Regression Analysis

Cordero Perez

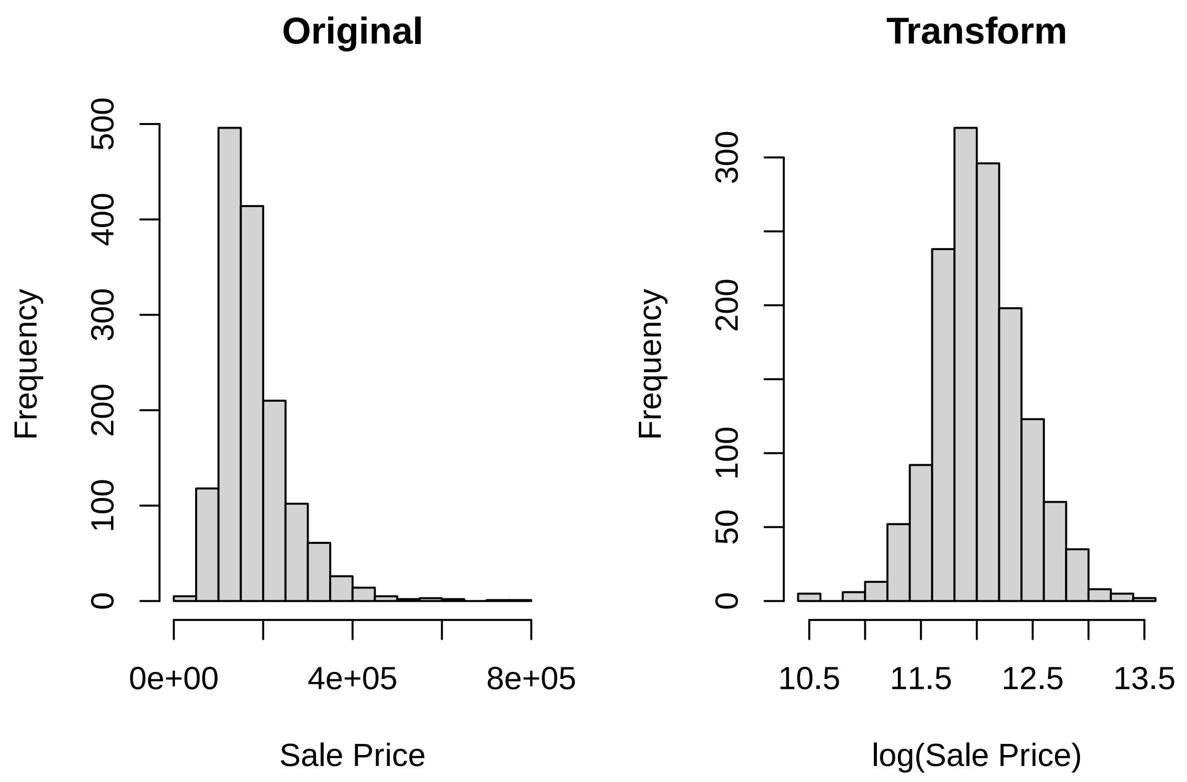
Data Multiple features for houses sold in 2017 was collected, along with the sale price. I will be using the sale price as the response variable, and 40 quantitative variables as the features. Missing data points were replaced by the mean of the remaining values in the feature to avoid bias and keep observations. Two variables required recoding to a numeric, and the final result was then standardized according to the function created in the script. Here P (**features**) = 40 (inclusive of intercept) and n (**observations**) = 1460. The data used for this project can be found at (https://www.kaggle.com/alphaepsilon/housing-prices-dataset#train.csv).

Purpose This analysis compares the following 5 methods of regression (**4 linear, 1 non-linear**) via implementation on the given dataset:

1. Ordinary Least Squares
    - Ordinary Least Squares
2. Ridge
    - Special Case of Elastic-Net
3. Lasso
    - Special Case of Elastic-Net
4. Elastic-Net
5. Random Forest
    - Non-linear

The following code shows how the data was cleaned and standardized, and provides a sample view of the data afterwards. A thing to note is that while the log transform of the sale price removes skewness from the response variable, there is not much benefit to normalizing the response variable in this excercise. Typically the skewness in response variables are addressed for things like salary because the ranges are more likely to differ in magnitude. Although this analysis uses sale price, which is similar, the homes are all 6 figure sales and therefore do not differ in magnitude. The check for this skewness and the result of the transform can be seen in the plots between the code below.

```r
# Regression Analysis using 2017 Home Data from Kaggle


# reset global environment
rm(list = ls(all.names = TRUE))

# libraries
library(tidyverse)
library(knitr)
library(PerformanceAnalytics)
library(reshape2)
library(caret)
library(glmnet)
library(randomForest)
library(kableExtra)
library(gridExtra)


#----functions---------------------------------

#standardized in accordance with statistical learning methods
standardize <- function(vector){

  numerator <- vector
  denominator <- sqrt(mean((vector-mean(vector))**2))

  return(numerator/denominator)

}


#----cleaning-------------------------------------

# load data
I <- read_csv("~/Documents/R/RProjects-Public/Regression-Analysis-Data/homes_data.csv")
homes_load <- I

# select 40 key columns (qualitative) from the data
homes_full <- cbind(select_if(homes_load, .predicate =  is.double),
                    select(homes_load, Utilities, CentralAir, Street)) %>% select(-Id)

# recode the last 3 variable to get 0,1
homes_full$Utilities <- as.numeric(recode(homes_full$Utilities,
                                          "AllPub" = "1", "NoSeWa" = "0"))
homes_full$CentralAir <- as.numeric(recode(homes_full$CentralAir,
                                           "Y" = "1", "N" = "0"))
homes_full$Street <- as.numeric(recode(homes_full$Street,
                                       "Pave" = "1", "Grvl" = "0"))

#replace NA values in LotFrontage
for (i in 1:length(homes_full$LotArea)){

  if (is.na(homes_full$LotFrontage[i])){
    homes_full$LotFrontage[i] <- mean(homes_full$LotFrontage, na.rm = TRUE)
  }

  if (is.na(homes_full$GarageYrBlt[i])){
    homes_full$GarageYrBlt[i] <- mean(homes_full$GarageYrBlt, na.rm = TRUE)
  }

  if (is.na(homes_full$MasVnrArea[i])){
    homes_full$MasVnrArea[i] <- mean(homes_full$MasVnrArea, na.rm = TRUE)
  }
}

# SalePrice is skewed right: using log(SalePrice) for model
par(mfrow = c(1,2))
hist(homes_full$SalePrice, main = "Original", xlab = "Sale Price")
hist(log(homes_full$SalePrice), main = "Transform", xlab = "log(Sale Price)")
```



```
SalePrice <- homes_full$SalePrice
 #remove comment and delete above to run with log transform
#SalePrice <- log(homes_full$SalePrice)

#standardize the independent columns of the data and combine the response variable back
homes_full <- select(homes_full,-SalePrice) %>% mutate_all(.funs = standardize)
homes_full <- cbind(SalePrice,homes_full)

#create full dataset with errors for bootstrap error bar construction
 homes_full_bs <- homes_full

kable(homes_full[1:5,c(2:5,8:9)],
      caption = "Full Size: 1460 x 40 | Data Sample After Standardizing") %>%
  kable_styling()
```

In the next section 100 pairs of training and test sets are created and stored within corresponding lists. The

Table 1: Full Size: 1460 x 40 | Data Sample After Standardizing

MSSubClass	LotFrontage	LotArea	OverallQual	YearRemodAdd	MasVnrArea
1.4189066	2.952334	0.8468762	5.063207	97.05240	1.0858289
0.4729689	3.633642	0.9621315	4.339892	95.74416	0.0000000
1.4189066	3.088596	1.1274978	5.063207	97.00395	0.8974708
1.6553910	2.725231	0.9571204	5.063207	95.45344	0.0000000
1.4189066	3.815324	1.4291662	5.786523	96.90704	1.9389801

purpose of this is to capture 100 samples of what are typically static error statistics and identify a distribution among them. Cross-validation is used within each training set to identify the best lambda (here best refers to the lambda that minimizes the cross-validation error) to serve as the model parameter, however only one R-squared statistic is produced. This statistic changes depending upon the training and test sets provided, hence producing 100 measures of this statistic, and viewing the distribution, allows us to accurately measure this metric for each method.

