Table of Content

1. TASK A

1.1 Exploratory Data Analysis

Overall, the provided dataset contains 14 attributes. One of them is the response variable, the median value of the owner-occupied homes in Boston, and the remaining 1 binary and 12 continuous are predictors representing various environment and property information of the houses such as the crime rate by town.

1.1.1 Response Variable Visualization

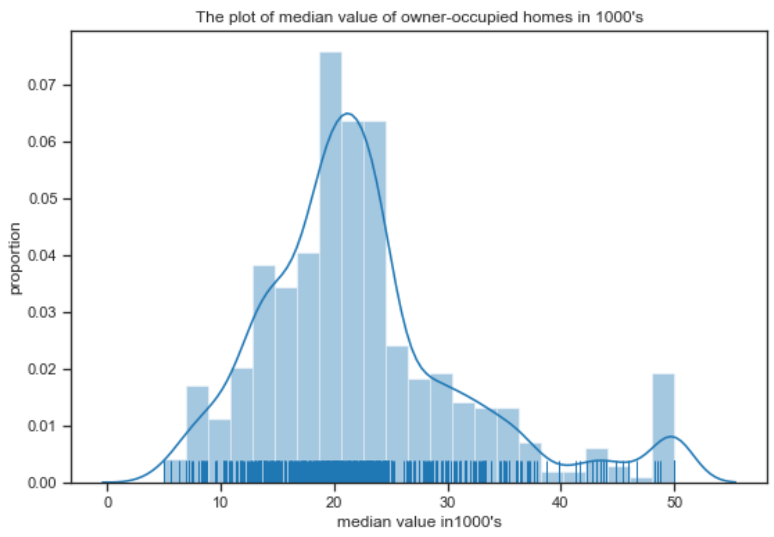


Figure 1: Distribution of response variable

Histogram is used to visualize the distribution of the dependent variable, the median house price (Figure 1). Majority of median prices of the houses are around $25,000. And it can be seen that the variable is right skewed, which means there are more houses with above-average prices among all samples.

1.1.2 Predictors Visualization

1.1.2.1 Binary Variable

The conditional distribution plot is used to shown the relationship between the median value of home price and the binary variable, the Charles river dummy variable. It indicates that the two distribution plots of the dependent variable approximately converge with each other, given the variable ‘chas’ equals 1 or 0, respectively. That shows that the value of the dummy variable which represents whether the house tract bounds the Charles river does not affect the house price significantly (Figure 2).

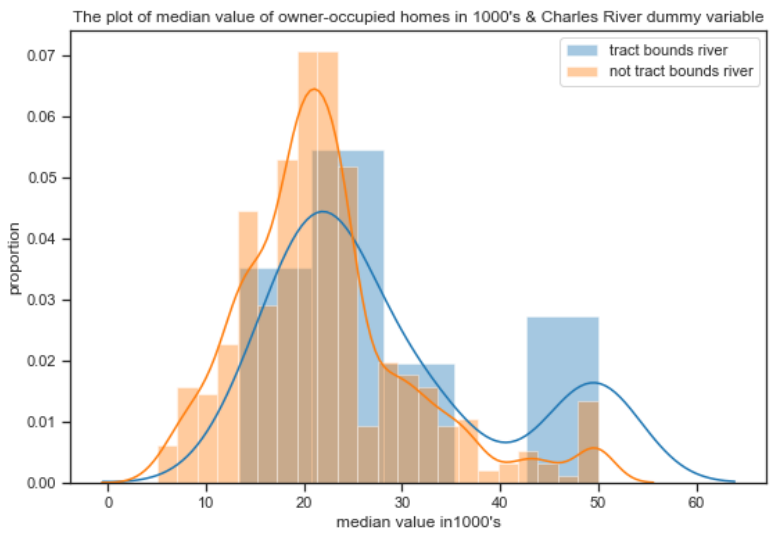


Figure 2: Conditional distribution of response variable

<u>1.1.2.2 Continuous Variable</u>

Above all, the correlations with the response variable are considered for the remaining 12 continuous variables. The 3 variables with the largest absolute correlation coefficient are ‘lstat’, ‘rm’, ‘ptratio’, which represent the proportion of lower status population of the town, the average room number and the pupil-teacher ratio by town. The result aligns with commence knowledge that the dwelling price always varies with the social class of the suburb, the size of the house and education resources significantly.

The correlation between all predictors with the response variable is visualized with the scatter plot with the linear regression line (Figure 1 in Appendix A). And the correlation between all the attributes including predictors and the dependent variable is shown in the heatmap (Figure 3).

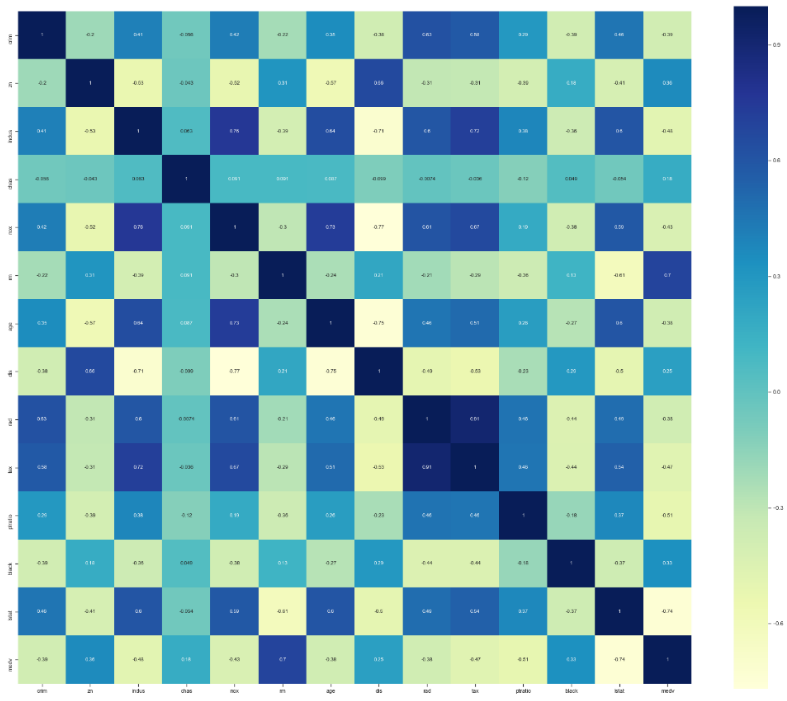


Figure 3: Heatmap

To build an effective linear regression model to predict the house price, features with high correlations with price should be considered. So that the 3 most important features should be ‘lstat’, ‘rm’, ‘ptratio’.

1.2 Gradient Descent

1.2.1 Data Processing and Standardization

According to the requirement, three features including ‘rm’, ’dis’, ’tax’ denoting the room number, weighted distances to five Boston employment centers and the full-value property-tax rate per 10,000, respectively, are chosen to do the linear regression. It is shown that these three features are different from each other not only in the data type but also in the data size. For instance, the mean values of ‘rm’ and ‘tax’ are 6.2846 and 408.2372. To improve the predictability of the linear regression, these three features are linearly transformed with standardization to make their average values go to 0 and standard variation become 1.

1.2.2 Methodology

The gradient descent method is to update the estimates of the parameters of the linear regression following the below principles until the loss function $L(\beta)$ become as close as possible to its local minimum value.

$$L(\beta) = \frac{1}{2N}\sum_{n=1}^{N}(X\beta - y)^T(X\beta - y)$$

$$\beta_0 = [0 \quad 0 \quad 0 \quad 0]^T$$

$$\beta_{t+1} = \beta_t - \lambda\,\frac{\partial\, L(\beta_t)}{\partial\, \beta_t} \qquad t\text{: } the\ time\ of\ iteration$$

In this case, the stopping criteria of the iteration is set to be that the iteration number reaches 10,000 or the change of loss function after a step is less than 0.00001 (Code 1 in Appendix A).

1.2.3 Optimal Learning Rate Selection

The learning rate λ controls the spend of the gradient descent in a reasonable range. If the learning rate is too large, the loss function could never reach its local minimum and if the learning rate is too small, the iteration process would be ineffective and need more iterations to find the local minimum value of loss function. The best learning rate to be select should be the one that minimizes the loss function, which means the linear regression prediction with the corresponding estimated parameter should have the least MSE.

The cross validation with 5 folds method is used to tune the optimal learning rate. The dataset is split into 5 folds. Each time 4 of them is used to train the linear regression and the remaining validation set is predicted to get the MSE score. For each candidate learning rate, the average MSE score of the five validation folds in the 5 iterations, respectively, is calculated and compared with each other. The learning rate with the least MSE score is selected to be the optimal learning rate. In this case, the select learning rate is 0.0010. The estimated parameter of the linear regression model is $[0 \quad 0.6137 \quad -0.0417 \quad -0.3115]^T$, the corresponding MSE is 0.4381. Consider the mean value of the response variable is 22.5328, the result of the learning rate is reasonable.

2. TASK B

2.1 Introduction

The purpose of this study is to construct model using machine learning techniques for predicting house sale price. For potential buyers, developers and other participants in real east market, it is important to have an accurate prediction on the price of house (Frew, 2003). With a house price prediction model, the information gap in the real estate market can be filled and improve the efficiency of market (Calhoun, 2003). Hence, as a real estate analytics company, build up a powerful model which can accurately predict price the house can effectively satisfy customers' needs.

With a provided dataset which contains 80 features of house might affect its sales price, 5 models are constructed in this study for analyse the relationship between features and sales price. Errors of model were measured and compared, which will be explain in detail in later section of this report. With the lowest error of prediction compare to houses real sale price, model 'Stacking' and 'XGBoost' are chose as two best model for predicting house sale price.

2.2 Data Processing and Exploratory Data Analysis

2.2.1 Data Processing

2.2.1.1 Data Understanding

The provided dataset stores comprehensive information on house price. This comma-separated values (CSV) file has been divided into training and testing set, which contains 1,570 and 1,210 observations respectively. Both training and testing set have 80 independent variables, while 'SalePrice' is only provided in training set as a dependent variable for supervised learning process.

It can be seen from Table 1 in Appendix B that the number of variables in training set with dtype as object, integer and float is 43, 29 and 9 respectively. The raw dataset is not ideal for statistical analysis with a large number of object dtype columns, thus, data processing is needed to convert original one into structured format for further analysis.

2.2.1.2 Missing Values

As shown in Table 1 in Appendix B, 24 variables in training set contain missing values (See Figure 2 in Appendix B), 9 of them are numerical data, others are object. 'Pool QC' is the variable with the most missing values, followed by 'Misc Feature', 'Alley' and 'Fence' with more than 1,000 missing values.

