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.

```r
## 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.

```r
## 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

```r
## 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
```

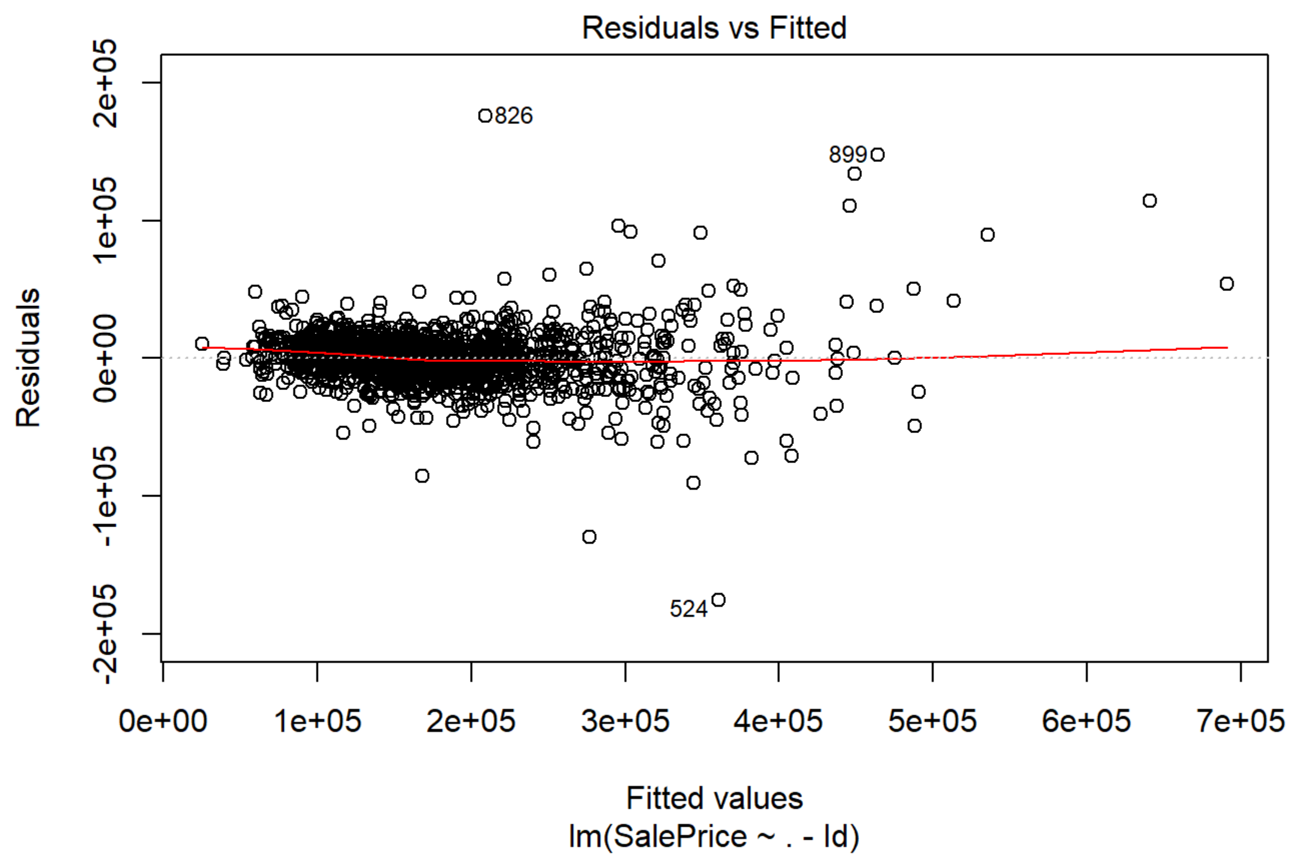
```
na_testcol_boo <- ifelse(ct_na_testcol!=0, T, F)
colnames(test)[na_testcol_boo]
```

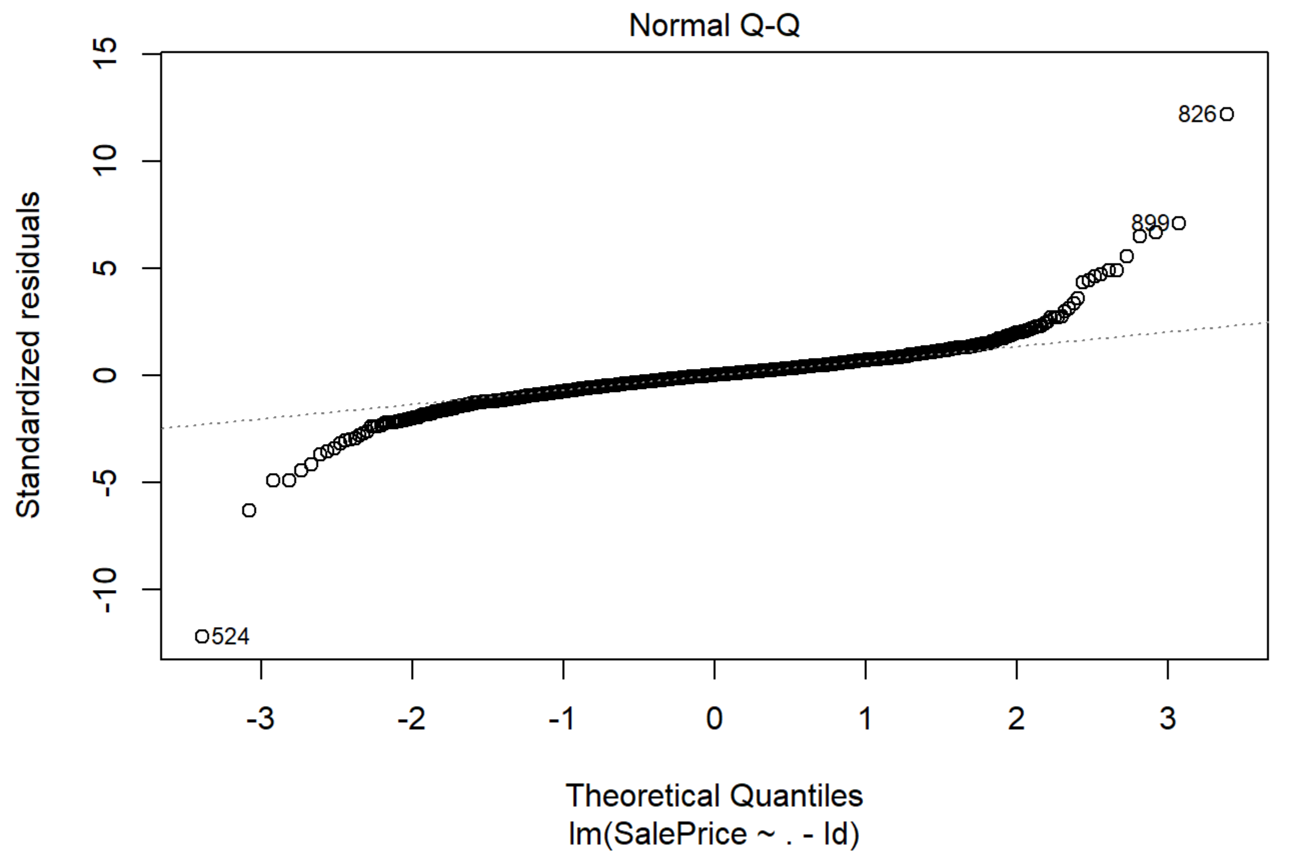
