Kaggle Competition

House Prices: Advanced Regression Techniques

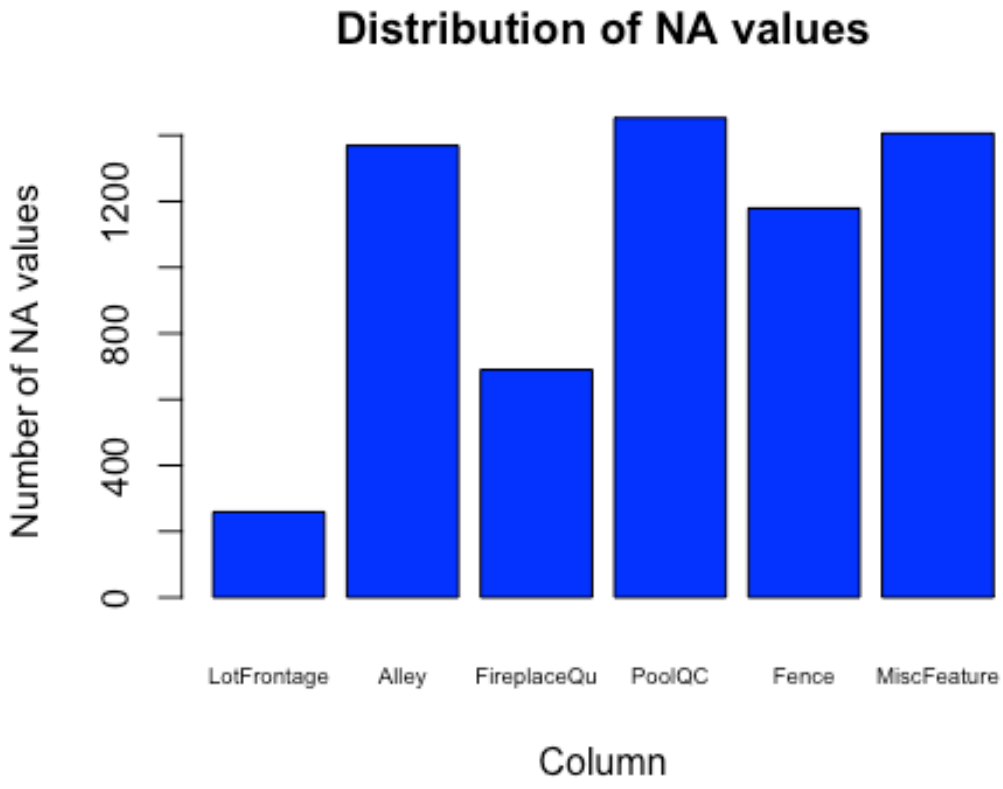
Part 1: Reading, Cleaning, and Splitting the Data

The code below reads the train dataset, which initially has 1460 rows and 80 predictors/columns. As the null values in the dataset have hindered us from fitting the data into predictive models, we obtained the NA count for each column. The barplot shows the columns which have NA counts that are over 100. We then deleted the columns that had over 100 NA values, although they had much larger number of NA values - 259, 1369, 690, 1453, 1179, and 1406 respectively.

```
set.seed(123)
train = read.csv("train.csv", header = T)
train$Id = NULL
dim(train)

## [1] 1460   80

na_count <- c(sapply(train, function(y) sum(is.na(y)))) # get NA count of eac
h column
na_count2 <- na_count[na_count > 100]
barplot(na_count2, main = "Distribution of NA values", xlab = "Column", cex.n
ames = 0.6,
        ylab = "Number of NA values", col = "blue", names.arg = rownames(na_c
ount2))
```




```
# delete categorical variables that have more than 100 NAs
train[ ,c('Alley', 'FireplaceQu', 'PoolQC', 'Fence', 'MiscFeature')] <- list(
NULL)
train$MiscVal = NULL # not categorical but to match deletion of MiscFeature
```

