List of Figures

Contents

0.1 Abstract

The aim of the project is investigate the given data of houses and their sale prices, and create a learning model that is able to predict the price for new house data with least error.

The project will investigate several models and compare their accuracy while using a database to handle the huge amount of data.

The goal is to create a hypothesis capable of accurately predicting house price from given data. The tasks involved are:

1. Investigate the data[1], separate it randomly into leaning data and test data, and create appropriate database system for the data to be imported into.
2. Import both learning and testing data into separate tables
3. Engineer the features based on given data points.
4. Investigate several different models with a reduced data set and select a model to be trained on full data.[2]
5. Train selected model to generate the final hypothesis and test the hypothesis on test data.
6. Predict prices for kaggle test data and export it from database system

[1]Date is taken from kaggle dataset : https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

[2]Selection criterion to be determined

Chapter 1

Introduction

The aim of the project is investigate the given data of houses and their sale prices, and create a learning model that is able to predict the price for new house data with least error.

The project investigates several models and compares their accuracy while using a database to handle the huge amount of data. The data used is taken from Kaggle house pricing dataset[1]. The major part of project is data analysis and engineering where the given data is analyzed and modified to make it ready for learning models. The objective is to determine House price which is a linear regression problem, and thus linear regression models are studied. Over the course of project, L1 regularized linear regression model showed most promising results so it was used for final predictor.

SQL database is used to store the data on disk, while Pandas system acts as a database system for jupyter notebook. After jupyter notebook is done with the data, Pandas system makes appropriate changes to SQL database.

[1] Kaggle dataset

Chapter 2

Sytem Analysis

2.1 Existing System

The model(s) used for this dataset can be extended to other statistical problems like stock prediction, trend prediction, and eve sales prediction. In general, this project can be extended to any linear regression problem. There are several existing systems that solve these problems and employ different regression techniques and even ensemble techniques. Most models do not provide very good accuracy. One reason for that is poor data analysis and handling.

2.2 Proposed System

The system proposed in this project is expected to be much more efficient than existing systems. One metric to judge the accuracy of system is to let it be judged against other kernels on respective leaderboard at Kaggle.
A position in top 30% is expected for the selected dataset. There are more than 4500 model that are already making predictions with top 1500 having really low difference in their accuracy. Being judged at a position among these will prove effectiveness of the model.
Current model takes the data stored in database and learns from it. After that, it is able to make predictions for the data provided, either directly or from database.

The major problem is that Pandas stores all the data in RAM. This causes a major problem is data is too large. Using SQL system to store data, and only fetching data currently in use for Pandas prevents RAM from running out.

2.3 System Requirements

The system needs to be, at minimum, be able to run the kernel used to teach the model. The amount of memory used is equivalent to size of data processed during learning phase. Other than disk space requirements, other hardware requirements are not concrete, but low end system will exponentially slow down learning process.

2.3.1 Hardware Requirements

- Processor : Any modern x64 processor.
- RAM : Minimum 190 MB.
- Hard Disk Drive : Minimum 2 GB.

2.3.2 Software Requirements

- Operating System : Windows 7+ or any modern POSIX System.
- Python 3.5+ and its libraries : iPython
  SciPy
  NumPy
  Pandas
  Scikit-Learn
  Matplotlib and seaborn framework
  SQLAlchemy
  ConfigParser
  Pickle
  JobLib
- Jupyter Notebook.
- Database : MySQL (MariaDB) or SQLite.

Chapter 3

System Design

SQLite is used as primary database, and Pandas is used to manage data inside Python. The SQL database contains 2 tables called **train** and **test**. A python script imports this into a Pandas table, where relevant processing of data is done.
After processing the data, Pandas exporst the table into SQL database as 2 new tables called **testData** and **trainData**.
The second python script imports the tables **trainData** and **Y** into Pandas to create two new tables. The learning algorithm works on these and then pickles itself into a binary file.
The third python script imports the table **testData** into Pandas and unpacks pickled predictor. The predictor generates a new Pandas table that contains predictions for **testData** and exports it into SQL as a new table called **predictions**.
This table **predictions** are the desired predictions.

ER diagram in **Fig. 3.1** shows these relations. The project is split into 3 scripts to ensure modularity. The three scripts have to be run sequentially in order when doing the first run.
If new data is added, only script 1, and script 3 are needed to be run. script 2 that is responsible for training the model doesn't need to be executed unless the size of new data is very high.

The predicted values can be retrieved from the table **prediction**.
The need for using SQL database is that even though Pandas behaves like a RDBMS system, it doesn't store any data in disk. All the data during runtime is stored in RAM, and after the script is completed, data is lost forever. SQL helps in this case by first, preventing RAM from getting completely filled up due to large amount of data being processed and second, allowing data to be retained even after program is over.

The table Train is original training data. It has 81 columns. Removing the 2 columns Id, and index, it contains 79 columns of interest that give information about the house, and the column SalePrice tells the selling price of that house.
Preprocessor splits the table Train into two tables : Y and traindata. Y contains only one column of interest i.e. SalePrice. This SalePrice has been log transformed. TrainData contains 296 columns of interest that have been generated by preprocessor from the table train. The predictor learns from these who tables and stores the parameters on the disk.
The column Test is the data for which we need to predict the values of SalePrice. It is exactly same as the column Train except it doesn't have the column SalePrice. It is also processed to create table testData made up of 296 columns of interest.
Once predictor has learned from training data, it imports the testData table, creates a new table

and writes data in the table Prediction.
Prediction has two columns. One is index, other is SalePrice. SalePrice is a set of predicted values.

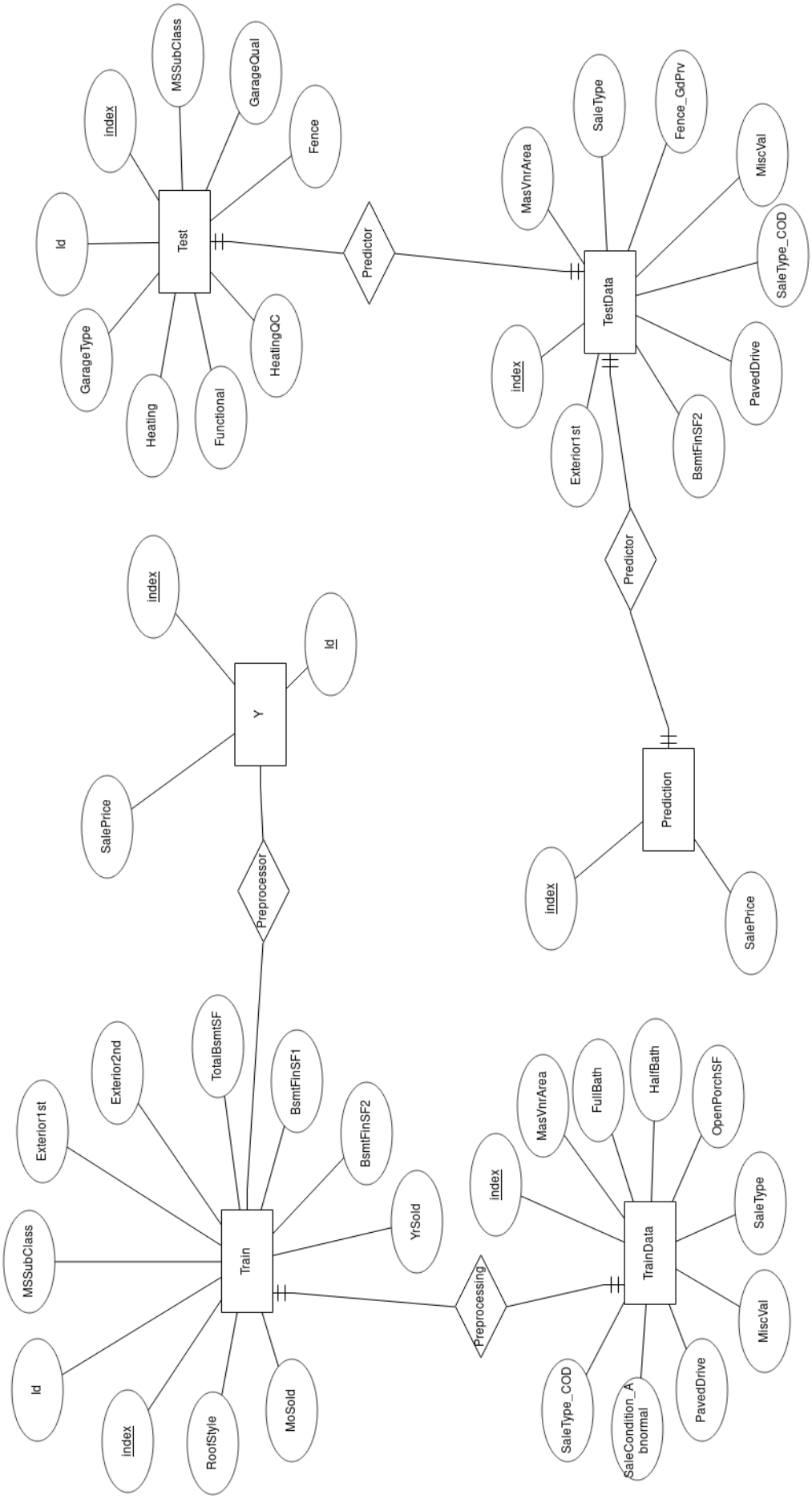


Figure 3.1: ER Digram

Chapter 4

System Implementation

4.1 Exploratory Data Analysis

This comprises of data input, data analysis, data cleaning and pre-processing. We need to eliminate multicollinearity, clean useless data ad transform given data appropriately to be used by the training model.

4.1.1 SalePrice

First the column ”SalePrice” is analyzed. This is sale price of the house and thus the parameter that our hypothesis is supposed to predict. We have 1460 values for SalePrice in the dataset.

