House Prices Prediction

We know that house sale prices follow specific trends depending on Location, condition,

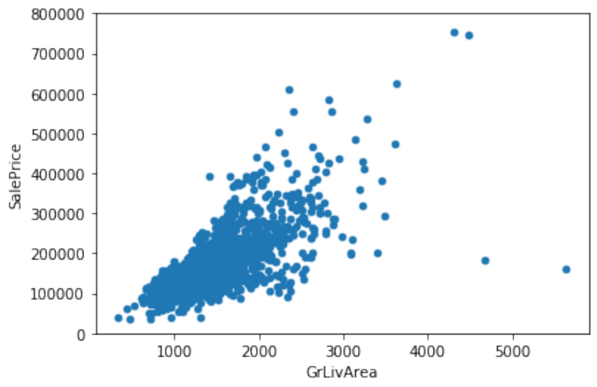
features etc. This project aims at applying advanced regression techniques to predict house sale prices given a set of features. The dataset will be a combination of several features such as interior, exterior, area, location, heating, number and condition of rooms etc. We will then train our models depending on these features and the respective sale prices. In the end we will run our model to predict house sale prices and test the accuracy of our results.

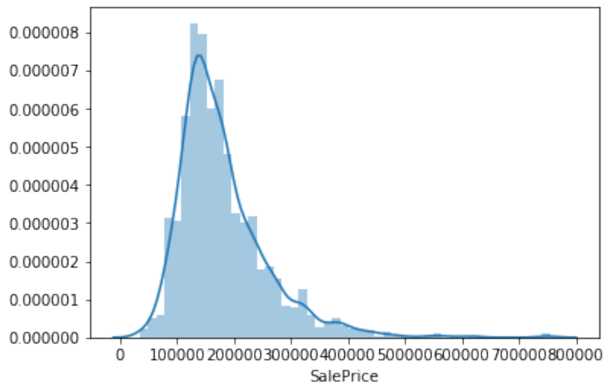
Analyzing the Dataset

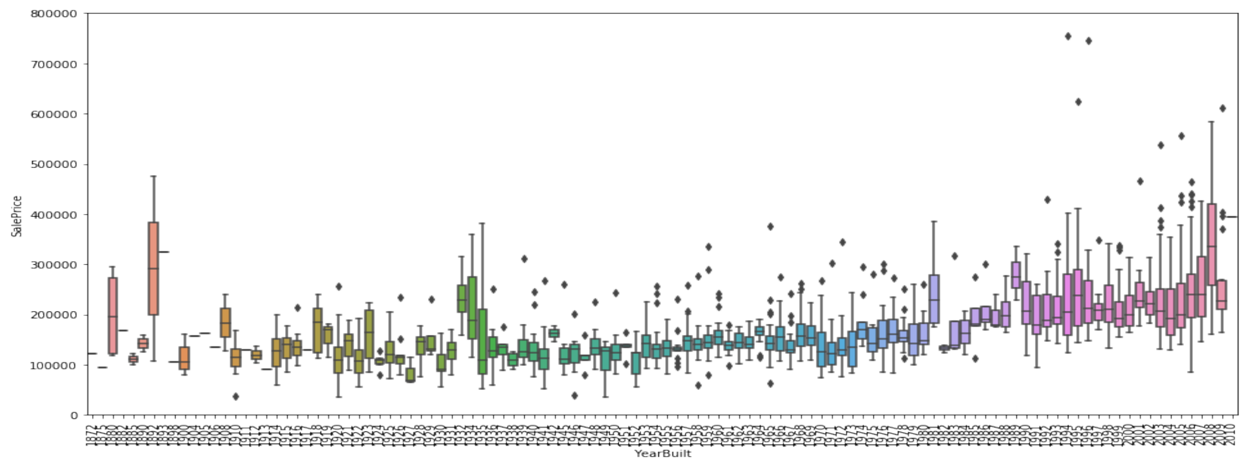
The dataset consists of several features of each house and the respective prices. Having a birds eye view of the data shows that the data is strongly and linearly correlated which forms the basis for our machine learning models.

