University of Essex

Department of Mathematical Sciences

MA321-7-SP: Applied Statistics

Group Project

Registration numbers:

1802655, 1907394, 2007519, 2003861, 2004352, 2007161

Group: 9

Date of submission: (26 March 2021)

Word count: 3578

Abstract

As machine learning techniques develop, an area of interest in their application is on the housing market. In this study exploratory data analysis is performed, identifying appropriate pre-processing techniques to deal with areas such as missing values, feature selection and skewed distributions. Classification methods are explored and implemented to model whether the overall condition of houses can be predicted. Further to this regression analysis is also undertaken to model whether the sale price of houses can be predicted, and resampling methods are used to estimate the test error of these models. Finally, a research question on whether the year a property is built can be predicted is developed and implemented through the use of multiple linear regression.

Introduction

Predicting house prices is a complex task – but one that has become increasingly feasible with the development of advanced statistical methods and machine learning. Given the availability of house data, the task is often employed as a benchmark task for machine learning models. In this example, we are provided with a data set of 1460 properties in 15 neighbourhoods, with 50 informative features such as the overall quality of the house, the year it was built, and the type of roof it has.

The algorithms used throughout our analysis broadly belong to the domain of supervised learning. Supervised learning assigns one variable to be the 'Target' label (which we will try to predict) and uses other variables as informative features. This ground truth allows us to train a model on the data we have, with the intention of making a model capable of estimating real values for new properties. Within supervised learning, we carry out two main tasks:

Section 3.1-3.2 is an example of a classification task, we bin 'OverallCondition' into 3 separate factors, and try to predict which bin a property would belong in (thereby *classifying* each example).

Section 3.3 is an example of a regression task, where we try to predict a continuous variable, SalePrice. We use a similar methodology in section 3.5 to predict the year the house was built.

2. Preliminary analysis

2.1 Data

The data provided is a simple dataset with 1460 rows, each representing a single property, and 50 columns, all representing individual features of the houses. The provided 'description.txt' provides details of all the variables provided – both categorical and continuous variables are included.

2.2 Missing Values

We start by investigating the missing values within the dataset. Figure 1 illustrates the issue of missing values within the data, with some variables missing for more than 90% of observations.

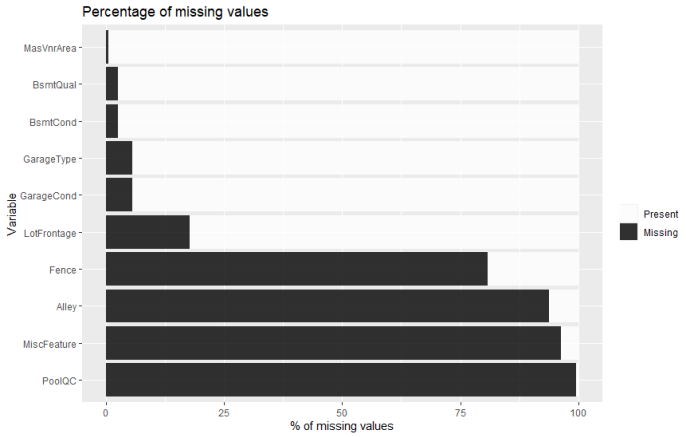


Figure 1: Percentage of missing values

While it would usually make sense to drop such variables, closer analysis of the provided descriptions explains the meaning of NA's - which is unrepresentative of the true meaning of a missing value. In the example of 'Fence', those marked missing are properties without a Fence at all. This is true for many of the highest percentage missing value variables, so we impute these with a new factor 'None'.

This leaves the only true missing values as 'MasVnrArea' and 'LotFrontage' which we impute using the mean of the feature, and the median of the feature of the associated neighbourhood of that property respectively.

2.2 Distribution

We also inspect variables to gain an insight into their respective distributions. Appendix 1 highlights some of the key categorical features.

For categorical features, we use count plots to inspect class balance. Features such as ‘Alley’ are dominated by None - so this is mostly likely to be targeted during our feature selection to be dropped.

For continuous variables, we use histograms. Figure 2 shows the response variable, SalePrice to be skewed.

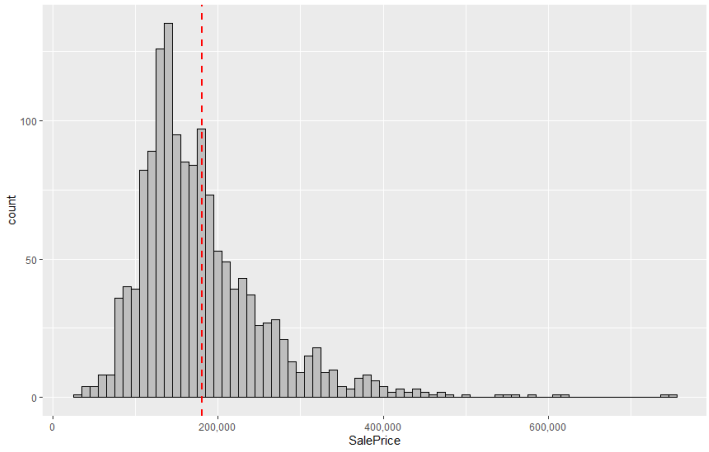


Figure 2: Sale price distribution

A log transformation described in section 3.3 ensures that this key variable is normally distributed, thereby increasing the chances that our residuals are normally distributed and increasing model fit (3).

2.3 Correlation Matrix

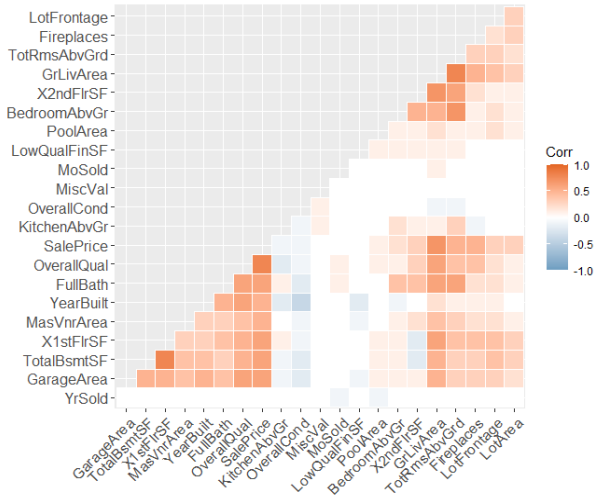


Figure 3: Correlation matrix

An additional area of investigation is correlation, both between the individual variables and their correlation with 'SalePrice' (which will be the primary response variable throughout this report). The correlation matrix shows the variables with the highest correlation to our main target variable, including GrLivArea and OverallQual. Appendix 2 displays these relationships further.
The problem of multicollinearity can be seen from the correlation matrix. Variables with high correlation with each other, such as TotRmsAbvGrd (total rooms above grade) and GrLivArea (above grade living area) could be approximated to linear transformations of one another.

2.4 Feature Selection

The Boruta library utilises random forests in order to perform feature selection by finding statistically significant variables in a model and reporting the variable importance measure of each (8). We run the Boruta() function once each for three different models with different variables as the response: OverallCond, SalePrice, and YearBuilt. We do this as we want a handpicked dataset for each class to return the best possible results for each section of our report, OverallCond for our logistic regression and random forest in section 3.1-3.2, SalePrice for our predictions in section 3.3, and YearBuilt for section 3.5.

The Boruta function extracts the confirmed, tentative, and rejected significant attributes for a response variable. Tentative attributes are ones that that Boruta has not yet decided on whether it is significant or not. appendix 3, 4 and 5 visualise the ascending importance of each variable in regard to each response variable we have specified. Red shows rejected variables, yellow is tentative, and green is confirmed significant variables. Blue bars can be ignored as they are shadow plots used by Boruta to help decide if figures are significant or not. As the plots illustrate, most of the variables within the data are not relevant to the estimators. This is likely due to the large number of categorical values which become their own dummy column, when in fact the value is only actually present in a handful of rows in the data.

We can apply a further test to tentative variables to check whether they are in fact relevant or not by using the TentativeRoughFix function. We do not want to mistakenly exclude tentative variables as they may possibly be relevant. The output of this function provides a set

of confirmed and rejected variables only. Following this, we create a smaller dataset containing only the confirmed significant variables and the corresponding response variable. The result is a subset of data with only relevant variables which helps solve the variable excess issue caused by one hot encoding our data. In total we have 63 variables that are relevant for OverallCond, and 80 variables for SalePrice, and 108 for YearBuilt.

