Data Cleaning, Data Imputation, Feature Transformation and Dimensionality Reduction for Housing Data

Wei Sern Thum, Yue Tan

May 5, 2018

1 Introduction

In the real world, collected data is not complete and complex in nature. This means that the presence of large numbers of missing data and huge amounts of predictor variables are inevitable. With that in mind, this means that we cannot use conventional methods such as deleting data rows with missing values because we will end up deleting a massive number of rows which will not produce valuable results. On top of that, most data are recorded in non-standardized formats where machine learning learning models cannot interpret them in a meaningful way to produce accurate predictions. One such example is factor levels which are expressed in the form of string variables. Therefore, this paper aims to solve the above-mentioned issues for one such dataset found on Kaggle, a data science platform which has the attributes as described above. First, we perform preliminary data cleaning by intuition derived from the schema for the dataset. This is carried out by replacing missing values for categorical and numerical variables with meaningful values (such as 0 for missing numerical values and "None" for missing categorical values). Next, we utilize more advanced data imputation techniques to estimate missing data for the remaining variable with missing values. Before that can be done, we use simulation techniques to simulate the missing values in the

training data and comparing it with the actual values to find out which data imputation method produces the best outcome (i.e. the smallest Mean Average Error). From the results of the simulation, we adopt the imputation method with the best outcome to replace the values for the missing numerical values in the training dataset. After filling in all the missing values, we transform each non-meaningful data features into useful ones that can be easily interpreted. Two of such techniques are One Hot Encoding and Label Encoding. We shall explain each encoding method, discuss their drawbacks below and show the outcome. Since the encoding method that we chose after careful consideration is the One Hot Encoding method, we have to perform dimensionality reduction to identify which predictor variable explains the most variability for the dependent variable. We trained a Random Forest with 100 trees and used the importance function from the randomForest package. We shall also introduce the general idea of decision trees and how the idea is adopted in the Random Forest. Finally, we fit a simple multiple linear regression model to show the outcomes of our data handling methods.

2 Statistical Analysis

2.1 Preliminary Data Analysis

In this project, the number predictor variables in the housing data is 79, which is huge. Among the 79 predictor variables, 56 of them are numeric variables including: LotFrontage and basement square feet; 23 are factor variables, such as street type and central air conditioning type. Our prediction outcome (Y) will be the house price. Training data consists of 1460 rows and there is at least one missing value per row. Therefore, the process of data cleaning plays a major part in this paper.

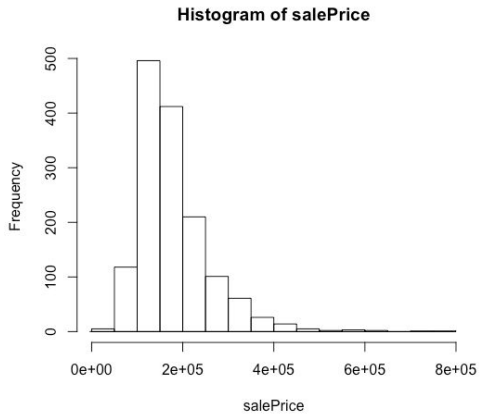

Diagram 2.1 (a)

Visualizing the distribution of house sale price which is the dependent variable, we see that the values are skewed. There seems to be a lot of houses sold between the $100,000 - $250,000 range.

Besides, we found a lot of missing data (NA) values and many of them are factor data, as you can see according to Diagram 2.1(b) and Diagram 2.1 (c), which shows the amount of missing values that we have in our dataset. We need to examine the reason behind the occurrence of NA values and fix them prior to data modeling.

