Ames House Prices Predictions

Machine Learning Project

Alex Romero

Aungshuman Zaman

Nacho Moreno

Melanie Uhde

Index

1 – Workflow

2 – Exploratory Data Analysis

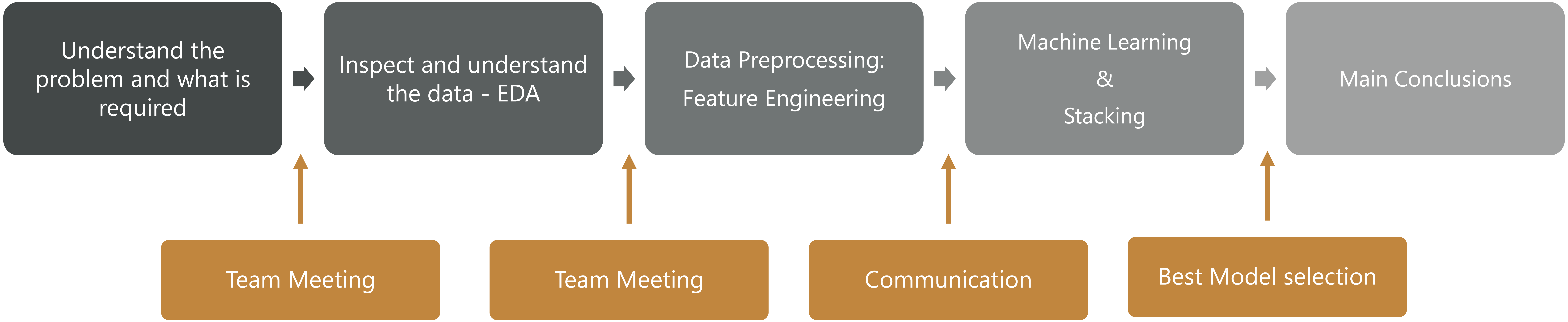
3 – Feature Engineering

4 – Machine Learning Algorithms

5 – Stacking

1- Workflow

A TEAM EFFORT ON SEPARATE TRACKS



2- Exploratory Data Analysis

EDA MAIN FINDINGS

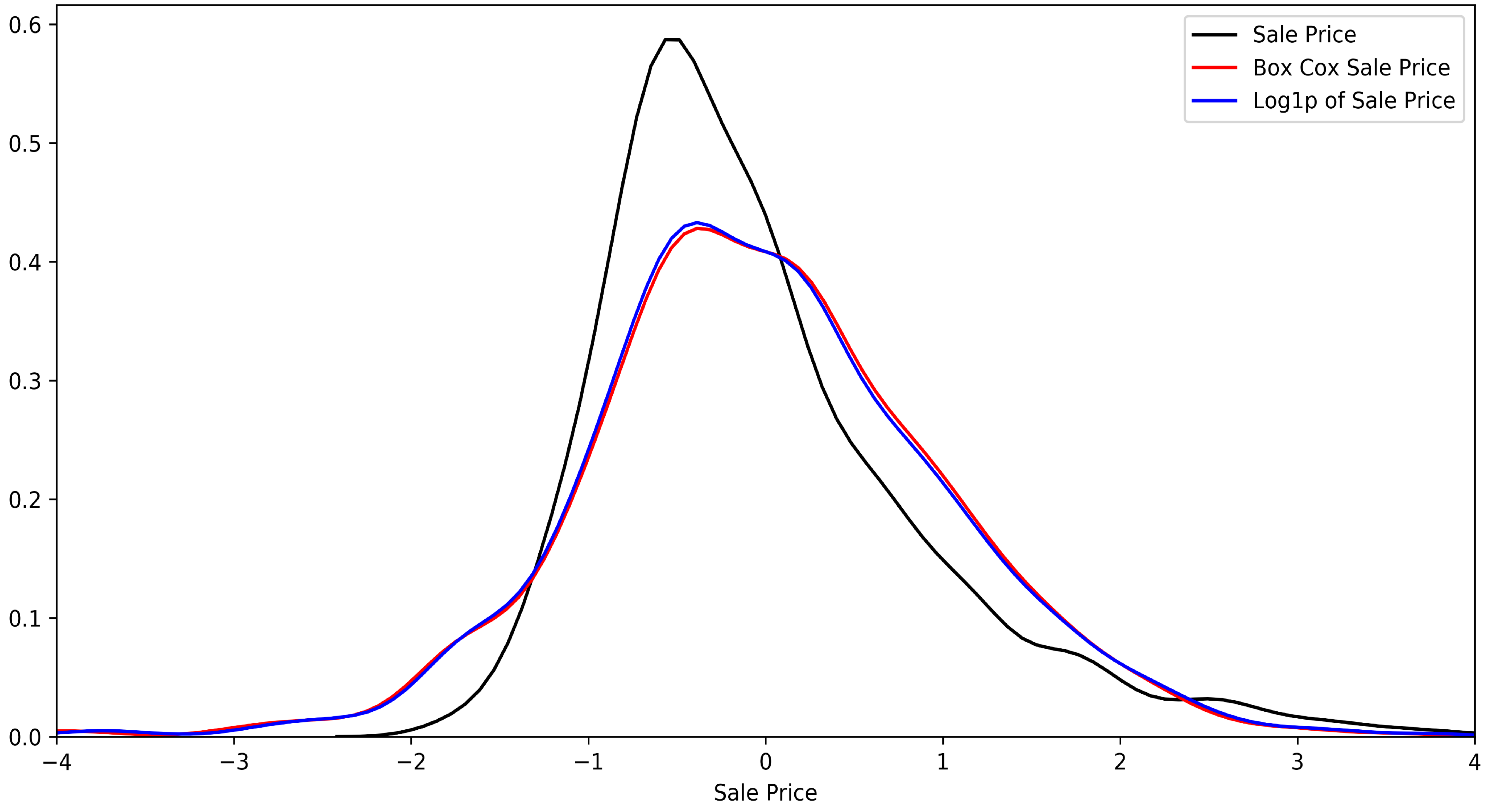
- Highly skewed Sale Price data as well as other variables
- Collinearity between variables which are similar: Garage features, Basement, Bath and Porch.
- There seem to be few important variables correlated to Sale Price: OverAllQual, Sf of living area and Garage cars/area. We will pay special attention to these variables during feature engineering.
- We identify many columns with NULL values as well categorical features that will need to be imputed.

