House price prediction using Machine Learning

Vedanth Sirimalla

Introduction

- The project aims to use machine learning for predicting house prices. Various different kinds of machine learning models will be trained and tested for the task.
- Predicting house prices is an important problem as it leads to financial planning for the people involved and machine learning can help greatly in this aspect.
- It helps in fostering market efficiency, facilitation of financial planning, risk management and concluding decisions that support investments and policy making.
- Using machine learning can automate the task of predicting house prices and can let people have an estimate.

Problem Statement

- The problem is a regression task where given the features of a house, the task is to predict its sales price.
- The features of the house include area, number of bedrooms, location which serve as the predictor variables X.
- The sales price is the target variable to be predicted which is the output y.
- Thus, the machine learning task here is a supervised learning problem that uses regression to predict a continuous outcome, which in this case is the sales price.

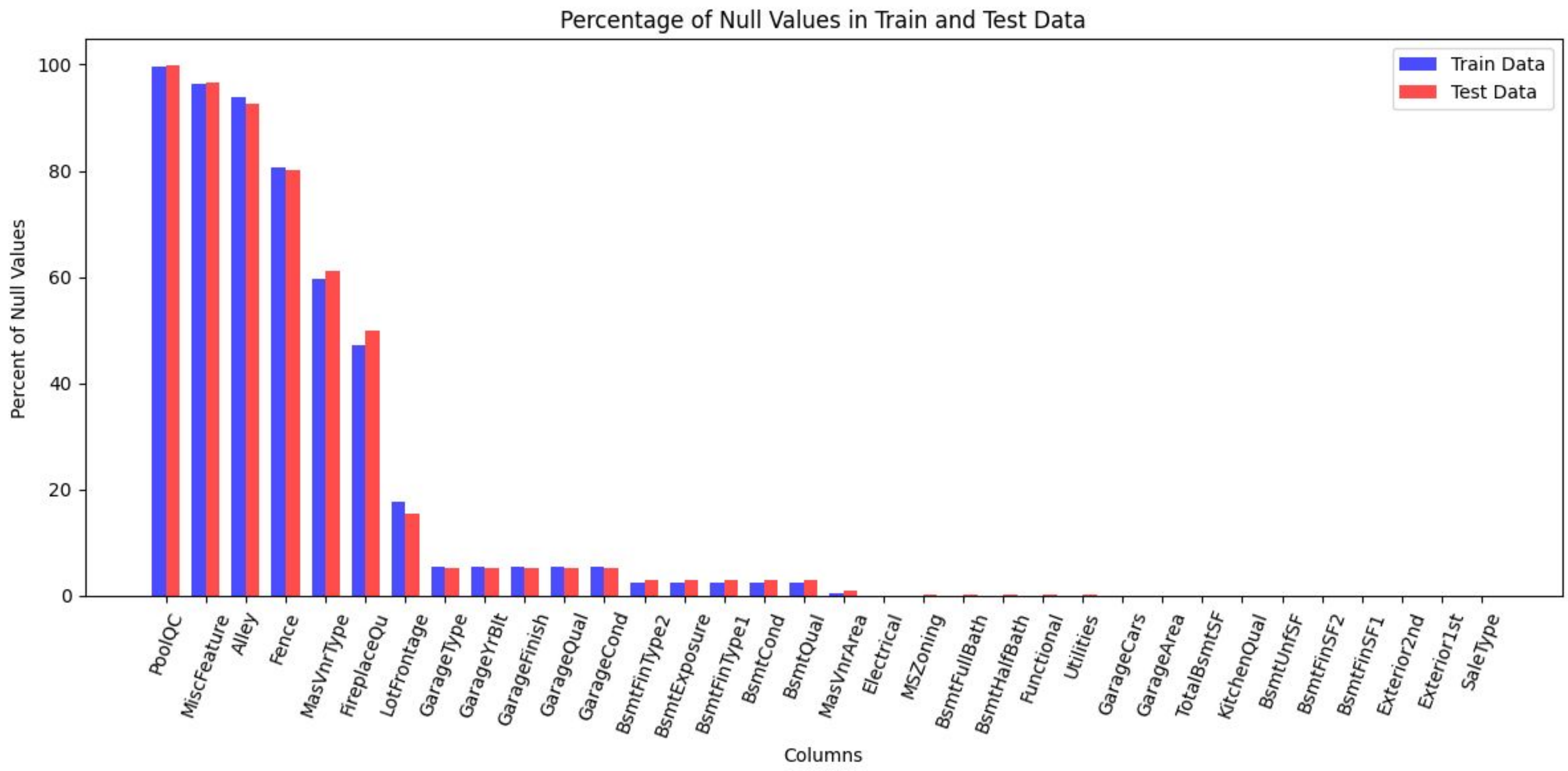
Data Description

- The data used for the project is sourced from Kaggle: House Prices: Advanced Regression Techniques.
- It contains data_description.txt, train.csv and test.csv for the task where train.csv is used for model training and validation, and data_description for feature selection and feature engineering.
- The test dataset is used to make prediction on the sales price of cars given the features.
- The dataset has a total of 81 columns out of which one column, sales price is the target column.
- It contains descriptors for the house that contains many features that are relevant to the purpose of predicting prices.