To fill the missing values, the variables are grouped together based on their dtype. Group 1 is for numerical variables, which contains variables with dtype integer and float, while Group 2 includes variables with dtype object. For numerical variables, missing values are filled with 0; while for categorical variables, missing values are filled with the most frequent value (See Table 1 in Appendix B). Same method is applied to both training and testing set.

2.2.2 Exploratory Data Analysis

2.2.2.1 Descriptive Summary

Descriptive summary is used to understand the statistic of the variables. 32 variables including the response variable 'SalePrice' have positive skewness, which will be applied log transformation in feature engineering phase.

2.2.2.2 Response Variable Visualization

The response variable ‘SalePrice’ is ranging from 13,100 to 755,000, with the mean of 183,176.0522. As shown in Figure 4, the distribution of ‘SalePrice’ is right skewed, which means data transformation should be performed in feature engineering.

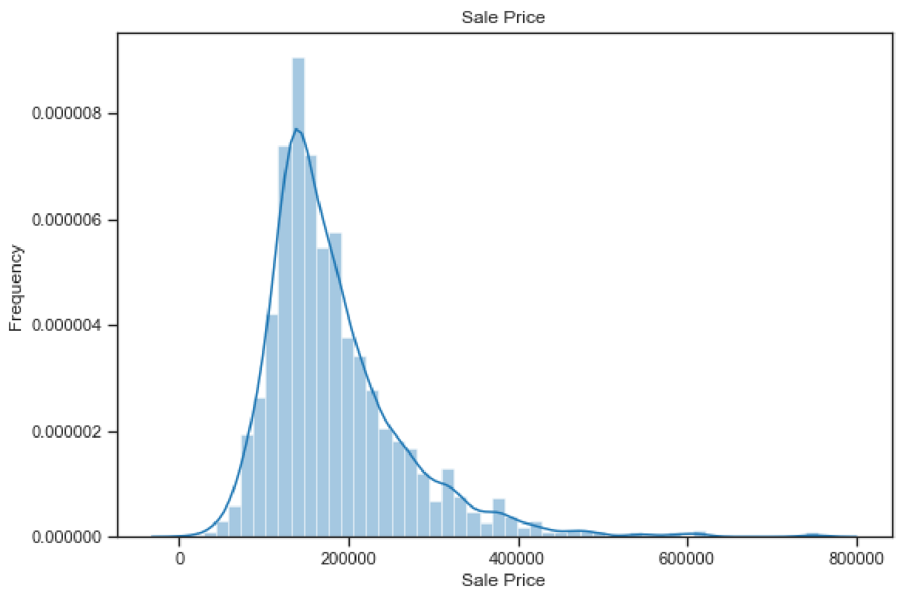


Figure 4: Distribution of response variable

<u>2.2.2.3 Predictors Visualization</u>

Histogram is used to visualize the distribution of each numerical feature, shown in Figure 3 in Appendix B. It is obvious that right skewness exists in predictors such as ‘Lot Area’, ‘Bsmf Unf SF and ‘Gr Liv Area’, therefore, data transformation should be performed in feature engineering phase to solve this problem.

Scatter plot is conducted to visualize the relationships between all numerical predictors and response variable ‘SalePrice’, shown in Figure 4 in Appendix B. Visually, predictors like ‘Total Bsmt SF, ‘Gr Liv Area’ and ‘Garage Area’ have strong positive effects on price, no obvious patterns shown in other plots.

Heatmap is used to illustrate the correlation among predictors, shown in Figure 5. ‘Overall Qual’ has the highest correlation with ‘SalePrice’, which is 0.7977, followed by ‘Gr Liv Area’, ‘Total Bsmt SF’ and ‘Garage Cars’. There are 8 predictors having less than 0.05 correlation with ‘SalePrice’, which can be considered removed in feature engineering phase. Among the predictors, ‘Garage Car’ and ‘Garage Area’ have a strong positive correlation of 0.9, which needs to be death with since it would cause multicollinearity problem.

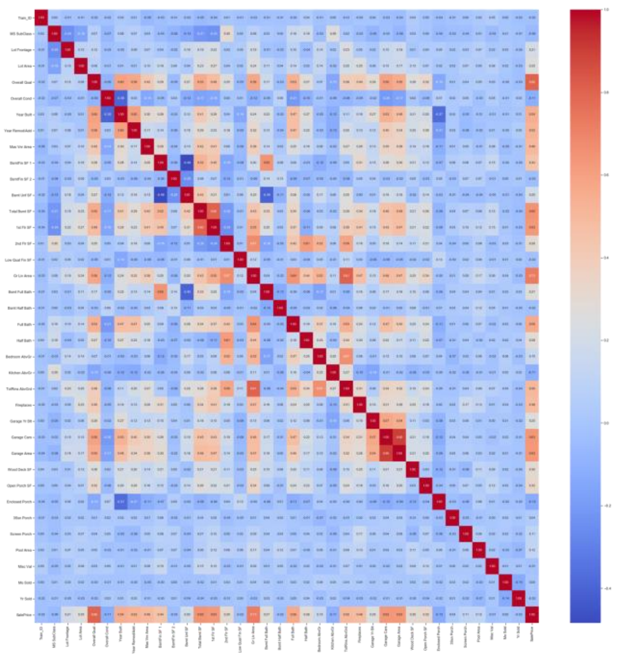


Figure 5: Heatmap

Boxplot is conducted to show the effect of binary variables on the response variable. As shown in Figure 6, pavel alley, pavel street and with central air would result in higher house price.

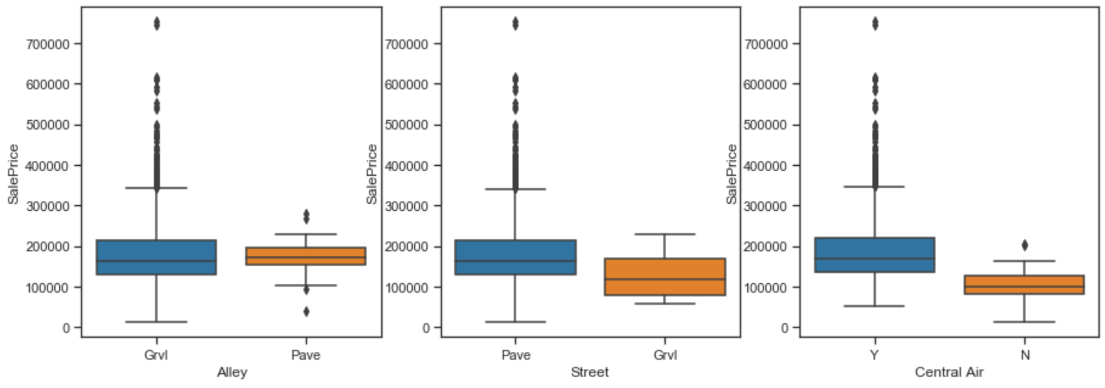


Figure 6: Boxplot of binary variables

2.3 Feature Engineering

After previous section, 80 features in training set will be used to fit model, while 43 of them are object and 37 of them are numerical. Feature engineering is used to extract features from raw data to improve data understanding and performance of model prediction. In this study, training set and testing set will be applied with same methods of feature engineering, while there is no special instruction.

2.3.1 Ordinal and Dummy Features

To fit data with model, ordinal and dummy features need to be encoded into numerical. Totally there are 27 ordinal variables need to be encoded as listed in Table 1 in Appendix B. For example, ordinal variable ‘Exter Cond’ have five categories including ‘Ex’, ‘Gd’, ‘TA’, ‘Fa’ and ‘Po’, in order to fit ‘Exter Cond’ with model, five categories are encoded into ‘4’, ‘3’, ‘2’, ‘1’ and ‘0’. In addition, 3 dummy variables also need to proceed with encoding. For dummy variable ‘Central Air’, ‘0’ and ‘1’ are used to encode category ‘N’ and ‘Y’ respectively. Details of variable encoding are listed in Table 1 in Appendix B.

2.3.2 Categorical Features

In the dataset, 13 variables are considered as categorical features, which needed to be transformed into numerical using dummy encoding. For n categories, n-1 dummy variable will be constructed to fit model. The first step to encode categorical features is reducing categories. To avoid overfitting problem and save computational cost resulting from by too many features, the number of created dummy variables from categorical features should be as little as possible. Categorical feature ‘Sale Condition’ will be taken as an example. The ‘Sale Condition’ have 6 condition including ‘Normal’, ‘Partial’, ‘Family’, ‘Alloca’, ‘AdjLand’ and ‘Abnormal’, condition ‘Normal’ has highest frequency of 1,293, while other conditions sum up to have a frequency of 277 in training set. As ‘Normal’ condition takes more than 80 percent of entries in training set, the categories of variable ‘Sale Condition’ are reduce to ‘Normal’ and a new generated type ‘Other’. ‘Other’ is the type of condition including data entries of ‘Partial’, ‘Family’, ‘Alloca’, ‘AdjLand’ and ‘Abnormal’. Details of category reducing process of all categorical features are listed in Table 1 in Appendix B. After reduce category, new dummy variables are created from updated categorical features and used to fit model.

2.3.3 Log Transformation

As normal distributed data are more friendly for constructing model, the log transformation is used to transformed skewed data to be normally distributed. As mentioned in EDA, 32 variables with skewness greater than 0 including target variable ‘SalePrice’ are applied with log transformation. List of log transformed variable are listed in Table 1 in Appendix B.

2.4 Methodology

2.4.1 Methodology Description

2.4.1.1 Decision Tree

Decision tree is a non-parametric model which includes classification tree and regression tree. In this assignment, regression tree is used to predict the house price. Decision tree classifies a population into branch-like segments that construct an inverted tree with a root node, internal nodes, and leaf nodes. The algorithm is non-parametric and can efficiently deal with large, complicated dataset without imposing a complicated parametric structure.

Decision tree performs well even if its assumptions are somewhat violated by the true model from which the data were generated. However, decision trees can be unstable because small variations in the data might result in a completely different tree being generated.

2.4.1.2 Random Forest

Random Forest is composed of multiple decision trees and there is no correlation between each tree. Instead of using bootstrap method, Random Forest uses sampling without replacement method to ensure randomness. In addition, each node is segmented using the best values from the randomly selected subset of predictors on that node (Breiman, 2001). The randomness reduces the variance at the cost of increasing of bias.

Random Forest is easy to implement and can generate relatively accurate prediction. However, overfitting may occur in some noisy classification or regression problems.

2.4.1.3 XGBoost

XGBoost is short for ‘Extreme Gradient Boosting’ which applies the adoption of a gradient descent structure to enhance the learning ability of weak learners. It implements machine learning algorithms under the Gradient Boosting framework and improves the basic GBM framework through system optimization and algorithm enhancement. XGboost adds regular terms to the cost function to control the complexity of the model. From the perspective of Bias-Variance Trade-off, the realization reduces variance of the model, and prevents overfitting.

