Case Study: Code Explanation

Jason Carpenter, Nishan Madawanarachchi, Jose A. Rodilla, Kaya Tollas

10/6/2017

PREPARING THE DATA

EXPLORATORY DATA ANALYSIS

Our first step was to try to understand the data we were dealing with. To do this, we wrote code in order to generate summary statistics and plots for each variable.

```
library(tidyverse)
library(gridExtra)
# Load housing data
housing <- read.csv("housing.txt", stringsAsFactors = T)  #make sure the file data file is in the worki
```

```
# Define function for standardization of variables
zscore <- function(x) {
    return((x[!is.na(x)] - mean(x, na.rm = TRUE))/sd(x, na.rm = TRUE))
}
```

Response variable:

For the response variable, we outputed summary statistics including quartiles and information on NAs and dispersion. We also produced three graphs; a boxplot, a histogram of the original variable and a histogram of the log. A first insightful finding was that the histogram of the logarithm more closely resembled a normal distribution than that of the original variable. This already suggested that we might need to apply some kind of transformation to the response variable further down the line.

```
varName <- "SalePrice"
varName
myNumVar <- unlist(housing[varName])
countNA <- sum(is.na(myNumVar))  # Count NA
propNA <- countNA/length(myNumVar)  # Prop NA

summary(myNumVar)
variance <- var(myNumVar)
SD <- sd(myNumVar)

knitr::kable(data_frame(countNA, propNA, variance, SD))
par(mfrow = c(2, 2))
boxplot(myNumVar)
hist(housing$SalePrice, breaks = 50)
hist(log(housing$SalePrice), breaks = 50)
# hist(zscore(housing$SalePrice), breaks = 50) # zscore plots
# provide same info as previous plots
# hist(zscore(log(housing$SalePrice)),breaks = 50)
```

Explanatory variables:

For each variable we produced summary statistics, which included information on NAs and correlation with the response variable.

We also distinguished between numerical and categorical variables in order to produce insightful plots. For each numerical variable we produced three plots: a boxplot, a histogram and a scatterplot against the response variable.

For each categorical variable we produced two plots: a barplot –capturing the proportion of data for each category– and an additional plot that captured the mean value of `SalePrice` for each category in the variable –as well as the dispersion around the mean–.

This last plot was particularly helpful later on, when we had to group categories together for some categorical variables. One of the criteria employed was to group categories that appeared to have a similar effect on the response variable.

```
# coercing factor vaibles falsely identified as integer
# variables
housing$MSSubClass = as.factor(housing$MSSubClass)
housing$OverallQual = as.factor(housing$OverallQual)
housing$OverallCond = as.factor(housing$OverallCond)
```

```
for (nm in names(housing)[c(-1, -81)]) {
    # remove Id and salesprice columns
    cat("\n")
    print(nm)
    cat("\n")
    if (!is.factor(housing[nm][, 1])) {
        # Numerical Analysis
        varName <- nm
        myNumVar <- unlist(housing[varName])
        countNA <- sum(is.na(myNumVar))  # Count NA
        propNA <- countNA/length(myNumVar)  # Prop NA

        print(knitr::kable(as.matrix(summary(myNumVar))))
        variance <- var(myNumVar, na.rm = T)
        SD <- sd(myNumVar, na.rm = T)

        correlation <- cor(myNumVar, housing$SalePrice, use = "complete.obs")

        mySTDNumVar <- zscore(myNumVar)
        mySTDSalePrice <- zscore(housing$SalePrice)
        STDcorrelation <- cor(mySTDNumVar, mySTDSalePrice[!is.na(myNumVar)])  #removing sales values at
        cat("\n")
        print(knitr::kable(data_frame(varName, countNA, propNA,
            variance, SD, correlation)))
        cat("\n")
        par(mfrow = c(2, 2))
        boxplot(myNumVar)
        if (length(unique(myNumVar)) > 500) {
            # make sure only continous varibles get binned
            hist(myNumVar, breaks = 50)
            hist(log(myNumVar), breaks = 50)
        } else {
            hist(myNumVar)
        }
        plot(myNumVar, housing$SalePrice, xlab = varName)
        # plot(mySTDNumVar, mySTDSalePrice) #does not provide a
        # better insight compared to previous plot
```

```
    } else {
        # Categorical Analysis
        varName <- nm
        myCatVar <- housing[varName]

        countNA <- sum(is.na(myCatVar))  # Count NA
        propNA <- countNA/length(myCatVar[, 1])  # Prop NA, need to get the list of values

        mySummaryDF <- housing %>% group_by_(.dots = varName) %>%
            summarise(mean = mean(SalePrice), se = sd(SalePrice))  #since we are not looking at the dis

        SD_of_category_means = sd(mySummaryDF$mean)
        cat("\n")
        print(knitr::kable(data_frame(varName, countNA, propNA,
            SD_of_category_means)))
        cat("\n")
        par(mfrow = c(1, 2))

        myPlotDF <- data.frame(myCatVar)
        plot1 <- ggplot(data = myPlotDF, aes_string(x = varName)) +
            geom_histogram(aes(y = ..count../sum(..count..)),
                stat = "count") + theme(axis.text.x = element_text(angle = 90)) +
            labs(x = varName, y = "Proportion")

        myCatNames <- levels(as.factor(unlist(myCatVar)))

        plot2 <- ggplot(data = mySummaryDF, aes_string(x = varName)) +
            geom_errorbar(aes(ymin = mean - se, ymax = mean +
                se)) + geom_point(aes(y = mean)) + theme(axis.text.x = element_text(angle = 90)) +
            labs(x = varName, y = "mean(salesPrice)")

        grid.arrange(plot1, plot2, nrow = 2)

    }
}
```

HANDLING NAS

When faced with the task of handling NAs it was important to look closely into the data. We realized that in the case of most categorical variables the category labeled as "NA" could not be interpreted as data being "not available". In such cases an NA value would often refer to the absence of a particular feature.

As an example, one can see in the data dictionary that the NA category for the variable `GarageType` means that there is "No Garage". The data is thus available, and telling us something important that should be captured in the data. For all these variables we decided to change the name of the category from "NA" to "ABSENT".

Once taken care of this issue, we decided to impute the remaining NA values with the `mice` library, which takes into consideration the value of similar points to come up with a reasonable value for imputation.

