LSE, ST443, Project, Part 1

Group 9

Michealmas Term 2017

Data Cleaning and Transformation

```
library(dplyr)
```

```
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
```

```
train_raw = read.csv("train.csv", row.names = "Id", stringsAsFactors=FALSE)
testing_raw = read.csv("test.csv", row.names = "Id", stringsAsFactors=FALSE)
```

```
#combining train and test data for quicker data prep
testing_raw$SalePrice <- NA
train_raw$isTrain <- 1
testing_raw$isTrain <- 0
df <- rbind(train_raw,testing_raw)
```

Missing Values and imputation.

```
colSums(sapply(df, is.na))
```

```
##    MSSubClass      MSZoning   LotFrontage       LotArea        Street
##             0             4           486             0             0
##         Alley      LotShape   LandContour     Utilities     LotConfig
##          2721             0             0             2             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             1             1            24
##    MasVnrArea     ExterQual     ExterCond    Foundation      BsmtQual
##            23             0             0             0            81
##      BsmtCond  BsmtExposure  BsmtFinType1    BsmtFinSF1  BsmtFinType2
##            82            82            79             1            80
##    BsmtFinSF2     BsmtUnfSF   TotalBsmtSF       Heating     HeatingQC
##             1             1             1             0             0
##    CentralAir    Electrical     X1stFlrSF     X2ndFlrSF  LowQualFinSF
##             0             1             0             0             0
##     GrLivArea  BsmtFullBath  BsmtHalfBath      FullBath      HalfBath
##             0             2             2             0             0
##  BedroomAbvGr  KitchenAbvGr   KitchenQual  TotRmsAbvGrd    Functional
##             0             0             1             0             2
##    Fireplaces   FireplaceQu    GarageType   GarageYrBlt  GarageFinish
```

```
##             0          1420           157           159           159
##    GarageCars    GarageArea    GarageQual    GarageCond    PavedDrive
##             1             1           159           159             0
##    WoodDeckSF   OpenPorchSF EnclosedPorch    X3SsnPorch   ScreenPorch
##             0             0             0             0             0
##      PoolArea        PoolQC         Fence   MiscFeature       MiscVal
##             0          2909          2348          2814             0
##        MoSold        YrSold      SaleType SaleCondition     SalePrice
##             0             0             1             0          1459
##       isTrain
##             0
```

```
df[,c('PoolQC','PoolArea')] %>%
  group_by(PoolQC) %>%
  summarise(mean = mean(PoolArea), counts = n())
```

```
## # A tibble: 4 x 3
##   PoolQC        mean counts
##    <chr>       <dbl>  <int>
## 1     Ex 359.7500000      4
## 2     Fa 583.5000000      2
## 3     Gd 648.5000000      4
## 4   <NA>   0.4719835   2909
```

```
df[(df$PoolArea > 0) & is.na(df$PoolQC),c('PoolQC','PoolArea')]
```

```
##      PoolQC PoolArea
## 2421   <NA>      368
## 2504   <NA>      444
## 2600   <NA>      561
```

Imputing the missing values of pools, if no pool then assign ‘None’

```
df[2421,'PoolQC'] = 'Ex'
df[2504,'PoolQC'] = 'Ex'
df[2600,'PoolQC'] = 'Fa'
df$PoolQC[is.na(df$PoolQC)] = 'None'
```

```
garage.cols <- c('GarageArea', 'GarageCars', 'GarageQual', 'GarageFinish', 'GarageCond', 'GarageType')
#df[is.na(df$GarageCond),garage.cols]
```

Imputing the missing values of Garages. If the no garage then assigning 0 or None

```
#length(which(df$GarageYrBlt == df$YearBuilt))
df[(df$GarageArea > 0) & is.na(df$GarageYrBlt), c(garage.cols, 'GarageYrBlt')]
```

```
##      GarageArea GarageCars GarageQual GarageFinish GarageCond GarageType
## 2127        360          1       <NA>         <NA>       <NA>     Detchd
## NA           NA         NA       <NA>         <NA>       <NA>       <NA>
##      GarageYrBlt
## 2127          NA
## NA            NA
```

```
df$GarageYrBlt[2127] <- df$YearBuilt[2127]
df[2127, 'GarageQual'] <- Mode(df$GarageQual)
df[2127, 'GarageFinish'] <- Mode(df$GarageFinish)
df[2127, 'GarageCond'] <- Mode(df$GarageCond)
df$GarageYrBlt[which(is.na(df$GarageYrBlt))] <- 0
```

to numeric - 0, to categorical = 'None'

```
for(i in garage.cols){
if (sapply(df[i], is.numeric) == TRUE){
    df[,i][which(is.na(df[,i]))] <- 0
  }
  else{
    df[,i][which(is.na(df[,i]))] <- "None"
  }
}
```

```
df$KitchenQual[which(is.na(df$KitchenQual))] <- Mode(df$KitchenQual)
```

```
df[is.na(df$MSZoning),c('MSZoning','MSSubClass')]
```

```
##      MSZoning MSSubClass
## 1916     <NA>         30
## 2217     <NA>         20
## 2251     <NA>         70
## 2905     <NA>         20
```

```
table(df$MSZoning, df$MSSubClass)
```

```
##
##              20   30   40   45   50   60   70   75   80   85   90  120  150
##   C (all)     3    8    0    0    7    0    4    0    0    0    0    0    0
##   FV         34    0    0    0    0   43    0    0    0    0    0   19    0
##   RH          4    2    0    1    2    0    3    0    0    0    4    6    0
##   RL       1016   61    4    6  159  529   57    9  115   47   92  117    1
##   RM         20   67    2   11  119    3   63   14    3    1   13   40    0
##
##             160  180  190
##   C (all)     0    0    3
##   FV         43    0    0
##   RH          0    0    4
##   RL         21    0   31
##   RM         64   17   23
```

```
df$MSZoning[c(2217, 2905)] = 'RL'
df$MSZoning[c(1916, 2251)] = 'RM'
```

There are 486 Nas in LotFrontage, setting the NAs to median.

```
df$LotFrontage[which(is.na(df$LotFrontage))] <- median(df$LotFrontage,na.rm = T)
```

There are 2721 NAs in Alley, set them equal to 'None'

```
df$Alley[which(is.na(df$Alley))] <- "None"
```

One of the data is missing the rest set to 0 or 'None'

```
#df[(df$MasVnrArea > 0) & (is.na(df$MasVnrType)),c('MasVnrArea','MasVnrType')]
df[2611, 'MasVnrType'] = 'BrkFace'
df$MasVnrType[is.na(df$MasVnrType)] = 'None'
df$MasVnrArea[is.na(df$MasVnrArea)] = 0
```

For small number of NAs we apply Mode to the categorical, and median to the continous

```r
for(i in colnames(df[,sapply(df, is.character)])){
  if (sum(is.na(df[,i])) < 5){
    df[,i][which(is.na(df[,i]))] <- Mode(df[,i])
  }
}

for(i in colnames(df[,sapply(df, is.integer)])){
  if (sum(is.na(df[,i])) < 5){
    df[,i][which(is.na(df[,i]))] <- median(df[,i], na.rm = T)
  }
}
```

For large number of NAs we apply string "None" to the categorical as a seperate Level, and 0 to the continous

```r
for(i in colnames(df[,sapply(df, is.character)])){
    df[,i][which(is.na(df[,i]))] <- "None"
}
```

We have filled in all the missing values. The remaining ones are the SalesPrice in the predicting Dataset that is fine!

```r
#colSums(sapply(df, is.na))
sum(is.na(df)) == 1459
```

```
## [1] TRUE
```

Creating categorical variables and checking whether and some problem appear. if f.e testing has more levels than the training data!

```r
train_df <- df[df$isTrain==1,]
test_df <- df[df$isTrain==0,]

train_df$isTrain <- NULL
test_df$isTrain <- NULL
test_df$SalePrice <- NULL
```

```r
train_df$MSSubClass <- as.factor(train_df$MSSubClass)
test_df$MSSubClass <- as.factor(test_df$MSSubClass)

train_df$OverallQual <- as.factor(train_df$OverallQual)
test_df$OverallQual <- as.factor(test_df$OverallQual)

train_df$OverallCond <- as.factor(train_df$OverallCond)
test_df$OverallCond <- as.factor(test_df$OverallCond)
```

```r
for(i in colnames(train_df[,sapply(train_df, is.character)])){
    train_df[,i] <- as.factor(train_df[,i])
}
for(i in colnames(test_df[,sapply(test_df, is.character)])){
    test_df[,i] <- as.factor(test_df[,i])
}
```

```r
#Check is some there are more levels in some of the categorical factors in the testing compared to the
for(i in colnames(train_df[,sapply(train_df, is.factor)])){
  if (length(levels(train_df[,i])) < length(levels(test_df[,i]))) {
```

```
    print(i)
    print(levels(train_df[,i]))
    print(levels(test_df[,i]))
  }
}
```