The sales price shows the following trends:

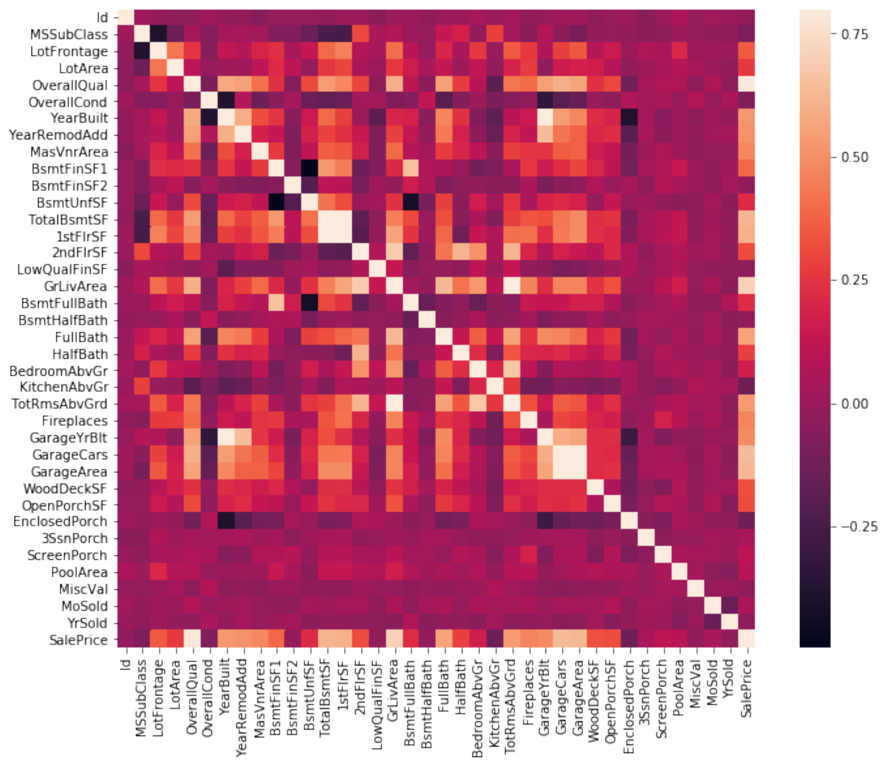
- Deviate from the normal distribution.
- Have appreciable positive skewness.
- Show peakedness.
- Sales Price and GrLivArea have a linear relationship
- Sale prices increase per year.