After this process, we updated the values in the dataset and saved them into a new file.

```
library(VIM)
library(mice)
```

```r
data <- housing
# Select all variables except for Id and SalePrice
data.rmNA <- data[-which(names(data) == "Id" | names(data) ==
    "SalePrice")]
# Create integer factors
data.rmNA$MSSubClass = as.factor(data.rmNA$MSSubClass)
data.rmNA$OverallQual = as.factor(data.rmNA$OverallQual)
data.rmNA$OverallCond = as.factor(data.rmNA$OverallCond)

# For some variables, NA means Absence of that attribute Here
# we add a category for ABSENT for each of those variables
data.rmNA.absent <- data.rmNA
for (var in c("Alley", "BsmtQual", "BsmtCond", "BsmtExposure",
    "BsmtFinType1", "BsmtFinType2", "FireplaceQu", "GarageType",
    "GarageFinish", "GarageQual", "GarageCond", "PoolQC", "Fence",
    "MiscFeature")) {
    data.rmNA.temp <- as.character(data.rmNA[[var]])
    nas <- is.na(data.rmNA.temp)
    data.rmNA.temp[nas] <- "ABSENT"
    data.rmNA.absent[[var]] <- factor(data.rmNA.temp, levels = unique(data.rmNA.temp))
}

# Split into NA and VALs
isna <- lapply(data.rmNA.absent, function(x) any(is.na(x)))
if (any(isna == T)) {
    housing.na <- data.rmNA.absent[unlist(isna)]
    housing.vals <- data.rmNA.absent[!unlist(isna)]

    # Of the NAs, split into numeric and categorical
    iscategorical.na <- lapply(housing.na, function(x) is.factor(x))

    housing.na.numeric <- housing.na[!unlist(iscategorical.na)]
    housing.na.categorical <- housing.na[unlist(iscategorical.na)]

    # Imputing variables
    imputed_data.numeric <- mice(bind_cols(housing.na.numeric,
        housing.vals), m = 3, maxit = 5, method = "cart", seed = 99)
    imputed_data.categorical <- mice(bind_cols(housing.na.categorical,
        housing.vals), m = 3, maxit = 5, method = "polyreg",
        seed = 99)

    # Use first imputation to complete data
    complete_data.numeric <- complete(imputed_data.numeric)
    complete_data.categorical <- complete(imputed_data.categorical)

    # Inner join complete numeric and categorical data to get
    # final housing data set
    X.final <- inner_join(complete_data.numeric, complete_data.categorical)
} else {
    X.final <- data.rmNA.absent
}
# Include response (Sale Price)
y <- list(data$SalePrice)
```

```r
names(y) <- "SalePrice"
data.rmNA.final <- X.final %>% bind_cols(y)

# Write housing.final to file
write.table(data.rmNA.final, "housing_rmNA.csv", sep = ",", row.names = F)
```

ISSUES WITH PERFECT COLLINEARITY

After imputing NAs and fitting our first OLS model (which included all variables) we noticed that some of the coefficients for this fit returned an NA value. This issue arises when there is a presence of perfect collinearity amongst the variables in the model.

With the help of the EDA analysis carried out in the first section, we went through each of the variables that presented an NA coefficient and tried to determine what might be causing the linear dependence in each case.

We ended up eliminating numerical variables that were perfect linear combinations of other variables (e.g: TotalBsmtSF –Total square feet of basement area– was a linear combination of other variables that captured the basement area for different types of basement).

Some categorical variables presented NA coefficients in some but not all categories. In order to solve this problem we grouped categories together, making sure that the grouping made sense from a logical standpoint and/or the effects of these categories was similar on the response variable.

Again, we updated the data and saved it into a new file.

```r
library(glmnet)
data.rmNA.final <- read.csv("housing_rmNA.csv", stringsAsFactors = T)
data.fixMC <- data.rmNA.final
## NUMERICAL VARIABLES FIRST

## FIX WRONGLY IMPUTED GARAGEYRBLT Split GarageYrBlt into
## GarageRecentf GarageYrBlt is a weird numerical variable
## because the min == 1900 and max == 2010 Only recent year
## garages have a major impact on sale price, particularly >=
## 1990 (inclusive)
data.fixMC$GarageRecent <- ifelse(data.fixMC$GarageYrBlt >= 1990,
    1, 0)
data.fixMC <- data.fixMC[-which(names(data.fixMC) == "GarageYrBlt")]

## CATEGORICAL VARIABLES Group categories of BldgType 2 of the
## categories have the same impact on SalePrice
data.fixMC$BldgType <- as.character(data.fixMC$BldgType)
data.fixMC$BldgType[data.fixMC$BldgType == "2fmCon" | data.fixMC$BldgType ==
    "Duplex"] <- "2fmCon_Duplex"
data.fixMC$BldgType <- as.factor(data.fixMC$BldgType)

# Exterior2nd5 -- 1 row is causing collinearity from CBlock
# Join CBlock into Other category (also 1 row)
data.fixMC$Exterior2nd <- as.character(data.fixMC$Exterior2nd)
data.fixMC$Exterior2nd[data.fixMC$Exterior2nd == "CBlock" | data.fixMC$Exterior2nd ==
    "AsbShng"] <- "CBlock_AsbShng"
data.fixMC$Exterior2nd <- as.factor(data.fixMC$Exterior2nd)

# BsmtCond3 -- Group ABSENT with FA(FAIR)
data.fixMC$BsmtCond <- as.character(data.fixMC$BsmtCond)
```

```r
data.fixMC$BsmtCond[data.fixMC$BsmtCond == "ABSENT" | data.fixMC$BsmtCond ==
    "Fa"] <- "Fa_ABSENT"
data.fixMC$BsmtCond <- as.factor(data.fixMC$BsmtCond)

# BsmtFinType1 -- Group all average/below average types
data.fixMC$BsmtFinType1 <- as.character(data.fixMC$BsmtFinType1)
data.fixMC$BsmtFinType1[data.fixMC$BsmtFinType1 != "GLQ"] <- "BadLQ"
data.fixMC$BsmtFinType1 <- as.factor(data.fixMC$BsmtFinType1)

# GarageFinish -- Group unfinished and no garage
data.fixMC$GarageFinish <- as.character(data.fixMC$GarageFinish)
data.fixMC$GarageFinish[data.fixMC$GarageFinish == "Unf" | data.fixMC$GarageFinish ==
    "ABSENT"] <- "Unf_ABSENT"
data.fixMC$GarageFinish <- as.factor(data.fixMC$GarageFinish)

# GarageQual -- Fair, Poor, and ABSENT garageQuals are
# grouped
data.fixMC$GarageQual <- as.character(data.fixMC$GarageQual)
data.fixMC$GarageQual[data.fixMC$GarageQual == "Fa" | data.fixMC$GarageQual ==
    "Po" | data.fixMC$GarageQual == "ABSENT"] <- "Fa_Po_ABSENT"
data.fixMC$GarageQual <- as.factor(data.fixMC$GarageQual)

# GarageCond -- Fair, Poor, and ABSENT GarageConds are
# grouped
data.fixMC$GarageCond <- as.character(data.fixMC$GarageCond)
data.fixMC$GarageCond[data.fixMC$GarageCond == "Fa" | data.fixMC$GarageCond ==
    "Po" | data.fixMC$GarageCond == "ABSENT"] <- "Fa_Po_ABSENT"
data.fixMC$GarageCond <- as.factor(data.fixMC$GarageCond)

# Electrical -- Group mix and FuseP (poor)
data.fixMC$Electrical <- as.character(data.fixMC$Electrical)
data.fixMC$Electrical[data.fixMC$Electrical == "FuseP" | data.fixMC$Electrical ==
    "Mix"] <- "FuseP_Mix"
data.fixMC$Electrical <- as.factor(data.fixMC$Electrical)

# Final form of data after fixing multicollinearity
data.fixMC.final <- data.fixMC
remove <- c("Utilities", "PoolQC", "TotalBsmtSF", "GrLivArea",
    "GarageYrBlt")
data.fixMC.final <- data.fixMC.final[, !names(data.fixMC.final) %in%
    remove]

write.table(data.fixMC.final, "housingMCSolved.csv", sep = ",",
    row.names = F)
```

REMOVAL OF INFLUENTIAL POINTS

In order to remove influential points we compared hat values and DFFITS values against pertinent thresholds. By doing this we identified data points suspect of being high-leverage points and/or outliers.