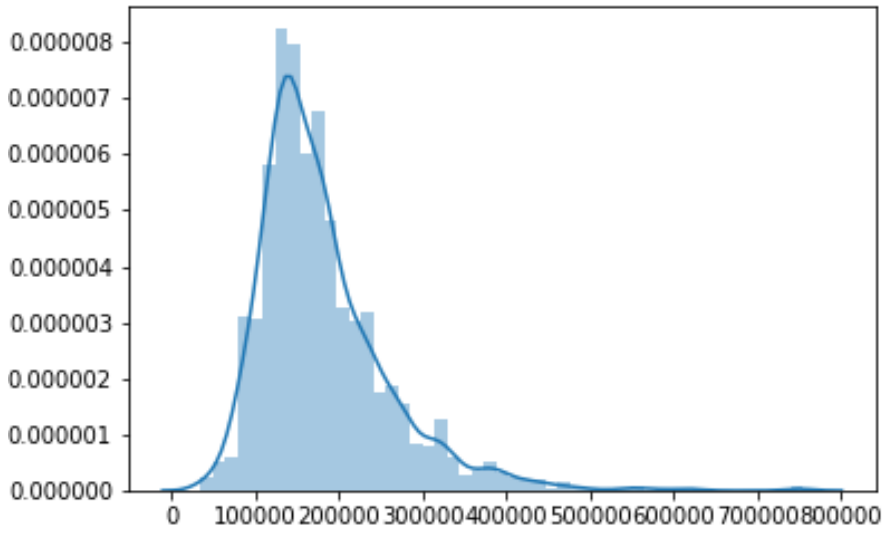


Figure 4.1: Frequency distribution for SalePrice

mean	180921.195890
std	79442.502883
min	34900.000000
25%	129975.000000

50%	163000.000000
75%	214000.000000
max	755000.000000

The distribution is roughly a positively skewed Gaussian distribution with a skewness of 1.88287

Pearson correlation coefficient[1] is used to determine relationships of various features. We will transform SalePrice later to make it closer to gaussian distribution. A heatmap based on correlation coefficient gives a pictorial representation of these relationships4.2.

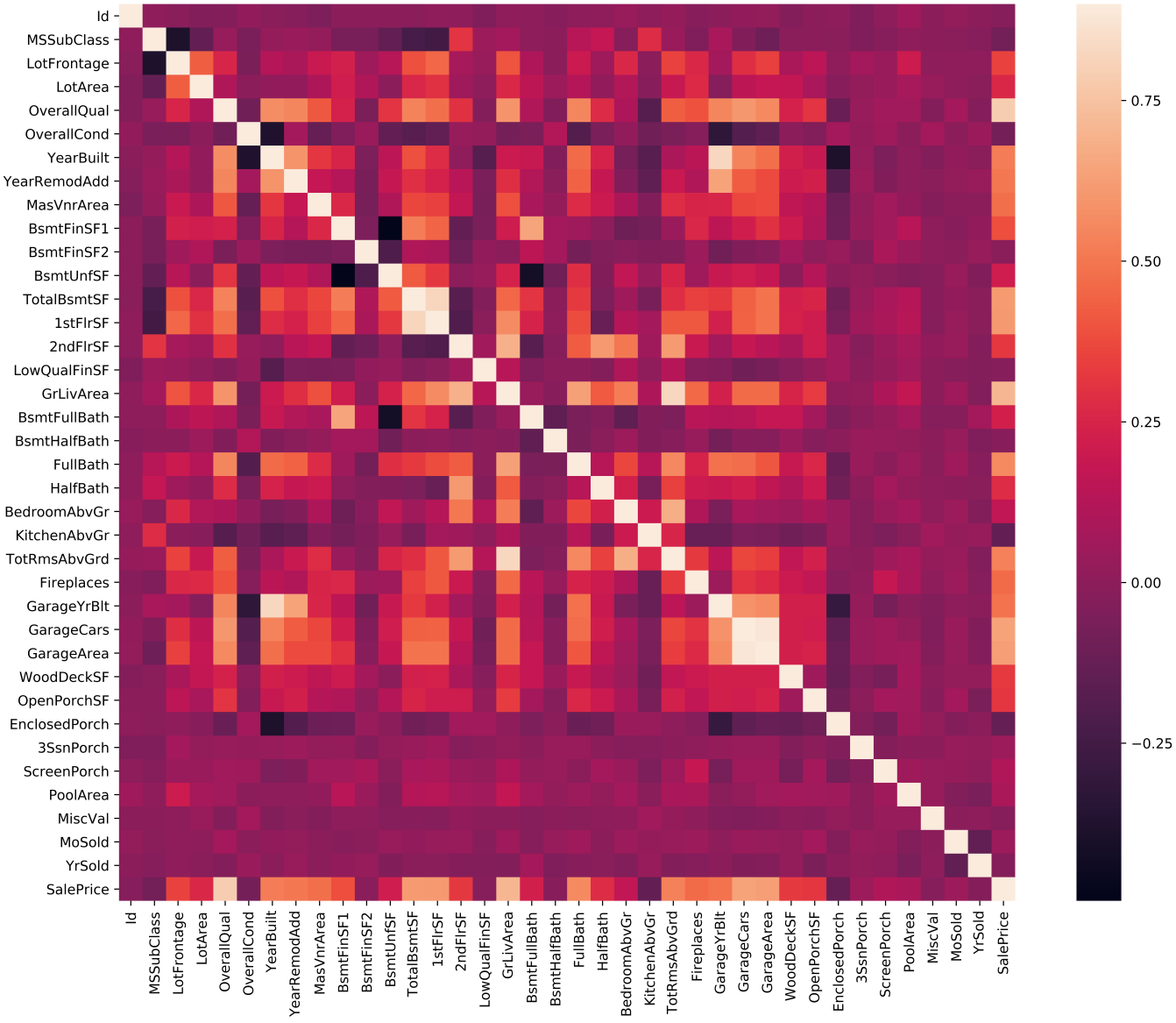


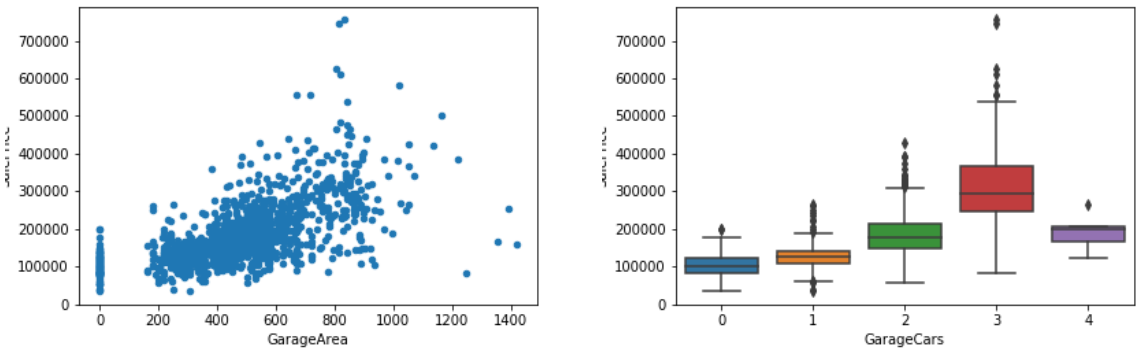
Figure 4.2: Heatmap made from correlation matrix of training data

[1] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

Lighter areas means the 2 corresponding features have high linear correlation. This helps us eliminate redundant features.

4.1.2 GarageArea and GarageCars

These two features have a high correlation coefficient of 0.882. According to data description provided:



(a) GarageArea vs SalePrice

(b) GarageCars vs SalePrice

Figure 4.3

GarageArea: Size of garage in square feet
GarageCars: Size of garage in car capacity

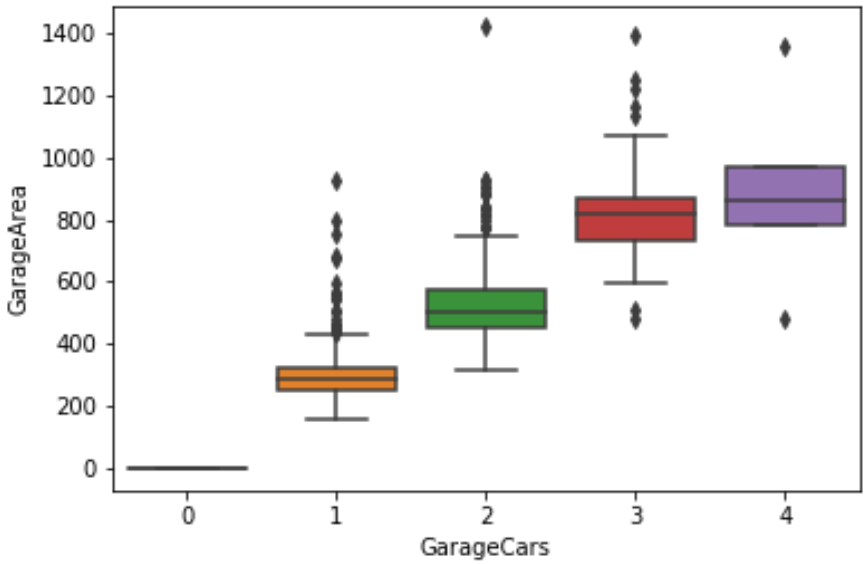


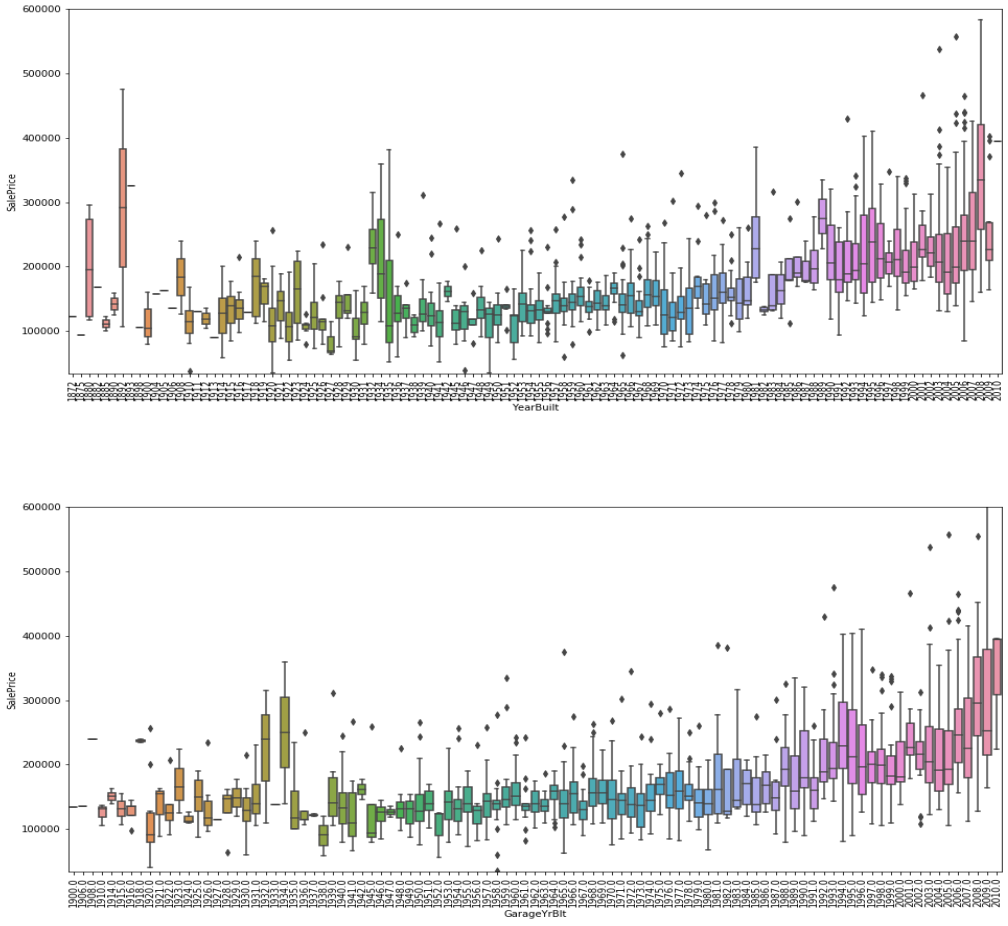
Figure 4.4: Scatter plot of GarageArea vs GarageCars.

