STT 481 Midterm Project

Jess VanElls

November 7, 2018

Data Pre-Processing

Data Preparation

```
train_orig <- read.csv("train.csv", na.strings="placeholder")  # some of the categorical variabl
es have value "NA" but it doesn't mean null
test_orig <- read.csv("test.csv", na.strings="placeholder")

## Store copies to edit
train <- train_orig
test <- test_orig
```

Dealing with NA’s

Because there were also strings that were “NA” as part of some scales, I noted which columns shouldn’t contain the string “NA”“, and I change those strings to a true `NA` .

```
## Store all columns that can have "NA" as a valid entry
na_names = c("Alley", "BsmtQual", "BsmtCond", "BsmtExposure", "BsmtFinType1", "BsmtFinType2", "F
ireplaceQu", "GarageType", "GarageQual", "GarageCond", "PoolQC", "Fence", "MiscFeature", "MasVnr
Type")

## Replace "NA" strings with true NA in training data
for (j in 1:ncol(train)) {
  if (sum(colnames(train)[j]==na_names)==0) {  # if the column shouldn't contain "NA"...
    for (i in 1:nrow(train)) {
      if (train[i,j]=="NA") {
        train[i,j] <- NA  # if the column shouldn't contain NA but the cell is "NA", then give i
t a null
      }
    }
  }
}
for (j in 1:ncol(test)) {
  if (sum(colnames(test)[j]==na_names)==0) {  # if the column shouldn't contain "NA"...
    for (i in 1:nrow(test)) {
      if (test[i,j]=="NA") {
        test[i,j] <- NA  # if the column shouldn't contain NA but the cell is "NA", then give it
 a null
      }
    }
  }
}
```

Checking and Changing Data Types

I will be converting scales to factors (e.g., quality) because they describe a condition, not a quantity. There are mixed opinions on how these should be handled, but I am chosing to use the “nominal categorical” method. Years will be treated as integers; months will be treated as factors.

```
old_types_tr <- as.vector(sapply(X=train_orig, FUN=class))  # store original types for reference

train$MSSubClass <- as.factor(train$MSSubClass)

## Scales
train$OverallQual <- as.factor(train$OverallQual)
train$OverallCond <- as.factor(train$OverallCond)

## Should be numeric...
train$LotFrontage <- as.integer(train$LotFrontage)
train$MasVnrArea <- as.integer(train$MasVnrArea)

## Dates: years as integers, months as factors
train$GarageYrBlt <- as.integer(train$GarageYrBlt)
train$MoSold <- as.factor(train$MoSold)

new_types_tr <- as.vector(sapply(X=train, FUN=class))  # store new data types
# cbind(colnames(train), old_types_tr, new_types_tr)
```

```
old_types_te <- as.vector(sapply(X=test_orig, FUN=class))  # store original types for reference
colnames(test)[old_types_te!=new_types_tr[-81]]  # which columns in the test set aren't right
```

```
 [1] "MSSubClass"   "LotFrontage"  "OverallQual"  "OverallCond"
 [5] "MasVnrArea"   "BsmtFinSF1"   "BsmtFinSF2"   "BsmtUnfSF"
 [9] "TotalBsmtSF"  "BsmtFullBath" "BsmtHalfBath" "GarageYrBlt"
[13] "GarageCars"   "GarageArea"   "MoSold"
```

```
## Scales
test$MSSubClass <- as.factor(test$MSSubClass)
test$OverallQual <- as.factor(test$OverallQual)
test$OverallCond <- as.factor(test$OverallCond)

## Should be numeric...
test$LotFrontage <- as.integer(test$LotFrontage)
test$MasVnrArea <- as.integer(test$MasVnrArea)
test$BsmtFinSF1 <- as.integer(test$BsmtFinSF1)
test$BsmtFinSF2 <- as.integer(test$BsmtFinSF2)
test$BsmtUnfSF <- as.integer(test$BsmtUnfSF)
test$TotalBsmtSF <- as.integer(test$TotalBsmtSF)
test$BsmtFullBath <- as.integer(test$BsmtFullBath)
test$BsmtHalfBath <- as.integer(test$BsmtHalfBath)
test$GarageCars <- as.integer(test$GarageCars)
test$GarageArea <- as.integer(test$GarageArea)

## Dates: years as integers, months as factors
test$GarageYrBlt <- as.integer(test$GarageYrBlt)
test$MoSold <- as.factor(test$MoSold)

new_types_te <- as.vector(sapply(X=train, FUN=class))  # store new data types
# cbind(colnames(test, old_types_te, new_types_te)
```