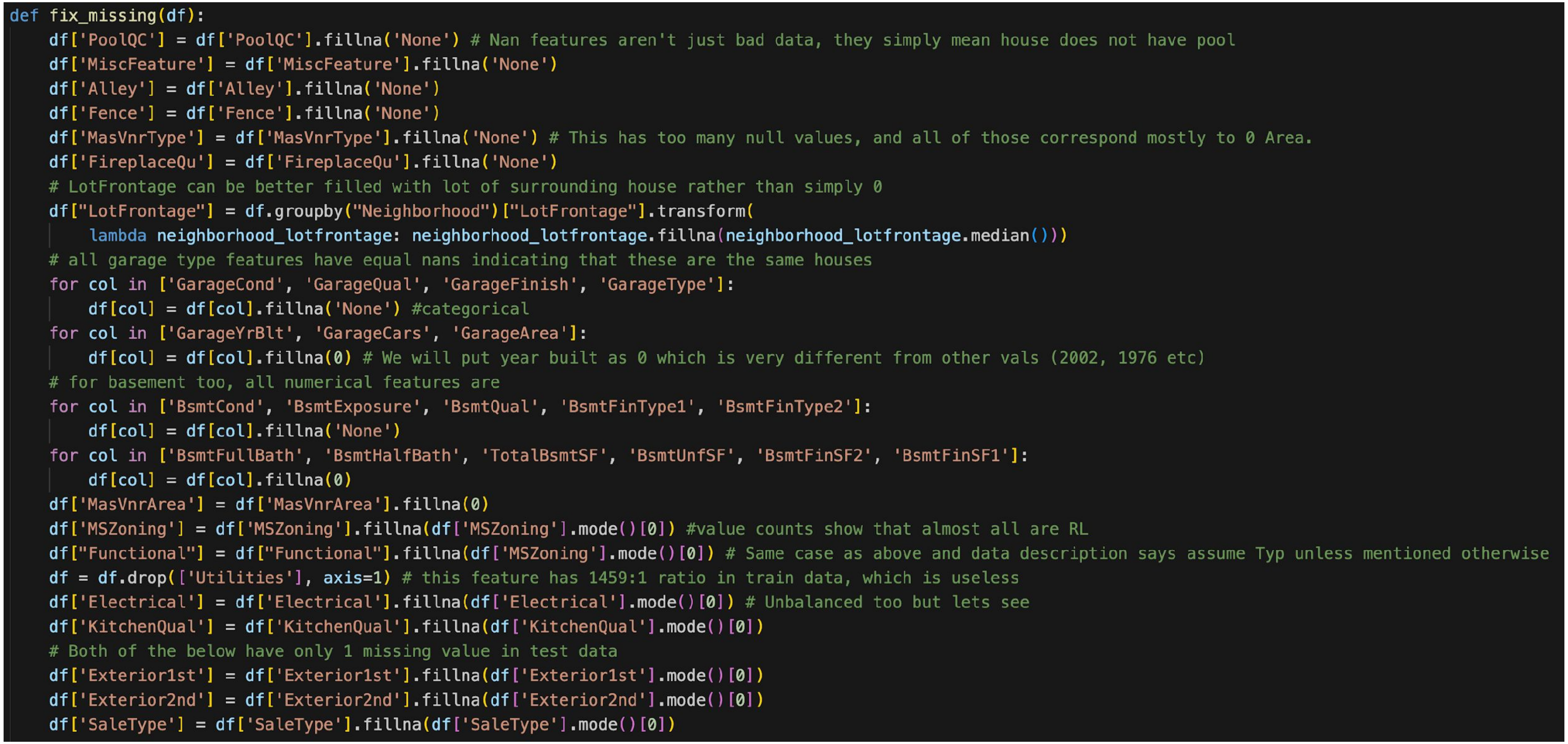
Data Exploration

- First, we tackle the null values in our dataset. One approach could be to simply drop features with excessive null values, but according to the data description, some null values could be interpreted as ‘None’.
- First we take a look at the percentage of null values in each feature and deal with each features separately.
- The special case is LotFrontage, where since it is correlated to Neighborhood, we fill the null values of LotFrontage as medians obtained after grouping by Neighborhood.
- Other than this features, the rest are filled either with 0 if numerical, mode if a certain value dominates and ‘None’ attribute if it is explicitly mentioned in the data description that NaN can be interpreted as ‘None’.
- Examples of ‘None’ included basement and garage features, where if an attribute is missing, we can consider as the house having no basement and/or garage since these attributes are all missing in the same rows (same houses).

