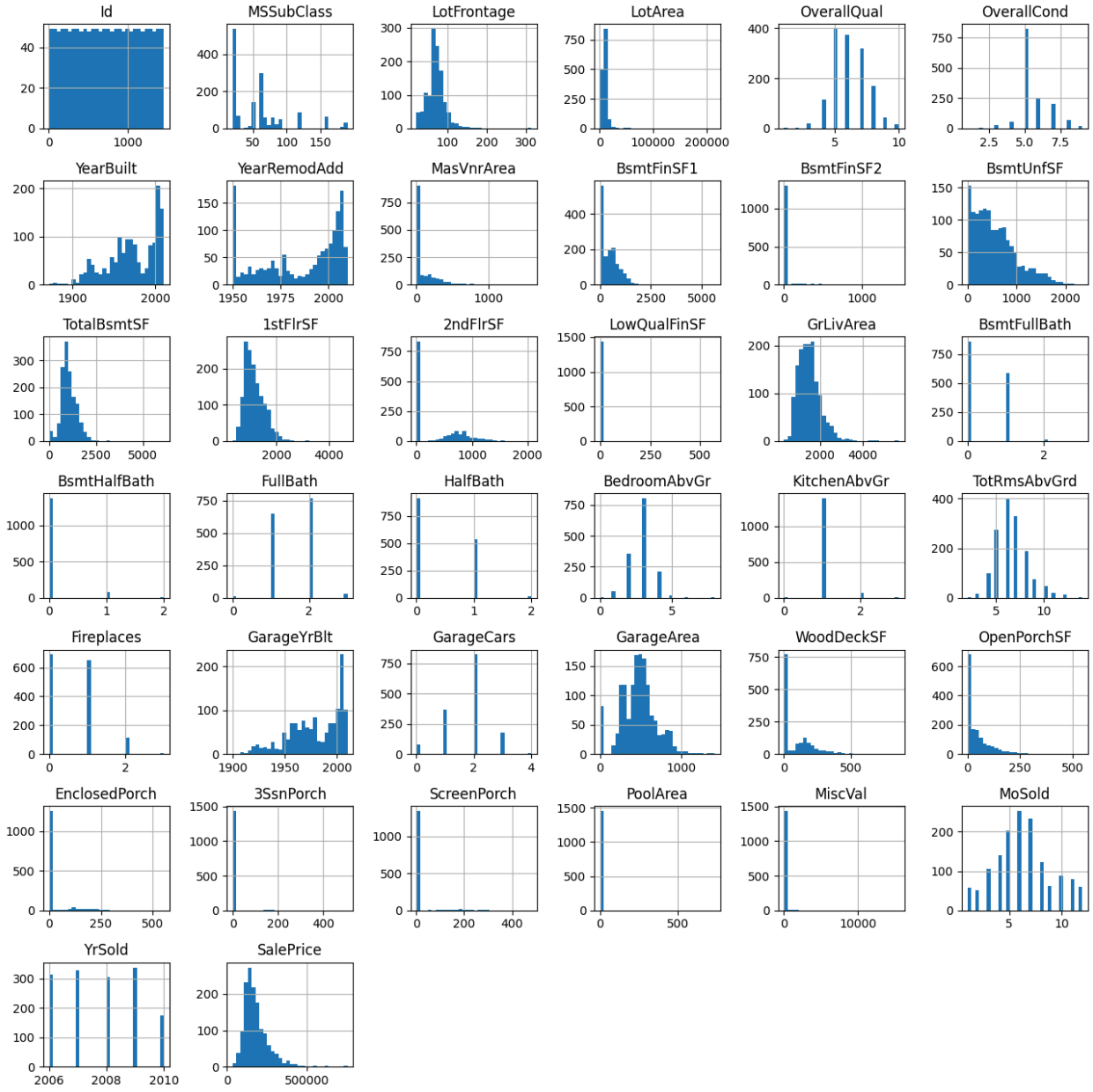
Excluding the target `SalePrice`, both the train and test datasets consist of 36 numerical and 43 categorical features. The datasets were then checked for missing values. Two separate dataframes, `missing_value_df` and `missing_value_test`, were created to save the names of features that had null values, the percentage and the number of null values, and the number of unique values they had, as this will help better assist with imputation and even feature engineering later on. An image of `missing_value_df` is shown below (the layout of `missing_value_test` is identical):