We then applied formal tests on the residuals and deletion residuals associated to suspect points in order to determine which of them should be removed from the data.

Finally, we updated the data and saved it into a new file.

```r
library(magrittr)
housing <- read.csv("housingMCSolved.csv", stringsAsFactors = T)

# Conversion of integer variables into factors
housing$MSSubClass = as.factor(housing$MSSubClass)
housing$OverallQual = as.factor(housing$OverallQual)
housing$OverallCond = as.factor(housing$OverallCond)

# Fit linear model and extract hat values
g <- lm(SalePrice ~ ., data = housing)
ginf <- influence(g)
hat <- ginf$hat

# Check that numbers make sense
sum(hat)
length(hat)

# Define relevant parameters and threshold for heuristic
n <- nrow(housing)
p <- length(coef(g))  # Includes intercept (p+1)
th <- 2 * p/n

sum(hat > th)

# High leverage heuristic
suspectHL <- which(hat > th)

# Influential heuristic
dfts <- dffits(g)
th2 <- 2 * sqrt(p/n)
suspectINF <- which(dfts > th2)

suspect_both <- union(suspectHL, suspectINF)

s_squared <- sum(g$residuals^2)/(n - p - 1)

# Function to generate values for high_leverage hypothesis
# testing
di_star <- function(ei, hi, n, p, s_squared) {
    num <- n - p - 1
    den <- (1 - hi) * (n - p) * s_squared - ei^2
    return(ei * sqrt(num/den))
}

## High leverage formal testing
ei <- g$residuals[suspectHL]
hi <- hat[suspectHL]
di_values <- di_star(ei, hi, n, p - 1, s_squared)
# check <- (1-hat[1371])*(n-p)*s_squared - ei^2
hat[suspectHL]
sum(is.na(di_values))
sum(hat == 1)
critical <- qt(p = 0.975, df = n - p)
```

```
di_values_nonans <- di_values[which(is.na(di_values) == FALSE)]
di_values_nans <- di_values[which(is.na(di_values))]
nan_di_index <- as.integer(names(di_values_nans))
crit_dis <- di_values_nonans[abs(di_values_nonans) > critical]
crit_di_index <- as.integer(names(crit_dis))

high_leverage <- c(nan_di_index, crit_di_index)

## Influential formal testing From DFFITS heuristic, test high
## leverage
ei <- g$residuals[suspectINF]
hi <- hat[suspectINF]
di_values <- di_star(ei, hi, n, p - 1, s_squared)

hat[suspectINF]
sum(is.na(di_values))
critical <- qt(p = 0.975, df = n - p)
di_values_nonans <- di_values[which(is.na(di_values) == FALSE)]
di_values_nans <- di_values[which(is.na(di_values))]
nan_di_index <- as.integer(names(di_values_nans))
crit_dis <- di_values_nonans[abs(di_values_nonans) > critical]
crit_di_index <- as.integer(names(crit_dis))

influential_points.HL <- c(nan_di_index, crit_di_index)


# From DFFITS heuristic, test outliers
ei_star <- function(ei, hi, s_squared) {
    return(ei/sqrt((1 - hi) * s_squared))
}

ei_values <- ei_star(ei, hi, s_squared)
critical <- qt(p = 0.975, df = n - p + 1)
crit_eis <- ei_values[abs(ei_values) > critical]
crit_ei_index <- as.integer(names(crit_eis))

influential_points.OL <- crit_ei_index
influential_points.both <- union(influential_points.HL, influential_points.OL)

all_influential <- union(high_leverage, influential_points.both)

# Remove influential points and write to file
housing_noINF <- housing[-all_influential, ]
write.table(housing_noINF, "housing_noINF_MCsolved.csv", sep = ",",
    row.names = F)
```

REMOVAL OF NON-INFORMATIVE VARIABLES

After removing influential points we realized that some of the categorical variables had been rendered useless, as only points belonging to one category had remained. In other words, these variables no longer gave us any insight.

The following code was used to identify these variables and eliminate them from the dataset. Once again, the

dataset was updated at the very end and saved into a new file.

```
housing <- read.csv("housing_noINF_MCsolved.csv", stringsAsFactors = T)
remove_useless <- numeric()
high_ABSENT_Prop <- numeric()

for (p in (1:ncol(housing))) {
    if (!is.numeric(housing[[p]])) {
        if (nrow(unique(housing[p])) == 1) {
            remove_useless <- c(remove_useless, names(housing[p]))
        }
    }
    if (sum(housing[[p]] == "ABSENT")/length(housing[[p]]) >
        0.5) {
        high_ABSENT_Prop <- c(high_ABSENT_Prop, names(housing[p]))
    }
}

# Don't remove any high_ABSENT_Prop because they have meaning
# Remove remove_useless because after solving MC and removing
# INF points, they only have one level. Therefore, no
# insight.


housing_noINF_MCSolved_uselessOut <- housing[, !names(housing) %in%
    remove_useless]
write.table(housing_noINF_MCSolved_uselessOut, "housing_noINF_MCSolved_uselessOut.csv",
    sep = ",", row.names = F)
```

EXPLANATORY MODELING

We are going to start our explanatory modeling using the final data after the preprocessing.

```
# load the final dataset
housing_final_exp <- read.csv("housing_noINF_MCSolved_uselessOut.csv",
    stringsAsFactors = T)

# coercing factor variables falsely identified as integer
# variables
housing_final_exp$MSSubClass = as.factor(housing_final_exp$MSSubClass)
housing_final_exp$OverallQual = as.factor(housing_final_exp$OverallQual)
housing_final_exp$OverallCond = as.factor(housing_final_exp$OverallCond)

# Data seperation for the lasso model
X <- model.matrix(SalePrice ~ ., data = housing_final_exp)
y <- housing_final_exp$SalePrice
```

Using our preprocessed data, we split into train and test data

```
# held out test to check the predictive power of the
# explanatory model
set.seed(99)
train.data <- sample(1:nrow(X), nrow(X) * 0.8)
test.data <- (-train.data)
```

Variable Selection

Our plan for variable selection involves fitting lasso models over a grid of lambdas and using the variables which retain non-zero beta coefficients as the predictors of an Ordinary Least Squares model. From this family of ordinary least squares models, we choose the one which has the maximum adjusted R-squared. This gives us the set of variables that best explain the variance in sale price, adjusted for the number of variables we include in the model.

```
# Find adjusted R squared and number of siginificant
# variables for linear models with different variable
# selections

# each level of categorical variable will have it own column
housing_final_exp_dum <- dummy.data.frame(housing_final_exp)
adjustedR_val <- list()
grid_lambda <- 10^seq(0, 4, length = 100)

for (i in 1:100) {
    lasso_modle_exp <- glmnet(X[train.data, ], y[train.data],
        alpha = 1, lambda = grid_lambda[i])
    lasso_modle_exp$beta
    beta <- as.matrix(lasso_modle_exp$beta)[-1, ]  # get beta value from the model removing intercept

    beta_0 <- beta[beta == 0]  #variables with zero beta

    # remove variables which have beta zero
    housing_selected <- housing_final_exp_dum[, !(names(housing_final_exp_dum) %in%
        names(beta_0))]
    lin_reg <- lm(SalePrice ~ ., data = housing_selected[train.data,
        ])
    adjustedR_val[i] <- summary(lin_reg)$adj
}
```