We continued the cleaning data process by substituting the datapoints in the remaining columns that have null values with their mean (for numerical variables) and mode (for categorical variables). We also deleted variables that had limited variation in data, as many of these variables hindered us from building our linear models - the limited levels of factors caused difficulty when stratifying the data. Additionally, we combined various house size metric variables to make new variables that measured the total size of the house, porch, and number of bathrooms, to give a more holistic picture.

```
colnames(train)[!complete.cases(t(train))] #remaining columns with NAs

##  [1] "LotFrontage"  "MasVnrType"   "MasVnrArea"   "BsmtQual"     "BsmtCond
"
##  [6] "BsmtExposure" "BsmtFinType1" "BsmtFinType2" "Electrical"   "GarageTy
pe"
## [11] "GarageYrBlt"  "GarageFinish" "GarageQual"   "GarageCond"

#function for finding mode
Mode <- function (x, na.rm) {
```

```r
  xtab <- table(x)
  xmode <- names(which(xtab == max(xtab)))
  if (length(xmode) > 1) xmode <- ">1 mode"
  return(xmode)
}

train$MasVnrArea[is.na(train$MasVnrArea)] <- 0 # substitute NA with 0 for Mas
VnrArea

# replace numerical variable NAs with mean and categorical variable NAs with
mode
for (var in 1:ncol(train)) {
  if (class(train[,var])=="integer") {
    train[is.na(train[,var]),var] <- mean(train[,var], na.rm = TRUE)
  } else if (class(train[,var]) %in% c("character", "factor")) {
    train[is.na(train[,var]),var] <- Mode(train[,var], na.rm = TRUE)
  }
}

# Remove variables with limited variation in data
# Out of 1460 datapoints,
sum(train$Street != 'Pave') # only 6 datapoints are not Pave
```

```
## [1] 6
```

```r
sum(train$Utilities != 'AllPub') # only 1 datapoint is not AllPub
```

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

```r
sum(train$Condition2 != 'Norm') # only 15 datapoints are not 'Norm'
```

```
## [1] 15
```

```r
sum(train$ExterCond != 'TA') # only 178 are not TA
```

```
## [1] 178
```

```r
sum(train$BsmtCond != 'TA') # only 112 are not TA
```

```
## [1] 112
```

```r
sum(train$BsmtFinSF2 != 0) # only 167 are not 0
```

```
## [1] 167
```

```r
sum(train$Heating != 'GasA') # only 32 are not GasA
```

```
## [1] 32
```

```r
sum(train$CentralAir != 'Y') # only 95 are not Y
```

```
## [1] 95
```

```r
sum(train$LowQualFinSF != 0) # only 26 are not 0
```

```
## [1] 26
```

```r
sum(train$KitchenAbvGr != 1) # only 68 are not 1
```

```
## [1] 68
```

```r
sum(train$Functional != 'Typ') # only 100 are not Typ
```

```
## [1] 100
```

```r
train[ ,c('Street', 'Utilities', 'Condition2', 'Exterior2nd', 'ExterCond', 'BsmtCond', 'BsmtFinSF2','GarageCars','Heating','HeatingQC', 'CentralAir','LowQualFinSF', 'KitchenAbvGr', 'Functional', 'GarageCond', 'GarageQual', 'PoolArea', 'Electrical')] <- list(NULL)

# Combining variables
train$TotalSF <- train$TotalBsmtSF + train$X1stFlrSF + train$X2ndFlrSF # obtaining total house size
# deleting individual sizes
train[ ,c('TotalBsmtSF', 'X1stFlrSF', 'X2ndFlrSF', 'BsmtFinSF1', 'BsmtFinSF2')] <- list(NULL)

train$TotalBathrooms <- train$FullBath + 0.5 * train$HalfBath +
  train$BsmtFullBath + 0.5 * train$BsmtHalfBath # getting total number of bathrooms
train[ ,c('FullBath', 'HalfBath', 'BsmtFullBath', 'BsmtHalfBath')] <- list(NULL)

train$TotalPorchSF <- train$OpenPorchSF + train$ScreenPorch + train$X3SsnPorch +
  train$EnclosedPorch + train$WoodDeckSF # getting total size of porches
train[ ,c('OpenPorchSF', 'ScreenPorch', 'X3SsnPorch', 'EnclosedPorch', 'WoodDeckSF')] <- list(NULL)
```