NA Revisited

Change the string “NA” to “N/A” for variables that are allowed to have “NA” as a value (e.g., Alley). I don’t change the “NA” strings to `NA` here (I did it earlier) because otherwise the change of class insert interpolated values instead of `NA` s.

```
## Store all columns that can have "NA" as a valid entry
na_names = c("Alley", "BsmtQual", "BsmtCond", "BsmtExposure", "BsmtFinType1", "BsmtFinType2", "F
ireplaceQu", "GarageType", "GarageQual", "GarageCond", "PoolQC", "Fence", "MiscFeature", "MasVnr
Type")

## Re-level "NA" columns
for (j in 1:ncol(train)) {
  if (sum(colnames(train)[j]==na_names)!=0) {  # if the column can contain "NA" as a string in t
he training data...
    levels(train[,j])[levels(train[,j])=="NA"] <- "N/A"
  }
}
for (j in 1:ncol(test)) {
  if (sum(colnames(test)[j]==na_names)!=0) {  # if the column can contain "NA" as a string in th
e test data...
    levels(test[,j])[levels(test[,j])=="NA"] <- "N/A"
  }
}
```

Remove NA Columns

Remove columns from both sets which have too many NAs in the training set, and then remove rows from the training set with NAs left.

```
## Remove NA columns
ct_na_traincol <- rep(0, length=ncol(train))
for (j in 1:ncol(train)) {
  ct_na_traincol[j] <- sum(is.na(train[,j]))
}
train <- train[-c(1:80)[ct_na_traincol>50]]
test <- test[-c(1:80)[ct_na_traincol>50]]  # if I'm not predicting on it, there is no point in s
toring it in the test data

## Remove NA rows
na_row_train <- c()
for (i in 1:nrow(train)) {
  if (sum(is.na(train[i,]))>0) {
    na_row_train <- c(na_row_train, i)
  }
}
train <- train[-na_row_train,]
```

Interpolate NA values in the test data

```
## Note which columns need to have NAs interpolated
ct_na_testcol <- rep(0, length=ncol(test))
for (j in 1:ncol(test)) {
  ct_na_testcol[j] <- sum(is.na(test[,j]))
}
ct_na_testcol
```

```
 [1]  0  0  4  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  1
[24]  1  0 15  0  0  0  0  0  0  0  1  0  1  1  1  0  0  0  0  0  0  0  0
[47]  2  2  0  0  0  0  1  0  2  0  0  0  1  1  0  0  0  0  0  0  0  0  0
[70]  0  0  0  0  0  0  1  0
```

```
na_testcol_boo <- ifelse(ct_na_testcol!=0, T, F)
colnames(test)[na_testcol_boo]
```