The subsequent figure shows the plot of adjusted R-squared of the corresponding OLS model as a function of lambda for the corresponding lasso model.

```
# plot of adjustedR against different lambda
plot(x = log10(grid_lambda), y = adjustedR_val, ylim = c(0.95,
    0.96))
```

Now we want to choose the model which maximizes adjusted R-squared.

```
# get maximum lambda and the respective explanatory model
index <- which.max(adjustedR_val)
best_lambda <- grid_lambda[index]

lasso_modle_exp <- glmnet(X[train.data, ], y[train.data], alpha = 1,
    lambda = best_lambda)
beta <- as.matrix(lasso_modle_exp$beta)[-1, ]  # get beta value from the model removing intercept
beta_0 <- beta[beta == 0]  #variables with zero beta

# remove variables which have beta zero
housing_selected <- housing_final_exp_dum[, !(names(housing_final_exp_dum) %in%
    names(beta_0))]
lin_reg_exp <- lm(SalePrice ~ ., data = housing_selected[train.data,
    ])
```

Now that we have performed variable selection using lasso and created a final OLS model for explanatory purposes, we must test our assumptions.

Dealing with multicollinearity

To test for multicollinearity, we estimated the variance inflation factors (VIFs). Since these suggested multicollinearity, We thus decided to perform singular value decomposition on our design matrix, in order to build the πkj statistics. As a rule of thumb, we decided to remove variables (k) that presented at least two πkj values above 0.8. As a result, we ended up removing the variables SaleTypeCOD and SaleConditionAdjLand from our data.

```
# remove multicollinearity
fmsb::VIF(lin_reg_exp)
htrain <- housing_selected[train.data, ]
htrain_noY <- htrain %>% dplyr::select(-SalePrice)
X_train_noY <- as.matrix(htrain_noY)
svd_X <- svd(X_train_noY)
V <- as.matrix(svd_X$v)
d <- svd_X$d
PIkj <- function(j, k, V, d) {
    muk_sq <- d[k]^2
    Vjk_sq <- V[j, k]^2
    num <- Vjk_sq/muk_sq
    den <- sum((V[j, ]^2)/d^2)
    return(num/den)
}
results <- matrix(0, dim(V)[1], dim(V)[2])
for (j in 1:dim(V)[1]) {
    for (k in 1:dim(V)[2]) {
        results[k, j] <- PIkj(j, k, V, d)
    }
}

detect_mat <- matrix(0, dim(V)[1], 4)
for (i in 1:dim(results)[1]) {
    vect <- results[i, ]
    max1 <- max(vect[-i])
    max1_index <- which(vect == max1)
    max2 <- max(vect[-c(i, max1_index)])
    max2_index <- which(vect == max2)
    # detect[i] <- c(detect, c(max1, max2)) detect_ind[i] <-
    # c(detect_ind, c(max1_index, max2_index))
    detect_mat[i, 1] <- max1
    detect_mat[i, 2] <- max2
    detect_mat[i, 3] <- max1_index
    detect_mat[i, 4] <- max2_index
}
which(detect_mat[, 1] > 0.8)
which(detect_mat[, 2] > 0.8)
# detect_mat[1, ] > 0.8
rem_mc_var <- names(htrain_noY)[which(detect_mat[, 2] > 0.8)]
```

Here we remove the variables SaleTypeCOD and SaleConditionAdjLand from our data.

```
# remove the variables giving multicollinearity
housing_selected <- housing_selected[, !(names(housing_selected) %in%
    rem_mc_var)]
```

Transforming the response variable

After removing the multicollinearity, we fit another OLS linear regression model. Next we tested the linearity assumption of our residuals against our response variable SalePrice. This will indicate to us whether or not we need to transform our data. Since there is no a perfectly 0 correlation, we will subsequently test how best to transform our response variable.

```
# Linear regression after removing MC
lin_reg_exp1 <- lm(SalePrice ~ ., data = housing_selected[train.data,
    ])
# plot of standard normals and standardized residuals
plot(housing_selected[train.data, ]$SalePrice, lin_reg_exp1$residuals)
```

Next we plot the boxcox plot, which attempts to choose an appropriate transformation based on the linearity assumption, normality of residuals, and variance stabilization.

```
# Getting box cox transformation
boxcox_val <- boxcox(lin_reg_exp1)
best_trans_lambda <- boxcox_val$x[which.max(boxcox_val$y)]
```

The boxcox plot gave us a best transformation value, which was very near a square-root transformation. Here we transform our response variable using this transformation value, and fit another linear model.

```
# transfroming SalesPrice based on best translation from
# boxcox
housing_selected$SalePrice <- ((housing_selected$SalePrice)^best_trans_lambda -
    1)/best_trans_lambda
lin_reg_exp2 <- lm(SalePrice ~ ., data = housing_selected[train.data,
    ])
```

Next we formally test for the normality of residuals. They are not normal, but we decided to proceed because we applied an appropriate transformation and the adjusted r-squared of the model is quite high.

```
# test for normality of residuals
Std_residuals <- rstandard(lin_reg_exp2)  # standardized residuals
ks.test(Std_residuals, "pnorm", mean = 0, sd = 1)  # distirubtions are not normal
```