SALE PRICES SKEWNESS

Sale Price (original) = 1.88

Box Cox = -0.-008

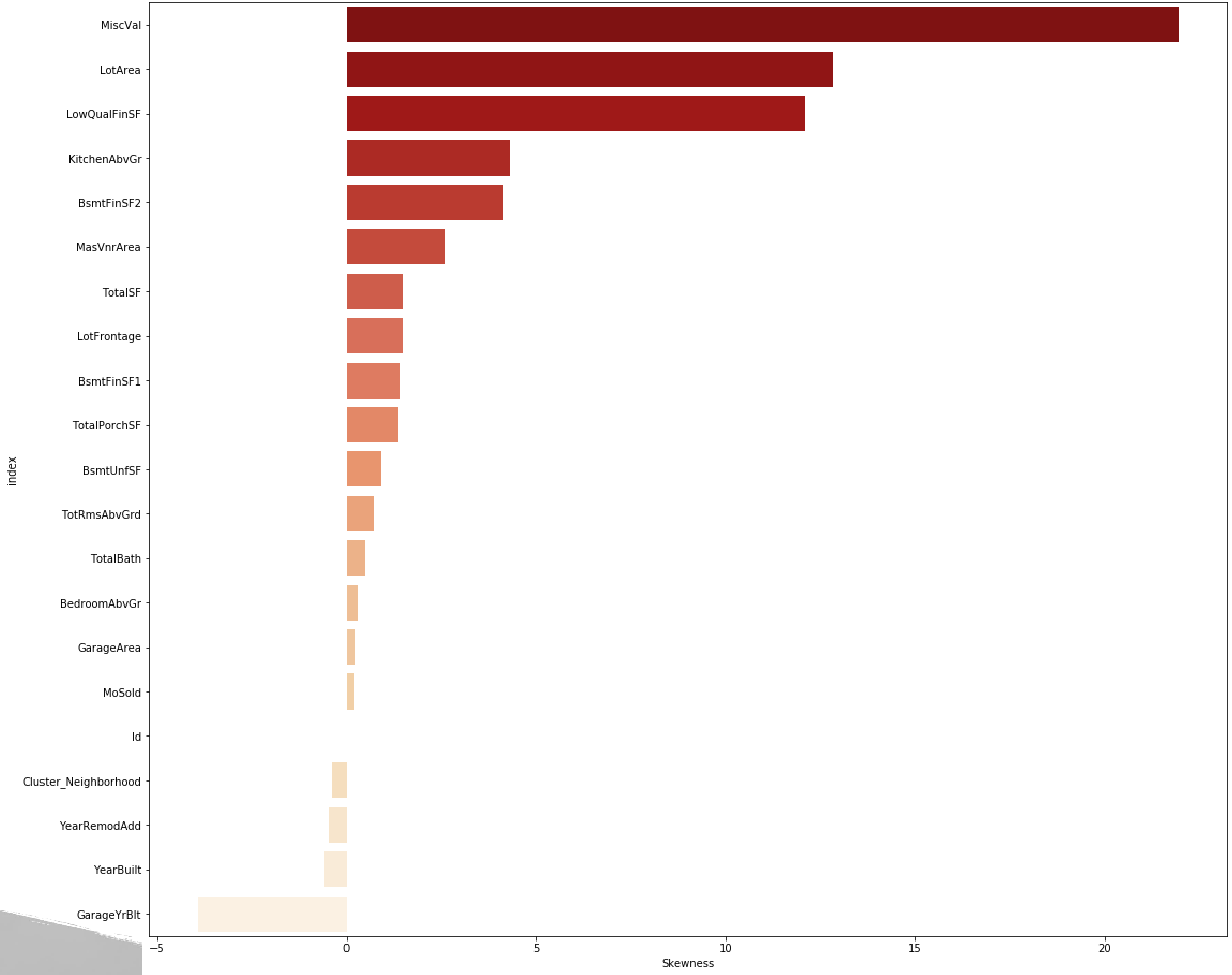
Log1p = 0.12



SALE PRICES DATA

DATA TRANSFORMATION

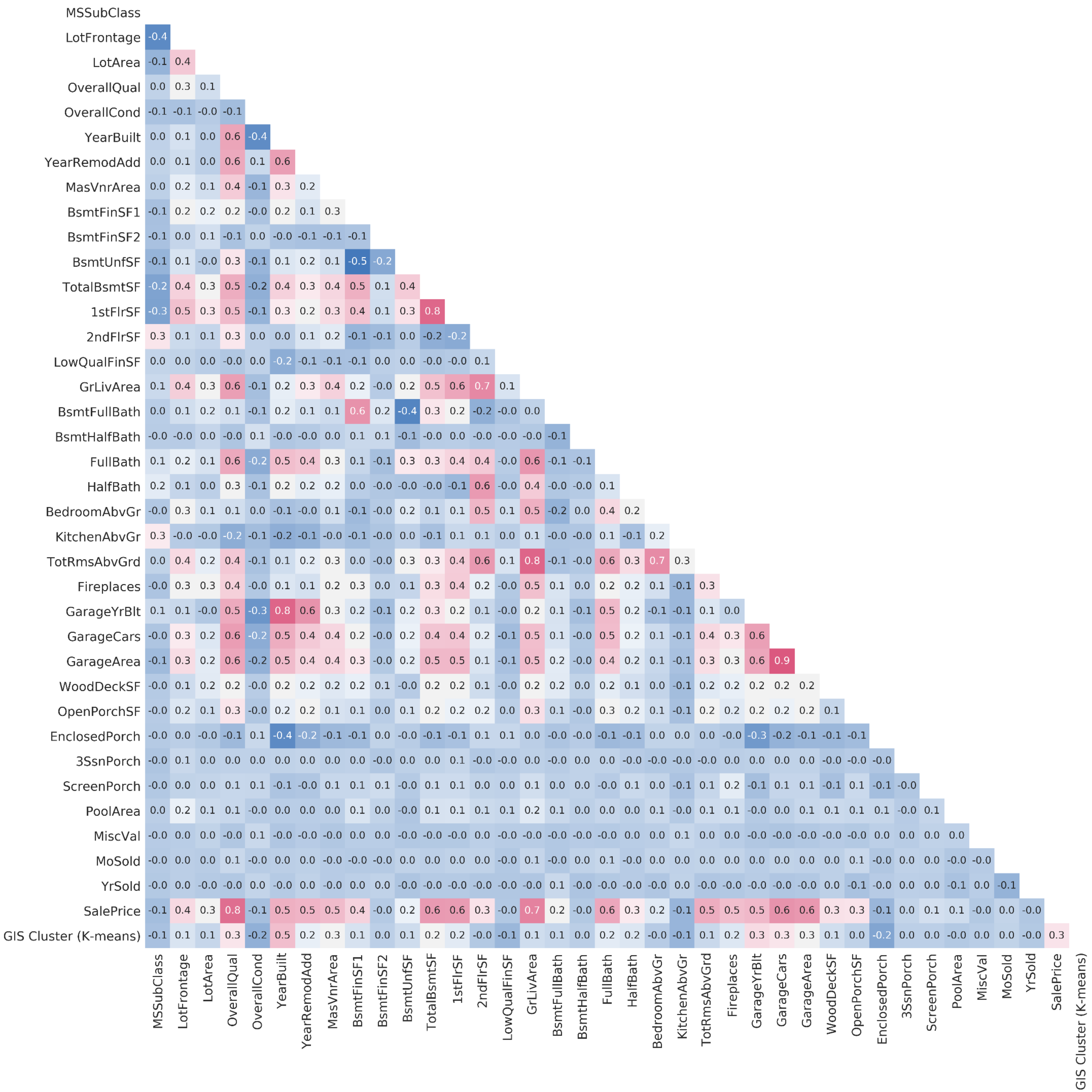
Due to the highly skewed distribution of the Sale Prices data we explored different ways to normalize the data so it can be used in linear models. As the graph shows both Box Cox and Log1p transformation gave very similar results.



Skewness

DATA TRANSFORMATION

We also identify many variables that present high skew. We will transform these for linear models.