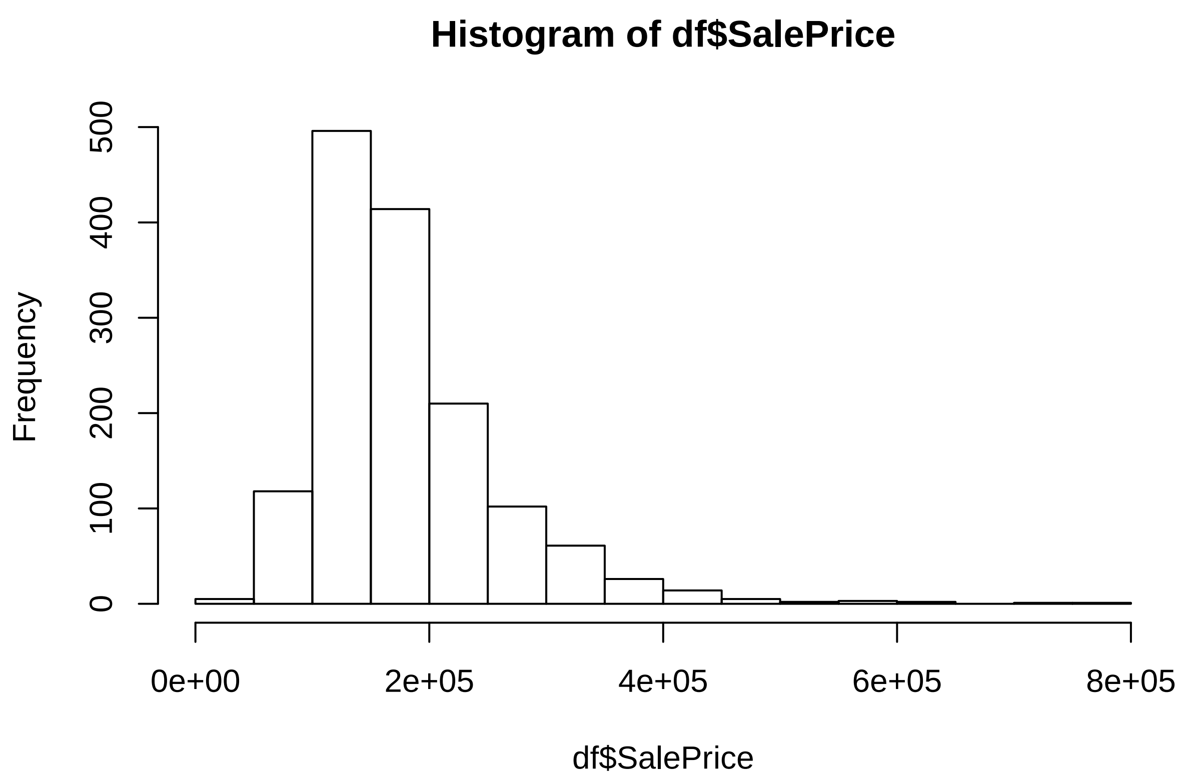
```
## [1] "MSSubClass"
##  [1] "20"  "30"  "40"  "45"  "50"  "60"  "70"  "75"  "80"  "85"  "90"
## [12] "120" "160" "180" "190"
##  [1] "20"  "30"  "40"  "45"  "50"  "60"  "70"  "75"  "80"  "85"  "90"
## [12] "120" "150" "160" "180" "190"
```

level '150' appears once in the testing data and no such level is in the training data. Remove this level.

```
#df[df$MSSubClass == 150,]
df[df$MSSubClass == 150,"MSSubClass"] <- 120
```

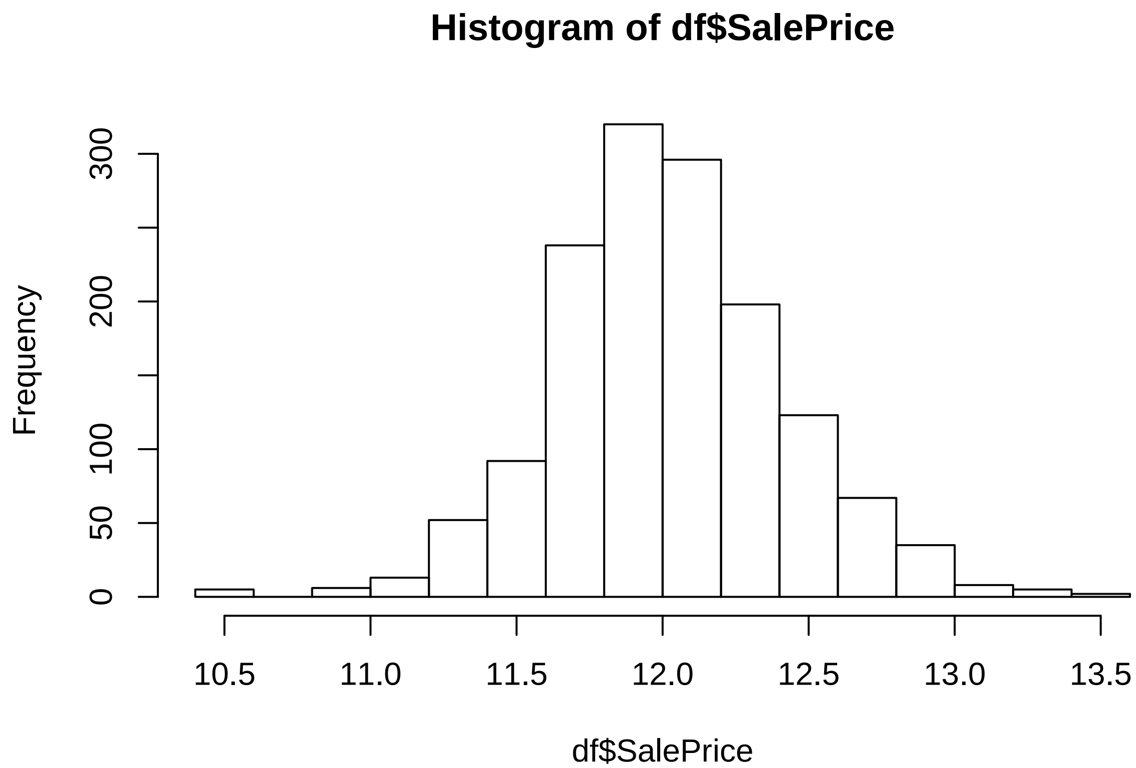
Transformations

```
hist(df$SalePrice)
```




```
df$SalePrice <- log(df$SalePrice)
```

```
hist(df$SalePrice)
```