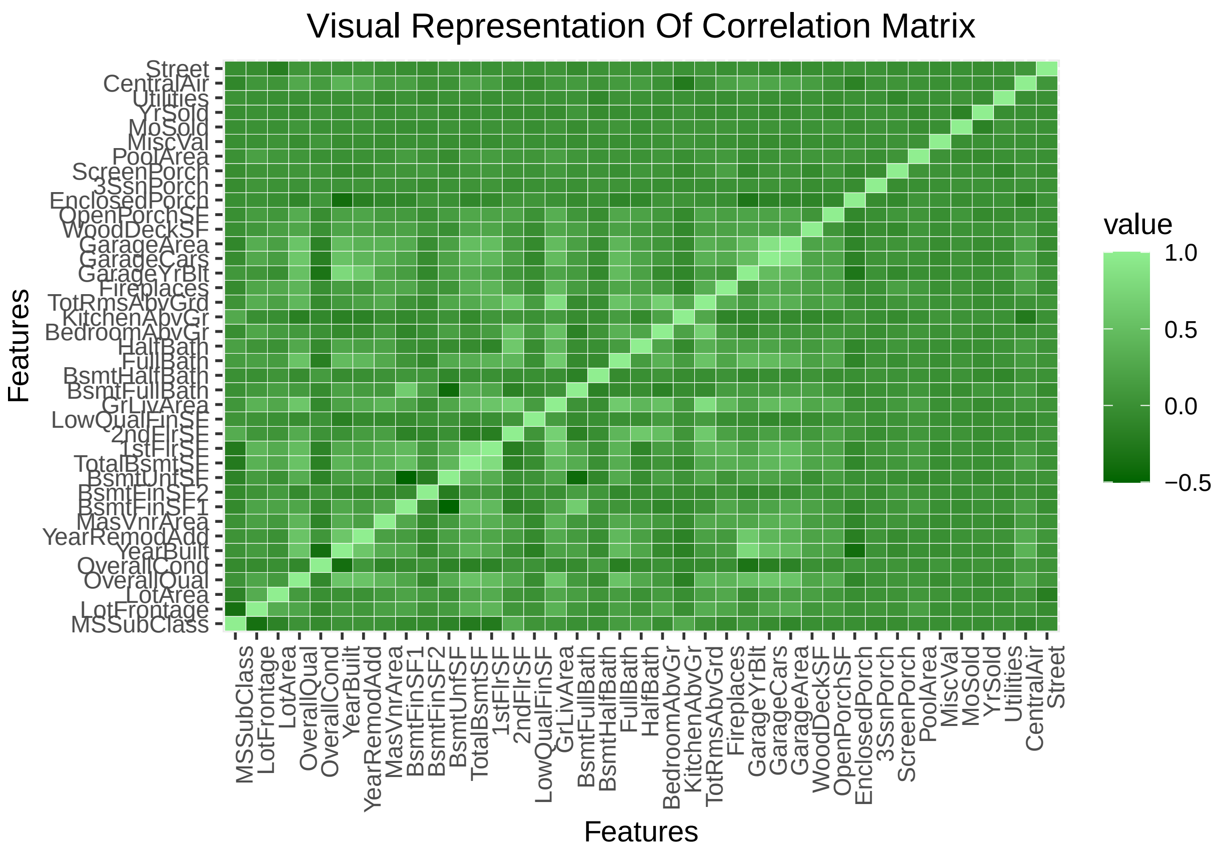
This can be thought of as manual cross-validation to identify the true R-squared and within each manual split there is a cross-validation model to identify the best lambda based on that split. This will be seen later within the code for each model.

The reason for splitting and saving the 100 sets here, rather than looping through them, becomes apparent when viewing the processing times discussed later in the document. Minimizing the amount of loops greatly reduces the processing time at the expense of memory. Seeing as this dataset is rather small, the memory is not a concern and maximizing processing time leads to increased efficiency.

```r
#----data split-------------------------------------------------

# seed values and empty vector creation
seed_values <- seq(200,800, length.out = 100)

homes_train <- list(c("1"))
homes_test <- list(c("1"))

X_train <- list(c("1"))
Y_train <- list(c("1"))
X_test <- list(c("1"))
Y_test <- list(c("1"))

# set seed within loop for reproducibility: 100 repetitions for eror boxplot evaluation
for (i in 1:100){
  set.seed(seed_values[i])

  # split data using caret package, step 1: create train indices
  train_indices <- as.vector(createDataPartition(homes_full$SalePrice,
                                                  p = .8, list = FALSE, times = 1))

  # split accoring to train indices
  homes_train[[i]] <- homes_full[train_indices,]
  homes_test[[i]] <- homes_full[-train_indices,]

  # step 3: split further into matrix sets.
  X_train[[i]] <- select(homes_train[[i]],-SalePrice)
  Y_train[[i]] <- homes_train[[i]]$SalePrice
  X_test[[i]] <- select(homes_test[[i]], -SalePrice)
  Y_test[[i]] <- homes_test[[i]]$SalePrice

} # end of loop

# sample sizes
n_train <- length(train_indices)
n_test <- length(homes_full$SalePrice) - n_train
```

The following is essentially a heat map to show the correlation between features prior to running the regression. As shown below, by lighter squares, there are some very high correlations between features and hence demonstrate some features are not essential.

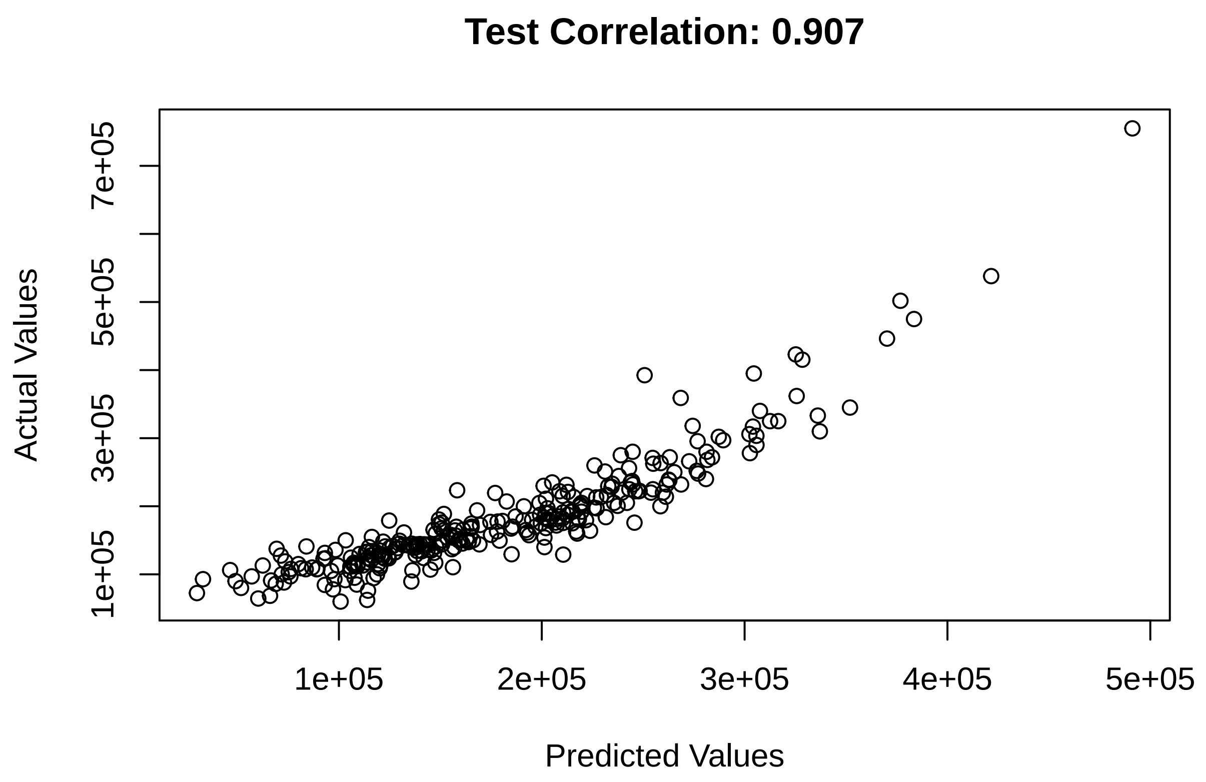
```
#----collinearity check amongst features-----------------------

# set up matrix view of the features
correlation_matrix <- round(cor(select(homes_full, -SalePrice)),2)

# graph the correlation values using shape package
ggplot(data = melt(correlation_matrix), mapping = aes(x = Var1, y = Var2,
                                                      fill = value))+
  geom_tile(color = "white")+xlab("Features")+ylab("Features")+
  scale_fill_gradient(low = "darkgreen", high = "lightgreen")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        plot.title = element_text(hjust = .5))+
  labs(title = str_to_title("visual representation of correlation matrix"))
```