3. Analysis

3.1 Predicting OverallCond Using Multinomial Logistic Regression

```
          Reference
Prediction Good Poor Average
   Good      52    0      44
   Poor       0    1       9
   Average   24    2     307

                Accuracy : 0.82
                  95% CI : (0.7809, 0.8549)
     No Information Rate : 0.82
     P-Value [Acc > NIR] : 0.53

                   Kappa : 0.4706

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Good Class: Poor Class: Average
Sensitivity               0.6842     0.333333         0.8528
Specificity               0.8788     0.979358         0.6709
Pos Pred Value            0.5417     0.100000         0.9219
Neg Pred Value            0.9300     0.995338         0.5000
Prevalence                0.1731     0.006834         0.8200
Detection Rate            0.1185     0.002278         0.6993
Detection Prevalence      0.2187     0.022779         0.7585
Balanced Accuracy         0.7815     0.656346         0.7618
```

Figure 4: Confusion matrix and statistical summary for logistic regression

The houses are split into Poor, Average and Good based on their overall condition. Classifications are unbalanced with Poor: 31 houses, Average: 1130 houses and Good: 299 houses.

One-hot coding was performed to give 205 features. Feature selection using the Boruta method indicates the variation in the data is explained by 65 features.

Multinomial logistic regression (5) was performed using a “one-to-rest” strategy with ‘Good’ as the reference level with 10 k-fold cross validation to give an accuracy of 82% (figure 4).

The No Information Rate is equal to the accuracy rate, suggesting that the classifying observations as the majority class (Average) achieves an equal accuracy to applying the model to the dataset.

Sensitivity is highest for the Average class than Good, suggesting that the model is better at correctly classifying Average values than Good. Poor has a low sensitivity due to the small sample size for this class.

Specificity is highest for the Poor class showing the model correctly identifies observations that do not belong to the Poor class as Good or Average. Average shows the lowest rate of specificity, showing the model is less able to predict if observations belonging to the 'Average' class do not belong to 'Good' or 'Poor' classes.

3.2 Predicting OverallCond Using Random Forest Classifier

Random Forest is a powerful and robust machine learning algorithm. It can be used on classification as well as on regression problems. To understand how it works let's take a real-life scenario; whenever one plans to buy a mobile phone or a laptop one seeks advice/views from other people and adds this to their research. Usually, one takes the opinion of a lot of people because one person could be biased for or against the product and by asking more people this bias can be reduced. In data science, it is called Ensembling. In Ensembling multiple models are trained and their outputs are merged to get the final output using some method (commonly used method in classification is "Vote"). Random Forest is an ensemble model which generates several decision trees. Decision trees are easy to train but are weak models in terms of predicting power and can be easily biased. Random Forest works on several decision tree models and by combining the outputs removes the bias and increases the overall accuracy of the trained model.

Random Forest in R language can be implemented by using the library "randomForest". To get the best estimation of how well our trained model will perform on new data our model can't be trained and evaluated on the same dataset, so data is divided into train set and test set in the ratio 70:30 respectively. Feature scaling doesn't have any positive or negative effect on Random Forest Classifier. Then creating a Random Forest Classifier model with default parameters. There is an option to change default values of parameters such as number of trees(ntree). The default value of "ntree" is 500. Parameter "trcontrol" in Random Forest function is used to create 10 fold cross-validation. Cross validation is done to get a better approximation of the overall accuracy of the trained model. Parameter "importance" is used to calculate the importance of predictors. To evaluate the trained model, the model is made to predict the test set and then different evaluation metrics are calculated. To calculate different evaluation metrics function confusion matrix is used of library "caret". This function gives a confusion matrix, overall accuracy, and some other statistics such as sensitivity, specificity, pos pred value (precision), etc. 84 results from a total of 439 results in the test set were misclassified and the model achieved an overall accuracy of 80.86%.

3.3 Predicting house prices

In this study we approached the prediction of house prices as a regression problem due to the sale price being a continuous variable. Because the housing dataset contains a large number of features on many different scales and several outliers within the sale price, Random Forest was chosen as the first regression algorithm and Support Vector Machines (SVM) the second. Random Forest is an ensemble of decision trees which means it does not need to be normalised, can handle missing values and is robust to outliers as well as reducing the inherent risk of overfitting that comes with a single decision tree.

SVM, specifically Support Vector Regression (SVR) aims to fit the regression errors within an acceptable threshold in order to fit the best hyperplane. It is effective at dealing with a large number of features, as well as being robust to outliers, however, does require the dataset to be scaled.

During the training process for both models, a log transformation is applied to the target variable (Sale Price) to normalise the left skewed distribution (3). When evaluating the model results, the inverse of the log is applied before calculating the Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) and R-Squared scores (1) (4).

	Random Forest	SVR
R-Squared	0.92	0.711
Mean Absolute Error	15971	24267
Root Mean Squared Error	27189	48121
Training Error	-	0.0083

Table 1: Regression results

After training and testing the Random Forest model (2) on the pre-processed dataset, Table 1 shows the evaluation results of the random forest model predictions on the test set. We can see the model achieved an R-Squared score of 0.92 which shows the model is a good fit on the test data. The MAE shows that on average error in the model predictions is £15,971, which when taken in the context of house prices is a relatively small error.

When running the SVR model, one hot encoding, feature scaling and feature selection were applied to the dataset before modelling. The initial run of the model produced a training error of 0.0092 but R-Squared score of 0.355 on the test data. Table 1 shows the evaluation metrics after removing the one hot encoding and feature selection, where we can see the training

error reduced down to 0.0083 and R-Squared increased to 0.711. The MAE shows that on average error in the model predictions is £24,267 which is higher than the Random Forest result.

3.4 Estimating the test error

Following the implementation of the Random Forests and Support Vector Machines, the next stage is to utilise re-sampling methods to estimate the test error associated with fitting these methods. The methods we will employ are cross validation, and bootstrapping. For our evaluation metric, we will use the average R^2 over all folds/bootstraps.

3.4.1 Cross Validation

After applying a tenfold cross validation to the Random Forest Regression, we receive a mean R^2 value of 0.872. This is slightly lower than our original RF score, however as the scores are similar within a reasonable degree this gives us confidence that RF performs well on many different splits of the dataset showing that it is not overfitting on any part of the data.

Following a tenfold cross validation on Support Vector Machine Regression we receive a mean R^2 score of 0.695. Performing this re-sampling gives us further evidence that outliers within the data may be adversely affecting the performance of this method.

3.4.2 Bootstrapping

For our bootstrapping resampling, we have chosen to use 1000 replications to give us an appropriate spread of samples across the dataset to give us a statistically reliable result. Random Forests with bootstrapping returns an R^2 score of 0.854 for the first replication. Appendix 6 shows the bootstrap histogram and the quantiles of the normal distribution, with a dotted line to represent the original R^2 score. The histogram shows that after all the replications, the distribution skews toward a higher R^2 score of around 0.9. This is supported by the mean of the standard normal distribution meeting at 0.9. We can see bootstrapping highlights that our original R^2 value is not accurate of the wider picture of the entire dataset, and that Random Forests performs very well on our data.

For Support Vector Machines, bootstrapping returns an original R^2 score of 0.750, but as replications occur, we can see in figure X+1 that the true distribution of R^2 scores are in fact skewed higher than this. In fact, the distribution has a mean of around 0.85, which is a far

more acceptable score in terms of test error. This re-sampling method highlights that perhaps performing regression on segments of the data rather than a larger portion allows SVM to perform better, giving us further evidence that perhaps outliers in the data are causing SVM to perform significantly worse and that it is in fact a viable option for regression on this data if the outliers were removed.

3.5 Research question – predicting YearBuilt

In this section, we switch our target variable to YearBuilt, with the aim of using Multiple Linear Regression to build a model for predicting the year a house was built. We use the same Boruta feature selection that was employed previously to select key variables, then move onto splitting our data. The data is split into 80 % train set (with 1169 rows) and 20% test set (with 291 rows) to begin our investigation. The plot in Figure X shows the actual versus predicted variable.

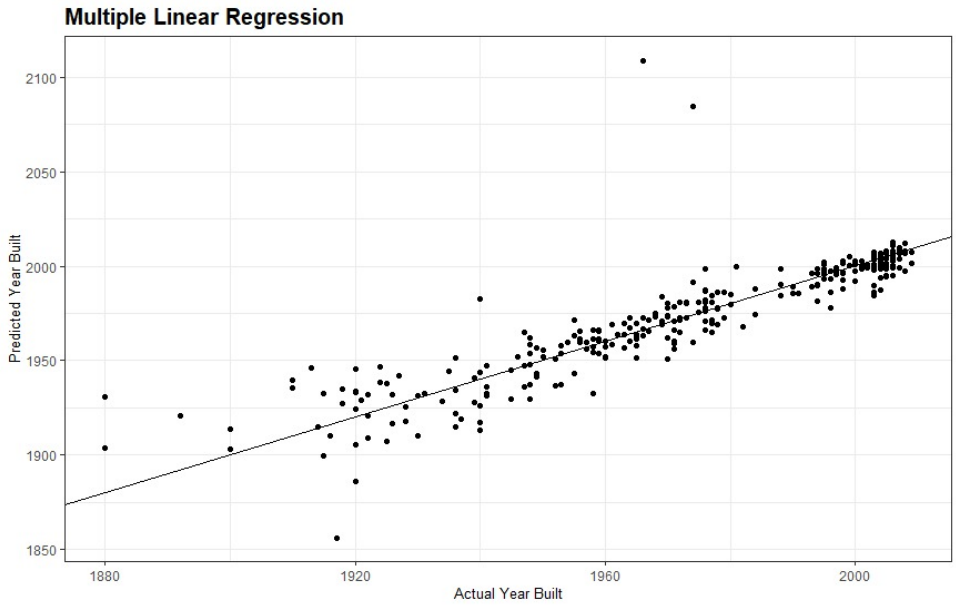


Figure 5: Plot of actual versus predicted year built from the multiple regressor model.