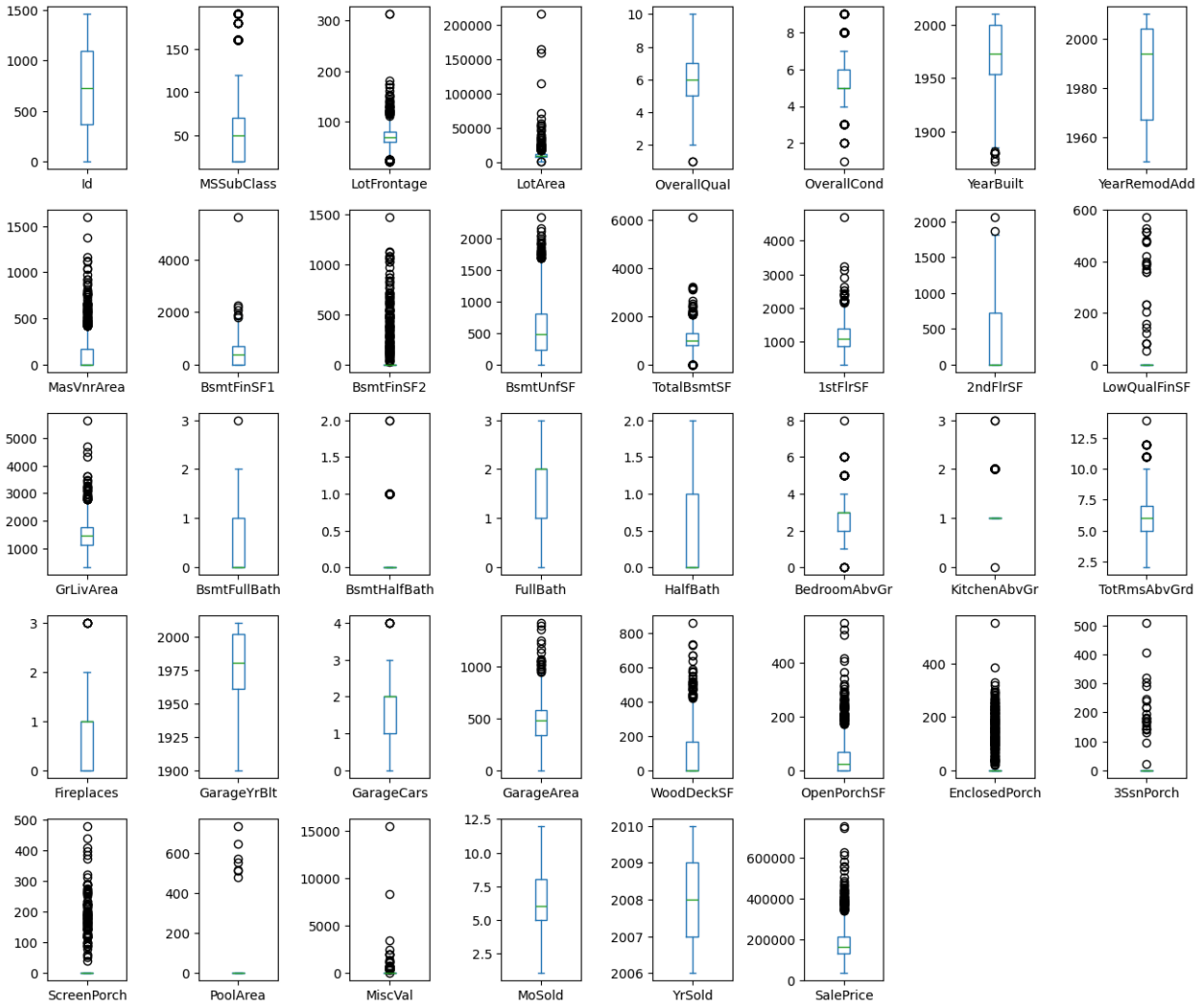
	column_name	percent_missing	num_missing	nunique_vals
0	PoolQC	99.520548	1453	3
1	MiscFeature	96.301370	1406	4
2	Alley	93.767123	1369	2
3	Fence	80.753425	1179	4
4	MasVnrType	59.726027	872	3
5	FireplaceQu	47.260274	690	5
6	LotFrontage	17.739726	259	110
7	GarageQual	5.547945	81	5
8	GarageFinish	5.547945	81	3
9	GarageType	5.547945	81	6
10	GarageYrBlt	5.547945	81	97
11	GarageCond	5.547945	81	5
12	BsmtFinType2	2.602740	38	6
13	BsmtExposure	2.602740	38	4
14	BsmtCond	2.534247	37	4
15	BsmtQual	2.534247	37	4
16	BsmtFinType1	2.534247	37	6
17	MasVnrArea	0.547945	8	327
18	Electrical	0.068493	1	5

Immediately, we see that `PoolQC`, `MiscFeature`, `Alley`, and `Fence` contain a majority of nulls. To visualize the sparsity of the train dataset, we utilized the `missingno` (`msno` for short) library:

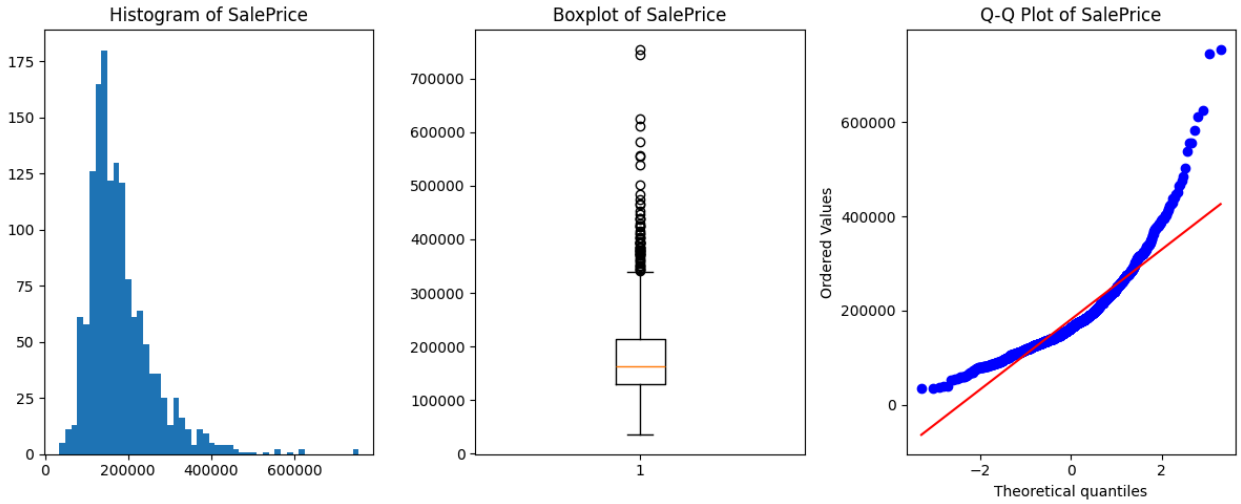


After assessing the sparsity of the data, we then checked for skewness within the numerical features by plotting out their respective histograms and boxplots.

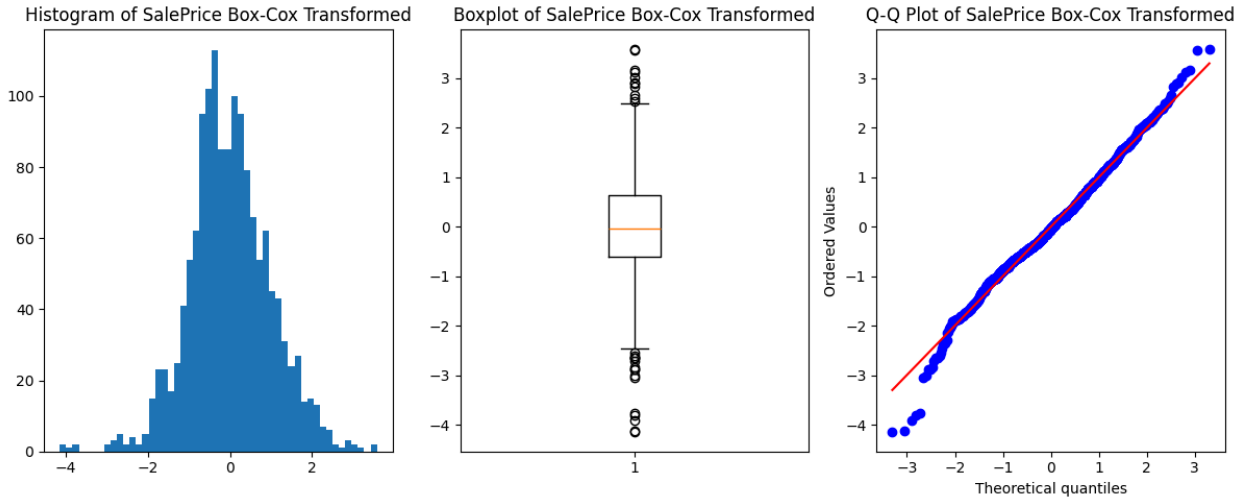

Id
MSSubClass
LotFrontage
LotArea
OverallQual
OverallCond
YearBuilt
YearRemodAdd
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageYrBlt
GarageCars
GarageArea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