OLS regression was conducted as a basis of comparison for the extensions (Ridge, Lasso, and Elastic- Net) that follow. The following shows the OLS regression code along with the histogram of test residuals. The correlation between the test set and the predictions should be closer to 1 if the model is good, and that value here is present as the title of the histogram. This is a pretty high correlation and shows that a linear model may be sufficient for decent predicitions.

```r
#----linear regression model------------------------------------
# form: BetaHat(LR) = argmin(Beta) [(1/n)* ||Y-XBeta|| 2 ]  | using homes_train[[1]]

# model fit and test
model_lr <- lm(SalePrice ~ ., homes_train[[1]])
Beta_star <- as.matrix(model_lr$coefficients) #beta star values for matrix

# model coefficient check, should NA values be present, variables need to be dropped
# as they are highly collinear with other variables
removal <- (which(is.na(Beta_star)))

#remove variables at indices 13 and 17
homes_train_lr <- select(homes_train[[1]], -removal)

# new model
model_lr_new <- lm(SalePrice ~ ., homes_train_lr)
Beta_star <- as.matrix(model_lr_new$coefficients) #beta star values for matrix
```

```r
# set values using model parameters
Y_train_hat_lr <- as.matrix(cbind(rep(1,length(train_indices)),
                                  X_train[[1]][,-removal])) %*% Beta_star
Y_test_hat_lr <- as.matrix(cbind(rep(1,length(homes_full$SalePrice)-length(train_indices)),
                                 X_test[[1]][,-removal])) %*% Beta_star

# errors for linear model
rse_train_lr <- sqrt(mean((Y_train_hat_lr - Y_train[[1]])**2))
rse_test_lr <- sqrt(mean((Y_test_hat_lr - Y_test[[1]])**2))

# test correlation
tcor <- round(cor(as.vector(Y_test_hat_lr), as.vector(Y_test[[1]])),3)

# quick view of errrors
par(mfrow = c(1,1))
plot(Y_test_hat_lr, Y_test[[1]], main = paste0("Test Correlation:"," ",tcor),
     xlab = "Predicted Values", ylab = "Actual Values")
```



The following code creates the models, stores the R-squared values, and stores the residuals for the 100th iteration.

```r
#----elastic net models--------------------------------------
# form: BetaHat(EN) = argmin(Beta) [(1/n)* ||Y-XBeta|| 2 + lambda* r(Beta)]
#     : r(Beta) = [ (1-alpha)*(1/2) ||Beta|| 2 + alpha*||Beta|| 1 ]
#     : Ridge (alpha = 0), Lasso (alpha = 1), Elastic_Net (alpha = arbitrary)


#----ridge----

# create empty R2 vector
R2_train_ridge <- rep(1,100)
R2_test_ridge <- rep(1,100)
rse_train_ridge <- rep(1,100)
rse_test_ridge <- rep(1,100)

# create empty matrix to store bootstrap methods
boots_ridge <- matrix(0, 39, 100)

# start time for loop to evaluate method efficiency
ridge_start <- Sys.time()

#for loop to run 100 times
for (i in 1:100){

  # cast data as matrices
  X_train <- as.matrix(select(homes_train[[i]],-SalePrice))
  Y_train <- as.matrix(homes_train[[i]]$SalePrice)
  X_test <- as.matrix(select(homes_test[[i]], -SalePrice))
  Y_test <- as.matrix(homes_test[[i]]$SalePrice)


  # 10 fold cv model to generate lambda that minimizes cross-validation error
  cv_model_ridge <- cv.glmnet(x = X_train, y = Y_train, alpha = 0)

  # create ridge model with cv lambda
  model_ridge <- glmnet(x = X_train, y = Y_train, alpha = 0,
                       lambda = cv_model_ridge$lambda.min)

  # generate y_hat matrices for the train and test sets
  Y_train_hat_ridge <- predict.glmnet(model_ridge, newx = X_train, type = "response")
  Y_test_hat_ridge <- predict.glmnet(model_ridge, newx = X_test, type = "response")

  # errors for linear model
  rse_train_ridge[i] <- sqrt(mean((Y_train_hat_ridge - Y_train)**2))
  rse_test_ridge[i] <- sqrt(mean((Y_test_hat_ridge - Y_test)**2))

  # R**2 calculation
  R2_train_ridge[i] <- model_ridge$dev.ratio #pull directly from the model
  R2_test_ridge[i] <- 1 - (mean((Y_test_hat_ridge - Y_test)**2)/var(Y_test))

  # cbind bootstrap samples
  boots_ridge[,i] <- as.vector(model_ridge$beta)

} # end of loop

# end time for loop
ridge_stop <- ridge_start - Sys.time()
```