```
 [1] "MSZoning"     "Utilities"    "Exterior1st"  "Exterior2nd" 
 [5] "MasVnrArea"   "BsmtFinSF1"   "BsmtFinSF2"   "BsmtUnfSF"   
 [9] "TotalBsmtSF"  "BsmtFullBath" "BsmtHalfBath" "KitchenQual" 
[13] "Functional"   "GarageCars"   "GarageArea"   "SaleType"
```

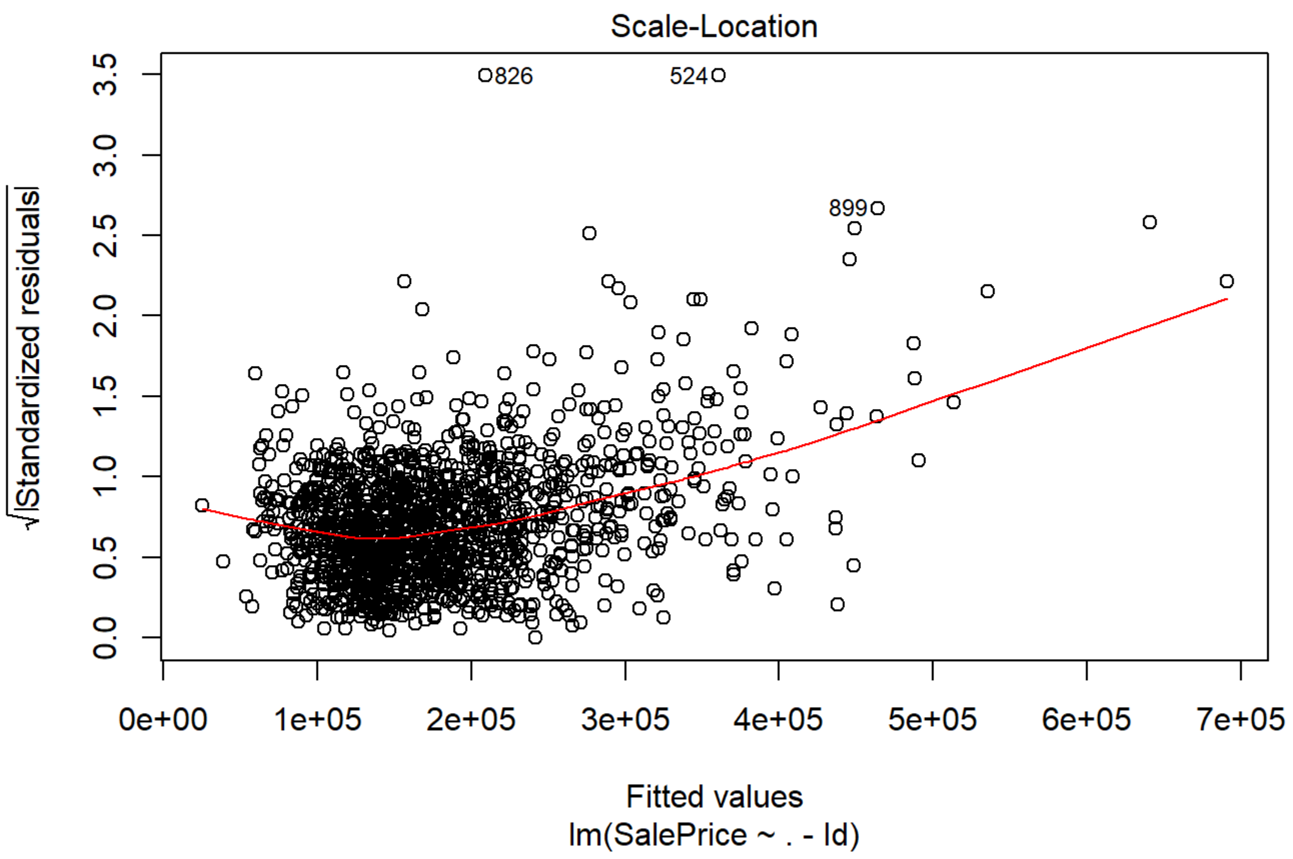
```
## Choose most common factor
test$MSZoning[is.na(test$MSZoning)] <- "RL"