We see that many of them contain outliers, and some features including `LotFrontage`, `GrLivArea`, `BsmtUnfSF`, `GarageArea`, and the target `SalePrice`, are highly skewed, so we might be interested in applying logarithmic transformations to reduce their influence when building our model. More importantly, since our target is skewed, we needed to further assess its normality by visualizing its histogram, boxplot, and Q-Q plot together.

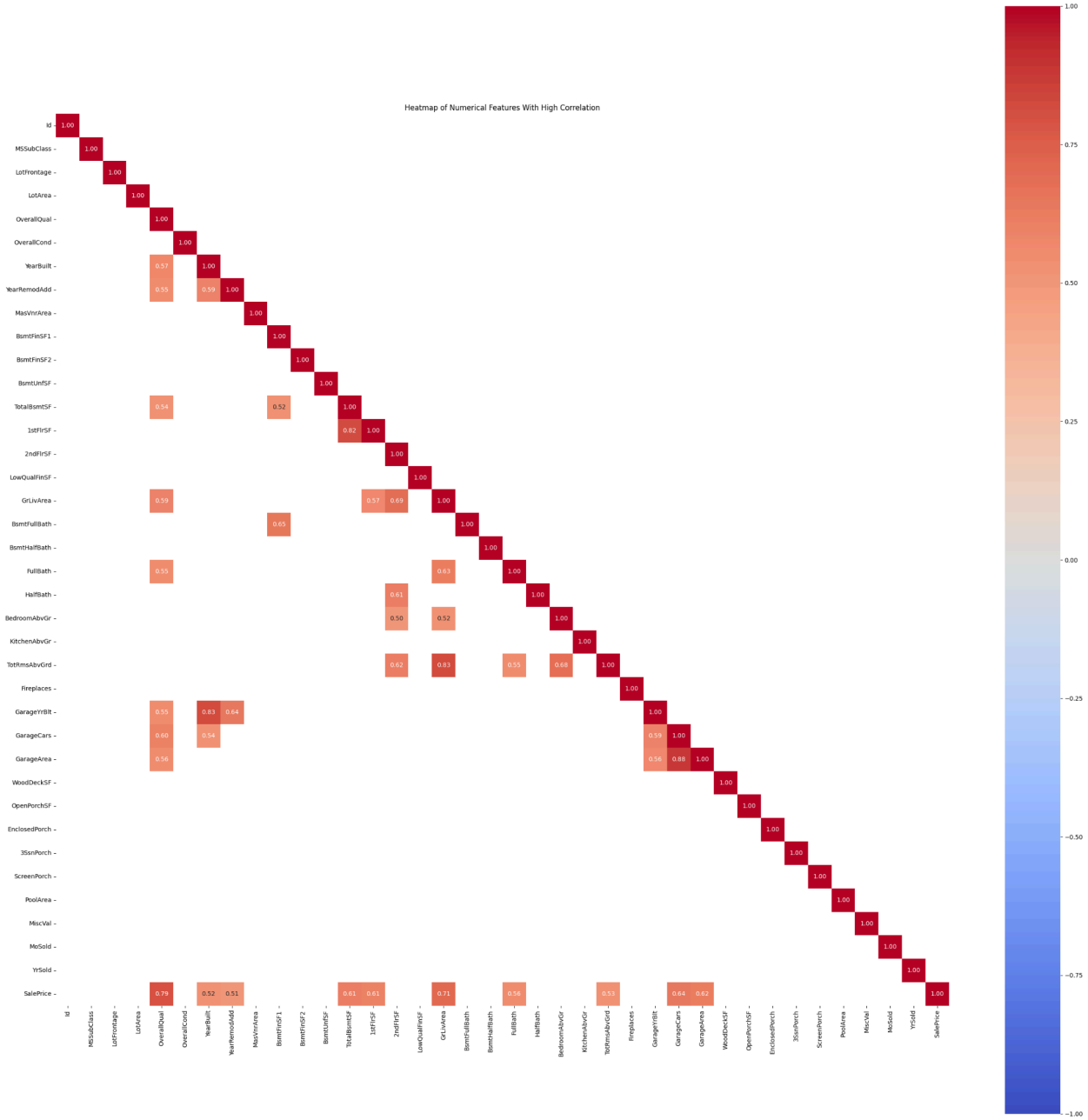


In this case, `SalePrice` is heavily right-skewed, with many highly-priced houses, and this is especially evident in the corresponding Q-Q plot, where the lower to mid quantiles are somewhat normal, but then the higher quantiles greatly deviate afterwards. If we use our target variable as is and use it to develop our model, it can heavily hinder its performance. Therefore, we will apply a Box-Cox power transformation to address this. The same plots are shown after applying the transformation:



We see that our target is now much more normal after applying the transformation.

After assessing skewness for the numerical features, we then proceeded by mapping out a correlation matrix and then filtering out values that were greater than or equal to 0.5, since those features might play a significant role in our model's performance. Because the correlation matrix is symmetric along the diagonal, where a given feature is correlated to itself with a value of 1, we omitted the upper half for clearer visualization.