Data Exploration



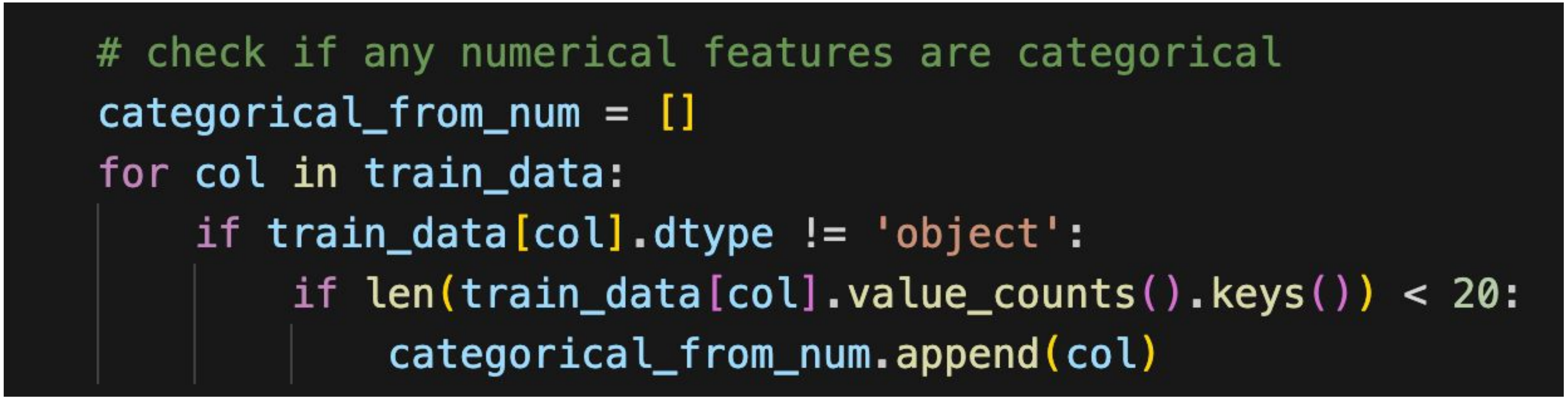
Data Exploration



```
def fix_missing(df):
    df['PoolQC'] = df['PoolQC'].fillna('None') # Nan features aren't just bad data, they simply mean house does not have pool
    df['MiscFeature'] = df['MiscFeature'].fillna('None')
    df['Alley'] = df['Alley'].fillna('None')
    df['Fence'] = df['Fence'].fillna('None')
    df['MasVnrType'] = df['MasVnrType'].fillna('None') # This has too many null values, and all of those correspond mostly to 0 Area.
    df['FireplaceQu'] = df['FireplaceQu'].fillna('None')
    # LotFrontage can be better filled with lot of surrounding house rather than simply 0
    df["LotFrontage"] = df.groupby("Neighborhood")["LotFrontage"].transform(
        lambda neighborhood_lotfrontage: neighborhood_lotfrontage.fillna(neighborhood_lotfrontage.median()))
    # all garage type features have equal nans indicating that these are the same houses
    for col in ['GarageCond', 'GarageQual', 'GarageFinish', 'GarageType']:
        df[col] = df[col].fillna('None') #categorical
    for col in ['GarageYrBlt', 'GarageCars', 'GarageArea']:
        df[col] = df[col].fillna(0) # We will put year built as 0 which is very different from other vals (2002, 1976 etc)
    # for basement too, all numerical features are
    for col in ['BsmtCond', 'BsmtExposure', 'BsmtQual', 'BsmtFinType1', 'BsmtFinType2']:
        df[col] = df[col].fillna('None')
    for col in ['BsmtFullBath', 'BsmtHalfBath', 'TotalBsmtSF', 'BsmtUnfSF', 'BsmtFinSF2', 'BsmtFinSF1']:
        df[col] = df[col].fillna(0)
    df['MasVnrArea'] = df['MasVnrArea'].fillna(0)
    df['MSZoning'] = df['MSZoning'].fillna(df['MSZoning'].mode()[0]) #value counts show that almost all are RL
    df["Functional"] = df["Functional"].fillna(df['MSZoning'].mode()[0]) # Same case as above and data description says assume Typ unless mentioned otherwise
    df = df.drop(['Utilities'], axis=1) # this feature has 1459:1 ratio in train data, which is useless
    df['Electrical'] = df['Electrical'].fillna(df['Electrical'].mode()[0]) # Unbalanced too but lets see
    df['KitchenQual'] = df['KitchenQual'].fillna(df['KitchenQual'].mode()[0])
    # Both of the below have only 1 missing value in test data
    df['Exterior1st'] = df['Exterior1st'].fillna(df['Exterior1st'].mode()[0])
    df['Exterior2nd'] = df['Exterior2nd'].fillna(df['Exterior2nd'].mode()[0])
    df['SaleType'] = df['SaleType'].fillna(df['SaleType'].mode()[0])
```

Feature Engineering

- Some features have their datatype as int64, meaning they could be considered numerical features although these are categorical by nature.
- We identified such features by getting value counts of each column and selecting columns with less that 20 unique keys.



```python
# check if any numerical features are categorical
categorical_from_num = []
for col in train_data:
    if train_data[col].dtype != 'object':
        if len(train_data[col].value_counts().keys()) < 20:
            categorical_from_num.append(col)
```