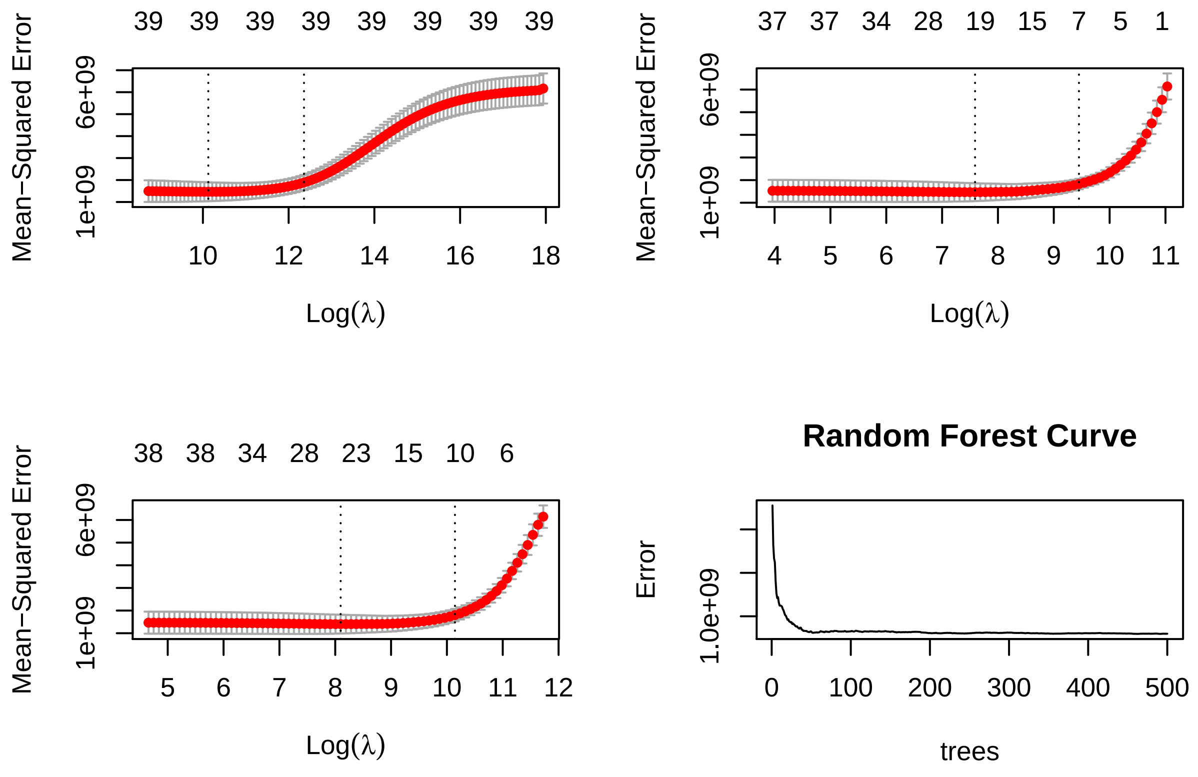
```r
# calculate bootstrap standard errors
boots_ridge_sd <- apply(boots_ridge, 1, "sd")

#----lasso----

# create empty R2 vector
R2_train_lasso <- rep(1,100)
R2_test_lasso <- rep(1,100)
rse_train_lasso <- rep(1,100)
rse_test_lasso <- rep(1,100)

# create empty matrix to store bootstrap methods
boots_lasso <- matrix(0, 39, 100)

# start time for loop to evaluate method efficiency
lasso_start <- Sys.time()


#for loop to run 100 times
for (i in 1:100){

  # cast data as matrices
  X_train <- as.matrix(select(homes_train[[i]],-SalePrice))
  Y_train <- as.matrix(homes_train[[i]]$SalePrice)
  X_test <- as.matrix(select(homes_test[[i]], -SalePrice))
  Y_test <- as.matrix(homes_test[[i]]$SalePrice)


  # 10 fold cv model to generate lambda that minimizes cross-validation error
  cv_model_lasso <- cv.glmnet(x = X_train, y = Y_train, alpha = 1)

  # create ridge model with cv lambda
  model_lasso <- glmnet(x = X_train, y = Y_train, alpha = 1,
                        lambda = cv_model_lasso$lambda.min)

  # generate y_hat matrices for the train and test sets
  Y_train_hat_lasso <- predict.glmnet(model_lasso, newx = X_train, type = "response")
  Y_test_hat_lasso <- predict.glmnet(model_lasso, newx = X_test, type = "response")

  # errors for linear model
  rse_train_lasso[i] <- sqrt(mean((Y_train_hat_lasso - Y_train)**2))
  rse_test_lasso[i] <- sqrt(mean((Y_test_hat_lasso - Y_test)**2))

  # R**2 calculation
  R2_train_lasso[i] <- model_lasso$dev.ratio #pull directly from the model
  R2_test_lasso[i] <- 1 - (mean((Y_test_hat_lasso - Y_test)**2)/var(Y_test))

  # cbind bootstrap samples
  boots_lasso[,i] <- as.vector(model_lasso$beta)

} # end of loop

# end time for loop
```

```r
lasso_stop <- lasso_start - Sys.time()

# calculate bootstrap standard errors
boots_lasso_sd <- apply(boots_lasso, 1, "sd")


#----elastic net----

# data previoulsy converted to matrix format

# create empty R2 vector
R2_train_en <- rep(1,100)
R2_test_en <- rep(1,100)
rse_train_en <- rep(1,100)
rse_test_en <- rep(1,100)

# create empty matrix to store bootstrap methods
boots_en <- matrix(0, 39, 100)

# start time for loop to evaluate method efficiency
en_start <- Sys.time()

#for loop to run 100 times
for (i in 1:100){

  # cast data as matrices
  X_train <- as.matrix(select(homes_train[[i]],-SalePrice))
  Y_train <- as.matrix(homes_train[[i]]$SalePrice)
  X_test <- as.matrix(select(homes_test[[i]], -SalePrice))
  Y_test <- as.matrix(homes_test[[i]]$SalePrice)


  # 10 fold cv model to generate lambda that minimizes cross-validation error
  cv_model_en <- cv.glmnet(x = X_train, y = Y_train, alpha = 0.5)

  # create ridge model with cv lambda
  model_en <- glmnet(x = X_train, y = Y_train, alpha = 0.5,
                     lambda = cv_model_en$lambda.min)

  # generate y_hat matrices for the train and test sets
  Y_train_hat_en <- predict.glmnet(model_en, newx = X_train, type = "response")
  Y_test_hat_en <- predict.glmnet(model_en, newx = X_test, type = "response")

  # errors for linear model
  rse_train_en[i] <- sqrt(mean((Y_train_hat_en - Y_train)**2))
  rse_test_en[i] <- sqrt(mean((Y_test_hat_en - Y_test)**2))

  # R**2 calculation
  R2_train_en[i] <- model_en$dev.ratio #pull directly from the model
  R2_test_en[i] <- 1 - (mean((Y_test_hat_en - Y_test)**2)/var(Y_test))

  # cbind bootstrap samples
  boots_en[,i] <- as.vector(model_en$beta)
} # end of loop

# end time for loop
en_stop <- en_start - Sys.time()

# calculate bootstrap standard errors
boots_en_sd <- apply(boots_en, 1, "sd")


#----random forest model--------------------------------------------

# create empty R2 vector
R2_train_rf <- rep(1,100)
R2_test_rf <- rep(1,100)
rse_train_rf <- rep(1,100)
rse_test_rf <- rep(1,100)

# create
boots_rf <- matrix(0, 39, 100)

# start time for loop to evaluate method efficiency
rf_start <- Sys.time()

#for loop to run 100 times
for (i in 1:100){

  # cast data as matrices
  X_train <- as.matrix(select(homes_train[[i]],-SalePrice))
  Y_train <- as.matrix(homes_train[[i]]$SalePrice)
  X_test <- as.matrix(select(homes_test[[i]], -SalePrice))
  Y_test <- as.matrix(homes_test[[i]]$SalePrice)

  # create random forest model with sqrt(p) parameter, here p = 40
  model_rf <- randomForest(x = X_train, y = Y_train, importance = TRUE)

  # generate y_hat matrices for the train and test sets
  Y_train_hat_rf <- predict(model_rf, newdata = X_train)
  Y_test_hat_rf <- predict(model_rf, newdata = X_test)

  # errors for linear model
  rse_train_rf[i] <- sqrt(mean((Y_train_hat_rf - Y_train)**2))
  rse_test_rf[i] <- sqrt(mean((Y_test_hat_rf - Y_test)**2))

  # R**2 calculation
  R2_train_rf[i] <- 1 - (mean((Y_train_hat_rf - Y_train)**2)/var(Y_train))
  R2_test_rf[i] <- 1 - (mean((Y_test_hat_rf - Y_test)**2)/var(Y_test))

  # cbind bootstrap samples
  boots_en[,i] <- as.vector(model_rf$importance[,1])

} # end of loop

# end time for loop
```

```r
rf_stop <- rf_start - Sys.time()

# calculate bootstrap standard errors
boots_rf_sd <- apply(boots_rf, 1, "sd")
```

The following plots show the cross-validation and error curves for the 4 models:

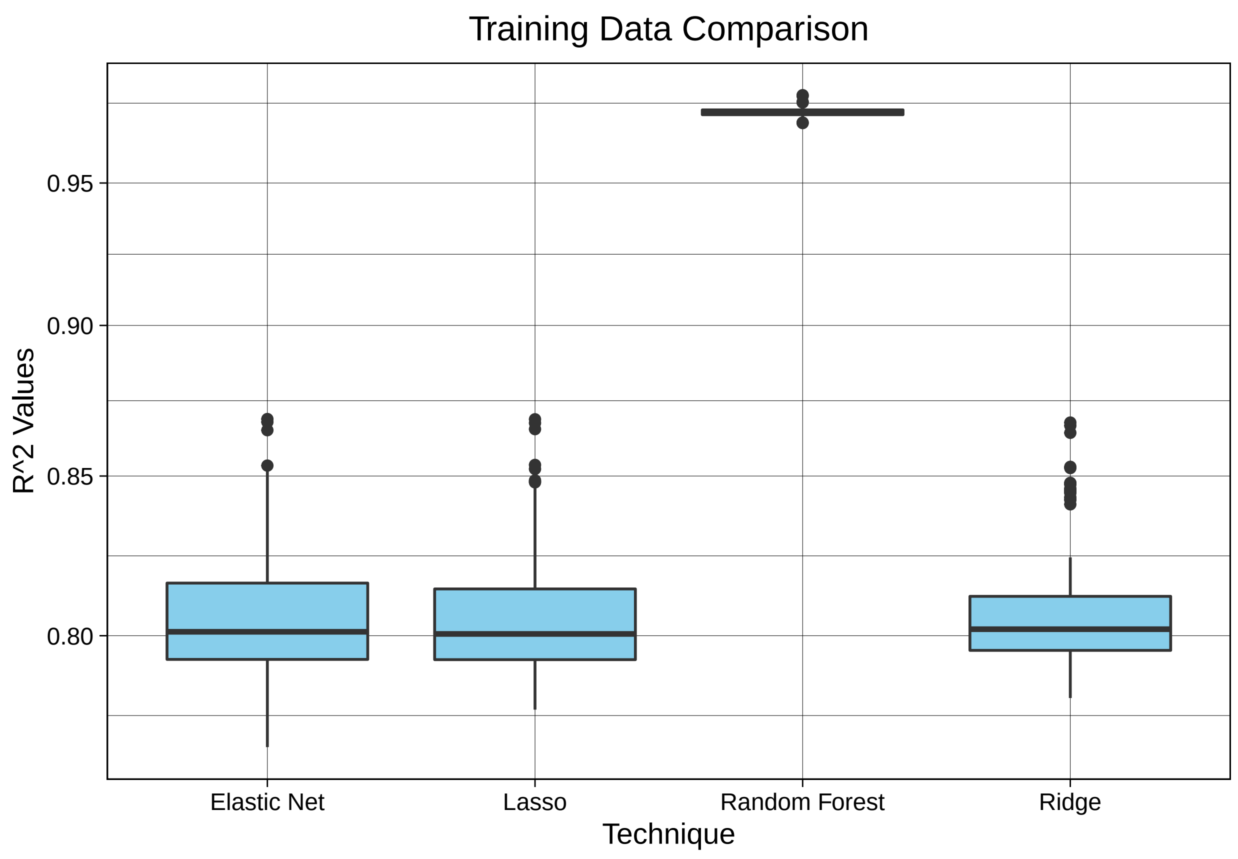
```r
# view cross validation plots for the last cv models generated
par(mfrow = c(2,2))
plot(cv_model_ridge)
plot(cv_model_lasso)
plot(cv_model_en)
plot(model_rf, main = "Random Forest Curve")
```



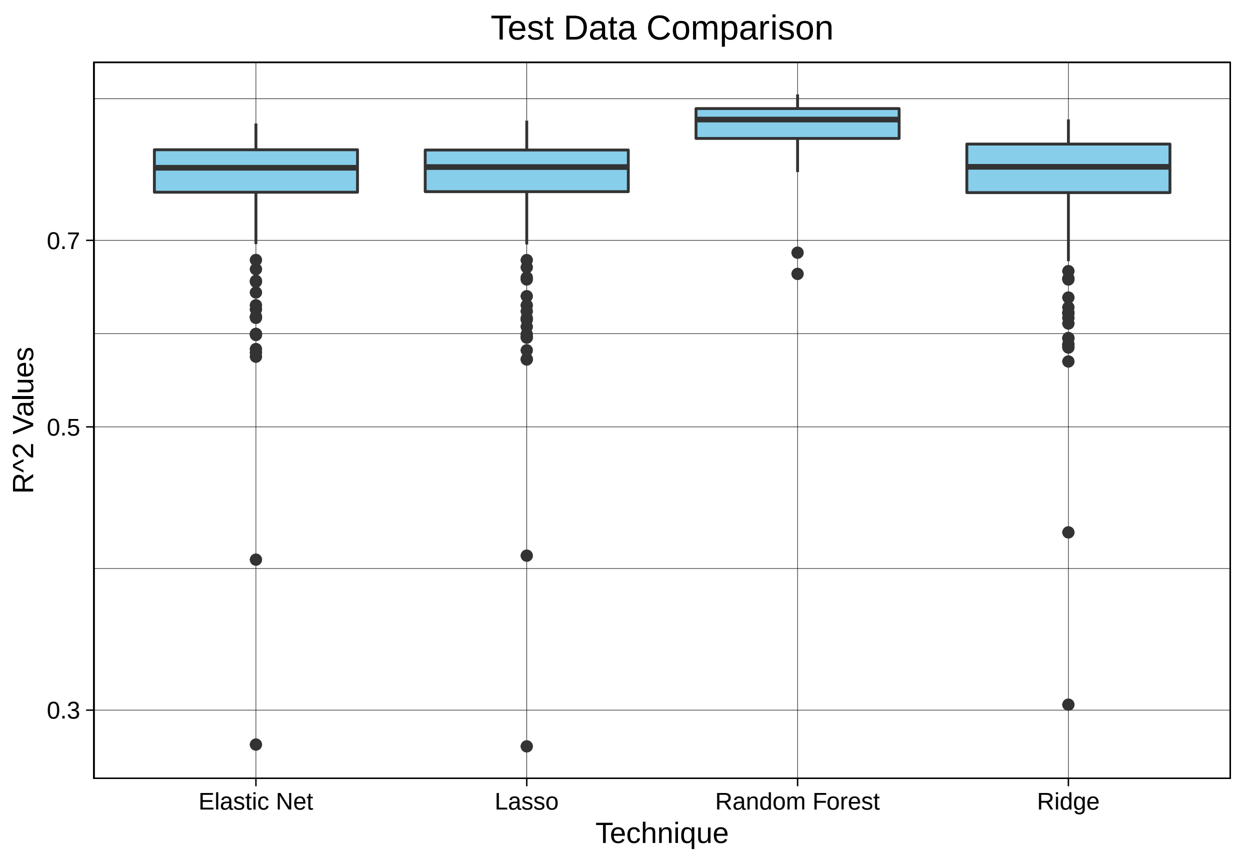
```r
par(mfrow = c(1,1))
```

The following plots show the distribution of train and test R_squared values over 100 iterations:

```r
#aggregate and tidy R2 data
R_sq_train <- as.data.frame(cbind(R2_train_ridge,R2_train_lasso, R2_train_en,
                                  R2_train_rf))
names(R_sq_train) <- c("Ridge", "Lasso", "Elastic Net", "Random Forest")
R_sq_test <- as.data.frame(cbind(R2_test_ridge,R2_test_lasso, R2_test_en,
                                 R2_test_rf))
names(R_sq_test) <- c("Ridge", "Lasso", "Elastic Net", "Random Forest")

R_sq_train <- gather(R_sq_train, key = "Technique", value = "R2")
R_sq_test <- gather(R_sq_test, key = "Technique", value = "R2")

# view boxplots of R2 values
ggplot(data = R_sq_train, mapping = aes(x = Technique, y = R2, group = Technique)) +
  geom_boxplot(mapping = aes(group = Technique), fill = "skyblue")+
  ylab("R^2 Values")+
  scale_y_log10()+
  theme_linedraw()+
  labs(title = str_to_title("Training Data Comparison"))+
  theme(plot.title = element_text(hjust = .5))
```