We then re-organized this visualization as a separate dataframe, `strong_corr_df`, to methodically lay out which features were correlated with which ones, and then we sorted by highest to lowest correlation value. A preview of it is shown below:

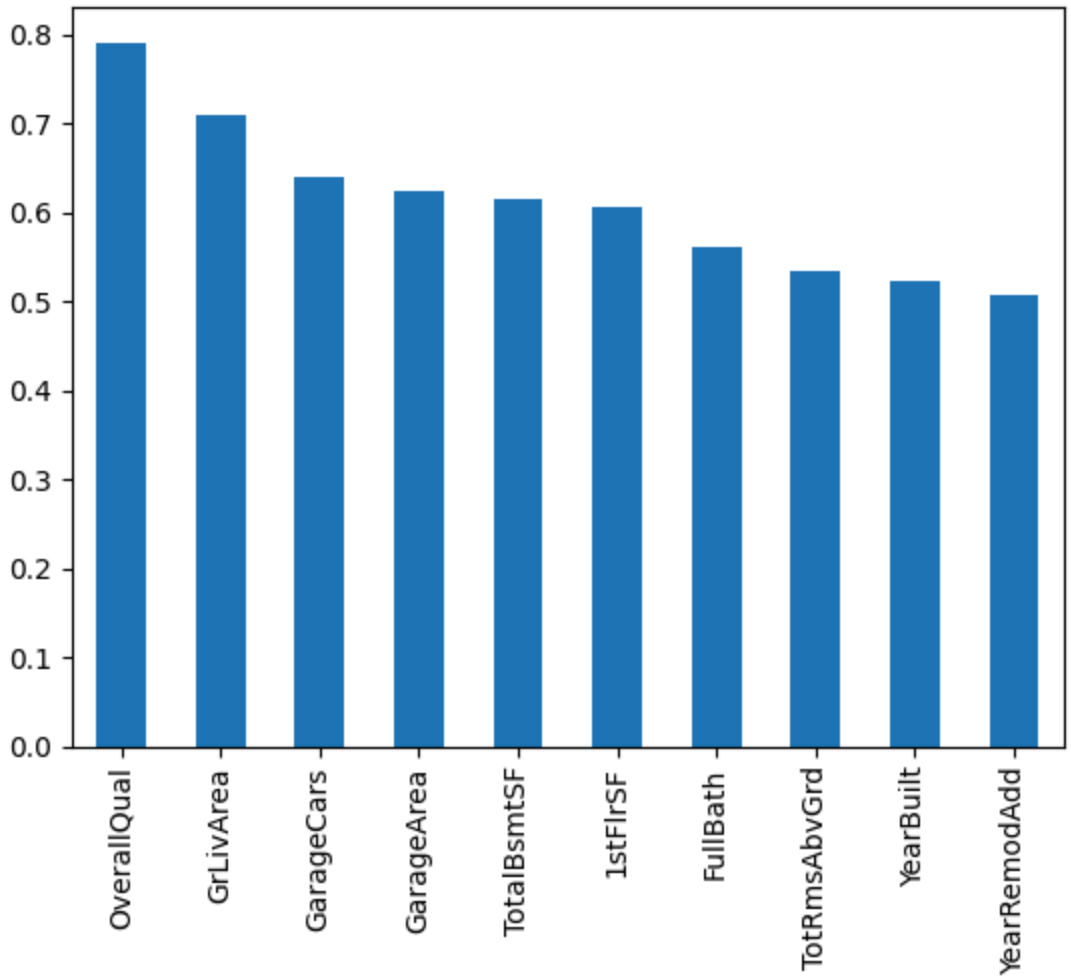
	Feature1	Feature2	Correlation
35	GarageCars	GarageArea	0.882475
10	YearBuilt	GarageYrBlt	0.825667
27	GrLivArea	TotRmsAbvGrd	0.825489
17	TotalBsmtSF	1stFlrSF	0.819530
8	OverallQual	SalePrice	0.790982
28	GrLivArea	SalePrice	0.708624
21	2ndFlrSF	GrLivArea	0.687501
31	BedroomAbvGr	TotRmsAbvGrd	0.676620
16	BsmtFinSF1	BsmtFullBath	0.649212
13	YearRemodAdd	GarageYrBlt	0.642277
36	GarageCars	SalePrice	0.640409

Based on this, it seems that the pairs `GarageCars` with `GarageArea` and `YearBuilt` with `GarageYrBlt` exhibit multicollinearity. For the former, it makes sense that a house is more attractive to buy if it has a garage. With that in mind, we can construct a boolean-valued feature called `HasGarage` to indicate whether or not a house has a garage, as this might help with our model's predictive capabilities.

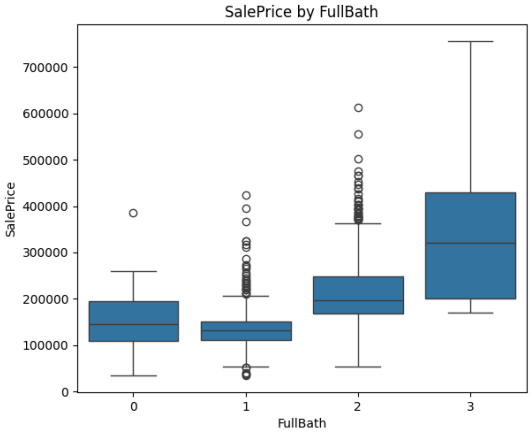
Similarly, we also wanted to check which numerical features were correlated to the target `SalePrice`. A separate dataframe called `corr_SP` was created, with a preview of it, filtered above our threshold of equal to or greater than 0.5, shown below:

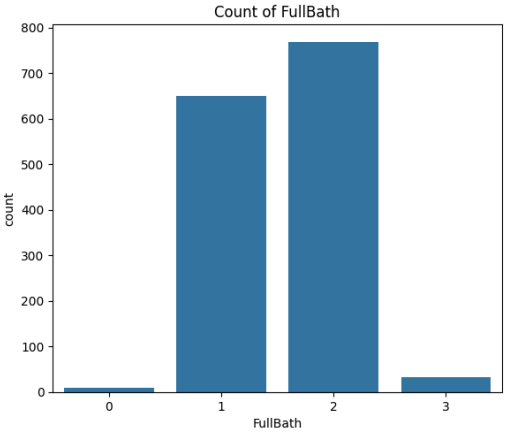
	SalePrice
SalePrice	1.000000
OverallQual	0.790982
GrLivArea	0.708624
GarageCars	0.640409
GarageArea	0.623431
TotalBsmtSF	0.613581
1stFlrSF	0.605852
FullBath	0.560664
TotRmsAbvGrd	0.533723
YearBuilt	0.522897
YearRemodAdd	0.507101

