

# Machine Learning Engineer Nanodegree

## Capstone Project – KAGGLE Housing Price Predictions using advanced Regression Techniques.

---

Leon Paul

November 31st, 2017

### I. Definition

#### Project Overview

---

Regression analysis has been in wide use since the start of the 19<sup>th</sup> century beginning with the application of the Ordinary Least Squares method, by Legendre and Gauss, to determine the orbits of comets around of Sun based upon astronomical observations <sup>[1]</sup>. Regression methods have seen widespread usage within two key areas, namely **Forecasting** and **Optimization**, with a view to extract valuable insights and drive business decisions.

Regression is a supervised Machine Learning approach wherein an algorithm is fed with a matrix of input features and a response label. These inputs will be correlated with the response variables in varying degrees, i.e. some may have positive correlations while others may have zero or even negative correlations. The regression algorithms will attempt to learn these relationships between the inputs (independent variables) and the response label (dependent variable). Once the algorithm has learnt these relationships, it will be able to take in a brand-new set of inputs that it has never seen before and predict a continuous output with a certain level of accuracy. My aim, as a Machine Learning engineer, will be to select the best possible model and also try and improve its accuracy to the best of my abilities.

Forecasting the sale price of a house given a variety of information pertaining to it and it's neighbourhood is a prime importance in the present, with companies like Zillow and ZipRealty using empirical data to forecast estimates of property prices for their customers in real time. The ability to create models which can make these predictions with as high an accuracy as possible is what gives these organizations a competitive edge. Some previous applications of regression applications in predicting/modelling house prices have been seen in "Modelling Home Prices Using Realtor Data, 2008, Iain Pardoe" <sup>[5]</sup> and "House Price Prediction: Hedonic Price Model vs. Artificial Neural Network, 2004, Visit Limsombunchai". <sup>[6]</sup> I plan to use these academic papers as guides for my capstone.

I have an interest in forecasting trends or predictions within the real estate, financial and retail sectors and as a result I decided to work within this area for my Capstone. The aim will be to work on a dataset containing features that describes various aspects taken into consideration when evaluating a property and use regression techniques to predict its price

## Problem Statement

For my Capstone, I will be working with the Ames Housing dataset <sup>[2]</sup>. This dataset consists of 79 explanatory features describing a variety of informative details about residential houses sold in the city of Ames, Iowa between 2006 and 2010. The datasets have been made available through a Kaggle competition "**House Prices: Advanced Regression Techniques**" <sup>[3]</sup>.

The problem requires us to predict the Sale Price of a house given the set of 79 explanatory features. The solution submitted to Kaggle will be evaluated by them using the Root Mean Square Error between the logarithm of the predicted value and the logarithm of the observed sales price. Taking the logarithm will normalize the errors ensuring that the prediction errors for expensive and cheap houses will be weigh equally on the model's evaluation.

$$RMSE = \sqrt{E((\log(\hat{\theta}) - \log(\theta))^2)}$$

$\theta$  := Actual Sale Price

$\hat{\theta}$  := Predicted Sale Price

The dataset in question is available freely from a number of sources. I was able to download the training and testing datasets from the Kaggle Competition webpage <sup>[3]</sup>. The dataset consists of 79 input features which are a combination of categorical and numeric data types. The response variable is a continuous value that represents the Sale Price of the houses. The datasets have the following characteristics:

### TRAINING DATASET:

- **1460 rows**
- **79 input features**
- **1 response feature ~ Sale Price**
- **43 Categorical features**
- **36 Numerical features**

### TESTING DATASET:

- **1459 rows**
- **79 input features**
- **43 Categorical features**
- **36 Numerical features**

The response variable will be modelled using the given 79 input features. The key area of focus will be on Feature engineering and regression models that minimize the RMSE score.

The essential approach would be to perform Exploratory Data Analysis in order to determine the distribution of the dataset's numerical features, determine the number features having missing values as well as the percentage of missing values. Following this, the next step would be to use feature engineering to transform, create or drop features so that any feature that does not contribute significantly to a model's information gain would not be used and help reduce the dataset's

dimensionality. Lastly, a series of regression models would be run using the response variable as a target label and a combination of the pre-processed predictors to train the model. Some of the models I aim to try are Multivariate Linear Regression and Decision Trees etc. Cross-Validation and ensemble methods like Random Forests, Regularization or Gradient Boosting would be used to improve the performance and reduce overfitting. Finally, I will use the Evaluation Metric of RMSLE to select the best possible model or combination of models (e.g. stacking/averaging) that minimize the error/ maximize the accuracy.

## Metrics

The Evaluation metric used for this problem would be the logarithmic Root Mean Square Error (RMSLE). RMSE is defined as *"measure of the differences between values (sample and population values) predicted by a model or an estimator and the values actually observed"* [4]. It is represented as follows:

$$RMSE = \sqrt{E((\hat{\theta} - \theta)^2)}$$

$\theta := \text{Actual Feature Value}$

$\hat{\theta} := \text{Predicted Feature Value}$

The logarithmic RMSE is calculated simply by using the logarithms of the predicted and actual values, as shown below:

$$RMSLE = \sqrt{E((\log(\hat{\theta}) - \log(\theta))^2)}$$

This serves to normalize the results ensures that errors from expensive or cheap houses are given equal weightage while improving the model performance. Since the Sale price values are huge; in the region of \$100,000 to \$1,000,000; the errors between actual and predicted values could be very large and taking the logarithm of these values ensures that such large errors are not penalized.

In addition, KAGGLE will be using this metric to evaluate my submissions and hence I will be using this metric to evaluate my models.

## II. Analysis

---

### Data Exploration

The datasets for this competition are provided by Kaggle in the form of a training dataset with the 'SalePrice' label provided and a testing dataset with only input features. As discussed earlier, the input space consists of 79 input features, some numeric and others categorical. These features describe various details like the condition of the house and various areas of it like the basement, kitchen, garage, 1<sup>st</sup> floor or 2<sup>nd</sup> floor if present. There are features which define the square foot areas of different parts of the houses. Categorical features are present to describe features like materials

used for the roofing of the house, the garage or the type of neighbourhood or nearest type of road (e.g. arterial road, main road etc.) A detailed description of these features has been provided via a data dictionary <sup>[7]</sup> whose link has been made available in the References section.

The first item of interest is to explore the response label viz. 'SalePrice'

```
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
```

From here I can tell that the SalePrice label are provided for 1460 houses. The mean selling price for a house in Ames, Iowa is about \$180,000 while the most expensive house was sold for \$755,000. This is indicative that the data is not Normally distributed. I will use a visualization to explore this later in the next section.

Let's also take a look at all the features available.

```
Index([u'Id', u'MSSubClass', u'MSZoning', u'LotFrontage', u'LotArea',
       u'Street', u'Alley', u'LotShape', u'LandContour', u'Utilities',
       u'LotConfig', u'LandSlope', u'Neighborhood', u'Condition1',
       u'Condition2', u'BldgType', u'HouseStyle', u'OverallQual',
       u'OverallCond', u'YearBuilt', u'YearRemodAdd', u'RoofStyle',
       u'RoofMatl', u'Exterior1st', u'Exterior2nd', u'MasVnrType',
       u'MasVnrArea', u'ExterQual', u'ExterCond', u'Foundation', u'BsmtQual',
       u'BsmtCond', u'BsmtExposure', u'BsmtFinType1', u'BsmtFinSF1',
       u'BsmtFinType2', u'BsmtFinSF2', u'BsmtUnfSF', u'TotalBsmtSF',
       u'Heating', u'HeatingQC', u'CentralAir', u'Electrical', u'1stFlrSF',
       u'2ndFlrSF', u'LowQualFinSF', u'GrLivArea', u'BsmtFullBath',
       u'BsmtHalfBath', u'FullBath', u'HalfBath', u'BedroomAbvGr',
       u'KitchenAbvGr', u'KitchenQual', u'TotRmsAbvGrd', u'Functional',
       u'Fireplaces', u'FireplaceQu', u'GarageType', u'GarageYrBlt',
       u'GarageFinish', u'GarageCars', u'GarageArea', u'GarageQual',
       u'GarageCond', u'PavedDrive', u'WoodDeckSF', u'OpenPorchSF',
       u'EnclosedPorch', u'3SsnPorch', u'ScreenPorch', u'PoolArea', u'PoolQC',
       u'Fence', u'MiscFeature', u'MiscVal', u'MoSold', u'YrSold', u'SaleType',
       u'SaleCondition', u'SalePrice'],
      dtype='object')
```

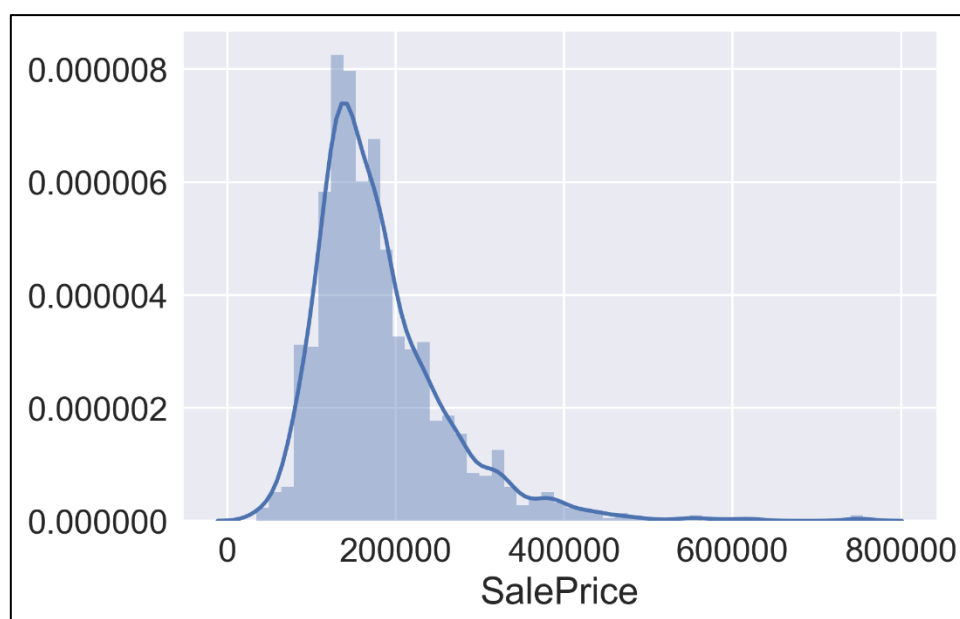
At a first glance, I tried to use my intuition and domain knowledge to determine which features would heavily influence the SalePrice of a house. Based upon my judgement, the following features would be important:

1. GrLivArea - The square foot area of General Living spaces in the house.
2. 1stFlrSF - The square foot area of the 1st floor of the house
3. 2ndFlrSF - The square foot area of the 2nd floor of the house
4. OverallQual - The Overall quality of the house
5. FullBath - The number of full bathrooms in the house.
6. YearBuilt/YearRemodAdd - The Year in which the house was built/remodelled
7. GarageArea - The square foot area of the garage.
8. Neighborhood - The quality of the neighbourhood.

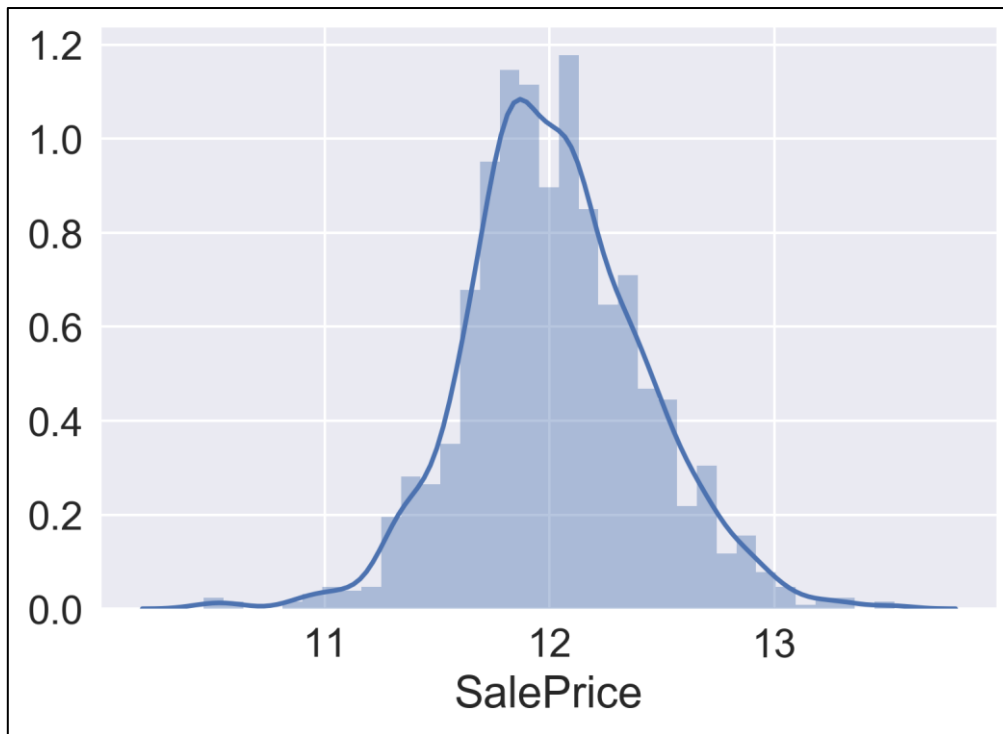
The rest of the features, while also important, would not be as influential as the above features in deciding the SalePrice. The next step would be to use visualisations on order to gain more insight into the dataset

## Exploratory Visualization

The first step is to visualize the SalePrice variable since the summary statistics suggested that it was not normally distributed.

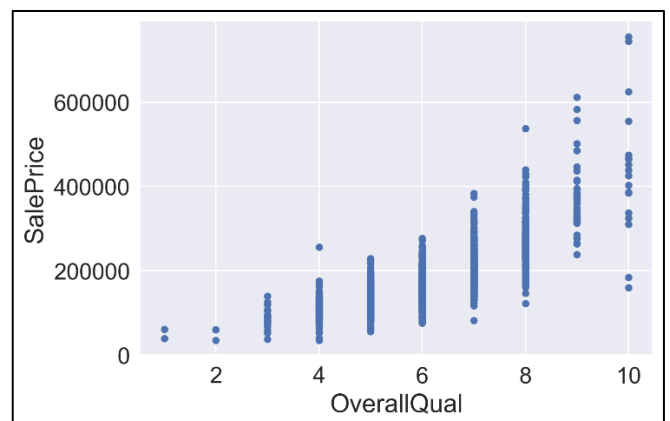
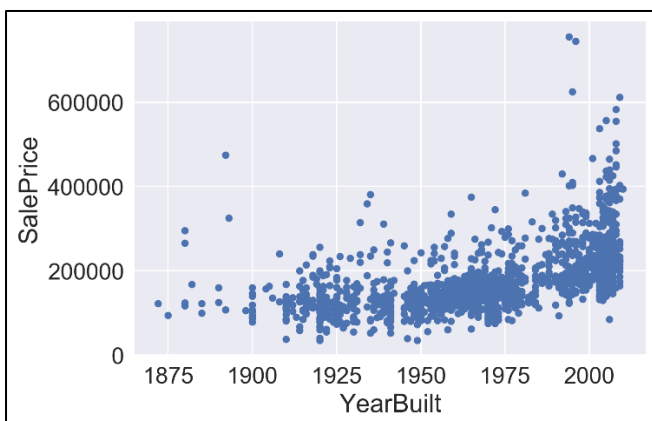
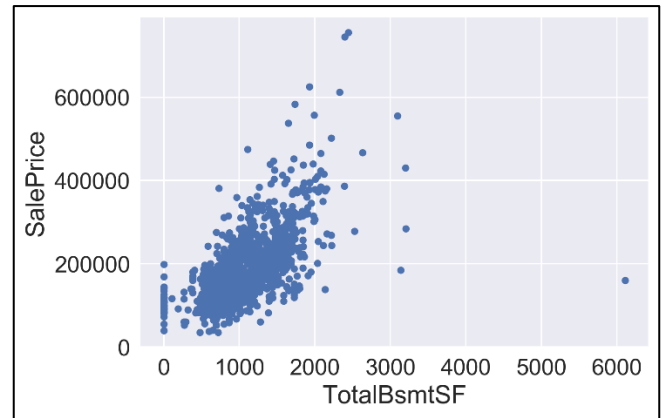
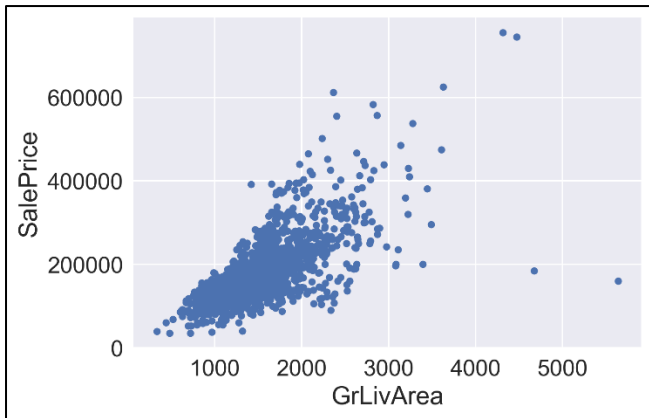


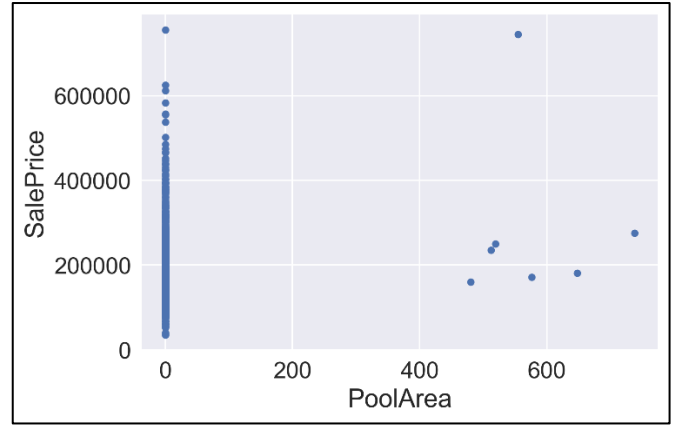
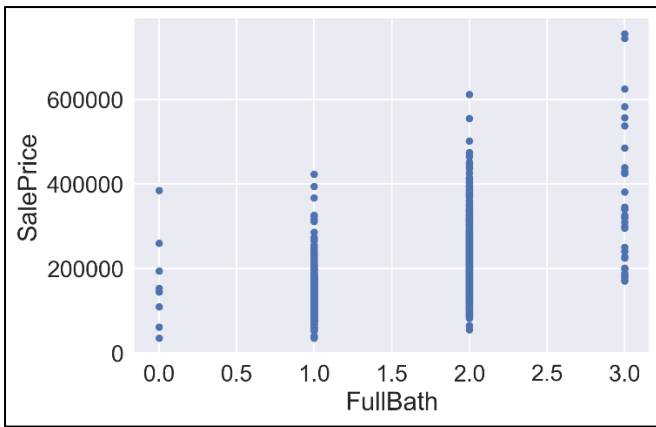
This plot confirmed my suspicion. The response variable is heavily left skewed. The SalePrice had a skew of 1.88 and a kurtosis score of 6.54. This would be an issue while fitting any model since methods like Ordinary Least Squares work on the assumption that the data and its residuals are normally distributed. Hence, I will log transform the SalePrice variable to normalize it and make it easier to fit into a Linear regression model.