Since the test.csv from Kaggle does not come with the SalePrice values, we decided to create our own test data by splitting the given training data into 80% train and 20% test. We used stratified sampling to balance the data between the training and testing sets, so we can generate a better fit for the model and also prevent errors in fitting different factors between the datasets.

```r
library(splitstackshape)
rmse = function(a,b) {sqrt(mean((a-b)^2))}
#split train into train_1 and test_1
train_stra <- stratified(train, c('RoofMatl','Exterior1st'), 0.8, bothSets=TRUE)
train_1 = train_stra[[1]]
test_1 = train_stra[[2]]
```

Part 2: Statistical algorithm - Linear Regression

1. Fit a simple linear model: look at plots and check assumptions

```
simple_model = lm(SalePrice ~ . , data = train_1)
summary(simple_model)$r.squared
```

```
## [1] 0.9153568
```

```
rmse(train_1$SalePrice, predict(simple_model, train_1))
```

```
## [1] 23417.2
```

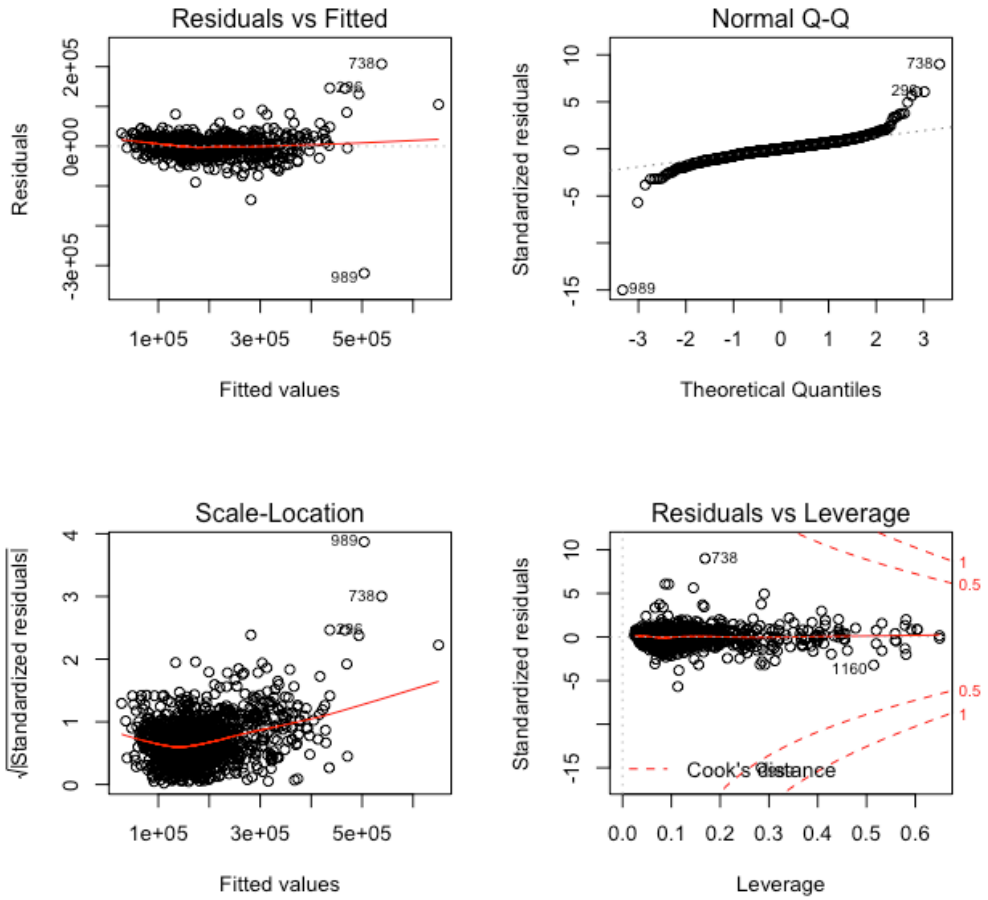
```
rmse(test_1$SalePrice, predict(simple_model, test_1))
```

```
## [1] 25587.8
```

```
par(mfrow = c(2, 2))
plot(simple_model)
```

```
## Warning: not plotting observations with leverage one:
##   1145, 1147, 1149, 1165, 1169, 1170, 1171, 1172
```

```
## Warning: not plotting observations with leverage one:
##   1145, 1147, 1149, 1165, 1169, 1170, 1171, 1172
```