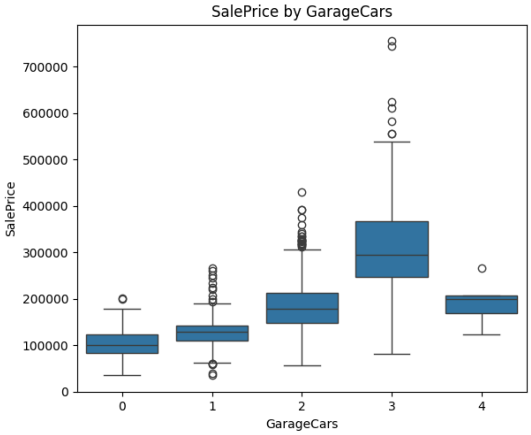
We can then visualize this as a simple bar graph:

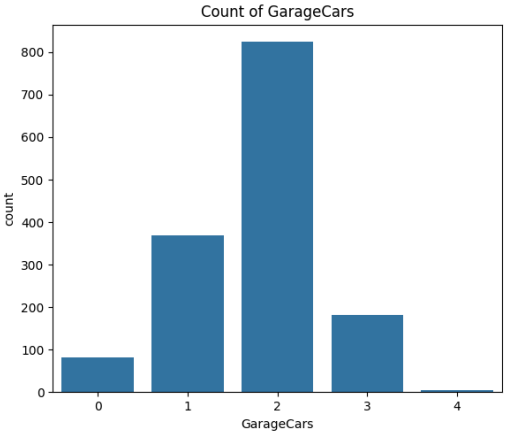


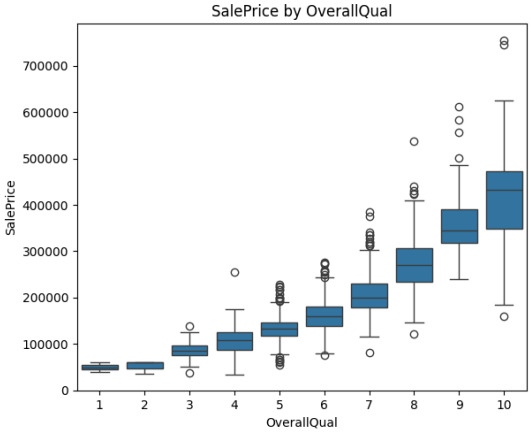
We then further investigate how some of these numerical features correlate with `SalePrice` by plotting out their boxplots and a histogram of their counts side-by-side.