Thus we can eliminate one of these 2 features since they give the same information. Both have a positive linear relation with SalePrice. **We choose to eliminate GarageArea** because

GarageCars has higher correlation with SalePrice. Correlation coefficient of GarageArea is 0.62 while that of GarageCars is 0.64. This is verified by the fact that even though GarageCars is not a very accurate metric, its more practical.
The SalePrice for GarageCars = 4 cannot be treated as an accurate representation as there are just four data points.

4.1.3 YearBuilt and GarageYrBlt

These features have a correlation coefficient of 0.825. According to data description provided:
YearBuilt: Original construction date
GarageYrBlt: Year garage was built



(b) GarageYrBlt vs SalePrice

Figure 4.5

Both parameters have roughly positive linear realtion with SalePrice. It shows the newly built houses tend to cost higher.
Even Though the two features have high correlation, this is due to the fact that in several cases, garage and house is built at same time. But this will not be true for all the cases. We will need the data from when garage was built in later dates for more accurate results. **Thus neither**

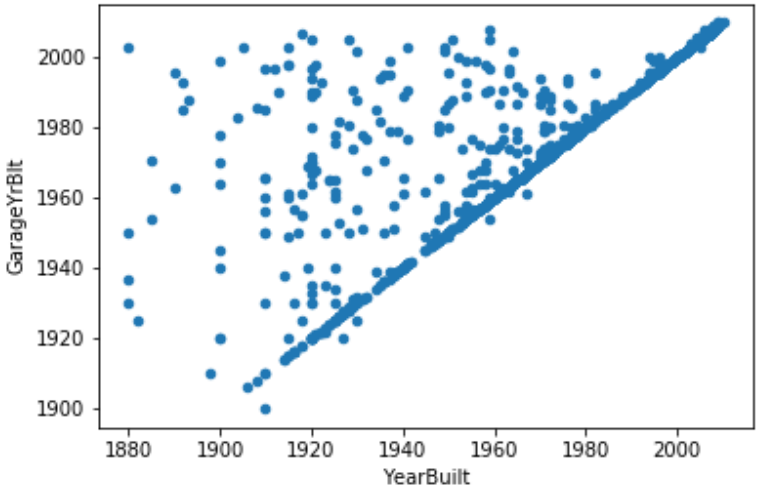
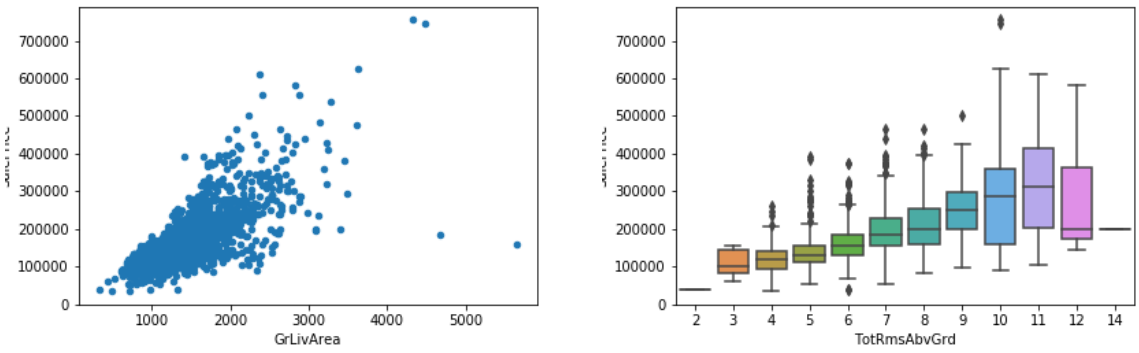


Figure 4.6: Scatter plot of YearBuilt vs GarageYrBlt.

feature is dropped.

4.1.4 GrLivArea and TotRmsAbvGrd

These two features have a correlation coefficient of 0.825. According to data description provided:
GrLivArea: Above grade (ground) living area square feet
TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)



(a) GrLivArea vs SalePrice

(b) TotRmsAbvGrd vs SalePrice

Figure 4.7

Both features tell more or less the same thing. They have a positive linear relation with SalePrice but there are too many outliers.
Since both featues show a strong linear relation, we can drop one feature. GrLivArea has a correlation coefficient of 0.71 with SalePrice while TotRmsAbvGrd has just 0.53, so **TotRmsAbvGrd is dropped**.

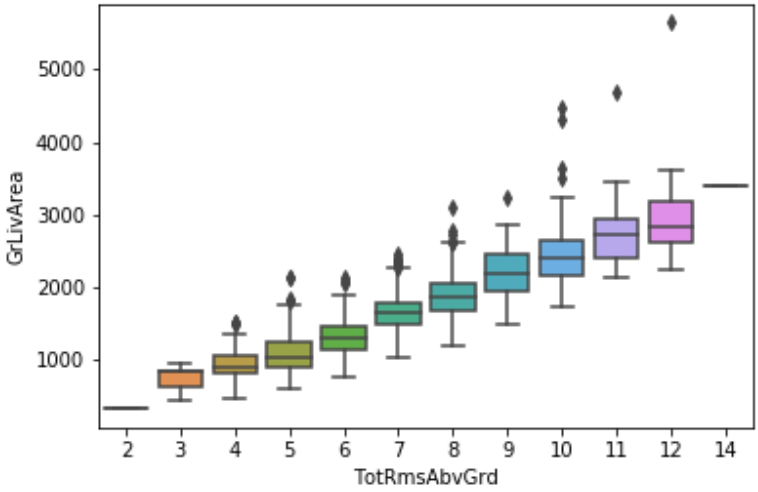


Figure 4.8: Scatter plot of TotRmsAbvGrd vs GrLivArea.

No other pair of *independent* features show strong correlation.
We further separate categorical and numerical features.

Categorical Features	
LotShape	OverallQual
OverallCond	LandContour
YearBuilt	Utilities
LotConfig	YearRemodAdd
GarageFinish	MoSold
GarageQual	YrSold
GarageCond	PavedDrive
PoolQC	Fence
MiscFeature	SaleType
SaleCondition	MSZoning
MSSubClass	Street
Alley	LandSlope
Neighborhood	Condition1
Condition2	BldgType
HouseStyle	RoofStyle
RoofMatl	Exterior1st
Exterior2nd	MasVnrType
ExterQual	ExterCond
Foundation	BsmtQual
BsmtCond	BsmtExposure
BsmtFinType1	BsmtFinType2
Heating	HeatingQC
CentralAir	Electrical
KitchenQual	Functional
FireplaceQu	GarageType
GarageYrBlt	

Numerical Features	
LotFrontage	LotArea
MasVnrArea	BsmtFinSF1
BsmtFinSF2	BsmtUnfSF
WoodDeckSF	1stFlrSF
2ndFlrSF	LowQualFinSF
GrLivArea	BsmtFullBath
BsmtHalfBath	FullBath
HalfBath	BedroomAbvGr
KitchenAbvGr	TotRmsAbvGrd
Fireplaces	GarageCars
OpenPorchSF	EnclosedPorch
3SsnPorch	ScreenPorch
PoolArea	MiscVal

4.2 Feature Engineering

4.2.1 Features with high linear correlation

We consider features with correlation coefficient of atleast 0.6 to SalePrice for further investigation.

OverallQual	0.790982
GrLivArea	0.708624
GarageCars	0.640409
TotalBsmtSF	0.613581
1stFlrSF	0.605852