XGBoost is an accurate and flexible method to make predictions. It can also deal with samples with missing values by using the sparse perception algorithm and automatically learning its splitting direction. However, it’s not easy to optimize due to the large number of turning parameters. In addition, the predicting process is considered as ‘black-box’ which means it’s hard to interpret the result.

2.4.1.4 AdaBoosting

Full name of AdaBoosting is Adaptive Boosting. It uses linear combination of weak classifier to construct strong classifier. Samples and classifiers have different ways of allocating the weights. At the beginning of training, the weights of all samples are equal, and the weight of samples misclassified by the previous weak classifier will be increased in the next turn. Different from the weights of samples, the weak classifier with high accuracy will have a larger weight.

AdaBoosting has relatively accurate result and it can avoid the overfitting problems. However, this model it’s sensitive to outliers. As the training samples are weighted, higher weights may be given to the outliers in the training data which will affect the accuracy.

2.4.2.5 Staking

In stacking method, there are normally 2 layers. In the first layer, various strong classifiers can be used to make predictions. At the second layer, a weak classifier is used to form a new set of prediction based on the results generated in the first layer. Models used in the first layer are called ‘base model’ and model in the second layer is ‘meta model’.

2.4.3 Model Implementation

Scikit-learn in Python is used to implement above 5 models. All features are used in the first training process, and to avoid the overfitting problems, feather engineering is re-done according to the feature importance.

Since most advanced models are hard to interpret and there are inconsistency issues when using the feature_importance attribute in advanced models, a new method which is SHAP is introduced to calculate the importance of features and interpret the model in this assignment. SHAP uses the method of game theory, and the value obtained by this method is the average marginal contribution of eigenvalues in all possible alliances. It can also solve ‘black-box’ issue and interpret the model clearly.

2.4.3.1 XGBooost

Scikit-learn has XGBoost model which is used in this assignment. Tuning parameters are set as below to fine the best ones:

```
tuning_parameters = {
    'learning_rate': [0.01, 0.05, 0.1],
    'gamma' : [0.1,0.2],
    'n_estimators' : [0,500],
    'max_depth' : [5,6],
    'subsample' : [0.5,0.6,0.8,0.9, 1.0],
}
```

After model training, it can be found that the best parameters are: learning rate = 0.05, subsample = 0.6, n_estimators = 500 and the max depth = 5. The RMSE is 0.1304.

2.4.3.2 Stacking

In Stacking model, 4 advanced models which are Decision Tree, Random Forest, XGBoost and AdaBoost are used as the base model and Lasso is used as the meta model since it can avoid overfitting problem.

2.5 Model Validation and Evaluation

2.5.1 Model Validation

To find the best parameters, GridSearch is used in this assignment which loops through all the candidate parameters, and try every possibility to find the best parameters

5-folds validation is used to validate the model. The data was divided into 5 subsets. Each subset is tested once and the rest are used as training sets. Cross validation is repeated 5 times, and the average recognition accuracy of cross validation 5 times is taken as the result.

2.5.2 Measure Metric

The Root-Mean-Squared-Error (RMSE) is used to evaluate the performance of models. Formula of RMSE is $RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n}}$. The RMSE results of each model is shown in Table 1.

Table 1: Model performance and kaggle result

Model	Decision Tree	Random Forest	XGBoost	AdaBoost	Stacking
RMSE	0.1730	0.1485	0.1304	0.1649	0.0537
Kaggle Result	36563.2140	27060.1708	20329.3315	28915.6272	22827.6266

According to the table above, it’s clear that Stacking model has the lowest RMSE score which is only 0.0537. The rest of models have relatively similar RMSE and among them, XGBoost performs the best while Decision Tree is relatively less accurate. Thus, XGBoost and Stacking are selected as the final two model.

2.5.3 Model Interpretation and Insights

Mean SHAP value sort absolute values of the feature importance in descending order. From Figure 7, it can be seen that Decisions Tree, Random Forest and XGBoost indicates that there are 2 types of features play important role in house pricing.

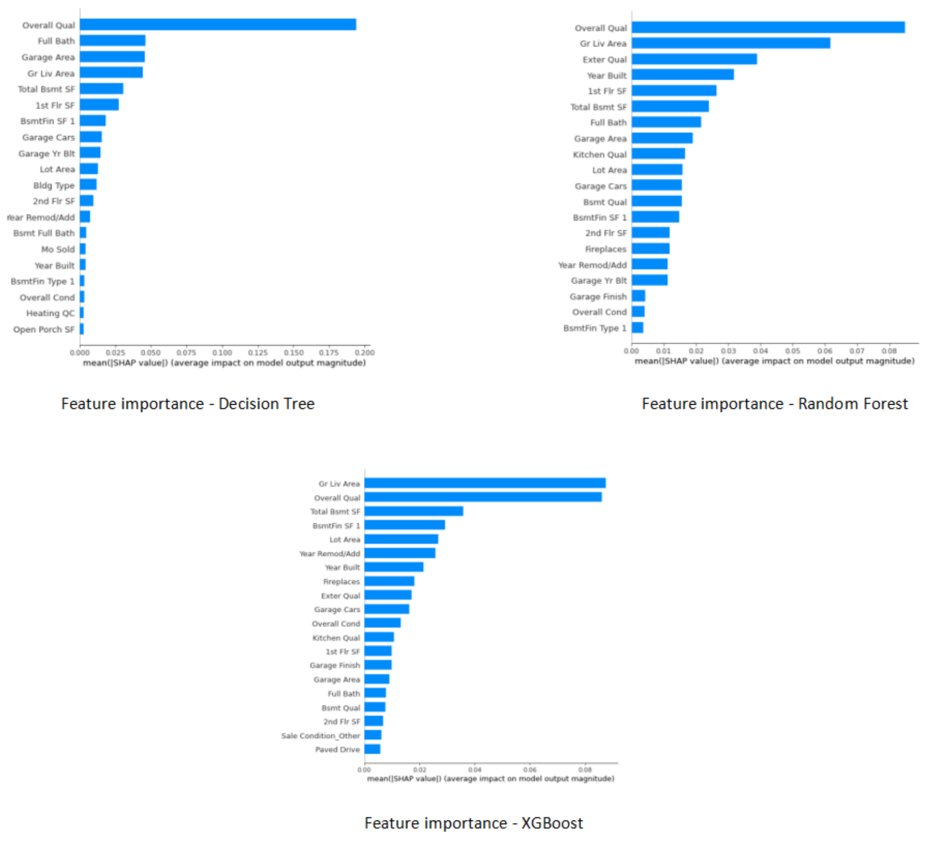


Figure 7: Feature importance for Decision Tree, Random Forest and XGBoost

The first one is related with the area, especially the size of above grade (ground) living, lot, basement and garage. The second feature is quality-related including the exterior quality, overall quality of the material used to build the house, and kitchen quality.

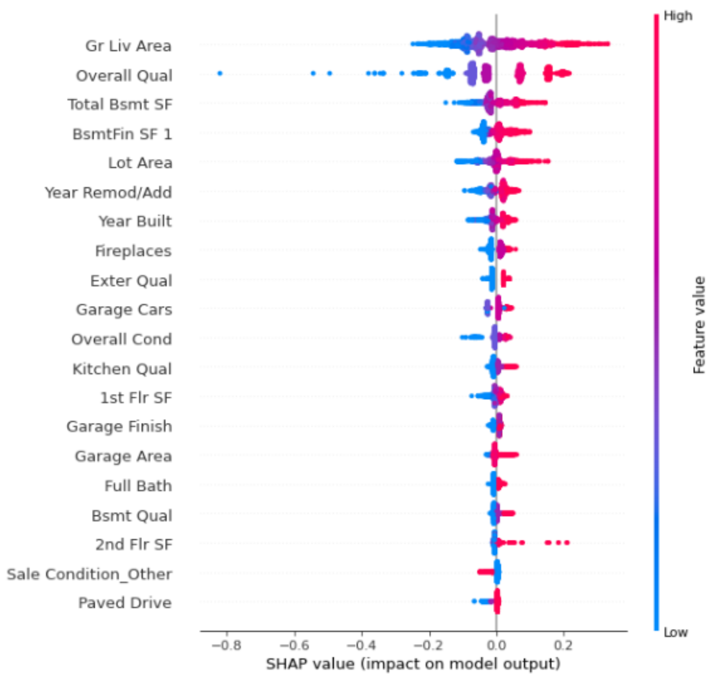


Figure 8: SHAP value impact on model output

SHAP value can also present how the feature will impact the house pricing. Take Figure 8 as an example, which is the feature importance generated by XGBoost, it’s clear that almost all the important features have positive influence on house pricing. For example, in this model, size of above living area is the most import feature, and the bigger the size is, the higher the price will be. Only one feature among the important features has negatively impact on the price which is Sale Condition_Other. Since the feature is the result of getting dummy, the value is 0 or 1 which means other sale condition will decrease the house price.

What’s more, some features that most people may care doesn’t have significant importance in XGBoost model. For example, the style of the house which refers how many floors does this house have, doesn’t affect the price greatly.

2.6 Conclusion

In conclusion, this study is aim to predict house sale price, with many features can affect the sale price, such as rates of the overall material and finish of the house, above ground living area, year built and others.

With the lowest error for prediction, ‘XGBoost’ and ‘Stacking’ model have been choosing as two best model. As suggesting by ‘XGBoost’ model and other models, there are several types of features contribute a lot for house sales price, such as area related factor such as the size of above grade (ground) living, and quality related factor such as exterior quality. Outlining those important features and help buyers and sellers of house to set their budgets price in an efficient way, and with the ‘XGBoost’ model, the sale price of house can be effectively predicted.

Nevertheless, there are still some limitations and potential improvement exist in this study. Firstly, as mentioned in the data processing section, the raw data contains a lot of missing values. Although the missing values have been filled with reasonable, it can still cause problem in accuracy compare to real content. If the data is with good quality and no missing value, the accuracy of the prediction should be more accurate. Secondly, for the model constructed in this study, the performance of model is not perfect for predicting sales price, further study and research is needed to be more accurate. Lastly, the sales price of house might be affected by factors not listed in the dataset for training model. Unexpected event happened can also affect the sale price of house, such as pandemic of Covid-19. To improve the accuracy of prediction, the time horizon of dataset provided should be considered.