Create factors in the combined dataframe and split the data into testing and training.

```
for(i in colnames(df[,sapply(df, is.character)])){
    df[,i] <- as.factor(df[,i])
}

df$MSSubClass <- as.factor(df$MSSubClass)
df$OverallQual <- as.factor(df$OverallQual)
df$OverallCond <- as.factor(df$OverallCond)

### THINGS TO CONSIDER:
#df$GarageYrBlt <- as.factor(df$GarageYrBlt) # treat as factor as some of them are '0'
#add years as dummies - POSSIBILITY - but a problem appears, the algorithms cannot treat categorical va
#df$YearBuilt <- as.factor(df$YearBuilt)
#df$YearRemodAdd <- as.factor(df$YearRemodAdd)
#df$YrSold <- as.factor(df$YrSold)
```

```
train_df <- df[df$isTrain==1,]
test_df <- df[df$isTrain==0,]

train_df$isTrain <- NULL
test_df$isTrain <- NULL
test_df$SalePrice <- NULL
```

```r
str(df)
```

```
## 'data.frame':    2919 obs. of  81 variables:
##  $ MSSubClass   : Factor w/ 15 levels "20","30","40",..: 6 1 6 7 6 5 1 6 5 15 ...
##  $ MSZoning     : Factor w/ 5 levels "C (all)","FV",..: 4 4 4 4 4 4 4 4 5 4 ...
##  $ LotFrontage  : int  65 80 68 60 84 85 75 68 51 50 ...
##  $ LotArea      : int  8450 9600 11250 9550 14260 14115 10084 10382 6120 7420 ...
##  $ Street       : Factor w/ 2 levels "Grvl","Pave": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Alley        : Factor w/ 3 levels "Grvl","None",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ LotShape     : Factor w/ 4 levels "IR1","IR2","IR3",..: 4 4 1 1 1 1 4 1 4 4 ...
##  $ LandContour  : Factor w/ 4 levels "Bnk","HLS","Low",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ Utilities    : Factor w/ 2 levels "AllPub","NoSeWa": 1 1 1 1 1 1 1 1 1 1 ...
##  $ LotConfig    : Factor w/ 5 levels "Corner","CulDSac",..: 5 3 5 1 3 5 5 1 5 1 ...
##  $ LandSlope    : Factor w/ 3 levels "Gtl","Mod","Sev": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Neighborhood : Factor w/ 25 levels "Blmngtn","Blueste",..: 6 25 6 7 14 12 21 17 18 4 ...
##  $ Condition1   : Factor w/ 9 levels "Artery","Feedr",..: 3 2 3 3 3 3 3 5 1 1 ...
##  $ Condition2   : Factor w/ 8 levels "Artery","Feedr",..: 3 3 3 3 3 3 3 3 3 1 ...
##  $ BldgType     : Factor w/ 5 levels "1Fam","2fmCon",..: 1 1 1 1 1 1 1 1 1 2 ...
##  $ HouseStyle   : Factor w/ 8 levels "1.5Fin","1.5Unf",..: 6 3 6 6 6 1 3 6 1 2 ...
##  $ OverallQual  : Factor w/ 10 levels "1","2","3","4",..: 7 6 7 7 8 5 8 7 7 5 ...
##  $ OverallCond  : Factor w/ 9 levels "1","2","3","4",..: 5 8 5 5 5 5 5 6 5 6 ...
##  $ YearBuilt    : int  2003 1976 2001 1915 2000 1993 2004 1973 1931 1939 ...
##  $ YearRemodAdd : int  2003 1976 2002 1970 2000 1995 2005 1973 1950 1950 ...
##  $ RoofStyle    : Factor w/ 6 levels "Flat","Gable",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ RoofMatl     : Factor w/ 8 levels "ClyTile","CompShg",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ Exterior1st  : Factor w/ 15 levels "AsbShng","AsphShn",..: 13 9 13 14 13 13 13 7 4 9 ...
##  $ Exterior2nd  : Factor w/ 16 levels "AsbShng","AsphShn",..: 14 9 14 16 14 14 14 7 16 9 ...
##  $ MasVnrType   : Factor w/ 4 levels "BrkCmn","BrkFace",..: 2 3 2 3 2 3 4 4 3 3 ...
##  $ MasVnrArea   : num  196 0 162 0 350 0 186 240 0 0 ...
##  $ ExterQual    : Factor w/ 4 levels "Ex","Fa","Gd",..: 3 4 3 4 3 4 3 4 4 4 ...
##  $ ExterCond    : Factor w/ 5 levels "Ex","Fa","Gd",..: 5 5 5 5 5 5 5 5 5 5 ...
##  $ Foundation   : Factor w/ 6 levels "BrkTil","CBlock",..: 3 2 3 1 3 6 3 2 1 1 ...
##  $ BsmtQual     : Factor w/ 5 levels "Ex","Fa","Gd",..: 3 3 3 5 3 3 1 3 5 5 ...
##  $ BsmtCond     : Factor w/ 5 levels "Fa","Gd","None",..: 5 5 5 2 5 5 5 5 5 5 ...
##  $ BsmtExposure : Factor w/ 5 levels "Av","Gd","Mn",..: 4 2 3 4 1 4 1 3 4 4 ...
##  $ BsmtFinType1 : Factor w/ 7 levels "ALQ","BLQ","GLQ",..: 3 1 3 1 3 3 3 1 7 3 ...
##  $ BsmtFinSF1   : num  706 978 486 216 655 ...
##  $ BsmtFinType2 : Factor w/ 7 levels "ALQ","BLQ","GLQ",..: 7 7 7 7 7 7 7 2 7 7 ...
##  $ BsmtFinSF2   : num  0 0 0 0 0 0 0 32 0 0 ...
##  $ BsmtUnfSF    : num  150 284 434 540 490 64 317 216 952 140 ...
##  $ TotalBsmtSF  : num  856 1262 920 756 1145 ...
##  $ Heating      : Factor w/ 6 levels "Floor","GasA",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ HeatingQC    : Factor w/ 5 levels "Ex","Fa","Gd",..: 1 1 1 3 1 1 1 1 3 1 ...
##  $ CentralAir   : Factor w/ 2 levels "N","Y": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Electrical   : Factor w/ 5 levels "FuseA","FuseF",..: 5 5 5 5 5 5 5 5 2 5 ...
##  $ X1stFlrSF    : int  856 1262 920 961 1145 796 1694 1107 1022 1077 ...
##  $ X2ndFlrSF    : int  854 0 866 756 1053 566 0 983 752 0 ...
##  $ LowQualFinSF : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ GrLivArea    : int  1710 1262 1786 1717 2198 1362 1694 2090 1774 1077 ...
##  $ BsmtFullBath : int  1 0 1 1 1 1 1 1 0 1 ...
##  $ BsmtHalfBath : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ FullBath     : int  2 2 2 1 2 1 2 2 2 1 ...
##  $ HalfBath     : int  1 0 1 0 1 1 0 1 0 0 ...
##  $ BedroomAbvGr : int  3 3 3 3 4 1 3 3 2 2 ...
```

```
## $ KitchenAbvGr : int  1 1 1 1 1 1 1 1 2 2 ...
## $ KitchenQual  : Factor w/ 4 levels "Ex","Fa","Gd",..: 3 4 3 3 3 4 3 4 4 4 ...
## $ TotRmsAbvGrd : int  8 6 6 7 9 5 7 7 8 5 ...
## $ Functional   : Factor w/ 7 levels "Maj1","Maj2",..: 7 7 7 7 7 7 7 7 3 7 ...
## $ Fireplaces   : int  0 1 1 1 1 0 1 2 2 2 ...
## $ FireplaceQu  : Factor w/ 6 levels "Ex","Fa","Gd",..: 4 6 6 3 6 4 3 6 6 6 ...
## $ GarageType   : Factor w/ 7 levels "2Types","Attchd",..: 2 2 2 6 2 2 2 2 6 2 ...
## $ GarageYrBlt  : num  2003 1976 2001 1998 2000 ...
## $ GarageFinish : Factor w/ 4 levels "Fin","None","RFn",..: 3 3 3 4 3 4 3 3 4 3 ...
## $ GarageCars   : num  2 2 2 3 3 2 2 2 2 1 ...
## $ GarageArea   : num  548 460 608 642 836 480 636 484 468 205 ...
## $ GarageQual   : Factor w/ 6 levels "Ex","Fa","Gd",..: 6 6 6 6 6 6 6 6 2 3 ...
## $ GarageCond   : Factor w/ 6 levels "Ex","Fa","Gd",..: 6 6 6 6 6 6 6 6 6 6 ...
## $ PavedDrive   : Factor w/ 3 levels "N","P","Y": 3 3 3 3 3 3 3 3 3 3 ...
## $ WoodDeckSF   : int  0 298 0 0 192 40 255 235 90 0 ...
## $ OpenPorchSF  : int  61 0 42 35 84 30 57 204 0 4 ...
## $ EnclosedPorch: int  0 0 0 272 0 0 0 228 205 0 ...
## $ X3SsnPorch   : int  0 0 0 0 0 320 0 0 0 0 ...
## $ ScreenPorch  : int  0 0 0 0 0 0 0 0 0 0 ...
## $ PoolArea     : int  0 0 0 0 0 0 0 0 0 0 ...
## $ PoolQC       : Factor w/ 4 levels "Ex","Fa","Gd",..: 4 4 4 4 4 4 4 4 4 4 ...
## $ Fence        : Factor w/ 5 levels "GdPrv","GdWo",..: 5 5 5 5 5 3 5 5 5 5 ...
## $ MiscFeature  : Factor w/ 5 levels "Gar2","None",..: 2 2 2 2 2 4 2 4 2 2 ...
## $ MiscVal      : int  0 0 0 0 0 700 0 350 0 0 ...
## $ MoSold       : int  2 5 9 2 12 10 8 11 4 1 ...
## $ YrSold       : int  2008 2007 2008 2006 2008 2009 2007 2009 2008 2008 ...
## $ SaleType     : Factor w/ 9 levels "COD","Con","ConLD",..: 9 9 9 9 9 9 9 9 9 9 ...
## $ SaleCondition: Factor w/ 6 levels "Abnorml","AdjLand",..: 5 5 5 1 5 5 5 5 1 5 ...
## $ SalePrice    : num  12.2 12.1 12.3 11.8 12.4 ...
## $ isTrain      : num  1 1 1 1 1 1 1 1 1 1 ...
```

Regression Code

```
library(tree) # Normal Tree
library(glmnet) # Lasso/Ridge
library(randomForest) # random Forest
library(xgboost) # boosting trees
library(caret) # for tuning xgboost
library(gbm) # for the gradient boosting model
```

Generating matrix from the data frames, no intercept

```
X_train<- model.matrix(SalePrice~.-1, data = train_df)
y_train <- train_df$SalePrice
X_test <- model.matrix(~.-1, data=test_df)
```

Linear Regression

Regression with all the parameters - just as a benchmark

```
lm_fit_all = lm(SalePrice ~., data = train_df)
#summary(lm_fit_all)
```