Analysing plots and checking assumptions: From the Residuals vs Fitted plot, we see that the red line is approximately on the y=0 line and most datapoints are vertically evenly distributed relative to the red line so we can say that the relationship mostly satisfies the **linearity assumption**. However, there are a few outliers like points 738, 296, and 989 that do not follow this trend. Additionally, although the red line is slightly curved, there is no clear shape to the plots, so we believe that the **homoscedasticity assumption** is also met. However, the scale is actually very big for this plot, and we believe the residuals homoscedasticity could be more ideal with the log transformation. From the Normal QQ plot, we see that the datapoints mostly follow a straight line linear relationship, as the datapoints are mostly on the dotted trendline, and there is no clear curvature. However, the plot displays a heavy tail, as the points curve off in the extremities, meaning that our data has more extreme values than would be expected from a **Normal distribution**. Then, from the other two graphs of Scale-Location and Residuals vs Leverage, we see a commonality of having some extreme values in the data, as well as clear outliers like points 738, 296, 989, and 1160, as mentioned.

In order to scale down the dependent variables and allow our linear models to better fit the data, we decided to do a **log transformation** on the dependent variable Sale Price. We also decided to remove the common outliers across the plots of 738, 296, 989.

2. Apply log transformation, remove outliers, and fit a simple linear model

```
train_2 <- train_1[-c(296, 738, 989),]
test_2 <- test_1
train_2$SalePrice = log(train_2$SalePrice)
test_2$SalePrice = log(test_2$SalePrice)

simple_model_2 = lm(SalePrice ~ . , data = train_2)
summary(simple_model_2)$r.squared
```

```
## [1] 0.9376987
```

```
rmse(train_2$SalePrice, predict(simple_model_2, train_2))
```

```
## [1] 0.09924342
```

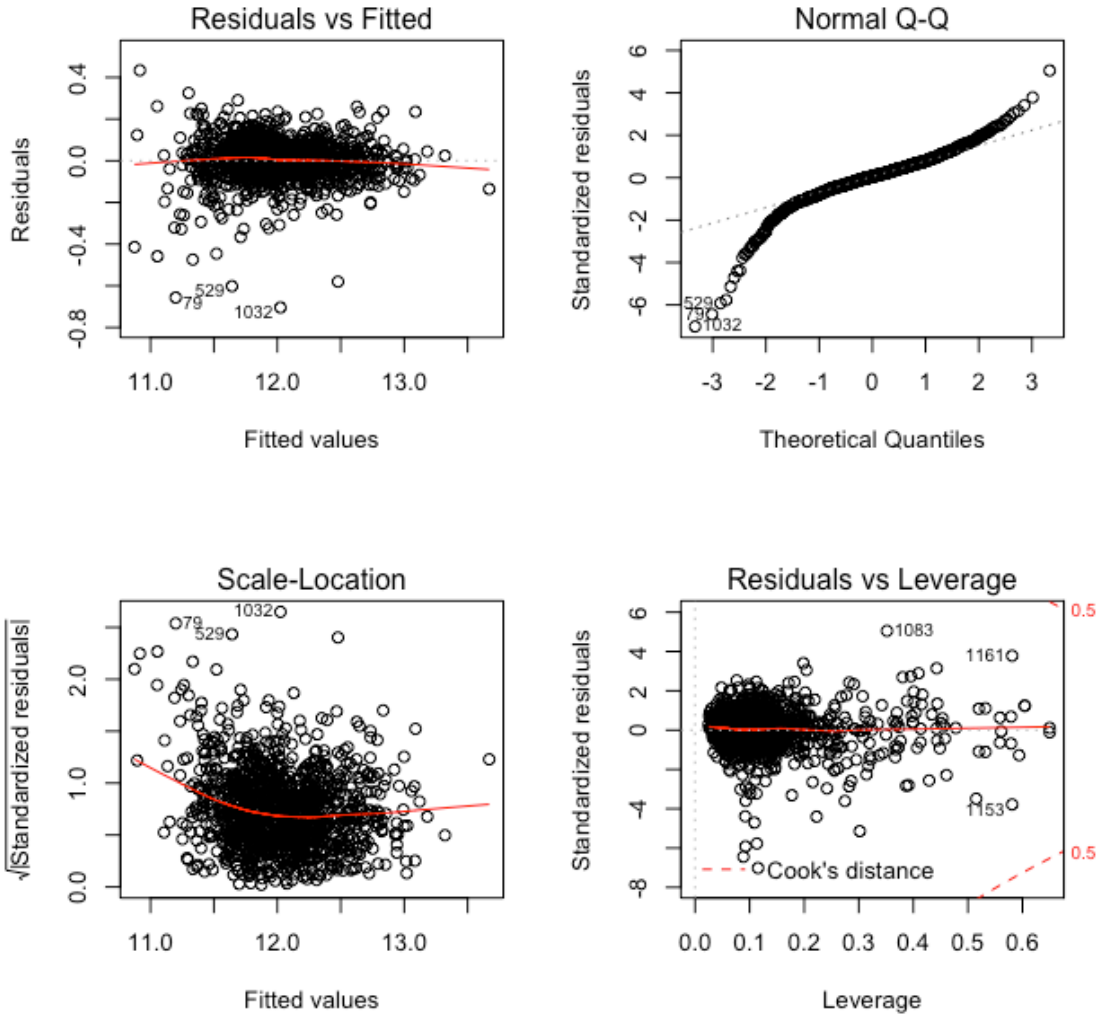
```
rmse(test_2$SalePrice, predict(simple_model_2, test_2))
```

```
## [1] 0.1157113
```

```
par(mfrow = c(2, 2))
plot(simple_model_2)
```

```
## Warning: not plotting observations with leverage one:
##   1142, 1144, 1146, 1162, 1166, 1167, 1168, 1169
```

```
## Warning: not plotting observations with leverage one:
##   1142, 1144, 1146, 1162, 1166, 1167, 1168, 1169
```