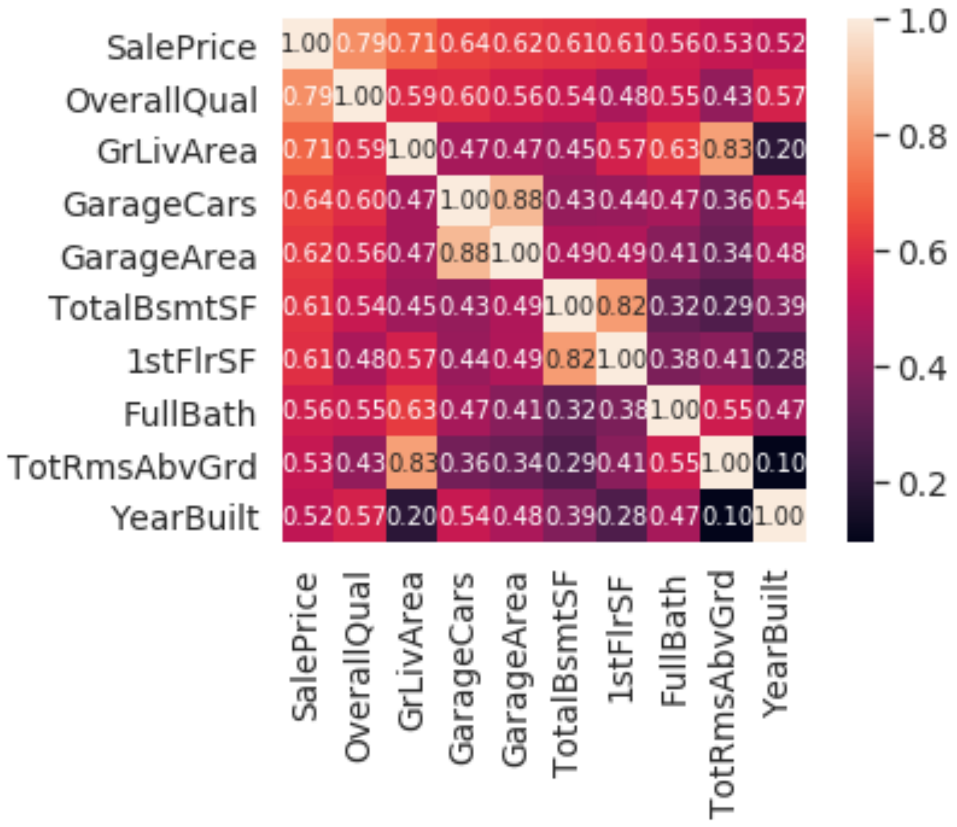






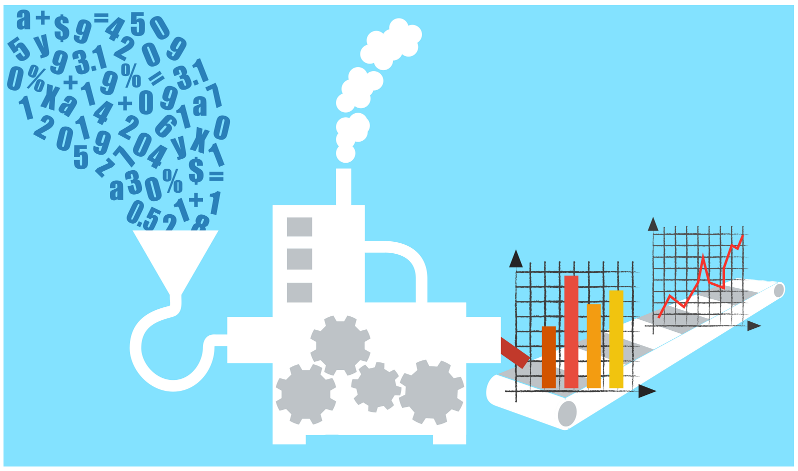
- OverallQual, GrLivArea and TotalBsmtSF are strongly correlated with SalePrice.
- GarageCars and GarageArea are also some of the most strongly correlated variables.
- TotalBsmtSF and 1stFloor are also strongly correlated variables.
- There are two red colored squares:
- The first one refers to the TotalBsmtSF and 1stFlrSF variables, and the second one refers to the Garage*X* variables.
- The correlation is so strong that it can indicate a situation of multicollinearity.
- TotRmsAbvGrd and GrLivArea are also correlated.
- TotalBsmtSF and 1st Floor also show co-relationship.





SalePrice correlation matrix (zoomed heatmap

Data Preprocessing



First we have to check for missing values. The following results were found after summing the missing values for each column.
Looking at the results it is quite evident that the there are features which have a large set of missing values and in turn gives us the motivation to preprocess the data and standardize it.

```
              Train    Test
Alley          1369  1352.0
BsmtCond         37    45.0
BsmtExposure     38    44.0
BsmtFinSF1        0     1.0
BsmtFinSF2        0     1.0
BsmtFinType1     37    42.0
BsmtFinType2     38    42.0
BsmtFullBath      0     2.0
BsmtHalfBath      0     2.0
BsmtQual         37    44.0
BsmtUnfSF         0     1.0
Electrical        1     0.0
Exterior1st       0     1.0
Exterior2nd       0     1.0
Fence          1179  1169.0
FireplaceQu     690   730.0
Functional        0     2.0
GarageArea        0     1.0
GarageCars        0     1.0
GarageCond       81    78.0
GarageFinish     81    78.0
GarageQual       81    78.0
GarageType       81    76.0
GarageYrBlt      81    78.0
KitchenQual       0     1.0
LotFrontage     259   227.0
MSZoning          0     4.0
MasVnrArea        8    15.0
MasVnrType        8    16.0
MiscFeature    1406  1408.0
PoolQC         1453  1456.0
SaleType          0     1.0
TotalBsmtSF       0     1.0
Utilities         0     2.0
```

After analyzing the data I decided to get rid of features which had more than half of the missing information or didn't correlate to sales price.