3. TASK C

3.1 Data Processing

3.1.1 Load the Data

pd.read_csv() is used to load the given CSV file. There are 4,000 rows and 12 columns in the data. The last column is defined as the response variable 'quality' while the rest 11 columns are features.

```
wine_data = pd.read_csv('Wine_2020.csv')
quality = wine_data.iloc[:,-1]
features = wine_data.iloc[:,:-1]
```

3.1.2 Scaling

The observations of each feature and target are scaled by dividing maximum of each feature and target respectively. 'x' regards to the features which is an array with 4,000 rows and 11 columns; 'y' is the target which is reshaped as an array with 4,000 rows and 1 column.

```
features_scaled = features.div(features.max())
quality_scaled = quality.div(quality.max())
x = features_scaled
y = quality_scaled.values.reshape(-1,1)
```

3.2 Build a 3-Layer Neural Network

3.2.1 Initialize Hyper Parameters

A 3-layer neural network will be built with one input layer with 11 neurons, one hidden layer with 10 neurons and one output layer with 1 neuron (Figure 9). $W^{(1)}$ is the weight with 11 rows (features) and 10 columns (neurons in hidden layer), connecting input layer and hidden layer. $W^{(2)}$ is the weight with 10 rows (neurons in hidden layer) and 1 column (neuron in output layer), connecting hidden layer and output layer. $W^{(1)}$ and $W^{(2)}$ are randomly initialized with the random seed of 0. The learning rate of gradient descent is initialized as 0.0001.

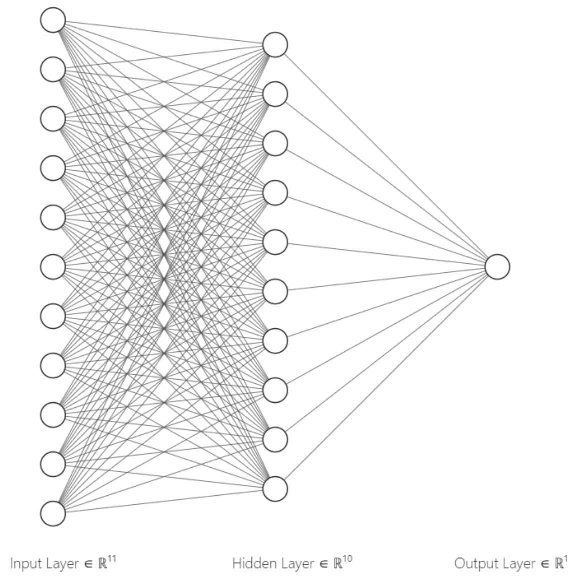


Figure 9: 3-layer neural network

```
input_layer_size = 11
hidden_layer_size = 10
```

```
output_layer_size = 1

np.random.seed(0)
W1 = np.random.randn(input_layer_size, hidden_layer_size)
W2 = np.random.randn(hidden_layer_size, output_layer_size)

alpha = 0.0001
```

3.2.2 Define Functions

The sigmoid activation function and the loss function which is the Sum of Square Error (SSE) are defined as below. The derivative of the sigmoid activation function is also shown below, which will be used in backpropagation.

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma'(z) = \sigma(z) * (1 - \sigma(z))$$

$$L(W) = \frac{1}{2}\sum_{n=1}^{N}(f(x_n, W) - t_n)^2$$

```
def sigmoid(z):
    return (1/(1 + np.exp(-z)))

def sse(z):
    return np.sum(np.power(z,2))/2
```

3.2.3 Train a 3-Layer Nerual Network

Iterate the following procedures 10,000 times. For each iteration, forward propagation, backpropagation, gradient descent are implemented and the loss is updated (Code 2 in Appendix C).

3.2.3.1 Forward Propagation

Propagation forward through the network to generate the output values. $a^{(1)}$ is the exmaple matrix with 4,000 rows (examples) and 11 columns (features). $W^{(1)}$ is the weight with 11 rows (features) and 10 columns (neurons in hidden layer), connecting input layer and hidden layer. Multiplying $a^{(1)}$and $W^{(1)}$ will get $Z^{(2)}$, the input of neuron (no activation) with 4,000 rows (examples) and 10 columns (neurons in hidden layer). $Z^{(2)}$ is then activated by sigmoid function to get $a^{(2)}$. $W^{(2)}$ is the weight with 10 rows (neurons in hidden layer) and 1 column (neuron in output layer), connecting hidden layer and output layer. Multiplying $a^{(2)}$ and $W^{(2)}$ will get $Z^{(3)}$, the input of neuron (no activation) with 4,000 rows (examples) and 1 column (neuron in output layer). $Z^{(3)}$ is then activated by sigmoid function to get $a^{(3)}$, which is the output value of a 3-layer neural network.

$$a^{(1)} = X$$

$$Z^{(2)} = a^{(1)}W^{(1)}$$

$$a^{(2)} = \sigma(Z^{(2)})$$

$$Z^{(3)} = a^{(2)}W^{(2)}$$

$$f(X, W) = a^{(3)} = \sigma(Z^{(3)})$$

```
    a_1 = x
    z_2 = np.dot(a_1,W1)
    a_2 = sigmoid(z_2)
    z_3 = np.dot(a_2,W2)
```

```
a_3 = sigmoid(z_3)
```

3.2.3.2 Calculate Loss

The loss is calculated using the predict values subtracted by observed values.

$$Loss = f(X, W) - t$$

```
loss = a_3 - y
```

3.2.3.3 Backpropagation

Propagation of the output activations back through the network to generate the deltas δ of all output and hidden neurons.

Step 1:

$\sigma(Z^{(3)})$ is $Z^{(3)}$ after activation which is $a^{(3)}$. With the derivative chain rule $\sigma'(z) = \sigma(z) * (1 - \sigma(z))$, the weight's output $\delta^{(3)}$ is caculated as the matrix element wise product of $Loss$ and $\sigma'(Z^{(3)})$. The weight's output $\delta^{(3)}$ and the transpose of input activation $a^{(2)}$ are multiplied to find the gradient of the weight $\frac{\partial L(W)}{\partial W^{(2)}}$. The weight's output $\delta^{(3)}$ has 4,000 rows (examples) and 1 column (neuron in output layer), while the gradient $\frac{\partial L(W)}{\partial W^{(2)}}$ has 10 rows (neurons in hidden layer) and 1 column (neuron in output layer).

$$\delta^{(3)} = Loss.* \sigma'(Z^{(3)})$$

$$\frac{\partial L(W)}{\partial W^{(2)}} = (a^{(2)})^T \delta^{(3)}$$

```
d_3 = loss * a_3 * (1 - a_3)
g_2 = np.dot(a_2.T,d_3)
```

Step 2:

$\sigma(Z^{(2)})$ is $Z^{(2)}$ after activation which is $a^{(2)}$. With the derivative chain rule $\sigma'(z) = \sigma(z) * (1 - \sigma(z))$, the weight's output $\delta^{(2)}$ is caculated as the matrix element wise product of $(\delta^{(3)}(W^{(2)})^T)$ and $\sigma'(Z^{(2)})$. The weight's output $\delta^{(2)}$ and the input X^T are multiplied to find the gradient of the weight $\frac{\partial L(W)}{\partial W^{(1)}}$. The weight's output $\delta^{(2)}$ has 4,000 rows (examples) and 10 columns (neurons in hidden layer), while the gradient $\frac{\partial L(W)}{\partial W^{(1)}}$ has 11 rows (features) and 10 columns (neurons in hidden layer).

$$\delta^{(2)} = (\delta^{(3)}(W^{(2)})^T).* \sigma'(Z^{(2)})$$

$$\frac{\partial L(W)}{\partial W^{(1)}} = X^T \delta^{(2)}$$

```
loss_2 = d_3 * W2.T
d_2 = loss_2 * a_2 * (1 - a_2)
g_1 = np.dot(x.T,d_2)
```

3.2.3.4 Gradient Descent

A ratio (learning rate) of the weights' gradient is subtracted from the weights to update the new weights.

$$W^{(1)} := W^{(1)} - \alpha \frac{\partial L(W)}{\partial W^{(1)}}$$

$$W^{(2)} := W^{(2)} - \alpha \frac{\partial L(W)}{\partial W^{(2)}}$$

```
W1 = W1 - (alpha * g_1)
W2 = W2 - (alpha * g_2)
```

3.2.3.5 Incorporate the Updated Weights

The updated weights are then incorporated into forward propagation and the loss is calculated and appended to a loss list.

```
a_1 = x
z_2 = np.dot(a_1,W1)
a_2 = sigmoid(z_2)
z_3 = np.dot(a_2,W2)
a_3 = sigmoid(z_3)
loss = a_3 - y
loss_list.append(sse(loss))
```

3.3 Neural Network Diagnosis

3.3.1 Loss Plot and Final Loss

As shown in Figure 10, as the number of iteration increasing, the loss drops rapidly at the beginning when the number of iterations is small and decreases slowly when the number of iterations is large. But the loss hasn't totally been converged yet, if the number of iteration keeps increasing, the loss will turn to converge. The final estimated loss is 8.7248 after iterating 1,000 times.

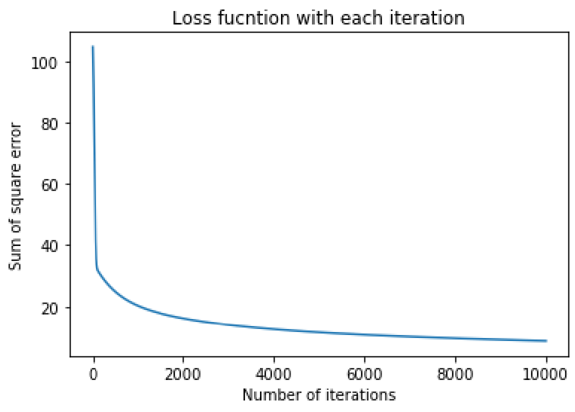


Figure 10: Loss function with each iteration

3.3.2 Final Estimated Weights

The final estimated weights are shown below.

```
W1 =
[[ 1.56166955  0.78966505  0.12458661  2.30770085  1.93746203 -0.87709126
   1.41698331 -0.30327564  0.04623755  0.34773496]
 [-0.21818819  0.99547501  1.50269693 -0.24824729  0.5372052   0.21662432
   1.13571222 -0.09947502  0.19286905 -1.13518417]
 [-2.62894952  0.77872664  0.46647456 -0.66957941  2.31902182 -1.39992322
   0.26672385 -0.28015774  1.63836509  1.35376455]
 [ 0.05029951  0.26956458 -0.7227979  -2.05801332 -0.31864967  0.13097791
   1.16147309  1.22403124 -0.41195902 -0.37848528]
 [-1.43437235 -1.59408771 -1.31144574  1.59725856 -0.39171407 -0.50716884
  -1.33454628  0.80234192 -1.66230645 -0.44248044]
 [-0.76705126  0.88797384 -1.32294823 -0.79247064  0.01091677  0.58766245
   0.57264241  0.11991656 -0.41035697 -0.40462157]
[-0.81920115 -0.14623516 -1.10767468 -1.61509959  0.25604096 -0.35140463
  -1.35092399  0.37750791 -0.81921975 -0.07454705]
[-0.05524363 -0.37346421  1.7351261  -1.79021133  0.63491497 -0.81453124
  -1.11362811 -0.51967548 -0.40904194 -0.42989148]
[-1.65496188  0.95663918  0.1665545  -1.70589376  1.65037434  1.90323735
   1.39126703 -0.25844477 -1.02873166  0.80488766]
[-0.72005874  1.18159763  0.05794214  0.80721336  0.47450998  0.70352546
   0.1173126   1.73849575  0.153675    0.20076373]
[ 1.34940426 -1.43565464 -1.26776123  0.68596714 -1.00570253  1.90865003
  -0.35139933 -0.77556952  1.91822615  1.19637698]]

W2 =
[[ 0.73704807]
 [ 1.26593371]
 [-2.2746167 ]
 [ 1.11868777]
 [-0.66899316]
 [ 0.43559584]
 [ 1.05454494]
 [-0.32888345]
 [ 0.17833581]
 [ 0.95878048]]
```

3.3.3 Predict Values and Observed Values

As shown in Figure 11, there's a strong relationship between the predict target values and observed target values. As the predict target values increasing, observed target values would increase, which means this 3-layer neural network predicts well in its performance.