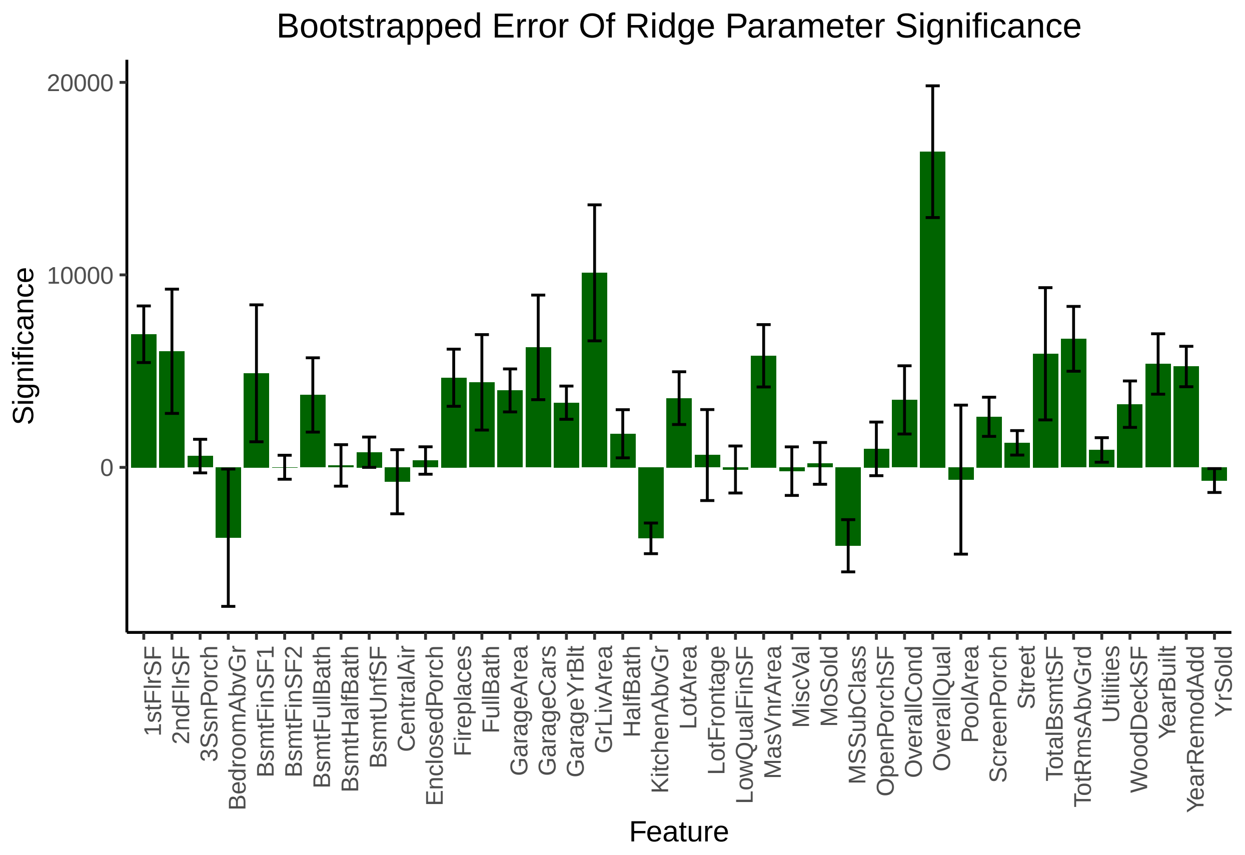
```r
ggplot(data = R_sq_test, mapping = aes(x = Technique, y = R2, group = Technique)) +
  geom_boxplot(mapping = aes(group = Technique), fill = "skyblue")+
  ylab("R^2 Values")+
  scale_y_log10()+
  theme_linedraw()+
  labs(title = str_to_title("Test Data Comparison"))+
  theme(plot.title = element_text(hjust = .5))
```



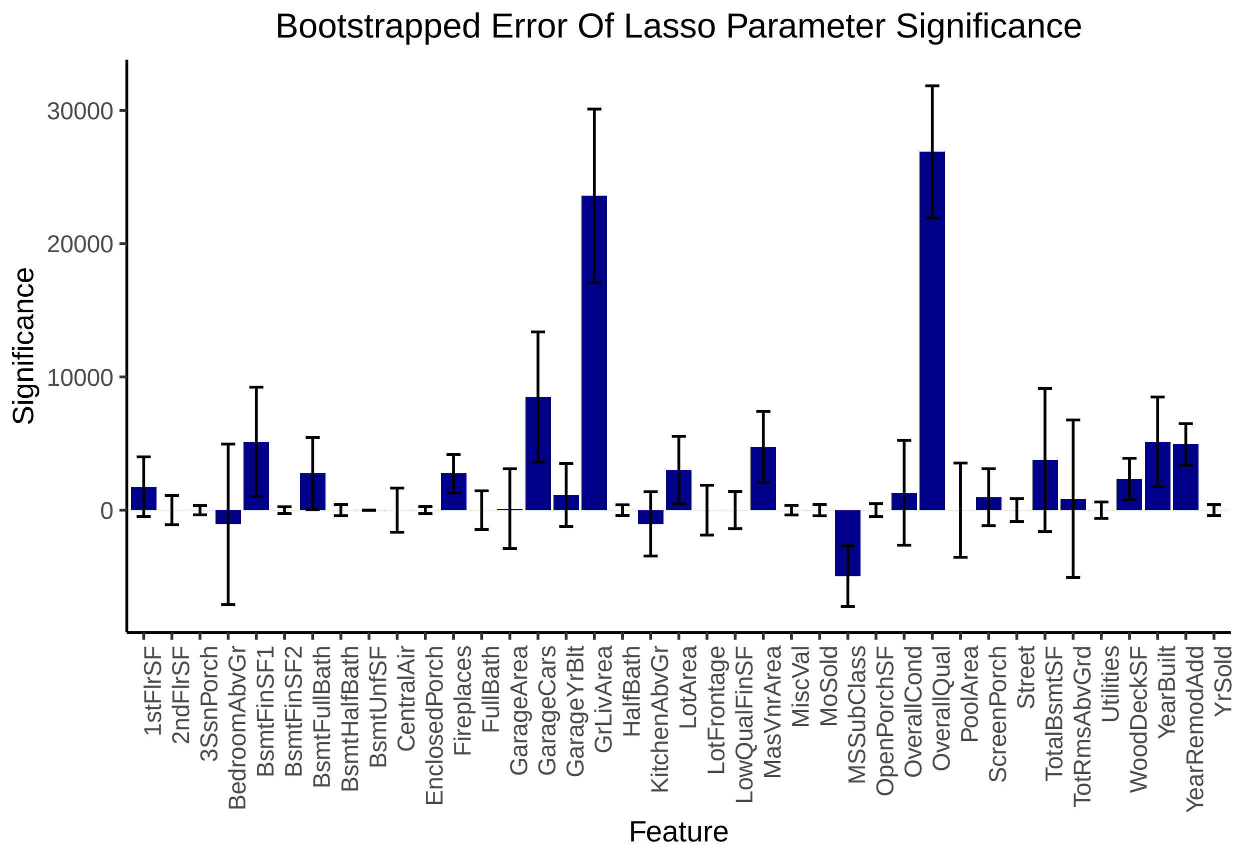
The following plots show the barplots of parameter significance with bootstrapped error bars:

```r
#----barplot with bootstrap errors-------------------------

# ridge
model_ridge_full <- glmnet(x =as.matrix(homes_full_bs[,2:40]),
                           y = as.matrix(homes_full_bs[,1]), alpha = 0,
                           lambda = cv_model_ridge$lambda.min)
bsp_df_ridge <- data.frame(rep("Ridge",39), names(homes_full_bs)[2:40],
                           as.vector(model_ridge_full$beta), 2*boots_ridge_sd)
names(bsp_df_ridge) <- c("Method","Feature", "Significance", "Error (95% CI)")

# lasso
model_lasso_full <- glmnet(x =as.matrix(homes_full_bs[,2:40]),
                           y = as.matrix(homes_full_bs[,1]), alpha = 1,
                           lambda = cv_model_lasso$lambda.min)
bsp_df_lasso <- data.frame(rep("Lasso",39), names(homes_full_bs)[2:40],
                           as.vector(model_lasso_full$beta), 2*boots_lasso_sd)
names(bsp_df_lasso) <- c("Method", "Feature", "Significance", "Error (95% CI)")

#elastic net
```