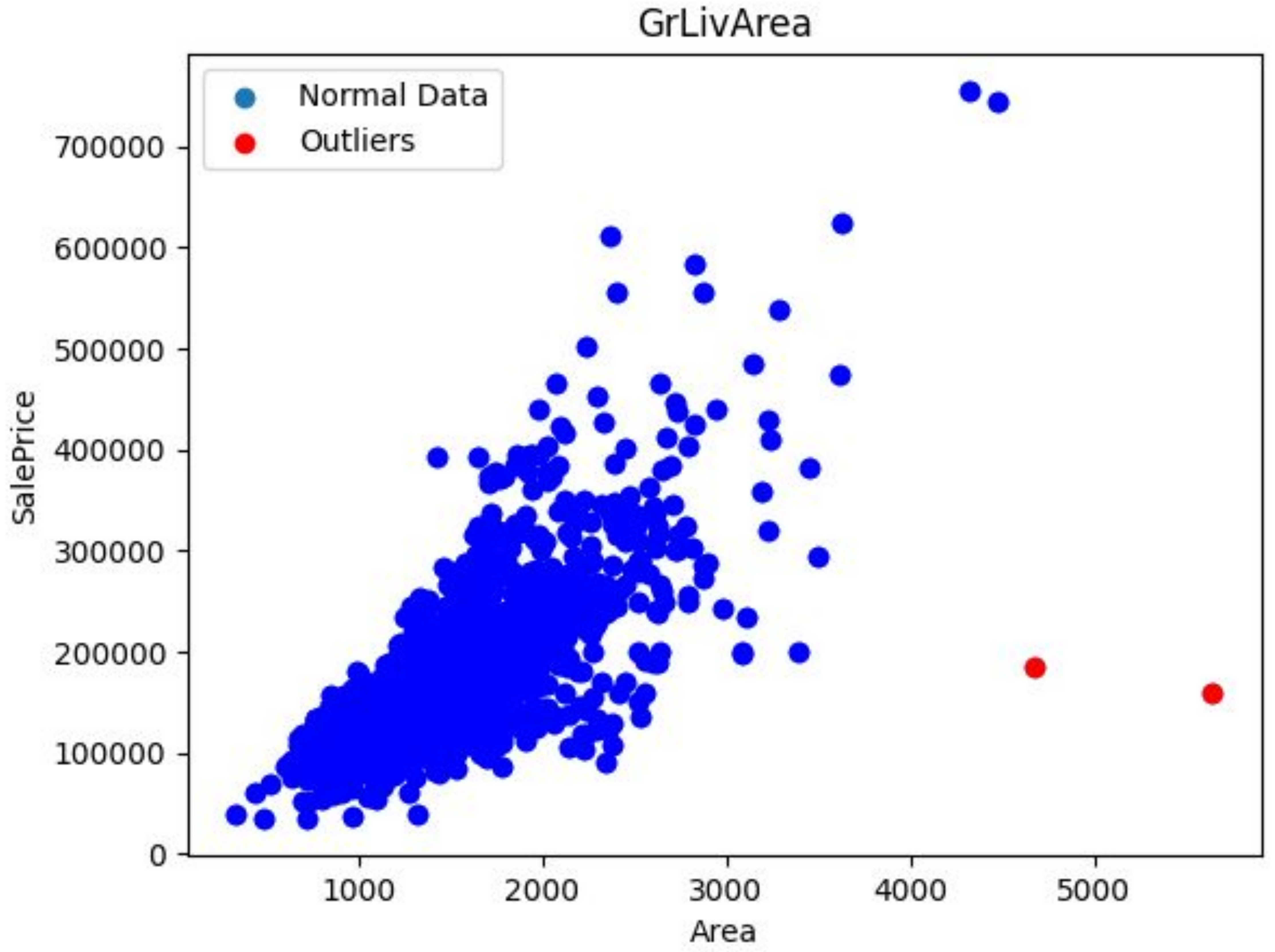
Feature Engineering

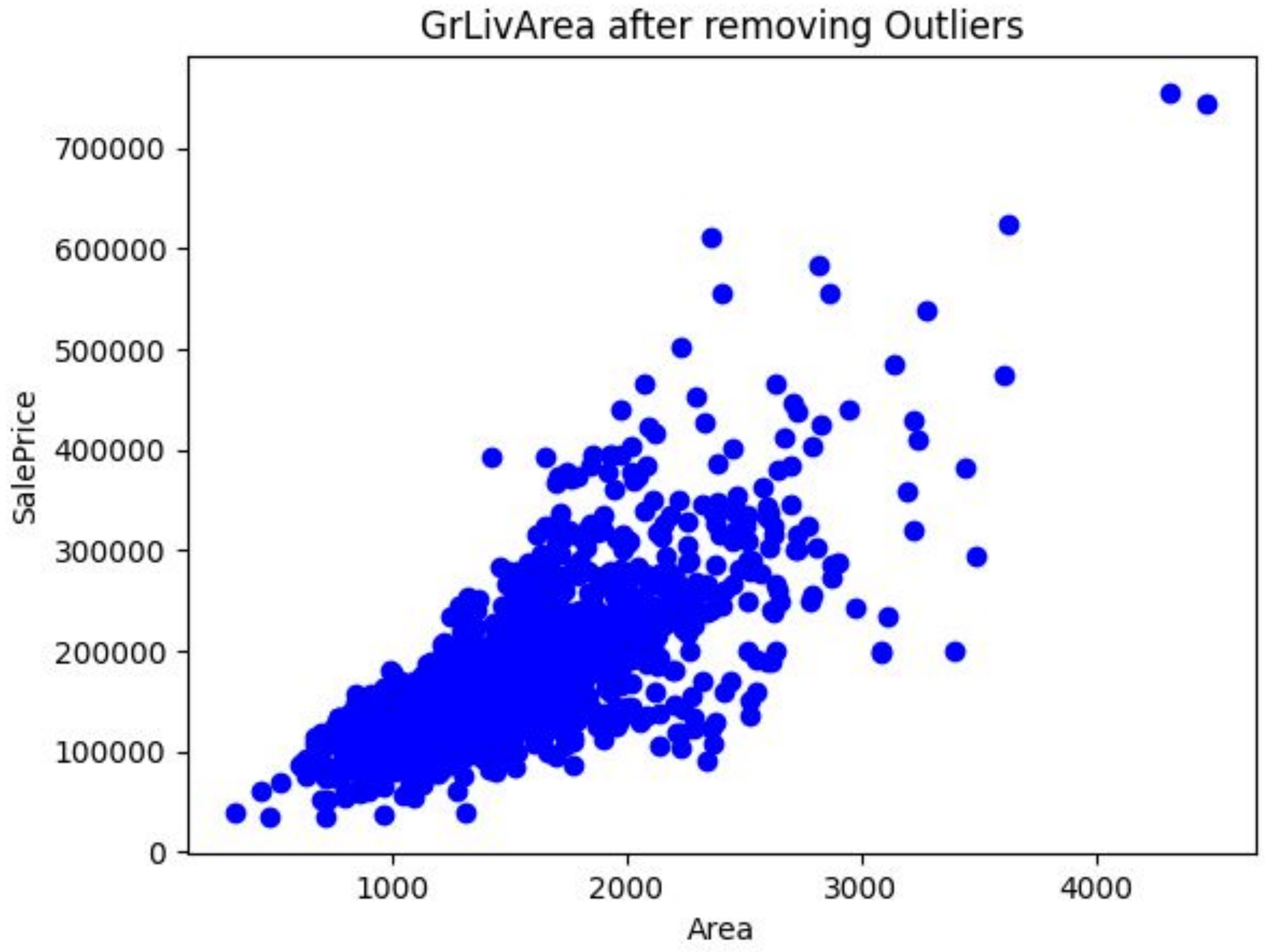
- Furthermore, among the categorical features, there are ordinal features too, which need to be handled differently.
- Specifically, ordinal features would be LabelEncoded and the remaining non-ordinal categorical features would be OneHotEncoded.
- Some examples of ordinal features are: Year Sold, Month Sold, Overall Quality, Overall Condition, etc.
- We have a total of 27 ordinal features and 30 non-ordinal features making a total of 57 categorical columns.
- We add an extra feature too:

```
train_data['TotalSF'] = train_data['TotalBsmtSF'] + train_data['1stFlrSF'] + train_data['2ndFlrSF']
test_data['TotalSF'] = test_data['TotalBsmtSF'] + test_data['1stFlrSF'] + test_data['2ndFlrSF']
```