To further evaluate the effectiveness of this model, we use the R-squared metrics. This returns a score of **0.764.** With this score, we can deduce that 76.4% is the fraction by which the errors' variance is less than the dependent variable's variance.

4. Discussion

4.1 Classification of OverallCond

The accuracy of the model after feature selection was performed showed little improvement with a limited number of features suggesting that the model can effectively deal with a high number of features.

The multinomial logistic regression model is shown to have low value due to equal Accuracy/No Information Rate and high p-value. Logistic regression is sensitive to scales and so further experiments to scale and normalise the features can be investigated to create a more valuable model.

The number of Average houses is significantly higher than Poor or Good houses creating a class imbalance. The Model could be modified to implement an oversampling method such as SMOTE to create new observations to achieve an equal ratio of classes - (6). Another method to investigate is stratified sampling or to setting a higher class weight to Poor to penalise the misclassification of this class.

Classification accuracy of Random Forest could have been increased by tuning hyper-parameters such as the number of trees, number of features sampled at each split, etc. Another way of increasing accuracy in the random forest is to add more data but, in this case, this was not possible.

4.2 Predicting house prices and evaluating test error

From the analysis of the model performance, we can see that the Random Forest algorithm performed particularly well in comparison to SVR. This could be down to a number of factors such as not having to choose a certain method to normalise the dataset or the fact that as an ensemble method Random Forest reduces the risk of overfitting.

When training the SVR model with one hot encoding and feature selection applied to the dataset it performed well on the training set with a low training error but performed poorly at predicting on the test set. This indicated the model was struggling with overfitting on the training data and the one hot encoding/feature selection steps were removed to reduce the complexity of the model which improved the model's performance. The final results of SVR were good but had a lot of room for improvement and further investigation into areas such as hyperparameter selection, normalisation methods and outlier removal may lead to a decrease in prediction errors.

When applying the re-sampling methods, we can see that bootstrapping (7) provides a better estimate of the test error due to its higher number of replications, giving a more honest representation of the performance of each of the models. In addition, we can conclude with

confidence that Random Forests perform marginally better than Support Vector Machines when predicting SalePrice on our dataset.

4.3 Research question – Predicting YearBuilt

The house data set is a very flexible data set – although we choose to focus our analysis on YearBuilt for a couple of reasons:

- YearBuilt contained no missing values (no missing labels to train with)
- The same data set could be used – since there is relationships between the other variables and YearBuilt. For example, houses with a certain type of "modern" roof are only built after the year 2000
- It is easily modelled as a regression problem

The feasibility of this is described by the box plots in appendix 7. Certain Garage and House types are most common in certain periods, for example, Basements have been heavily phased out.

The algorithm we chose to use, Multiple Linear Regression, is often cited as one of the most interpretable models – assigning each independent variable in the data set a co-efficient that explains the linear relationship between that variable and the YearBuilt. We prefer this model for the precise understanding of each individual factor and its effect on the outcome.

5. Conclusion

In conclusion, we have thoroughly investigated the housing dataset and implemented a number of pre-processing steps including imputing missing values, dealing with multicollinearity and normalising skewed distributions through the use of methods such as log transformation. Following this we have successfully classified the overall condition of the houses through methods such as Multinomial Logistic Regression and Random Forest, achieving accuracy rates of 82% and 80.86% respectively.

As well as performing classification, we performed regression analysis on the Sale price of houses utilising Random Forest and Support Vector Regression to successfully predict the price of houses with the best model achieving an average error of £15,971. Finally, we developed and investigated a research question into whether the year a house was built could be predicted using the dataset. Multiple Linear Regression was used, and the model achieved an R-Squared score of 0.764 showing that our model was a relatively good fit on the test data.

6. References

1. Documentation for tidymodels. Available here:
https://www.tidymodels.org/start/models/

2. Documentation for parsnip. Available here:
https://parsnip.tidymodels.org/index.html

3. Documentation for pre-processing data with recipes. Available here:
https://www.tidymodels.org/start/recipes/

4. Documentation for yardstick. Available here
https://yardstick.tidymodels.org/

5. Multinomial Logistic Regression Using R, Mohit Sharma, December 2018. Available here:
https://datasciencebeginners.com/2018/12/20/multinomial-logistic-regression-using-r/

6. Dealing with Imbalanced Data, Tara Boyle, February 2019. Available here:
https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18

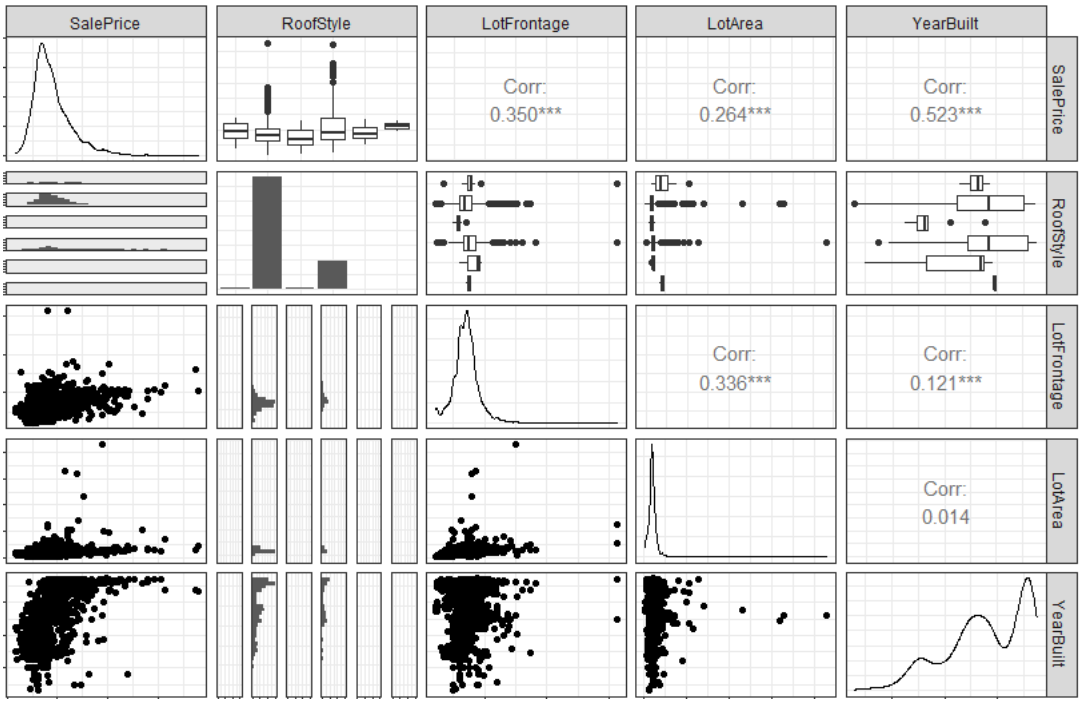
7. Bootstrapping documentation code. Available here:
https://www.statmethods.net/advstats/bootstrapping.html

8. Feature Selection – Ten Effective Techniques with Examples, Selva Prabhakaran.
Available here:
https://www.machinelearningplus.com/machine-learning/feature-selection/

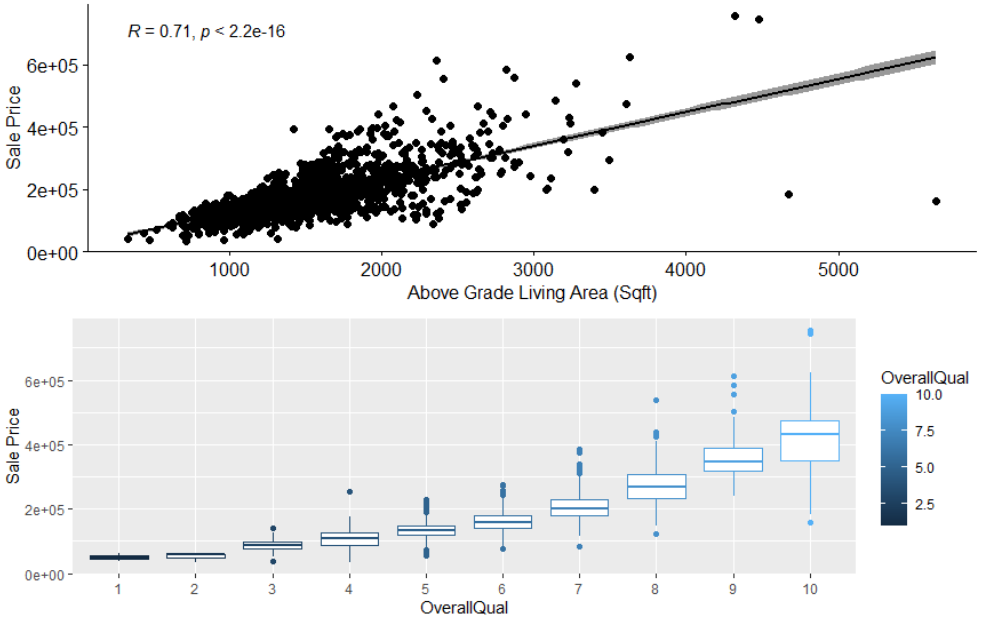
7. Appendix

Group Member	Work completed
Carter Gibson	Q1, compiling report and code plus testing
Laura Osede	Q4
Jake Teague	Q3b, feature selection
Gurleen Oberoi	Q2b
Ryan O'Missenden	Q3a, compiling report and code plus testing, creation of powerpoint
Stephanie Ruscillo	Q2a