PoolQC	MiscFeature	Alley	Fence	FireplaceQu
1453	1406	1369	1179	690
LotFrontage	GarageType	GarageYrBlt	GarageFinish	GarageQual
259	81	81	81	81
GarageCond	BsmtExposure	BsmtFinType2	BsmtQual	BsmtCond
81	38	38	37	37
BsmtFinType1	MasVnrType	MasVnrArea	Electrical	MSSubClass
37	8	8	1	0

Diagram 2.1 (b)

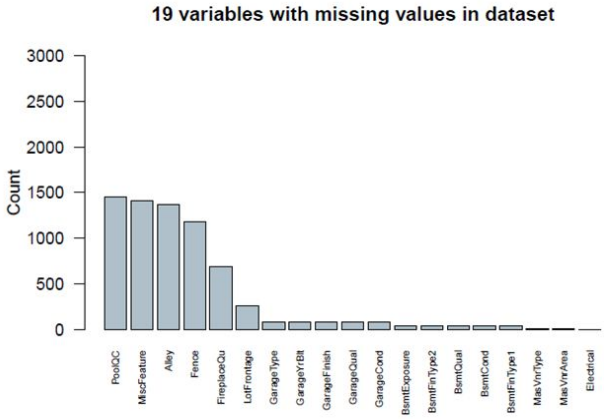

Diagram 2.1 (c)

2.2 Simple Data Cleaning and Imputation by Intuition

Based on the definition of the predictor variables described in the schema, we recognized that many of the missing values were simply because of the absence of particular amenities in the sold house such as fences, alleys et cetera. Therefore, for the numerical variables which had missing values, we set the values equal to zero. For the factor variables that had missing values, we assign a distinct factor name. The following snippet of R code shows the essence of this idea:

```
+ df$PoolQC[df$PoolArea %in% c(0,NA) & is.na(df$PoolQC)] <- "NoPool"
+ df$BsmtUnfSF[df$TotalBsmtSF %in% c(0,NA) & is.na(df$BsmtUnfSF)] <- 0
```

For example, PoolQC represents pool quality and if pool area equals to zero, we assign factor “NoPool”. This is because the house probably does not have any pools. For BsmtUnfSf which represents unfinished square feet of basement area, since the total basement square feet is 0, the house probably does not have a basement, so we set BsmtUnfSF to 0. These assumptions and inferences are based off the schema attached along with the dataset, which describes the attribute of each variable. After preliminary cleaning, there is only one variable left, which is LotFrontage shown as Diagram 2.2(a) and Diagram 2.2(b):

LotFrontage	MSSubClass	MSZoning	LotArea	Street
259	0	0	0	0
Alley	LotShape	LandContour	Utilities	LotConfig
0	0	0	0	0
LandSlope	Neighborhood	Condition1	Condition2	BldgType
0	0	0	0	0
HouseStyle	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	0	0	0	0
RoofStyle	RoofMatl	Exterior1st	Exterior2nd	MasVnrType
0	0	0	0	0

Diagram 2.2(a)

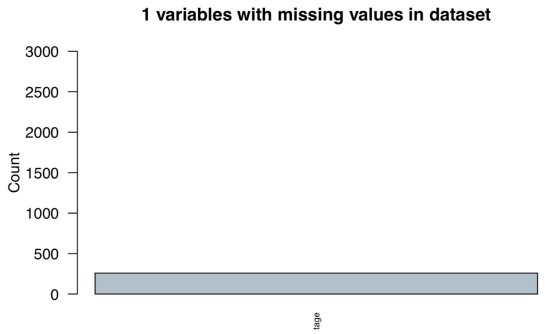

Diagram 2.2(b)

LotFrontage is a numerical variable indicating the linear amount of street in feet connected to the house. This is interesting because we can apply algorithms such as mean imputation and EM to impute numerical data which is continuous in nature.