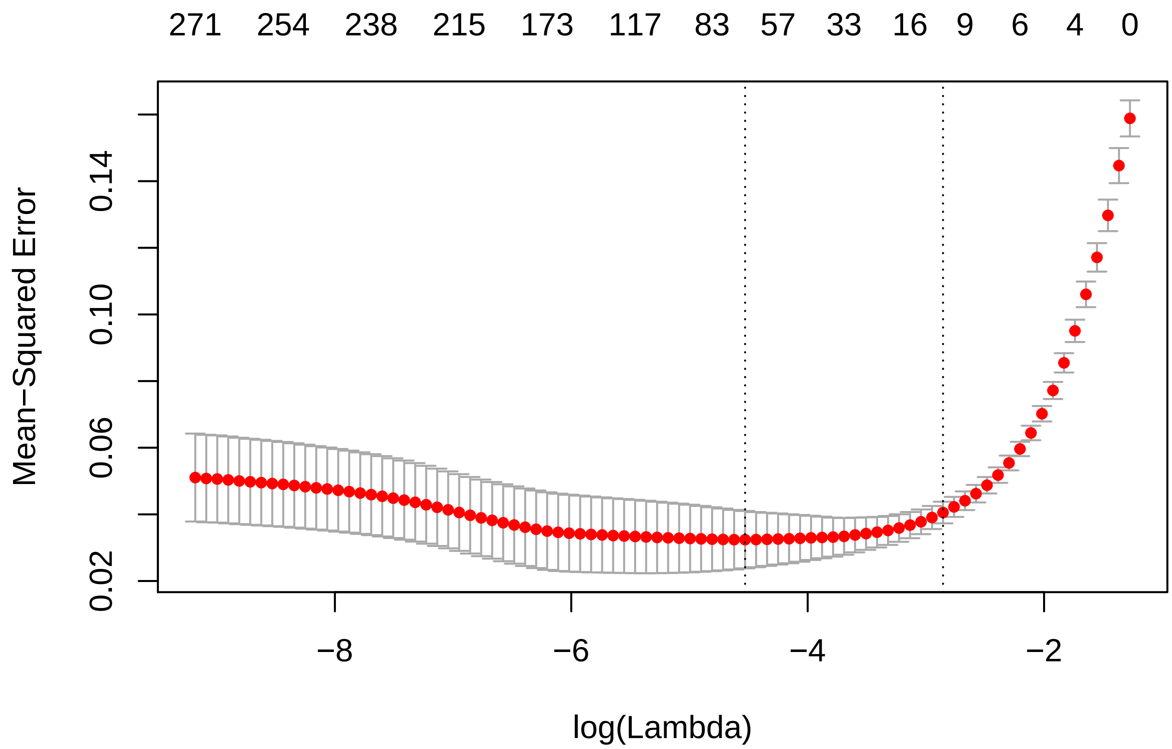
```
prediction_LR_ALL_log <- predict(lm_fit_all, test_df, type="response")
prediction_LR_ALL <- exp(prediction_LR_ALL_log)
```

```
prediction_LR_ALL <- cbind(Id = rownames(test_df), SalesPrice = prediction_LR_ALL)
```

```
write.table(prediction_LR_ALL, file="prediction_LR_ALL.csv",col.names = c("Id","SalePrice"), sep =',',
```

Lasso

```
cv.lasso <-cv.glmnet(X_train, y_train, nfolds = 10, alpha = 1)
plot(cv.lasso)
```




```r
penalty_min <- cv.lasso$lambda.min #optimal lambda
penalty_1se <- cv.lasso$lambda.1se # 1 Standard Error Apart
fit.lasso_min <-glmnet(X_train, y_train, alpha = 1, lambda = penalty_min) #estimate the model with min
fit.lasso_1se <-glmnet(X_train, y_train, alpha = 1, lambda = penalty_1se) #estimate the model with 1se
```

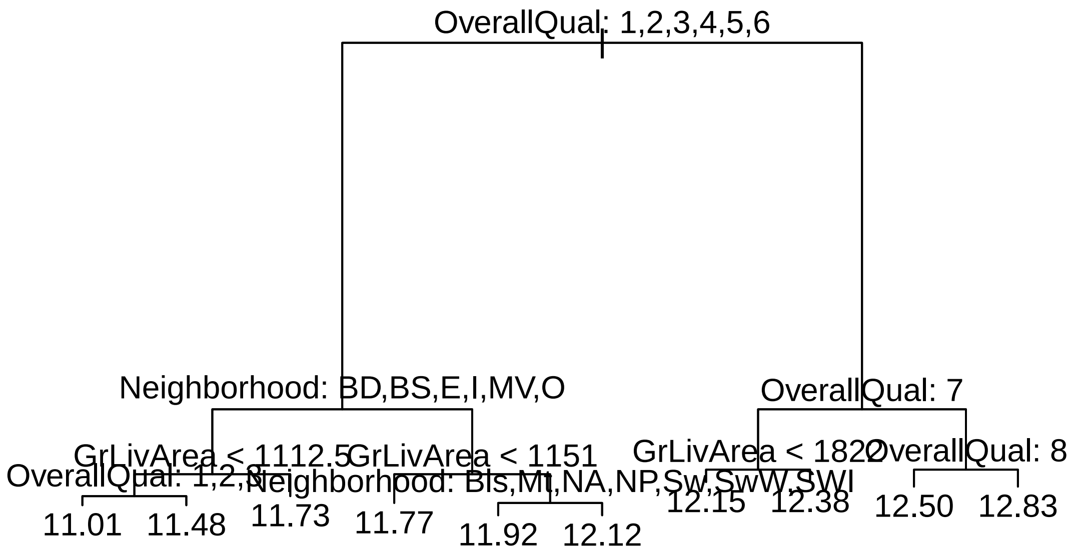
```r
prediction_LASSO_min_log <- predict(fit.lasso_min, X_test)
prediction_LASSO_1se_log <- predict(fit.lasso_1se, X_test)

prediction_LASSO_min <- exp(prediction_LASSO_min_log)
prediction_LASSO_1se <- exp(prediction_LASSO_1se_log)
```

```r
prediction_LASSO_min <-cbind(Id = rownames(test_df), SalesPrice = prediction_LASSO_min)
prediction_LASSO_1se <- cbind(Id = rownames(test_df), SalesPrice = prediction_LASSO_1se)
```

```r
write.table(prediction_LASSO_min, file="prediction_LASSO_min.csv",col.names = c("Id", "SalePrice"), sep
write.table(prediction_LASSO_1se, file="prediction_LASSO_1se.csv",col.names = c("Id", "SalePrice"), sep
```

Regression Tree

```r
tree.SalePrice <-tree(SalePrice~., data = train_df)
#summary(tree.SalePrice)
plot(tree.SalePrice)
text(tree.SalePrice, pretty=1)
```




```r
cv.SalePrice <-cv.tree(tree.SalePrice, K = 10)
plot(cv.SalePrice$size, cv.SalePrice$dev, type="b")
## In this case, the most complex tree is selected by cross-validation
prune.SalePrice <-prune.tree(tree.SalePrice, best=10)
plot(prune.SalePrice)
text(prune.SalePrice, pretty=1)
cv.SalePrice$dev # no PRUNING DONE
```

```r
prediction_TREE_log <- predict(prune.SalePrice ,test_df)
prediction_TREE <- exp(prediction_TREE_log)
```

```r
prediction_TREE <- cbind(Id = rownames(X_test), SalesPrice = prediction_TREE)
```

```r
write.table(prediction_TREE, file="prediction_TREE.csv",col.names = c("Id", "SalePrice"), sep =',', row
```

Bagging

```r
set.seed(1)
bag.SalePrice <-randomForest(SalePrice~., data=train_df, mtry= 79, importance=TRUE)
bag.SalePrice
```

```r
prediction_bag_log <- predict(bag.SalePrice, newdata = test_df)
prediction_bag <- exp(prediction_bag_log)
```

```r
prediction_bag <- cbind(Id = rownames(X_test), SalesPrice = prediction_bag)
```

```r
write.table(prediction_bag, file="prediction_bag.csv",col.names = c("Id", "SalePrice"), sep =',', row.na
```

Random Forest $m = \frac{p}{3}$

```r
RF_p3.SalePrice <-randomForest(SalePrice~., data=train_df, mtry = 26, importance=TRUE)
```

```r
prediction_RF_p3_log <- predict(RF_p3.SalePrice, newdata = test_df)
prediction_RF_p3 <- exp(prediction_RF_p3_log)
```

```r
prediction_RF_p3 <- cbind(Id = rownames(X_test), SalesPrice = prediction_RF_p3)
```

```r
write.table(prediction_RF_p3, file="prediction_RF_p3.csv",col.names = c("Id", "SalePrice"), sep =',', r
```

Random Forest $m = \sqrt(p)$

```r
## We could change the number of trees grown by randomForest() using ntree argument
RF.SalePrice <-randomForest(SalePrice~., data=train_df, importance=TRUE)
```

```r
prediction_RF_log <- predict(RF.SalePrice, newdata = test_df)
prediction_RF <- exp(prediction_RF_log)
```

```r
prediction_RF <- cbind(Id = rownames(X_test), SalesPrice = prediction_RF)
```

```r
write.table(prediction_RF, file="prediction_RF.csv",col.names = c("Id", "SalePrice"), sep =',', row.name
```