This is much better. The transformed variable has a skew of 0.63 and kurtosis of 1.88 which is more acceptable. The next step would be to explore my selected variables in the Data Exploration section and visualize

their relationship with the SalePrice.

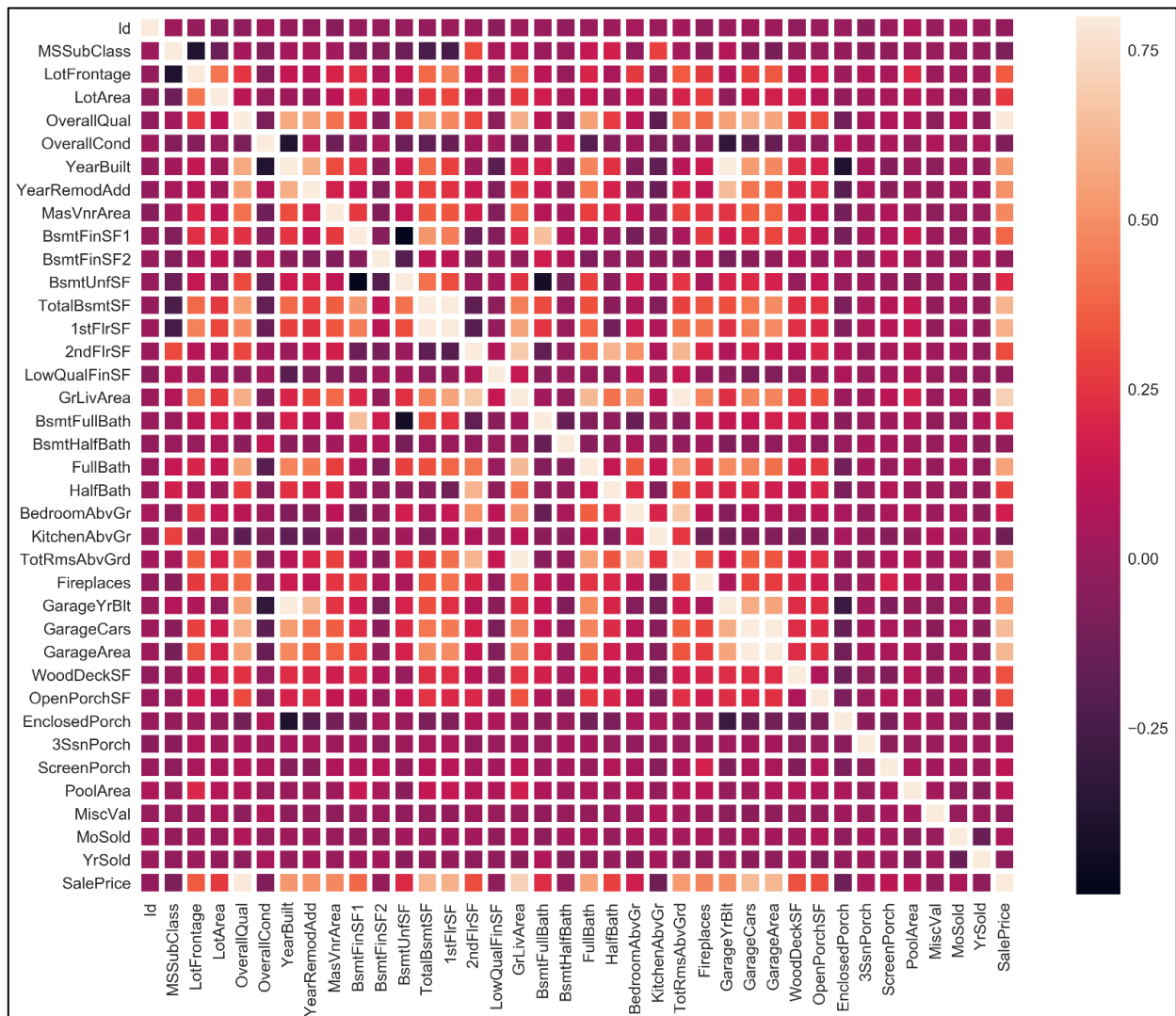




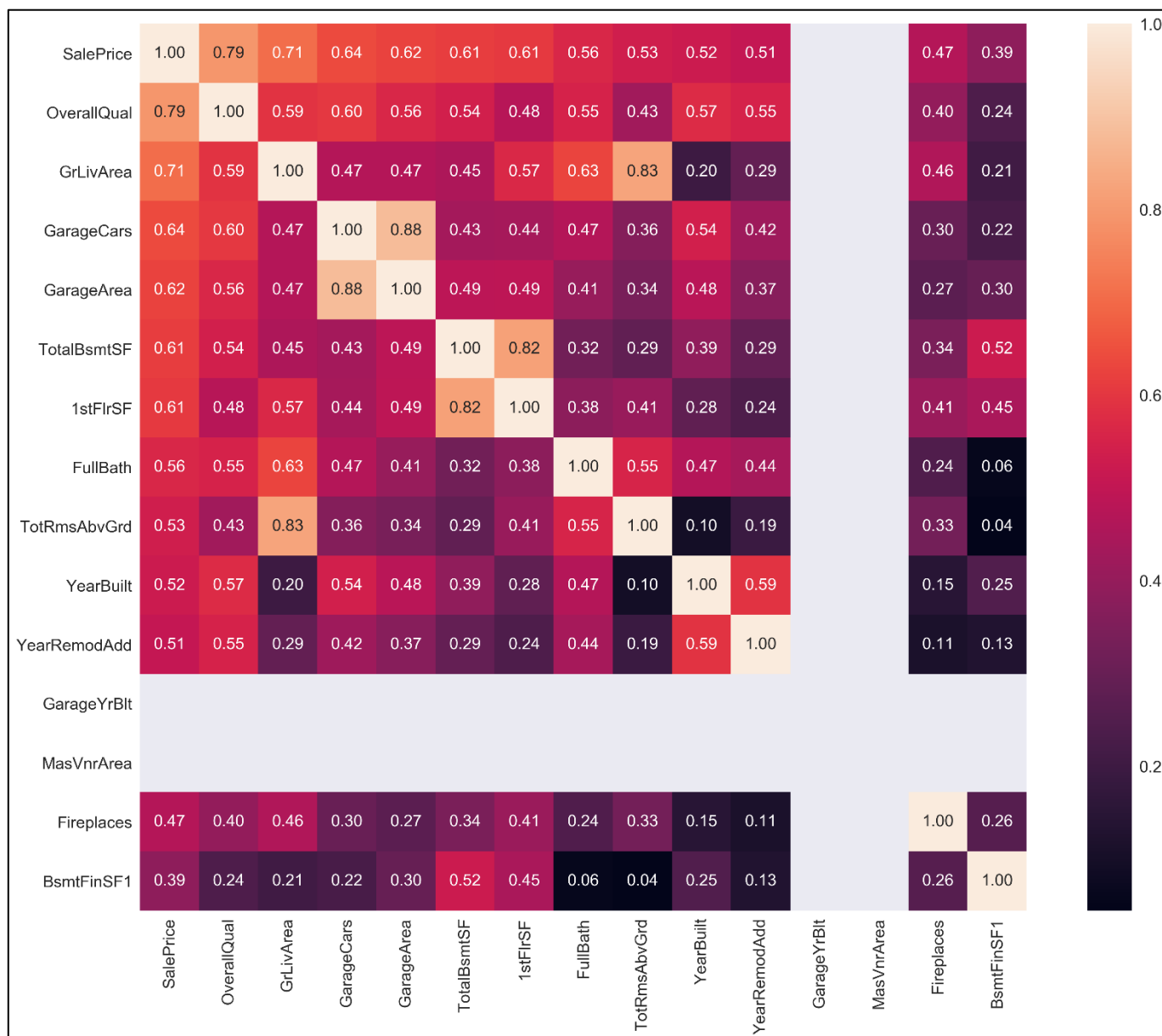
Based upon these plots, I can make the following conclusions:

Its visible that the 1stFlrSF and 2ndFlrSF correlate quite strongly with the GrLivArea. Hence, they might probably be dropped and the GrLivArea feature should be sufficient to provide the necessary information gain provided the correlation score is high enough ( $>0.85$ ). Similarly the scatter plot between the GarageArea and GarageCars features indicates that they share a strong correlation and that one of them could be dropped without reducing the information gain.

Hence, it seems that there might be multiple features with high correlations that could be dropped to reduce the dimensionality of the dataset. A correlation Heatmap and scatter plots between the highest correlated variables will help identify these features.



This heatmap offers quite some insight into the correlations between most of the features. I personally however am interested in the correlation between the SalePrice and the top 15 positively correlated variables. I was tempted to explore negative correlations as well, but decided against it since the number of features aren't too many. So I created a heatmap of the top 15 positively correlated features and the SalePrice feature.



Immediately we can see some features that stand out as important and some which might be dropped. For example, OverallQual, GrLivArea, GarageCars, GarageArea, TotalBsmtSF and 1stFlrSF all have correlation scores higher than 0.6 with the response feature meriting their compulsory inclusion amongst the numerical features. At the same time the GarageCars and GarageAreas features are also highly correlated at 0.88 and so one of them could be dropped without too much information loss.

Similarly, the TotRmsAbvGrd and GrLivArea are also quite highly correlated with a score of 0.83. This makes sense since the General Living area of the house would increase as the number of rooms



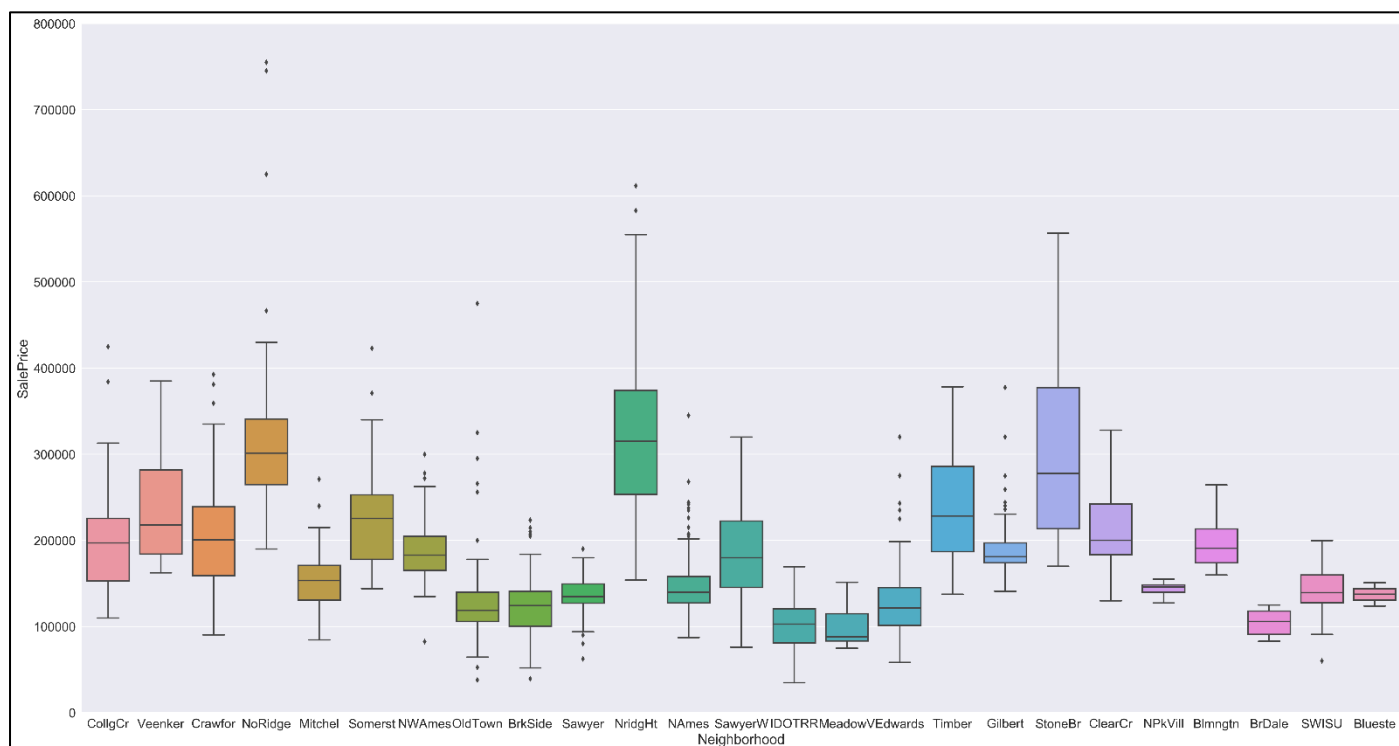
above ground increased. Another interesting pair is that of TotalBsmtSF and 1stFlrSF which have a score of 0.82. This makes sense since the floor plan of a house would be similar on all of the stories as a result of which the floor area of all the levels would be approximately equal and hence the 0.82 correlation score makes sense.

The YearBuilt and YearRemodAdd features aren't as important as I imagined with only scores of 0.52 and 0.51 respectively. I thought that the age of the structure would've had a higher influence on the SalePrice than 0.5 but the data says otherwise.

To summarize, most of the numerical features that made it to the top 15 in the correlation matrix made sense. Now to explore the categorical features. I will select features that based on my intuition would be important while deciding the SalePrice of a house. These are:

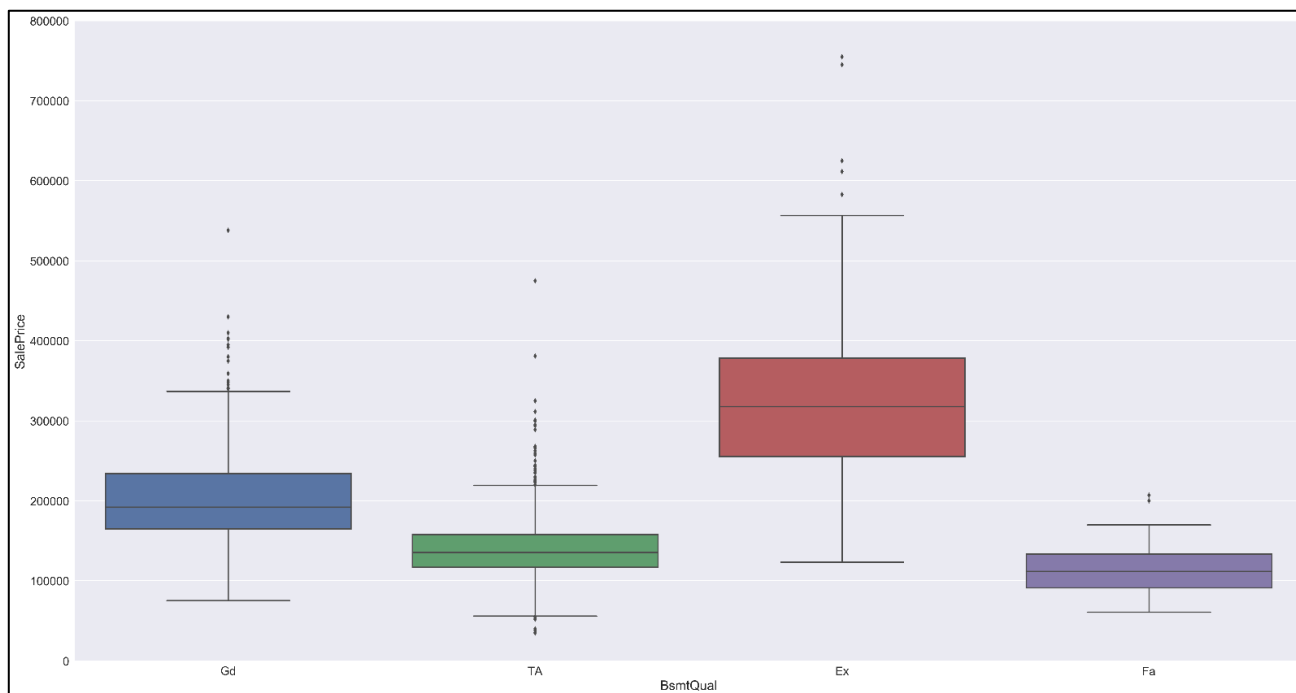
1. Neighborhood
2. FullBath
3. GarageQual
4. GarageCond
5. KitchenQual
6. CentralAir
7. BsmtQual
8. ExterCond

I will use boxplots to visualize these features.