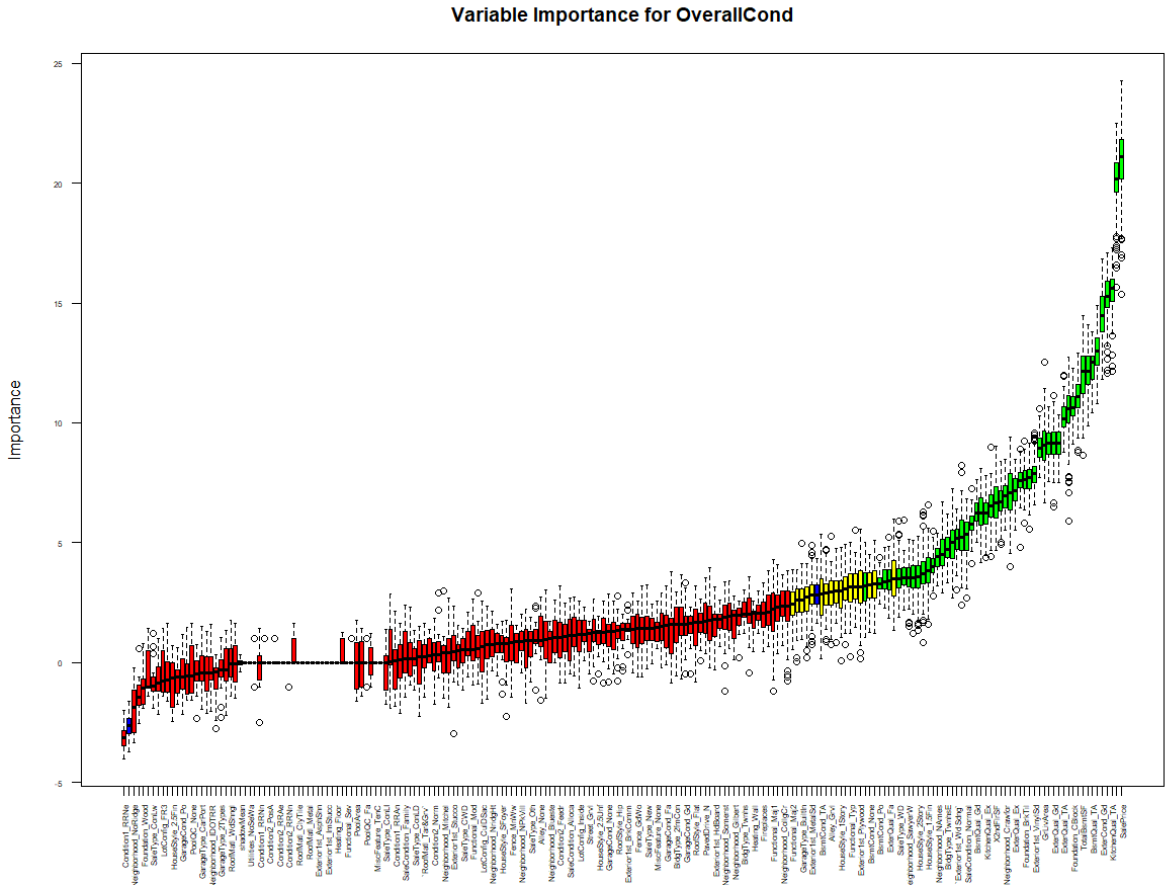
Appendix 1



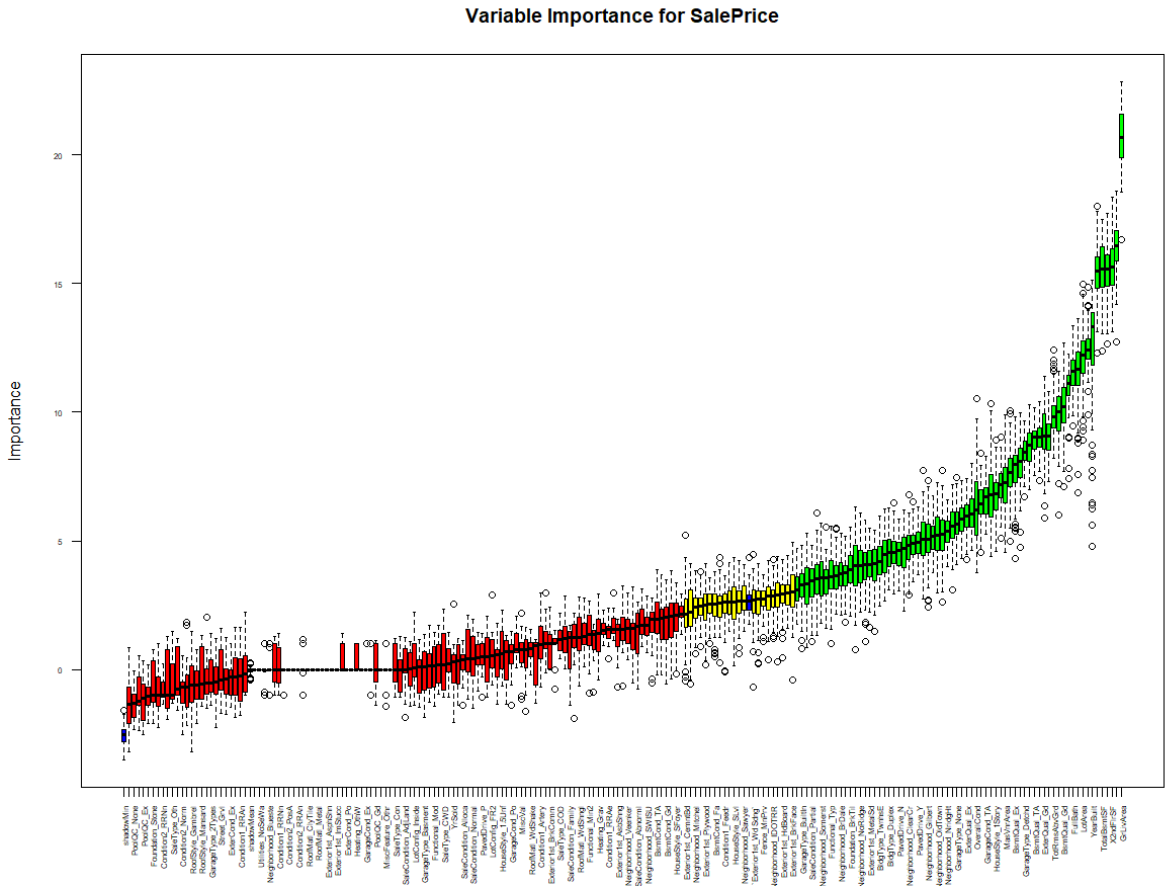
Appendix 2



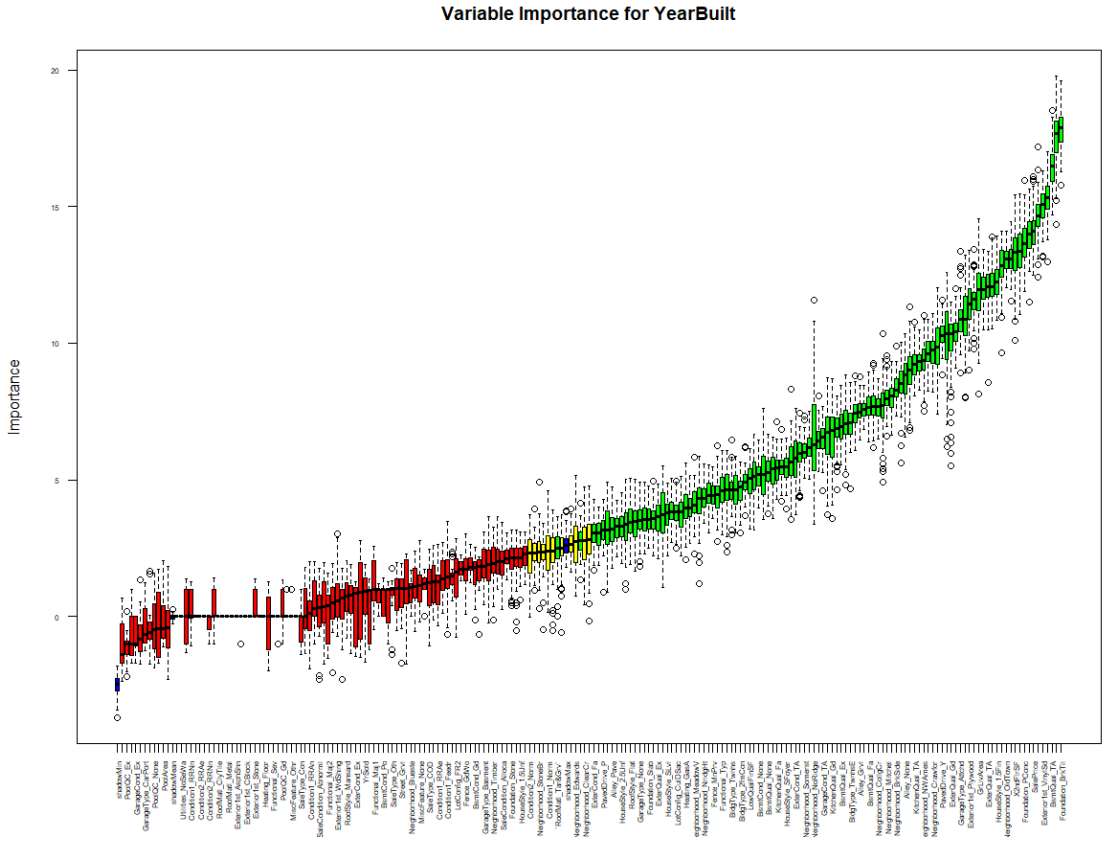
Appendix 3



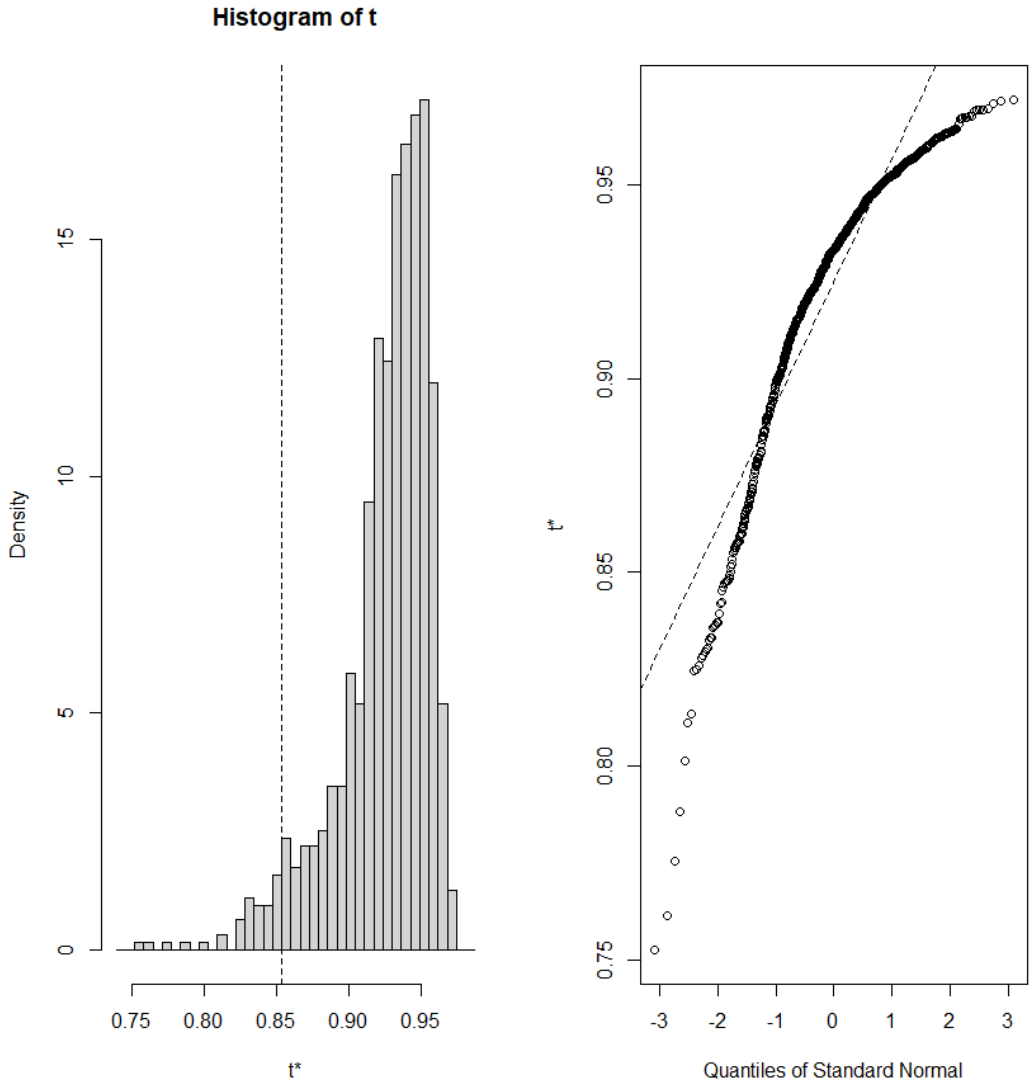