Remove influential points from training data

We further remove influential points from the training data because it should help the residuals conform to the normality assumption.

```
# changing housing_selected before removal of leverage points
# to be the dataset based on training set
housing_selected2 <- housing_selected[train.data, ]

# Fit linear model and extract hat values
g <- lm(SalePrice ~ ., data = housing_selected2)
ginf <- influence(g)
hat <- ginf$hat
```

```r
# Check that numbers make sense
sum(hat)
p <- length(coef(g))  # Includes intercept (p+1)

# Remove coe variables with NAs
rownames_tokeep <- c(rownames(summary(g)$coef), "MSZoningC (all)",
    "SalePrice")
housing_selected_rna <- housing_selected2[, names(housing_selected2) %in%
    rownames_tokeep]

# run linear regression again
g <- lm(SalePrice ~ ., data = housing_selected_rna)
ginf <- influence(g)
hat <- ginf$hat

# Check that numbers make sense
sum(hat)
p <- length(coef(g))  # Includes intercept (p+1)

# Define relevant parameters and threshold for heuristic
n <- nrow(housing_selected_rna)
th <- 2 * p/n

sum(hat > th)
# High leverage heuristic
suspectHL <- which(hat > th)

# Influential heuristic
dfts <- dffits(g)
th2 <- 2 * sqrt(p/n)
suspectINF <- which(dfts > th2)

suspect_both <- union(suspectHL, suspectINF)

s_squared <- sum(g$residuals^2)/(n - p - 1)

# Function to generate values for high_leverage hypothesis
# testing
di_star <- function(ei, hi, n, p, s_squared) {
    num <- n - p - 1
    den <- (1 - hi) * (n - p) * s_squared - ei^2
    return(ei * sqrt(num/den))
}

## High leverage formal testing
ei <- g$residuals[suspectHL]
hi <- hat[suspectHL]
di_values <- di_star(ei, hi, n, p - 1, s_squared)
# check <- (1-hat[1371])*(n-p)*s_squared - ei^2
hat[suspectHL]
sum(is.na(di_values))
sum(hat == 1)
critical <- qt(p = 0.975, df = n - p)
```

```
di_values_nonans <- di_values[which(is.na(di_values) == FALSE)]
di_values_nans <- di_values[which(is.na(di_values))]
nan_di_index <- as.integer(names(di_values_nans))
crit_dis <- di_values_nonans[abs(di_values_nonans) > critical]
crit_di_index <- as.integer(names(crit_dis))

high_leverage <- c(nan_di_index, crit_di_index)

## Influential formal testing From DFFITS heuristic, test high
## leverage
ei <- g$residuals[suspectINF]
hi <- hat[suspectINF]
di_values <- di_star(ei, hi, n, p - 1, s_squared)

hat[suspectINF]
sum(is.na(di_values))
critical <- qt(p = 0.975, df = n - p)
di_values_nonans <- di_values[which(is.na(di_values) == FALSE)]
di_values_nans <- di_values[which(is.na(di_values))]
nan_di_index <- as.integer(names(di_values_nans))
crit_dis <- di_values_nonans[abs(di_values_nonans) > critical]
crit_di_index <- as.integer(names(crit_dis))

influential_points.HL <- c(nan_di_index, crit_di_index)


# From DFFITS heuristic, test outliers
ei_star <- function(ei, hi, s_squared) {
    return(ei/sqrt((1 - hi) * s_squared))
}

ei_values <- ei_star(ei, hi, s_squared)
critical <- qt(p = 0.975, df = n - p + 1)
crit_eis <- ei_values[abs(ei_values) > critical]
crit_ei_index <- as.integer(names(crit_eis))

influential_points.OL <- crit_ei_index
influential_points.both <- union(influential_points.HL, influential_points.OL)

all_influential <- union(high_leverage, influential_points.both)

# Remove influential points and write to file
housing_selected_noINF <- housing_selected_rna[-all_influential,
    ]
write.table(housing_selected_noINF, "housing_selected_noINF.csv",
    sep = ",", row.names = F)
```

Here, we fit another linear model and test our normality assumption once again after removing the influential points from the training data.

```
# New regression model after removing influential points
lin_reg_exp3 <- lm(SalePrice ~ ., data = housing_selected_noINF)

# test for normality of residuals
```

```r
Std_residuals <- rstandard(lin_reg_exp3)  # standardized residuals
ks.test(Std_residuals, "pnorm", mean = 0, sd = 1)  # distirubtions are not normal
```

Using the new linear model, after removing influential points, we made plots to test for the linearity and normality assumptions.

```r
# plot of true sales values and residuals
plot(housing_selected_noINF$SalePrice, lin_reg_exp3$residuals)
# plot of Q-Q plot of residuals
plot(lin_reg_exp3, which = 2)
```

Now we want to check our predictive accuracy of the explanatory model on the test data.

```r
# Calculating prediction accuracy on test data
housing_test <- housing_selected[test.data, ]  #get test_data

# removing NA columns removed during influential points
# removal
housing_test <- housing_test[, names(housing_test) %in% rownames_tokeep]

# predict Salesprice on testdata
pred_price <- predict(lin_reg_exp3, newdata = housing_test)

# convert predicted price to orginal scale
pred_price_con <- (pred_price * best_trans_lambda + 1)^(1/best_trans_lambda)

# plot predicted prices and original sales prices
plot(x = 1:length(pred_price), y = housing_final_exp[test.data,
    ]$SalePrice, type = "l", col = "red", xlab = "")
points(x = 1:length(pred_price), y = pred_price_con, col = "green")
```

Building an explanatory model using the full data set

Here we build our near-complete explanatory model using the full set of data (both training and test). We scaled the data to ensure equivalently scaled coefficient estimates. Interpretation of these coefficients is now in terms of standard deviations, rather than raw values. Before fitting the final explanatory model, we performed final variable selection, removing variables which had newly introduced exact collinearity after removal of influential points.

```r
# sales prices are already tranformed with boxcox Building
# the final model with standardized data
mean_price <- mean(housing_selected$SalePrice)
sd_price <- sd(housing_selected$SalePrice)

pp = preProcess(housing_selected, method = c("center", "scale"))
housing_selected_scaled <- predict(pp, housing_selected)

# removing NA columns removed during influential points
# removal
housing_selected_scaled <- housing_selected_scaled[, names(housing_selected_scaled) %in%
    rownames_tokeep]

# linear regression based on scaled values
lin_reg_exp4 <- lm(SalePrice ~ ., data = housing_selected_scaled)
```

The last step before fitting our final explanatory model was to remove the remaining influential points.

```
# Remove leverage points in the dataset removing leverage
# points

housing_selected_scaled2 <- housing_selected_scaled

# Fit linear model and extract hat values
g <- lm(SalePrice ~ ., data = housing_selected_scaled2)
ginf <- influence(g)
hat <- ginf$hat

# Check that numbers make sense
sum(hat)
p <- length(coef(g))  # Includes intercept (p+1)

# Remove coe variables with NAs
rownames_tokeep <- c(rownames(summary(g)$coef), "MSZoningC (all)",
    "SalePrice")
housing_selected_scaled_rna <- housing_selected_scaled2[, names(housing_selected_scaled2) %in%
    rownames_tokeep]

# run linear regression again
g <- lm(SalePrice ~ ., data = housing_selected_scaled_rna)
ginf <- influence(g)
hat <- ginf$hat

# Check that numbers make sense
sum(hat)
p <- length(coef(g))  # Includes intercept (p+1)

# Define relevant parameters and threshold for heuristic
n <- nrow(housing_selected_scaled_rna)
th <- 2 * p/n

sum(hat > th)
# High leverage heuristic
suspectHL <- which(hat > th)

# Influential heuristic
dfts <- dffits(g)
th2 <- 2 * sqrt(p/n)
suspectINF <- which(dfts > th2)

suspect_both <- union(suspectHL, suspectINF)

s_squared <- sum(g$residuals^2)/(n - p - 1)

# Function to generate values for high_leverage hypothesis
# testing
di_star <- function(ei, hi, n, p, s_squared) {
    num <- n - p - 1
    den <- (1 - hi) * (n - p) * s_squared - ei^2
    return(ei * sqrt(num/den))
```

```
}

## High leverage formal testing
ei <- g$residuals[suspectHL]
hi <- hat[suspectHL]
di_values <- di_star(ei, hi, n, p - 1, s_squared)
# check <- (1-hat[1371])*(n-p)*s_squared - ei^2
hat[suspectHL]
sum(is.na(di_values))
sum(hat == 1)
critical <- qt(p = 0.975, df = n - p)
di_values_nonans <- di_values[which(is.na(di_values) == FALSE)]
di_values_nans <- di_values[which(is.na(di_values))]
nan_di_index <- as.integer(names(di_values_nans))
crit_dis <- di_values_nonans[abs(di_values_nonans) > critical]
crit_di_index <- as.integer(names(crit_dis))

high_leverage <- c(nan_di_index, crit_di_index)

## Influential formal testing From DFFITS heuristic, test high
## leverage
ei <- g$residuals[suspectINF]
hi <- hat[suspectINF]
di_values <- di_star(ei, hi, n, p - 1, s_squared)

hat[suspectINF]
sum(is.na(di_values))
critical <- qt(p = 0.975, df = n - p)
di_values_nonans <- di_values[which(is.na(di_values) == FALSE)]
di_values_nans <- di_values[which(is.na(di_values))]
nan_di_index <- as.integer(names(di_values_nans))
crit_dis <- di_values_nonans[abs(di_values_nonans) > critical]
crit_di_index <- as.integer(names(crit_dis))

influential_points.HL <- c(nan_di_index, crit_di_index)


# From DFFITS heuristic, test outliers
ei_star <- function(ei, hi, s_squared) {
    return(ei/sqrt((1 - hi) * s_squared))
}

ei_values <- ei_star(ei, hi, s_squared)
critical <- qt(p = 0.975, df = n - p + 1)
crit_eis <- ei_values[abs(ei_values) > critical]
crit_ei_index <- as.integer(names(crit_eis))

influential_points.OL <- crit_ei_index
influential_points.both <- union(influential_points.HL, influential_points.OL)

all_influential <- union(high_leverage, influential_points.both)

# Remove influential points and write to file
```

```
housing_selected_scaled_noINF <- housing_selected_scaled_rna[-all_influential,
    ]
```