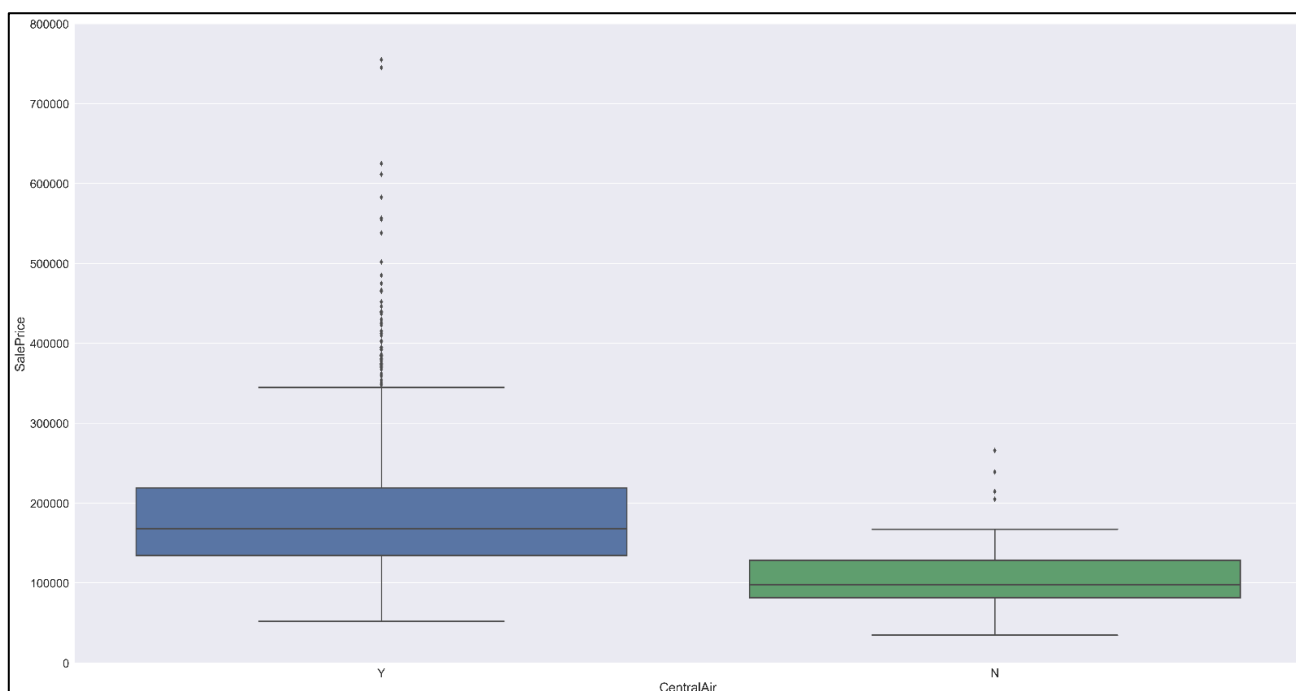


This is a slightly more complicated feature to interpret independently. Most of the neighborhoods seem to have almost similar mean SalePrice values, with some of them are showing some variance over the SalePrice. I expected to see each boxplot cover a small range of SalePrice i.e. having a small standard deviation. This is because most neighborhoods tend to have a certain range within which property rates vary as a result of factors like crime rate, the educational resources, type of residents

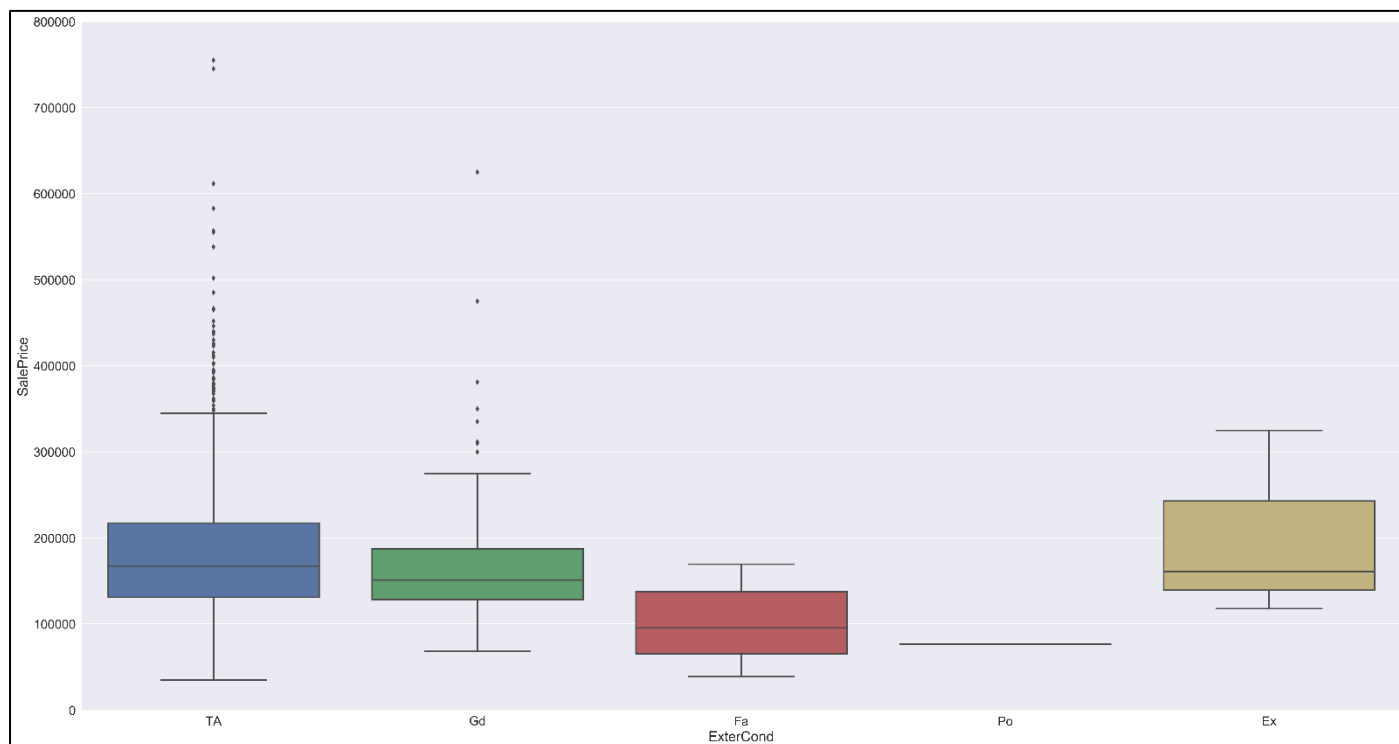
etc. While there are some neighborhoods that conform to this assumption, some neighborhoods like NridgHt, Timber and StoneBr have a very large standard deviation as well as some outliers. It seems funny that one neighborhood could have such a large deviation in property prices from the mean, pointing to the presence of large houses that might be considered as outliers. I would ratify that the neighborhood definitely does affect the SalePrice, in combination with features like OverallQual, GrLivArea, GarageArea and TotalBsmtSF among others, based upon my intuition and domain knowledge but on it's own does not offer a lot of insight.



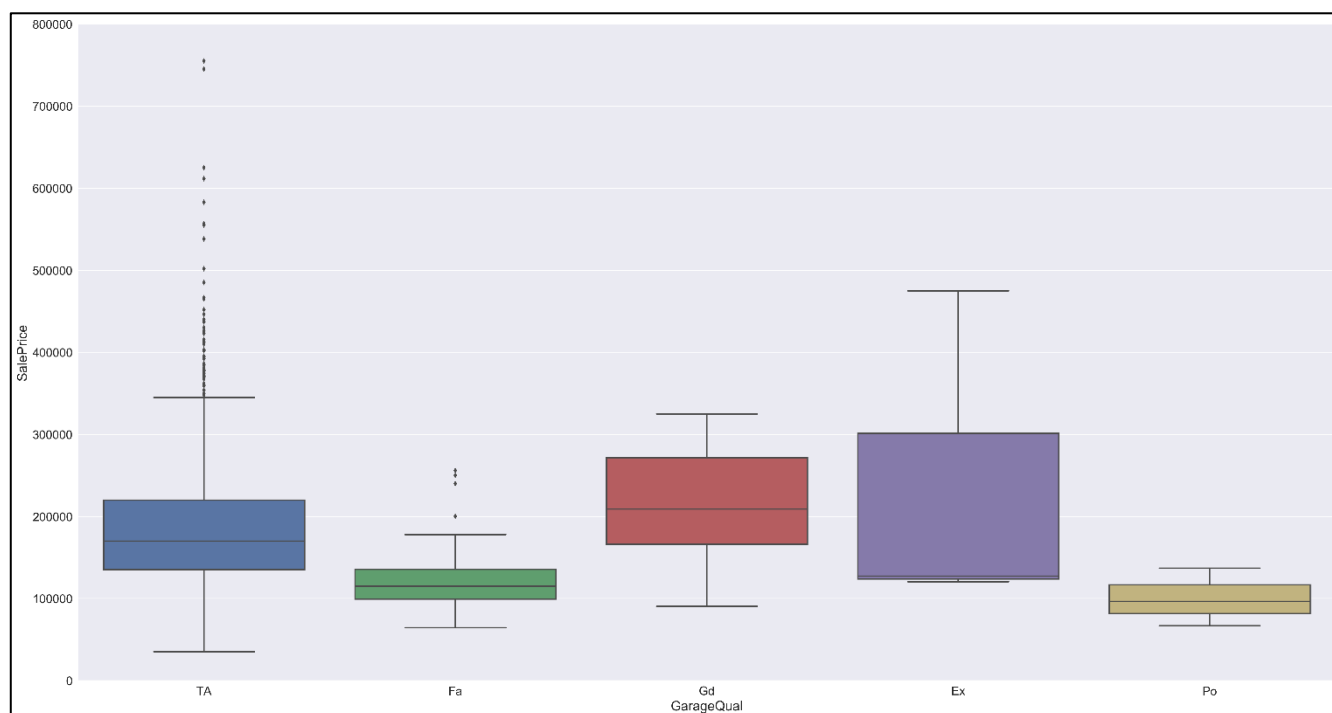
The BsmtQual Fa, TA and Gd levels don't contribute towards driving up the SalePrice, with their mean values between \$125,000 and \$200,000. The Ex level however does have a higher mean of about \$320,000.



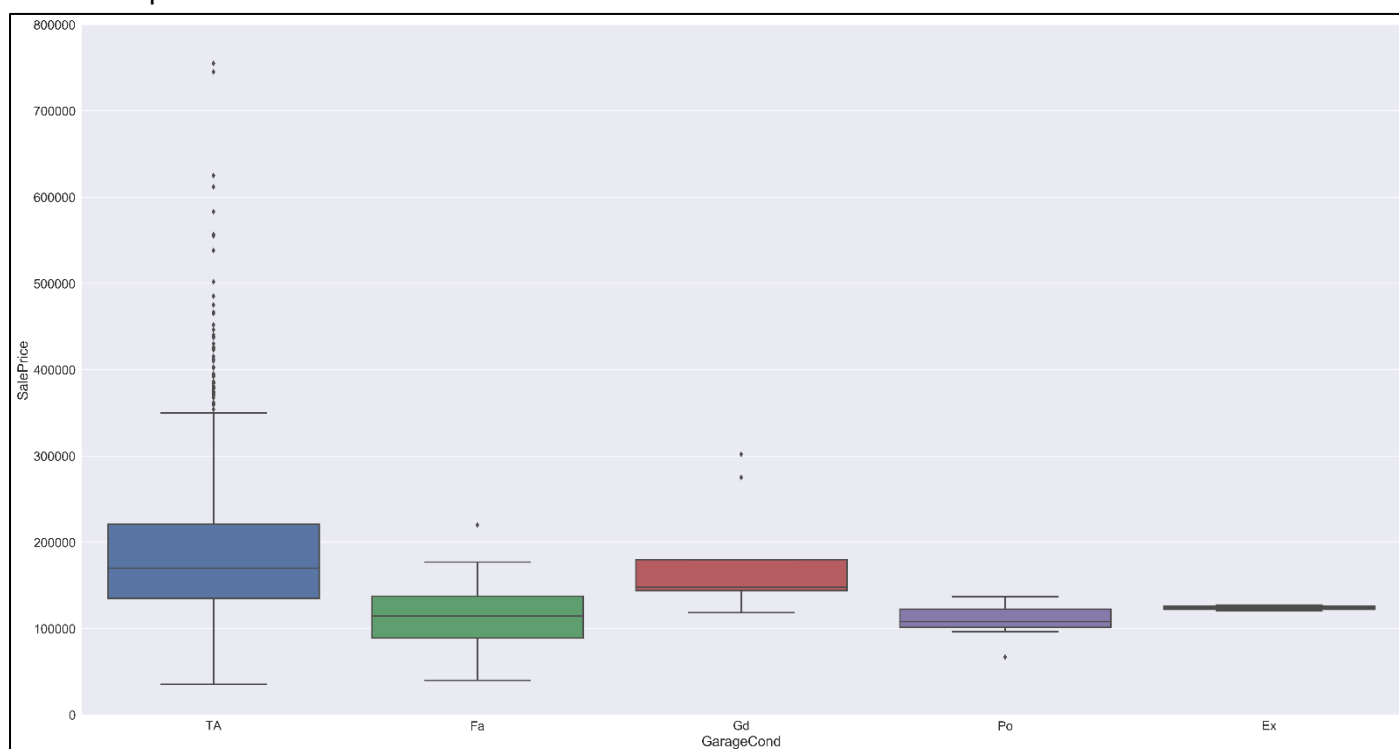
CentralAir surprisingly didn't explain a lot of variance for SalePrice, but the trend does make sense. Houses which have Central Airconditioning do have a higher mean SalePrice with quite some outliers while those that don't, do have a lower mean SalePrice.



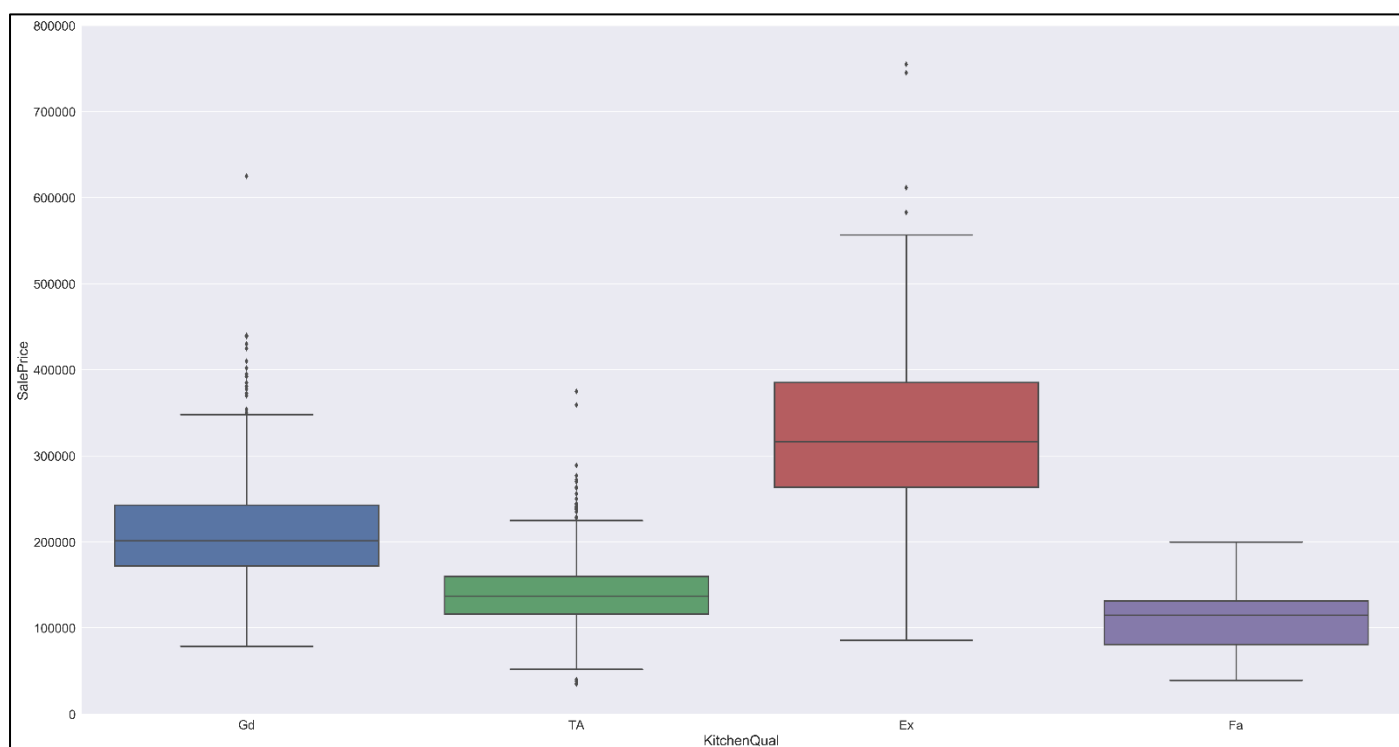
The External condition of the house also has almost the same mean across SalePrices. Something that stands out as odd is that there are quite some outliers which are houses with Average(TA) Exterior Conditions have rather SalePrices while houses with Excellent(Ex) Exterior Conditions have much lower SalePrices. Ofcourse, it's highly likely that the average exterior condition are offset by a large floor area, more number of bathrooms, a large garage etc. which results in a the SalePrice still being quite high.



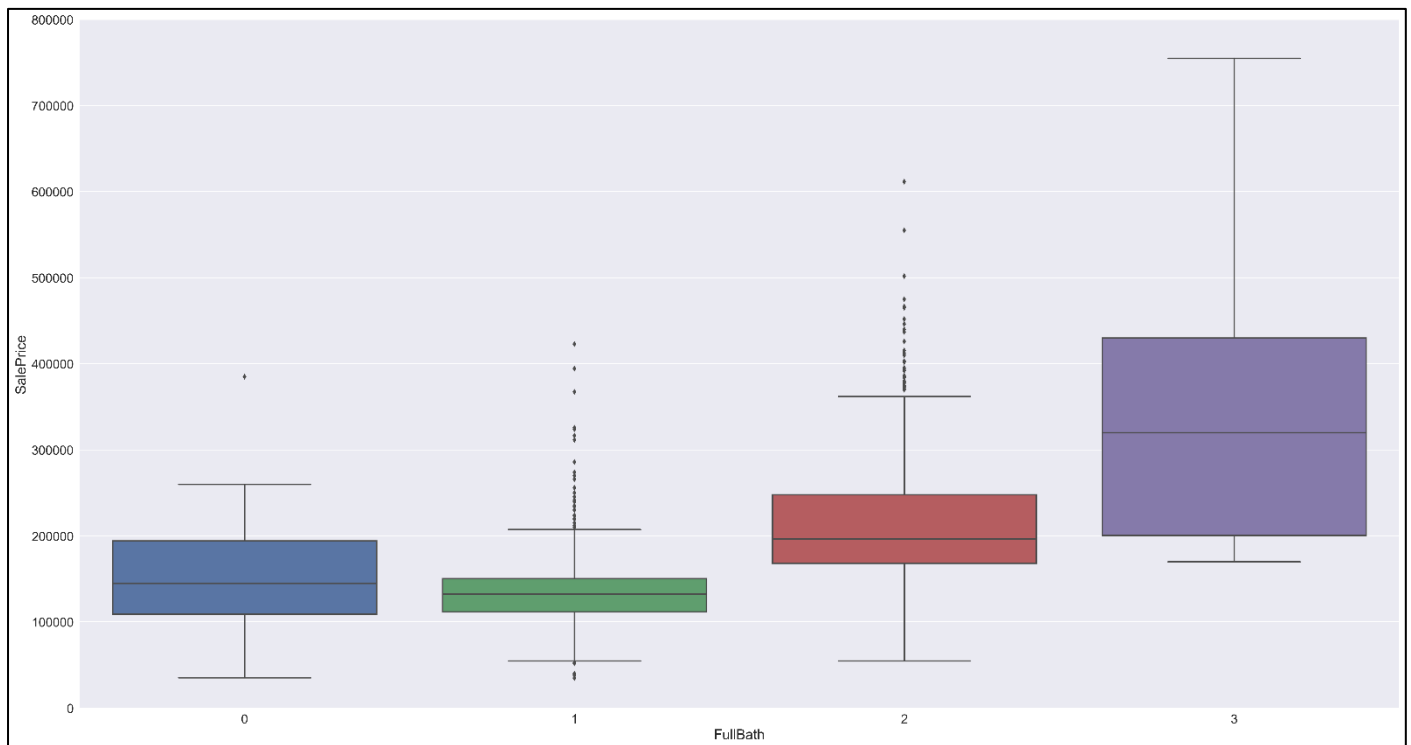
Again the mean SalePrice remains between \$100,000 to \$200,000 for most of the houses while there are still quite some houses with average quality garages that have high SalePrices seen as outliers in the boxplot.