Appendix 4



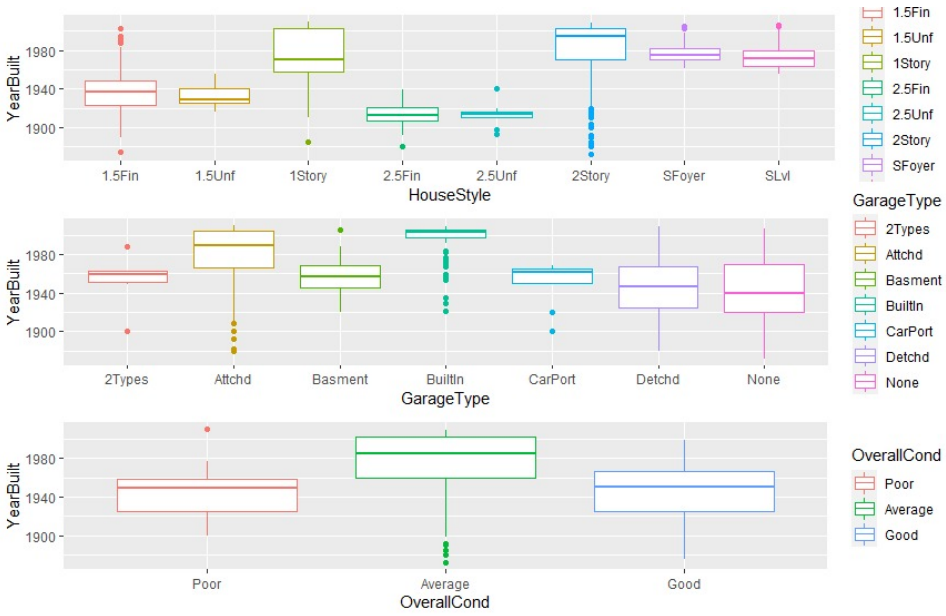
Appendix 5



Appendix 6


Histogram of t
Density
0
5
10
15
0.75
0.80
0.85
0.90
0.95
t*
t*
0.75
0.80
0.85
0.90
0.95
-3
-2
-1
0
1
2
3
Quantiles of Standard Normal