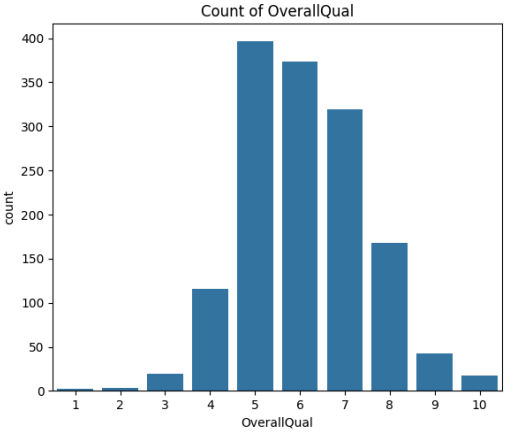

SalePrice by FullBath
SalePrice
FullBath

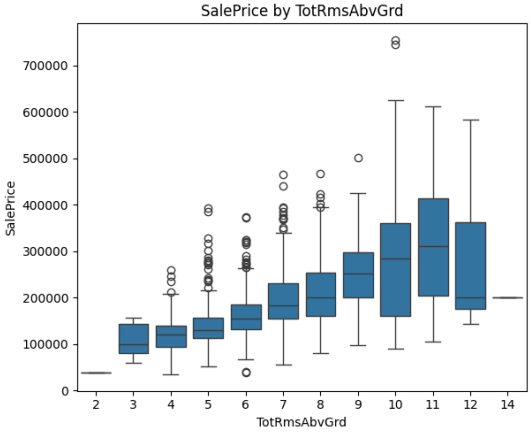

Count of FullBath
count
FullBath

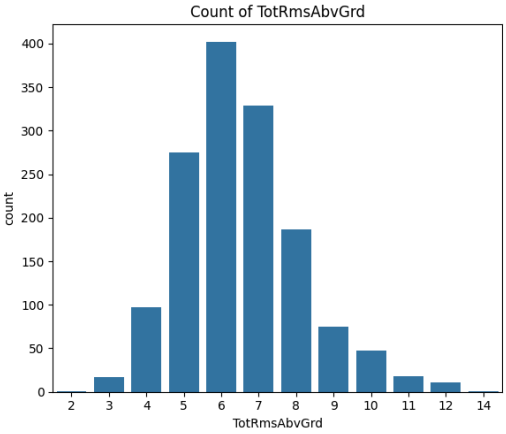

SalePrice by GarageCars
SalePrice
GarageCars


Count of GarageCars
count
GarageCars


SalePrice by OverallQual
SalePrice
OverallQual


Count of OverallQual
count
OverallQual

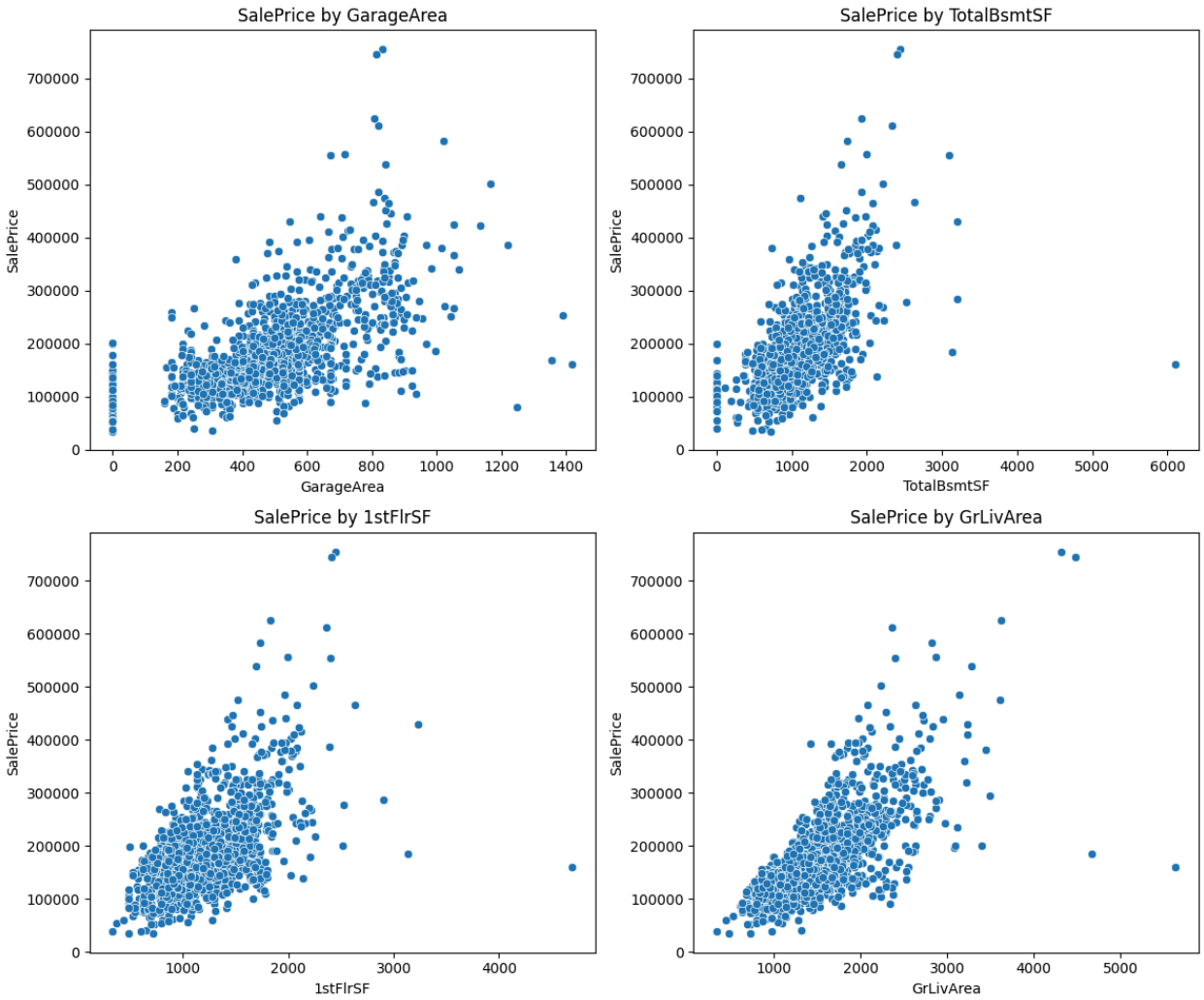

SalePrice by TotRmsAbvGrd
SalePrice
TotRmsAbvGrd


Count of TotRmsAbvGrd
count
TotRmsAbvGrd

Some observations that we note include:

- For `FullBath`, most of the houses have either 1 or 2 full bathrooms. As the number of full bathrooms increases, so does the `SalePrice`. There is a considerable amount of outliers in `SalePrice` for houses that have 1 or 2 full bathrooms.
- For `GarageCars`, most of the houses can fit 1 or 2 cars. We see a clear increase of `SalePrice` when there is a garage built, albeit not so much if the garage can carry 4 cars. There are many outliers in `SalePrice` for houses with garages that can fit between 1 or 2 cars, while not as much for the case of 3 cars.
- For `OverallQual`, we definitely see that most of the houses are rated either 5, 6, and 7, which can be considered 'average'; there aren't as many houses rated 8 or above. We also see a definite increase in `SalePrice` as the `OverallQual` score increases. There are outliers in `SalePrice` for houses rated between 5 and 7 in `OverallQual`.
- For `TotRmsAbvGrd`, we see most of the houses have between 5 to 8 rooms above ground. As the number of rooms increase, so does `SalePrice`. It seems that many of the outliers are concentrated for houses with 5 to 7 rooms above ground. Interestingly enough, we also see some extreme cases of houses with 11, 12, or even 14 rooms.

For the remaining features correlated to `SalePrice`, we can visualize their relationship by plotting their corresponding scatter plots; we'll exclude `YearBuilt` and `YearRemodAdd` for easier interpretability.

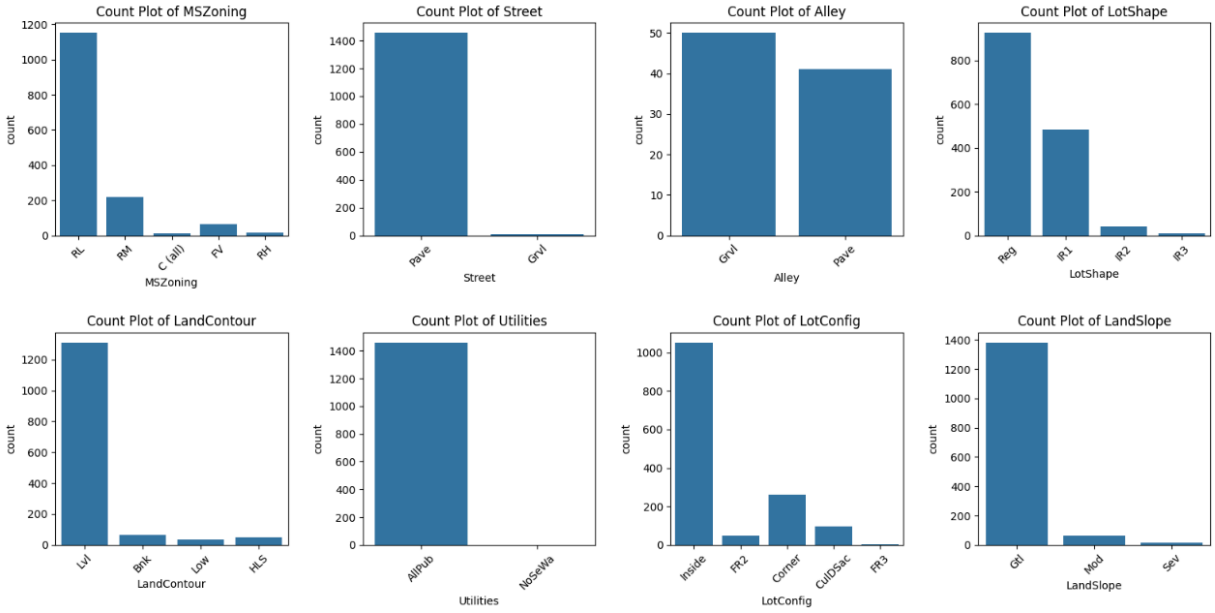


Based on these scatter plots, we generally observe that as the feature increases, so does SalePrice. However, we should also take note of the outliers:

- For `GarageArea`, we see that a bigger garage does not necessarily mean a more expensive house. In this case, we see that some houses with a `GarageArea` of around 800 are significantly more expensive than those with a `GarageArea` of around 1400. We should also note the many data points that have no garage.
- For `TotalBsmtSF`, while we do generally see a steep increase of `SalePrice`, we observe one outlier with a `TotalBsmtSF` value of slightly over 6000, but its respective price is around $150,000. Similarly to above, we should also take note of houses that do not have a basement.