Final explanatory model

Here we fit our final model.

```
# new linear regression after removing leverage points
lin_reg_exp5 <- lm(SalePrice ~ ., data = housing_selected_scaled_noINF)
```

Final tests for linearity and normality of residuals as well as a test for constant variance.

```
# Tests on the final model

# test for normality of residuals
Std_residuals <- rstandard(lin_reg_exp5)  # standardized residuals
ks.test(Std_residuals, "pnorm", mean = 0, sd = 1)  # distirubtions are not normal

# plot of true sales values and residuals
plot(housing_selected_scaled_noINF$SalePrice, lin_reg_exp5$residuals)
# plot of Q-Q plot of residuals
plot(lin_reg_exp5, which = 2)

ncvTest(lin_reg_exp5)
```

Morty's house

After building the final model (`lin_reg_exp5`) we are testing on the Morty data provided. We choose the maximum value that morty can sell his house for by using the upper bound of the prediction interval.

```
morty <- read.csv("mortyTransformed.csv", stringsAsFactors = T)

# coercing factor vaibles falsely identified as integer
# variables
morty$MSSubClass = as.factor(morty$MSSubClass)
morty$OverallQual = as.factor(morty$OverallQual)
morty$OverallCond = as.factor(morty$OverallCond)

# Generate morty with dummy variables
temp_df <- rbind(housing_final_exp, morty)
temp_df <- dummy.data.frame(temp_df)  # converted with dummy variables
temp_df <- temp_df[1402, ]  #final entry is morty
temp_df <- temp_df[, !(names(temp_df) %in% names(beta_0))]  #removed columns from lasso
temp_df <- predict(pp, temp_df)
temp_df <- temp_df[, names(temp_df) %in% rownames_tokeep]  #removed columns with high influential point

# predict housing value for morty
pred_morty <- predict(lin_reg_exp4, newdata = temp_df, interval = "predict")

# convert prdicted morty value to original scale
pred_morty_cov <- pred_morty * sd_price + mean_price
pred_morty_cov <- (pred_morty_cov * best_trans_lambda + 1)^(1/best_trans_lambda)
```

```r
# Maximum price morty can sell the house for,
pred_morty_cov[3]

# Variables that will increase the value of housing,
changes_tohouse <- as.data.frame(summary(lin_reg_exp4)$coef)
changes_tohouse$variablenames <- rownames(changes_tohouse)
changes_tohouse <- arrange(changes_tohouse, desc(Estimate))
changes_tohouse <- changes_tohouse[1:3, c(1, 5)]
```

PREDICTIVE MODELING

To come up with the best predictive model we considered OLS, Ridge regression, Lasso regression and Elastic Net. After separating a hold-out set –for later use in model comparison– we came up with the best model for each of these methods using the remaining training data.

For OLS we first ran a Lasso model and chose the lambda that minimized the mean cross validated error. We then fed into OLS the variables that presented non-zero coefficients after running Lasso.

For both Lasso and Ridge regression we considered a grid of lambdas with values ranging from 0.01 to 10,000,000,000. We then performed 10-fold cross validation to come up with the lambda value that produced the best predictive model in each of these cases.

In the case Elastic Net, in addition to the grid of lambdas, we had to consider a grid of alphas with values ranging from 0 to 1, with 100 evenly spaced steps in between. In order to come up with the best alpha-lambda combination we performed 10-fold cross validation across both grids in paralel.

For each method we wrote a separate function that returns the best possible model for each respective method given the training data. These functions are then called by the main function –`testing`– which assures that the extraction of each best model is performed under the same seed, thus selecting the same training and hold-out partitions in each case.