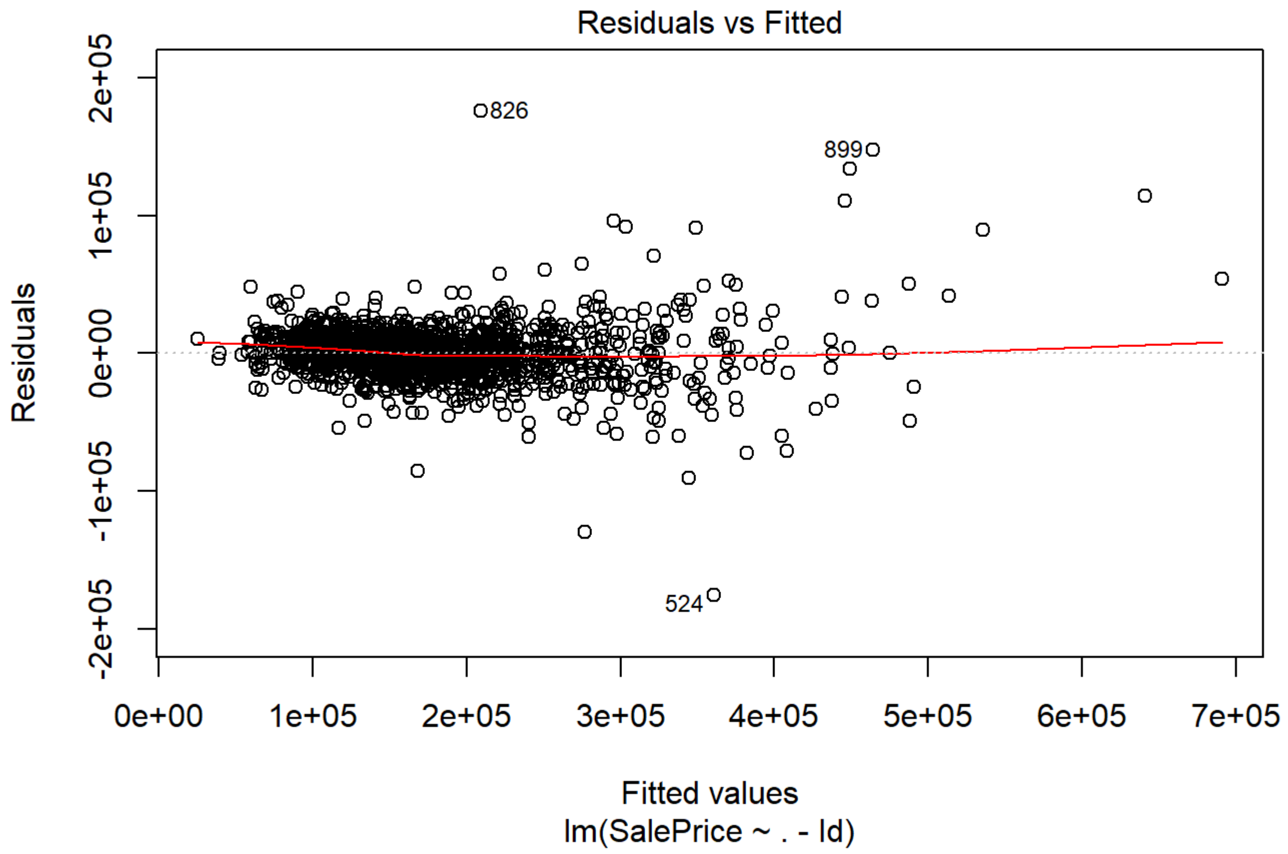
```
 [1] "MSZoning"     "Utilities"    "Exterior1st"  "Exterior2nd"
 [5] "MasVnrArea"   "BsmtFinSF1"   "BsmtFinSF2"   "BsmtUnfSF"
 [9] "TotalBsmtSF"  "BsmtFullBath" "BsmtHalfBath" "KitchenQual"
[13] "Functional"   "GarageCars"   "GarageArea"   "SaleType"
```