2.3 Numerical Data Imputation

To impute the values for LotFrontage, we consider two different algorithms - mean imputation and Expectation-Maximization. Before deciding which imputation method to use, we have to find out which algorithm yields a better outcome by doing a simulation study on the combination of the training and testing data. We use a quantitative measure for the accuracy of the algorithm using the Mean Absolute Error for the missing data. According to Sammut and Webb, the notations are as follows: y_i is the true target value for test instance x_i , $\lambda(x_i)$ is the predicted target value for test instance x_i , and n is the number of test instances.

$$mae = \frac{\sum_{i=1}^{n} abs\left(y_i - \lambda(x_i)\right)}{n}$$

However, we do not have the true values for the missing instances of LotFrontage values so we remove those rows to obtain rows with the true LotFrontage values, which is about 2431 rows. We then simulate by setting ~20% of these rows to be missing (approximately the same proportion as the amount of missing rows in the training data). The simulation algorithm will be presented below. We implement the two algorithms as shown below and compare their MAE to find the outcome.

2.3.1 Missing Value Simulation Algorithm

To find the indexes of the rows that we have to delete, we take the floor of the random realized values of the uniform variable ~ [1, 2431]. We then iteratively remove data until we reach 20% of data removed from the data.

The following snippet of code shows the simulation in detail:

```
+ LFvect = dfFull$LotFrontage
+ LFvect = as.vector(LFvect)
+ n = 0.2*length(LFvect)
+ set.seed(40)
+ counter = 0
+ while (n >= counter){
+ rand = runif(1, min = 1, max = length(LFvect))
+ index = floor(rand)
+   if (is.na(LFvect[index]) == FALSE){
+     counter = counter + 1
+    LFvect[index] = NA
+   }
+ }
```

2.3.2 Mean Imputation

According to the lecture notes from Columbia, mean imputation "replaces each missing value with the mean of the observed values for that variable". Using this method, we obtain a MAE value of 16.42308.

2.3.3 Expectation-Maximization (EM) Using Amelia Package

Using the Amelia package in R, we were able to implement the EM algorithm. According to Honaker, King et al., Amelia assumes that the complete data is multivariate normal and the data is MAR (missing at random). This assumption means that the pattern of missingness only depends on the observed data D^{obs}, not the unobserved data D^{mis}. The package then take m draws from the posterior probability of the likelihood function for each of the missing values. Using the default settings of the package, we carry out imputation of LotFrontage based on 4 other variables - Total Basement Square Feet, Above Ground Living Area Square Feet, Garage Area and House Lot Size since these variables show the highest correlation out of all the numerical variables. Using this methodology, we obtain a MAE value of 20.67608.

2.3.3 Imputation Simulation Outcome

Based on the MAE metric that we have specified, mean imputation seems to produce the better outcome. After looking at the LotFrontage distribution of the true values, the reason of this outcome seems to be apparent.

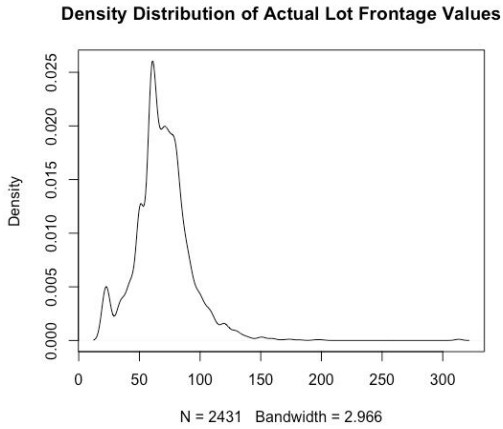

Diagram 2.3.3 (a)

Based on Diagram 2.3.3 (a), we see that the values for LotFrontage are highly clustered between 50 and 100. It is evident that mean imputation will give us smaller values of bias for each observation since less LotFrontage values lie on the extreme.

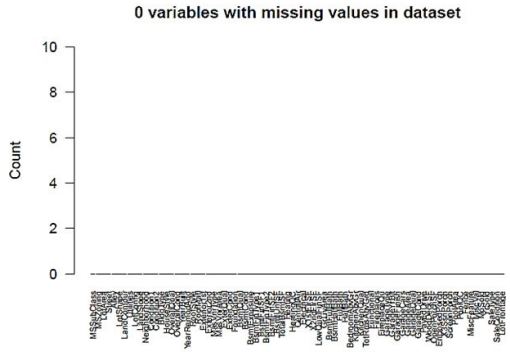

Diagram 2.3.3 (b)

After using the mean imputation method for the missing LotFrontage values, we can see that there are no more missing values in the dataset.