- For `1stFlrSF`, similarly to `TotalBsmtSF`, we see a steep increase of `SalePrice`. One outlier with a `TotalBsmtSF` value of around 5000 has a respective `SalePrice` of around $160,000 to $170,000. The same applies with a couple houses with a `1stFlrSF` value of around 2500, but their respective prices are a bit over $750,000.
- For `GrLivArea`, we definitely see a few outliers where some houses might have expansive ground living area but aren't as expensive. Additionally, we see houses that do have considerable ground living area but are much more expensive than all other houses. In this case, we see many of the outliers located where `GrLivArea` is approximately greater than 3500.

Lastly, for the categorical features, we investigated the distribution of the value counts by plotting each of them in bar graphs. A preview is shown below:



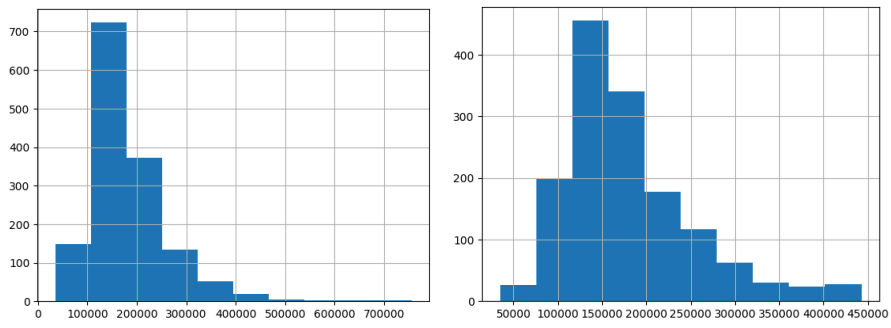
In some cases, especially with `Street` and `Utilities`, many of the observations took on a single value, so it might be optimal to drop those features altogether. To quantify and lay out this “dominance,” a separate dataframe called `dominance_df` was created and then sorted by highest to lowest in dominance percentage. A preview is shown below:

	Feature	Dominance_Percentage
5	Utilities	99.931507
1	Street	99.589041
10	Condition2	98.972603
14	RoofMatl	98.219178
26	Heating	97.808219
36	GarageCond	96.156635
35	GarageQual	95.068891

Similarly with `missing_value_df` and `missing_value_test` that we created earlier, this dataframe will better assist with imputation and feature engineering later on.

DATA WRANGLING AND PREPROCESSING

Before everything else, the first thing to preprocess was the target `SalePrice`. Namely, we wanted to reduce the influence of observations that had much higher prices than the rest, since, as aforementioned, this could hinder our model's performance. One effective strategy is to winsorize (to clip or to "cap out") the outliers, which means to still keep the outlier observations in the dataset, but then replace their value by a certain threshold. In this case, we winsorize only the top 1% of observations. The before and after effects of the distribution of winsorization is shown below:



In this case, the 99th percentile sits at $442,567.01, so any observation that is above that gets changed to that threshold. More importantly, we already see that the distribution of `SalePrice` is a lot less right-skewed, which, on top of the Box-Cox power transformation that will get applied later on, will help better normalize the target.

For the sake of briefness, we will only discuss the general strategies that were employed for imputation and feature engineering.

We will first discuss how the numerical features were preprocessed. In terms of imputation, preliminary imputation was done, depending on the feature itself. For example, for features like `MasVnrArea`, `BsmtFinSF1`, and `BsmtFinSF2`, a 0 indicates no presence of masonry veneer or basement. If some observations were missing values in such features, they were imputed with 0. After all the necessary preliminary imputation was done, the rest of the data was imputed with KNN-imputation. This type of imputation works by finding the k-nearest neighbors to a data point with a missing value and imputing the missing value using the mean or median of the neighboring data points. One key advantage is that this approach preserves the relationships between features, which can lead to better model performance compared to simpler imputation methods like mean or median imputation, which may introduce potential bias.

For feature engineering, the following strategies were done:

- Applying logarithmic transformations to heavily-skewed features like `MasVnrArea`
- Taking the sum of interrelated features to obtain a net total, such as
  `TotalBsmtFinSF = BsmtFinSF1 + BsmtFinSF2`
  `TotalFlrSF = 1stFlrSF + 2ndFlrSF`
  We might also take a linear combination of two features to give one feature more importance than the other, such as
  `TotalBsmtBath = BsmtFullBath + 0.5 * BsmtHalfBath`
- Taking the ratio between two features such as
  `FinishedBsmtRatio = TotalBsmtFinSF / TotalBsmtSF`
- Multiplying two features together to create “interactive” features that can help capture potential nonlinearity such as

`Qual_GrLivArea = GrLivArea * OverallQual`
This was particularly done on the pairs of features (excluding `SalePrice`) that were correlated to each other in `strong_corr_df`, where each pair is multiplied together (i.e. pairwise multiplication)
- Convert the feature into a boolean-valued (i.e. 0 or 1) feature such as `Fireplaces`, which take on values of 0, 1, 2, or 3, and collapse all observations that take on 2 or 3 into 1
- If a feature is time-based and cyclical in nature, we apply sine and cosine transformations such as
$MoSold_{sin} = sin(2\pi \frac{MoSold}{12})$
$MoSold_{cos} = cos(2\pi \frac{MoSold}{12})$

After all the numerical features were imputed and feature engineered, robust scaling was applied, especially to take into account outliers.