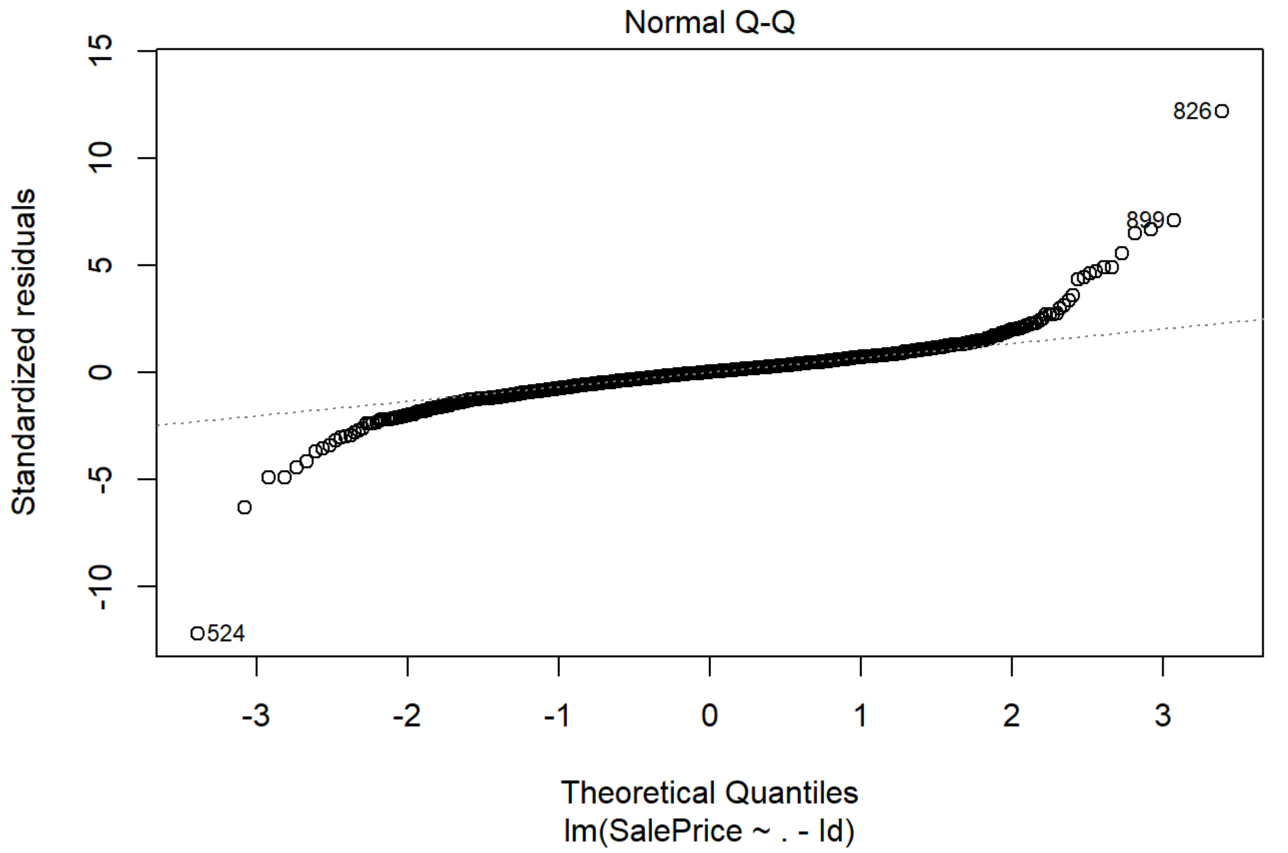
```
## Choose most common factor
test$MSZoning[is.na(test$MSZoning)] <- "RL"
test$Exterior1st[is.na(test$Exterior1st)] <- "VinylSd"
test$Exterior2nd[is.na(test$Exterior2nd)] <- "VinylSd"
test$KitchenQual[is.na(test$KitchenQual)] <- "TA"
test$Functional[is.na(test$Functional)] <- "Typ"
test$SaleType[is.na(test$SaleType)] <- "WD"

for (i in 1:nrow(test)) {
  ## Check logic on Masonry veneer
  if (is.na(test$MasVnrArea[i])) {
    if (test$MasVnrType[i] == "None") {
      test$MasVnrArea[i] <- 0  # if there isn't any masonry veneer, then the NA should be replac
ed with 0
    } else {
      test$MasVnrArea[i] <- mean(test$MasVnrArea, na.rm=T)  # if there is veneer, replace with t
he average
    }
  }

  ## Basement
  if (is.na(test$BsmtFinSF1[i])) {
    if (test$BsmtQual[i] != "N/A") {
      # if there is a basement, then use the average
      test$BsmtFinSF1[i] <- mean(test$BsmtFinSF1, na.rm=T)
    } else {
      # if there isn't a basement, use 0
      test$BsmtFinSF1[i] <- 0
    }
  }
  if (is.na(test$BsmtFinSF2[i])) {
    if (test$BsmtQual[i]!="N/A") {
      test$BsmtFinSF2[i] <- mean(test$BsmtFinSF2, na.rm=T)
    } else {
      test$BsmtFinSF2[i] <- 0
    }
  }
  if (is.na(test$BsmtUnfSF[i])) {
    if (test$BsmtQual[i]!="N/A") {
      test$BsmtUnfSF[i] <- mean(test$BsmtUnfSF, na.rm=T)
    } else {
      test$BsmtUnfSF[i] <- 0
    }
  }
  if (is.na(test$TotalBsmtSF[i])) {
    if (test$BsmtQual[i]!="N/A") {
      test$TotalBsmtSF[i] <- mean(test$TotalBsmtSF, na.rm=T)
    } else {
      test$TotalBsmtSF[i] <- 0
    }
  }
  if (is.na(test$BsmtFullBath[i])) {
    if (test$BsmtQual[i]!="N/A") {
```