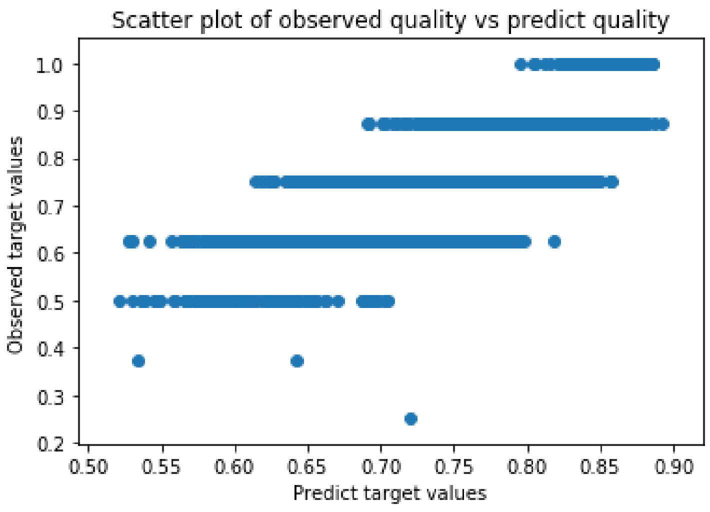


Figure 11: Scatter plot of observed values and predict values

3.4 Optimisation

To increase the predicting accuracy of neural network, different learning rates, number of neurons in the hidden layers and number of iteration need to be tested to find a lower SSE. Instead fixed learning rate alpha to 0.0001, a list of alpha including 0.1, 0.01, 0.001, 0.0001 and 0.00001 is applied to loop. In addition, number neurons in hidden layer are also change to a list including 8, 9, 10, 11 and 12. For the number of iteration, a stopping criteria is included in the loop to reduce overfitting and improve the generalization of neural network. The stopping criterial is listed below.

```
counter = 0
min_delta = 0.00005
patience = 100
best_score = None
```

The ‘min_delta’ is used to critique the minimal change in loss should be at lease 0.005%. After the change of iteration is smaller than 0.005%, the stopping criterial will use ‘patience’ to further test 100 events, make sure no events result in a minimal 0.005% change. The result of iteration with a list of alpha and hidden layer with stopping criteria is listed below.

```
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 107
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 107
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 124
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 100
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 3252
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 100
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 3696
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 102
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 4422
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 3213
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 2915
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 3430
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 3301
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 4138
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 8300
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 6098
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 867
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 6908
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 5534
Early Stopped: counter = 100, min_delta = 0.000050, iteration = 2983
Best parameters are: Alpha = 0.010000, hidden_layer_size = 8
Corresponding loss: 5.780460
```

As highlighted in above code, the maximal number of iterations only reach 8,300, hence, there is no need to increase the number of iteration in the algorithm. Besides, the estimated loss of neurons network has been reducing to 5.7805 with the learning rate alpha 0.01 and the number of neurons in hidden layer of 8. Hence, to optimise the prediction accuracy of this model, alpha should be set to 0.01 and with number of neurons in hidden layer set to 8.

4. Reference

Breiman, L 2001, ‘Random Forests.(Author abstract)’ *Machine Learning*, vol. 45, no. 1, pp. 5–32, doi: 10.1023/A:1010933404324.

Calhoun, C 2003, ‘Property valuation models and house price indexes for the Provinces of Thailand: 1992-2000’ *Housing Finance International*, vol. 17, no. 3, pp. 31–41, Retrieved from http://search.proquest.com/docview/216210879/.

Frew, J 2003, ‘Estimating the value of apartment buildings’ *Journal of Real Estate Research*, vol. 25, no. 1, pp. 77–86, Retrieved from http://search.proquest.com/docview/38456613/.

Appendix A (Task A)

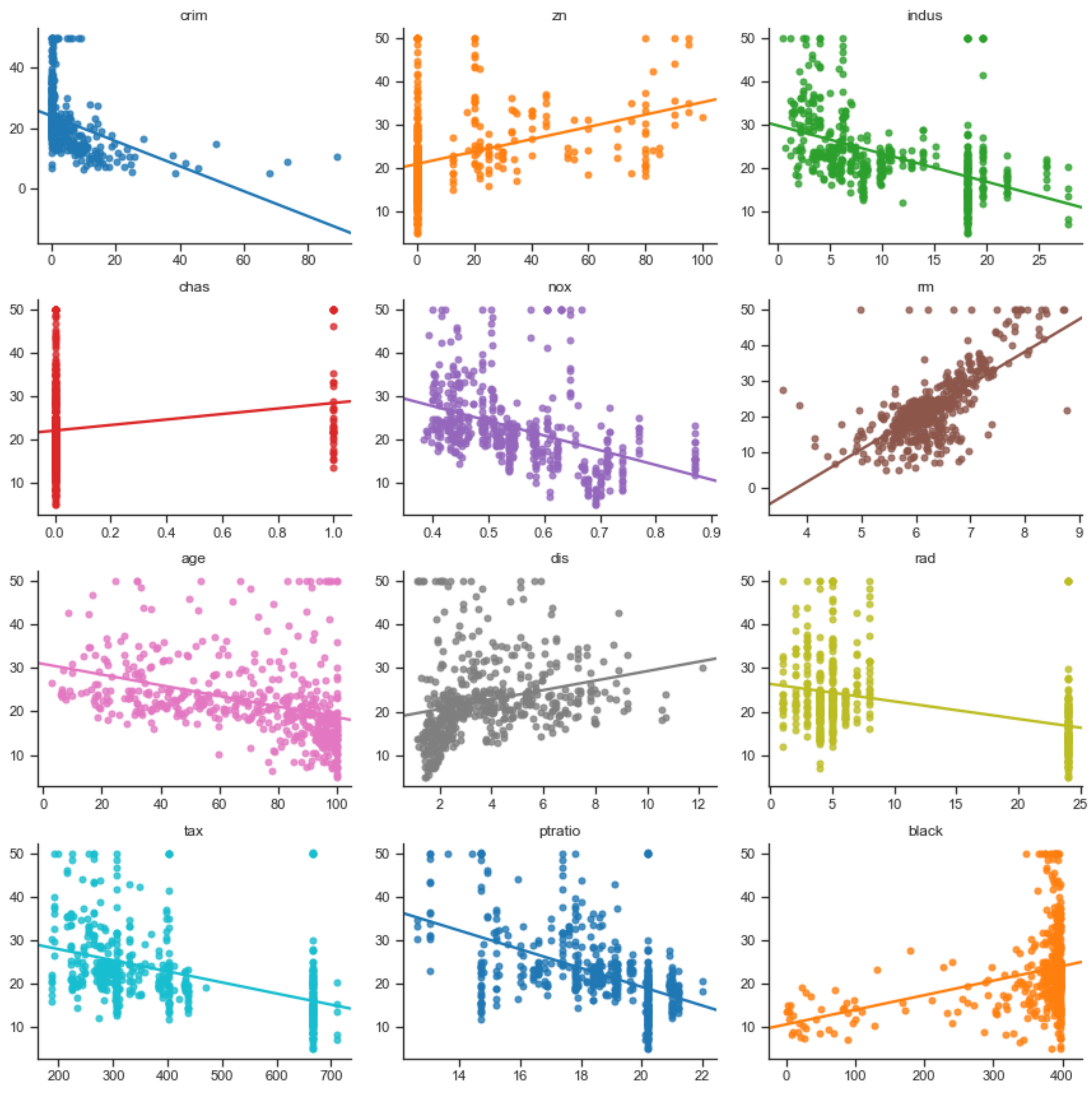


Figure 1: Scatter plot of numeric predictors with the response variable

```python
        # When the likelihood doesn't change, break the iteration and return W
       if (abs(loss_total[-1]-loss)) < 0.00001:
           break
       else:
           # save all the loss function values at each step
            loss_total[i]= loss
```

Code 1: Gradient descent

Appendix B (Task B)