Appendix 7


YearBuilt
1980
1940
1900
1.5Fin
1.5Unf
1Story
2.5Fin
2.5Unf
2Story
SFoyer
SLvl
HouseStyle
1.5Fin
1.5Unf
1Story
2.5Fin
2.5Unf
2Story
SFoyer
GarageType
2Types
Attchd
Basment
BuiltIn
CarPort
Detchd
None
YearBuilt
1980
1940
1900
2Types
Attchd
Basment
BuiltIn
CarPort
Detchd
None
GarageType
GarageType
2Types
Attchd
Basment
BuiltIn
CarPort
Detchd
None
YearBuilt
1980
1940
1900
Poor
Average
Good
OverallCond
OverallCond
Poor
Average
Good

```
########################## Loading Libraries ##########################

library(recipes)
library(tidyverse)
library(ggcorrplot)
library(mltools)
library(data.table)
library(Boruta)
library(writexl)
library(dplyr)
library(readr)
library(olsrr)
library(leaps)
library(nnet)
library(mlogit)
library(rfUtilities)
library(caret)
library(e1071)
library(parsnip)
library(yardstick)
library(e1071)
library(kernlab)
library(rsample)
library(cvTools)
library(randomForest)

########################## LOADING DATA ##########################

data <- read.csv(file = 'house-data.csv', stringsAsFactors = TRUE)
data = subset(data, select = -c(Id))

### Summaries

dim(data)

str(data)

head(data)

########################## EDA (Q1) ##########################

# Missing Value (% of Features)
missing.values <- data %>%
  gather(key = "key", value = "val") %>%
  mutate(isna = is.na(val)) %>%
  group_by(key) %>%
  mutate(total = n()) %>%
  group_by(key, total, isna) %>%
  summarise(num.isna = n()) %>%
  mutate(pct = num.isna / total * 100)

levels <-
  (missing.values  %>% filter(isna == T) %>% arrange(desc(pct)))$key

percentage.plot <- missing.values %>%
  ggplot() +
  geom_bar(aes(x = reorder(key, desc(pct)),
               y = pct, fill=isna),
           stat = 'identity', alpha=0.8) +
  scale_x_discrete(limits = levels) +
  scale_fill_manual(name = "",
                    values = c('White', 'Black'), labels = c("Present", "Missing")) +
  coord_flip() +
  labs(title = "Percentage of missing values", x =
         'Variable', y = "% of missing values")

percentage.plot

# Distribution of SalePrice
```

```
ggplot(data, aes(x=SalePrice)) +
  geom_histogram(binwidth=10000, colour="black", fill="grey") +
  geom_vline(aes(xintercept=mean(SalePrice, na.rm=T)),
             color="red", linetype="dashed", size=1)+
  scale_x_continuous(labels = scales::comma)

# Distribution of Neighborhood
ggplot(data, aes(x=Neighborhood))+
  geom_bar(stat="count", width=0.7, fill="steelblue")+
  theme_minimal()+
  coord_flip()

# Distribution of YearBuilt
ggplot(data, aes(x=YearBuilt)) +
  geom_histogram(binwidth=10, colour="black", fill="grey")

######################### PREPROCESSING #########################

# Impute categorical 'NA' values to new factor 'None' where contextually appropriate
data$PoolQC = factor(data$PoolQC, levels=c(levels(data$PoolQC), 'None'))
data$PoolQC[is.na(data$PoolQC)] = 'None'

data$MiscFeature = factor(data$MiscFeature, levels=c(levels(data$MiscFeature), 'None'))
data$MiscFeature[is.na(data$MiscFeature)] = 'None'

data$Alley = factor(data$Alley, levels=c(levels(data$Alley), 'None'))
data$Alley[is.na(data$Alley)] = 'None'

data$Fence = factor(data$Fence, levels=c(levels(data$Fence), 'None'))
data$Fence[is.na(data$Fence)] = 'None'

data$GarageCond = factor(data$GarageCond, levels=c(levels(data$GarageCond), 'None'))
data$GarageCond[is.na(data$GarageCond)] = 'None'

data$GarageType = factor(data$GarageType, levels=c(levels(data$GarageType), 'None'))
data$GarageType[is.na(data$GarageType)] = 'None'

data$BsmtQual = factor(data$BsmtQual, levels=c(levels(data$BsmtQual), 'None'))
data$BsmtQual[is.na(data$BsmtQual)] = 'None'

data$BsmtCond = factor(data$BsmtCond, levels=c(levels(data$BsmtCond), 'None'))
data$BsmtCond[is.na(data$BsmtCond)] = 'None'

# Impute MasVnrArea with mean of feature
data$MasVnrArea[is.na(data$MasVnrArea)] <- mean(data$MasVnrArea, na.rm = TRUE)

# Impute LotFrontage with median of neigbourhood
data$LotFrontage[is.na(data$LotFrontage)] <- with(data, ave(LotFrontage, Neighborhood,
                                          FUN = function(x) median(x, na.rm = TRUE)))[is.na(data$LotFrontage)]

# Correlation Matrix
corr_data <- select_if(data, is.numeric)

corr <- round(cor(corr_data), 1)

ggcorrplot(corr, hc.order = TRUE, type = "lower",
           outline.col = "white",
           ggtheme = ggplot2::theme_gray,
           colors = c("#6D9EC1", "white", "#E46726"))


# Bar plots
bar1 <- ggplot(data, aes(x=Alley))+
  geom_bar(stat="count", width=0.7)+
  theme_minimal()

bar2 <- ggplot(data, aes(x=HouseStyle))+
  geom_bar(stat="count", width=0.7)+
  theme_minimal()
```

```
bar3 <- ggplot(data, aes(x=LotConfig))+
  geom_bar(stat="count", width=0.7)+
  theme_minimal()

library('gridExtra')
g <-grid.arrange(bar1, bar2, bar3)

#Captures the data before one hot encoding for a later SVM model
svm_data <- data

# One Hot Encoding for Categorical variables
data <- one_hot(as.data.table(data))

# Group data by OverallCond
data$OverallCond <- cut(data$OverallCond,
                        c(-Inf, 3, 6, Inf),
                        labels = c('Poor', 'Average', 'Good'))

# Create non ordinal OverallCond for purpose of logistic regression
data$OverallCond <- factor( data$OverallCond, ordered = FALSE )

########################## MODELLING - CONDITION (Q2) ##########################

## Feature Selection
boruta_output2 <- Boruta(OverallCond ~ ., data=data, doTrace=0)

#prints confirmed and tentative variables
boruta_signif2 <- getSelectedAttributes(boruta_output2, withTentative = TRUE)
print(boruta_signif2)

#tests whether tentative variables are significant or not
roughFixMod2 <- TentativeRoughFix(boruta_output2)
boruta_signif2 <- getSelectedAttributes(roughFixMod2)
print(boruta_signif2)

#extracts confirmed important variables and orders them in descending order
impsx <- attStats(roughFixMod2)
impsx2 = impsx[impsx$decision != 'Rejected', c('meanImp', 'decision')]
impsx2[order(-impsx2$meanImp), ]

#plot of variable importance
plot(boruta_output2, cex.axis=.5, las=2, xlab="", main="Variable Importance for OverallCond")

row.names(impsx2)

#subset of important variables only
OverallCondRelevantData = subset(data, select = c("OverallCond", "LotFrontage", "LotArea", "Alley_Grvl",
"Neighborhood_BrkSide", "Neighborhood_CollgCr", "Neighborhood_Crawfor", "Neighborhood_NAmes",
"Neighborhood_NWAmes", "Neighborhood_OldTown", "Neighborhood_SawyerW", "Condition1_PosA",
"BldgType_1Fam", "BldgType_Duplex", "BldgType_TwnhsE", "HouseStyle_1.5Fin", "HouseStyle_2Story",
"OverallQual", "YearBuilt", "Exterior1st_MetalSd", "Exterior1st_VinylSd", "Exterior1st_Wd Sdng",
"MasVnrArea", "ExterQual_Ex", "ExterQual_Fa", "ExterQual_Gd", "ExterQual_TA", "ExterCond_Ex",
"ExterCond_Fa", "ExterCond_Gd", "ExterCond_TA", "Foundation_BrkTil", "Foundation_CBlock",
"Foundation_PConc", "BsmtQual_Ex", "BsmtQual_Gd", "BsmtQual_TA", "BsmtQual_None",
"BsmtCond_Po", "BsmtCond_None", "TotalBsmtSF", "X1stFlrSF", "X2ndFlrSF", "GrLivArea", "FullBath",
"BedroomAbvGr", "KitchenAbvGr", "KitchenQual_Ex", "KitchenQual_Gd", "KitchenQual_TA",
"TotRmsAbvGrd", "Functional_Maj2", "Functional_Typ", "Fireplaces", "GarageType_Attchd",
"GarageType_Detchd", "GarageArea", "PavedDrive_Y", "Fence_MnPrv", "Fence_None", "SaleType_WD",
"SaleCondition_Normal", "SaleCondition_Partial", "SalePrice"))

## Train/Test Split

train <- sample(nrow(OverallCondRelevantData), 0.7*nrow(OverallCondRelevantData), replace = FALSE)
cond_training <- OverallCondRelevantData[train,]
cond_testing <- OverallCondRelevantData[-train,]

## Logistic Regression
```

```
cond_training$OverallCond <-relevel(cond_training$OverallCond, ref="Good")

#create repeated kfold cross validation
cond_train.control <- trainControl(method = "cv",
                           number = 10)

model <- train(OverallCond~., data = cond_training, method = "multinom",
               trControl = cond_train.control)


cond_training$predicted <- predict(model, newdata = cond_training) #predict values for the train dataset