Outliers

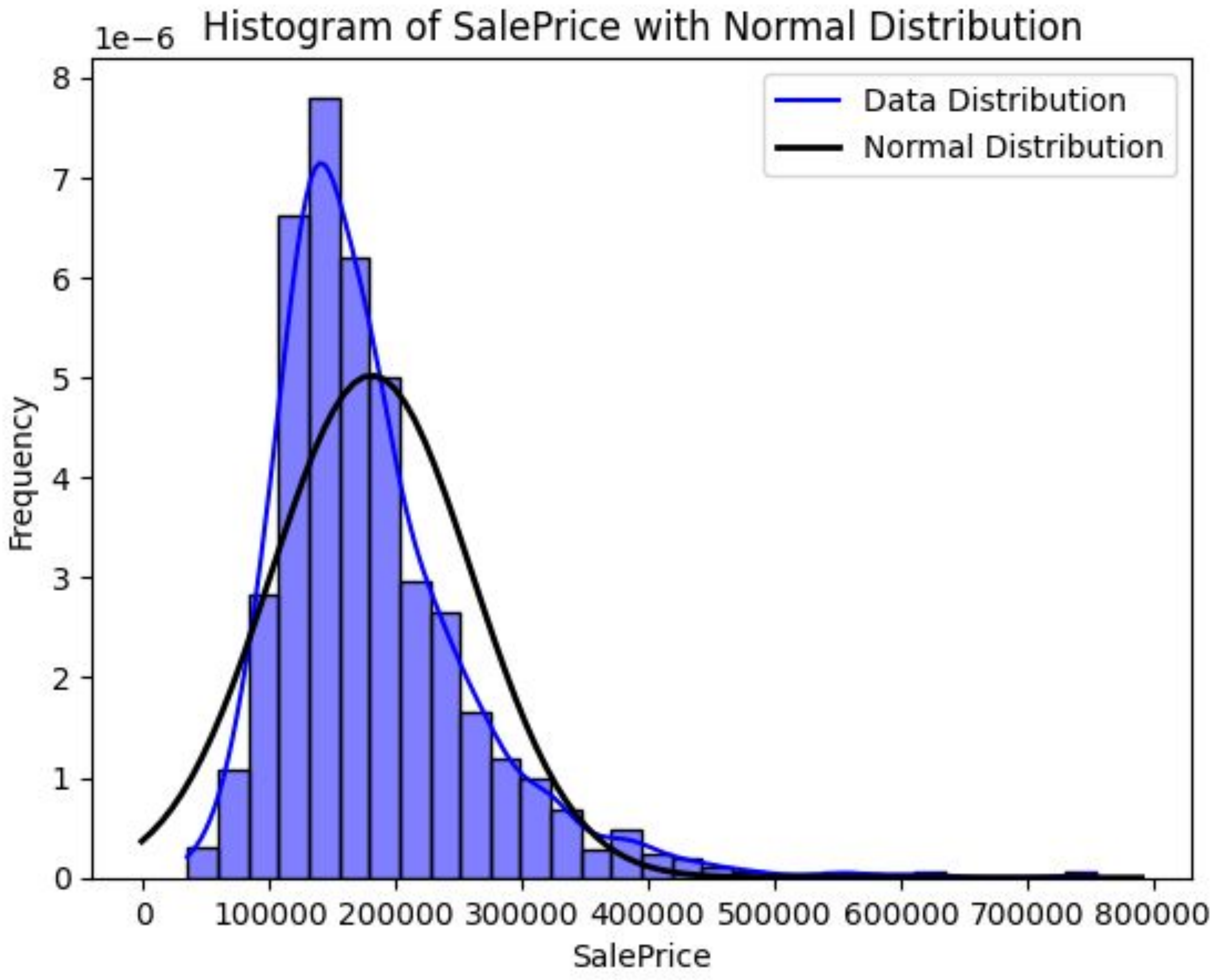
- The provided data description has specifically mentioned the any house having its ground living area above 4000 with Sale price below 300000 is an outlier. There are **2** such outliers in our data.