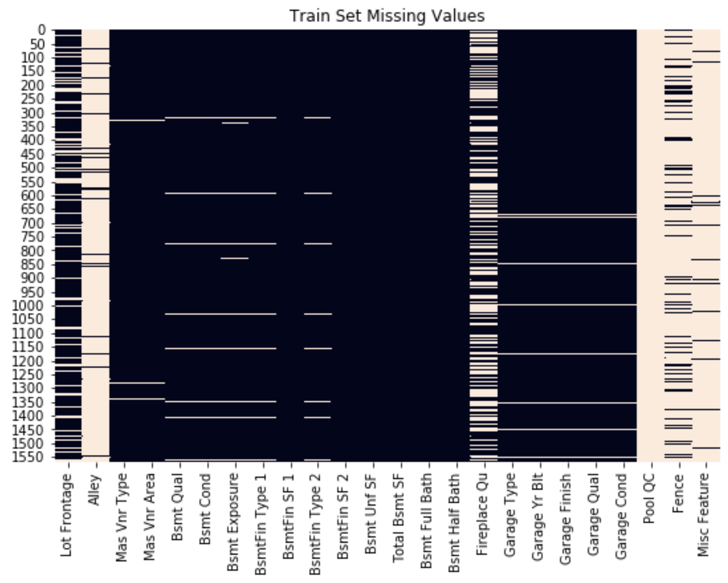
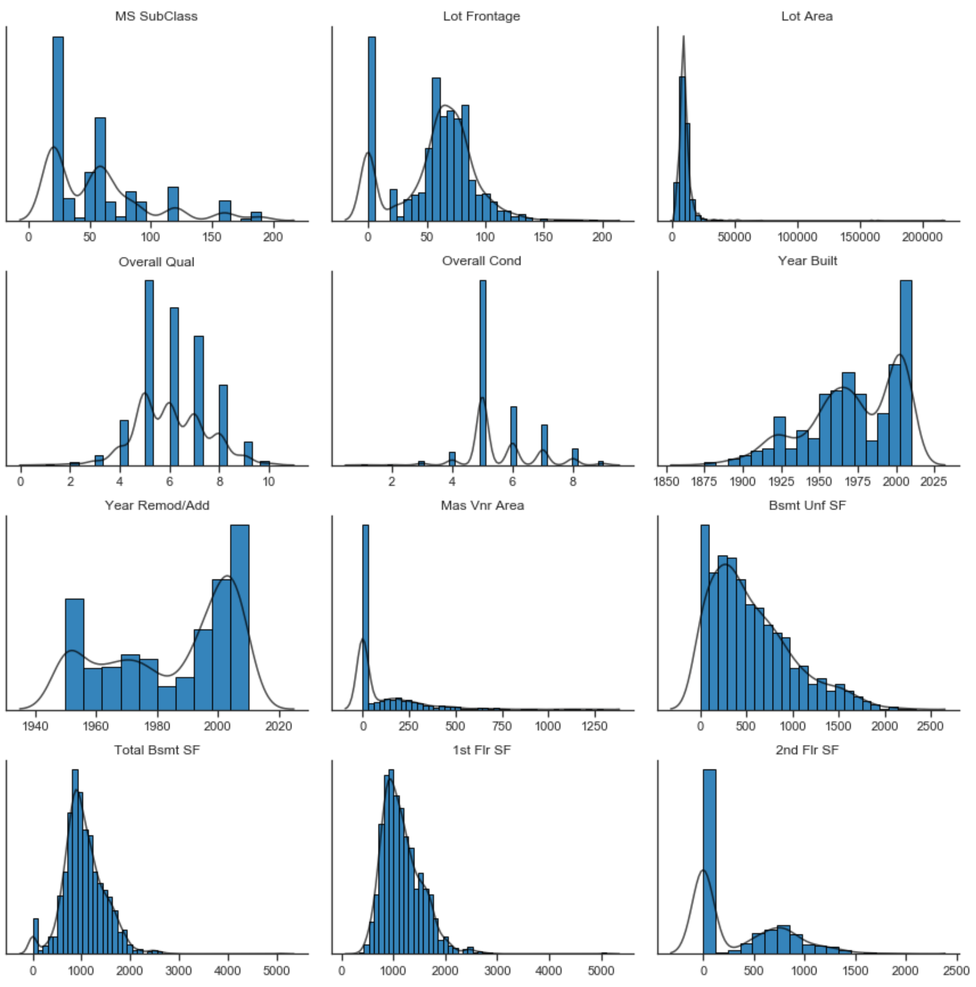


Figure 2: Missing values in training set



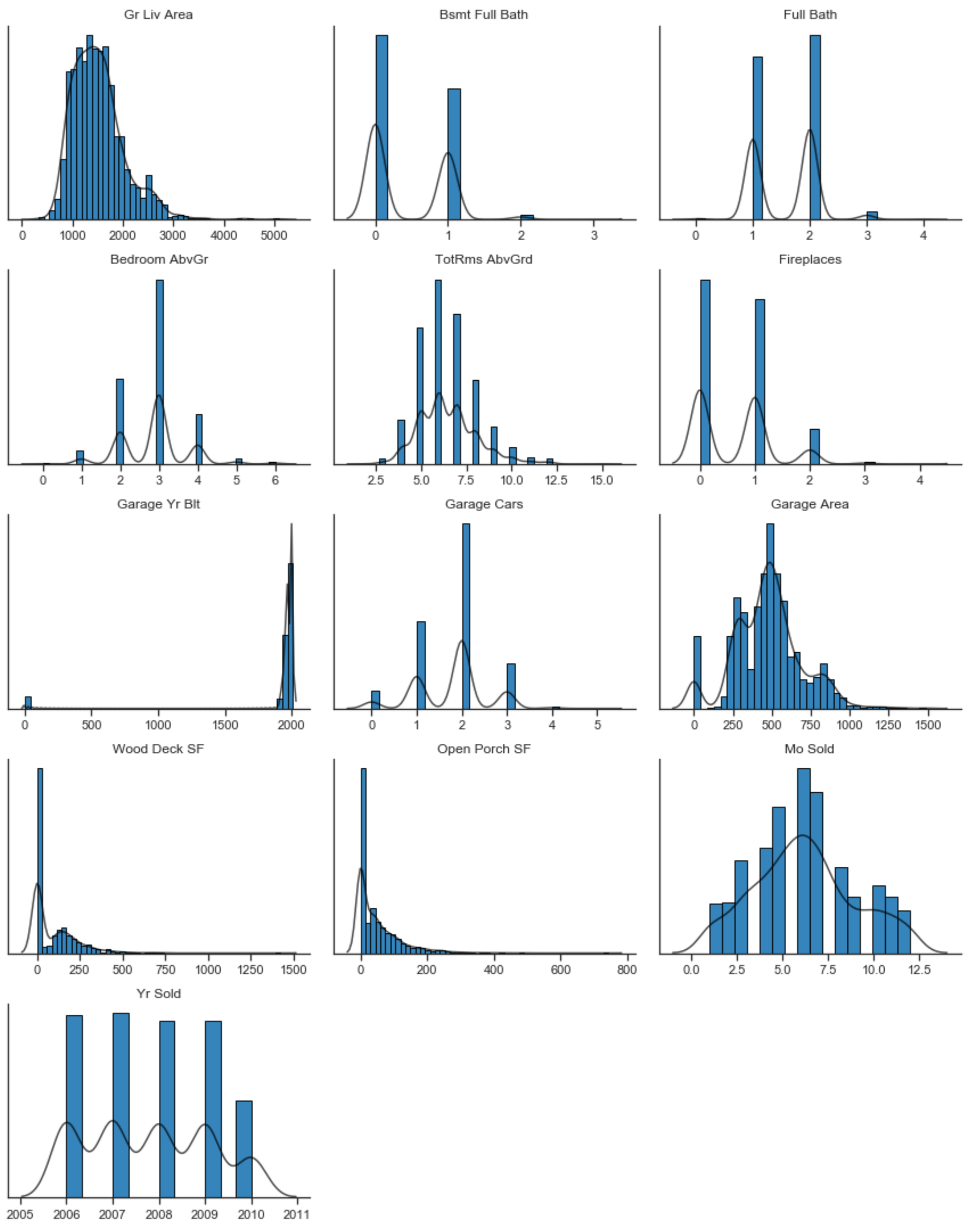
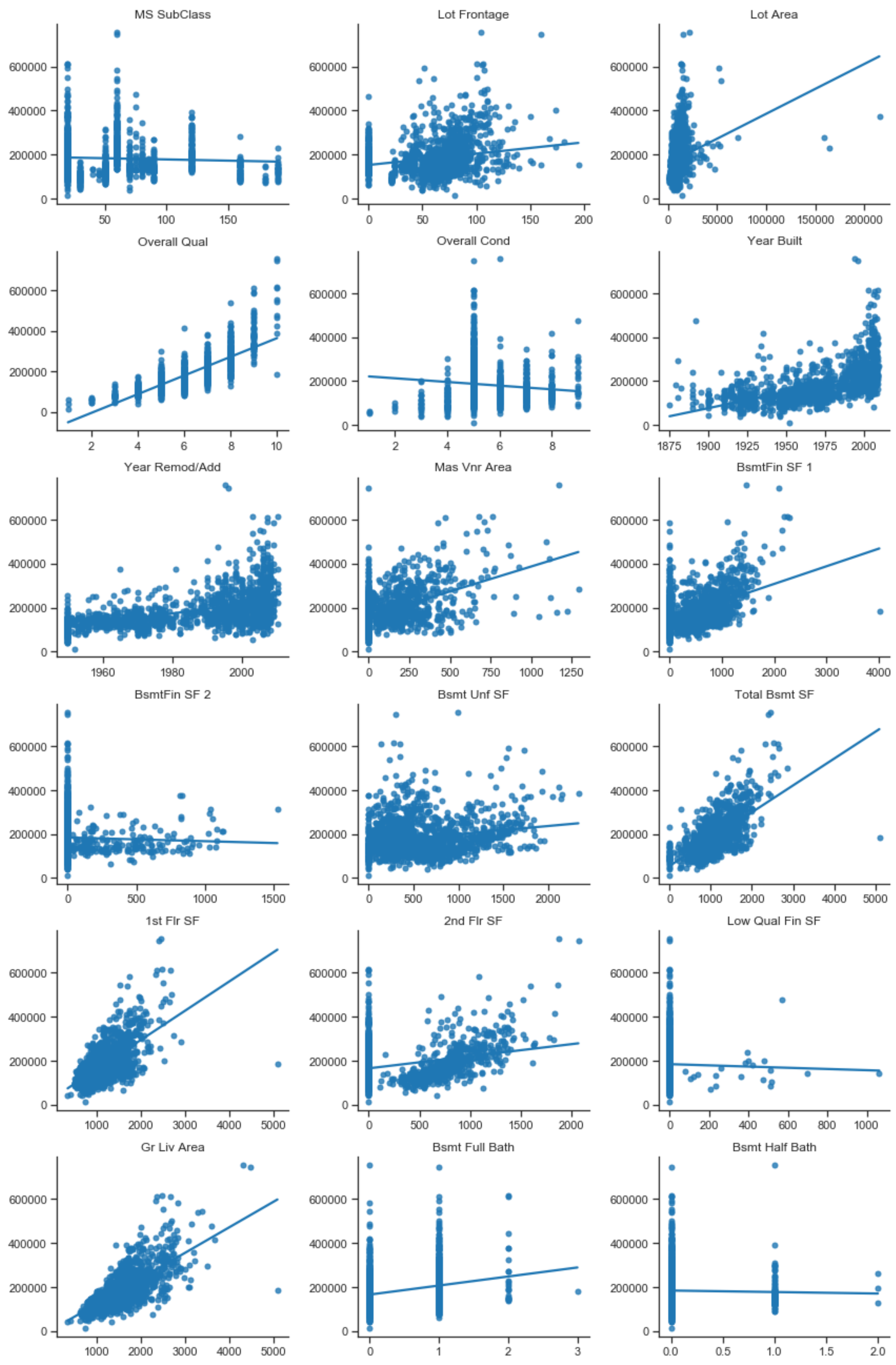