The same situation as the GarageQual feature exists for the GarageCond variable, with similar mean SalePrices for all types of garageQual but outliers for Average(TA) condition garages. I would hazard the same assumption that other features like floor area, number of bathrooms etc offset the GarageCondition in driving up the SalePrice for these points.



There aren't any surprises here as the KitchenQual definitively drives the SalePrice with a stronger relation than the GarageQual and GarageCond. There are some outliers for the Good(Gd) quality kitchens but nothing of too much concern.



This is the last variable I explored. Once again, there isn't too much out of the ordinary here. Houses having zero or one full bathroom sold for almost the same mean SalePrice. In fact, the variance for the zero Fullbath boxplot is larger than that of the one FullBath. Besides this small deviation, the general trend of SalePrice increasing with the number of Full Bathrooms seems well within normal bounds.

This concludes my Variable Exploration using visualizations. In the next section I will list out what Algorithms and Techniques I intend to use.

## Algorithms and Techniques

In terms of technique the main plan of action will be pre-processing the data and getting it ready for use in my training algorithms.

### Pre-Processing:

1. I plan to check for missing values in the dataset and then decide between two strategies of imputing missing values or dropping those features depending upon how important I think they are in predicting the SalePrice.
2. I also plan to convert the categorical features into dummies to facilitate the use of these datasets in algorithms like Decision trees or Boosting models.

## Algorithms:

1. The first Algorithm is my benchmark model which will be a simple Multivariate Linear regression. For this I will select a subset of features, that I believe are important predictors based upon my intuition and reinforced by insights from the visualizations in the previous section of Data Exploration and Visualization. These features would be those that had an almost linear relationship with the dependent variable. The Linear regression is the simplest of all regression models and for a small set of features it will train fast and give me a quick result that would allow me to gauge where exactly I should be in terms of the RMSE score.
2. The next algorithm I decided to explore after the benchmark was a Random Forest (RF). Most of the features in the dataset aren't exactly non-linear and a decision tree would do a pretty job of capturing that non-linearity and handling the categorical variables as well. Unfortunately, a simple decision tree would easily be prone to overfitting and would need to be pruned. I would rather use an ensemble of decision trees and average out their results; and hence I decided to explore the performance of a Random Forest on the data. RFs work by creating N number of 'fully grown' decision trees/estimators that have a Low Bias and High Variance and then averaging their results. These trees/estimators are uncorrelated so that the variance of the model is reduced i.e. it reduces error by minimizing variance. But the bias of the model will be bounded by the bias of the individual trees in the ensemble. As, a result this is a Low Variance and High Bias model. Normally the initial bias of the model is kept low by estimating with large trees that are unpruned. I plan to use a Grid Search Cross Validation approach to find the best estimator for the Random Forest, fit it to the training data and then make predictions on the test dataset.
3. Gradient Boosting was another approach I decided to select for my set of models. Gradient boosting like Random Forests also uses decision trees with one fundamental difference. Random forests work by training multiple decision trees on random samples of data and then averaging their results. However, unlike Random Forests, Boosting works differently by using a bag of 'Weak Learners' i.e. trees with shallow depths or perhaps even simple decision stumps that have high bias and low variance. The aim is to reduce the error by minimizing the bias of the boosted model. Additionally, by aggregating the results from the weak learners, Boosting can reduce variance to some extent as well. Here as well, I plan to use GridSearch Cross Validation to find the best set of parameters that minimize the training data RMSE, fit it to the training data and use it to predict for the test dataset.
4. The last model I decided to try was a Regularized Linear regression, more specifically a Lasso Regularized Linear Regression Model. LASSO (Least Absolute Shrinkage and Selection Operator) is a regression method that penalizes the coefficients of a regression. By penalizing the coefficients, some of the parameter estimates will be driven towards a smaller value. Reducing the coefficients that are rather large helps to prevent them from having a dominating effect on the dependent variable. The larger the coefficient value, the larger will be the penalty applied. An added advantage of Lasso regression is that some coefficients are reduced to zero or negligible values, which results in a sparse coefficient matrix. This is

convenient since it automatically performs feature/variable selection which is convenient in this case where there are a number of highly correlated predictors.

5. As a final approach, I plan to combine/stack predictions from multiple models in order to see if that helps improve the accuracy of the prediction. This may or may not make a difference, but I still plan to explore it.

## Benchmark

The benchmark for this model is a Simple Multivariate Linear Regression fitted onto a small subset of features chosen by intuition. I chose this as a benchmark simply because it would be quick and simple to fit allowing me to get a fast judgement of what range of RMSE values I would be working on as well as to set an Upper Bound on the problem. The Linear Regression is the simplest of regression algorithms that uses the method of Ordinary Least Squares to fit a 'Line of Best Fit' to the data that minimizes a metric like Root Mean Square Error and maximizes the  $R^2$  (Goodness of fit). Hence, it essentially provides an equation of a line with coefficients for each input feature and a constant term.

I used SKLearn's `LinearRegression()` class to run the model on the following subset of features chosen by me using my judgement of what aspects of a house would drive its SalePrice:

1. GrLivArea – The General Liveable Area of the House in square feet.
2. OverallQual – The overall Quality of the house on a scale of 1=Very Poor to 10=Very Excellent.
3. GarageCars – The number of cars that the Garage can hold.
4. TotalBsmtSF – The total floor area of the Basement in square feet.
5. 1stFlrSF – The total floor area of the 1<sup>st</sup> floor in square feet.
6. FullBath – The number of full bathrooms in the house.

These features, in my opinion, would exhibit a linear relationship with the SalePrice. A minor notification is that I did my data pre-processing before running the benchmark model due to the presence of missing values in the above chosen features. As a result, I decided to first pre-process the entire dataset and then run each model sequentially.

## III. Methodology

---

### Data Preprocessing

The first step of data pre-processing was to combine both training and testing datasets run a script to check for missing values in the combined dataset. I wrote a script that would extract the sum of missing values for each feature and print that along with the percentage of missing values.

	Total	Percent
PoolQC	2909	0.996574
MiscFeature	2814	0.964029
Alley	2721	0.932169
Fence	2348	0.804385
FireplaceQu	1420	0.486468
GarageCond	159	0.054471
GarageFinish	159	0.054471
GarageQual	159	0.054471
GarageYrBlt	159	0.054471
GarageType	157	0.053786
BsmtCond	82	0.028092
BsmtExposure	82	0.028092
BsmtQual	81	0.027749
BsmtFinType2	80	0.027407
BsmtFinType1	79	0.027064
MasVnrType	24	0.008222
MasVnrArea	23	0.007879
MSZoning	4	0.001370
BsmtHalfBath	2	0.000685
BsmtFullBath	2	0.000685
Functional	2	0.000685
Utilities	2	0.000685
KitchenQual	1	0.000343
GarageArea	1	0.000343
TotalBsmtSF	1	0.000343
Electrical	1	0.000343
BsmtFinSF2	1	0.000343
Exterior2nd	1	0.000343
GarageCars	1	0.000343
SaleType	1	0.000343
BsmtUnfSF	1	0.000343
BsmtFinSF1	1	0.000343

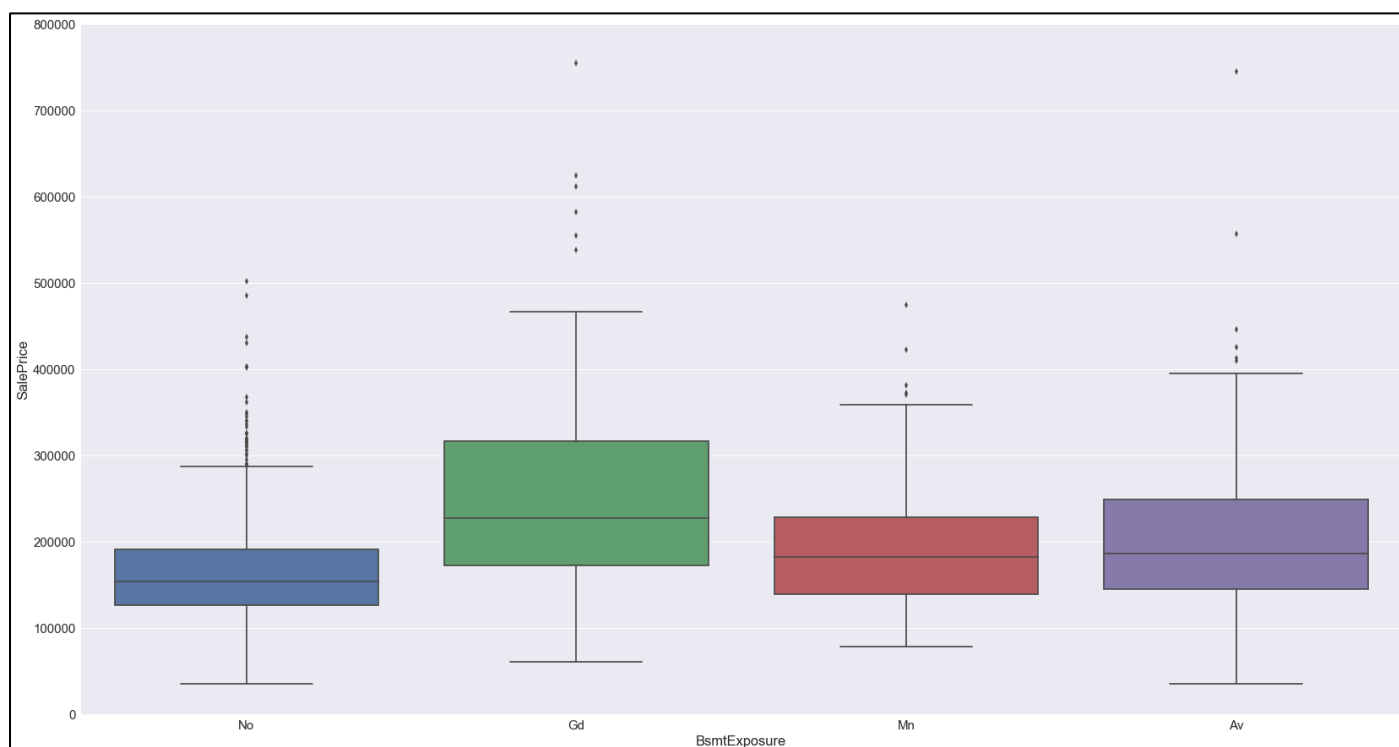


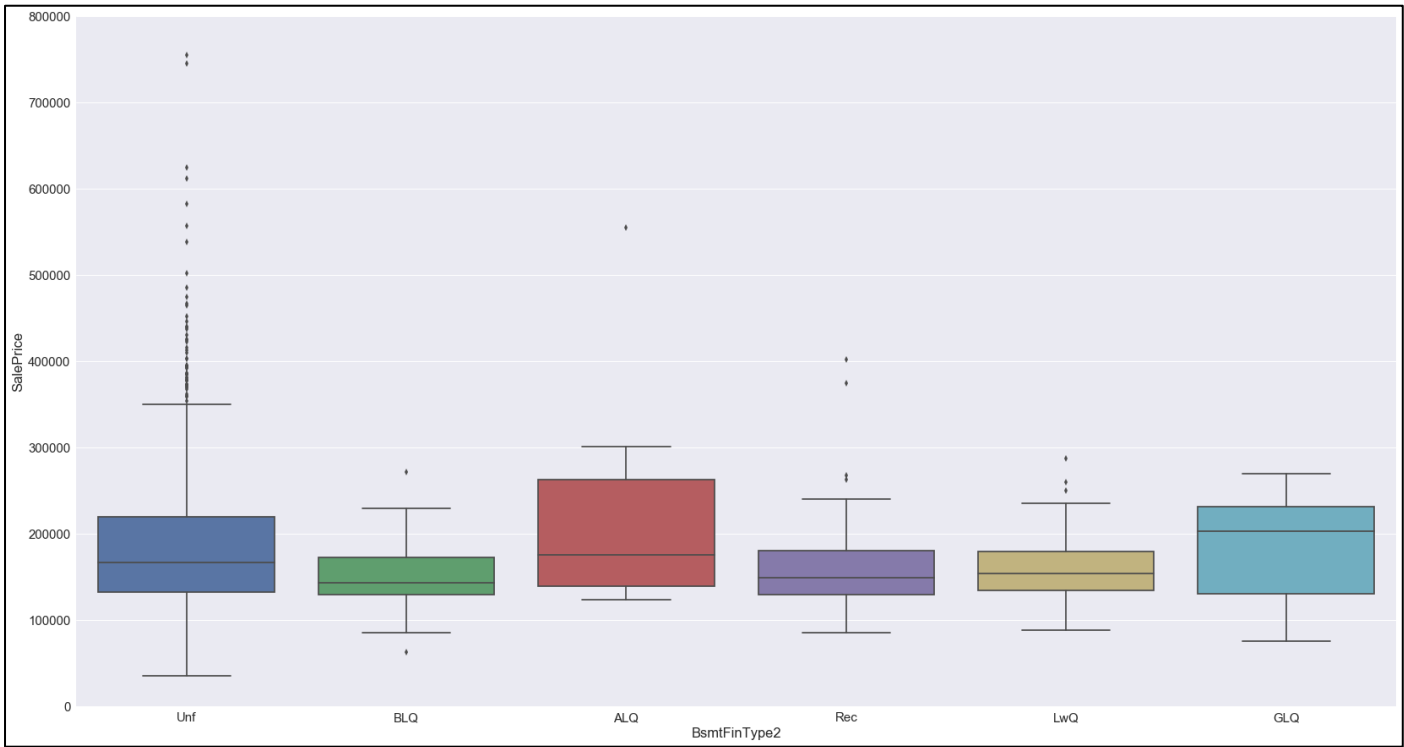
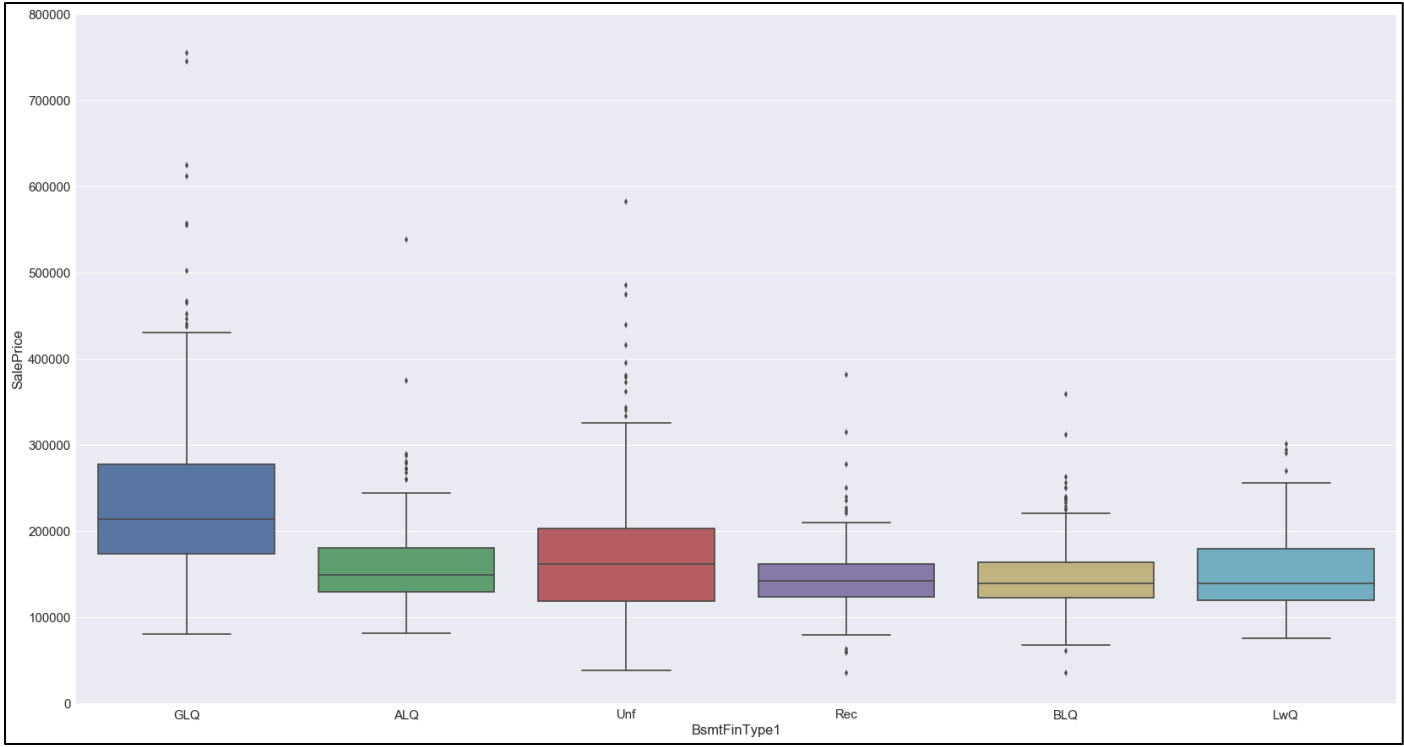
	Total	Percent
Exterior1st	1	0.000343