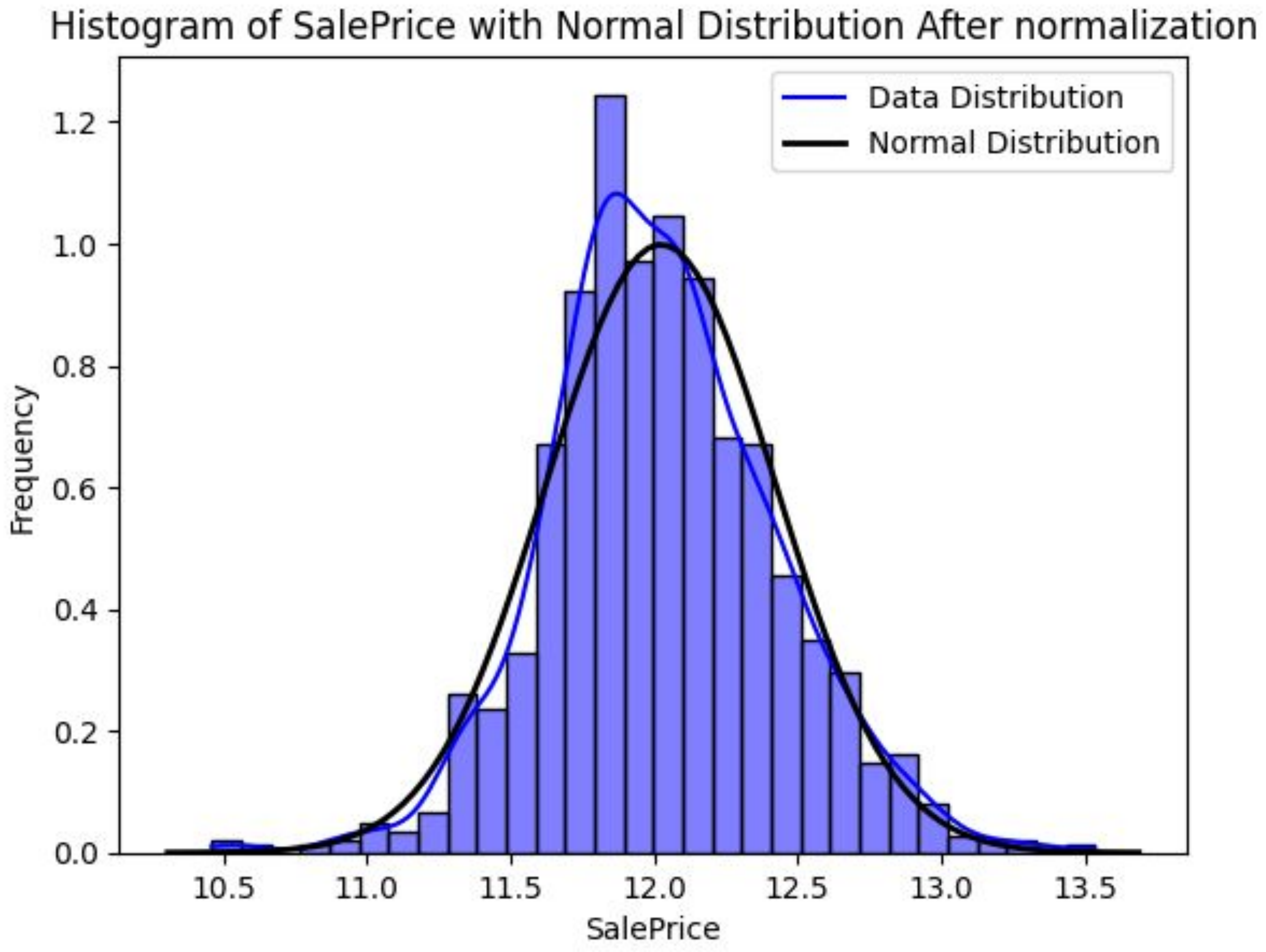




Handling the target variable

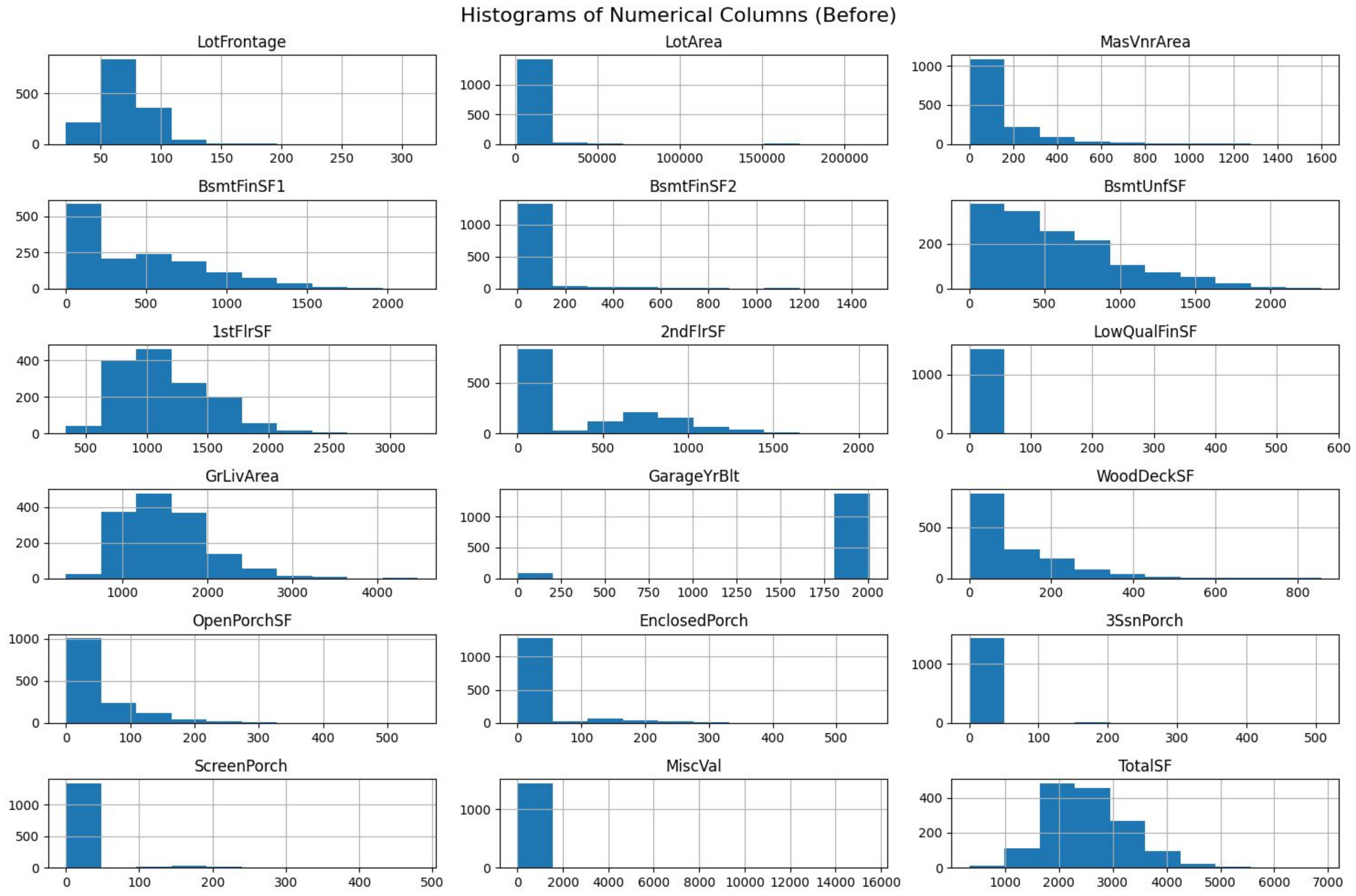
- Our target variable is skewed along with other numerical variables. We handle our target variable with log-normalization and the others with box-cox.