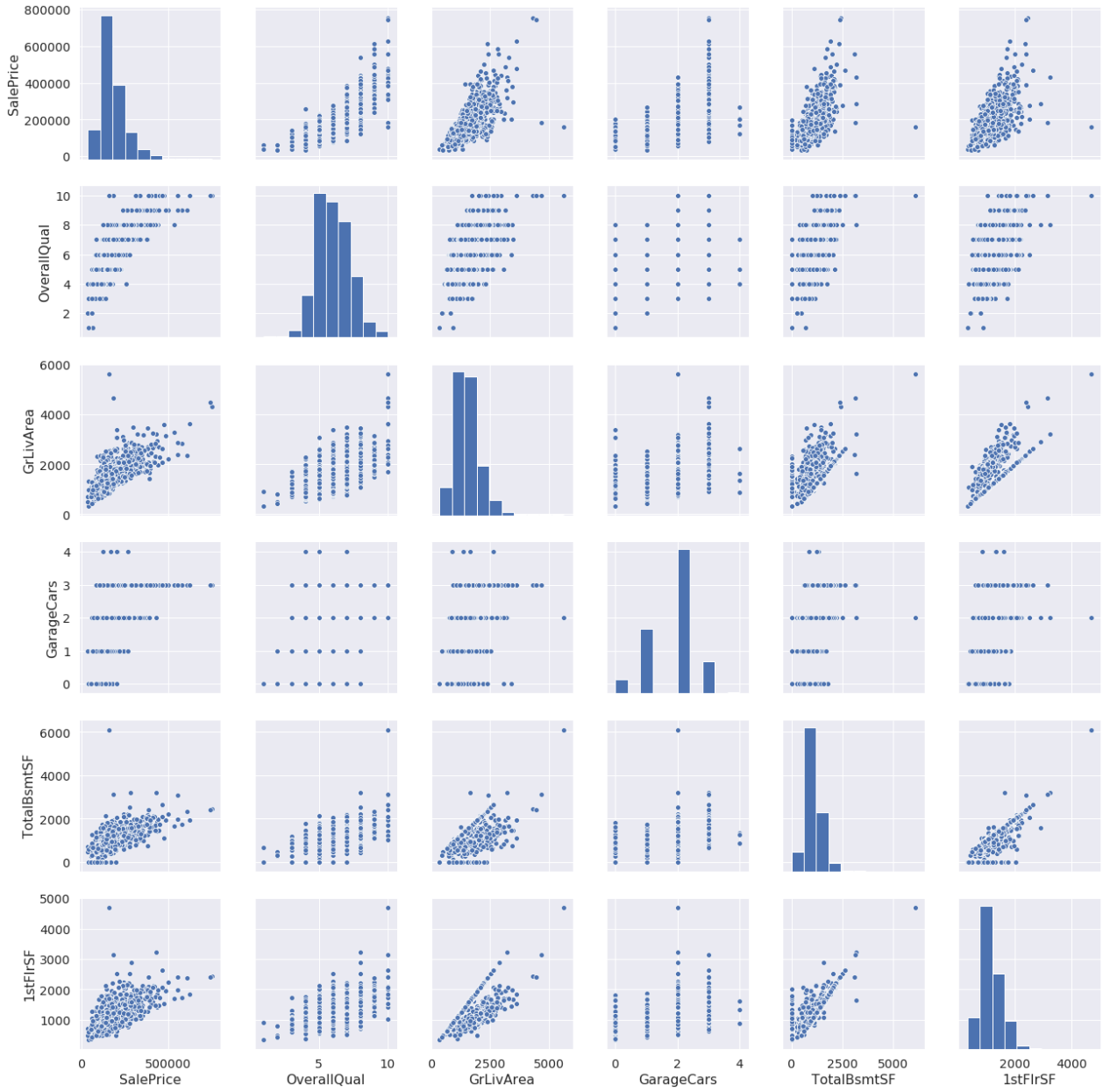
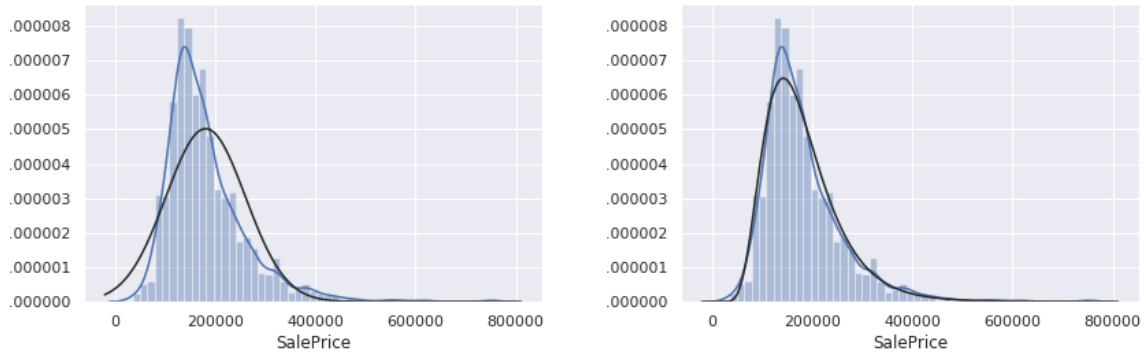


Figure 4.9: Scatter plots between SalePrice and selected features.

4.2.2 SalePrice

From SalePrice vs OverallQual plot, we see that bottom of dots and top of dots form an exponential boundary. TotalBsmtSF also shows a rough exponential relation with SalePrice. First, we need to make SalePrice closer to standard Gaussian distribution. SalePrice is almost like a standard logarithmic normal distribution. So we apply logarithmic transformation to SalePrice



(a) (b)

Figure 4.10: Sale price show close resemblance to logNorm curve.

to get it closer to normal distribution.

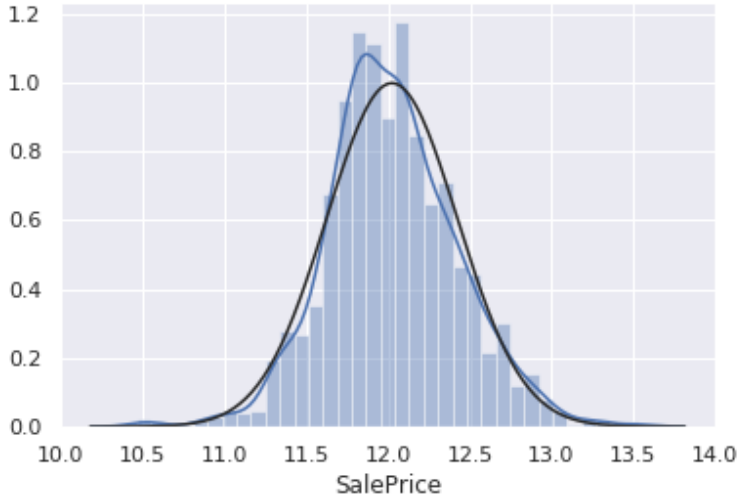


Figure 4.11: Scatter plots between SalePrice and selected features.

4.2.3 Missing Data

There are several features that have data missing. If we use this data as is, it will cause problems with learning algorithm.
We will have to fill these missing values manually.

The features with missing data are:

Feature name	Missing %	Feature name	Missing %
PoolQC	99.66	MasVnrArea	0.79
MiscFeature	96.40	MSZoning	0.14
Alley	93.22	BsmtFullBath	0.07
Fence	80.44	BsmtHalfBath	0.07
FireplaceQu	48.65	Functional	0.07
LotFrontage	16.65	Utilities	0.07
GarageCond	5.45	BsmtFinSF1	0.03
GarageFinish	5.45	BsmtFinSF2	0.03
GarageQual	5.45	BsmtUnfSF	0.03
GarageYrBlt	5.45	Electrical	0.03
GarageType	5.38	Exterior1st	0.03
BsmtCond	2.81	Exterior2nd	0.03
BsmtExposure	2.81	GarageCars	0.03
BsmtQual	2.77	KitchenQual	0.03
BsmtFinType2	2.74	SaleType	0.03
BsmtFinType1	2.71	TotalBsmtSF	0.03
MasVnrType	0.82	1stFlrSF	0.00

We have to manually fill missing data.

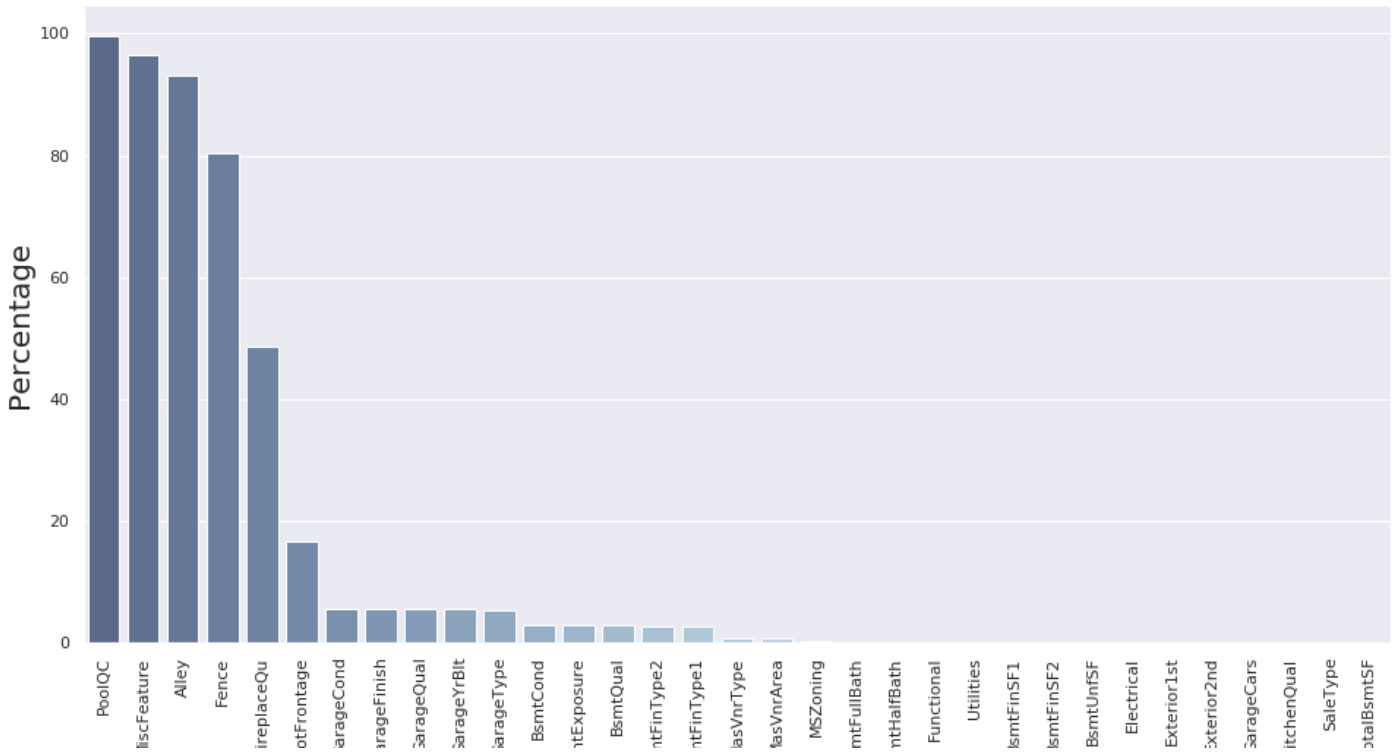


Figure 4.12: Null value percentages of given features.

We start with categorical features:

- *PoolQC: Pool quality*
  Here the missing values, i.e NA means No Pool. So we keep those values as None. Dropping PoolQC is also an option since more than 99% samples have missing data.

- *MiscFeature: Miscellaneous feature not covered in other categories*
  NA means no miscellaneous, so we keep it as None.

- *Alley: Type of alley access to property*
  NA means no alley access, so we keep it as None.

- *Fence: Fence quality*
  NA means no fence, so we keep it as None.

- *FireplaceQu: Fireplace quality*
  NA means no fireplace, so we keep it as None.

- *GarageCond, GarageQual, GarageFinish, GarageType*
  NA means no garage for all these features, so we keep it as None.

- *BsmtCond, BsmtExposure, BsmtQual, BsmtFinType1, BsmtFinType2*
  NA means no basement for all these features, so we keep it as None.