Tuning Random Forest - selecting 'm'

```r
x_train_df <- train_df
x_train_df$SalePrice <- NULL
```

```r
results <- rfcv(x_train_df, y_train, cv.fold=10, scale="log", step=0.5)
```

```r
results$error.cv
```

```r
## We could change the number of trees grown by randomForest() using ntree argument
RF.SalePrice_tuned <-randomForest(SalePrice~., data=train_df, importance=TRUE, ntree = 1000, mtry=38)
```

```r
prediction_RF_tuned_log <- predict(RF.SalePrice_tuned, newdata = test_df)
prediction_RF_tuned <- exp(prediction_RF_tuned_log)
```

```r
prediction_RF_tuned <- cbind(Id = rownames(X_test), SalesPrice = prediction_RF_tuned)
```

```r
write.table(prediction_RF_tuned, file="prediction_RF_tuned.csv",col.names = c("Id", "SalePrice"), sep =
```

Gradient Boosting Model - library (gbm)

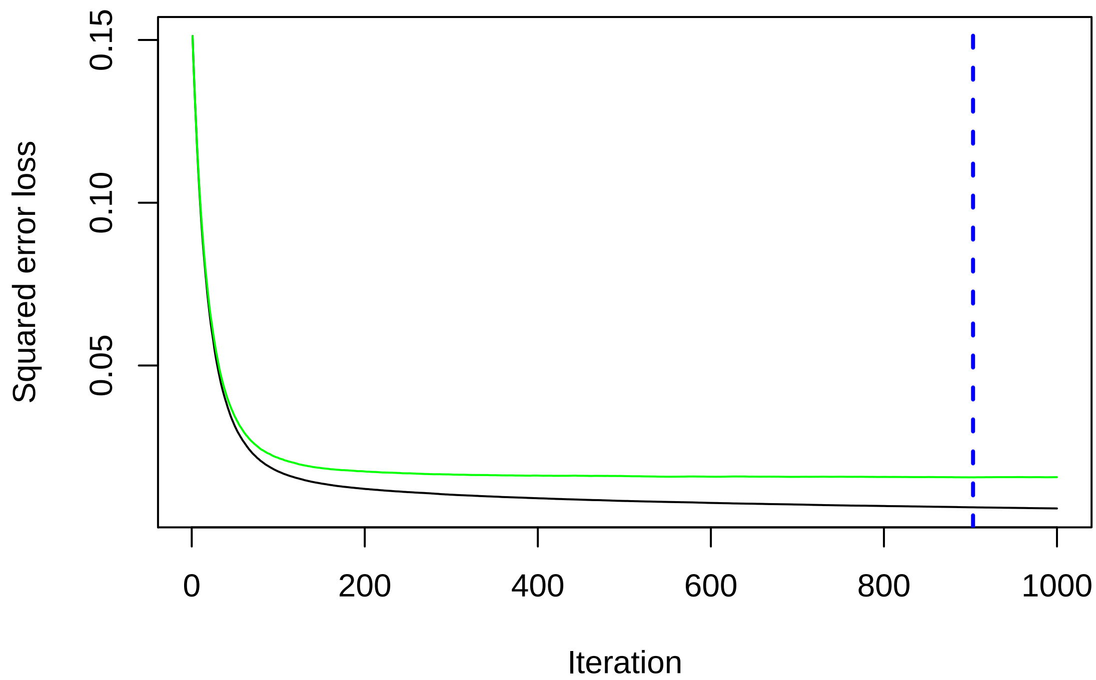
```r
set.seed(1)
boost.train = gbm(SalePrice ~. , data=train_df,
                  distribution = "gaussian",
                  n.trees = 1000,
                  shrinkage = 0.05,
```

```r
                   interaction.depth = 2,
                   bag.fraction = 0.66,
                   cv.folds = 10,
                   verbose = FALSE,
                   n.cores = 8)

min(boost.train$cv.error)
```

```
## [1] 0.01562838
```

```r
#attributes(boost.train)
bestTreeForPrediction = gbm.perf(boost.train)
```

```
## Using cv method...
```




```r
prediction_BOOST_log <- predict(boost.train, test_df)
prediction_BOOST <- exp(prediction_BOOST_log)
```

```r
prediction_BOOST <- cbind(Id = rownames(X_test), SalesPrice = prediction_BOOST)
```

```r
write.table(prediction_BOOST, file="prediction_BOOST.csv",col.names = c("Id", "SalePrice"), sep =',', r
```

XGBoost

```r
#setup
library(parallel) #checking the number of cores
```

```r
detectCores() #=> 8
dtrain <- xgb.DMatrix(X_train, label = y_train)
```

Cross Validation in XGBoost - CV
Source: StackOverflow

```r
best_param2 = list()
best_rmse = Inf
best_rmse_index = 0
best_seednumber = 1234

for (iter in 1:20) {
    param <- list(objective = "reg:linear",
          max_depth = sample(2:6, 1),
          eta = runif(1, .01, .05),
          gamma = runif(1, 0.0, 0.013),
          subsample = runif(1, .7, .8),
          colsample_bytree = runif(1, .6, .7),
          min_child_weight = sample(1:3, 1)
          )
    cv.nround = 1000
    cv.nfold = 10
    seed.number = sample.int(10000, 1)[[1]]
    set.seed(seed.number)
    mdcv <- xgb.cv(data=dtrain, params = param, nthread=8,
                    nfold=cv.nfold, nrounds=cv.nround,
                    verbose = F, early.stop.rounds=8, maximize=FALSE)

    min_rmse = min(mdcv$evaluation_log[,test_rmse_mean])
    min_rmse_index = which.min(mdcv$evaluation_log[,test_rmse_mean])


    if (min_rmse < best_rmse) {
        best_rmse = min_rmse
        best_rmse_index = min_rmse_index
        best_seednumber = seed.number
        best_param2 = param
    }
    print(iter)
}
```

```r
print(best_param) # rmse = 0.12567
```

```r
nround = best_rmse_index
set.seed(best_seednumber)
model_XGB_tune2 <- xgb.train(data=dtrain, params=best_param2, nrounds=nround, nthread=8)
```

```r
prediction_XGB_tune2_log <- predict(model_XGB_tune2, X_test)
prediction_XGB_tune2 <- exp(prediction_XGB_tune2_log)
```

```r
prediction_XGB_tune2 <- cbind(Id = rownames(X_test), SalesPrice = prediction_XGB_tune2)
```

```r
write.table(prediction_XGB_tune2, file="prediction_XBG_tune2.csv",col.names = c("Id", "SalePrice"), sep
```

Classification Code

```r
mean(train_df$SalePrice)
```

Coverting SalePrice into a binary variable

```r
train_df$Classifier <- ifelse(train_df$SalePrice <= 12.024,"Low","High")
train_df$Classifier <- as.factor(train_df$Classifier)
train_df$Classifier <- factor(train_df$Classifier, levels = c("Low", "High"))
```

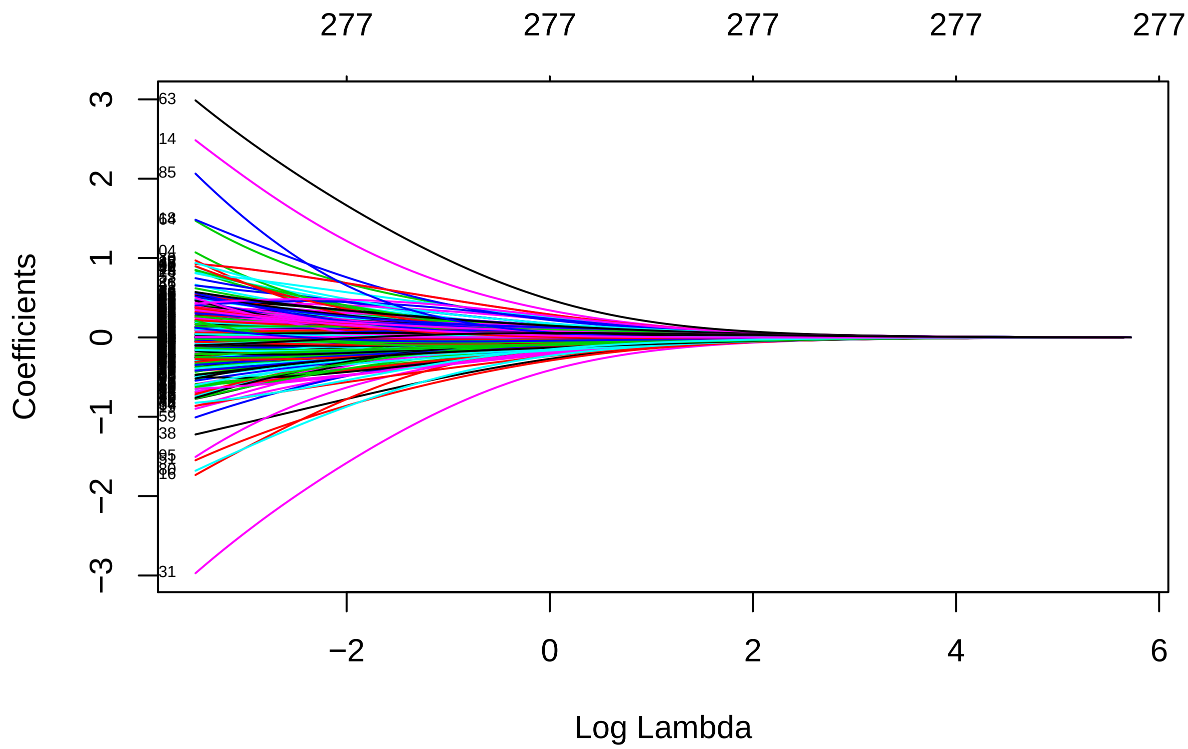
Splitting of data into training and test set

```r
attach(train_df)
set.seed(1)
training<- sample(1:nrow(train_df), 1460*0.5)
test.train_df <- train_df[-training,]
train.train_df <- train_df[training,]
Classifier.test <- test.train_df$Classifier
```