Figure 3: Distribution of numerical variables


MS SubClass
Lot Frontage
Lot Area
Overall Qual
Overall Cond
Year Built
Year Remod/Add
Mas Vnr Area
BsmtFin SF 1
BsmtFin SF 2
Bsmt Unf SF
Total Bsmt SF
1st Flr SF
2nd Flr SF
Low Qual Fin SF
Gr Liv Area
Bsmt Full Bath
Bsmt Half Bath

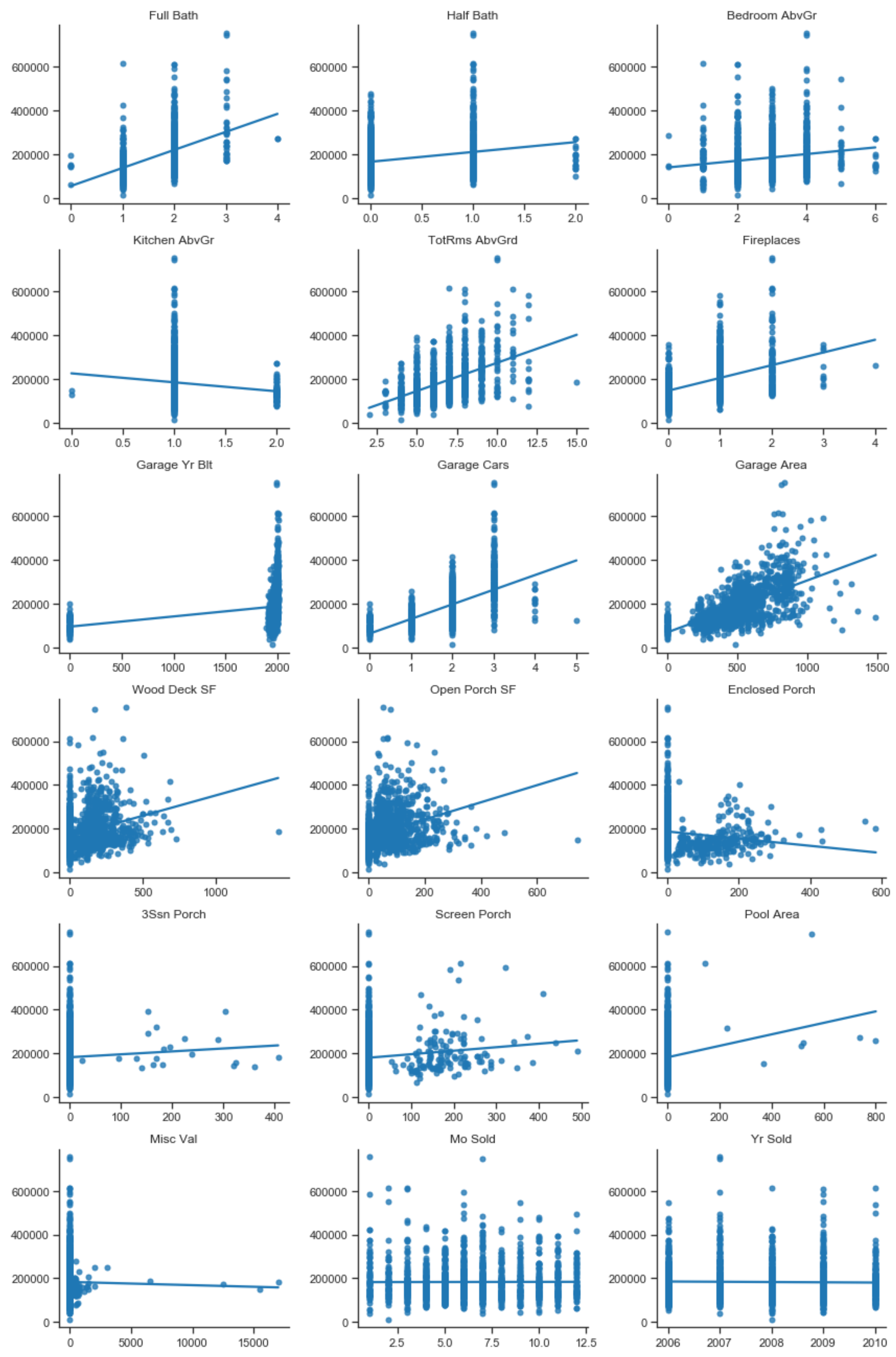


Figure 4: Scatter plot of numeric predictors with the response variable

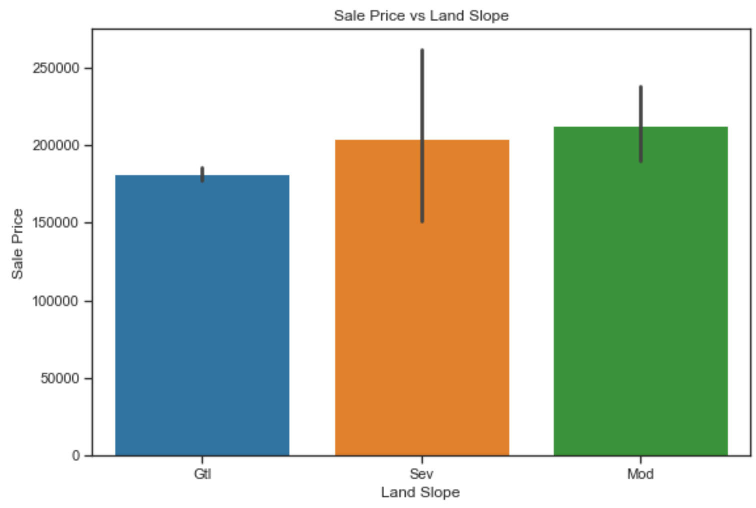


Figure 5: Sale price vs land slope

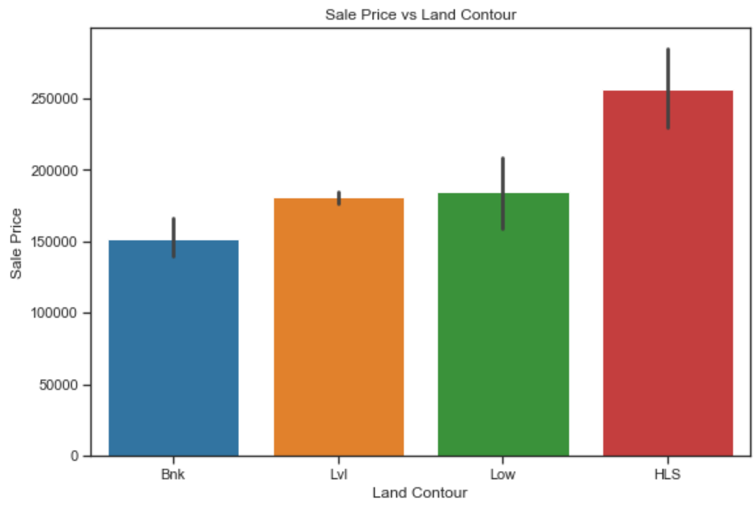


Figure 6: Sale price vs land contour

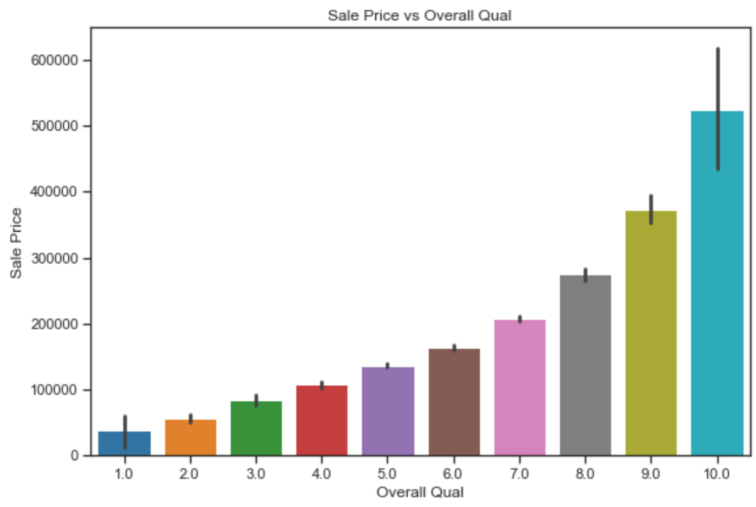


Figure 7: Sale price vs overall quality

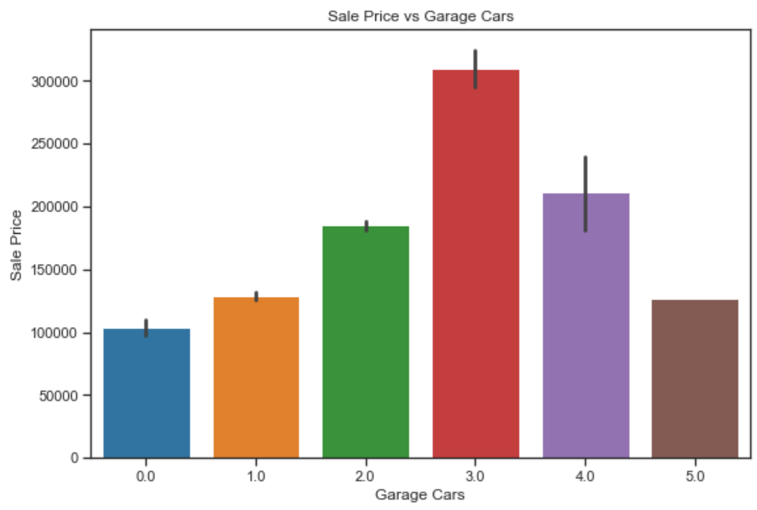


Figure 8: Sale price vs garage cars

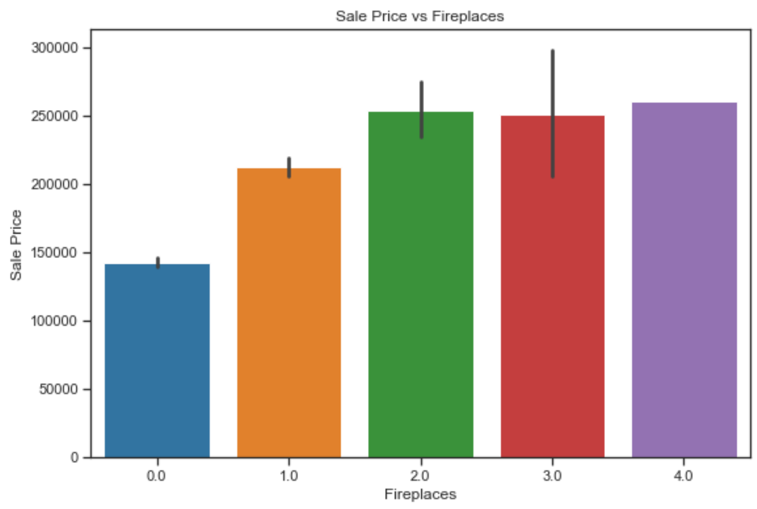


Figure 9: Sale price vs fireplaces

Table 1: Data processing and feature engineering details

Index	Variable	Null Count	Dtype	Data Type	Deal with Null Count	Feature Engineering
1	**Train_ID**	0	int64	Numerical	0	NAN
2	**MS SubClass**	0	int64	Numerical	0	Log Transformation
3	**MS Zoning**	0	object	Categorical	Mode	Only keep "Residential" and "Others"
4	**Lot Frontage**	264	float64	Numerical	0	NAN
5	**Lot Area**	0	int64	Numerical	0	Log Transformation
6	**Street**	0	object	Binary	Mode	'Grvl':0, 'Pave':1
7	**Alley**	1472	object	Binary	Mode	'Grvl':0, 'Pave':1
8	**Lot Shape**	0	object	Ordinal	Mode	'IR3':3,'IR2':2,'IR1':1,'Reg':0
9	**Land Contour**	0	object	Ordinal	Mode	'Low':3,'HLS':2,'Bnk':1,'Lvl':0
10	**Utilities**	0	object	Ordinal	Mode	'NoSeWa':2,'NoSewr':1,'AllPub':0
11	**Lot Config**	0	object	Categorical	Mode	Only keep 'Inside','Corner' and other
12	**Land Slope**	0	object	Ordinal	Mode	'Sev':2,'Mod':1,'Gtl':0
13	**Neighborhood**	0	object	Categorical	Mode	Only keep'CollgCr','Edwards','NAmes','OldTown','other'
14	**Condition 1**	0	object	Categorical	Mode	Only keep 'Norm', and other
15	**Condition 2**	0	object	Categorical	Mode	Only keep 'Norm', and other
16	**Bldg Type**	0	object	Ordinal	Mode	'Twnhs':4,'TwnhsE':3,'Duplex':2,'2fmCon':1,'1Fam':0
17	**House Style**	0	object	Ordinal	Mode	SLvl':7,'SFoyer':6,'2.5Unf':5,'2.5Fin':4,'2Story':3,'1.5Unf':2,'1.5Fin':1,'1Story':0
18	**Overall Qual**	0	int64	Numerical	0	Log Transformation
19	**Overall Cond**	0	int64	Numerical	0	Log Transformation
20	**Year Built**	0	int64	Numerical	0	NAN
21	**Year Remod/Add**	0	int64	Numerical	0	NAN
22	**Roof Style**	0	object	Categorical	Mode	Only keep 'Gable','Hip' and other
23	**Roof Matl**	0	object	Categorical	Mode	Only keep 'CompShg'and other
24	**Exterior 1st**	0	object	Categorical	Mode	Only keep'HdBoard','MetalSd','VinylSd','Wd Sdng','other'
25	**Exterior 2nd**	0	object	Categorical	Mode	Only keep'HdBoard','MetalSd','VinylSd','Wd Sdng','other'
26	**Mas Vnr Type**	15	object	Ordinal	Mode	'BrkCmn':3,'BrkFace':2,'Stone':1,'None':0
27	**Mas Vnr Area**	15	float64	Numerical	0	Log Transformation
28	**Exter Qual**	0	object	Ordinal	Mode	'Ex':3,'Gd':2,'TA':1,'Fa':0
29	**Exter Cond**	0	object	Ordinal	Mode	'Ex':4,'Gd':3,'TA':2,'Fa':1,'Po':0
30	**Foundation**	0	object	Ordinal	Mode	'BrkTil':5,'CBlock':4,'PConc':3,'Slab':2,'Stone':1,'Wood':0
31	**Bsmt Qual**	42	object	Ordinal	Mode	'Ex':4,'Gd':3,'TA':2,'Fa':1,'Po':0
32	**Bsmt Cond**	42	object	Ordinal	Mode	'Ex':4,'Gd':3,'TA':2,'Fa':1,'Po':0
33	**Bsmt Exposure**	44	object	Ordinal	Mode	'Gd':3,'Av':2,'Mn':1,'No':0
34	**BsmtFin Type 1**	42	object	Ordinal	Mode	'GLQ':5,'ALQ':4,'BLQ':3,'Rec':2,'LwQ':1,'Unf':0
35	**BsmtFin SF 1**	1	float64	Numerical	0	Log Transformation
36	**BsmtFin Type 2**	42	object	Ordinal	Mode	'GLQ':5,'ALQ':4,'BLQ':3,'Rec':2,'LwQ':1,'Unf':0
37	**BsmtFin SF 2**	1	float64	Numerical	0	Log Transformation
38	**Bsmt Unf SF**	1	float64	Numerical	0	Log Transformation