cond_train_table <- table(cond_training$OverallCond, cond_training$predicted) #classification table to compare
predicted values v actual values

confusionMatrix(cond_training$OverallCond, cond_training$predicted)


cond_testing$predicted <- predict(model, newdata = cond_testing, "raw")


confusionMatrix(cond_testing$OverallCond, cond_testing$predicted)

## Random Forest
model = randomForest(cond_training$OverallCond ~ ., data = cond_training ,importance = TRUE, trControl =
cond_train.control)

model


predTest <- predict(model, cond_testing, type = "raw")
table(predTest, cond_testing$OverallCond)
mean(predTest == cond_testing$OverallCond)

confusionMatrix(predTest,cond_testing$OverallCond)

########################## MODELLING - HOUSE PRICES (Q3a) ##########################
## Feature Selection

boruta_output <- Boruta(SalePrice ~ ., data=svm_data, doTrace=0)

#prints confirmed and tentative variables
boruta_signif <- getSelectedAttributes(boruta_output, withTentative = TRUE)
print(boruta_signif)

#tests whether tentative variables are significant or not
roughFixMod <- TentativeRoughFix(boruta_output)
boruta_signif <- getSelectedAttributes(roughFixMod)
print(boruta_signif)

#extracts confirmed important variables and orders them in descending order
imps <- attStats(roughFixMod)
imps2 = imps[imps$decision != 'Rejected', c('meanImp', 'decision')]
imps2[order(-imps2$meanImp), ]

#plot of variable importance
plot(boruta_output, cex.axis=.5, las=2, xlab="", main="Variable Importance for SalePrice")

row.names(imps2)

#subset of important variables only
```

```
SalePriceRelevantData = subset(data, select = c("SalePrice", "LotFrontage", "LotArea", "Alley_Grvl",
"Alley_Pave", "Alley_None", "Neighborhood_BrDale", "Neighborhood_ClearCr", "Neighborhood_CollgCr",
"Neighborhood_Crawfor", "Neighborhood_Edwards", "Neighborhood_Gilbert", "Neighborhood_IDOTRR",
"Neighborhood_MeadowV", "Neighborhood_NAmes", "Neighborhood_NoRidge", "Neighborhood_NPkVill",
"Neighborhood_NridgHt", "Neighborhood_NWAmes", "Neighborhood_OldTown", "Neighborhood_Sawyer",
"Neighborhood_Somerst", "Condition1_Norm", "BldgType_1Fam", "BldgType_Duplex", "BldgType_Twnhs",
"BldgType_TwnhsE", "HouseStyle_1.5Fin", "HouseStyle_1Story", "HouseStyle_2Story", "HouseStyle_SLvl",
"OverallQual", "OverallCond", "YearBuilt", "RoofStyle_Hip", "Exterior1st_HdBoard", "Exterior1st_MetalSd",
"Exterior1st_Plywood", "Exterior1st_VinylSd", "MasVnrArea", "ExterQual_Ex", "ExterQual_Fa",
"ExterQual_Gd", "ExterQual_TA", "Foundation_BrkTil", "Foundation_CBlock", "Foundation_PConc",
"Foundation_Slab", "BsmtQual_Ex", "BsmtQual_Fa", "BsmtQual_Gd", "BsmtQual_TA", "BsmtQual_None",
"BsmtCond_Fa", "BsmtCond_None", "TotalBsmtSF", "X1stFlrSF", "X2ndFlrSF", "GrLivArea", "FullBath",
"BedroomAbvGr", "KitchenAbvGr", "KitchenQual_Ex", "KitchenQual_Fa", "KitchenQual_Gd",
"KitchenQual_TA", "TotRmsAbvGrd", "Functional_Typ", "Fireplaces", "GarageType_Attchd",
"GarageType_BuiltIn", "GarageType_Detchd", "GarageType_None", "GarageArea", "GarageCond_TA",
"GarageCond_None", "PavedDrive_N", "PavedDrive_Y", "SaleType_New", "SaleCondition_Partial"))

## Train/Test Split

set.seed(4595)
#Create train/test split on the relevant data only
train_test_split <- initial_split(SalePriceRelevantData, strata = "SalePrice", prop = 0.8)

#Create training and test sets
price_train <- training(train_test_split)
price_test  <- testing(train_test_split)

## Random Forest

#Set model hyperparameters
rf_model <- rand_forest(mode = "regression", mtry = 10, min_n = 3, trees = 2000)

#Train the model by fitting X and Y and performing log transformation on SalePrice
rf_fit_model <-
  rf_model %>%
  fit_xy(
    x = price_train[,-1],
    y = log10(price_train$SalePrice)
  )

#Print model information
rf_fit_model

#Use the model to predict on the test set, applying log transformation to test set
rf_model_results <-
  price_test %>%
  dplyr::select(SalePrice) %>%
  mutate(SalePrice = log10(SalePrice)) %>%
  bind_cols(
    predict(rf_fit_model, new_data = price_test[,-1])
  )

#Print the first ten rows of ground truth and model prediction
rf_model_results %>% slice(1:10)
#Print RMSE, MAE and R-Squared metrics to evaluate the model
rf_model_results %>% metrics(truth = SalePrice, estimate = .pred)

#Inverse the log transformation and print results
rf_inverse_log = 10^rf_model_results
rf_inverse_log %>% slice(1:10)
rf_inverse_log %>% metrics(truth = SalePrice, estimate = .pred)

## SVM

set.seed(4595)

#Creates a test/train split using the SVM data without one hot encoding
train_test_split <- initial_split(svm_data, strata = "SalePrice", prop = 0.8)
```

```
#Create training and test sets specifically for SVM
svr_price_train <- training(train_test_split)
svr_price_test  <- testing(train_test_split)

#Center and scale all numeric values (except Sale Price) using the training data
svr_scale <-
  recipe(SalePrice ~ ., data = svr_price_train) %>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes()) %>%
  prep()

#Extract the scaled training data
price_train_scaled = juice(svr_scale)
#Fit and transform the test set using the scaling metrics from the training set
price_test_scaled = bake(svr_scale, svr_price_test)

#Selecting SVM radial basis function and hyperparameters
svm_model <- svm_rbf(mode = 'regression', cost = 10, rbf_sigma = 0.1)

#Fitting model to X_train, Y_train and taking log of Sale Price for better predictions
svm_fit_model <-
  svm_model %>%
  fit_xy(
    x = price_train_scaled[,0:49],
    y = log10(price_train$SalePrice)
  )

#Model fit results
svm_fit_model

#Use the model to predict on the test set, applying log transformation to test set
svm_model_results <-
  price_test %>%
  dplyr::select(SalePrice) %>%
  mutate(SalePrice = log10(SalePrice)) %>%
  bind_cols(
    predict(svm_fit_model, new_data = price_test_scaled[,0:49])
  )

#Print the first ten rows of ground truth and model prediction
svm_model_results %>% slice(1:10)

#Print RMSE, MAE and R-Squared metrics to evaluate the model
svm_model_results %>% metrics(truth = SalePrice, estimate = .pred)

#Inverse the log transformation and print results
svm_inverse_log = 10^svm_model_results
svm_inverse_log %>% slice(1:10)
svm_inverse_log %>% metrics(truth = SalePrice, estimate = .pred)

########################## RESAMPLING (Q3b) ##########################

## Cross Validation

folds <- cvFolds(NROW(data), K=10) #creates 10 folds from the data

rf_model <- rand_forest(mode = "regression", mtry = 10, min_n = 3, trees = 2000) #our model

rfallresults <- data.frame(metric = c('rmse', 'rsq', 'mae')) #creates dataframe to save results in

for(i in 1:10){
  data_train <- SalePriceRelevantData[folds$subsets[folds$which != i], ] #Set the training set
  data_test <- SalePriceRelevantData[folds$subsets[folds$which == i], ] #Set the validation set

  rf_fit_model <-
    rf_model %>%
    fit_xy(
      x = data_train[,-1],
      y = log10(data_train$SalePrice)
```

```
    )

  rf_model_results <-
    data_test %>%
    dplyr::select(SalePrice) %>%
    mutate(SalePrice = log10(SalePrice)) %>%
    bind_cols(
      predict(rf_fit_model, new_data = data_test[,-1])
    )

  # Inverse the log transformation and print results
  rf_inverse_log = 10^rf_model_results
  results <- rf_inverse_log %>% metrics(truth = SalePrice, estimate = .pred) #saves results of this fold
  rfallresults <- cbind(rfallresults, results[,3]) #appends the results to the dataframe
}

rfallresults

# Calculate the mean of each fold
rmsemean <- rowMeans(rfallresults[1,-1])
rsqmean <- rowMeans(rfallresults[2,-1])
maemean <- rowMeans(rfallresults[3,-1])

# Final table displaying mean of each metric for all folds
rfmeanresults <- data.frame(metric = c('rmse', 'rsq', 'mae'), mean = c(format(rmsemean, digits = 3),
format(rsqmean, digits = 3), format(maemean, digits = 3)))
rfmeanresults


## Support Vector Regression

# Selecting SVM radial basis function and hyperparameters
folds <- cvFolds(NROW(svm_data), K=10) #creates 10 folds from the data