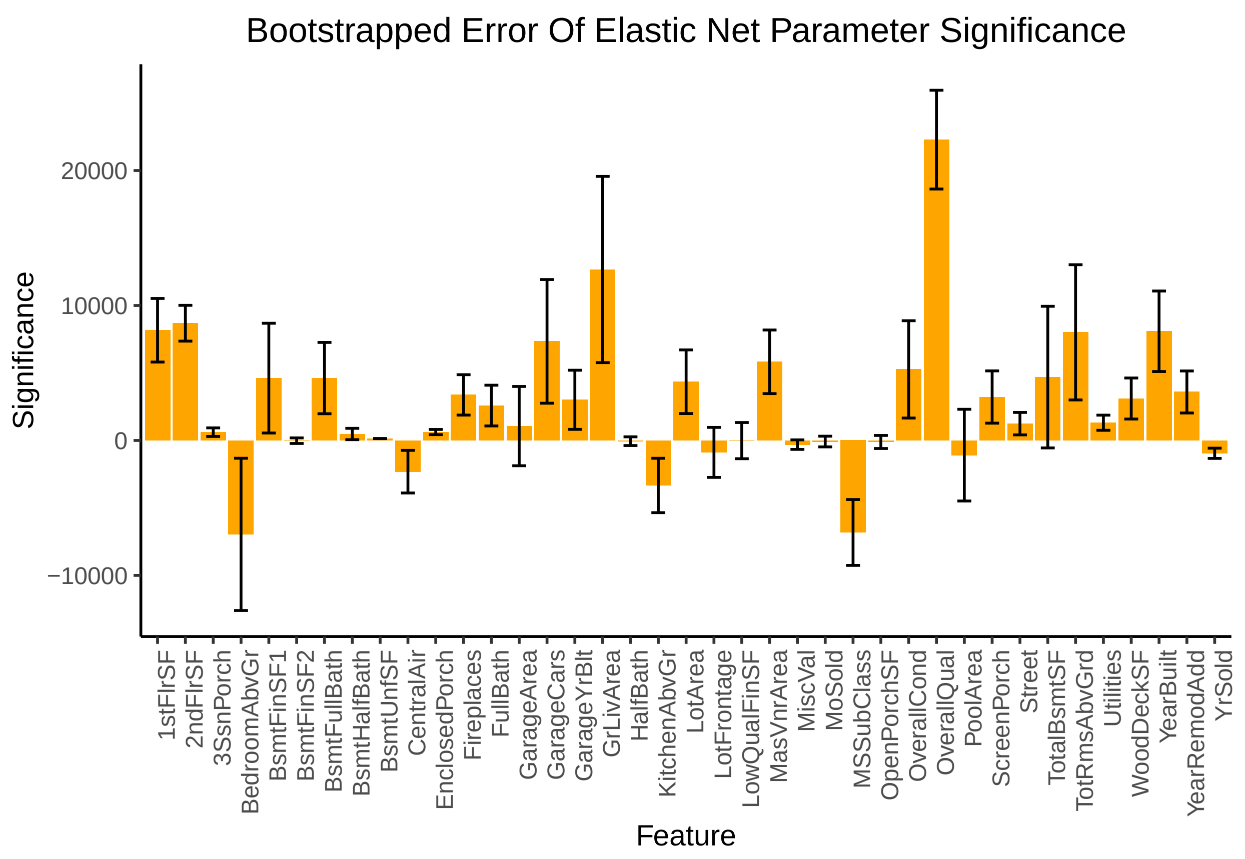
```r
model_en_full <- glmnet(x =as.matrix(homes_full_bs[,2:40]),
                        y = as.matrix(homes_full_bs[,1]), alpha = 0,
                        lambda = cv_model_en$lambda.min)
bsp_df_en <- data.frame(rep("Elastic Net",39), names(homes_full_bs)[2:40],
                        as.vector(model_en_full$beta), 2*boots_en_sd)
names(bsp_df_en) <- c("Method","Feature", "Significance", "Error (95% CI)")

# random forest
model_rf_full <- randomForest(x =as.matrix(homes_full_bs[,2:40]),
                              y = as.matrix(homes_full_bs[,1]), importance = TRUE)
bsp_df_rf <- data.frame(rep("Random Forest",39), names(homes_full_bs)[2:40],
                        as.vector(model_rf_full$importance[,1]), 2*boots_rf_sd)
names(bsp_df_rf) <- c("Method","Feature", "Significance", "Error (95% CI)")

# combine the data frames to face_wrap() for better visual comparison
boot_data <- rbind(bsp_df_ridge, bsp_df_lasso, bsp_df_en, bsp_df_rf)

# plot the data
ggplot(data = bsp_df_ridge, mapping = aes(x = Feature, y = Significance))+
  geom_col(fill = "darkgreen")+
  geom_errorbar(aes(ymin = Significance - `Error (95% CI)`,
                    ymax = Significance + `Error (95% CI)`),
                width = .5)+
  theme_classic()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        plot.title = element_text(hjust = .5))+
  labs(title = str_to_title("bootstrapped error of ridge parameter significance"))
```



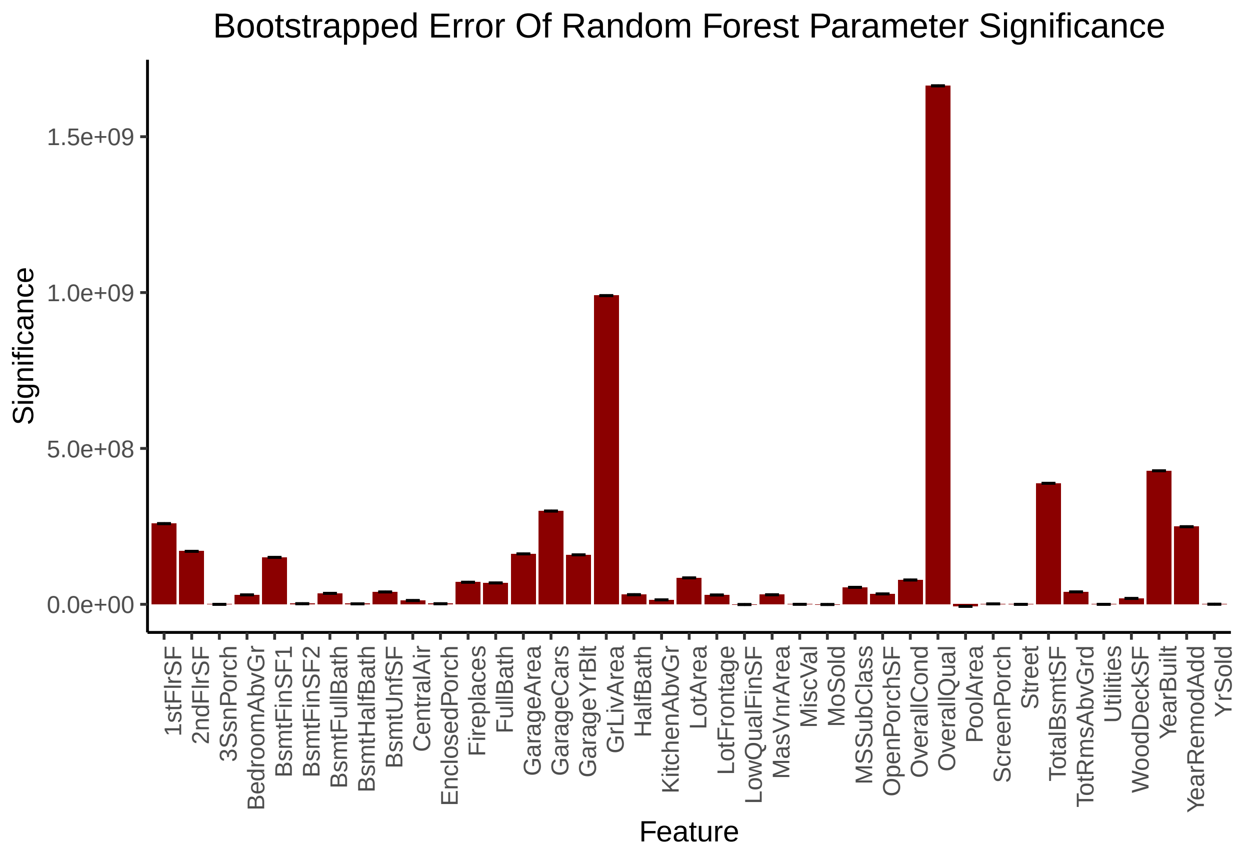
```r
ggplot(data = bsp_df_lasso, mapping = aes(x = Feature, y = Significance))+
  geom_col(fill = "darkblue")+
  geom_errorbar(aes(ymin = Significance - `Error (95% CI)`,
                    ymax = Significance + `Error (95% CI)`),
                width = .5)+
  theme_classic()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        plot.title = element_text(hjust = .5))+
  labs(title = str_to_title("bootstrapped error of lasso parameter significance"))
```



```
ggplot(data = bsp_df_en, mapping = aes(x = Feature, y = Significance))+
  geom_col(fill = "orange")+
  geom_errorbar(aes(ymin = Significance - `Error (95% CI)`,
                    ymax = Significance + `Error (95% CI)`),
                width = .5)+
  theme_classic()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        plot.title = element_text(hjust = .5))+
  labs(title = str_to_title("bootstrapped error of elastic net parameter significance"))
```