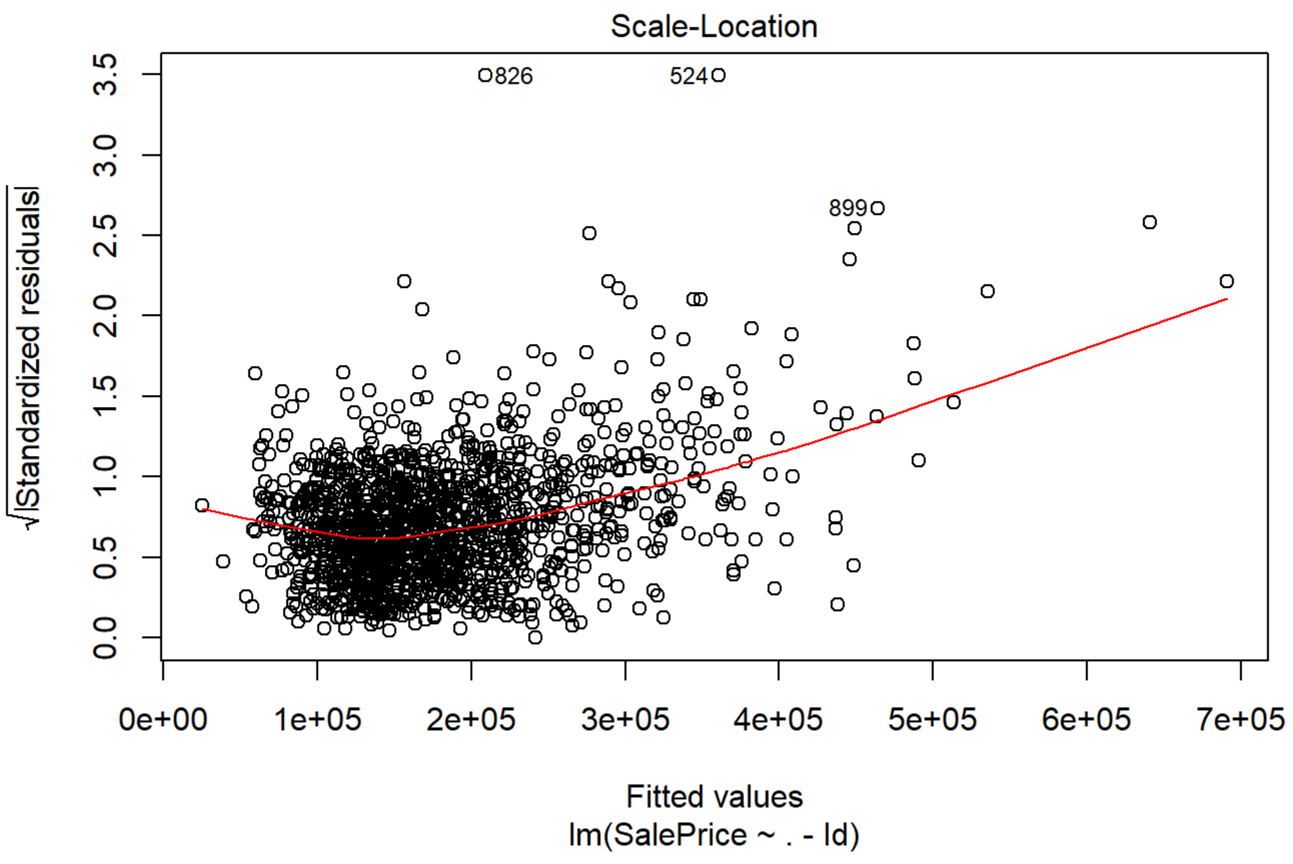
```r
      test$BsmtFullBath[i] <- mean(test$BsmtFullBath, na.rm=T)
    } else {
      test$BsmtFullBath[i] <- 0
    }
  }
  if (is.na(test$BsmtHalfBath[i])) {
    if (test$BsmtQual[i]!="N/A") {
      test$BsmtHalfBath[i] <- mean(test$BsmtHalfBath, na.rm=T)
    } else {
      test$BsmtHalfBath[i] <- 0
    }
  }

  ## Garage
  if (is.na(test$GarageCars[i])) {
    if (test$GarageType[i] != "N/A") {
      # if there is a garage, then use the average
      test$GarageCars[i] <- mean(test$GarageCars, na.rm=T)
    } else {
      # if there isn't a basement, use 0
      test$GarageCars[i] <- 0
    }
  }
  if (is.na(test$GarageArea[i])) {
    if (test$GarageType[i] != "N/A") {
      # if there is a garage, then use the average
      test$GarageArea[i] <- mean(test$GarageArea, na.rm=T)
    } else {
      # if there isn't a basement, use 0
      test$GarageArea[i] <- 0
    }
  }
}
```