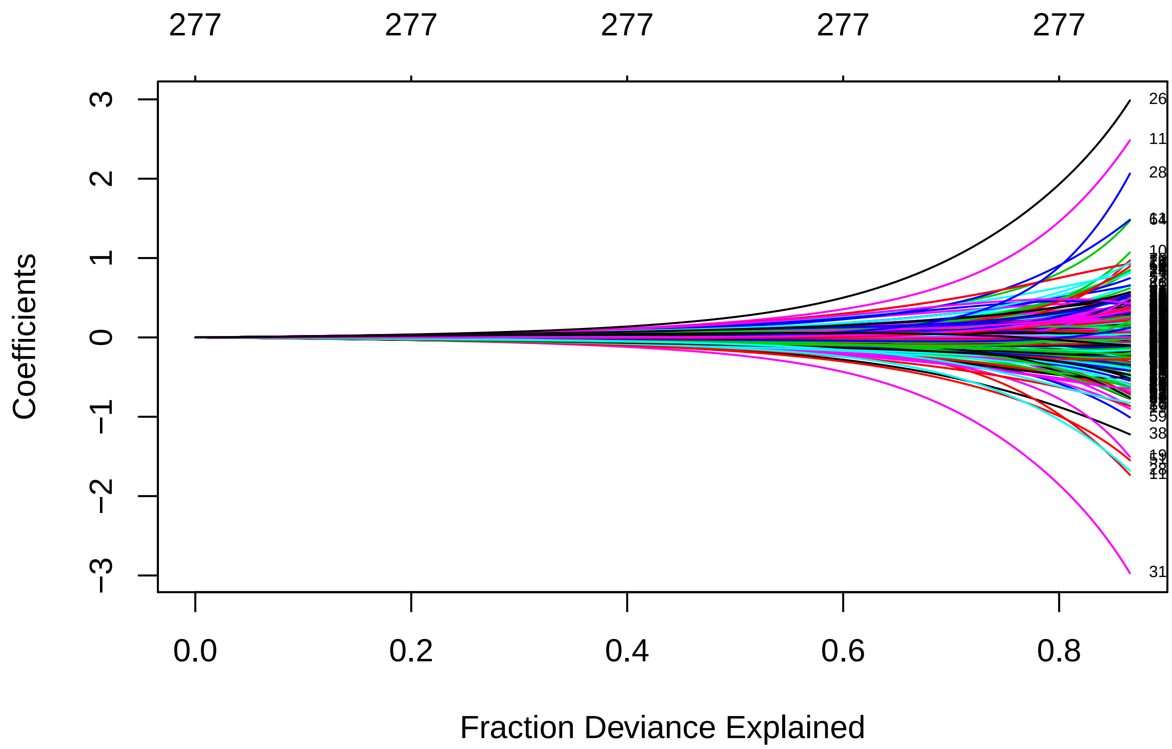
Ridge

```r
x.train.classifier <- model.matrix(Classifier~.-SalePrice, data=train.train_df)
y.train.classifier <- train.train_df$Classifier
x.test.classifier <- model.matrix(Classifier~.-SalePrice, data=test.train_df)
```

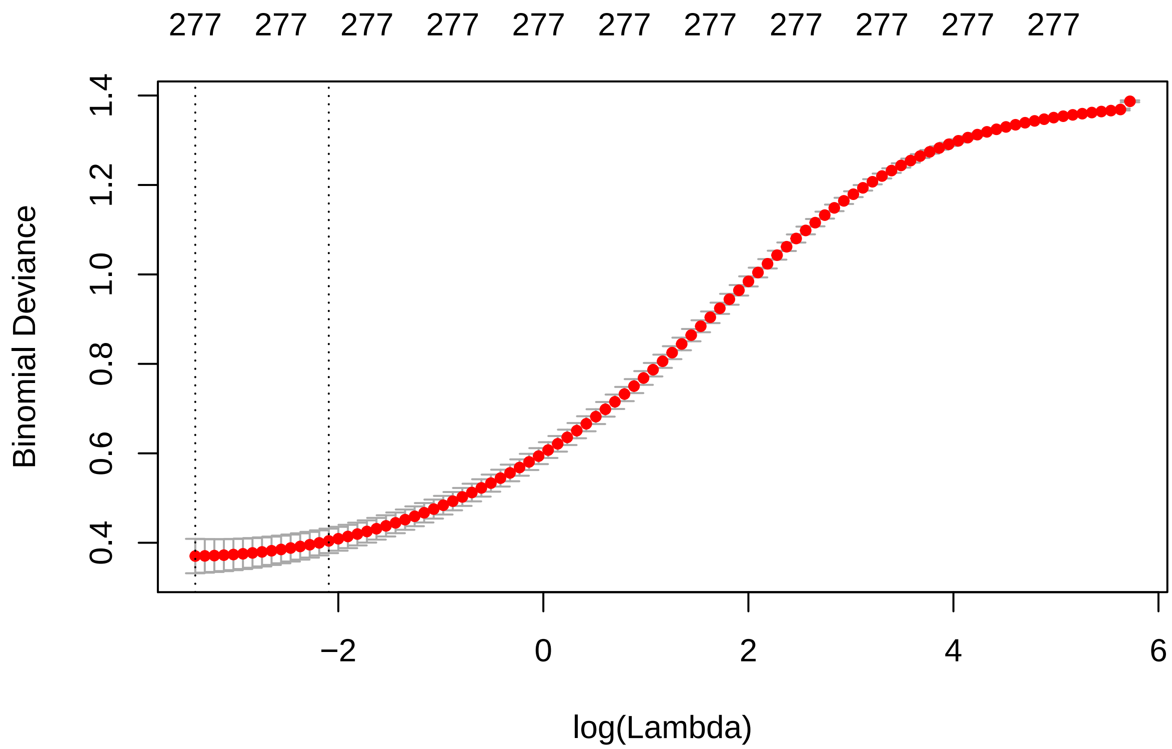
```r
fit.ridge <- glmnet(x.train.classifier,y.train.classifier,alpha=0, family="binomial")
plot(fit.ridge, xvar='lambda', label=TRUE)
```




```
plot(fit.ridge, xvar='dev', label=TRUE)
```




```
cv.ridge <- cv.glmnet(x.train.classifier,y.train.classifier,alpha=0,family="binomial")
plot(cv.ridge)
```



Ridge-Minimum Lambda

```
ridge.min.lambda=cv.ridge$lambda.min
fit.ridge.min <- glmnet(x.train.classifier,y.train.classifier,alpha=0, family="binomial", lambda=ridge.r
prediction.ridge.min.log <- predict(fit.ridge.min, x.test.classifier)
prediction.ridge.min.log.classifier <- (ifelse(prediction.ridge.min.log >0.5,1,0))
table(prediction.ridge.min.log.classifier, Classifier.test)
```

```
##                                    Classifier.test
## prediction.ridge.min.log.classifier Low High
##                                   0 362   40
##                                   1  27  301
```

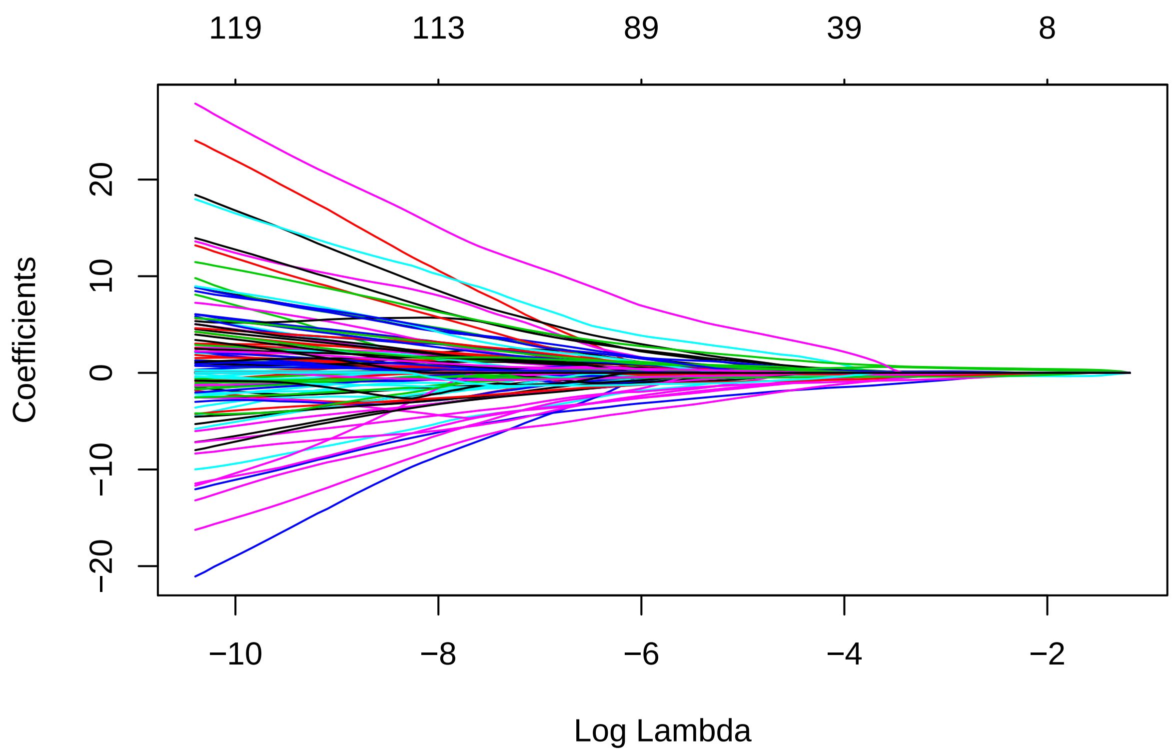
Ridge-1se

```
ridge.1se.lambda=cv.ridge$lambda.1se
fit.ridge.1se <- glmnet(x.train.classifier,y.train.classifier,alpha=0, family="binomial", lambda=ridge.1
prediction.ridge.1se.log <- predict(fit.ridge.1se, x.test.classifier)
prediction.ridge.1se.log.classifier <- ifelse(prediction.ridge.1se.log >0.5,1,0)
table(prediction.ridge.1se.log.classifier, Classifier.test)
```