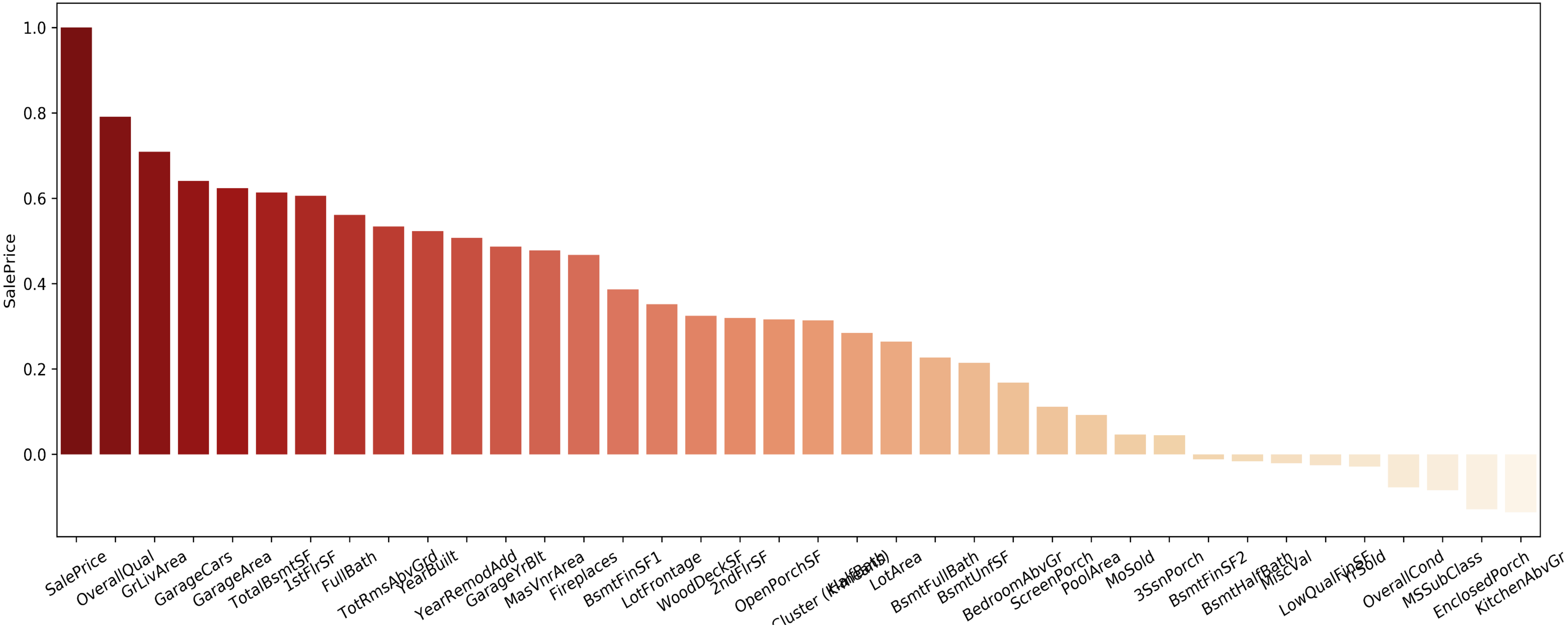


Correlation Matrix

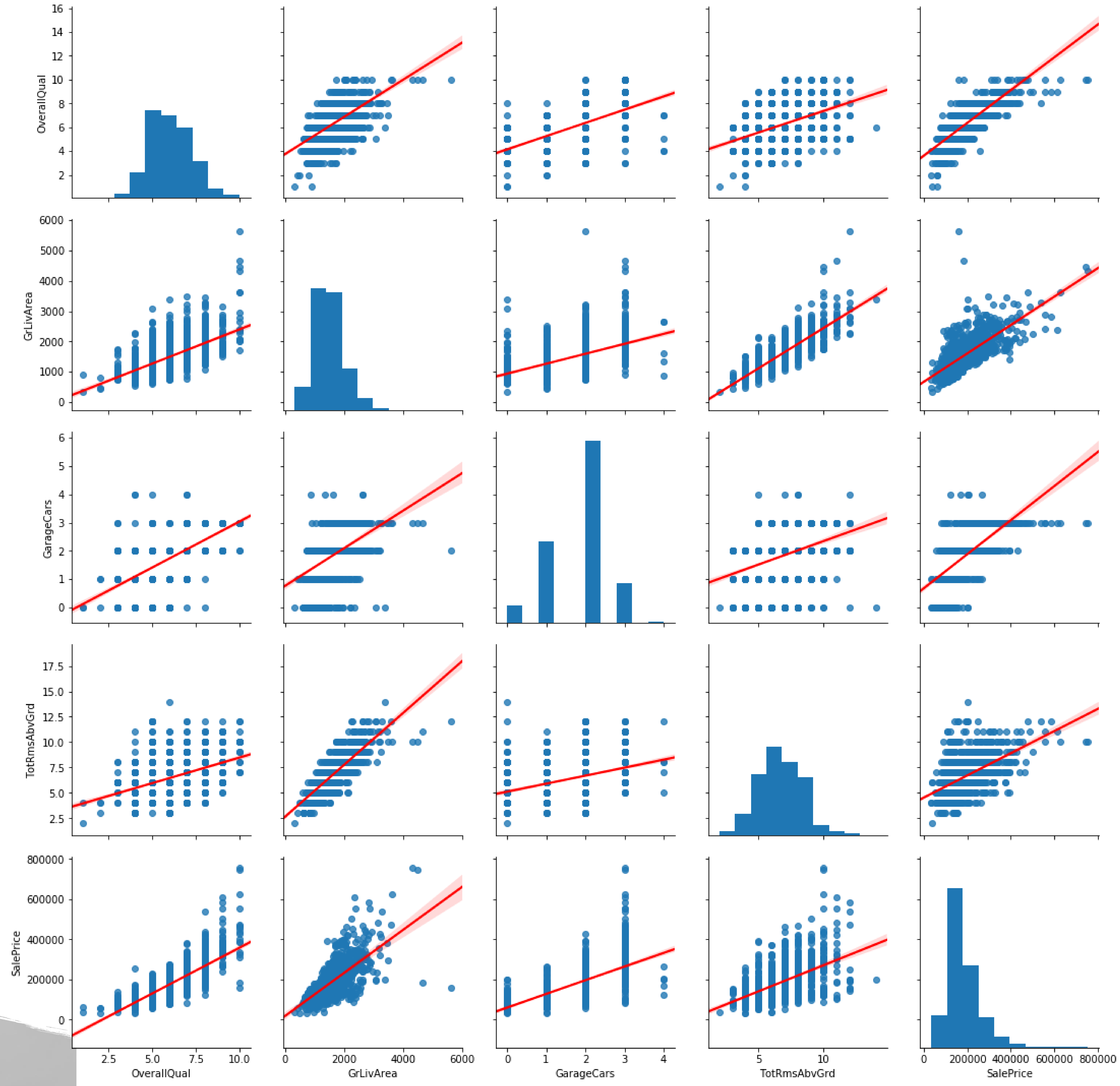
COLLINEARITY

The correlation matrix helped us to identify feature that were highly correlated like all feature related to SF, Garage, Porch and Basement. This insight will help us during the feature engineering to attempt to put together some of those.

Sale Prices Correlation



The barplot clearly shows how few variables are highly correlated with the Sale Price, among them OverAll Quality and GrLiving Area.



Main Correlated Features

MAIN FINDINGS

If we take a closer look to the main correlated variables we see as many of them have a skewed distribution (barcharts). We also identify some outliers that will need to be assessed.

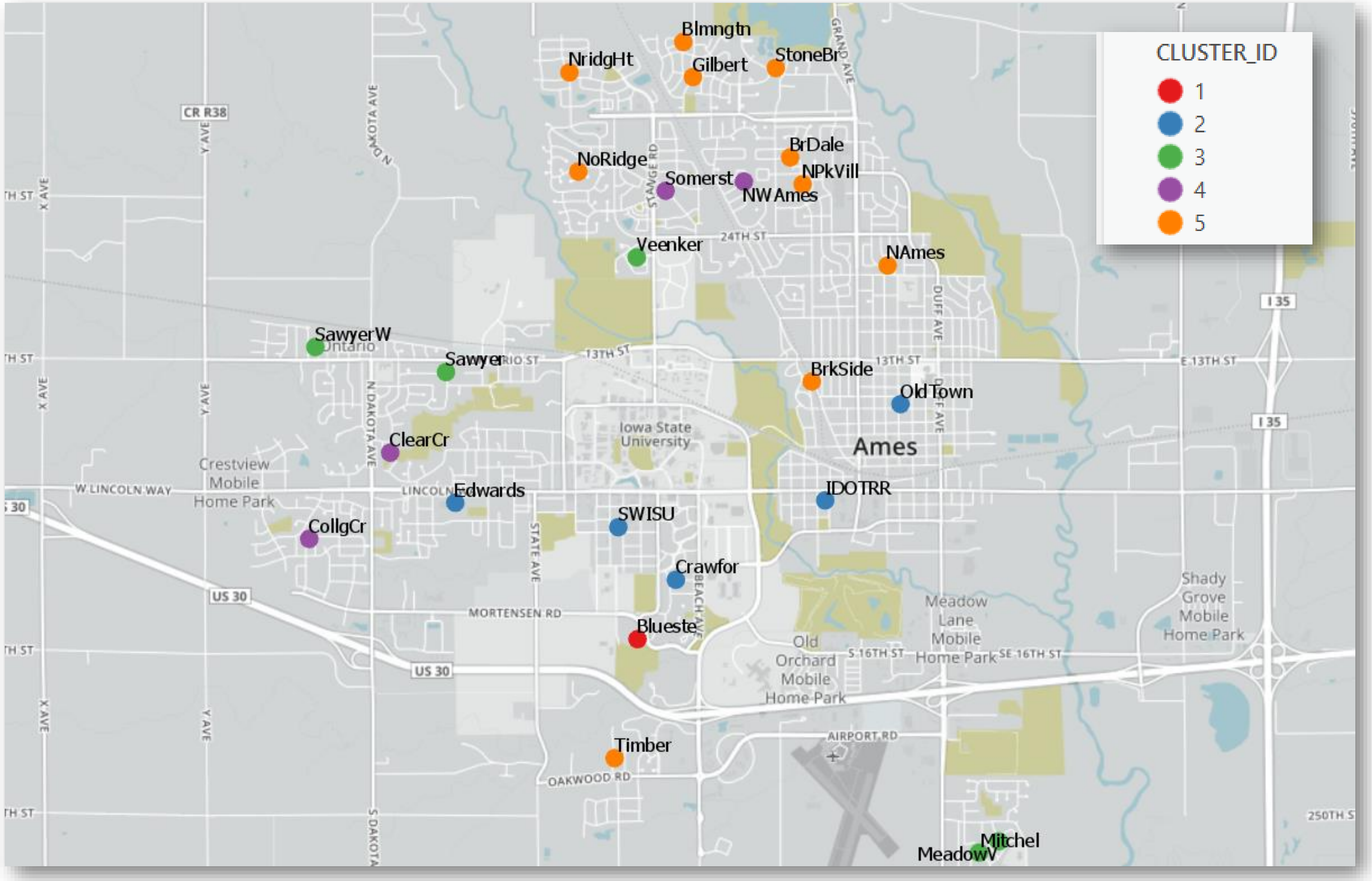
3 – Missing Values & Feature Engineering

Goals:

- Remove features that we don’t what to use in the model, just based on the number of missing values or data leakage
- Transform features into the proper format (numerical to categorical, scaling numerical, filling in missing values, etc.)
- Create new features by combining other features

DATA ENRICHMENT

DEMOGRAPHICS



	Average Household Income	Median Age	Median Household Income	Neighborhood	Per Capita Income	Food Stores (SIC54)	Eating & Drinking (SIC58)	Hotel/Lodging (SIC70)	Dominant Tapestry LifeMode Group Code	Dominant Tapestry LifeMode Group Name	Total Population
0	107120	45.0	73234	Timber	54059	1	1	1	5	GenXurban	1841
1	77663	28.3	53564	Veenker	34486	0	4	0	2	Upscale Avenues	3923
2	98860	40.0	75050	StoneBr	42476	2	5	0	4	Family Landscapes	3323

Given the lack of actual location of each of the properties we added extra information to each neighborhood and clustered them in order to get a better insight on each of them and find out if this had a relation with the sale price.

Using ArcPy we were able to enrich each of the neighborhoods with the demographic then we use the Multivariate Clustering tool (K-Means) to group the neighborhoods in 5 groups information.