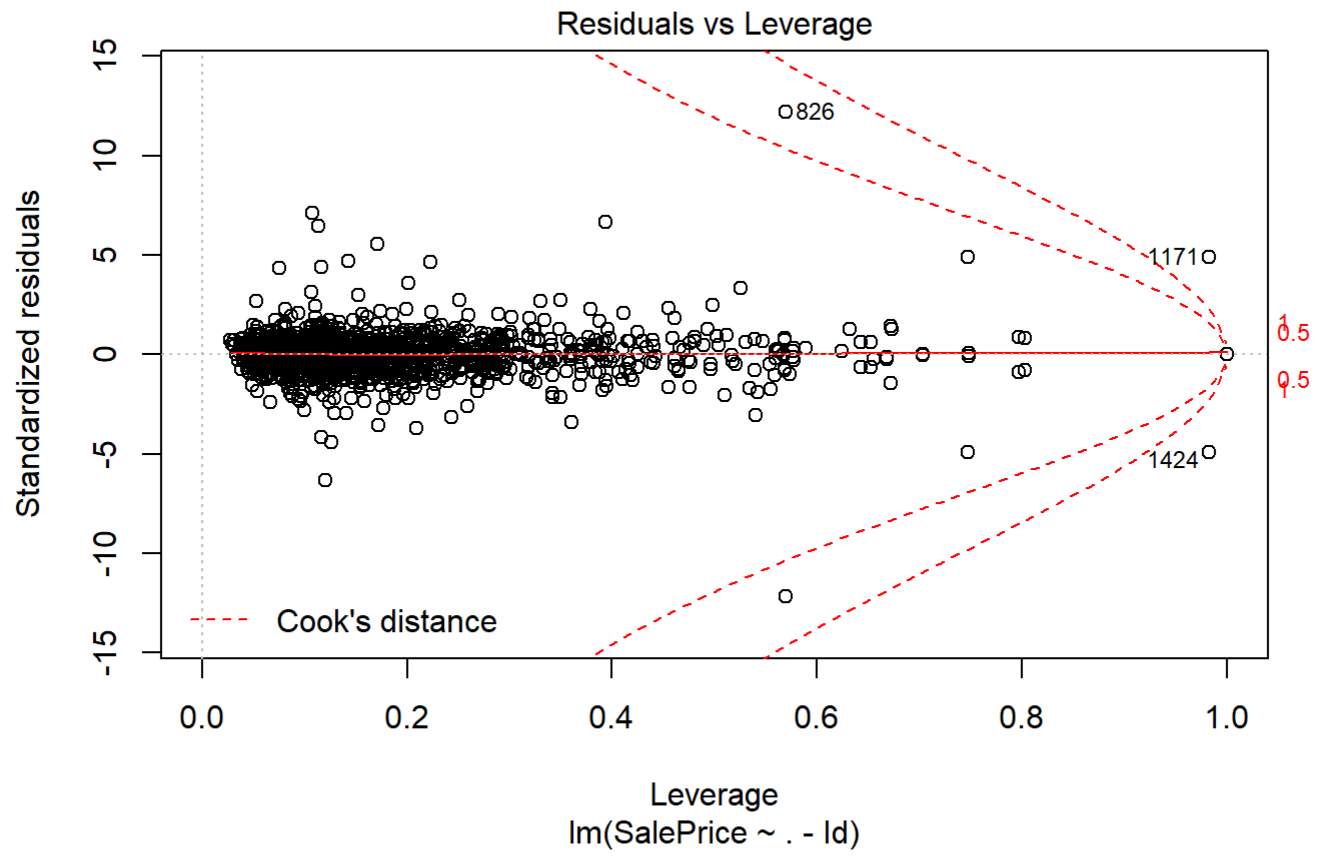
Linear Diagnostics for Training Data

```r
lin_fit <- lm(SalePrice~.-Id, data=train)
plot(lin_fit)
```