For the categorical features, imputation was mainly done by the mode, or the most frequently-occuring value. However, there were two main challenges:

- (1) Some of the features were of high-cardinality, meaning they had many unique values. This could pose a problem when one-hot encoding them, as this might cause a lot of sparsity due to increased dimensionality
- (2) Some values appeared in the train dataset but not in the test dataset or vice-versa. On Kaggle, a common strategy to overcome this is to concatenate both the train and test datasets together and then proceed with encoding the categorical features, which then captures all possible values. However, this can cause potential data leakage, which can hinder our model's performance

With those challenges in mind, one way we can overcome this is to consolidate the values, so that both the train and test datasets are consistent and to reduce cardinality. An example is shown below with the feature `Exterior2nd`, with the train and test dataset counts side-by-side:

Exterior2nd	count	count
VinylSd	504.0	510.0
MetalSd	214.0	233.0
HdBoard	207.0	199.0
Wd Sdng	197.0	194.0
Plywood	142.0	128.0
CmentBd	60.0	66.0
Wd Shng	38.0	43.0
Stucco	26.0	21.0
BrkFace	25.0	22.0
AsbShng	20.0	18.0
ImStucc	10.0	5.0
Brk Cmn	7.0	15.0
Stone	5.0	1.0
AsphShn	3.0	1.0
Other	1.0	NaN
CBlock	1.0	2.0
Unknown	NaN	1.0

Exterior2nd	count	count
VinylSd	504	510
MetalSd	214	233
HdBoard	207	199
Wd Sdng	197	194
Other	196	195
Plywood	142	128

For features that were rank-based, such as `BsmtCond`, ordinal encoding was used. Otherwise, every other categorical feature was one-hot encoded.

After all the preprocessing was done, excluding the target and `Id` features, both the train and test datasets ended up with 214 features and were saved as `train_final.csv` and `test_final.csv`, respectively, along with the power transformer object serialized as `boxcox_transformer.pickle`, so that we can apply the inverse transformation on `SalePrice` back to its original dollar domain.

MODELING AND RESULTS

The `train_final.csv` dataset was 80-20 train-test split for model development and evaluation. Eight regression models were implemented, ranging from basic linear models to more advanced ensemble algorithms.

Each model was tuned using 5-fold cross-validation via GridSearchCV, which systematically tests combinations of hyperparameters, which are model-specific settings that control complexity and behavior. These include values like:

- `alpha` (used in Lasso, Ridge, and ElasticNet), which controls the strength of regularization
- `n_estimators`, `max_depth`, `learning_rate` (used in XGBoost and Random Forest), which manage the number, size, and learning sensitivity of decision trees

Tuning these hyperparameters helped optimize model performance and reduce overfitting. Below is a summary of each model and its performance:

1. Ordinary Least Squares (OLS)
A baseline linear regression model that minimizes residual error across all features.

Performance: RMSE (train) = $17,260, RMSE (test) = $21,200, adj R^2 (test) = 0.719- Solid benchmark, but slightly overfit (train R^2 = 0.944).

2. OLS with RFECV
This version of OLS incorporates Recursive Feature Elimination with Cross-Validation (RFECV), which prunes less useful features. A total of 173 features were retained after selection.

Performance: RMSE (train) = $17,254, RMSE (test) = $20,845, adj R^2 (test) = 0.823

3. Lasso Regression
A linear model with L1 regularization that shrinks less important features to zero (i.e., automatic feature selection). A total of 111 features remained afterwards.

Hyperparameter tuned: `alpha` via 5-fold GridSearchCV

Performance: RMSE (train) = $18,778, RMSE (test) = $19,297, adj R^2 (test) = 0.901

4. Ridge Regression
Uses L2 regularization to penalize large coefficients while retaining all features.

Hyperparameter tuned: `alpha` via 5-fold GridSearchCV

Performance: RMSE (train) = $18,593, RMSE (test) = $19,412, adj R^2 (test) = 0.764

5. ElasticNet
Combines both L1 and L2 penalties for a balance between feature selection and regularization. A total of 148 features remained afterwards.

Hyperparameters tuned: `alpha` and `l1_ratio` via 5-fold GridSearchCV

Performance: RMSE (train) = $19,159, RMSE (test) = $19,426, adj R^2 (test) = 0.873

6. XGBoost Regressor
A powerful tree-based ensemble model that uses gradient boosting.

Hyperparameters tuned: `n_estimators`, `learning_rate`, `max_depth`, `subsample`, `colsample_bytree`

Performance: RMSE (train) = $5336, RMSE (test) = $21,384, adj R^2 (test) = 0.715

7. Random Forest Regressor
An ensemble of decision trees trained with bagging to reduce variance.

Hyperparameters tuned: `n_estimators`, `max_depth`, `min_samples_split`, `min_samples_leaf`, `max_features`

Performance: RMSE (train) = $9519, RMSE (test) = $23,618, adj R^2 (test) = 0.652

8. Theil-Sen Regression

A robust linear model that reduces sensitivity to outliers by using median slopes.

Performance: RMSE (train) = $26,261, RMSE (test) = $21,208, adj R^2 (test) = 0.719

Observations

In summary, the model that best captured linearity was Lasso, as it also had the lowest test RMSE and highest adjusted R^2. Even though Lasso, Ridge, and Elastic Net yielded similar values of train and test RMSE, Lasso was the best regularized linear model, followed by Elastic Net and OLS with RFECV. However, Random Forest and XGBoost significantly overfit, which is highly evident due to the wide difference between the train and test RMSE metrics. Additionally, while Theil-Sen Regression was able to similarly capture linearity as with OLS, it underperformed since the train and test RMSE were higher.

CONCLUSION

Overall, through intentional exploratory data analysis, imputation, and feature engineering, we were able to construct many models and achieve an interpretable model — Lasso Regression — that captured strong linearity in our data. Moreover, winsorizing our target `SalePrice` allowed our model to gain stronger predictive power, as outliers can heavily affect its performance if not properly addressed.

In terms of future improvements, we could try out different models that can better capture any nonlinear relationships in our data, especially for houses with skewed prices. Moreover, we could try additional feature engineering and apply more extensive logarithmic transformations to improve model robustness. Additionally, we could conduct a random grid search to save on time and computational costs.

REFERENCES

Real-Life Applications of Correlation and Regression. (2024, April 10). GeeksforGeeks. https://www.geeksforgeeks.org/real-life-applications-of-correlation-and-regression/

Regression Analysis: Understanding the Why? | GEOG 586: Geographic Information Analysis. (n.d.). Www.e-Education.psu.edu. https://www.e-education.psu.edu/geog586/node/624

Regression analysis. (2019, March 22). Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Regression_analysis

Gupta, M. (2018, September 13). ML | Linear Regression - GeeksforGeeks. GeeksforGeeks. https://www.geeksforgeeks.org/ml-linear-regression/

GeeksforGeeks. (2024, August 13). Handling Missing Data with KNN Imputer. GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/handling-missing-data-with-knn-imputer/