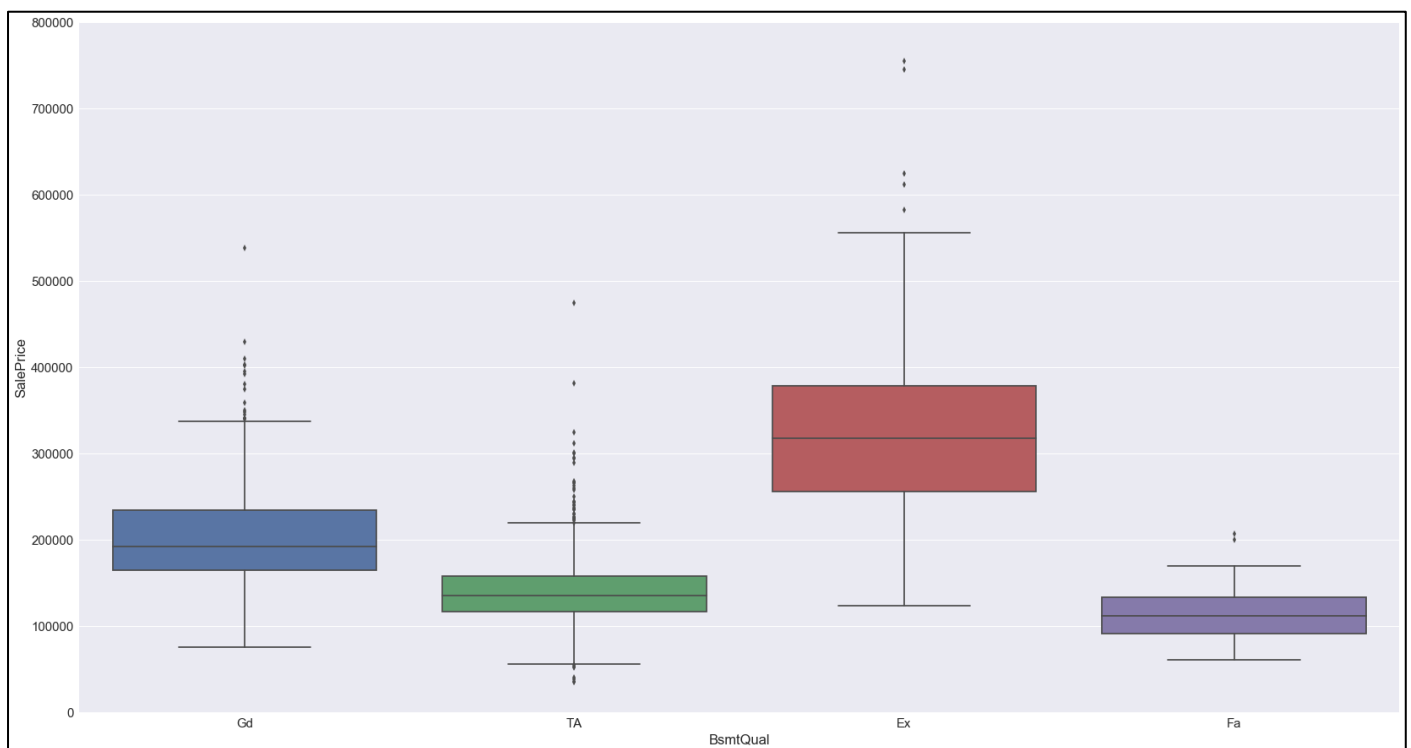
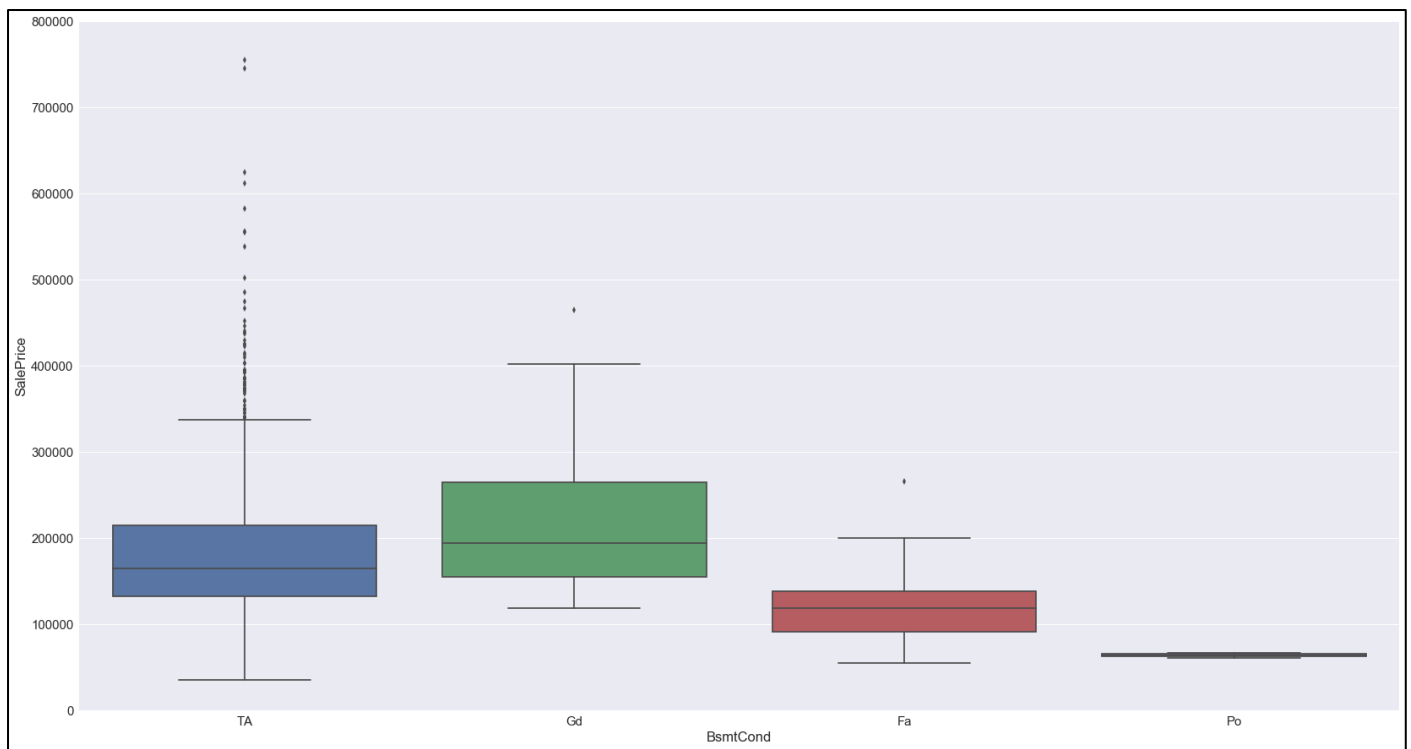
At first glance, it's easy to see that some of the features have more than 90% of their values missing. I have decided to arbitrarily select 20% missing as a threshold for missing values and any feature that has more than 20% of missing features will be dropped. This still leaves quite a few features to impute but I decided on 20% since I do not want to lose too much of information. The rest of the features with fewer missing values can be accounted for by checking the importance the feature and dropping it or simply deleting the rows with missing variables. It is possible that these are not missing features but instead indicate an absence of these features for the respective houses. In that case, since more than 90% of these houses do not have these features, the ones that do would more likely act as outliers for the rest of the data. Hence, dropping them would make more sense.

The PoolQC, MiscFeature, Alley, Fence and FirePlaceQU features are automatically dropped since they all have more than 20% of missing values. As far as the rest are concerned I will explore them a bit more before deciding how to deal with them.

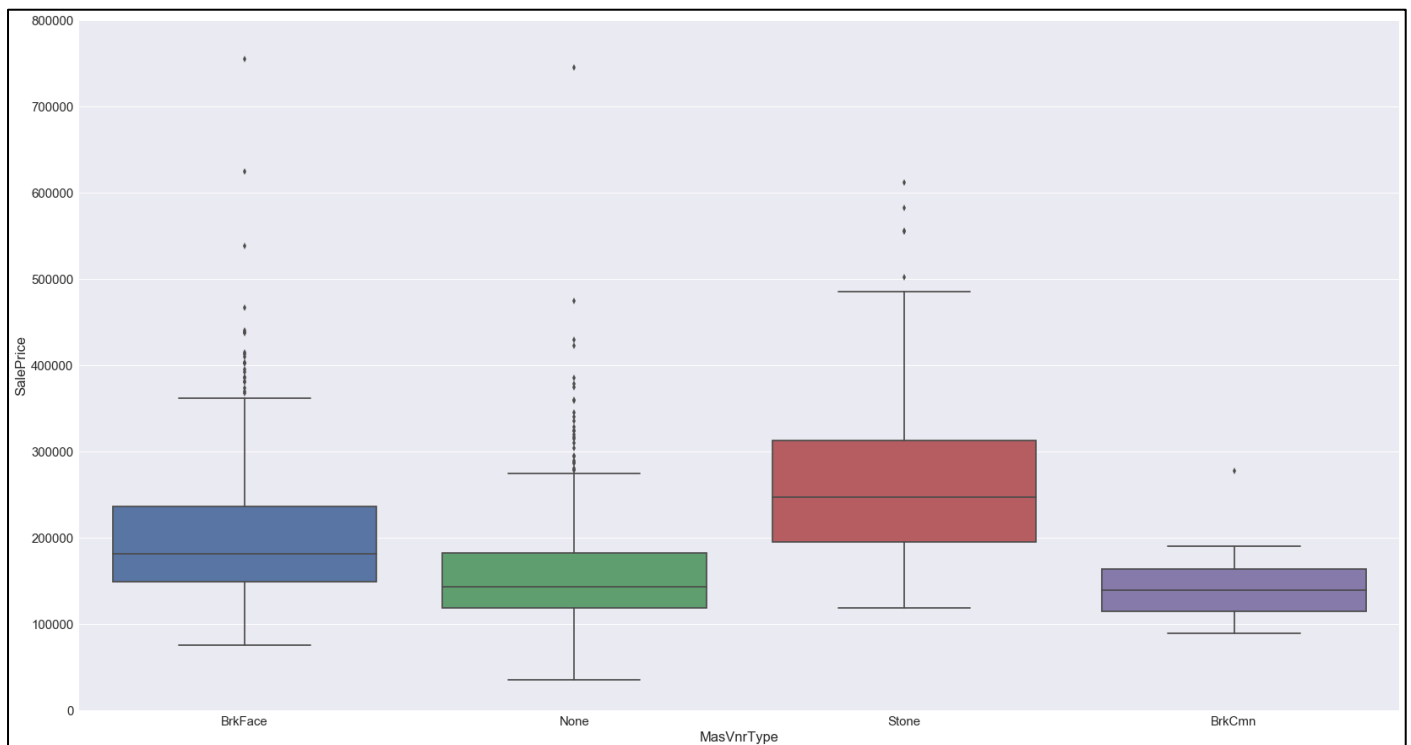
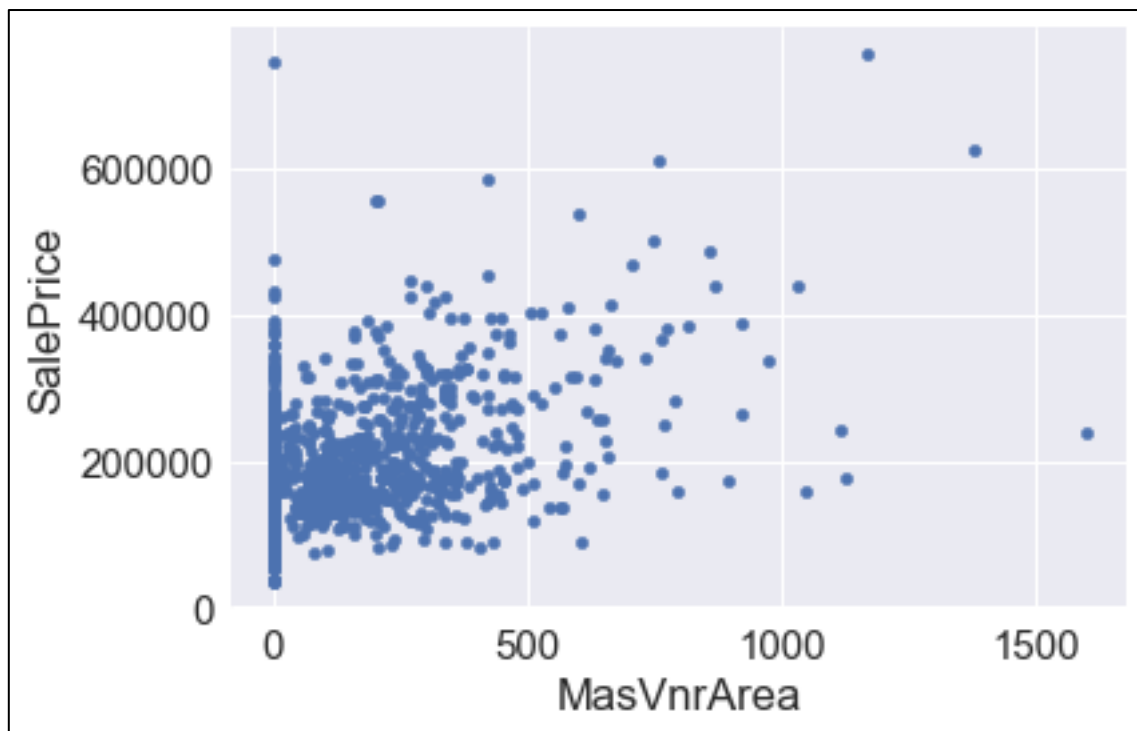
The Garage features involving the finish quality, Year built etc. don't seem to have too much of an influence on the SalePrice as has been seen earlier while exploring the categorical variables. The GarageCars feature seemed like the only feature that really mattered. So, I will drop all the other Garage features.







It is clear that none of these Basement features explain a lot of variance for the SalePrice. So I can drop them as well.



From the above plots, it is sufficient to say that the Mason Area feature shares a weak relationship with SalePrice while the Mason type feature again explains little or none of the SalePrice variance. Hence, they both can be dropped.

The last missing value feature is Electrical. Since this is just one missing row, I will simply drop the row from the dataset.

**NOTE: I am removing features for the combined train and test datasets since I am working on the assumption that both test and train datasets should be drawn from the same sample.**

**Hence, removal of features should be done on both the train and test data. Further I am not involving the target variable for the removal in any manner so as to avoid any chance of overfitting. After dropping the columns, I will split the datasets into the original train and test sets again and drop rows with single missing values from the train set only.**

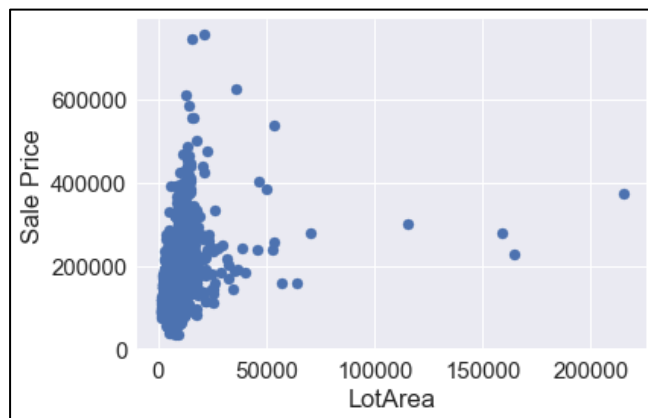
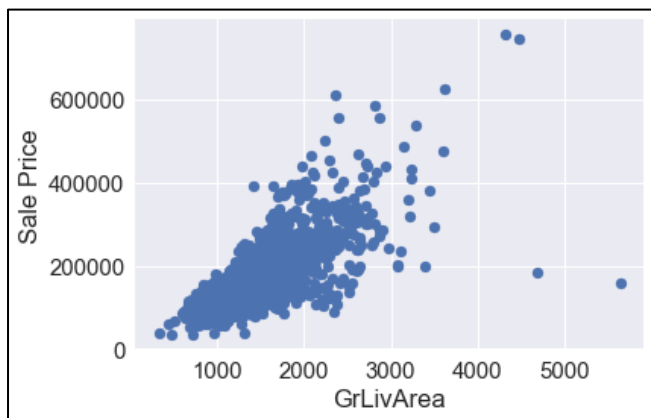
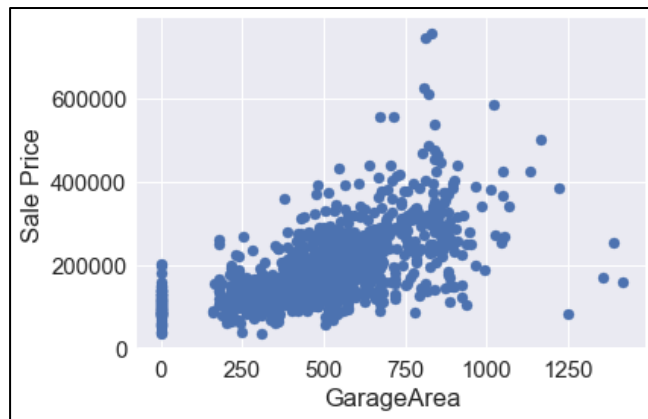
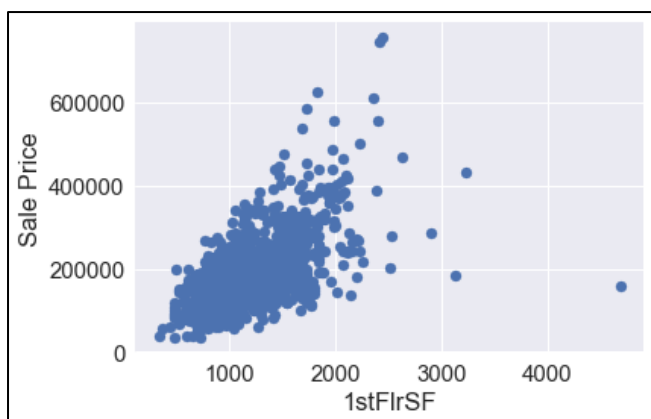
After all these steps, the missing data table looked like this:

	Total	Percent
MSZoning	4	0.001370
Functional	2	0.000685
Utilities	2	0.000685
BsmtHalfBath	2	0.000685
BsmtFullBath	2	0.000685
Electrical	1	0.000343
Exterior2nd	1	0.000343
GarageArea	1	0.000343
BsmtUnfSF	1	0.000343
BsmtFinSF2	1	0.000343
SaleType	1	0.000343
BsmtFinSF1	1	0.000343
GarageCars	1	0.000343
KitchenQual	1	0.000343
TotalBsmtSF	1	0.000343
Exterior1st	1	0.000343

Some of these missing values were from the training set while others were from the test set. I decided to split the combined data back into train and test sets so that any rows that I dropped in the training dataset would enable me to remove the corresponding SalePrice rows as well.

For the remaining missing values, since they account for less than 0.5% of the total rows, I decided to impute these NA's using either the mean or median values of the column depending on whether that feature is numeric or categorical.

The next step was to check for outliers and deal with them or keep them. Scatter plots of some important features were used for this:



From these plots, it's visible that there are some houses that act as outliers. For e.g. houses having quite large Living Areas or Lot Areas seem to still have abnormally low Sale Prices. This is the case for about 3 to 4 cases. I initially thought about dropping them, but then decided against it since I assumed that these were indicative of anomalous cases that might help generalize to similar cases in the test set and perhaps reduce the error. This was not a guarantee, but I decided to proceed with leaving the outliers untouched.

The last stage was to finalize the datasets for my models. For the Benchmark model, I created a vector of important features, chosen by me intuitively, and subsetting the training and testing datasets with these features. The log transformed SalePrice variable would serve as the response (dependent variable).

For the other model, I used dummy encoding to break up each categorical variable in the combined dataset into dummy binary variables and then proceeded to split them into train and test sets. The SalePrice feature was log transformed as earlier. This wrapped up all of the data preprocessing, and the next step was to start fitting models to the datasets.

## Implementation

This section will cover the different algorithms and techniques I used post the data preprocessing phase. In terms of changing the original data, I created a specific subset of features to train my benchmark model on and log transformed the skewed response feature.

For the rest of the models I simply used dummy encoding on the categorical features and log transformed the response feature.