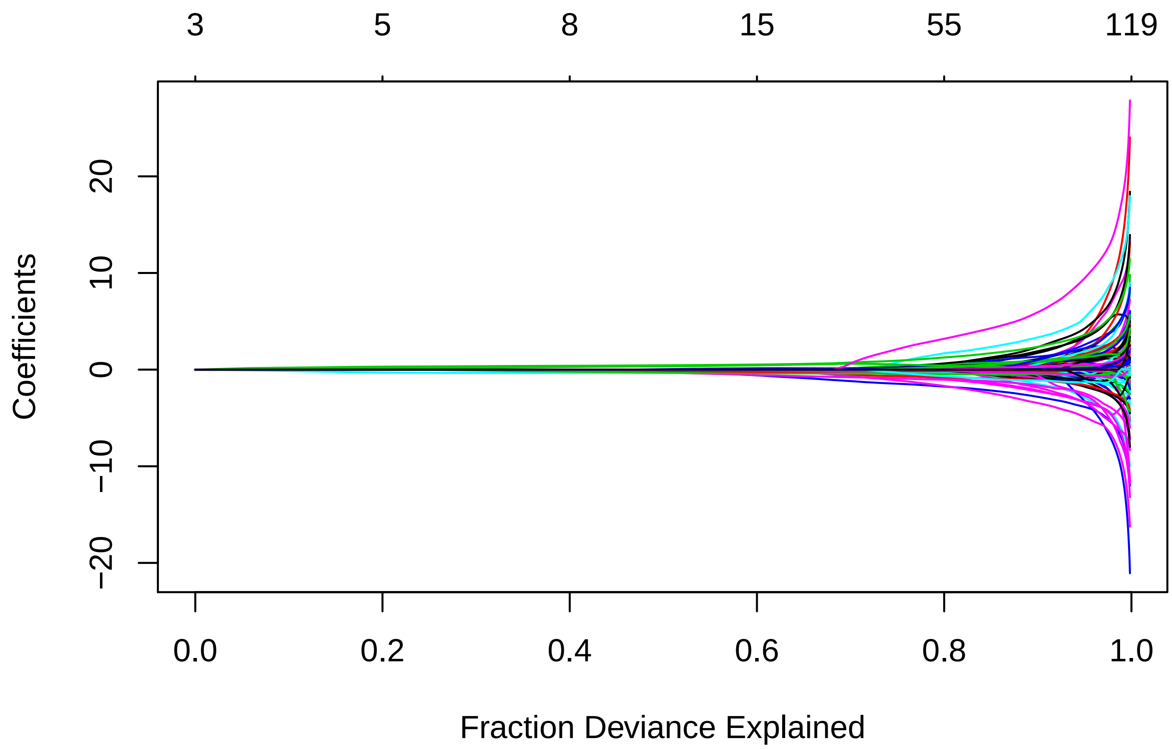
```
##                                    Classifier.test
## prediction.ridge.1se.log.classifier Low High
##                                   0 364   40
##                                   1  25  301
```

Lasso

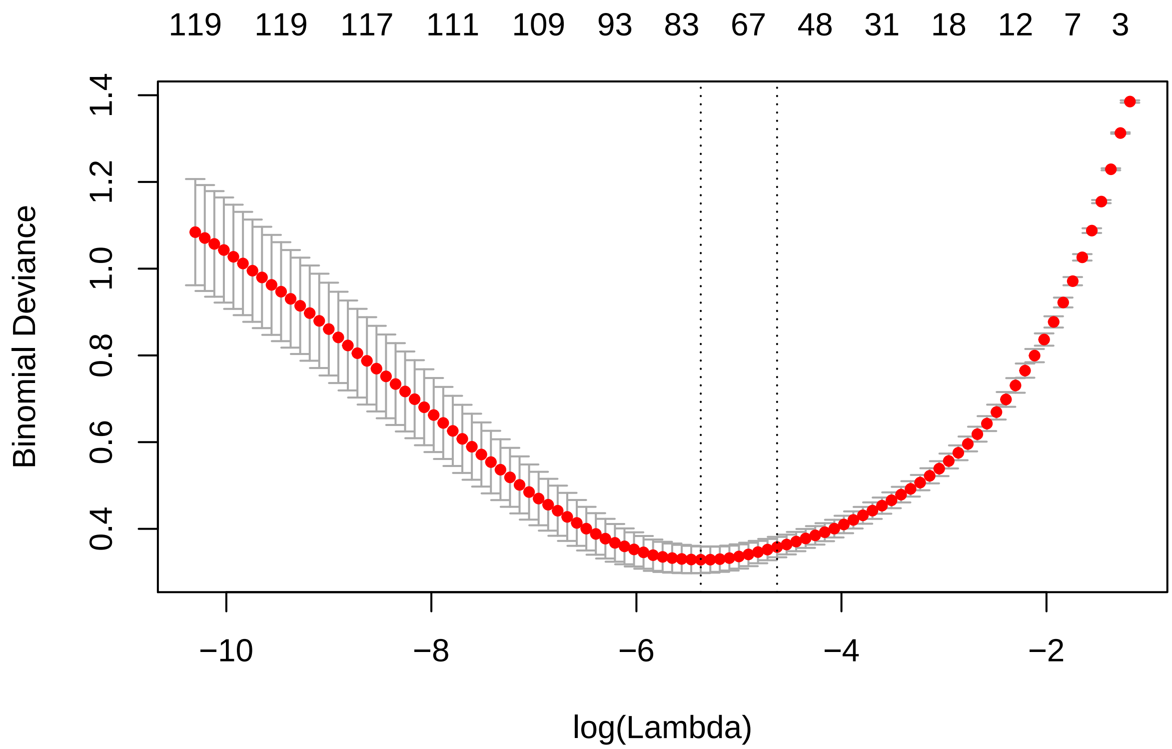
```
fit.lasso <- glmnet(x.train.classifier,y.train.classifier, family="binomial")
plot(fit.lasso, xvar='lambda', lanel=TRUE)
```




```
plot(fit.lasso, xvar='dev', lanel=TRUE)
```




```
cv.lasso <- cv.glmnet(x.train.classifier,y.train.classifier ,family="binomial")
plot(cv.lasso)
```



Lasso-Minimum Lambda

```
lasso.min.lambda=cv.lasso$lambda.min
fit.lasso.min <- glmnet(x.train.classifier,y.train.classifier, family="binomial", lambda=lasso.min.lambd
prediction.lasso.min.log <- predict(fit.lasso.min, x.test.classifier)
prediction.lasso.min.log.classifier <- (ifelse(prediction.lasso.min.log >0.5,1,0))
table(prediction.lasso.min.log.classifier, Classifier.test)
```

```
##                                      Classifier.test
## prediction.lasso.min.log.classifier Low High
##                                    0 354   33
##                                    1  35  308
```

Lasso-1se

```
lasso.1se.lambda=cv.lasso$lambda.1se
fit.lasso.1se <- glmnet(x.train.classifier,y.train.classifier,alpha=0, family="binomial", lambda=lasso.1
prediction.lasso.1se.log <- predict(fit.lasso.1se, x.test.classifier)
prediction.lasso.1se.log.classifier <- ifelse(prediction.lasso.1se.log >0.5,1,0)
table(prediction.lasso.1se.log.classifier, Classifier.test)
```

```
##                                      Classifier.test
## prediction.lasso.1se.log.classifier Low High
##                                    0 357   41
##                                    1  32  300
```