2.4 Feature Transformation

Since the categorical data is expressed in a non-standard form, it must be converted to a numerical format. For instance, for the variable MSSubClass (Type of dwelling), the factor levels do not correspond to the schema. Each factor is numeric and the number represents a type of house. As an example, numeric factor level 90 indicates the house is a duplex but the numeric value for the variable does not reflect this information. Therefore, we introduce two types of feature transformation methods, label encoding and one-hot encoding. Based on the idea, we decided to use one-hot encoding for the purpose of this paper.

2.4.1 Label Encoding

Label encoding converts each level of the factor variable to a new numeric value and assigns distinct values for each of the factor levels for the factor. For example, for variable colors, each string factor level is assigned distinct values, where "red" is changed to numeric values 4, "blue" is changed to 1 et cetera, which shown as Diagram 2.4.1.

```
> colors <- c("red", "blue", "green", "purple")
> factors <- factor(colors)
> as.numeric(factors)
[1] 4 1 2 3
```

Diagram 2.4.1

However, a regression model might interpret the transformed numerical values wrongly, especially if there are many factor levels. Relating to our color example, if red = 4, blue = 1, green = 2 , purple = 3; Does that mean that 4 > 3 > 2 > 1, so factor red is better than purple and so on? Therefore, we notice that label encoding shows an ordering relation when in fact there is none. With this in mind, we move on to the next factor encoding idea.

2.4.2 One Hot Encoding

For qualitative variables where an ordering relation does not exist, we suggest using the One Hot encoding method to convert each level of the factor variable to a new column and assign 0 or 1 to each of the columns to indicate if the factor level is present. For example, there are 15 categories in SubClass variable. We used One-Hot encoding method to substitute each of the factor levels with a binary variable, where value "1" is placed for a factor level column if the factor level exists and value "0" if it does not exist.
Diagram 2.4.2 describes the example:

```
> unique(df$MSSubClass)
 [1] 60  20  70  50  190 45  90  120 30  85  80  160 75  180 40
Levels: 20 30 40 45 50 60 70 75 80 85 90 120 160 180 190

> one_hot(as.data.table(df$MSSubClass))
      V1_20 V1_30 V1_40 V1_45 V1_50 V1_60 V1_70 V1_75 V1_80 V1_85 V1_90 V1_120 V1_160 V1_180 V1_190
   1:     0     0     0     0     0     1     0     0     0     0     0      0      0      0      0
   2:     1     0     0     0     0     0     0     0     0     0     0      0      0      0      0
   3:     0     0     0     0     0     1     0     0     0     0     0      0      0      0      0
   4:     0     0     0     0     0     0     1     0     0     0     0      0      0      0      0
   5:     0     0     0     0     0     1     0     0     0     0     0      0      0      0      0
  ---
1453:     0     0     0     0     0     1     0     0     0     0     0      0      0      0      0
1454:     1     0     0     0     0     0     0     0     0     0     0      0      0      0      0
1455:     0     0     0     0     0     0     1     0     0     0     0      0      0      0      0
1456:     1     0     0     0     0     0     0     0     0     0     0      0      0      0      0
1457:     1     0     0     0     0     0     0     0     0     0     0      0      0      0      0
```

Diagram 2.4.2

2.5 Dimensionality Reduction

After using the one hot encoding method, we are left with too many predictor variables to define a regression model properly as shown in Diagram 2.5 below:

```
> ncol(df)
[1] 79
> ncol(testdf)
[1] 616
```

Diagram 2.5

In order to find out the most significant variable for prediction, we use the randomForest package over all predictor variables to build a random forest over 100 trees. We shall explain the idea of decision trees used to build the random forest below.