**a. BENCHMARK Model:**

The Benchmark model used is a Simple Multivariate Linear Regression, trained on a chosen subset of features, and then used to make predictions on the test dataset. The `sklearn.linear_model` was imported and the `LinearRegression()` class was trained. The model was fed with the following input features that I chose using intuition and domain knowledge. Input features:

1. GrLivArea – General Liveable area (sq. feet)
2. OverallQual – Overall Quality of the house on a 1-10 scale
3. GarageCars – The number cars that the Garage can hold
4. TotalBsmtSF - The total square footage area of the basement
5. 1stFlrSF - The square footage area of the 1<sup>st</sup> floor
6. FullBath – The number of Full Bathrooms in the house















The final model output was as follows:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

	Features	Model Coefficients
0	GrLivArea	0.000173
1	OverallQual	0.125567
2	GarageCars	0.113717
3	TotalBsmtSF	0.000124
4	1stFlrSF	0.000038
5	FullBath	0.039277

This table shows the regression coefficients assigned to each of the features. I also created a plot of the line of least squares to visualize the model's goodness of fit.

I used this model to run predictions for the test set and then made the submission to the Kaggle competition submissions webpage. This benchmark achieved an RMSLE of 0.17649 and ranked me at 1800 out of 2307 teams as of 2017-11-13 18: 30. The scoring screenshot has been shown below

Overview	Data	Kernels	Discussion	Leaderboard	Rules	Team	My Submissions	Submit Predictions
1795	▼ 205	R_Mac						0.17575 14 1mo
1796	▼ 205	vimjian						0.17611 2 2mo
1797	new	Hairui Liu						0.17631 2 6d
1798	▼ 206	Praxis.Invincible						0.17637 2 12d
1799	new	Aegis_Michelle_Rebello						0.17646 3 6d
1800	new	UdaPaul						0.17649 1 ~10s
<b>Your Best Entry ↑</b> Your submission scored 0.17649, which is not an improvement of your best score. Keep trying!								
1801	▼ 208	Junwei Dong						0.17662 2 2mo
1802	▼ 208	Jyothsna Dronamraju						0.17702 1 11d
1803	new	limingwu						0.17712 2 1d
1804	new	Aegis_Snehal Patel						0.17716 1 6d
1805	new	Julio Pariona						0.17743 1 3d
1806	new	Allen Prasad						0.17762 1 3d
1807	▼ 210	TylerTuschhoff						0.17763 2 2mo

## b. Random Forest with GridSearch Cross Validation:

My first serious model that would be trained on the entire dataset was the Random Forest (RF). RFs work by creating N number of 'fully grown' decision trees/estimators that have a Low Bias and High Variance and then averaging their results. These trees/estimators are uncorrelated so that the variance of the model is reduced i.e. it reduces error by minimizing variance. But the bias of the model will be bounded by the bias of the individual trees in the ensemble. As, a result this is a Low Variance and High Bias model. Normally the initial bias of the model is kept low by estimating with large trees that are unpruned.

I ended up making two submissions with the RandomForest model since in my first attempt the RMSLE was higher than my benchmark.

### RF Attempt 1:

The first model I trained was obtained by tuning the number of estimator's hyper parameter via GridSearch Cross Validation. The best estimator is shown below:



```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=3,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=3,
                      min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=1,
                      oob_score=False, random_state=None, verbose=0, warm_start=False)
```

This model was trained on the training data and then used to predict on the test data. The resulting submission was uploaded to the Kaggle webpage and an RMSLE of about 0.2 was returned which was definitely worse than my benchmark. This indicated that the tree was probably overfitting due to cross validating it on the training data itself so I decided to explore some of its hyper parameters. I ended up tuning this model and have documented it in the next **Refinement** section.

### c. Gradient Boosting with GridSearch Cross Validation:

The next approach was using the XGBoost model. Unlike Random Forests, Boosting works differently by using a bag of 'Weak Learners' i.e. trees with shallow depths or perhaps even simple decision stumps that have high bias and low variance. The aim is to reduce the error by minimizing the bias of the boosted model. Additionally, by aggregating the results from the weak learners, Boosting is able to reduce variance to some extent as well.















I used the xgboost library and imported the XGBoostRegressor() class for this purpose. Once again, I used GridSearch Cross Validation on the training dataset to tune the following set of parameters:

- a. `n_estimators` = The number of decision trees trained in the ensemble
- b. `max_depth` = The maximum depth of each of the trees (weak learners)
- c. `learning_rate` = The step size while performing Gradient Descent.

A parameter grid for these hyperparameters was created and fed into the GridSearchCV() class to obtain the best estimator that minimizes the mean squared error metric. This estimator is shown below:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bytrees=1, gamma=0, learning_rate=0.2, max_delta_step=0,
             max_depth=2, min_child_weight=1, missing=None, n_estimators=500,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=True, subsample=1)
```

This model scored a training set RMSLE of 0.1273. This model was then fitted to the training set and then used to predict the SalePrice for the test dataset. This result was uploaded to the Kaggle webpage and the result is shown below:

Overview	Data	Kernels	Discussion	Leaderboard	Rules	Team	My Submissions	Submit Predictions
1311	▼ 144	SabihHaque					 0.14012	11mo
1312	▼ 144	The Son of Flynn					 0.14013	21mo
1313	▼ 144	Pharadorn Kawayapanik					 0.14013	66d
1314	▼ 122	vladconst					 0.14016	216d
1315	▼ 145	torresjeff11					 0.14018	115d
1316	▼ 145	Mauricio Prada					 0.14018	18d
1317	▼ 145	MikeChen					 0.14021	11mo
1318	new	UdaPaul					 0.14029	6now
<div><div>Your Best Entry ↑</div><div>You advanced 84 places on the leaderboard!</div><div>Your submission scored 0.14029, which is an improvement of your previous score of 0.14544. Great job!</div><div> Tweet this!</div></div>								
1319	▼ 146	Jordan Harrop					 0.14043	110d
1320	▼ 146	Zqun					 0.14055	31mo
1321	▼ 146	Paulski					 0.14057	12mo
1322	new	ArunaKumaraswamy					 0.14061	42d
1323	▼ 147	Mihail Ivanov					 0.14073	118d

This XGB model was Cross Validated using k-fold Cross Validation and the best parameters were estimated. The best model was fitted to the training data and then used to predict on the test dataset. The RMSE for this model was 0.14029, a 25.8% increase over our benchmark score of 0.17649. In terms of relative performance, over the 1800 rank achieved by the benchmark, the XGBoost model has achieved the highest rank so far of 1318 as of 2017-11-14 15:57.

#### d. Lasso Regression with GridSearch Cross Validation:

The `sklearn.Lasso()` class will be used for this. The Lasso approach will use L1 regularization to penalize larger coefficients and safeguard them from overshadowing the other features in the linear model.

This Lasso model was Cross Validated using k-fold Cross Validation and the best parameters were estimated. The best model was fitted to the training data and then used to predict on the test dataset. The RMSE for this model was 0.12671, approximately a 40% increase over our benchmark score of 0.17649. In terms, of relative performance, over the 1800 rank achieved by the benchmark, the XGBoost model has achieved the highest rank so far of 902, up from XGB's 1318, as of 2017-11-14 16:45.

## Refinement

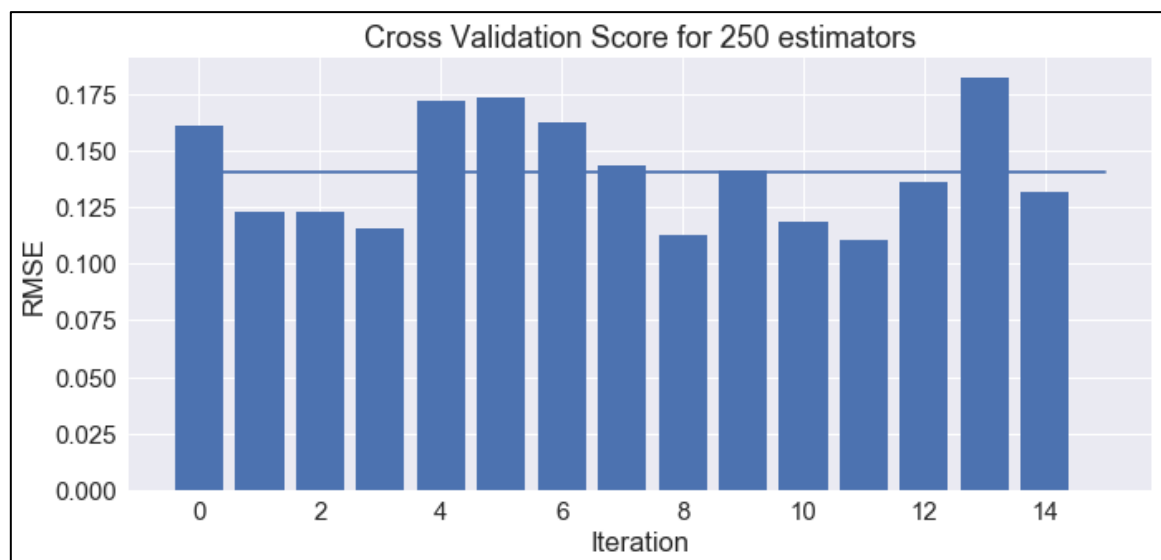
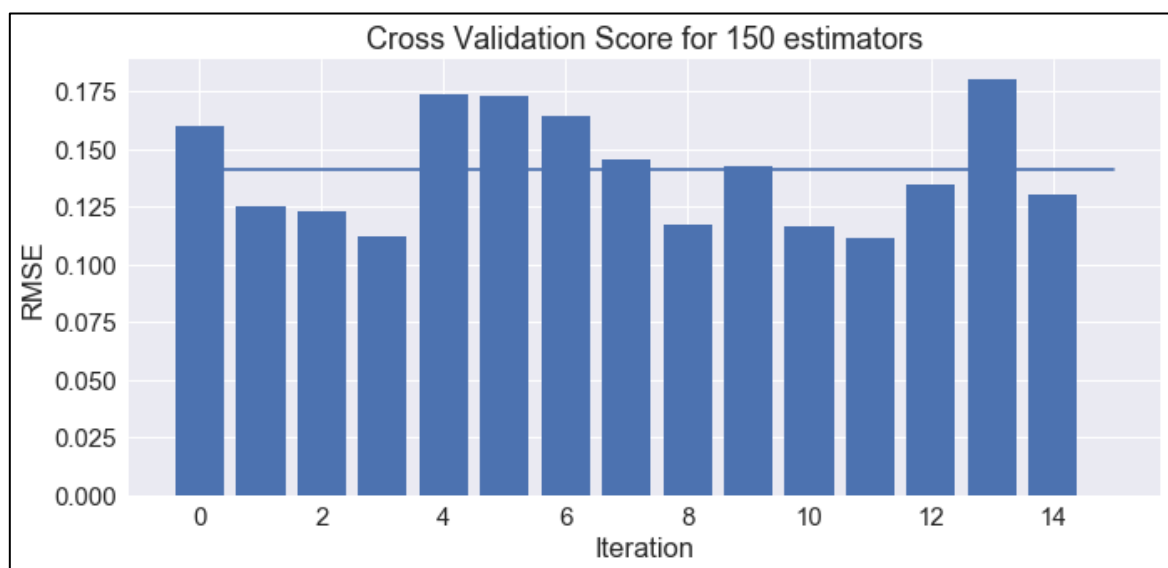
In this section I will there are two phases of refinement. The first is refining the Random Forest model which performed worse off than the Benchmark due to possible overfitting. The second involves an amateur attempt to stack model predictions by combining weighted averages of individual predictions.

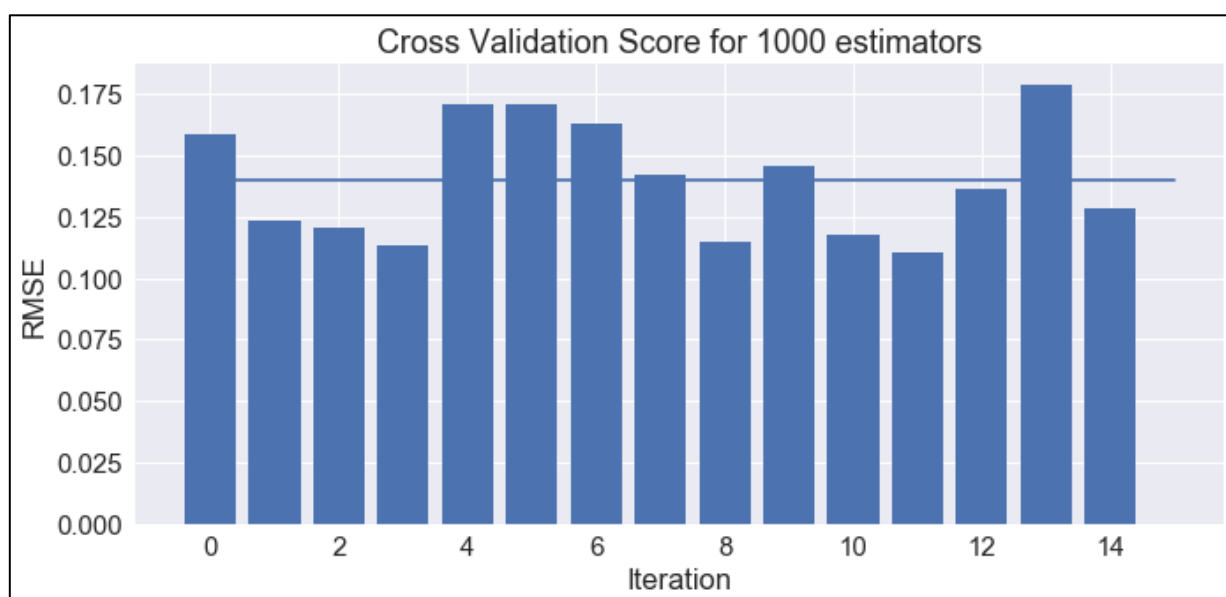
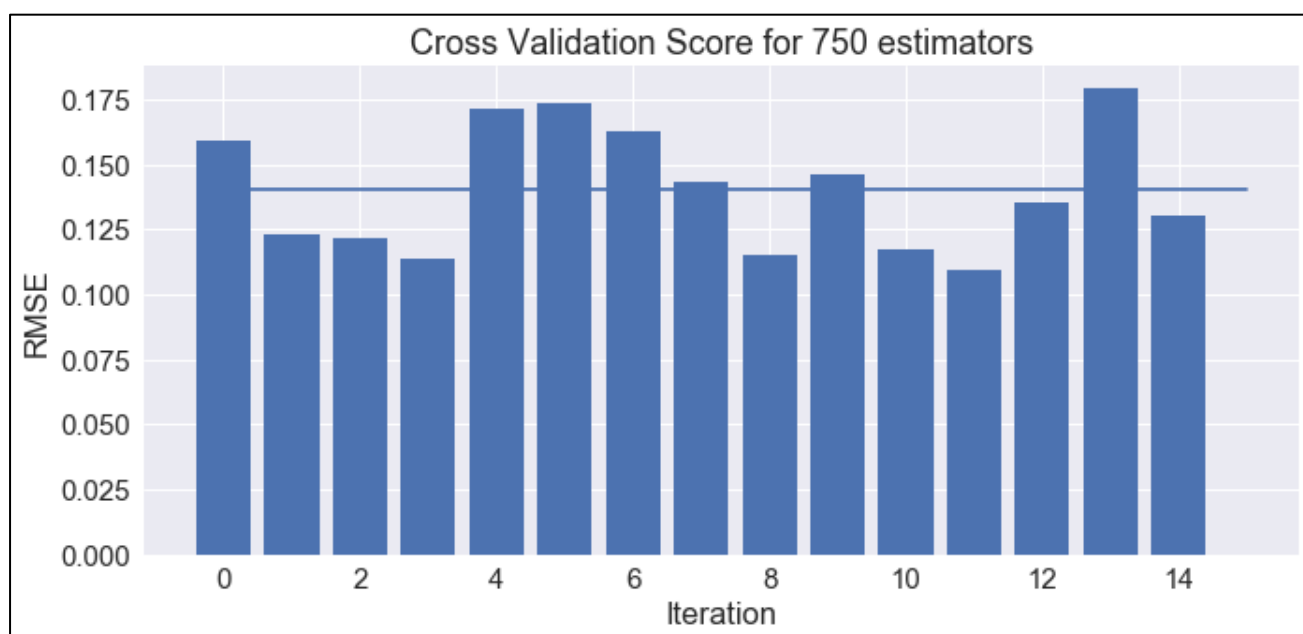
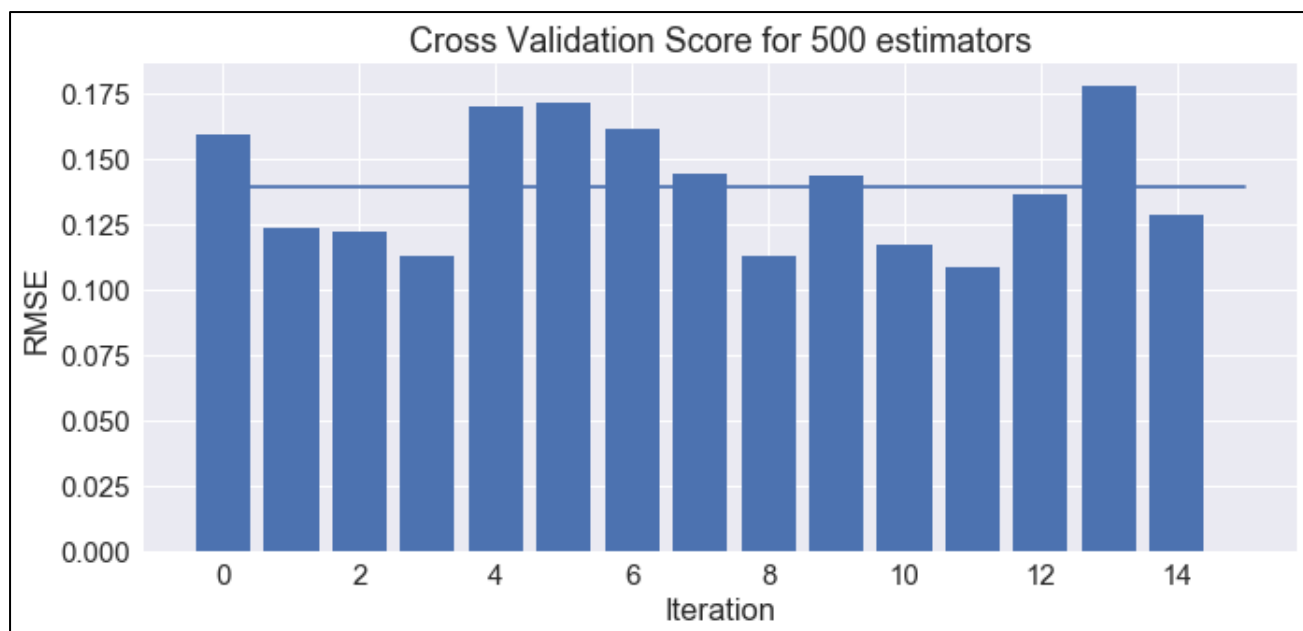
## Refinement 1: Random Forest

The Random Forest model performed badly in the first attempt in spite of cross validating parameters against the training set. Overfitting seems like a likely suspect. I decided to explore the model parameters to identify the problem. Two features that caught my eye are:

1. Max\_depth: this parameter defines the maximum depth of each estimator, the value for this model being 3. Conventionally, the more the depth of a tree, the more it will tend to overfit. This would explain why the RandomForest performed well on the training data but poorly on the test data. I decided to remove this and set it to 'None' so that there wasn't a bound on this parameter.
2. Min\_samples\_split: This is the minimum number of samples required at a node in order to split it. This was initially set to 3 by cross validation so I decided to reduce it to 2 and check if that reduced the error of the model.

Another feature that I wanted to check was if the number of estimators i.e. n\_estimators was sufficient or not. Ideally, the more the number of trees, the better the mode would perform. So, I created a series of plots of the Cross-Validation score for n\_estimators for values ranging from 150 to 1000.





Based on these plots the average Cross Validation score for a basic Random Forest model would be about 0.14 across the number of estimators ranging from 150 to 1000. I decided to stick with `n_estimators = 500` and `min_sample_split = 2`.

The second attempt on a Random Forest model was done using the following estimator:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                        max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=1,
                        oob_score=False, random_state=None, verbose=0, warm_start=False)
```

This model had a RMSLE cross validated score of 0.142 on the training data which looked promising. I used this model to make predictions on the test data and uploaded my submission to the Kaggle Webpage.