The following features were dropped:

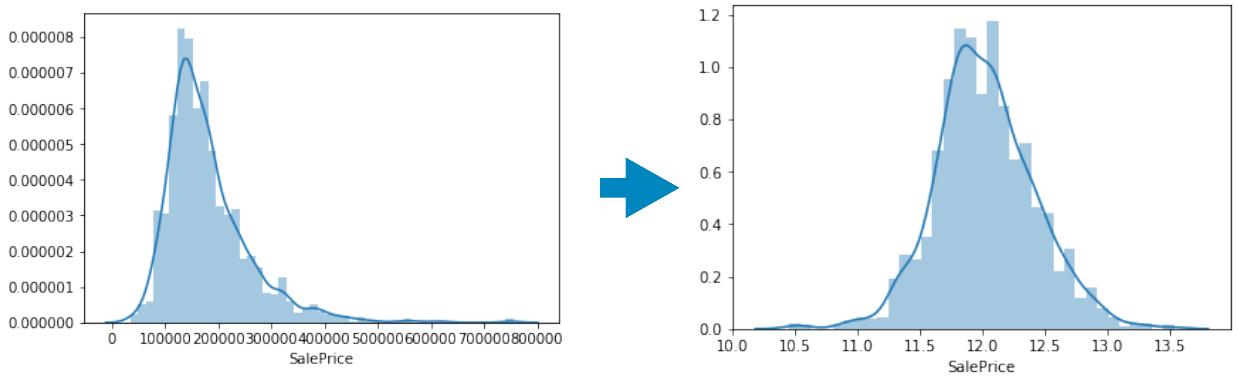
Utilities, RoofMatl, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, Heating, LowQualFinSF, BsmtFullBath, BsmtHalfBath, Functional, GarageYrBlt, GarageArea, GarageCond, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, PoolQC, Fence, MiscFeature, MiscVal.

The next task was to fill Nan values and convert types of features.

Feature Name	Conversion
MsSubClass	toStr
MsZoning	Fill Na with most common feature (mode)