After applying the log transformation to our dependent variable, we can see that our Residuals vs Fitted plot has improved to satisfy the linearity and homoscedasticity assumption more closely. The red line remains close to y=0, the datapoints are more concentrated in the centre, and, although the datapoints appear to be more spread out than the Residuals vs Fitted plot before the transformation, the **scale** of the plot has actually decreased a significantly from around 1e+05 and -3e+05 to 0.4 and -0.8, showing how the datapoints are now closer to the red line. The scale of the normal QQ plot ha also decreased, as a few outliers and high leverage points have been removed. The datapoints are also closer to the linear trendline, although the heavy tail phenomenon remains. Finally, our RMSE values for train and test have scaled down as expected, and the R-squared value also increased.

3. Stepwise Subset Selection Regression

Surprisingly, the simple linear model that contains all the predictors which we ran above already gives us a low test RMSE of 0.1157. We decided to run the stepwise subset

selection regression to see whether we can get a lower RMSE with fewer predictors. Notice that from the formula of the stepwise model, we only use 35 out of 45 independent variables which give us the best AIC value. At the end, we did get a slightly lower test RMSE of 0.1123, although the train RMSE for the simple linear model was lower than the stepwise subset model, which means that the simple linear model might have overfitted the training data.

```
library(MASS)
# Stepwise regression: both forward and backward selection
step.model <- stepAIC(simple_model_2, direction = "both", trace = FALSE)
formula(step.model)
```

```
## SalePrice ~ MSZoning + LotFrontage + LotArea + LotConfig + LandSlope +
##     Neighborhood + Condition1 + BldgType + OverallQual + OverallCond +
##     YearBuilt + YearRemodAdd + RoofMatl + Exterior1st + MasVnrType +
##     Foundation + BsmtExposure + BsmtFinType2 + BsmtUnfSF + GrLivArea +
##     KitchenQual + Fireplaces + GarageType + GarageArea + PavedDrive +
##     YrSold + SaleType + SaleCondition + TotalSF + TotalBathrooms +
##     TotalPorchSF
```

```
summary(step.model)$r.squared
```

```
## [1] 0.9354283
```

```
rmse(train_2$SalePrice, predict(step.model, train_2))
```

```
## [1] 0.1010355
```

```
rmse(test_2$SalePrice, predict(step.model, test_2))
```

```
## [1] 0.1122776
```

4. Two Lasso models

We then fit a Lasso shrinkage model to see whether we can improve even more from overfitting.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 3.0-2
```