2.5.1 Introduction to Decision Trees

A decision tree is a data structure represented is a form of a tree used for classification and regression problems. Each root node represents a single input variable (x) and a split point on that variable. The leaf nodes of the tree contain an output variable (y) which is used to make a prediction. At each node, the decision tree tries to find a criteria that best splits the data into two parts. This data structure can work on both factor variables and numerical variables. One such example of the decision tree is shown in the diagram below:

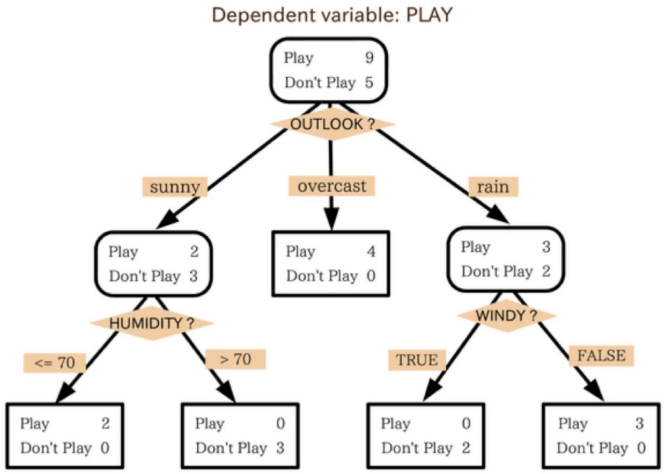

Diagram 2.5.1

To illustrate an example, the above tree tries to classify the prediction to two factors - play or don't play. It first splits based on the outlook of the weather - sunny, overcast or rain. Next, based on the outlook of the weather, it splits based on the humidity and windiness of the weather which clearly classifies the predicted data. This simplified example can be also used on regression type problems.

2.5.2 Introduction to Random Forest

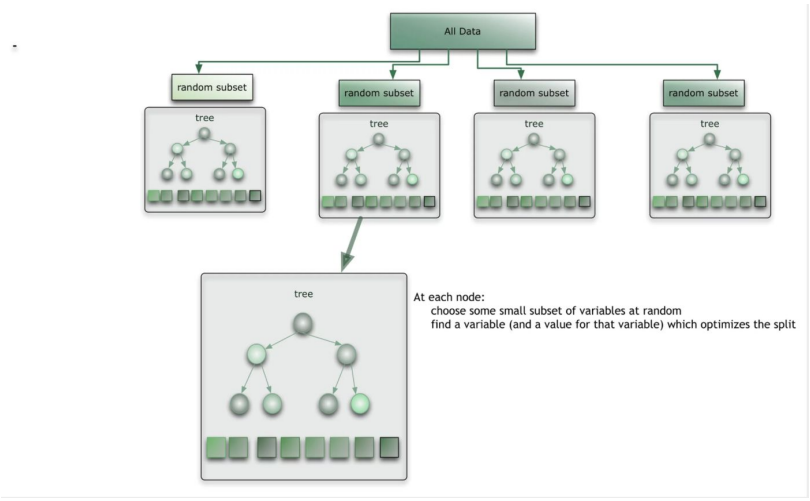

Diagram 2.5.2

Based on Diagram 2.5.2, we see that a random forest is basically an aggregation of a huge number of decision trees. Since a single decision tree is deemed a weak classifier, a random forest is an ensemble method constructing a multitude of decision trees to form a strong classifier. According to Liaw and Wiener in a paper describing the random forest algorithm implementation, the pseudocode is as follows:

1. Draw n bootstrap samples from the original data.
2. For each of the bootstrap samples, grow an unpruned classification or regression tree, with the following modification: at each node, rather than choosing the best split among all predictors, randomly sample mtry of the predictors and choose the best split from among those variables.
3. Predict new data by aggregating the predictions of the n trees (i.e., majority votes for classification, average for regression).