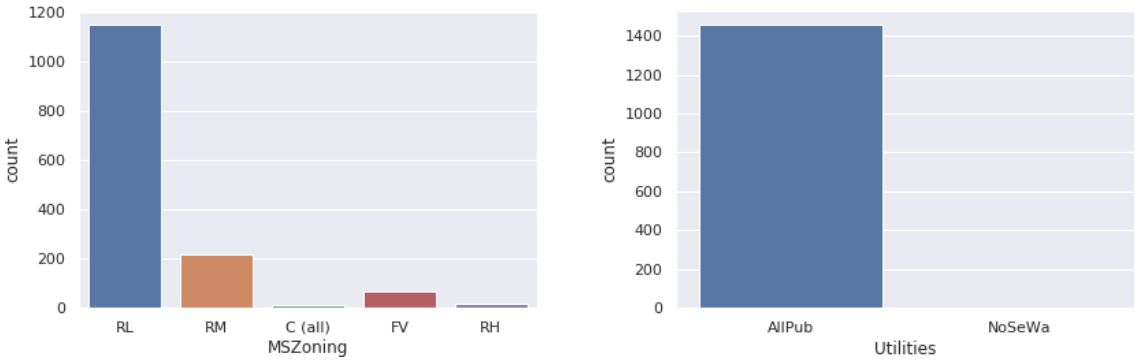
- *MasVnrType*
  We keep NA as None.

- *MSZoning*
  Here, NA means that data is actually missing. RL is most common value, so we replace all NA with RL.

- *Functional*
  According to data description, we assume NA as typical, i.e. Typ.

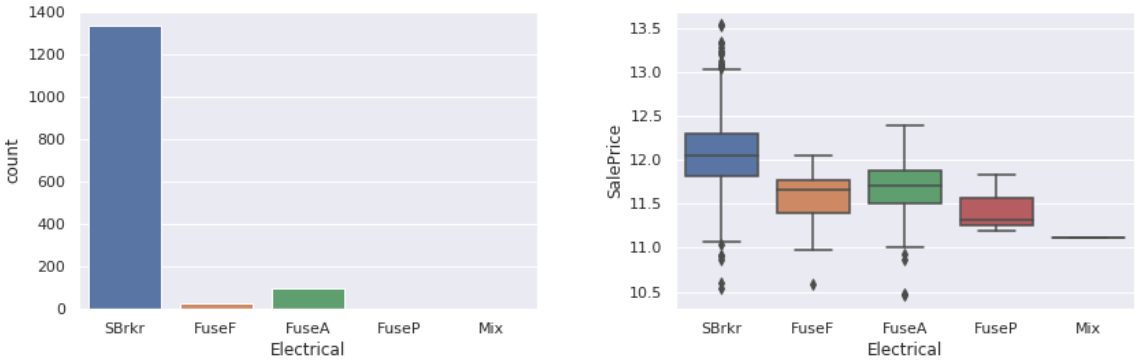
- *Utilities*
  We observe that the feature is useless since almost every sample is AllPub.
  So it is safe to drop the feature *Utilities*.

- *Electrical*
  SBrkr is most frequent value for Electrical, so it is safe to replace NA by SBrkr as NA is just 0.03% of values. There are too many outliers, so it is difficult to make some direct relation between SBrkr and SalePrice



(a) Distribution of MSZoning (b) Distribution of Utilities

Figure 4.13



(a) (b)

Figure 4.14: Analysis of Electrical.

- *SaleType*
  We replace NA with most frequent value WD.

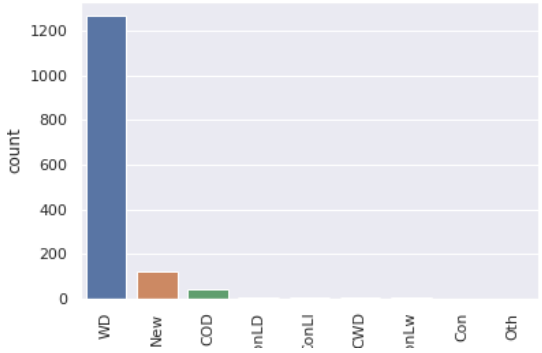
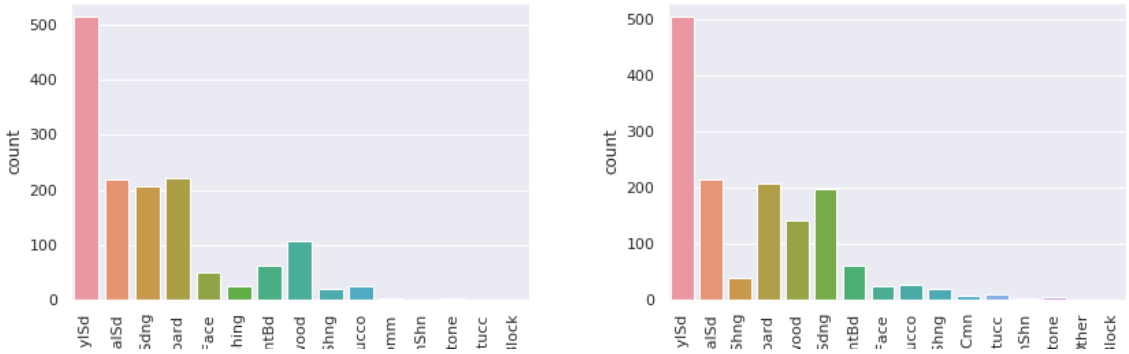


Figure 4.15: Distribution of SaleType.

- *KitchenQual*
  We replace NA with most frequent value TA.

Figure 4.16: Distribution of KitchenQual.

- *Exterior1st and Exterior2nd*
  We replace NA by most frequent value VinylSd.



(a) (b)

Figure 4.17: Analysis of Exterior1st and Exterior2nd.

Numerical Features:

- *GarageCars, BsmtFullBath, BsmtHalfBath*
  We replace NA by 0.

- *LotFrontage*
  LotFrontage refers to width of street connected to property. NA should be replaced by median of LotFrontage. Houses in same area ought to have similar LotFrontage value, so its better to replace missing values with median of houses in same Neighborhood.

- *MasVnrArea*
  This is masonry veneer area. NA likely means no masonry veneer for the houses. So we can fill 0.

- *BsmtFinSF1, BsmtFinSF2, TotalBsmtSF*
  NA can be taken to mean no basement, so we can fill NA with 0.

- *GarageYrBlt, GarageArea*
  NA can be taken to mean no garage, so we replace NA with 0.

4.2.4 Basement surface area variables

There are 4 independents that define basement surface area:

BsmtFinSF1: Type 1 finished square feet
BsmtFinSF2: Type 2 finished square feet
BsmtUnfSF: Unfinished square feet of basement area
TotalBsmtSF: Total square feet of basement area
Here, the feature TotalBsmtSF is redundant as it is just the sum of other 3 variables. It also gives less info compared to the other 3.

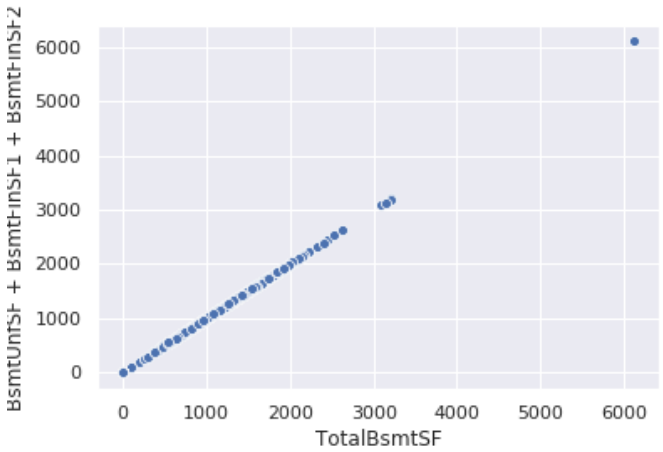


Figure 4.18: Scatter plot of TotalBsmtSF vs sum of other 3.

It is also confirmed from above plot that the relation is perfectly linear without any exceptions. So, we drop the variable TotalBsmtSF.

4.2.5 Normaziling skewed features

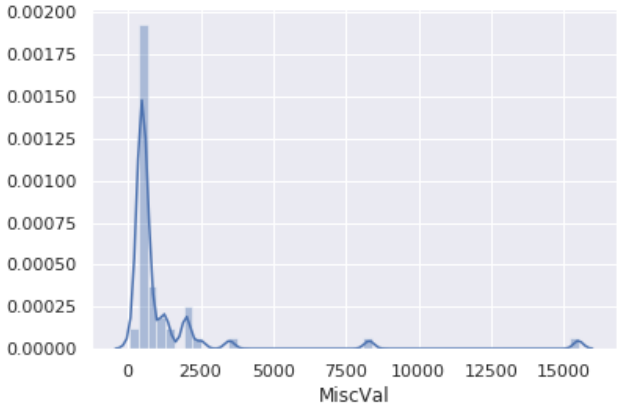
We need to normalize highly skewed features to achieve **heteroscedasticity**[2]. Such features are:

Feature	Skewness	Feature	Skewness
MiscVal	24.45	OpenPorchSF	2.36
PoolArea	14.81	LotFrontage	2.16
LotArea	12.20	BsmtFinSF1	1.68
3SsnPorch	10.29	WoodDeckSF	1.54
LowQualFinSF	9.00	1stFlrSF	1.38
KitchenAbvGr	4.48	GrLivArea	1.37
BsmtFinSF2	4.25	BsmtUnfSF	0.92
ScreenPorch	4.12	2ndFlrSF	0.81
BsmtHalfBath	4.10	HalfBath	0.68
EnclosedPorch	3.09		
MasVnrArea	2.67		

[2]https://en.wikipedia.org/wiki/Heteroscedasticity

- *MiscVal*
  MiscVal has extremely high skewness because most of its values are 0. More than 95% values are 0.

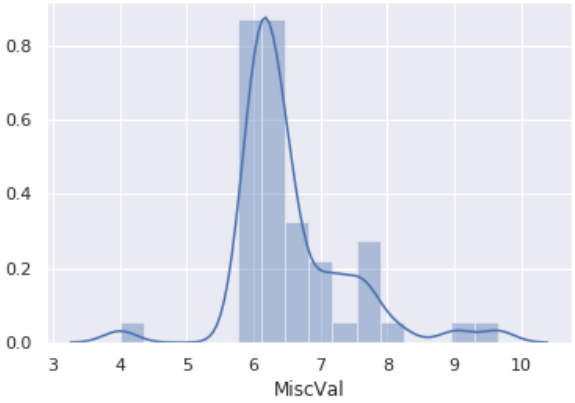
count	1460.000000
mean	43.489041
std	496.123024
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	15500.000000
skew	24.45



Distribution of MiscVal(ignoring zeroes)

  Applying a standard log transformation should bring other values closer to 0 and reduce skewness drastically.
  On applying *log plus 1* transformation, we get a skewness of 5.165:

count	1460.000000
mean	0.233456
std	1.226030
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	9.648660
skew	5.16



Distribution of transformed MiscVal(ignoring zeroes)

  If we ignore 0, the distribution is now appropriately normal.