Overview	Data	Kernels	Discussion	Leaderboard	Rules	Team	My Submissions	Submit Predictions
1389	▼ 147	Thibaut D			0.14524	1	1mo	
1390	▼ 147	WaRa-T Nim			0.14534	4	2mo	
1391	▼ 147	Xin Zhou			0.14535	2	19d	
1392	▼ 147	Ankitdev			0.14539	1	2mo	
1393	new	Zenan Wang			0.14539	1	3d	
1394	▼ 148	djinnyv			0.14540	4	3d	
1395	new	UdaPaul			0.14544	3	1h	
<b>Your Best Entry</b>								
You advanced 408 places on the leaderboard!								
Your submission scored 0.14544, which is an improvement of your previous score of 0.17649. Great job!								
<a href="#">Tweet this!</a>								
1396	▼ 149	Damien Benveniste			0.14544	4	2mo	
1397	▼ 149	Maltexx			0.14546	14	14d	
1398	new	tnlintw			0.14556	1	6d	
1399	▼ 150	lwt99			0.14560	1	1mo	
1400	▼ 150	griffid0r			0.14564	3	1mo	
1401	▼ 150	Irina			0.14565	1	24d	

This model scored a test set RMSLE of 0.14544 which was definitely better than the earlier attempt and proves that this model is able to generalize rather well with the RMSLE score being 3.1% lower than the benchmark of 0.17649. My leader board rank also jumped up from 1800 to 1395 out of 2309 teams as of 2017-11-13 19:34. That is a jump of 408 up the ranks. Hence, the Random Forest was successfully refined. Now to move towards pushing up our accuracy by stacking model predictions.

## Refinement 2: Stacking Model Predictions

Traditional stacking of models is a more complicated process in which one model is trained and generates predictions which are then fed into a second model. This normally improves the accuracy but not by a very large amount. But in a situation like a Kaggle competition, a 0.001% reduction in the error metric could result in a jump of 80 ranks on the leader board.

I have not implemented traditional stacking of models here but have instead simply generated predictions from the first and second-best models and then identify a combined weighted average of their results that might possibly reduce the error.

So far, I have trained a series of models and tested their Logarithmic RMSE scores using the Kaggle Submission portal. In conclusion, I will use a stacked regression model by taking a weighted average of the two best models, the XGBoost and the Lasso Regression, and checking its performance against the individual models' Logarithmic RMSE scores. The model performance summary has been shown below in the figure:

10 submissions for UdaPaul		Sort by	Most recent
All	Successful	Selected	
Submission and Description		Public Score	Use for Final Score
<a href="#">lasso_submission.csv</a> 9 minutes ago by UdaPaul Lasso Model with GridSearchCV Ver 1.0		0.12671	<input checked="" type="checkbox"/>
<a href="#">xgb_submission.csv</a> an hour ago by UdaPaul XGBRegressor with GridSearchCV Ver 1.0		0.14029	<input checked="" type="checkbox"/>
<a href="#">rf_submission.csv</a> 20 hours ago by UdaPaul RandomForest with GridSearchCV #2		0.14544	<input checked="" type="checkbox"/>
<a href="#">benchmark_submission.csv</a> a day ago by UdaPaul Benchmark Submission		0.17649	<input checked="" type="checkbox"/>

As is seen from the above figure, the Lasso Regression and XGBoost models had the best and second best RMSLE scores. So, I will explore weighted averages of these models.

The first step was to get RMSLE error scores on the training data using cross validation for each weight between 0 to 1 in increments of 0.1. The results are displayed in the table below:

	XGBoost Weight	Lasso Weight	Training RMSLE
0	0.0	1.0	0.106871
1	0.1	0.9	0.100217
2	0.2	0.8	0.093730
3	0.3	0.7	0.087444
4	0.4	0.6	0.081409
5	0.5	0.5	0.075684
6	0.6	0.4	0.070344
7	0.7	0.3	0.065483
8	0.8	0.2	0.061217
9	0.9	0.1	0.057676
10	1.0	0.0	0.055002

From this table I can see that the XGBoost model performs better on the training set. The RMSLE score reduces as the weight on the XGBoost model increases. From the earlier bar plots we had seen that the XGBoost model had a higher RMSLE score for the training set than the test set. This might be due to some level of overfitting since the XGBoost model was cross validated on the training set. The Lasso model does have a lower RMSLE error for the test set compared to the Training set. Hence, I feel like it should have a slightly higher weightage than 0.0. I will test the RMSLE scores for the test data from Kaggle for the following stacked models:















- Model 1 :: XGBoost(0.9) + Lasso(0.1)
- Model 2 :: XGBoost(0.7) + Lasso(0.3)
- Model 3 :: XGBoost(0.6) + Lasso(0.4)
- Model 4 :: XGBoost(0.2) + Lasso(0.8)

The results for these are displayed below:

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions Submit Predictions		
<div> <div>✓ Selected submissions updated</div> <div>You should thus choose submissions that will most likely be best overall, and not necessarily on the public subset.</div> </div>		
<div> <div>11 submissions for UdaPaul</div> <div>Sort by Most recent</div> </div>		
<div> <div>All Successful Selected</div> </div>		
Submission and Description	Public Score	Use for Final Score
<div>Stacked_Model_4.csv</div> <div>5 minutes ago by UdaPaul</div> <div>Model 4 :: XGBoost(0.2) + Lasso(0.8)</div>	0.12526	<input type="checkbox"/>
<div>Stacked_Model_3.csv</div> <div>6 minutes ago by UdaPaul</div> <div>Model 3 :: XGBoost(0.6) + Lasso(0.4)</div>	0.12883	<input type="checkbox"/>
<div>Stacked_Model_2.csv</div> <div>6 minutes ago by UdaPaul</div> <div>Model 2 :: XGBoost(0.7) + Lasso(0.3)</div>	0.13101	<input type="checkbox"/>
<div>Stacked_Model_1.csv</div> <div>6 minutes ago by UdaPaul</div> <div>Model 1 :: XGBoost(0.9) + Lasso(0.1)</div>	0.13677	<input type="checkbox"/>

The output for the 4th Stacked model is shown below. It scored the best at 0.12526, a 40.5% improvement over the benchmark score of 0.17649. This model advanced me in terms of relative scoring by 76 places from the previous best performance of 0.12671 achieved by the Lasso Regression as is shown in the figure below.



Overview	Data	Kernels	Discussion	Leaderboard	Rules	Team	My Submissions	Submit Predictions
834	▼ 98	Jin Wang						0.1251621mo
835	new	wolf@nyc						0.12516163d
836	new	Luxifer						0.12523134d
837	▲ 160	Guanjie						0.12523283d
838	▼ 101	wangjieyi						0.1252651mo
839	new	UdaPaul						0.1252615now
<div><div>Your Best Entry ↑ You advanced 76 places on the leaderboard! Your submission scored 0.12526, which is an improvement of your previous score of 0.12671. Great job!</div><div> Tweet this!</div></div>								
840	▼ 102	Mikhail A						0.12528922d
841	▼ 102	Jeffrey Pacino						0.1253011mo
842	▼ 102	Claus Gaarde Pedersen						0.1253022mo
843	▼ 102	Double Lee						0.1253411mo
844	▼ 102	BTsang						0.12534512d
845	▼ 102	w1215						0.12534221mo
846	new	Ding Ruixue						0.1253413d

## IV. Results

### Model Evaluation and Validation

The 4th stacked model was the best amongst every model I have trained and scored. The general trend was that the RMSLE was higher when the XGBoost model had a higher weight while the Lasso model having a higher weight gave a lower RMSLE score. Just to be sure I will check a couple of Stacked model with the following configuration where the Lasso model will have weights of 0.9 ,0.7, 0.6 and 0.5 to see if predictions that are closer to an average are more accurate.

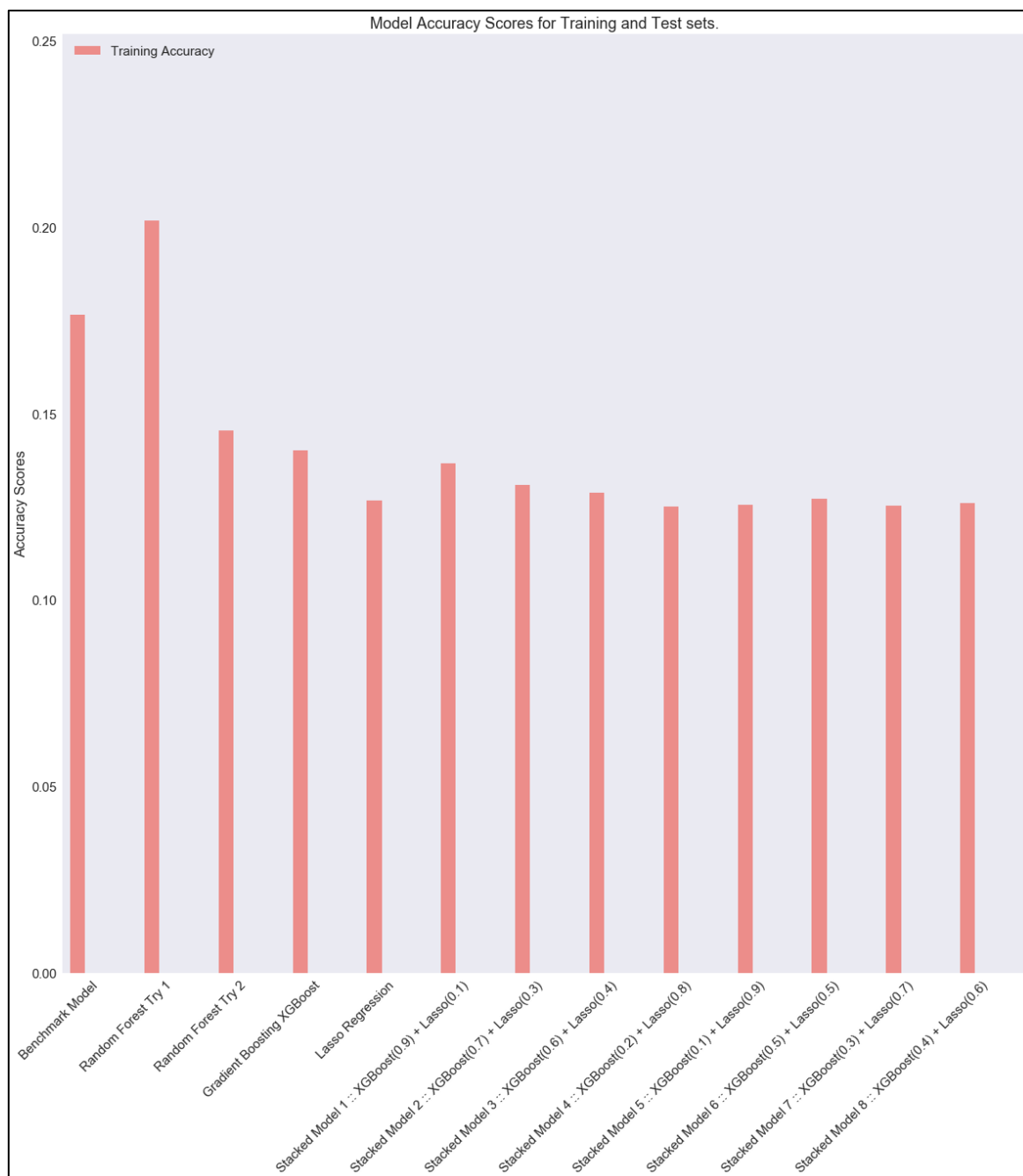
While these models did have better performances than those in which the XGBoost model had a weight larger than 0.5, the 4<sup>th</sup> Stacked model with the configuration: XGBoost(0.2) + Lasso(0.8) still had the lowest RMSLE error. I compiled all the models trained so far and their RMSLE error scores on the test data.

Models	Test data RMSLE from Kaggle
0 Benchmark Model	0.17649
1 Random Forest Try 1	0.20190
2 Random Forest Try 2	0.14544
3 Gradient Boosting XGBoost	0.14029
4 Lasso Regression	0.12671
5 Stacked Model 1 :: XGBoost(0.9) + Lasso(0.1)	0.13677
6 Stacked Model 2 :: XGBoost(0.7) + Lasso(0.3)	0.13101
7 Stacked Model 3 :: XGBoost(0.6) + Lasso(0.4)	0.12883
8 Stacked Model 4 :: XGBoost(0.2) + Lasso(0.8)	0.12526 **
9 Stacked Model 5 :: XGBoost(0.1) + Lasso(0.9)	0.12571
10 Stacked Model 6 :: XGBoost(0.5) + Lasso(0.5)	0.12714
11 Stacked Model 7 :: XGBoost(0.3) + Lasso(0.7)	0.12535
12 Stacked Model 8 :: XGBoost(0.4) + Lasso(0.6)	0.12598

(\*\* - Best Final Model)



It is easily visible that the Stacked Model 4 had the best performance. I visualized the progression of the test data RMSLE scores below:






















1. From this bar chart, it is visible that the RMSLE score was about 0.17649 for the Benchmark.
2. The Random Forest on the first attempt overfitted and gave an RMSLE score of 0.2. After refining it via tuning the hyperparameters the improved Random Forest model scored a 0.14544 which was better than the Benchmark score.
3. The XGBoost model still performed better scoring a 0.14029 which was a slightly better improvement.
4. The Lasso Regression gave a more significant drop in the RMSLE error with a score of 0.12671
5. The Stacked models had a range of error scores with the minimum error attained for the Stack Model 4 :: XGBoost(0.2) + Lasso(0.8) which had an error of 0.12526 and the best Leaderboard rank of 839, up from the Benchmark Leaderboard rank of 1800.

The final model performed much better than the benchmark, however I would not trust it's predictions blindly. It was tested on a wide range of inputs of unseen data was able to generalize with an accuracy of 87.44% (1.0- 0.1256). However, I believe that it is still not very robust to outliers and may need validation on more unseen data. Unfortunately, I did not have any more unseen data on which to validate the model. I might have split the test set into a holdout validation set, but did not do that since I didn't have the test set labels and needed to submit predictions for the whole set to Kaggle in order to get the model scored. Hence, I would summarize by saying that this model, while 87.44% accurate, is still not accurate enough. I would expect a model with more than at least 95% to be trustworthy.

## Justification

The final solution was a weighted combination of the predictions of the XGBoost and Lasso Regression models. It had an accuracy of 87.44% i.e. a Root Mean Square Logarithmic Error (RMSLE) of 0.12526. The Leaderboard rank for this model was 836 out of 2310, jumping up 964 ranks from the benchmark ranking of 1800. This put me in the top 35% of the Leaderboard compared to the benchmark which was in the bottom 20% of the Leaderboard. The RMSLE has reduced from the benchmark to the final model by a small amount of 5.12% but I believe the scale of improvement is more tangible in terms of the Leaderboard ranking with a jump up the ranks by 964 places.

The final solution is definitely a massive improvement over the benchmark. However, the top 10 of the Leaderboard, boasts models having RMSLE scores less than 5%. These are the top 20 models for the Kaggle Competition:

#	Δ1w	Team Name	Kernel	Team Members	Score 🏆	Entries	Last
1	—	totylkoja			0.00000	46	16d
2	—	totylkoja2			0.00000	10	16d
<a href="#">Overview</a> <a href="#">Data</a> <a href="#">Kernels</a> <a href="#">Discussion</a> <a href="#">Leaderboard</a> <a href="#">Rules</a>							
4	—	WorldStar			0.04139	10	13d
5	—	Pluto			0.04181	10	13d
6	—	facepalm			0.04181	66	13d
7	—	DSXL			0.06628	2	2mo
8	—	DongciDaci			0.10972	43	17d
9	—	RoeE Eliya			0.11089	31	2mo
10	—	ZavodRobotov			0.11140	1	1mo
11	—	Just Average			0.11215	5	1mo
12	—	kezman9			0.11256	155	13d
13	—	Luciano Paiva			0.11275	2	2mo
14	—	LTK			0.11297	1	2mo
15	—	Meng-Gen Tsai			0.11298	32	2mo
16	▲1	Mikhail Rubinov			0.11300	150	16h
17	▼1	fffffgggg			0.11319	65	1mo
18	—	Yeonsu			0.11324	1	1mo
19	—	Yakolle Zhang			0.11327	11	1mo
20	—	LogikChaos			0.11335	24	18d

Hence, it is obviously possible to create a model with 0% error. Hence, this final model is not the best solution to the problem. I will explore what went wrong and what improvements can be made in the next section.