2.5.3 Importance Ranking Using Trained Random Forest

We used the importance function in the randomForest package to rank each of the variables from most important to least important. Based on a paper written by Liaw and Wiener, variable importance is determined by:

1. Mean Squared Error after permuting each of the predictor variables trained on, averaged over all n trees.
2. Residual Sum of Squares from splitting on the node of each predictor variable, averaged over all n trees.

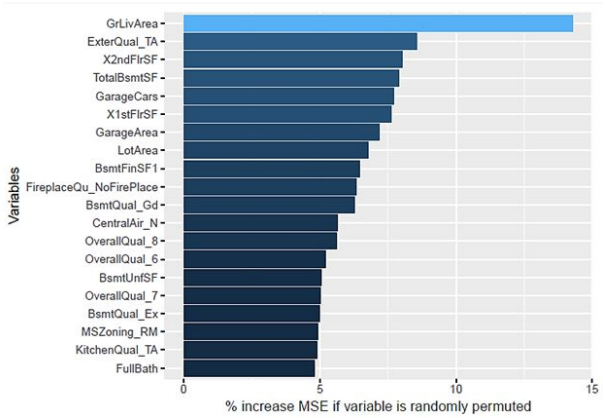

Diagram 2.5.3

We then generated a qqplot to visualize this information and we found that top three important variables are GrLivingArea (Above ground living area square feet), ExterQual_TA (Quality of material of exterior - average) (One hot encoded) and X2ndFlrSF (Second floor square feet).

2.6 Simple Data Modeling

Since the important variables were already transformed into numerical variables, we can use a simple regression model to find out how well the top variables identified by the importance function predicts house sale price. As a result, we used GrLivingArea, ExterQual_TA and X2ndFlrSF as our final predictor variables and use a simple linear regression model output of the lm call as following:

```
Call:
lm(formula = log(salePrice) ~ testdf$GrLivArea + testdf$ExterQual_TA +
    testdf$X2ndFlrSF)

Residuals:
     Min       1Q   Median       3Q      Max
-2.39912 -0.09959  0.02687  0.14900  0.64463

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)          1.144e+01  2.761e-02  414.31   <2e-16 ***
testdf$GrLivArea     5.613e-04  1.741e-05   32.24   <2e-16 ***
testdf$ExterQual_TA -2.956e-01  1.395e-02  -21.19   <2e-16 ***
testdf$X2ndFlrSF    -2.343e-04  1.971e-05  -11.89   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2367 on 1453 degrees of freedom
Multiple R-squared:  0.6498,    Adjusted R-squared:  0.6491
F-statistic: 898.6 on 3 and 1453 DF,  p-value: < 2.2e-16
```

Diagram 2.6 (a)

Our R-squared value shows that our model explains 0.6498 of variability in the dependent variable which shows good signs. Besides, the p-values for all variables are very significant as proven by the R output result in Diagram 2.6 (a).

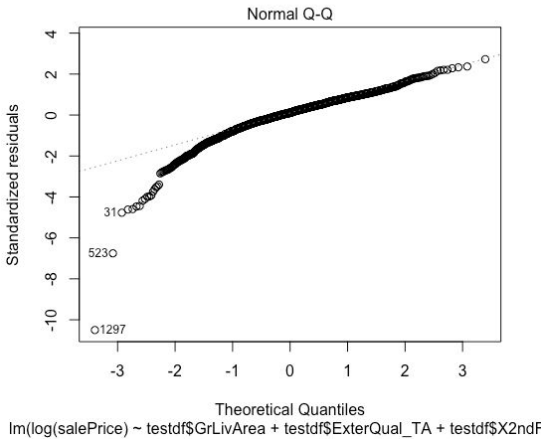

Diagram 2.6 (b)

However, the fitted model does not adhere to the normality assumption perfectly since the distribution of the house sale price is skewed as we can see in Diagram 2.6 (b).