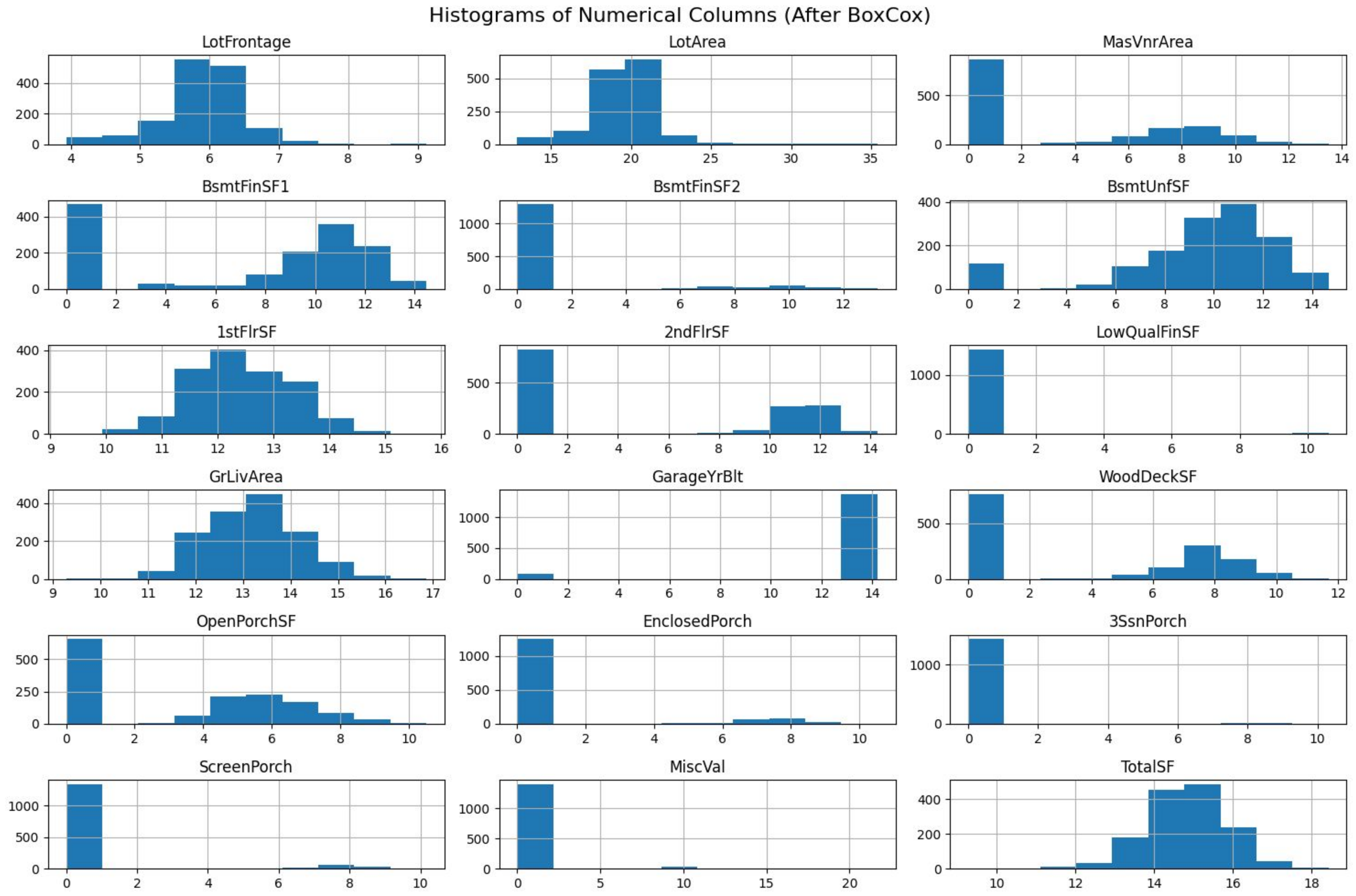
Handling numerical features

- We boxcoxed features with skew>0.75 and our lambda for boxcox was 0.15.