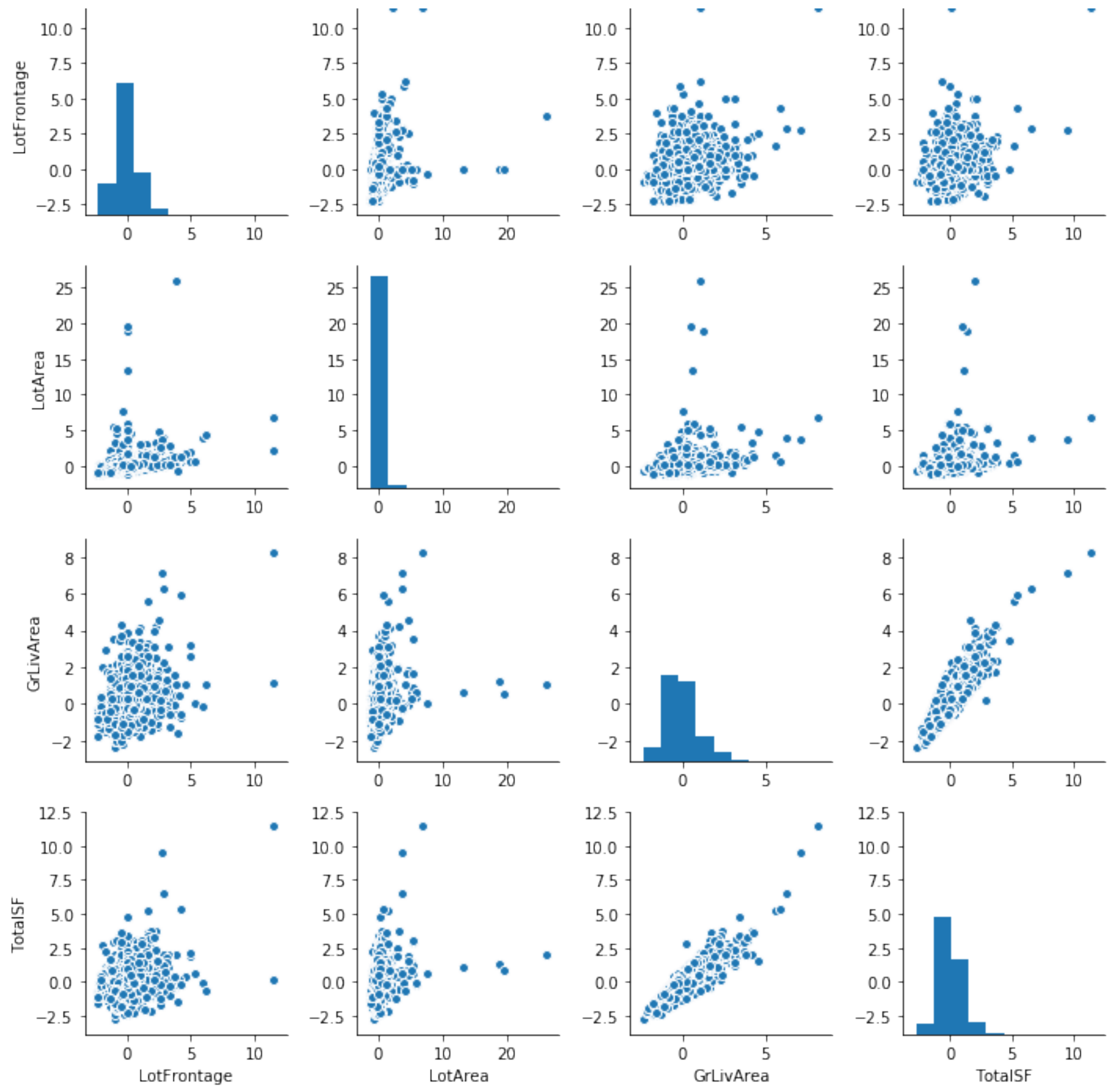
Feature Name	Conversion
LotFrontage	Fill Na with Mean
Alley	Fill Na with No Access
OverAllCond	toStr
MasVnrType	Mode
Bsmt	Fill Na with No Basement
TotalBsmtSF	Fill Na 0
Electrical	Mode
KitchenAbvGr	toStr
KitchenQual	Mode
FireplaceQu	Fill Na with NoFp
Garage	Fill Na with NoGRG
SaleType	Mode
YrSold	toStr
MoSold	toStr

The Sale price was skewed right and to make it symmetric i took log transform.
The resultant sales price looks much better and symmetric around a sale price of 12.0



Now I standardized the numeric data by taking the mean of each feature, subtracting it from each entry and dividing it by standard deviation.

The resultant plot is shown and it can be clearly seen that all of the values are now densified in the range 0-10, which will form the basis for our machine learning models.



Now that the data was standardized, I decided to split the data in training and test sets into 90% and 10% respectively using random selection.

	Matrix Rows	Matrix Columns
Training Data	1314	262
Test Data	146	262

Machine Learning Models

After analyzing and preprocessing it was quite clear that I needed to implement and compare multiple regression models.

There were two ways to approach it:

1. **Use methods learned in class like Gradient Descent and Normal Equation.**
2. **Use Libraries like Sklearn to implement Random Forest Regressor**

I decided to go with both of them, as I not only wanted to predict values from libraries like sklearn using Random Forest Regressor, but also test the accuracy of models learned in this course with my original python implementation.

The results were quite interesting and I succeeded in implementing both of them.