## V. Conclusion

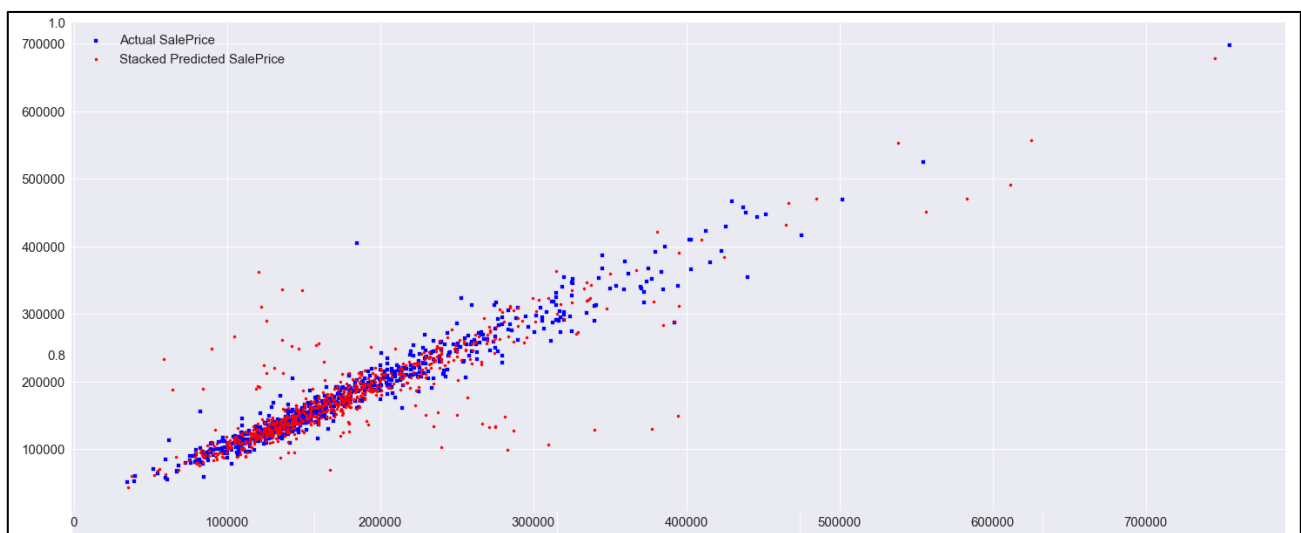
---

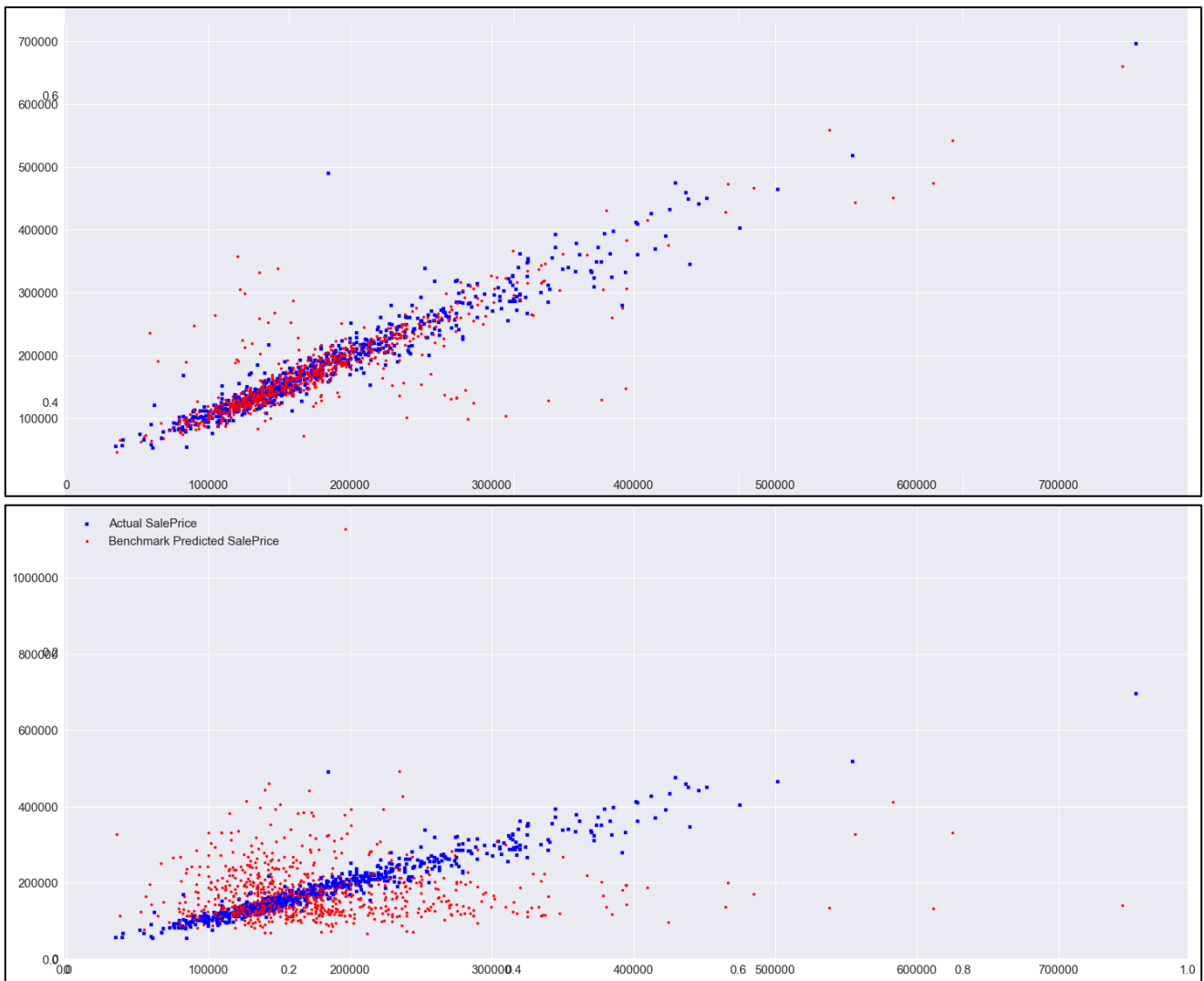
### Free-Form Visualization

The free-form visualization section was a part of the project that took me some time to figure out since I was not sure what type of a visualization would capture the essence of this project while also giving a clear depiction of the solution.

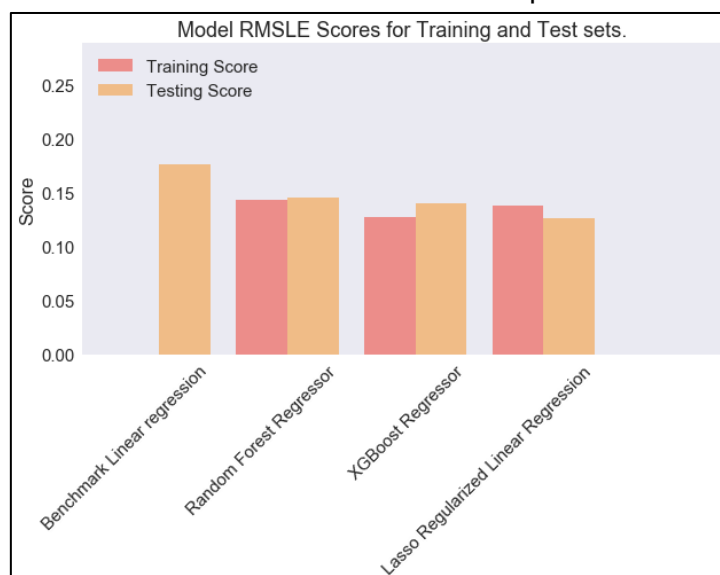
I finally decided that a good way to summarize the results of this project would be to create a scatter plot of the actual Training data SalePrices with the predicted SalePrices of the Benchmark, Lasso and Best Stacked model superimposed on it. In my opinion, this would give an accurate visual of how well each of these models were able to capture the linear and non-linear relationships in the training dataset. Ideally, I would have wanted to create this visual using the testing data but since Kaggle does not provide me with the actual test dataset SalePrice this was the best visual I could think off to capture how well each model performed. I also created a bar chart comparing the training and testing RMSLE scores of the Benchmark, the XGBoost and Lasso regression models prior to combining them. In addition to the model performances, I also decided to plot a bar chart of each model with its respective RMSLE score and the corresponding line plot of the Leaderboard rank superimposed on it. This will serve as a visual of how each model affected my Leaderboard ranking.

**Plot 1:** The comparison of the Stacked, Lasso Regression and Benchmark model. The first scatter plot shows that the Stacked model did quite a good job of fitting the training data but with some generalization visible as well. The second scatter plot is for the Lasso Regression model which looks almost similar and makes sense considering that their RMSLE scores were quite close; 0.12526 for Stacked and 0.12671 for the Lasso model. The last scatter plot shows the Benchmark model and it is clear that this model did not fit the training data very well at all resulting in the higher RMSLE score of 0.17.

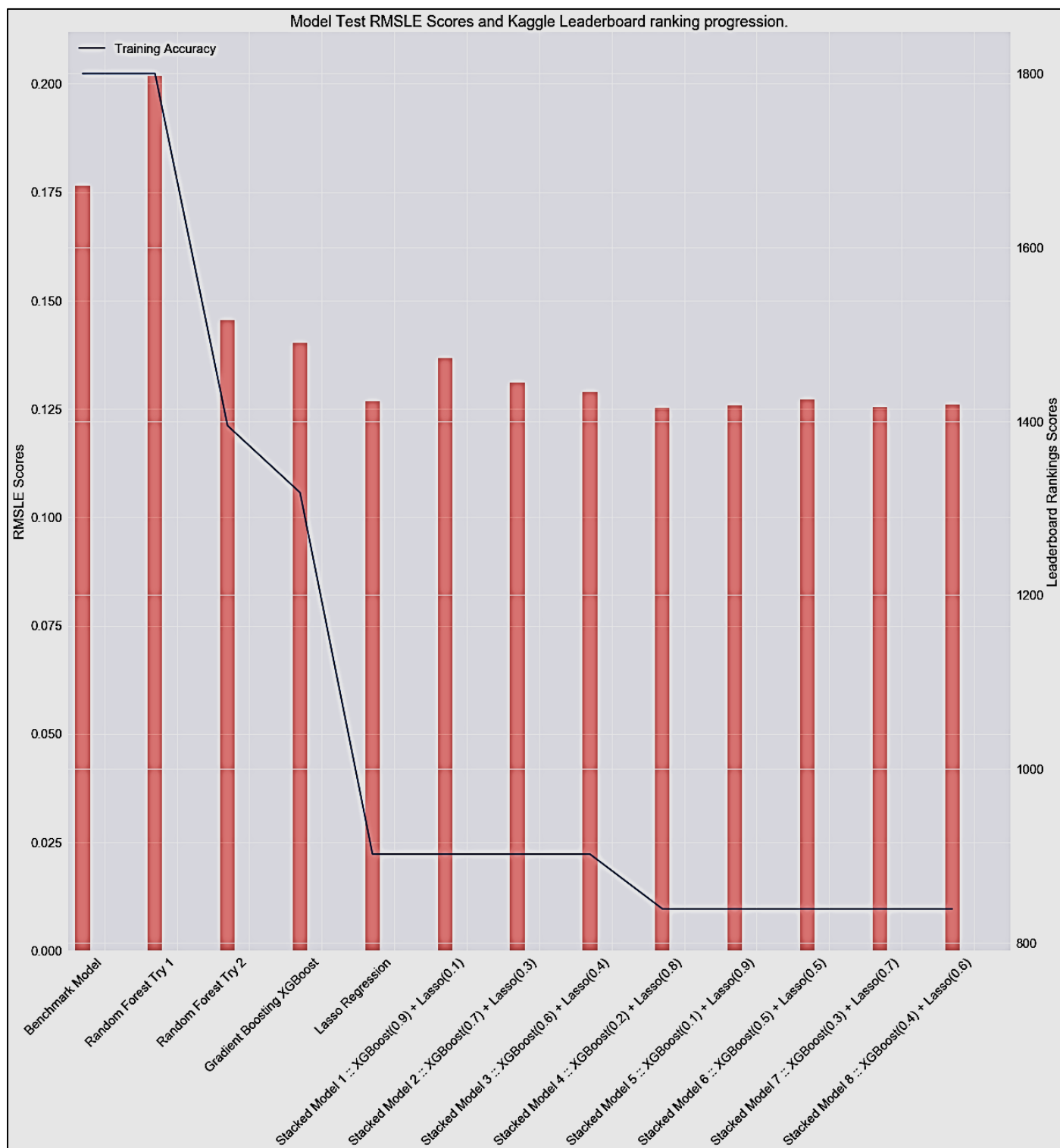




**Plot 2:** This second plot was one that I created before beginning to combine the predictions from the XGBoost and Lasso Regression models. It shows the relative RMSLE scores on the training and testing data. Also, it is visible how the XGBoost model performed much better on the training data than the testing data which led me to suspect some level of overfitting. On the other hand, the Lasso model did a much better job on the testing data which validates that it is definitely the better choice. This also explains why the stacked model having a higher weight for the Lasso regression gave a better accuracy compared to the when the XGBoost model's predictions had a higher weight



**Plot 3:** The last plot summarizes every model I tried during the course of this project, it's respective RMSLE score and the corresponding Leaderboard rank that model achieved. Hence, this plot acts as a depiction of my journey from the benchmark model to my best final model, showing how my Leaderboard ranking and Accuracy improved over time.



## Reflection

When I started off, this project was a simple Kaggle competition that required me to use Regression techniques to predict House Sale Prices. A training dataset with the corresponding dependent variable labels were provided as inputs for training models while a testing dataset was provided in order to make predictions after fitting the models to the training data. These predictions were then uploaded to the Kaggle webpage and the predictions were scored using the Root Mean Square Logarithmic Error (RMSLE) as an error metric.

I started by exploring the response variable i.e. SalePrice and found that it was heavily skewed. This was corrected and the SalePrice feature was normalized using a logarithmic transformation. Next, I proceeded to explore the various input features with an aim to understand their effect on the SalePrice variable. Correlation heatmaps allowed me to first identify the highly correlated variables as well as those which might be redundant. Next, I shortlisted features that I considered worth exploring further and used scatter plots and boxplots to analyse the shortlisted numerical and categorical features respectively.

After this, I summarized the percentage of missing values and depending on the percentage of missing values I dropped features that had more than 20% of missing features while I used imputation for the rest of the features. Lastly, I used dummy variable encoding on all of the categorical variables to prep the datasets for my Regression models. With this, my data preprocessing was completed and I could move on to training models.

The model was the benchmark Linear Regression which was trained on a small subset of features selected by me using my intuition as to which features would be important in driving the Sale Price of a house. This model had a test data RMSLE error of about 0.17 and gave me a Kaggle Leaderboard ranking of 1800 out of 2310 participating teams.

With the Benchmark ready, I then proceeded to train a Random Forest, XGboost and Lasso Regression models, tuning their parameters using Grid Search Cross Validation on the training dataset. The Random Forest seemed to show signs of overfitting in spite of using cross validation with an error of 0.2 which was higher than the benchmark, while the Lasso Regression model showed the best performance with the lowest test data RMSLE error of 0.12671.

I used tuning to improve the Random Forest's accuracy but could still achieve a minimum error of 0.145. Another approach of improving my accuracy and advancing up the Leaderboard rankings was by combining the predictions of multiple models using weighted averages. The XGBoost and Lasso models had given me the best performance so far so I chose these for my combined model. I iteratively explored different weighted combinations of these two models' predictions and found the best combination to be: XGBoost(0.2) + Lasso(0.8). This combined model gave the lowest RMSLE error of 0.12526 and a Leaderboard rank of 839.

This project was a challenge to work with because of rather anomalous results that I got in terms of the features available. Deciding on which regression model to choose was also not an easy task since

I was tempted to try a Neural net regression. However, I reasoned that due to the presence of so many categorical features, decision trees and related ensemble methods like Random Forests and Gradient Boosted Trees would perform better and help capture the smallest of relationships between the input features and the target variable.

My final solution had a 87.44% accuracy but based off the Kaggle Leaderboard, there were teams that managed to achieve a 100% accuracy using actual model stacking in which one model was fitted to the training data and it's predictions would then be fed into a second model and so on. This could be an improvement on my current approach. I will discuss possible improvements in the next section.

## Improvement

The final solution I got was definitely an improvement over my benchmark solution. However, I do not believe that it is the best solution. As discussed earlier, there are teams that have achieved 100% accuracy using other approaches. I did explore multiple kernels on the Kaggle competition page in an attempt to identify how I might improve on my current project. I identified the following improvements:

1. A number of kernels dealing with preprocessing identified multiple numerical input features that were skewed to varying degrees and normalized them using logarithmic or Box-Cox transformations. This was something that I did not think to check for and it is possible that normalizing these input features would have improved the performance of Linear regression models like my Lasso Regression algorithm. Decision tree based algorithms like Random Forests or Gradient Boosting Trees however would not have benefited a lot from this.
2. Another approach I saw in a lot of kernels was outlier removal. I did explore some of the features that I saw outliers in, but decided against removing them since I reasoned that they might capture information on cases that did look like outliers but could show up in the test set as well since, there do tend to be houses with large areas or garages etc. that might still have lower Sale Prices that do not keep with the general trend. However, it is possible that these outliers could be affecting my model performance and removing them might help improve the performance of my models.
3. Lastly, an approach that I researched at the beginning of this project was Model Stacking that uses a stack of different connected models in which the output of one model feeds into the next and so on. A lot of the teams that used this approach did show much lower error scores. Unfortunately, I was not able to quite grasp the implementation of this method in code and decided against using it. Nonetheless, based on my observations, I would agree that had I known and understood this approach I would have definitely used it and could have gotten better results.
4. Lastly, another approach that I think I might have overlooked is the use of Neural Network. I did reason that Neural Nets would perform better in a classification application and dismissed of this approach. But it is possible that one hot encoding all of the input features and

implementing a MLP architecture with a small number of hidden layers, removing the softmax activation function and defining the cost function in terms of Mean Squared Error could possibly give a solution that was comparable or perhaps even better. This is something that I would consider researching and applying to at least prove or disprove my assumption about Neural Nets not performing as well on regression problems as they do for classification or clustering problems.

Therefore, these are all the improvements that, in my opinion, could be made or at least researched in an attempt to improve on an accuracy score that, based on Kaggle's Leaderboard for this competition, can definitely be increased.

## References:

- [1] [https://en.wikipedia.org/wiki/Regression\\_analysis](https://en.wikipedia.org/wiki/Regression_analysis)
- [2] <https://www.openintro.org/stat/data/ames.php>
- [3] <https://www.kaggle.com/c/house-prices-advanced-regression-techniques#description>
- [4] [https://en.wikipedia.org/wiki/Root-mean-square\\_deviation](https://en.wikipedia.org/wiki/Root-mean-square_deviation)
- [5] <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=8&cad=rja&uact=8&ved=0ahUKEwjx5o2d2LTxAhVWwWMKHWwWDeAQFghbMAc&url=http%3A%2F%2Fww2.amstat.org%2Fpublications%2Fjse%2Fv16n2%2Fdatasets.pardoe.pdf&usg=AOvVaw2On9ZvayT59Lr5EjoDn1Qi>
- [6] <https://ageconsearch.umn.edu/bitstream/.../2004-9-house%20price%20prediction.pdf>
- [7] <https://ww2.amstat.org/publications/jse/v19n3/decock/datadocumentation.txt>