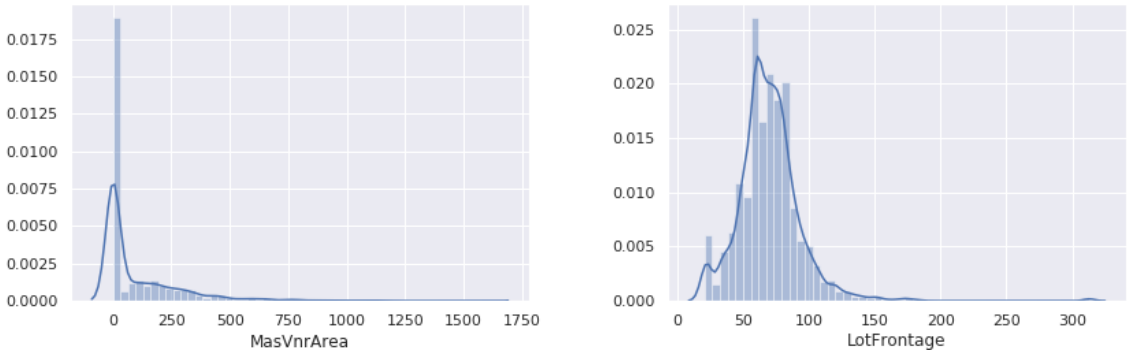
- *3SsnPorch*
  3SsnPorch has high skewness because 98% of its values are 0. Applying *log plus 1* transformation reduces its skewness to 7.72, and because of high amount of zeroes, its not possible to reduce skewness a lot.

- *LowQualFinSF*
  This feature has high skewness because 98.2% ofvalues are 0. We apply logarithmic transformation to get a skewness of 7.45.

- *KitchenAbvGr*
  KitchenAbvGr has high skewness because most values are 1. Applying logarithmic transform gives a skewness of 3.86.

- *MasVnrArea, OpenPorchSF*
  MasVnrArea is significantly improved by applying a logarithmic transformation giving

almost perfect Gaussian curve for non-zero set of values After transformation, we get a

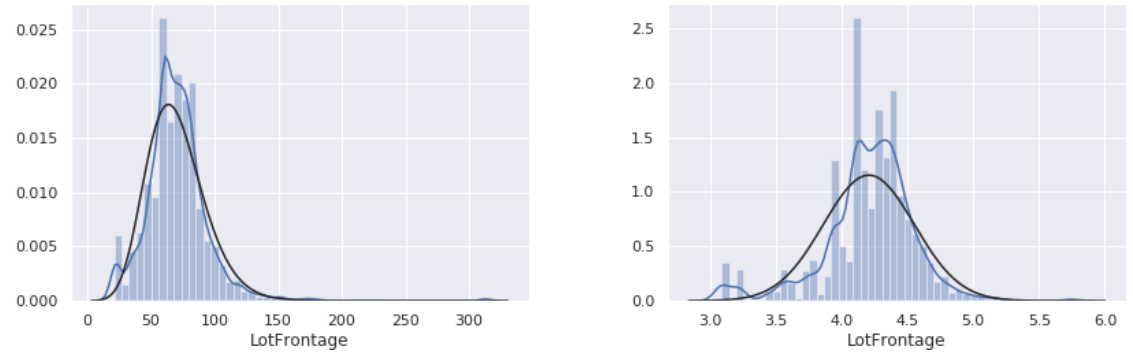


(a) Distribution of MasVnrArea (b) After applying logarithmic transformation.

Figure 4.19

skewness of 0.49.
OpenPorchSF behaves in similar way giving a skewness of -0.02

- *LotFrontage* LotFrontage is similar to lognorm distribution. Applying *log plus 1* transformation gives a -0.72.



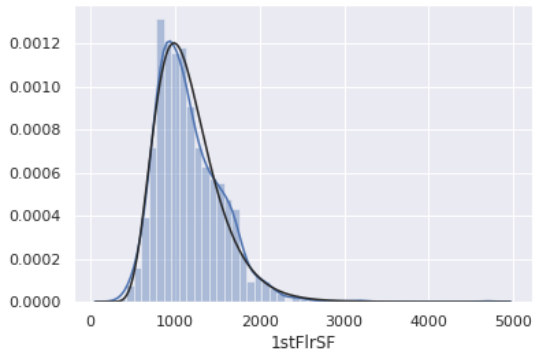
(a) Distribution of LotFrontage (b) After applying logarithmic transformation.

Figure 4.20

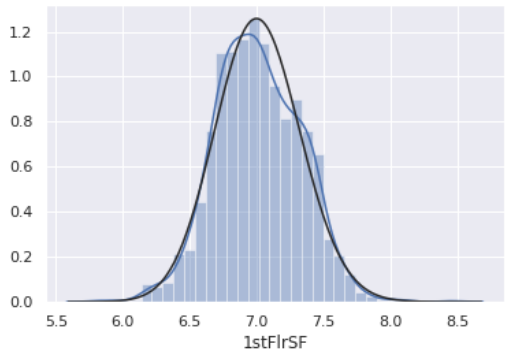
- *1stFlrSF* 1stFlrSF is almost exact lognorm curve with a skewness of 1.38. After applying log transformation, we get skewness of 0.08.

- *LotArea*
  LotArea can benefit a lot from logarithmic transformation as its shape is similar to that of lognorm distribution.
  On applying logarithmic transformation, we get a skewness of -0.13.

- *PoolArea*
  PoolArea, similar to MiscVal has most of its values as 0. Its skewness is 14.81

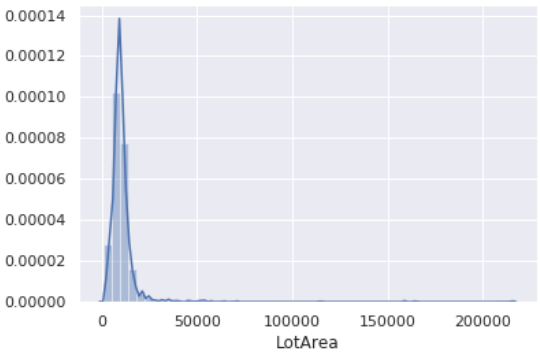


(a) Distribution of 1stFlrSF

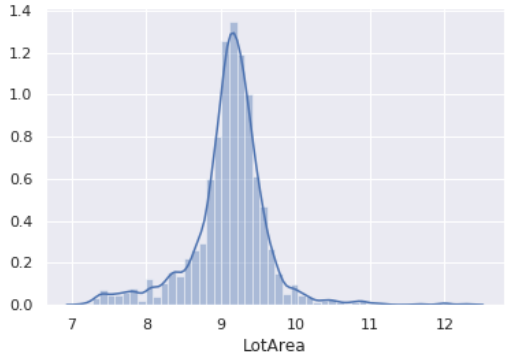


(b) After applying logarithmic transformation.

Figure 4.21



(a) Distribution of LotArea



(b) After applying logarithmic transformation.

Figure 4.22

We apply a *log plus 1* transformation to get a skewness of 14.31 so it doesn't have much effect. This is so because 99.5% values in it are 0.
The original distribution after ignoring zero was already almost normal to we choose to not apply transformation to it.

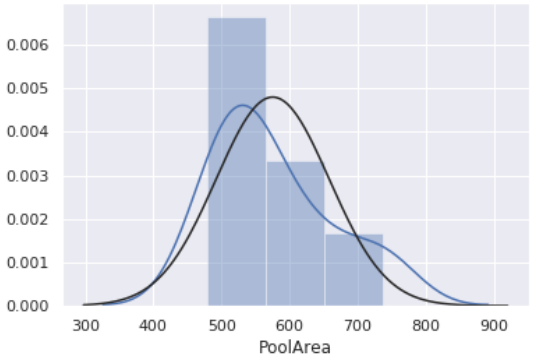


Figure 4.23: Distribution of PoolArea after removing zeroes.

- Other features have either high percentage of zero values, or show no special distribution.

Application of logarithmic transforma is able to decrease skewness of most variables so we choose to apply the transformation to them.

4.2.6 One-Hot Encoding

Final step of data pre-processing is to assign dummy variables to categorical features.
For this, we use one-hot encoding scheme[3]. It creates new binary columns for categorical values.
Pandas provides an inbuild function for this called *get_dummies*.

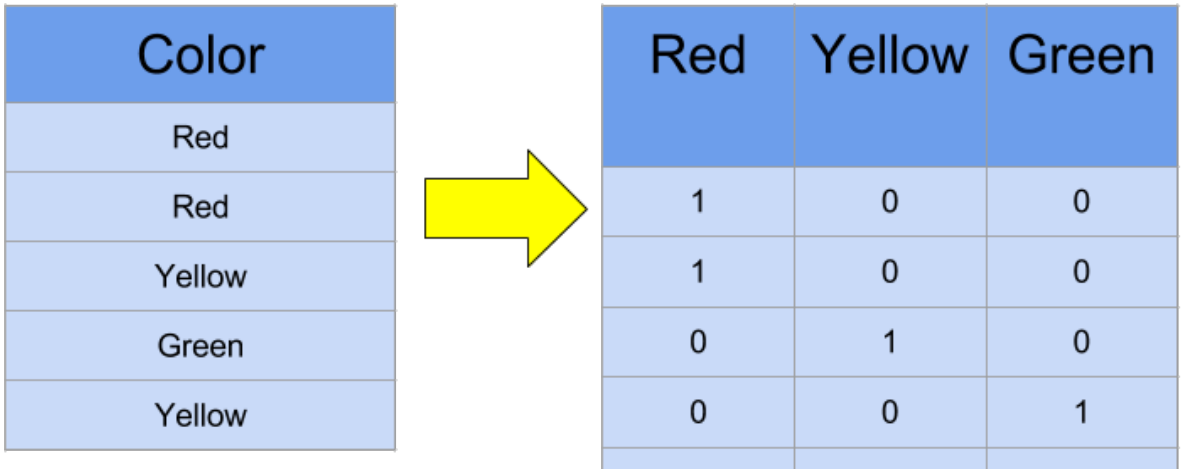


Figure 4.24: Illustration of one-hot encoding

[3]Reference : https://www.kaggle.com/dansbecker

4.3 Model Selection

We need to separate about 10% of training data before training our model to avoid overfitting. We call this data cross-validation set. Although general practice is to set aside more samples for cross-validation set, we keep 90% for training set so that the parameters obtained are as close as possible to final model. The data needs to be shuffled before separating it to get distributed data. This is needed to select appropriate value for the regression coefficient. First we do preliminary model comparison and then refine the selected model.