After comparing the performance of all the best models (the best for each method) on the hold-out data we selected the one with the lowest MSPE as our best overall model. We then saved our final model for future reference.

```r
library(glmnet)
library(ISLR)
library(magrittr)
library(caret)

rm(list = ls())

# Fit an OLS regression model on the training data
ols <- function(train_data, seed = 99) {
    set.seed(seed)
    # Get the most relevant variables of X to include in OLS by
    # first running lasso
    X <- lasso(train_data)[[4]]
    y <- list(train_data$SalePrice)
    names(y) <- "SalePrice"
    DF <- as.data.frame(X) %>% bind_cols(y)
    # Fit OLS model on SalePrice ~ the X variables chosen by
    # lasso
    ols <- lm(SalePrice ~ ., data = DF)
    return(ols)
```

```
}

# Fit a ridge regression model using 10-fold cross-validation
# to find the best lambda
ridge <- function(train_data, seed = 99) {
    X <- model.matrix(SalePrice ~ ., data = train_data)
    y <- train_data$SalePrice
    grid.lambda <- 10^seq(10, -2, length = 100)
    set.seed(seed)
    # 10-fold cross validation of ridge regression to choose the
    # best lambda over grid.lambda
    cv.out <- cv.glmnet(X, y, alpha = 0, lambda = grid.lambda)
    best.lambda <- cv.out$lambda.min
    final_model <- glmnet(X, y, alpha = 0, lambda = best.lambda)
    return(list(best.lambda, final_model))
}

# Fit a lasso regression model using 10-fold cross-validation
# to find the best lambda
lasso <- function(train_data, seed = 99) {
    set.seed(seed)
    X <- model.matrix(SalePrice ~ ., data = train_data)
    y <- train_data$SalePrice
    grid.lambda <- 10^seq(10, -2, length = 100)
    cv.out <- cv.glmnet(X, y, alpha = 1, lambda = grid.lambda)
    best.lambda <- cv.out$lambda.min
    final_model <- glmnet(X, y, alpha = 1, lambda = best.lambda)
    coefs <- ifelse(coef(final_model) == 0, 0, 1)[-2]
    new_X <- X[, seq(ncol(X)) * coefs]
    return(list(best.lambda, final_model, coef(final_model),
        new_X))
}


# Fit an elastic net regression model We create a grid fo
# lambdas and alphas To run 10-fold cross-validation over
# alpha and lambda using cva.glmnet in parallel We return the
# full cva model
elastic_net <- function(train_data, seed = 99) {
    X <- model.matrix(SalePrice ~ ., data = train_data)
    y <- train_data$SalePrice
    set.seed(seed)
    grid.lambda <- 10^seq(10, -2, length = 100)
    grid.alpha <- seq(0, 1, length = 100)
    clust <- parallel::makePSOCKcluster(4)  # Parallel cluster
    # Run cva.glmnet in parallel
    cva.out <- glmnetUtils::cva.glmnet(X, y, lambda = grid.lambda,
        alpha = grid.alpha, outerParallel = clust, checkInnerParallel = TRUE)
    parallel::stopCluster(clust)  # Close clusters
    return(cva.out)
}

# Function to split data into training and test sets (80/20
```

```
# split)
split_data <- function(data, seed = 99) {
    set.seed(seed)
    train <- sample(1:nrow(data), nrow(data) * 0.8)
    test <- (-train)
    return(list(data[train, ], data[test, ]))
}

# Tests the chosen model on the provided data Crucially using
# the same seed for all models, so that train/holdout data is
# the same across models Returns mspe, best.lambda,
# best.alpha, and model fit on all data for the chosen model
testing <- function(data, model.name, seed = 99) {
    set.seed(seed)
    # Split the housing data
    mySplitData <- split_data(data, seed)
    train_data <- mySplitData[[1]]
    holdout_data <- mySplitData[[2]]

    # Using all data, create X and y variables
    X.all <- model.matrix(SalePrice ~ ., data = data)
    y.all <- data$SalePrice

    # Using just holdout data, create X and y variables for
    # assessing model performance using MSPE
    X.ho <- model.matrix(SalePrice ~ ., data = holdout_data)
    y.ho <- holdout_data$SalePrice

    # OLS best mspe and model
    if (model.name == "ols") {
        O <- ols(train_data, seed)
        ols.pred <- predict(O, newdata = as.data.frame(X.ho),
            se.fit = T)
        mspe <- mean((ols.pred$fit - y.ho)^2)
        best.lambda <- NA
        best.alpha <- NA
        best.model <- O
    }

    # lasso best mspe, lambda, alpha, and model
    if (model.name == "lasso") {
        L <- lasso(train_data, seed)
        lambda <- L[[1]]
        model <- L[[2]]
        lasso.pred <- predict(model, s = lambda, newx = X.ho)
        mspe <- mean((lasso.pred - y.ho)^2)
        best.lambda <- lambda
        best.alpha <- 1
        best.model <- glmnet(X.all, y.all, alpha = best.alpha,
            lambda = best.lambda)

    }
```

```
    # ridge best mspe, lambda, alpha, and model
    if (model.name == "ridge") {
        R <- ridge(train_data, seed)
        lambda <- R[[1]]
        model <- R[[2]]
        # Predict on model using best lambda and model
        ridge.pred <- predict(model, s = lambda, newx = X.ho)
        mspe <- mean((ridge.pred - y.ho)^2)
        best.lambda <- lambda
        best.alpha <- 0
        best.model <- glmnet(X.all, y.all, alpha = best.alpha,
            lambda = best.lambda)
    }

    # elastic net best mspe, lambda, alpha, and model
    if (model.name == "elastic_net") {
        EN <- elastic_net(train_data, seed)
        alphas <- EN$alpha
        models <- EN$modlist
        en.pred <- list()
        mspe.lst <- list()
        # For each elastic net best.lambda/alpha combination model
        for (m in 1:length(models)) {
            # Predict on holdout data and get mspe
            en.pred[[m]] <- predict(models[[m]]$glmnet.fit, alpha = alphas[m],
                s = models[[m]]$lambda.min, newx = X.ho)
            mspe.lst[[m]] <- mean((en.pred[[m]] - y.ho)^2)
        }
        # Choose lambda/alpha combination which has min(mspe) over
        # all pairings
        mspe.lst <- unlist(mspe.lst)
        mspe.lst.index <- which(mspe.lst == min(mspe.lst))
        en.best <- list(min(mspe.lst), models[[mspe.lst.index]]$lambda.min,
            alphas[[mspe.lst.index]])
        mspe <- en.best[[1]]
        best.lambda <- en.best[[2]]
        best.alpha <- en.best[[3]]
        best.model <- glmnet(X.all, y.all, alpha = best.alpha,
            lambda = best.lambda)

    }
    return(list(mspe, best.lambda, best.alpha, best.model))
}

# Read in data
housing <- read.csv("housing_noINF_MCSolved_uselessOut.csv",
    stringsAsFactors = T)
seed <- 99

# Get best models of each type
best.ols <- testing(data = housing, model = "ols", seed)
best.lasso <- testing(data = housing, model = "lasso", seed)
best.ridge <- testing(data = housing, model = "ridge", seed)
```

```
best.en <- testing(data = housing, model = "elastic_net", seed)

# Compare mspe for all models and choose the min(mspe)
mspe.compare <- c(best.ols[[1]], best.lasso[[1]], best.ridge[[1]],
    best.en[[1]])
best.mspe <- min(mspe.compare)
which.model.index <- which(mspe.compare == best.mspe)

# Return name of and relevant details (lambda, alpha, model)
# for the best model
if (which.model.index == 1) {
    best.model <- list("ols", best.ols)
} else if (which.model.index == 2) {
    best.model <- list("lasso", best.lasso)
} else if (which.model.index == 3) {
    best.model <- list("ridge", best.ridge)
} else if (which.model.index == 4) {
    best.model <- list("elastic_net", best.en)
} else {
    best.model <- "ERROR!"
}
best.model[[1]]
save(best.model, file = "bestPredModel.RData")
```