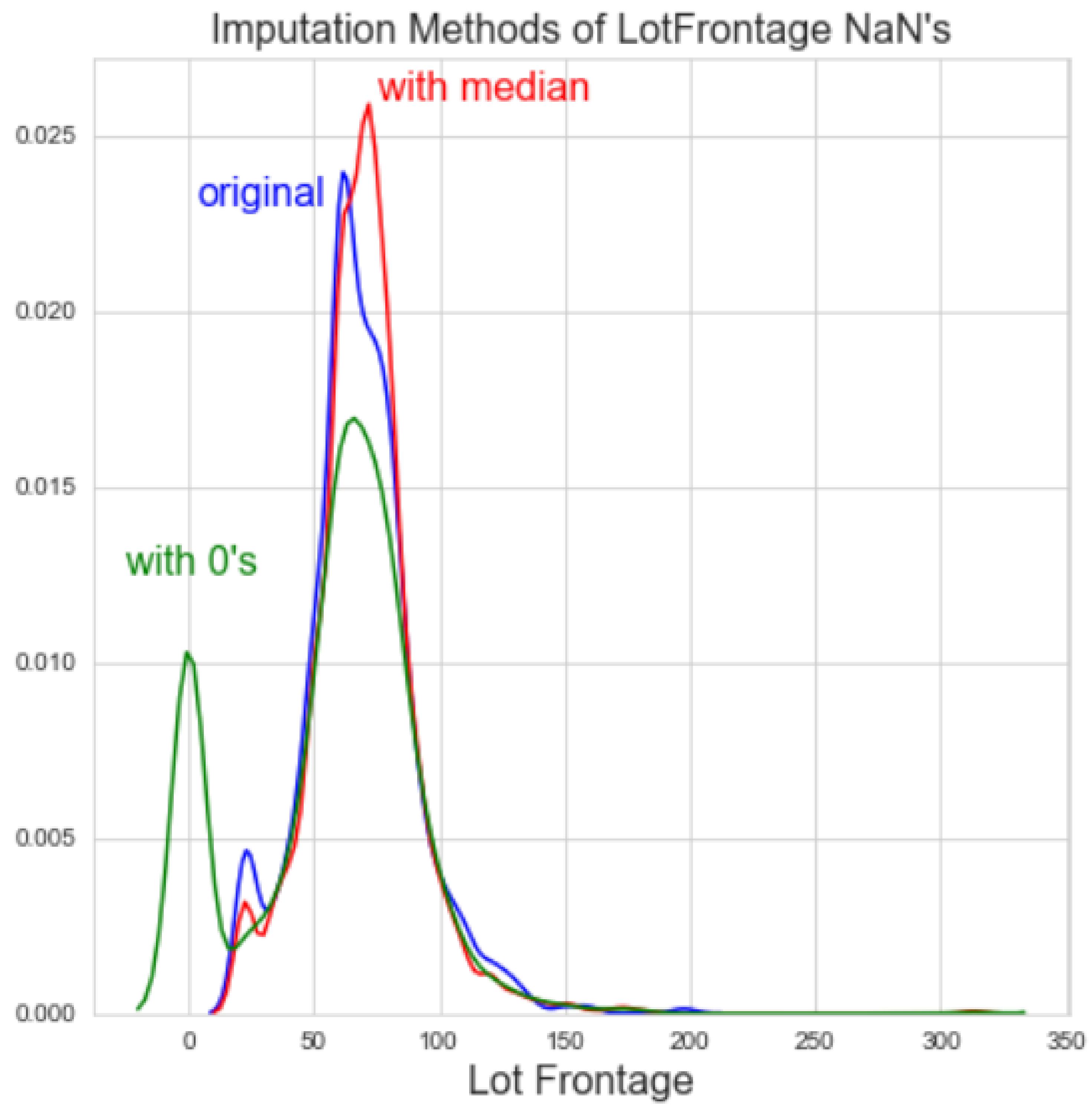
This feature enrichment though didn't have a great impact on our models.

Missing Values

- Some features have over 80% of values being missing.

	Missing Ratio
PoolQC	99.725180
MiscFeature	96.427345
Alley	93.198214
Fence	80.419100
FireplaceQu	48.746135
LotFrontage	16.660941

- In the case of `PoolQC`, `NaN` is when `PoolArea` is 0, or there is no pool.
  - Other variables have a similar "missingness" relationship, e.g. Garage-related columns.

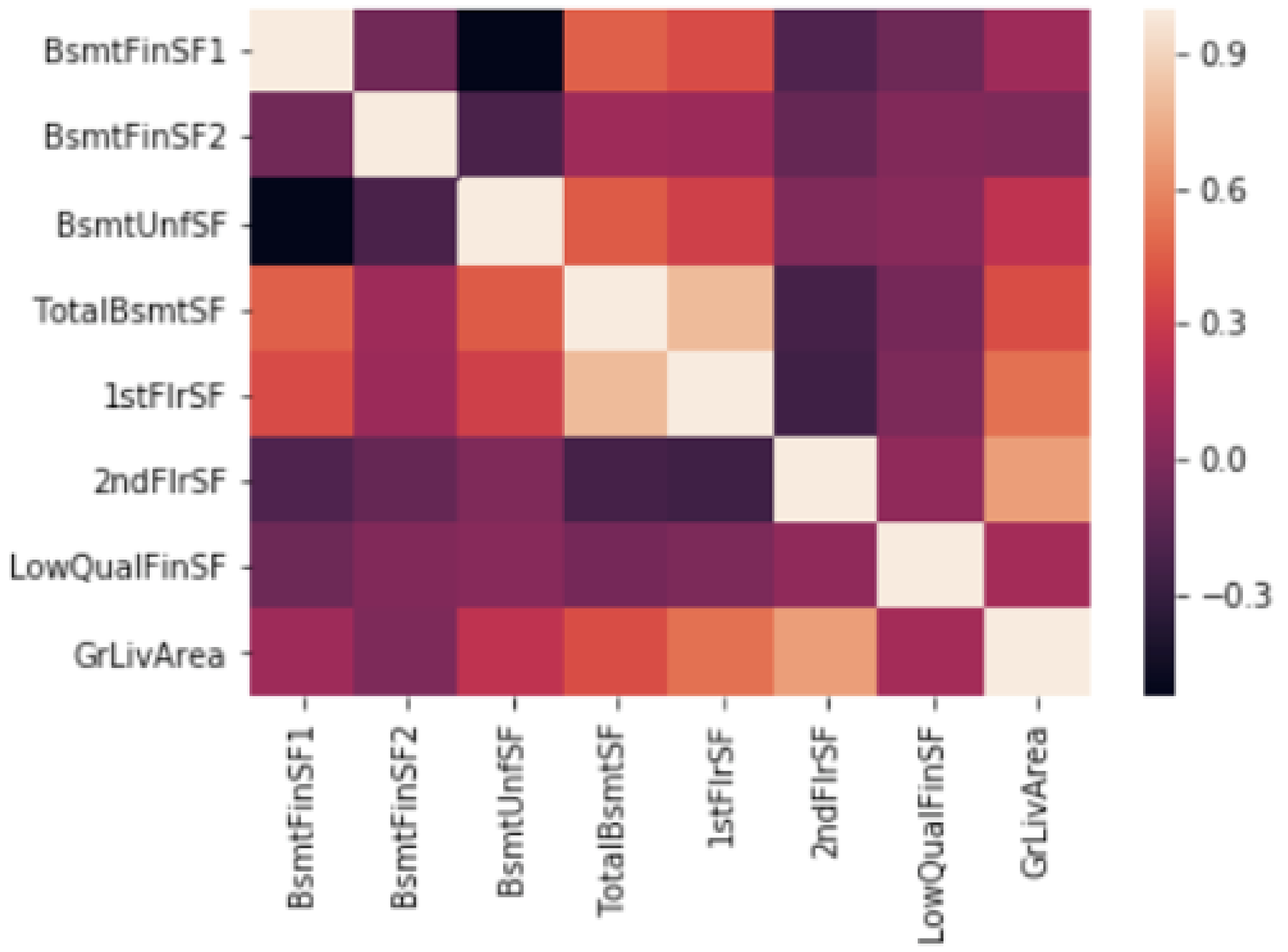


- Some features do not truly deserve a `0`. In the case of `LotFrontage`, for example, we can impute with median of `SalePrice` per `Neighborhood`.