Residuals vs Fitted
Residuals
2e+05
1e+05
0e+00
-1e+05
-2e+05
826
899
524
0e+00
1e+05
2e+05
3e+05
4e+05
5e+05
6e+05
7e+05
Fitted values
lm(SalePrice ~ . - Id)


Normal Q-Q
Standardized residuals
15
10
5
0
-5
-10
826
899
524
-3
-2
-1
0
1
2
3
Theoretical Quantiles
lm(SalePrice ~ . - Id)





Based on the warning message, points 121, 186, 250, 325, 332, 346, 375, 398, 532, 582, 594, 664, 808, 819, 941, 945, 998, 1006, 1182, 1225, 1264, 1269, 1291, 1314, 1363, 1378 should be removed. Points 524 and 826 are noticeably bad on the normal Q-Q plot and should be removed. Points 1171 and 1424 also have high leverage, so they will be removed. The residuals vs. fitted plot looks pretty good (so the data is reasonably linear and tere is a fairly constant variance of the error terms). The outliers (826, 524) are removed because they are also high leverage points.

```
train <- train[-c(121,186,250,325,332,346,375,398,524,532,582,594,664,808,819,826,941,945,998,10
06,1171,1182,1225,1264,1269,1291,1314,1363,1378,1424),]
```

After removing the statistically “bad” points, the variable `Utilities` can be removed because they all have the same value

```
train <- train[,-9]; test <- test[,-9]
```

Make Model Matrices

First, I add a dummy column to the test data in order to make a model matrix. Then I create the model matrices and remove the columns from the two matrices that don’t match each other.

```
test$SalePrice <- rep(1, nrow(test))  # add a dummy variable

train.mat <- model.matrix(SalePrice~.-Id, data=train); test.mat <- model.matrix(SalePrice~.-Id,
 data=test)

rm.train <- c()
for (i in 1:length(colnames(train.mat))) {
  if (sum(colnames(train.mat)[i] == colnames(test.mat))!=1) {
    rm.train <- c(rm.train, i)
  }
}

rm.test <- c()
for (j in 1:length(colnames(test.mat))) {
  if (sum(colnames(test.mat)[j] == colnames(train.mat))!=1) {
    rm.test <- c(rm.test, j)
  }
}
train.mat <- train.mat[,-rm.train]; test.mat <- test.mat[,-rm.test]


save(train.mat, file="train.mat.JV.RData")
save(test.mat, file="test.mat.JV.RData")
save(test, file="test.JV.RData")
save(train, file="train.JV.RData")
```

The following code creates dummy variable matrices including the response. It then removes columns that contain all zeros in either set.

```
train.mat.ext <- cbind(train$Id, train.mat[,-1], train$SalePrice)
colnames(train.mat.ext) <- c("Id", colnames(train.mat)[-1], "SalePrice")

test.mat.ext <- cbind(test$Id, test.mat[,-1], rep(1, nrow(test.mat)))
colnames(test.mat.ext) <- c("Id", colnames(test.mat)[-1], "SalePrice")

zeros.tr <- c(1:ncol(train.mat.ext))[apply(train.mat.ext, 2, sum)==0]
zeros.te <- c(1:ncol(test.mat.ext))[apply(test.mat.ext, 2, sum)==0]

train.mat.ext <- train.mat.ext[,-c(zeros.tr, zeros.te)]
test.mat.ext <- test.mat.ext[,-c(zeros.tr, zeros.te)]

save(train.mat.ext, file="train.mat.ext.JV.RData"); save(test.mat.ext, file="test.mat.ext.JV.RDa
ta")
```