Classification Tree

```
tree.train_df <- tree(Classifier~.-SalePrice, train.train_df)
```

```
tree.pred <- predict(tree.train_df, test.train_df, type="class")
length(tree.pred)
```

```
## [1] 730
```

```
length(Classifier.test)
```

```
## [1] 730
```

```
table(tree.pred, Classifier.test)
```

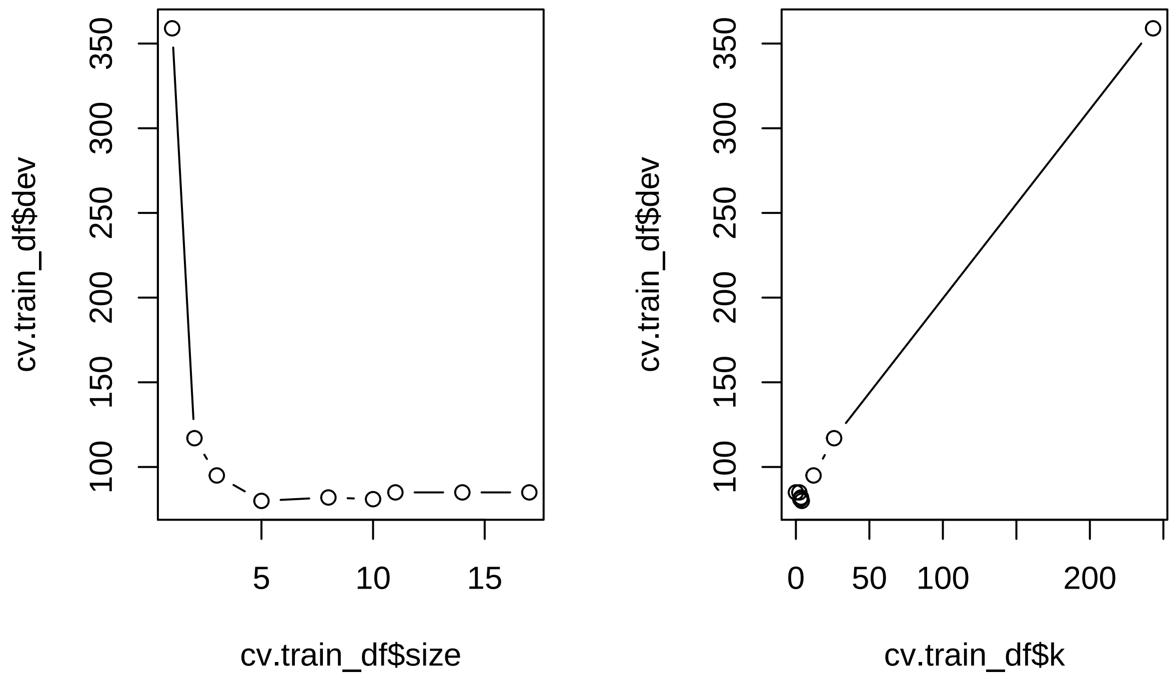
```
##          Classifier.test
## tree.pred Low High
##      Low  335   35
##      High  54  306
```

```
mean(tree.pred!=Classifier.test)
```

```
## [1] 0.1219178
```

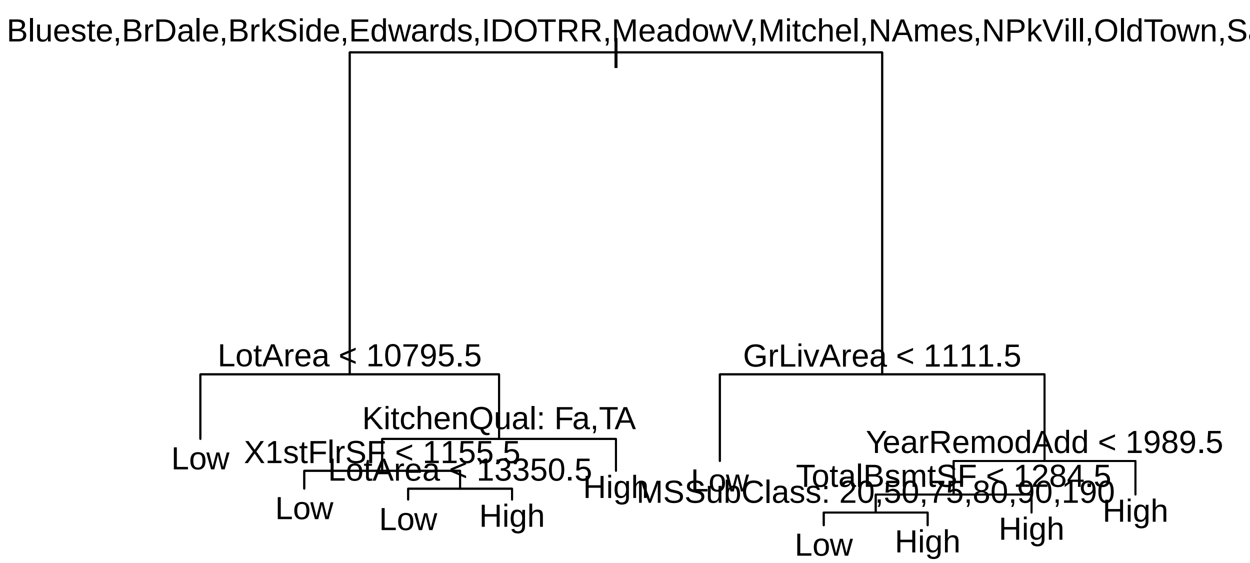
Prune the tree

```
cv.train_df <- cv.tree(tree.train_df, FUN= prune.misclass)
par(mfrow=c(1,2))
plot(cv.train_df$size, cv.train_df$dev, type="b")
plot(cv.train_df$k, cv.train_df$dev, type="b")
```



The optimal number of terminal node is 9

```
par(mfrow=c(1,1))
prune.train_df <-prune.misclass(tree.train_df, best=9)
plot(prune.train_df)
text(prune.train_df, pretty=0)
```



Compute the test error rate using the pruned tree

```
tree.pred <-predict(prune.train_df, test.train_df, type="class")
table(tree.pred,Classifier.test)
```

```
##          Classifier.test
## tree.pred Low High
##      Low  339   35
##      High  50  306
```

```
mean(tree.pred!=Classifier.test)
```

```
## [1] 0.1164384
```

Random Forest

Bagging: m=p

```
bag.train_df <- randomForest(Classifier~. -SalePrice, data=train.train_df, mtry=79, importance=TRUE)
bag.train_df
```

```
##
## Call:
##  randomForest(formula = Classifier ~ . - SalePrice, data = train.train_df,      mtry = 79, importanc
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 79
##
```

```
##         OOB estimate of  error rate: 7.81%
## Confusion matrix:
##      Low High class.error
## Low  343   28  0.07547170
## High  29  330  0.08077994
```

Predict

```r
bag.classifier <- predict(bag.train_df, newdata = test.train_df)
table(predict=bag.classifier, truth=Classifier.test)
```

```
##        truth
## predict Low High
##    Low  351   34
##    High  38  307
```

```r
mean(bag.classifier!=Classifier.test)
```

```
## [1] 0.09863014
```

Random Forest: m=p/3

```r
rf.train_df <- randomForest(Classifier~. -SalePrice, data=train.train_df, mtry=26, importance=TRUE)
rf.train_df
```

```
##
## Call:
##  randomForest(formula = Classifier ~ . - SalePrice, data = train.train_df,      mtry = 26, importanc
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 26
##
##         OOB estimate of  error rate: 7.53%
## Confusion matrix:
##      Low High class.error
## Low  346   25  0.06738544
## High  30  329  0.08356546
```

Predict

```r
rf.classifier <- predict(rf.train_df, newdata = test.train_df)
table(predict=rf.classifier, truth=Classifier.test)
mean(rf.classifier!=Classifier.test)
```

Support Vector Machine

Basic SVM Model

```r
svmfit <-svm(Classifier~.-SalePrice, data=train.train_df, kernel="linear", cost=10, scale=FALSE)
```

Tuning SVM Model

```r
tune.out <-tune(svm, Classifier~. -SalePrice , data=train.train_df, kernel="linear", ranges=list(cost=c
summary(tune.out)
```

We see that cost=0.01 results in the lowest cross validation error rate

tune() function stores the best model obtained, which can be assessed as follows

```r
bestmod <-tune.out$best.model
summary(bestmod)

svm_pred <-predict(bestmod, test.train_df)
table(predict=svm_pred, truth=Classifier.test)
mean(svm_pred!=Classifier.test)
```

Boosting

```r
train.train_df$Classifier2 <- ifelse(train.train_df$Classifier=="High",1,0)
boost.train=gbm(Classifier2~. -SalePrice -Classifier, data=train.train_df, distribution="bernoulli", n.t
```

Predicting

```r
boost.pred=predict(boost.train, newdata=test.train_df, n.trees=5000)
boost.classifier=ifelse(boost.pred >0.5,1,0)
Classifier2.test <- ifelse(Classifier.test=="High",1,0)
table(predict=boost.classifier, truth=Classifier2.test)
mean(boost.classifier!=Classifier2.test)
```