svm_model <- svm_rbf(mode = 'regression', cost = 10, rbf_sigma = 0.1)

svmallresults <- data.frame(metric = c('rmse', 'rsq', 'mae')) #creates dataframe to save results in

for(i in 1:10){
  data_train <- svm_data[folds$subsets[folds$which != i], ] #Set the training set
  data_test <- svm_data[folds$subsets[folds$which == i], ] #Set the validation set

  svm_fit_model <-
    svm_model %>%
    fit_xy(
      x = data_train[,0:49],
      y = log10(data_train$SalePrice)
    )

  svm_model_results <-
    data_test %>%
    dplyr::select(SalePrice) %>%
    mutate(SalePrice = log10(SalePrice)) %>%
    bind_cols(
      predict(svm_fit_model, new_data = data_test[,0:49])
    )

  # Inverse the log transformation and print results
  svm_inverse_log = 10^svm_model_results
  results <- svm_inverse_log %>% metrics(truth = SalePrice, estimate = .pred) #saves results of this fold
  svmallresults <- cbind(svmallresults, results[,3]) #appends the results to the dataframe
}

svmallresults
# Calculate the mean of each fold
rmsemean <- rowMeans(svmallresults[1,-1])
rsqmean <- rowMeans(svmallresults[2,-1])
maemean <- rowMeans(svmallresults[3,-1])
```

```
# Final table displaying mean of each metric for all folds
svmmeanresults <- data.frame(metric = c('rmse', 'rsq', 'mae'), mean = c(format(rmsemean, digits = 3),
format(rsqmean, digits = 3), format(maemean, digits = 3)))
svmmeanresults


## Bootstrapping - Random Forest
library(boot)

rf_model <- rand_forest(mode = "regression", mtry = 10, min_n = 3, trees = 2000) #our model

# Function that returns the rsquared score of the random forest method
rsqRF <- function(data, indices) {
  datasample <- data[indices,] #lets the bootstrap take a sample of the data

  train_test_split <- initial_split(datasample, strata = "SalePrice", prop = 0.8)
  data_train <- training(train_test_split)
  data_test  <- testing(train_test_split)

  rf_fit_model <- rf_model %>%
    fit_xy(
      x = data_train[,-1],
      y = log10(data_train$SalePrice)
    )

  rf_model_results <-
    data_test %>%
    dplyr::select(SalePrice) %>%
    mutate(SalePrice = log10(SalePrice)) %>%
    bind_cols(
      predict(rf_fit_model, new_data = data_test[,-1])
    )

  rf_inverse_log = 10^rf_model_results
  metric <- rf_inverse_log %>% metrics(truth = SalePrice, estimate = .pred)
  rsq <- rowMeans(metric[2,3])
  return(rsq)
}

# Testing the function on a sample of the first 100 rows of the data
rsqRF(data, 1:100)

# Bootstrap with 1000 samples
resultsRF <- boot(data, rsqRF, R=1000)
resultsRF
boot.ci(resultsRF, type = "basic")
plot(resultsRF)

## Bootstrapping - SVM

svm_model <- svm_rbf(mode = 'regression', cost = 10, rbf_sigma = 0.1)

# Function that returns the rsquared score of the SVM method
rsqSVM <- function(svm_data, indices) {
  datasample <- svm_data[indices,] #lets the bootstrap take a sample of the data

  train_test_split <- initial_split(datasample, strata = "SalePrice", prop = 0.8)
  data_train <- training(train_test_split)
  data_test  <- testing(train_test_split)

  svm_fit_model <- svm_model %>%
    fit_xy(
      x = data_train[,1:49],
      y = log10(data_train$SalePrice)
    )

  svm_model_results <-
    data_test %>%
    dplyr::select(SalePrice) %>%
```

```
    mutate(SalePrice = log10(SalePrice)) %>%
    bind_cols(
      predict(svm_fit_model, new_data = data_test[,1:49])
    )

  svm_inverse_log = 10^svm_model_results
  metric <- svm_inverse_log %>% metrics(truth = SalePrice, estimate = .pred)
  rsq <- rowMeans(metric[2,3])
  return(rsq)
}


# Bootstrap with 1000 samples
resultsSVM <- boot(svm_data, rsqSVM, R=1000)
resultsSVM
boot.ci(resultsSVM, type = "basic")
plot(resultsSVM)

######################### MODELLING - YEAR BUILT (Q4) ########################

## Feature Selection

boruta_output3 <- Boruta(YearBuilt ~ ., data=data, doTrace=0)

#prints confirmed and tentative variables
boruta_signif3 <- getSelectedAttributes(boruta_output3, withTentative = TRUE)
print(boruta_signif3)

#tests whether tentative variables are significant or not
roughFixMod3 <- TentativeRoughFix(boruta_output3)
boruta_signif3 <- getSelectedAttributes(roughFixMod3)
print(boruta_signif3)

#extracts confirmed important variables and orders them in descending order
impsxx <- attStats(roughFixMod3)
impsx3 = impsxx[impsxx$decision != 'Rejected', c('meanImp', 'decision')]
impsx3[order(-impsx3$meanImp), ]

#plot of variable importance
plot(boruta_output3, cex.axis=.5, las=2, xlab="", main="Variable Importance for YearBuilt")

row.names(impsx3)

## Train Test Split
YearBuiltRelevantData = subset(data, select = c("YearBuilt", "LotFrontage", "LotArea", "Alley_Grvl",
"Alley_Pave", "Alley_None", "LotConfig_CulDSac", "LotConfig_Inside", "Neighborhood_BrDale",
"Neighborhood_BrkSide", "Neighborhood_ClearCr", "Neighborhood_CollgCr", "Neighborhood_Crawfor",
"Neighborhood_Gilbert", "Neighborhood_IDOTRR", "Neighborhood_MeadowV", "Neighborhood_Mitchel",
"Neighborhood_NAmes", "Neighborhood_NoRidge", "Neighborhood_NridgHt", "Neighborhood_NWAmes",
"Neighborhood_OldTown", "Neighborhood_Sawyer", "Neighborhood_SawyerW", "Neighborhood_Somerst",
"Neighborhood_SWISU", "Condition1_Artery", "Condition1_Norm", "Condition1_PosA", "BldgType_1Fam",
"BldgType_2fmCon", "BldgType_Duplex", "BldgType_Twnhs", "BldgType_TwnhsE", "HouseStyle_1.5Fin",
"HouseStyle_1Story", "HouseStyle_2.5Fin", "HouseStyle_2.5Unf", "HouseStyle_2Story",
"HouseStyle_SFoyer", "HouseStyle_SLvl", "OverallQual", "OverallCond", "RoofStyle_Flat",
"RoofStyle_Gable", "RoofStyle_Hip", "RoofMatl_CompShg", "Exterior1st_AsbShng", "Exterior1st_BrkFace",
"Exterior1st_CemntBd", "Exterior1st_HdBoard","Exterior1st_MetalSd", "Exterior1st_Plywood",
"Exterior1st_Stucco", "Exterior1st_VinylSd", "MasVnrArea", "ExterQual_Ex", "ExterQual_Gd",
"ExterQual_TA", "ExterCond_Fa", "ExterCond_Gd", "ExterCond_TA", "Foundation_BrkTil",
"Foundation_CBlock", "Foundation_PConc", "Foundation_Slab", "BsmtQual_Ex", "BsmtQual_Fa",
"BsmtQual_Gd", "BsmtQual_TA", "BsmtQual_None", "BsmtCond_Fa", "BsmtCond_None", "TotalBsmtSF",
"Heating_GasA", "Heating_GasW", "X1stFlrSF", "X2ndFlrSF", "LowQualFinSF", "GrLivArea", "FullBath",
"BedroomAbvGr", "KitchenAbvGr", "KitchenQual_Ex", "KitchenQual_Fa", "KitchenQual_Gd",
"KitchenQual_TA", "TotRmsAbvGrd", "Functional_Typ", "Fireplaces", "GarageType_Attchd",
"GarageType_BuiltIn", "GarageType_Detchd", "GarageType_None", "GarageArea", "GarageCond_Fa",
"GarageCond_TA", "GarageCond_None", "PavedDrive_N", "PavedDrive_P", "PavedDrive_Y",
"Fence_MnPrv", "Fence_None", "SaleType_New", "SaleType_WD", "SaleCondition_Normal",
"SaleCondition_Partial", "SalePrice"))

names(YearBuiltRelevantData)[names(YearBuiltRelevantData) == "YearBuilt"] <- "Target"
```

```
set.seed(4595)

train_test_split <- initial_split(YearBuiltRelevantData, prop = 0.8)

data_train <- training(train_test_split)
data_test  <- testing(train_test_split)

## Multiple Linear Regression

# Training
regressor_lm = lm(formula = Target ~ .,
                  data = data_train)
print(regressor_lm)

summary(regressor_lm)

# Testing
y_pred_lm = predict(regressor_lm, newdata = data_test)


Pred_Actual_lm <- as.data.frame(cbind(Prediction_lm = y_pred_lm, Actual_lm = data_test$Target))

# Plot
gg.lm <- ggplot(Pred_Actual_lm, aes(Actual_lm, Prediction_lm )) +
  geom_point() + theme_bw() + geom_abline() +
  labs(title = "Multiple Linear Regression", x = "Actual Year Built",
       y = "Predicted Year Built") +
  theme(plot.title = element_text(family = "Lucida Sans", face = "bold", size = (15)),
        axis.title = element_text(family = "Lucida Sans", size = (10)))
gg.lm

#Calculate and print Mean Squared Error
MSE.lm <- sum((data_test$Target - y_pred_lm)^2)/nrow(data_test)
print(paste("Mean Squared Error (Multiple Linear Regression):", MSE.lm))

#Calculate and print R-Squared
R2.lm <- R2(data_test$Target, y_pred_lm, form = "traditional")
print(paste("R-square error  (Multiple Linear Regression):", R2.lm))
```