3 Conclusions

In conclusion, the methods that we applied to the data as described in the statistical analysis section seems to work reasonably well as proven using the multiple linear regression fit. This goes to show that no one method works for all cases as we proved in the simulation for numerical imputation. Lastly, there are several improvements that we propose that can be implemented besides the ones mentioned in this paper. First, we can use more advanced prediction techniques to increase prediction accuracy such as weighted predictions from various regression models. For example, we can train a linear regression model and a lasso regression model and then take the weighted average of the each of the predictions. Second, we can train better algorithms to make better predictions. Such examples are neural networks and stacked regression.

4 Bibliography

Gelman, A., & Hill, J. (2016) *Data analysis using regression and multilevel/hierarchical models, chapter 25*. Cambridge: University Press. UTL
http://www.stat.columbia.edu/~gelman/arm/missing.pdf

Moffitt, C. (2017, February 06). *Guide to Encoding Categorical Values in Python*. Retrieved April 28, 2018, Retrieved from http://pbpython.com/categorical-encoding.html

(2011) Mean Absolute Error. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. Retrieved from https://rd.springer.com/referenceworkentry/10.1007%2F978-0-387-30164-8_525

Gaudreau, P. (2017). *House Prices: Dealing with the missing data* | Kaggle. Retrieved from https://www.kaggle.com/clustersrus/house-prices-dealing-with-the-missing-data?scriptVersionId=868965

Bruin, E. (2018, February). *House prices: Lasso, XGBoost, and a detailed EDA* | Kaggle. Retrieved from https://www.kaggle.com/erikbruin/house-prices-lasso-xgboost-and-a-detailed-eda

Honaker, J., King, G., & Blackwell, M. (2015, December 5). *AMELIA II: A Program for Missing Data*. UTL https://cran.r-project.org/web/packages/Amelia/vignettes/amelia.pdf

RNews. (n.d.). Retrieved from https://www.r-project.org/doc/Rnews/Rnews_2002-3.pdf