4.3.1 Ordinary Least Squares

This is the most basic model that minimizes the residual sum of squares between predicted dependent variables and actual dependent variables. Mathematically, it minimizes the cost function i.e.

$$\underset{\omega}{\text{minimize}}||X\omega - y||_2^2$$

ω is called coefficient vector, X is independent variable matrix and y is dependent variable vector.
The problem encountered with using simple linear regression is that it is highly prone to cause over-fitting, especially with high features and samples. This is also called high bias. Cross-validation set is used to test for this case. We will use basic **root mean square error** to compare models.
Errors for this model are :

With intercept calculation and no normalization:

Training dataset error	0.0833
Cross-validation dataset error	0.1341

Without intercept calculation:

Training dataset error	0.0833
Cross-validation dataset error	0.1355

There is not much distinction between the two settings because the several prominent features were tranformed to be centered. But intercept calculation still gives better result so we use intercept calculation in our models.

4.3.2 Ridge Regression (L2 regularization)

Regression imposes a penalty on cost function based on the size of coefficients. This causes coefficients to get too unbalanced. Ridge regression adds sum of square of coefficients. Being squared causes larger coefficients to be penalized more.
In ridge regression

$$\underset{\omega}{\text{minimize}}||X\omega - y||_2^2 + \alpha||\omega||_2^2$$

α is called l2 regression coefficient. Higher the value of α, higher the coefficients are penalized. Best method to figure out optimum value for α is to plot errors of training set and cross-validation set vs. different values of alpha. This plot is called learning curve of the model.

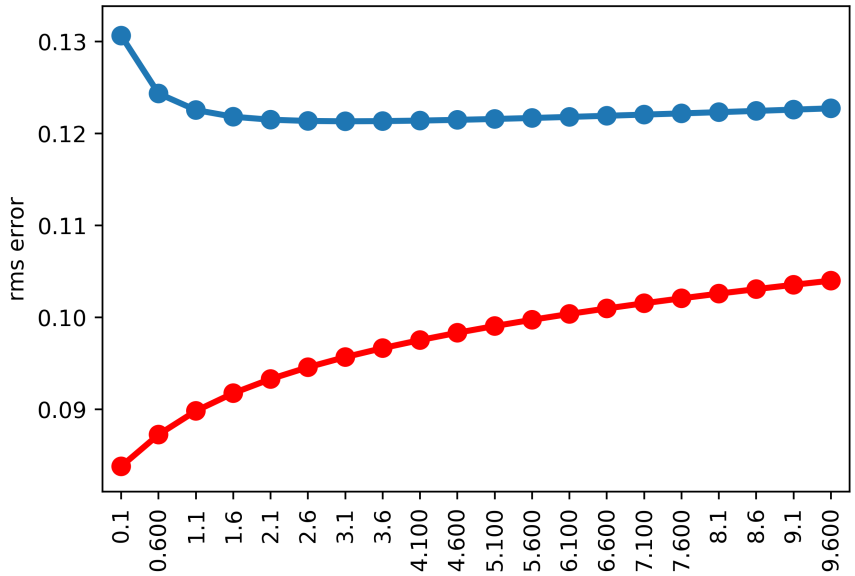


Figure 4.25: Learning curve for L2 regression

The red curve is for training-set error and blue curve is cross-validation error. The most optimum value for α is the point where error of cross-validation set is lowest. From the curve, the most optimum value for α is 3.1.

Error at $\alpha = 3.1$:

Training dataset error	0.0957
Cross-validation dataset error	0.1213

A learning curve of training data size vs error helps diagnose high bias and high variance. Figure 26 is the learning curve for ridge regression with $\alpha = 3.1$

Red curve is for training set error, and blue is for cross-validation set error. In learning curve of training sample size vs error, if a gap exists between two curves, the model is overfitting or has high variance.
And since the errors are decreasing with increasing training samples, only obvious way to improve results of L2 regression is by increasing no of training samples. Other than that, there is no obvious bias-variance imbalance.

4.3.3 Lasso Regression (L1 regularization)

Lasso model is good for eliminating sparse coefficients. It can be useful due to ints tendency to prefer solutions with fewer parameters, and focusing just on important features. The objective function to minimize is :

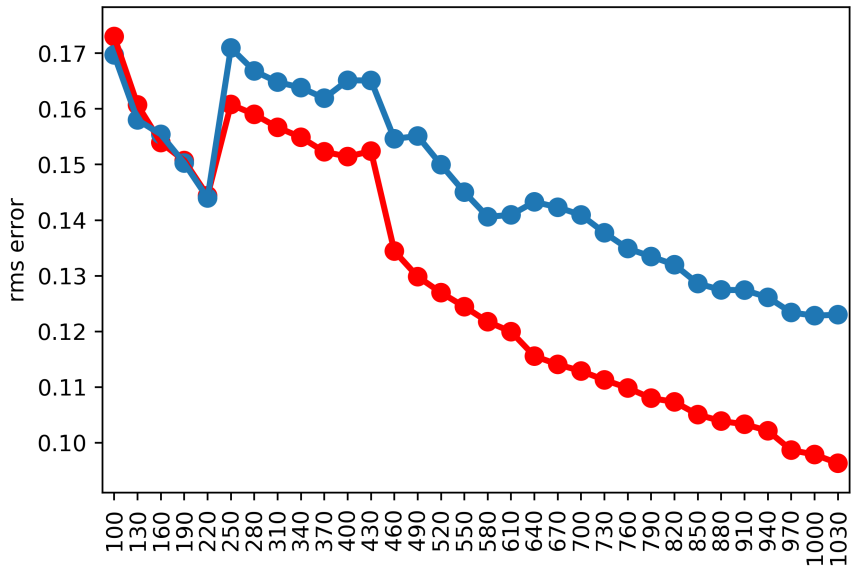


Figure 4.26: Learning curve for L2 regression

$$\underset{\omega}{\text{minimize}} ||X\omega - y||_2^2 + \alpha||\omega||_1$$

α is called L1 regression coefficient. While ridge regression takes square of parameter values, lasso regression only takes sum of absolute value of parameters. One advantage of using lasso regression is that it automatically selects the important features and eliminates the need of step regression. While L2 regression doesn't allow and weight to be zero, it is very likely that unimportant features will get zero weight in lasso regression. Figure 27 shows the learning curve of L1 regularization with alpha as experience.

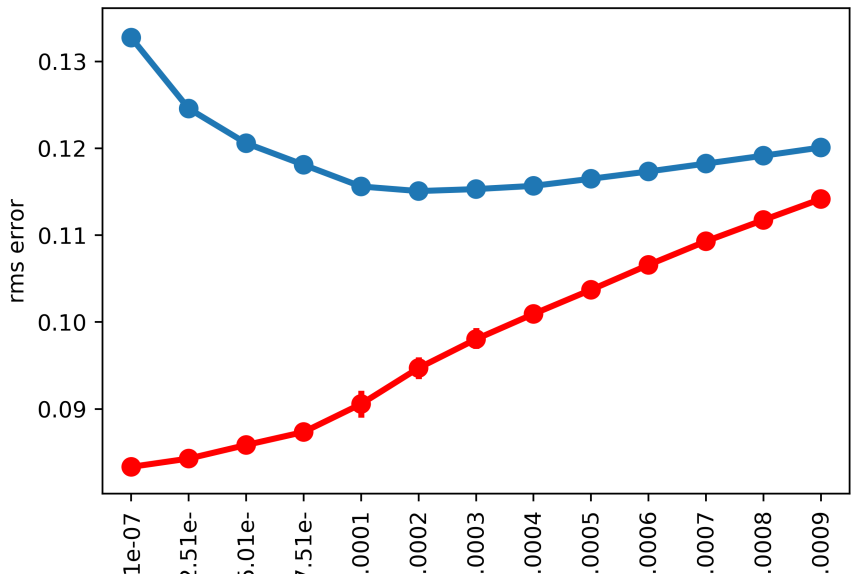


Figure 4.27: Learning curve for L1 regression

The red curve is for training-set error and blue curve is cross-validation error. The optimum α value here is 0.0002. At this value, error of cross-validation set in lowest.
Error at $\alpha = 0.0002$:

Training dataset error	0.0933
Cross-validation dataset error	0.1149

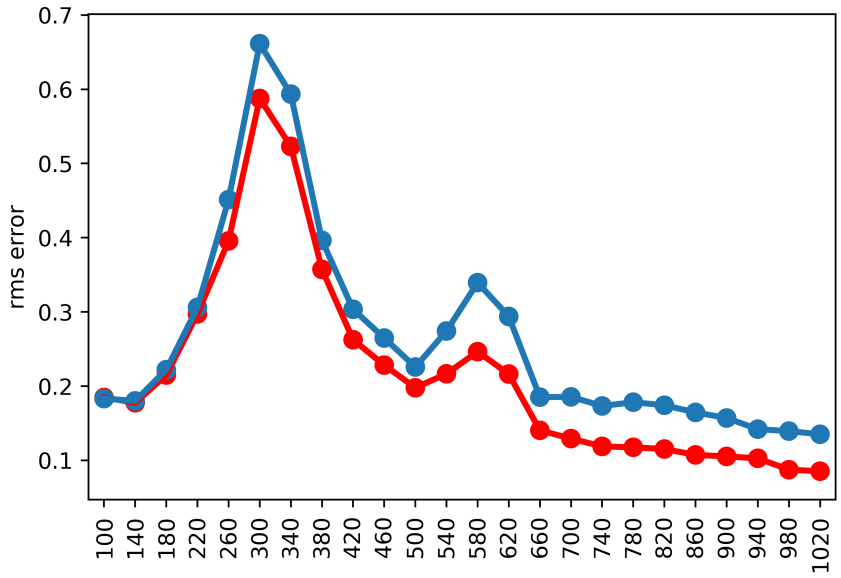


Figure 4.28: Learning curve for L1 regression

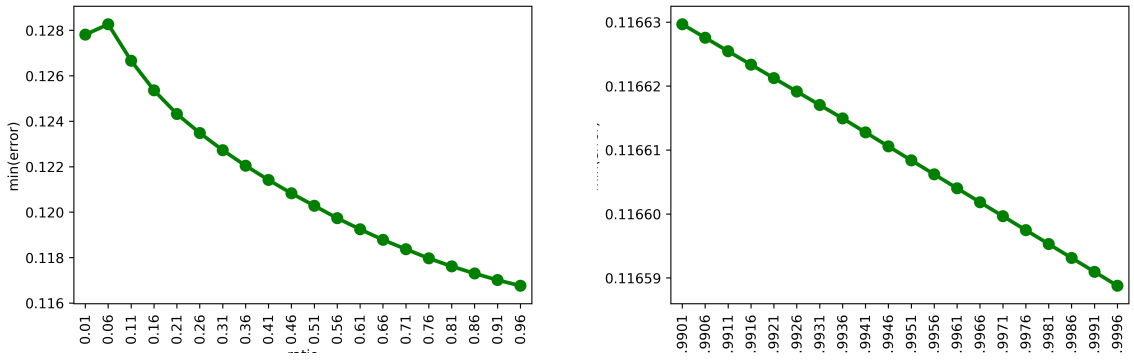
Figure 28 shows the curve of number of samples vs. respective errors. Red curve shows error of training set and blue curve shows error of cross-validation set. The learning curve shows a pretty nice shape and is evident of good balance between bias and variance with error decreasing with increasing training samples. The error should decrease after training with complete set.