39	**Total Bsmt SF**	1	float64	Numerical	0	Log Transformation
40	**Heating**	0	object	Categorical	Mode	Only keep 'GasA'and other
41	**Heating QC**	0	object	Ordinal	Mode	'Ex':4,'Gd':3,'TA':2,'Fa':1,'Po':0
42	**Central Air**	0	object	Binary	Mode	'N':0, 'Y':1
43	**Electrical**	0	object	Ordinal	Mode	'Mix':4,'FuseP':3,'FuseF':2,'FuseA':1,'SBrkr':0
44	**1st Flr SF**	0	int64	Numerical	0	Log Transformation
45	**2nd Flr SF**	0	int64	Numerical	0	Log Transformation
46	**Low Qual Fin SF**	0	int64	Numerical	0	Log Transformation
47	**Gr Liv Area**	0	int64	Numerical	0	Log Transformation
48	**Bsmt Full Bath**	1	float64	Numerical	0	Log Transformation
49	**Bsmt Half Bath**	1	float64	Numerical	0	Log Transformation
50	**Full Bath**	0	int64	Numerical	0	Log Transformation
51	**Half Bath**	0	int64	Numerical	0	Log Transformation
52	**Bedroom AbvGr**	0	int64	Numerical	0	Log Transformation
53	**Kitchen AbvGr**	0	int64	Numerical	0	Log Transformation
54	**Kitchen Qual**	0	object	Ordinal	Mode	Ex':4,'Gd':3,'TA':2,'Fa':1,'Po':0
55	**TotRms AbvGrd**	0	int64	Numerical	0	Log Transformation
56	**Functional**	0	object	Ordinal	Mode	'Typ':7,'Min1':6,'Min2':5,'Mod':4,'Maj1':3,'Maj2':2,'Sev':1,'Sal':0
57	**Fireplaces**	0	int64	Numerical	0	Log Transformation
58	**Fireplace Qu**	749	object	Ordinal	Mode	'Ex':4,'Gd':3,'TA':2,'Fa':1,'Po':0
59	**Garage Type**	82	object	Ordinal	Mode	'2Types':5,'Attchd':4,'Basment':3,'BuiltIn':2,'CarPort':1,'Detchd':0
60	**Garage Yr Blt**	83	float64	Numerical	0	NAN
61	**Garage Finish**	83	object	Ordinal	Mode	'Fin':2,'RFn':1,'Unf':0
62	**Garage Cars**	0	int64	Numerical	0	NAN
63	**Garage Area**	0	int64	Numerical	0	Log Transformation
64	**Garage Qual**	83	object	Ordinal	Mode	Ex':4,'Gd':3,'TA':2,'Fa':1,'Po':0
65	**Garage Cond**	83	object	Ordinal	Mode	'Ex':4,'Gd':3,'TA':2,'Fa':1,'Po':0
66	**Paved Drive**	0	object	Ordinal	Mode	'Y':2,'P':1,'N':0
67	**Wood Deck SF**	0	int64	Numerical	0	Log Transformation
68	**Open Porch SF**	0	int64	Numerical	0	Log Transformation
69	**Enclosed Porch**	0	int64	Numerical	0	Log Transformation
70	**3Ssn Porch**	0	int64	Numerical	0	Log Transformation
71	**Screen Porch**	0	int64	Numerical	0	Log Transformation
72	**Pool Area**	0	int64	Numerical	0	Log Transformation
73	**Pool QC**	1562	object	Ordinal	Mode	'Ex':3,'Gd':2,'TA':1,'Fa':0
74	**Fence**	1267	object	Ordinal	Mode	'GdPrv':3,'MnPrv':2,'GdWo':1,'MnWw':0
75	**Misc Feature**	1514	object	Categorical	Mode	Only keep 'Shed'and other
76	**Misc Val**	0	int64	Numerical	0	Log Transformation
77	**Mo Sold**	0	int64	Numerical	0	Log Transformation
78	**Yr Sold**	0	int64	Numerical	0	Log Transformation
79	**Sale Type**	0	object	Categorical	Mode	Only keep 'WD', 'New' and other
80	**Sale Condition**	0	object	Categorical	Mode	Only keep 'Normal'and other
81	**SalePrice**	0	int64	Numerical	0	Log Transformation

Appendix C (Task C)

```python
input_layer_size = 11
hidden_layer_size = 10
output_layer_size = 1

np.random.seed(0)
W1 = np.random.randn(input_layer_size, hidden_layer_size)
W2 = np.random.randn(hidden_layer_size, output_layer_size)

alpha = 0.0001

iterations = 10000
loss_list = []

# Forward propagation
for i in range(iterations):
    a_1 = x
    z_2 = np.dot(a_1,W1)
    a_2 = sigmoid(z_2)
    z_3 = np.dot(a_2,W2)
    a_3 = sigmoid(z_3)
    loss = a_3 - y

# Backpropagation
    d_3 = loss * a_3 * (1 - a_3)
    g_2 = np.dot(a_2.T,d_3)

    loss_2 = d_3 * W2.T
    d_2 = loss_2 * a_2 * (1 - a_2)
    g_1 = np.dot(x.T,d_2)

# Gradient decent
    W1 = W1 - (alpha * g_1)
    W2 = W2 - (alpha * g_2)

# Loss after iteration
    a_1 = x
    z_2 = np.dot(a_1,W1)
    a_2 = sigmoid(z_2)
    z_3 = np.dot(a_2,W2)
    a_3 = sigmoid(z_3)
    loss = a_3 - y

    loss_list.append(sse(loss))
```

Code 2: Train a 3-layer neural network