In order to use the best model to predict new data, we developed a pipeline to transform the new data, similar to how we transformed the training data.

```
library(tidyverse)
library(ggplot2)
library(VIM)
library(mice)
rm(list = ls())
cat("\f")

fixNA <- function(data) {
    data.rmNA <- data[-which(names(data) == "Id" | names(data) ==
        "SalePrice")]
    # Create integer factors
    data.rmNA$MSSubClass = as.factor(data.rmNA$MSSubClass)
    data.rmNA$OverallQual = as.factor(data.rmNA$OverallQual)
    data.rmNA$OverallCond = as.factor(data.rmNA$OverallCond)

    # For some variables, NA means Absence of that attribute Here
    # we add a category for ABSENT for each of those variables
    data.rmNA.absent <- data.rmNA
    for (var in c("Alley", "BsmtQual", "BsmtCond", "BsmtExposure",
        "BsmtFinType1", "BsmtFinType2", "FireplaceQu", "GarageType",
        "GarageFinish", "GarageQual", "GarageCond", "PoolQC",
        "Fence", "MiscFeature")) {
        data.rmNA.temp <- as.character(data.rmNA[[var]])
        nas <- is.na(data.rmNA.temp)
        data.rmNA.temp[nas] <- "ABSENT"
        data.rmNA.absent[[var]] <- factor(data.rmNA.temp, levels = unique(data.rmNA.temp))
    }
```

```
    # Split into NA and VALs
    isna <- lapply(data.rmNA.absent, function(x) any(is.na(x)))
    if (any(isna == T)) {
        housing.na <- data.rmNA.absent[unlist(isna)]
        housing.vals <- data.rmNA.absent[!unlist(isna)]

        # Of the NAs, split into numeric and categorical
        iscategorical.na <- lapply(housing.na, function(x) is.factor(x))

        housing.na.numeric <- housing.na[!unlist(iscategorical.na)]
        housing.na.categorical <- housing.na[unlist(iscategorical.na)]

        # Imputing variables
        imputed_data.numeric <- mice(bind_cols(housing.na.numeric,
            housing.vals), m = 3, maxit = 5, method = "cart",
            seed = 99)
        imputed_data.categorical <- mice(bind_cols(housing.na.categorical,
            housing.vals), m = 3, maxit = 5, method = "polyreg",
            seed = 99)

        # Use first imputation to complete data
        complete_data.numeric <- complete(imputed_data.numeric)
        complete_data.categorical <- complete(imputed_data.categorical)

        # Inner join complete numeric and categorical data to get
        # final housing data set
        X.final <- inner_join(complete_data.numeric, complete_data.categorical)
    } else {
        X.final <- data.rmNA.absent
    }
    # Include response (Sale Price)
    y <- list(data$SalePrice)
    names(y) <- "SalePrice"
    data.rmNA.final <- X.final %>% bind_cols(y)
    return(data.rmNA.final)
}

fixMC <- function(data) {
    data.fixMC <- data
    ## NUMERICAL VARIABLES FIRST

    ## FIX WRONGLY IMPUTED GARAGEYRBLT Split GarageYrBlt into
    ## GarageRecentf GarageYrBlt is a weird numerical variable
    ## because the min == 1900 and max == 2010 Only recent year
    ## garages have a major impact on sale price, particularly >=
    ## 1990 (inclusive)
    data.fixMC$GarageRecent <- ifelse(data.fixMC$GarageYrBlt >=
        1990, 1, 0)
    data.fixMC <- data.fixMC[-which(names(data.fixMC) == "GarageYrBlt")]

    ## CATEGORICAL VARIABLES Group categories of BldgType 2 of the
    ## categories have the same impact on SalePrice
    data.fixMC$BldgType <- as.character(data.fixMC$BldgType)
```

```
data.fixMC$BldgType[data.fixMC$BldgType == "2fmCon" | data.fixMC$BldgType ==
    "Duplex"] <- "2fmCon_Duplex"
data.fixMC$BldgType <- as.factor(data.fixMC$BldgType)

# Exterior2nd5 -- 1 row is causing collinearity from CBlock
# Join CBlock into Other category (also 1 row)
data.fixMC$Exterior2nd <- as.character(data.fixMC$Exterior2nd)
data.fixMC$Exterior2nd[data.fixMC$Exterior2nd == "CBlock" |
    data.fixMC$Exterior2nd == "AsbShng"] <- "CBlock_AsbShng"
data.fixMC$Exterior2nd <- as.factor(data.fixMC$Exterior2nd)

# BsmtCond3 -- Group ABSENT with FA(FAIR)
data.fixMC$BsmtCond <- as.character(data.fixMC$BsmtCond)
data.fixMC$BsmtCond[data.fixMC$BsmtCond == "ABSENT" | data.fixMC$BsmtCond ==
    "Fa"] <- "Fa_ABSENT"
data.fixMC$BsmtCond <- as.factor(data.fixMC$BsmtCond)

# BsmtFinType1 -- Group all average/below average types
data.fixMC$BsmtFinType1 <- as.character(data.fixMC$BsmtFinType1)
data.fixMC$BsmtFinType1[data.fixMC$BsmtFinType1 != "GLQ"] <- "BadLQ"
data.fixMC$BsmtFinType1 <- as.factor(data.fixMC$BsmtFinType1)

# GarageFinish -- Group unfinished and no garage
data.fixMC$GarageFinish <- as.character(data.fixMC$GarageFinish)
data.fixMC$GarageFinish[data.fixMC$GarageFinish == "Unf" |
    data.fixMC$GarageFinish == "ABSENT"] <- "Unf_ABSENT"
data.fixMC$GarageFinish <- as.factor(data.fixMC$GarageFinish)

# GarageQual -- Fair, Poor, and ABSENT garageQuals are
# grouped
data.fixMC$GarageQual <- as.character(data.fixMC$GarageQual)
data.fixMC$GarageQual[data.fixMC$GarageQual == "Fa" | data.fixMC$GarageQual ==
    "Po" | data.fixMC$GarageQual == "ABSENT"] <- "Fa_Po_ABSENT"
data.fixMC$GarageQual <- as.factor(data.fixMC$GarageQual)

# GarageCond -- Fair, Poor, and ABSENT GarageConds are
# grouped
data.fixMC$GarageCond <- as.character(data.fixMC$GarageCond)
data.fixMC$GarageCond[data.fixMC$GarageCond == "Fa" | data.fixMC$GarageCond ==
    "Po" | data.fixMC$GarageCond == "ABSENT"] <- "Fa_Po_ABSENT"
data.fixMC$GarageCond <- as.factor(data.fixMC$GarageCond)

# Electrical -- Group mix and FuseP (poor)
data.fixMC$Electrical <- as.character(data.fixMC$Electrical)
data.fixMC$Electrical[data.fixMC$Electrical == "FuseP" |
    data.fixMC$Electrical == "Mix"] <- "FuseP_Mix"
data.fixMC$Electrical <- as.factor(data.fixMC$Electrical)

# Final form of data after fixing multicollinearity
data.fixMC.final <- data.fixMC
remove <- c("Utilities", "PoolQC", "TotalBsmtSF", "GrLivArea",
    "GarageYrBlt")
data.fixMC.final <- data.fixMC.final[, !names(data.fixMC.final) %in%
```

```
        remove]
    return(data.fixMC.final)
}
```

Then we created a prediction function which accepts the file path of the testing dataset and returns the predicted SalePrice

```
library(glmnet)
predNewData <- function(newData_filePath) {
    # Load best model from finalPredModels
    load("bestPredModel.RData")
    model.name <- best.model[[1]]
    model <- best.model[[2]][[4]]
    lambda <- best.model[[2]][[2]]

    # Load housing data
    housing <- read.csv("housing_noINF_MCSolved_uselessOut.csv",
        stringsAsFactors = T)
    housing$MSSubClass = as.factor(housing$MSSubClass)
    housing$OverallQual = as.factor(housing$OverallQual)
    housing$OverallCond = as.factor(housing$OverallCond)

    # Load new data
    newData <- read.csv(newData_filePath, stringsAsFactors = T)
    # Transform new data
    newData.transform <- fixMC(fixNA(newData))
    newData.transform$MSSubClass = as.factor(newData.transform$MSSubClass)
    newData.transform$OverallQual = as.factor(newData.transform$OverallQual)
    newData.transform$OverallCond = as.factor(newData.transform$OverallCond)

    # Convert data to format used for prediction
    newData.plus.housing <- housing %>% rbind(newData.transform)
    newData.mm <- model.matrix(SalePrice ~ ., data = newData.plus.housing)
    newData <- newData.mm[seq(nrow(newData.mm) - nrow(newData.transform) +
        1, nrow(newData.mm)), , drop = F]
    newData.df <- data.frame(newData)

    # Predict new data used best prediction model
    newData.pred <- predict(model, s = lambda, newx = as.matrix(newData.df))
    return(newData.pred)
}

# Enter the filepath to the new data that you wish to predict
# into the predNewData() function Here is an example, using
# morty.txt
newData.pred <- predNewData("morty.txt")
```