```
y_glm = train_2$SalePrice
train_glm = train_2
test_glm = test_2
train_glm$SalePrice = NULL
test_glm$SalePrice = NULL
combined_data  = rbind(train_glm, test_glm)
x_train_matrix = model.matrix(  ~ ., data = combined_data )[,-1]
x_test_matrix  = model.matrix(  ~ ., data = combined_data )[,-1]
```

```
train_r = 1:nrow(train_glm)
test_r  = (nrow(train_glm)+1):(nrow(test_glm)+nrow(train_glm))
x_train_matrix = x_train_matrix[train_r, ]
x_test_matrix  = x_test_matrix[test_r, ]

fit.lasso = cv.glmnet(x_train_matrix, y_glm, type.measure = "mse", nfolds = 1
0, alpha = 1)
lasso_train_pred = predict(fit.lasso, newx = x_train_matrix, s = fit.lasso$la
mbda.min)
rmse(train_2$SalePrice, lasso_train_pred)

## [1] 0.126295

lasso_test_pred = predict(fit.lasso, newx = x_test_matrix, s = fit.lasso$lamb
da.min)
rmse(test_2$SalePrice, lasso_test_pred)

## [1] 0.125156

# minor tweak: remove prediction outliers #####
train.yp <- predict(fit.lasso, newx = x_train_matrix)
m <- order(-abs(y_glm - train.yp))
x_train_matrix2 <- x_train_matrix[-head(m, nrow(train_glm) * 0.05), ]
y_train <- y_glm[-head(m, nrow(train_glm) * 0.05)]

# fit again
fit.lasso2 <- cv.glmnet(x_train_matrix2, y_train, type.measure = "mse", nfold
s = 10, alpha = 1)
lasso_train_pred2 = predict(fit.lasso2, newx = x_train_matrix, s = fit.lasso2
$lambda.min)
rmse(train_2$SalePrice, lasso_train_pred2)

## [1] 0.1387816

lasso_test_pred2 = predict(fit.lasso2, newx = x_test_matrix, s = fit.lasso2$l
ambda.min)
rmse(test_2$SalePrice, lasso_test_pred2)

## [1] 0.124344
```

Firstly, we ran a Lasso model with nfolds = 10. However, we saw that our test RMSE (0.1252) for lasso was higher than that of the stepwise model (0.1123). Thus, as the normal QQ plot shows that there are extreme values at the tails, we tried to remove 5% of outliers and fit the Lasso model again. However, although the Lasso model with outliers removed shows a minor test RMSE improvement (0.1243), it was still higher than the test RMSE of the stepwise model.

Therefore, our linear model which gives the lowest error is the **stepwise subset regression model** with a test RMSE of 0.1123.

Part 3: Machine Learning Models

1. Fit a random forest model

Using the `caret` package, we constructed a random forest and a neural network model and compared their RMSE. 10-fold cross validation is used with the metric of minimizing RMSE. The `train_2` dataset is used for training the models so that we may compare their RMSE on the same scale.