Feature Engineering

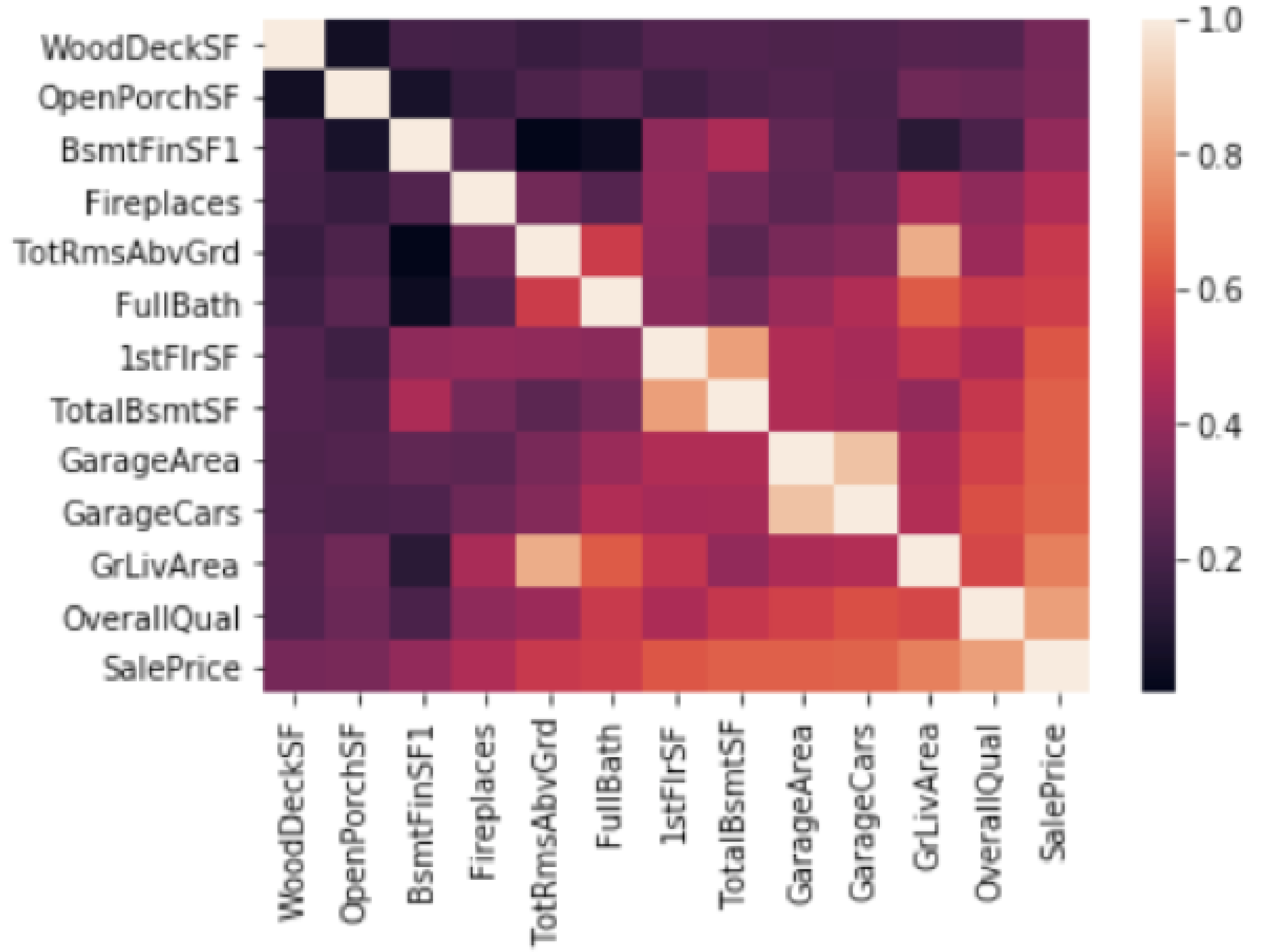
- How can we avoid *multicollinearity*? Some features can be explained by others and aid in creating new features and dropping some.

Square Footage Features



BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	Add Pinks	TotalBsmtSF
468	144	270	882	882
923	0	406	1329	1329
791	0	137	928	928
602	0	324	926	926
263	0	1017	1280	1280
0	0	763	763	763
935	0	233	1168	1168
0	0	789	789	789
637	0	663	1300	1300
804	78	0	882	882

Features with > 0.3 Correlations with Target



`GrLivArea` and `TotRmsAbvGrd`
`GarageArea` and `GarageCars`

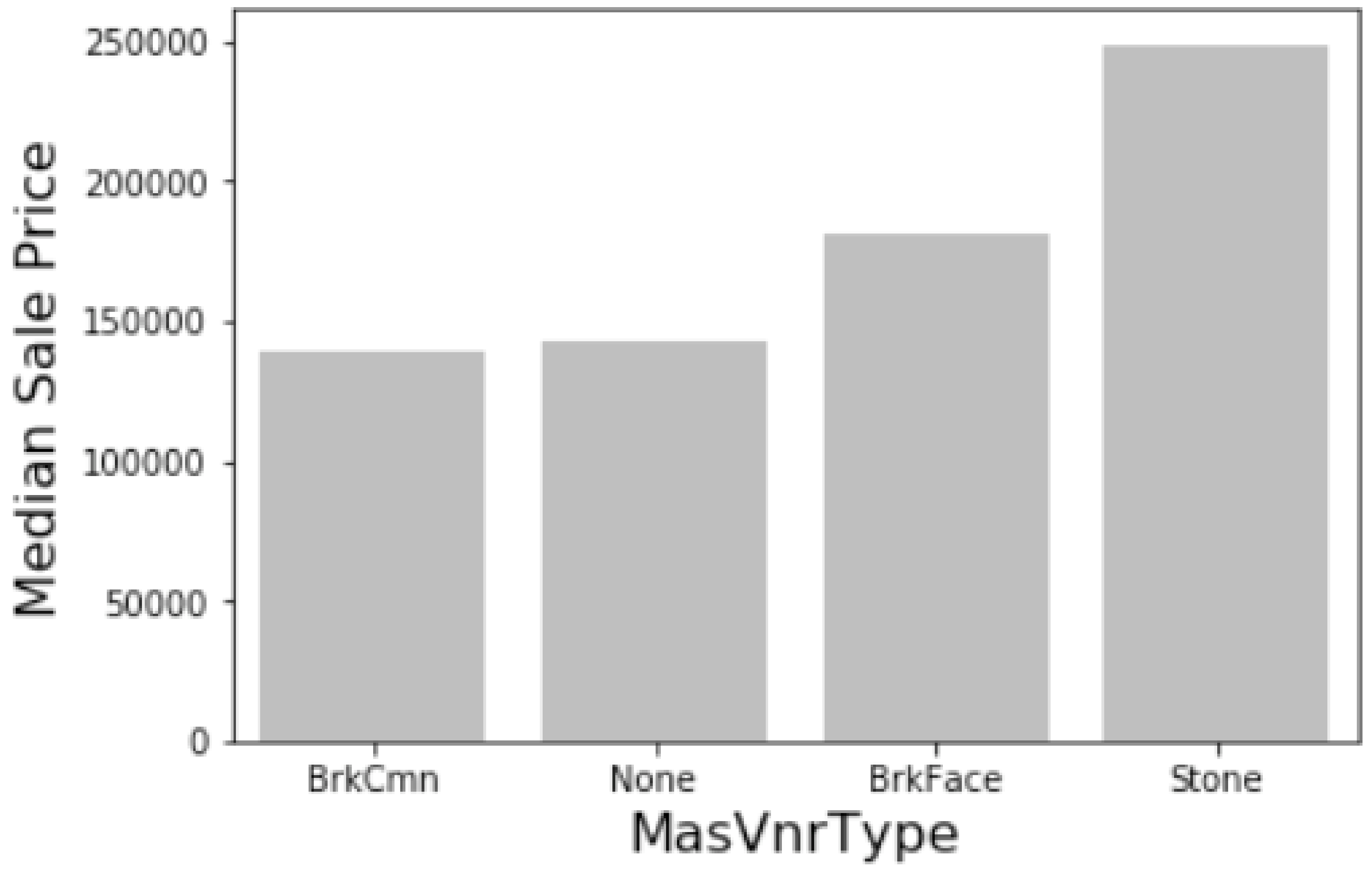
Because these are continuous variables and are strongly correlated, we drop `TotRmsAbvGrd` and `GarageCars` to avoid nuance.

Feature Selection (Encoding/Factorize)

- There are many variables that describe the quality or condition of a particular feature
  - Examples: `BasementQC`, `PoolQC`, GarageCond – these are ordinal and can be encoded.

```
ord_cols = ['ExterQual', 'ExterCond','BsmtCond', 'BsmtQual','HeatingQC',
            'KitchenQual', 'FireplaceQu', 'GarageQual', 'GarageCond', 'PoolQC']
ord_dic = {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa':2, 'Po':1, 'None': 0}
```

- Some features have multiple categories as values. While encoding, we can check if there are possible groupings of redundant categories.



After imputing missing values with mode

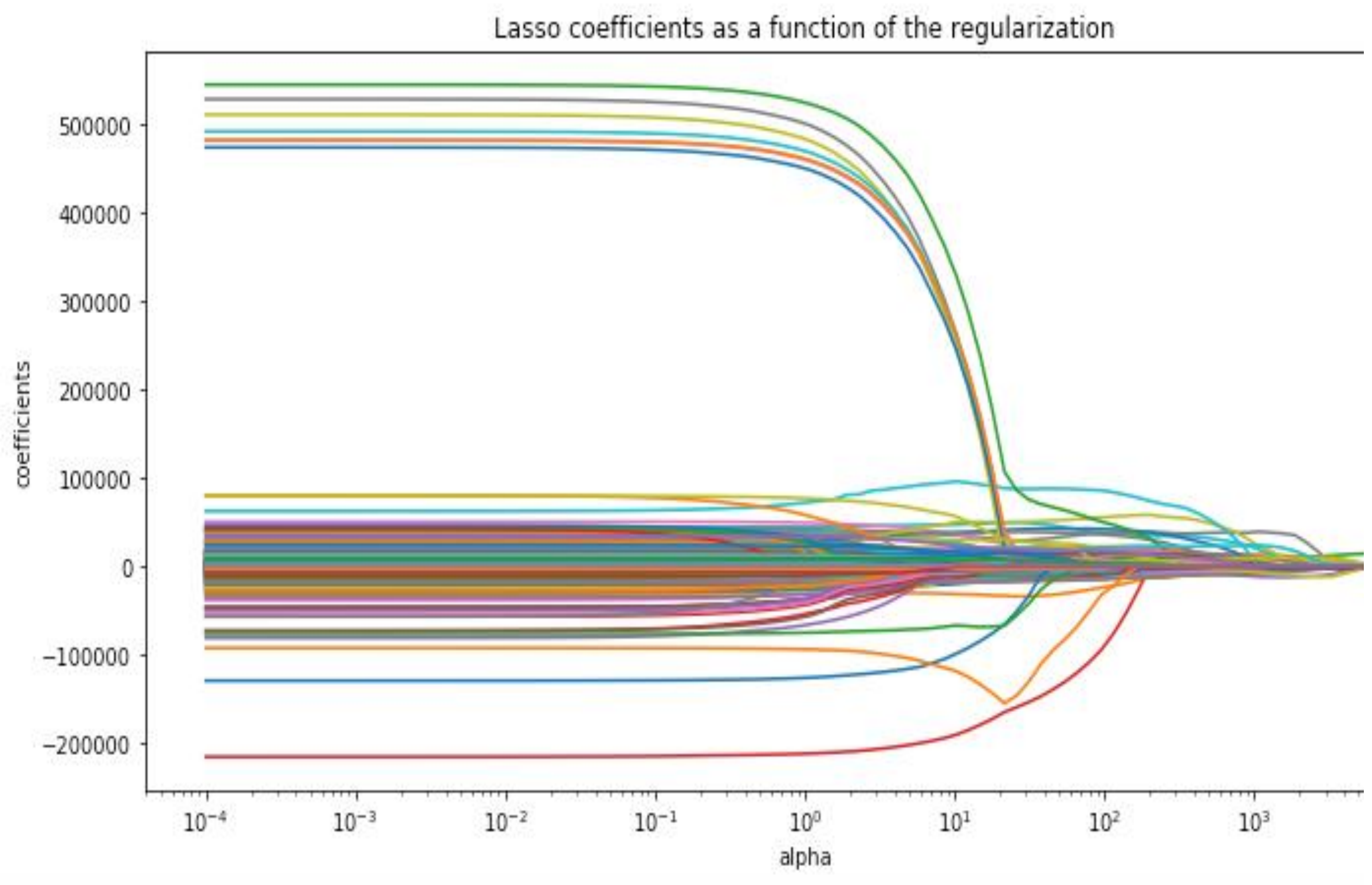
```
all_data["MasVnrType"] = all_data['MasVnrType'].map({'BrkCmn':1, 'None':1, 'BrkFace':2, 'Stone':3})
```