4.3.4 Elastic Net Regression

Elastic net is a linear regression that uses both L1 and L2 regression. This model allows for learning a sparse model where few weights are zero like L1 regression, while controlling shrinkage of weights like L2 regression. The objective function to minimize is :

$$\underset{\omega}{\text{minimize}} ||X\omega - y||_2^2 + \alpha\rho||\omega||_1 + \frac{\alpha(1-\rho)}{2}||\omega||_2^2$$

α is called regression coefficient and ρ is called L1 ratio. ρ defines the ratio of L1 penalty to L2 penalty. If $\rho = 0$, penalty is completely L2, and for $\rho = 1$, penalty is L1 penalty.
For elastic net regression, first we will try different values to L1 ratio to get alpha learning curve, and select the one with lowest error for cross-validation set.



(a) Ratio = 0 to 1

(b) Ratio = 0.9 to 1

Figure 4.29: Plot of L1-ratio vs min(validation error)

Figure 29 shows the plot of L1-ratio vs. minimum achieved value of cross-validation error. The optimum value of L1-ratio is 1, so we don't need to use elastic-net regression.

4.4 Final Model and SQLite integration

After comparing different models, it is evident that Lasso regression performs much better. One reason for that is existence of sparse features. A simple for loop is used to select best value for α. We now divide the model into 3 parts :

- File 1 : Imports all data from SQL database and processes it. After that, it stores training data and test data separately in SQL database.
- File 2 : Imports only training data and trains the predictor. After training is finished, the predictor is pickled and stored as a binary file.
- File 3 : Imports target data from SQL database and loads the predictor from memory. After the prediction is done, it is stored in the SQL database.

Interface can be built as a wrapper around these 3 files. Current choice of interface is Jupyter Notebook.

Chapter 5

System Testing

The predictor has thoroughly tested against outliers.
The System has been made safe from unexpected input. Even if all fields are left empty, the system gives mode of SalePrice as a prediction.
The database system is also protected from crashes. If some unexpected data is received by the system, the python code raises an exception. Data is written to database once verified.
The import-export between Pandas and SQLite is flawless:

Figure 5.1: SQLite console

Chapter 6

Conclusion and Future Enhancements

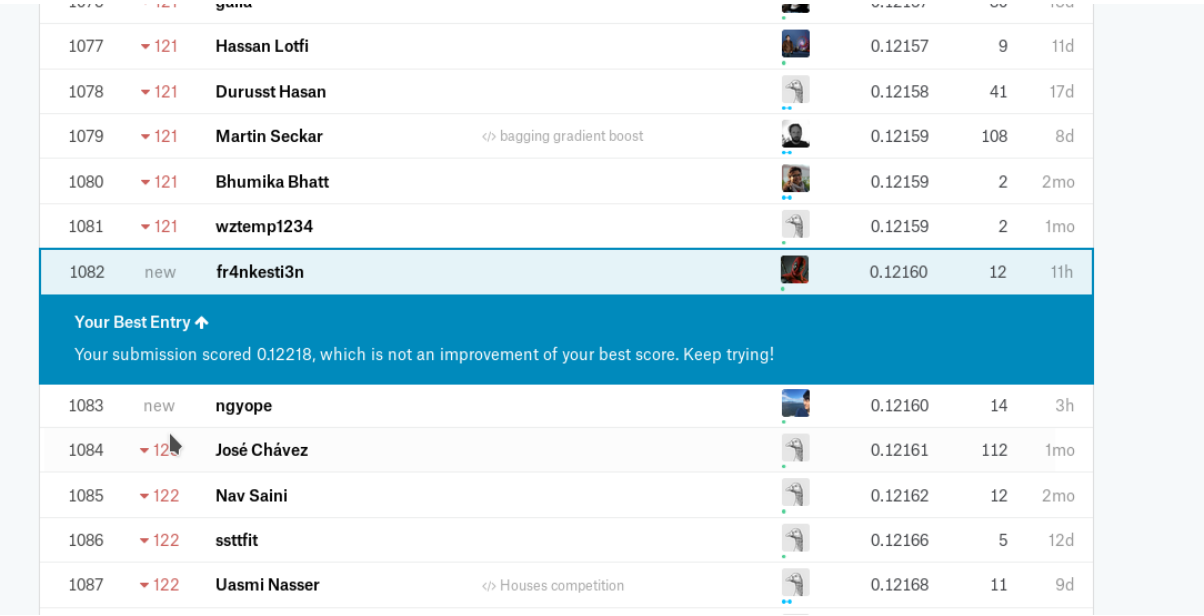
The model developed achieved position of 1077 on Kaggle leaderboard, i.e. 24.1% with Root mean square logarithmic error of 0.12218. The results were satisfactory proving the effectiveness of model.

The model can be further enhanced by improving upon the data analysis section. There are other functions other than logarithmic function that can be used for transforming variables. A study is needed into several possible function.
Improving learning model by using non linear methods is possible. Techniques like gradient boosting or Kernel tricks can be used to get lower error. Although neural networks need high amount of data to work with, a highly specialized system can improve the results.

Chapter 7

Results

The learning model performs well on Kaggle leaderboard. It achieved an error of 0.1216 which is in top 24% :



1077	▾ 121	Hassan Lotfi		0.12157	9	11d
1078	▾ 121	Durusst Hasan		0.12158	41	17d
1079	▾ 121	Martin Seckar	</> bagging gradient boost	0.12159	108	8d
1080	▾ 121	Bhumika Bhatt		0.12159	2	2mo
1081	▾ 121	wztemp1234		0.12159	2	1mo
1082	new	fr4nkesti3n		0.12160	12	11h

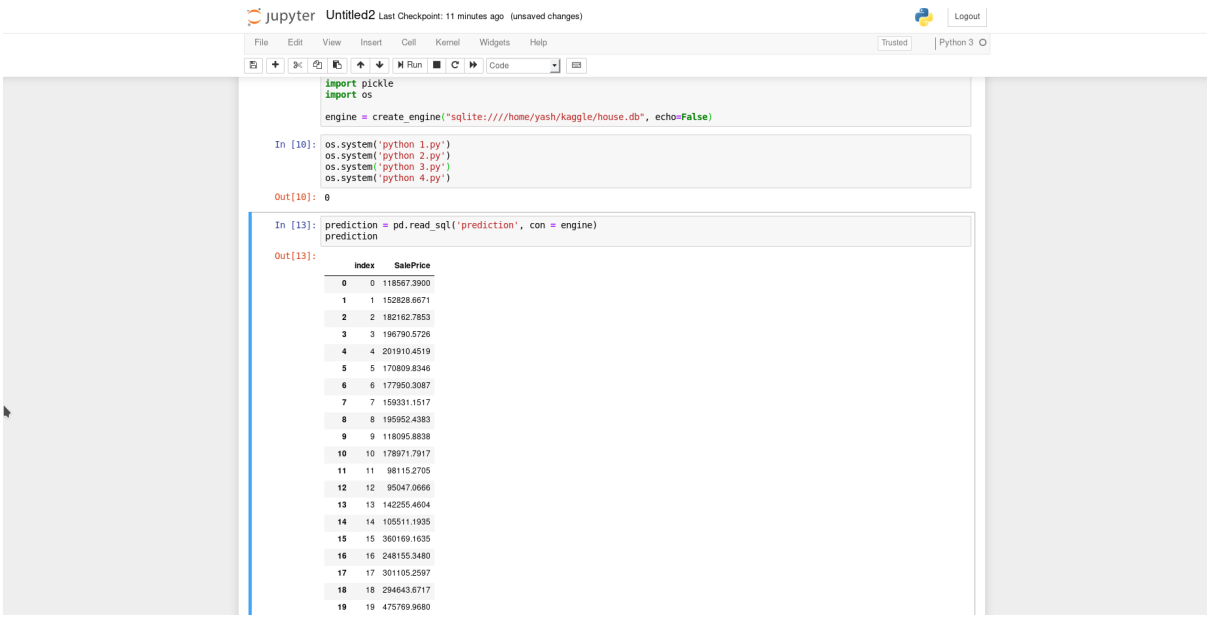
Your Best Entry ↑

Your submission scored 0.12218, which is not an improvement of your best score. Keep trying!

1083	new	ngyope		0.12160	14	3h
1084	▾ 12[illegible]	José Chávez		0.12161	112	1mo
1085	▾ 122	Nav Saini		0.12162	12	2mo
1086	▾ 122	ssttfit		0.12166	5	12d
1087	▾ 122	Uasmi Nasser	</> Houses competition	0.12168	11	9d

Figure 7.1: Kaggle Leaderboard

Jupyter notebook also works without any problems:



	index	SalePrice
0	0	118567.3900
1	1	152828.6671
2	2	182162.7853
3	3	196790.5726
4	4	201910.4519
5	5	170809.8346
6	6	177950.3087
7	7	159331.1517
8	8	195952.4383
9	9	118095.8838
10	10	178971.7917
11	11	98115.2705
12	12	95047.0666
13	13	142255.4604
14	14	105511.1935
15	15	360169.1635
16	16	248155.3480
17	17	301105.2597
18	18	294643.6717
19	19	475769.9680

Figure 7.2: Jupyter Notebook

References

- Kaggle House price dataset.
- Scikit-learn linear model documentation
- Kaggle blog : Gradient Boosting technique
- Analytics Vidhya : Parameter tuning guide to Gradient Boosting
- dansbecker's kernel on one-hot encoding
- Pedro Marcelino's kernel on data analysis
- Think Stats : Exploratory data analysis.