```
library(caret)
```

```
## Loading required package: lattice
```

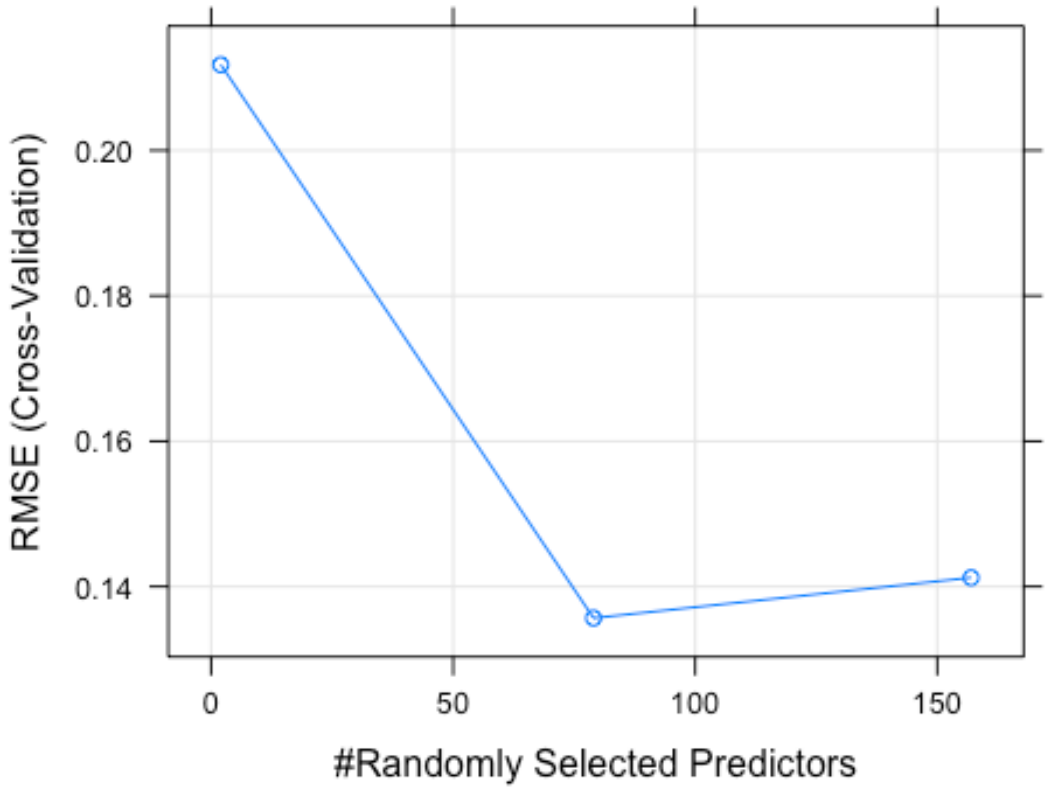
```
## Loading required package: ggplot2
```

```
control <- trainControl(method="cv", number=10)
metric <- "RMSE"
#Random Forest - caret does autotuning and uses the optimal cv from gridsearc
h
fit.rf <- train(SalePrice ~ ., data = train_2, method = "rf", trControl = con
trol, metric = metric)
print(fit.rf)
```

```
## Random Forest
##
## 1169 samples
##   45 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1053, 1051, 1052, 1053, 1052, 1053, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##     2   0.2117901  0.8102716  0.14636403
##    79   0.1356664  0.8830614  0.09119336
##   157   0.1412214  0.8716577  0.09639435
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 79.
```

We can see that the mtry of 79 gives the lowest RMSE of 0.1356664 and higheset R-squared that is 0.8830614. Thus, the ideal number of variable is randomly collected to be sampled at each split time is 79 for our model. We can also see the plot of RSME vs mtry below.

```
plot(fit.rf)
```



2. Fit a neural networks model

```
#Neural Network
fit.nn <- train(SalePrice ~ ., data = train_2, method = "nnet", trControl = c
ontrol, metric = metric)
```

Additionally, we also tried to fit the neural network model using the train function in caret. We thought the model would be interesting to try to use for our prediction. However, the RMSE calculated below shows that Neural Network is our worst performing model. Still, we believe it is useful to include in this analysis, which can form the baseline for further research and tuning.

Part 4: Comparing the test RMSEs of all models

```
c(SIMPLE = rmse(test_2$SalePrice, predict(simple_model_2, test_2)),
STEPWISE = rmse(test_2$SalePrice, predict(step.model, test_2)),
LASSO = rmse(test_2$SalePrice, lasso_test_pred2),
RF = rmse(test_2$SalePrice, predict(fit.rf, test_2)),
NN = rmse(test_2$SalePrice, predict(fit.nn, test_2)))

##     SIMPLE   STEPWISE      LASSO         RF         NN
##  0.1157113  0.1122776  0.1243440  0.1413393 11.0206102
```

Conclusion: The test RMSEs for our predictive models are summarised above. The linear regression models and random forest model all have test RMSEs below 0.15, which imply good predictive power, while the neural network model gives the worst test RMSE. Ultimately, the linear models give us better results than the random forest model, which has been automatically tuned for the optimal CV. Although we expected the random forest model to yield the best results, since it is the most complex, our models' results provide strong evidence that the data is actually very close to linear and so a linear model is legitamately a better fit. We also see from our residuals plot and Normal QQ plot that the data mostly follows a linear relationship, despite displaying heavier tails and some extreme values. It could also be the case that the random forest model was overfitting to the training data and could have been optimized even further. In the end, the **stepwise subset regression** model gave us the lowest test RMSE and is our optimal model.