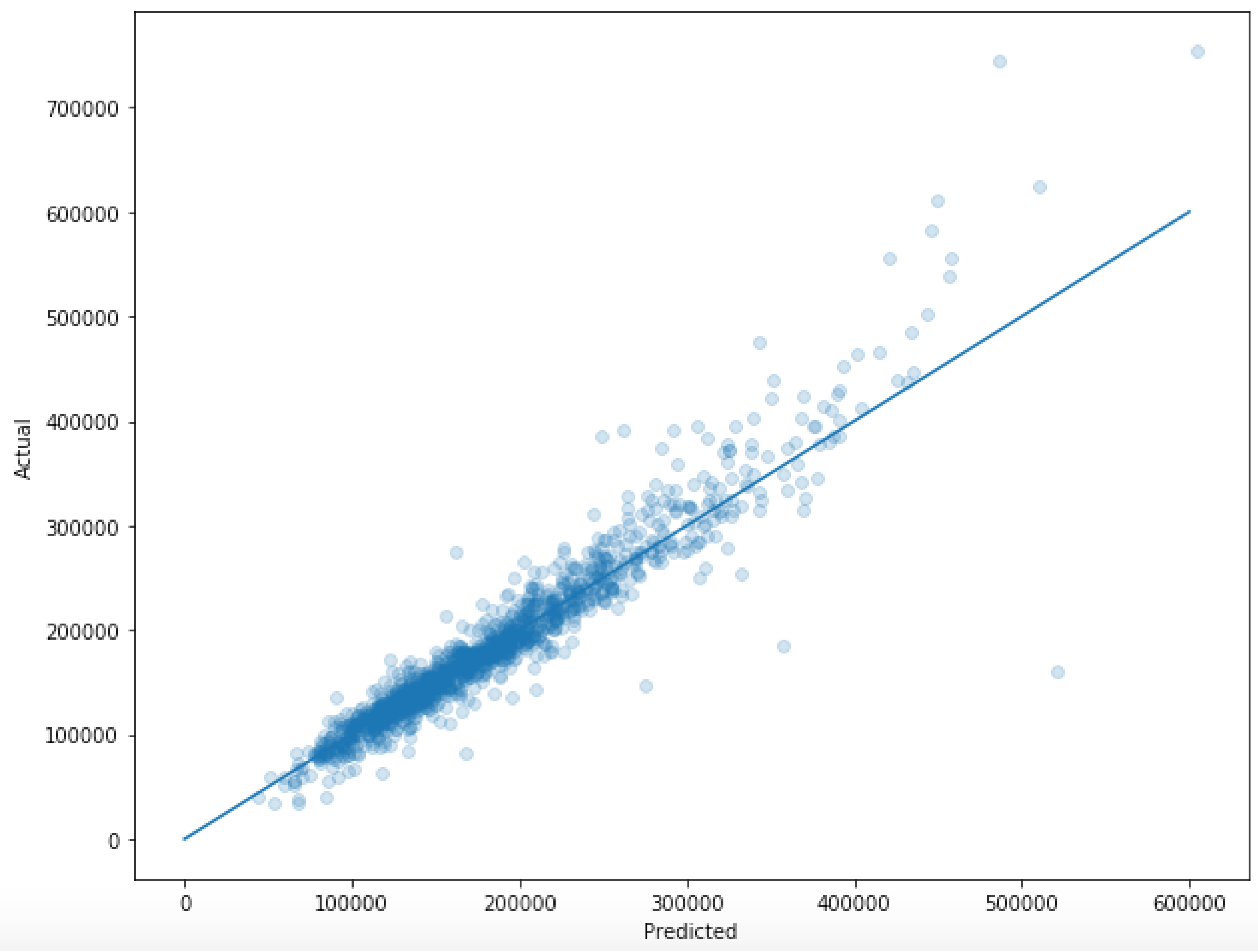
Modeling: Linear Regression

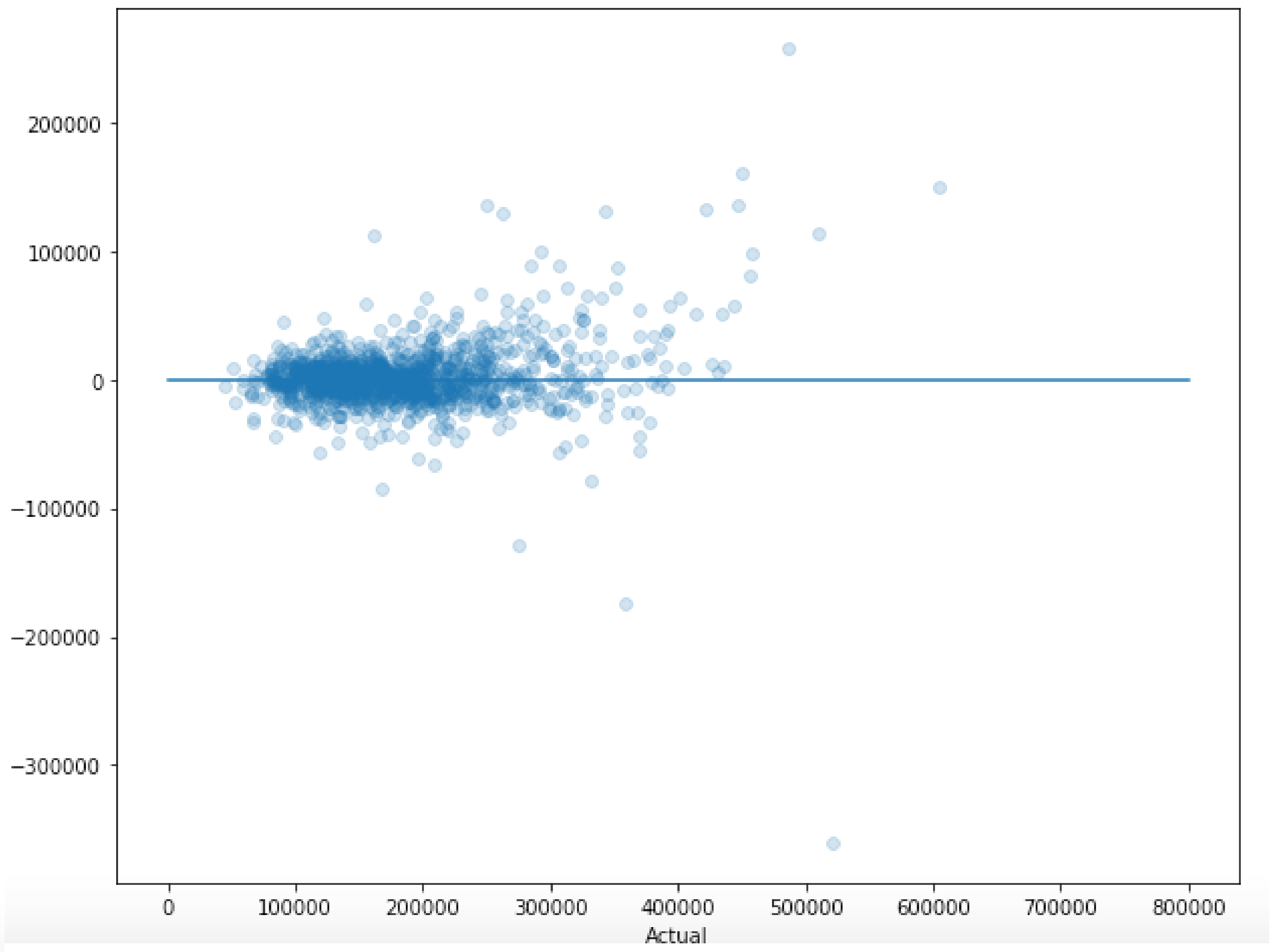
- Now that we have a clean dataset with desired features, we can start modeling.
- The first models tried were Regularized linear models: Ridge, Lasso, Elastic net
- Lasso regression was very useful in trimming down the number of features