Regression through Normal Equation:

The concept behind normal equation is that given a labeled dataset we can find a weight matrix that can eventually be applied on test data for regression purposes. For example if we have a Matrix X containing n individual elements with each element normalized to a size of s resulting in a nXs matrix, we can multiply it with a weight matrix W which will give us the resultant matrix Y.

As X and Y are already available in the training data we can find the weight matrix W by using the equation:

$$Y = W.X$$

We can find W through the formula:

$$W = (XT.X)-1.XT.Y$$

where XT is the transpose of the matrix X.

Once we have the matrix W we can take a test object X and multiplying them gives us the resultant matrix Y.

Implementing Normal Equation to Predict House Sale Prices:

The key idea here was to set the matrix **X** as the standardized features of our training example and vector **Y** was set to the respective sale prices of those houses.

The resultant weight vector **W** gave us the weights that will ultimately be used to predict house sale prices.

The results were quite impressive and I achieved a root mean squared error of **0.58**

A **portion** of the expected and output values looked like this:

Output Value	Expected Value
11.91	12.61
12.27	11.58
12.01	11.92
11.66	12.21
11.70	11.77
12.14	11.75
12.13	11.81
11.56	11.63
12.03	12.07

Regression through Gradient Descent:

The concept behind gradient descent is to find the set of values for the weight vector W that gives the best results and the most accuracy for our model. We can achieve this by minimizing the error between our resultant and expected result.

The gradient descent finds the minima by taking the derivative of the error function and moving down hill opposite to the direction of gradient.

The resultant is multiplied by leaning rate and the respective weights are updated

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\ x_{id}$$

Also we can apply stochastic gradient descent to find the global minima and a more robust model:

$$\Delta w_i = \eta(t - o)\ x_i$$

Implementing Gradient Descent to Predict House Sale Prices:

The key idea here was to set the matrix **X** as the standardized features of our training example and vector **Y** was set to the respective sale prices of those houses.

The resultant weights were learned by first initializing them to 0.1 and then applying gradient descent with a learning rate of 0.0000000001 and 0.0000001 for batch and stochastic gradient descent respectively.

Results of Stochastic Gradient Descent:

The results were quite reasonable with a root mean squared error of **0.72**

A **portion** of the expected and output values looked like this:

Output Value	Expected Value
11.98	12.61
12.37	11.58
11.97	11.92
11.39	12.12
13.20	11.71
12.19	12.34
11.92	11.10
13.02	11.88
12.74	12.27

Results of Batch Gradient Descent:

Although the results were not as good as stochastic gradient descent but they were reasonably good with a root mean squared error of **0.82**

A **portion** of the expected and output values looked like this:

Output Value	Expected Value
11.92	12.61
12.41	11.58
11.13	12.21
11.17	12.12
12.35	12.17
13.53	11.84
11.49	12.17
11.47	12.23
12.09	12.56

Random Forest Regressor:

Random forests can be used for regression analysis and are in fact called Regression Forests. They are an ensemble of different regression trees and are used for nonlinear multiple regression. Each leaf contains a distribution for the continuous output variable/s.

The model was applied to fit to our training set and then the accuracy was measured through our test data.

Results of Random Forest Regressor:

The random forest regressor gave the best results with a root mean squared error of **0.13**

A **portion** of the expected and output values looked like this:

Output Value	Expected Value
12.39	12.46
12.24	12.06
11.65	11.60
11.86	11.97
11.74	11.80
11.37	11.37
12.06	12.03
11.52	11.57
12.26	12.34

Conclusion:

In all the Random Forest Regressor gave the best results followed by Normal Equation, Stochastic and Batch Gradient descent respectively.

The reason for higher accuracy with Random Forest Regressor is primarily due to the fact that it implements Ensemble Learning with Boot Strapping techniques.

Normal Equation gives the optimal solution for our linear models.

Stochastic gradient descent works better than batch gradient descent as it focuses on updating the weight matrix for each example rather than the whole batch, which in turn increases the likelihood of converging to a global minima, rather than getting stuck on a local one.