Handling numerical features

- We boxcoxed features with skew>0.75 and out lambda for boxcox was 0.15.



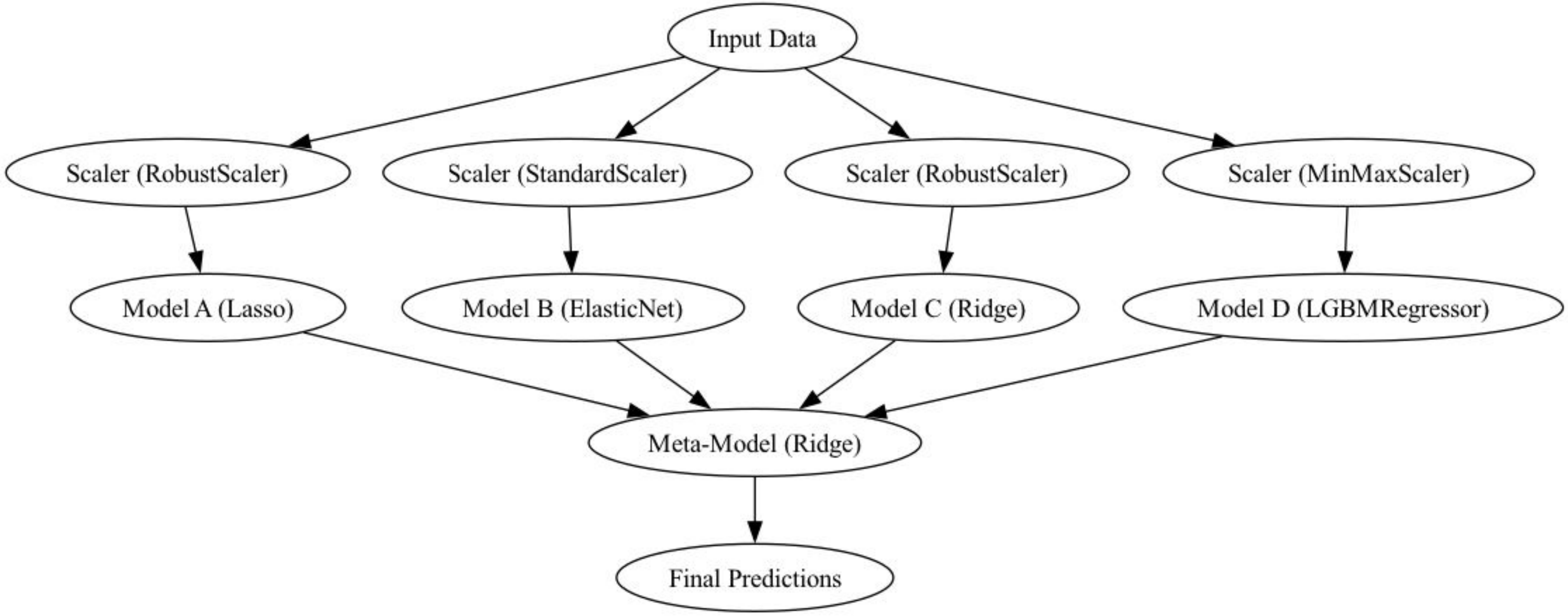
Modeling

- After OneHotEncoding, our dataset has 287 features.
- First, we tried using K-means and Gaussian Mixture to have 3 different clusters, where we train and predict each cluster differently, but this did not give good results on validation set.
- Our final model was a **Stacked Model**, with Lasso, Ridge, ElasticNet and LightGBM models and a Ridge meta model.
- These 4 models were selected after using GridSearch on several models, which also included: MLPRegressor, SVR, XGBoost, KNN and RandomForest.

Model Pipelines

- The reason we used pipelines, is so that the data would be scaled inside the pipeline itself since, each model works better with certain Scalers.
- **Lasso:** Best results with Robust Scaler.
- **Ridge:** Best results with Robust Scaler.
- **ElasticNet:** Best results with Standard Scaler.
- **LGBM:** Best results with MinMax Scaler.

Final Model


Input Data
Scaler (RobustScaler)
Scaler (StandardScaler)
Scaler (RobustScaler)
Scaler (MinMaxScaler)
Model A (Lasso)
Model B (ElasticNet)
Model C (Ridge)
Model D (LGBMRegressor)
Meta-Model (Ridge)
Final Predictions

Results

- Following are the results obtained using K-Fold CrossValidation where we use 4 folds. The differences are more pronounced using a single validation set.

Model	RMSE	MAE
Lasso	23466.32 +/- 2245.67	14388.97 +/- 816.01
ElasticNet	23347.56 +/- 2192.62	14245.75 +/- 726.31
Ridge	23117.11 +/- 2009.03	14370.70 +/- 771.35
LGBM	23280.09 +/- 1781.59	14082.24 +/- 632.56
Stacking	22186.11 +/- 2047.64	13446.57 +/- 666.99

Future Work

- The project can be extended by investigating multicollinearity issues in the data and doing further feature engineering.
- Time series effects can be analyzed on the house prices.
- External data could be integrated.

Conclusion

- In conclusion, the project demonstrated how various machine learning methods can be used for predicting house prices.
- It showed the various exploratory data analysis and the preprocessing steps necessary before training the model.
- The best performing model was the stacking regressor with lowest RMSE and MAE.
- The errors were substantially reduced after preprocessing and feature engineering and could be further reduced with better techniques.