Tuning parameter
Alpha in scikit learn

$$RSS + \lambda \sum_{j=1}^{p} |\beta_j|$$



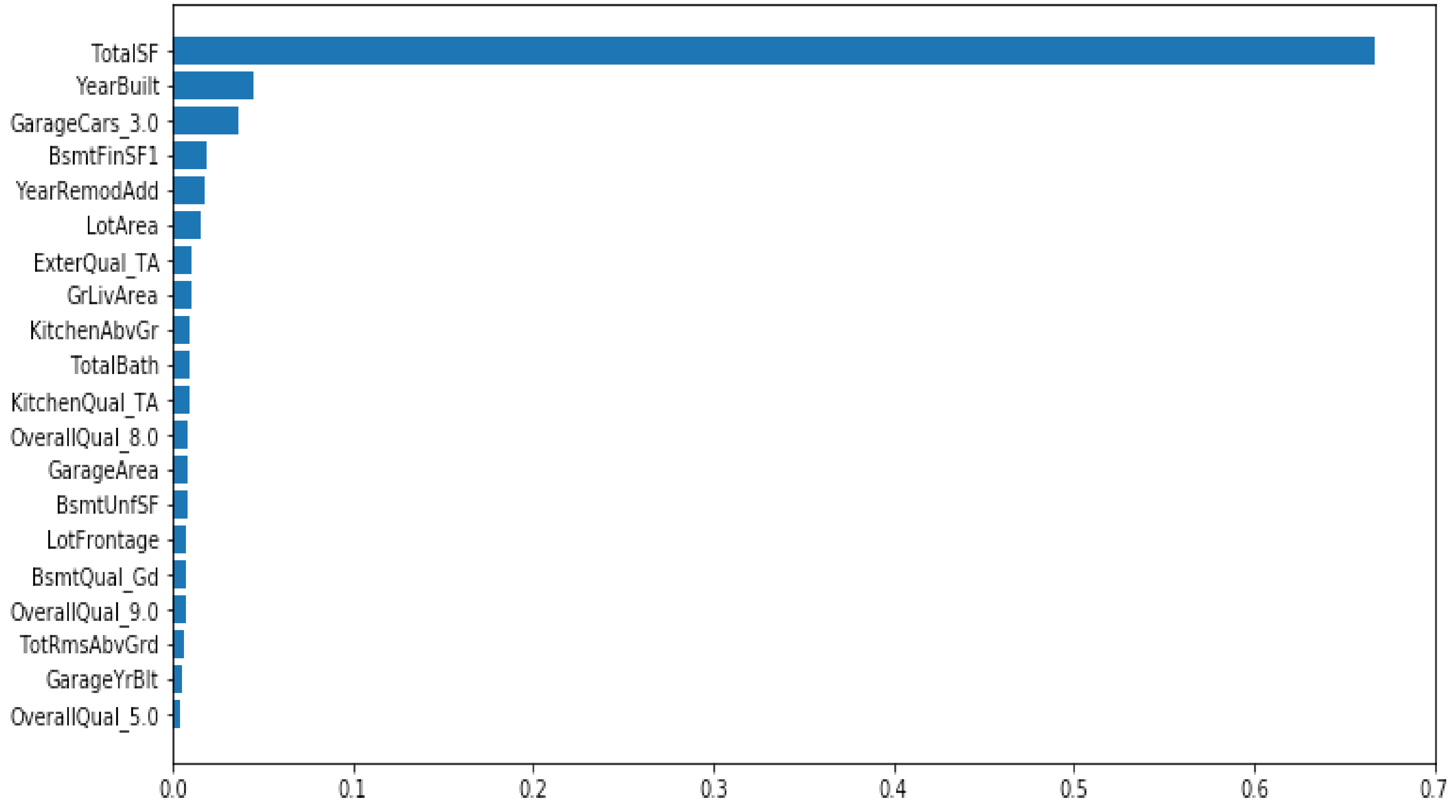


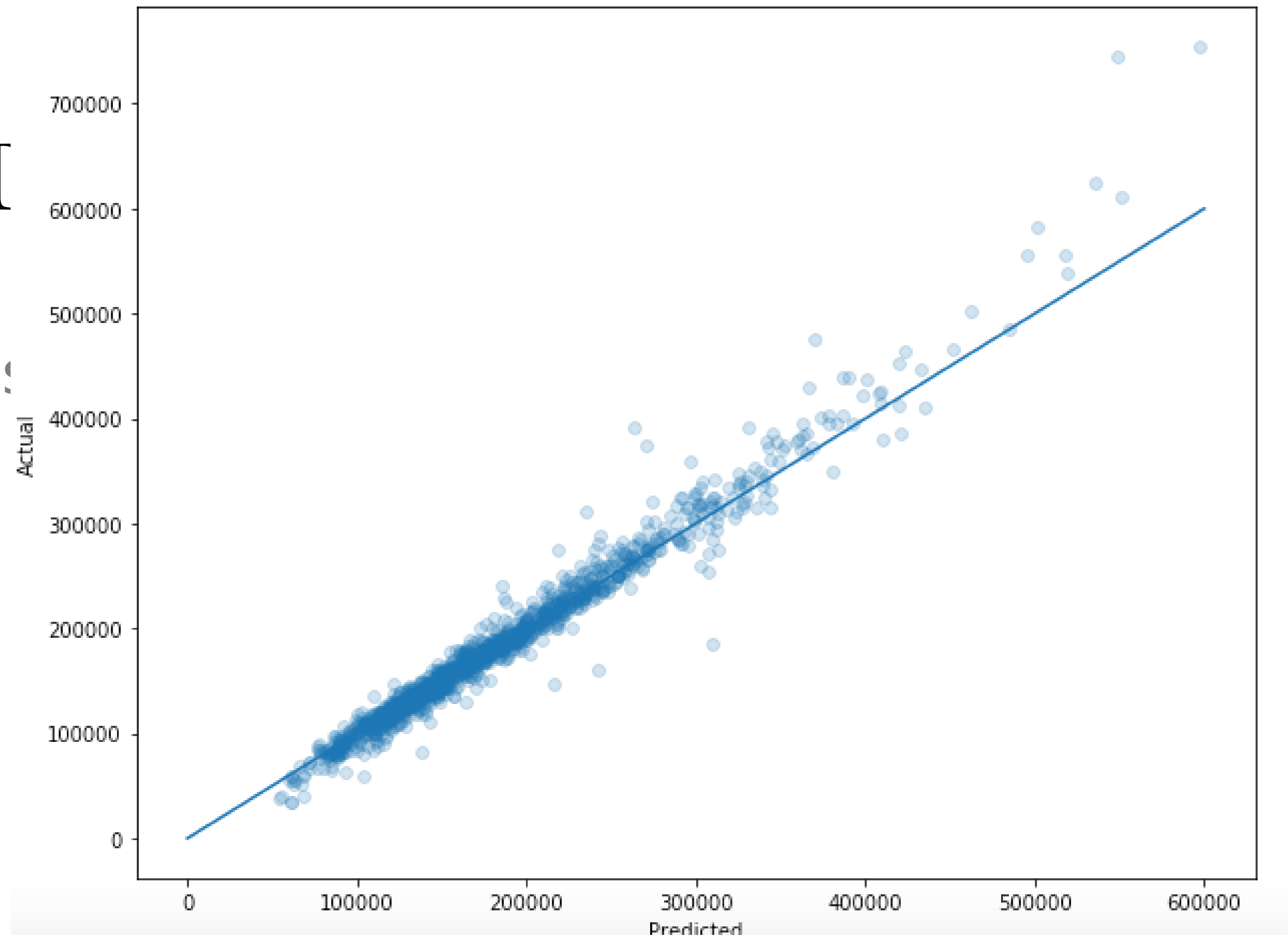


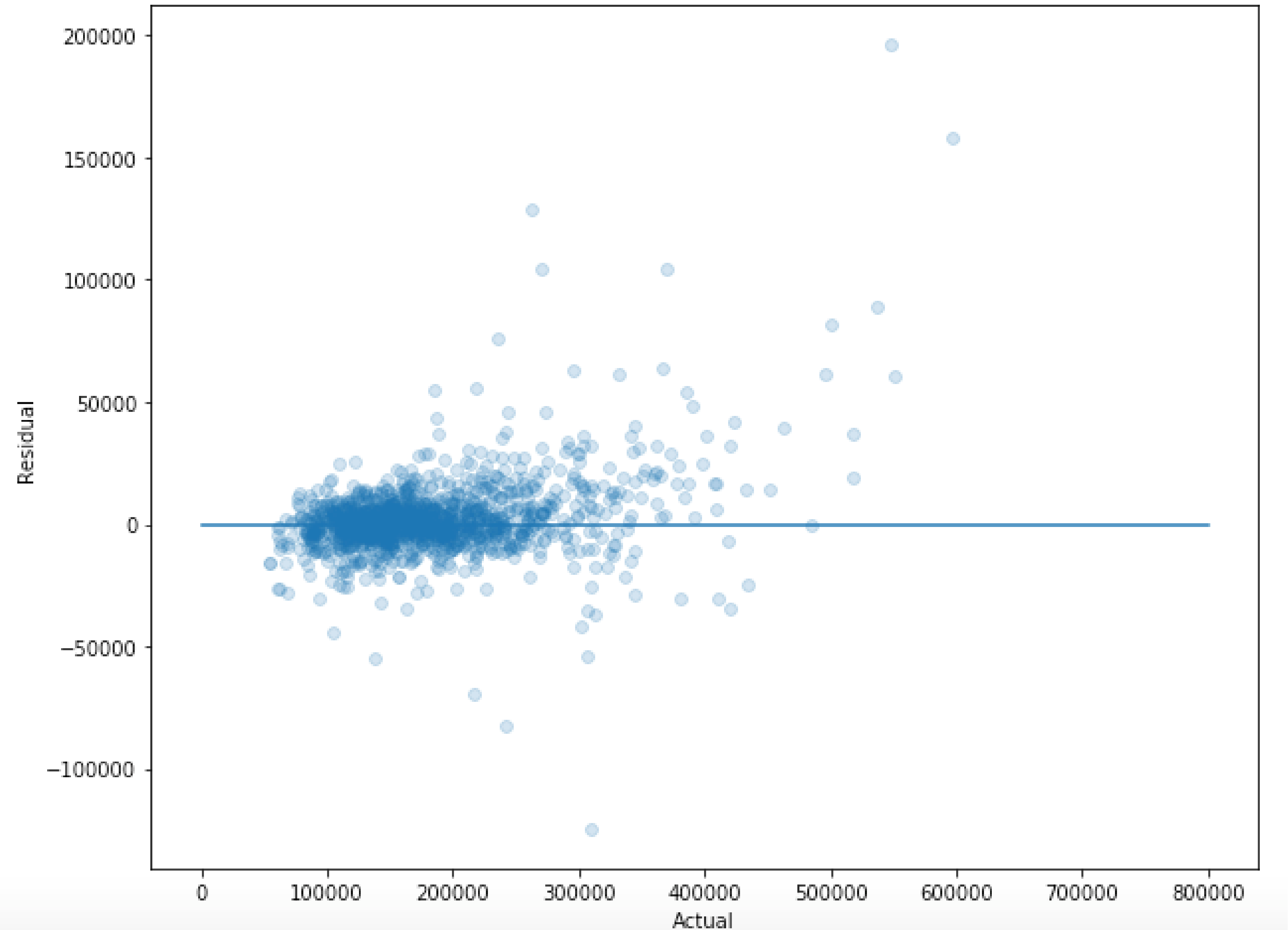
- Using Box-Cox transformation for ”SalePrice” improves quality of linear fit.
- Employed 5-fold cross-validation to find the best value of hyperparameter alpha.

Modeling: Random Forest

- Random Forest provides a list of features important for regres
- Most important ones are: TotalSF, YearBuilt, GarageCars







- Employed grid search to tune hyperparameters:- n_estimators = 25, min_sample_leaf =2, min_samples_split =6
- Random Forest scores were not quite as good as results from linear regression

THE FINAL BOOST

- Gradient Boost
- XGBoost:
  - Regularized boosting
  - Splits up to max_depth before pruning backwards