5 Appendix

Omitted Code From Data Cleaning due to repetitive code
(Note: Histogram plot was referenced from sample kernel that demonstrated how columns with NA rows can be visualized
https://www.kaggle.com/clustersrus/house-prices-dealing-with-the-missing-data?scriptVersionId=868965)

```
## Feature Engineering (OneHot)
library(mltools)
library(data.table)
library(randomForest)
library(ggplot2)
testdf = one_hot(as.data.table(df))
ncol(testdf)
set.seed(2018)
quick_RF <- randomForest(x=testdf[1:1457,], y=salePrice[1:1457], ntree=100,importance=TRUE)
imp_RF <- importance(quick_RF)

/** Taken from sample kernel that demonstrated how importance data can be visualized with qqplot
https://www.kaggle.com/erikbruin/house-prices-lasso-xgboost-and-a-detailed-eda**/
imp_DF <- data.frame(Variables = row.names(imp_RF), MSE = imp_RF[,1])
imp_DF <- imp_DF[order(imp_DF$MSE, decreasing = TRUE),]
ggplot(imp_DF[1:20,], aes(x=reorder(Variables, MSE), y=MSE, fill=MSE)) + geom_bar(stat = 'identity') + labs(x =
'Variables', y= '% increase MSE if variable is randomly permuted') + coord_flip() + theme(legend.position="none")
/** ----------------------                End of taken code                ----------**/

## Modeling Data
```

```
model1 = lm(salePrice ~ testdf$GrLivArea + testdf$ExterQual_TA + testdf$X2ndFlrSF)

## Including Plots
library(Amelia)
df = read.csv("train.csv", header = TRUE)
dfNumeric = cbind(df$LotFrontage, df$LotArea, df$TotalBsmtSF, df$GrLivArea, df$GarageArea)
colnames(dfNumeric) = c("LotFrontage", "LotArea", "TotalBsmtSF", "GrLivArea", "GarageArea")
cor(dfNumeric, use = "complete.obs")
dfPred = cbind(df$LotFrontage, df$LotArea)
colnames(dfPred) = c("LotFrontage", "LotArea")
out = amelia(dfPred, m = 1)
#complete = out$imputations$imp1
temp = as.data.frame(out$imputations$imp1[, 1])
complete = cbind(dfPred, temp)
names(complete)[3] = "EMLotFrontage"
salePrice = df$SalePrice
qqnorm(df$SalePrice)
qqnorm(log(df$SalePrice))
EM1 = lm(complete$LotArea ~ complete$EMLotFrontage)
#EM2 = lm(log(df$SalePrice) ~ complete$EMLotFrontage)
## Find out mean imputation vs multiple imputation
Error = function(x){
  sum = 0
  count = 0
  for (i in 1:nrow(x)){
    if (is.na(x$LotFrontage[i])){
      sum = sum + (abs(x$EMLF[i] - x$ActualLF[i]))
      count = count + 1
    }
  }
  return(sum/count)
}
library(Amelia)
# merge test and training data
df1 = read.csv("train.csv", header = TRUE)
df2 = read.csv("test.csv", header = TRUE)
df1 = subset(df1, select = -c(SalePrice))
testdf = rbind(df1, df2)
# Delete rows where LotFrontage is NA
dfFull = cbind(testdf$LotFrontage, testdf$LotArea, testdf$TotalBsmtSF, testdf$GrLivArea, testdf$GarageArea)
before = nrow(dfFull)
dfFull = as.data.frame(dfFull[complete.cases(dfFull), ])
names(dfFull) = c("LotFrontage", "LotArea", "TotalBsmtSF", "GrLivArea", "GarageArea")
after = nrow(dfFull)
cat("Before:", before, ", After:", after)
full = dfFull$LotFrontage
# Randomly set 20% of the data to NA
LFvect = dfFull$LotFrontage
```

```
LFvect = as.vector(LFvect)
n = 0.2*length(LFvect)
set.seed(40)
counter = 0
while (n >= counter){
  rand = runif(1, min = 1, max = length(LFvect))
  index = floor(rand)
  if (is.na(LFvect[index]) == FALSE){
    counter = counter + 1
    LFvect[index] = NA
  }
}
# Analyze correlation of LotFrontage against other predictors
cor(dfFull)
# All possible numeric predictors using Amelia multiple imputation for imputing LotFrontage
dfFull1 = cbind(as.data.frame(LFvect), dfFull$LotArea, dfFull$TotalBsmtSF, dfFull$GrLivArea,
dfFull$GarageArea)
out = amelia(dfFull1)
temp = as.data.frame(out$imputations$imp1[, 1])
complete1 = cbind(dfFull1, temp)
complete1 = cbind(complete1, as.data.frame(full))
names(complete1) = c("LotFrontage", "LotArea", "TotalBsmtSF", "GrLivArea", "GarageArea", "EMLF",
"ActualLF")
# Only LotArea which has the highest correlation with LotFrontage
dfFull2 = cbind(as.data.frame(LFvect), complete1$LotArea)
out = amelia(dfFull2)
temp = as.data.frame(out$imputations$imp1[, 1])
complete2 = cbind(dfFull2, temp)
complete2 = cbind(complete2, as.data.frame(full))
names(complete2) = c("LotFrontage", "LotArea", "EMLF", "ActualLF")
# Calculating Mean Average Error
Error(complete1)
Error(complete2)
dummy = rep(0, length(LFvect))
# Mean imputation using available LotFrontage values
complete3 = cbind(as.data.frame(LFvect), as.data.frame(dummy), full)
names(complete3) = c("LotFrontage","EMLF", "ActualLF")
mean = mean(complete3$LotFrontage, na.rm = TRUE)
for (i in 1:nrow(complete3)){
  complete3$EMLF[i] = mean
}
Error(complete3)
# Comparing the mean of the available data vs mean of the full data
mean
mean(full)
```