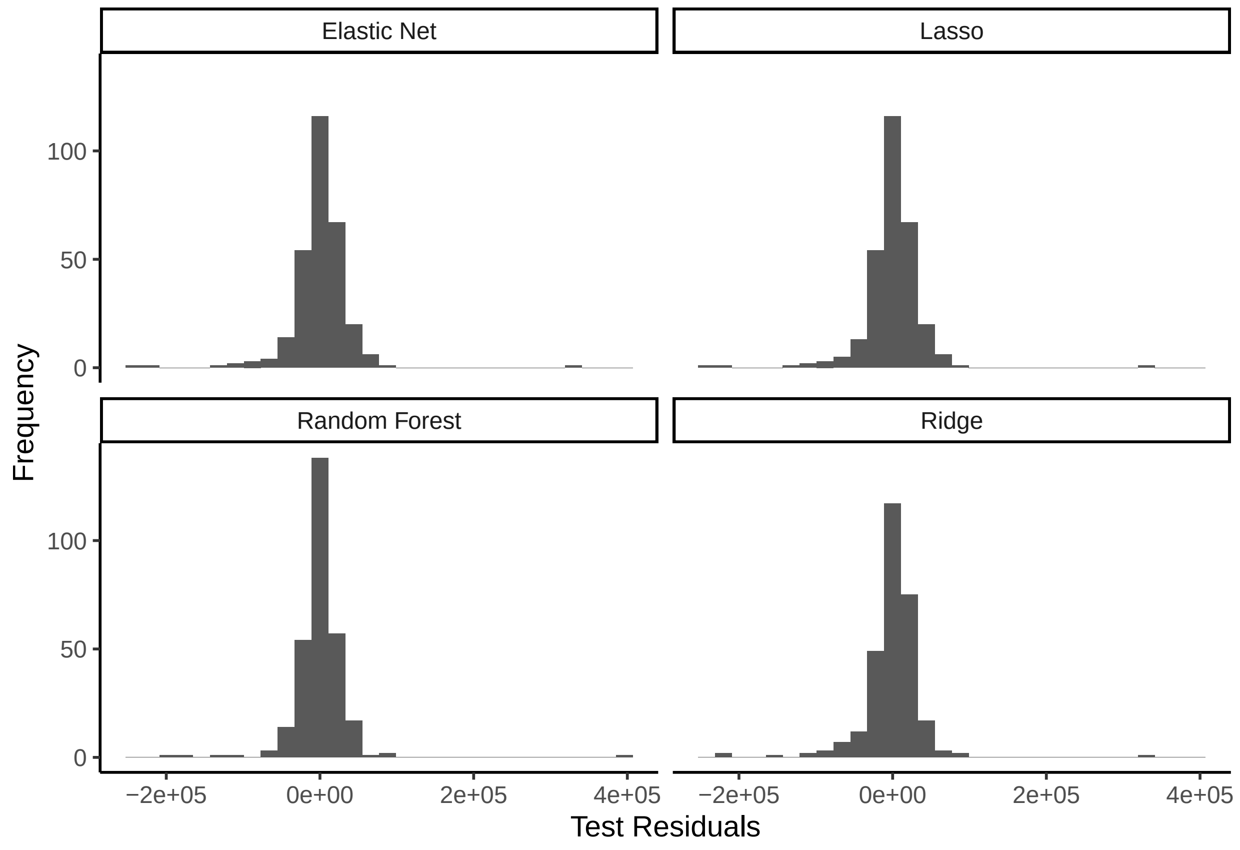


```r
ggplot(data = bsp_df_rf, mapping = aes(x = Feature, y = Significance))+
  geom_col(fill = "darkred")+
  geom_errorbar(aes(ymin = Significance - `Error (95% CI)`,
                    ymax = Significance + `Error (95% CI)`),
                width = .5)+
  theme_classic()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        plot.title = element_text(hjust = .5))+
  labs(title = str_to_title("bootstrapped error of random forest parameter significance"))
```



The following histograms show the distribution of residuals for each method and the process time for each method over 100 iterations:

```
#----residuals EDA for the 100th iteration of each model----

# combine residuals to explore with faceted ggplot
resid_ridge <- as.vector(Y_test_hat_ridge - Y_test)
resid_lasso <- as.vector(Y_test_hat_lasso - Y_test)
resid_en <- as.vector(Y_test_hat_en - Y_test)
resid_rf <- as.vector(Y_test_hat_rf - Y_test)

resid_data <- data.frame(resid_ridge, resid_lasso, resid_en, resid_rf)
names(resid_data) <- c("Ridge","Lasso", "Elastic Net", "Random Forest")

resid_data <- gather(resid_data, key = "Technique", value = "Test Residuals")

# plot
ggplot(data = resid_data, mapping = aes(x = `Test Residuals`))+
  geom_histogram()+facet_wrap(~Technique)+theme_classic()+
  ylab("Frequency")
```



```
#----list of process time by method-----------------------
paste0("Ridge Regression:"," ",abs(round(ridge_stop,2))," ","Seconds")
```

```
[1] "Ridge Regression: 12.44 Seconds"
```

```
paste0("Lasso Regression:"," ", abs(round(lasso_stop,2))," ","Seconds")
```

```
[1] "Lasso Regression: 10.21 Seconds"
```

```
paste0("Elastic Net Regression:"," ", abs(round(en_stop,2))," ","Seconds")
```

```
[1] "Elastic Net Regression: 10.51 Seconds"
```

```
paste0("Random Forest Regression:"," ", abs(round(rf_stop,2))," ","Minutes")
```

```
[1] "Random Forest Regression: 19.38 Minutes"
```

Conclusion Based on the distribution of R_squared values, it is clear the Random Forest model outperforms the other models as far as accuracy. By inspecting the residuals across histograms of the same scale it is clear that more residuals fall into the zero and neighboring bins than the other 3 models. All of the residuals are normally distributed with the exception of a few outliers in either tail.