  - Allows cv at each iteration
- Light GBM:
  - Faster and lower memory usage
  - complex trees by following leaf wise split approach

STACKING

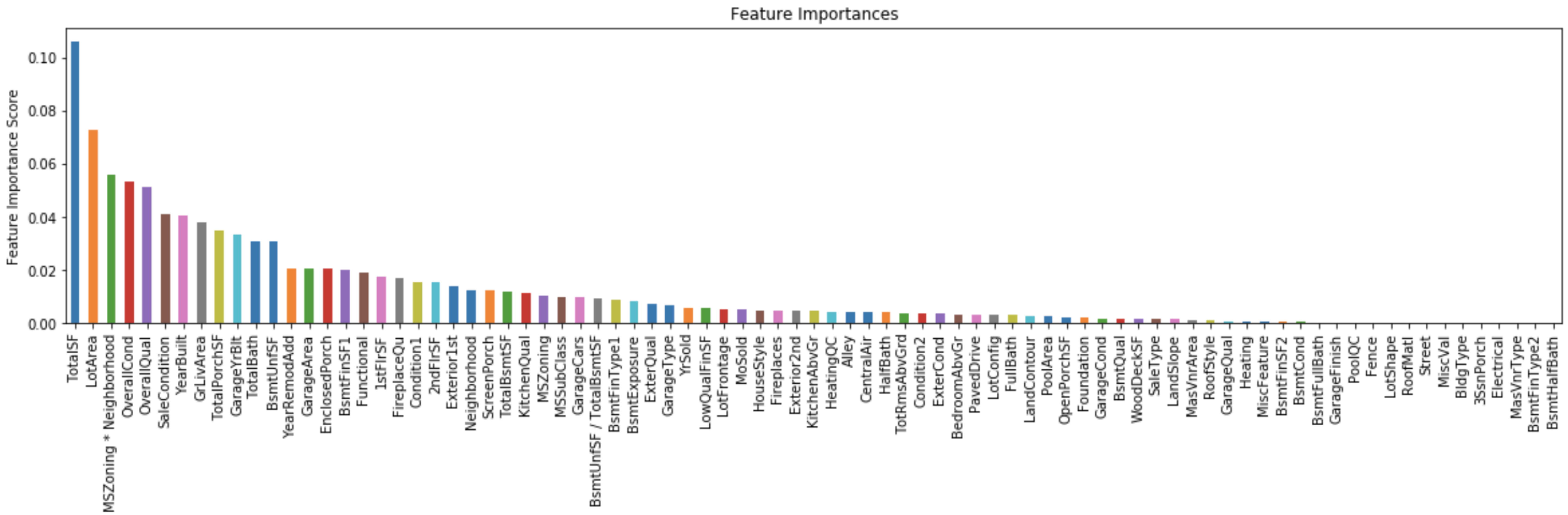
- Gradient Boost
- XGBoost
- Light GBM

THE FINAL BOOST

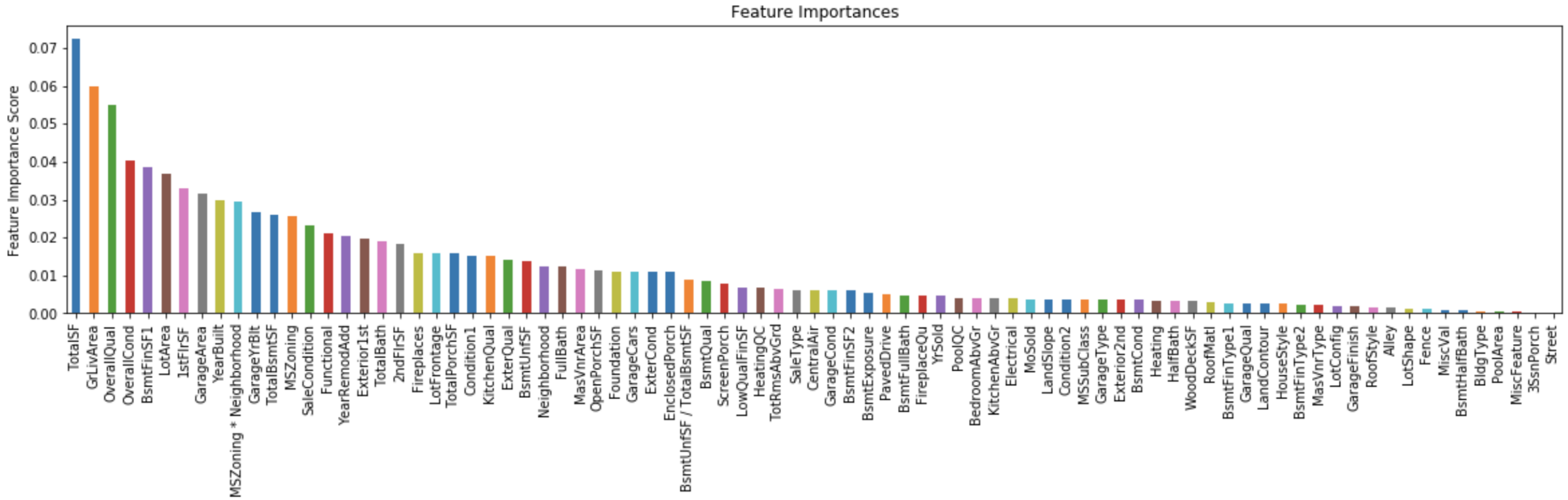
- relatively **high learning rate** (started with default of 0.1)
- Determined **optimum number of trees for this learning rate** (that also allows the system to run fairly fast)
- **Tune tree-specific parameters** for decided learning rate and number of trees
- **Lower the learning rate** and increase the estimators proportionally to get more robust models

- Gradient Boost
- XGBoost
- Light GBoost

THE FINAL BOOST



GradientBoost before tuning



GradientBoost after tuning

Ames House Prices Predictions

TEAM INTERGREAT