test$Exterior1st[is.na(test$Exterior1st)] <- "VinylSd"
test$Exterior2nd[is.na(test$Exterior2nd)] <- "VinylSd"
test$KitchenQual[is.na(test$KitchenQual)] <- "TA"
test$Functional[is.na(test$Functional)] <- "Typ"
test$SaleType[is.na(test$SaleType)]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

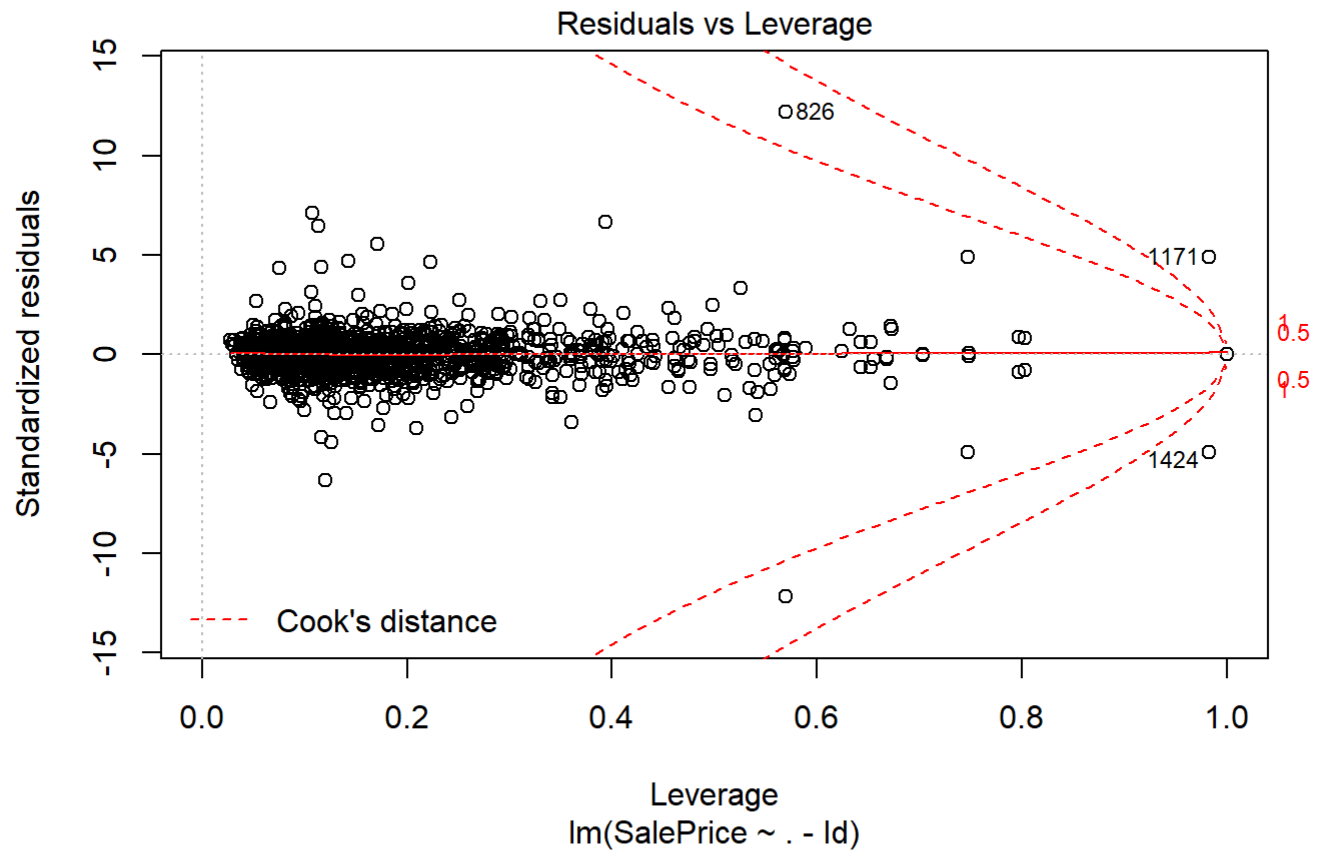
Linear Diagnostics for Training Data

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


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


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





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")
```

Subset Selection

The following code chunk imports the pre-processed data. The test data set is given a dummy variable so that future functions can work on this data (there needs to be a dummy response so that the predict.regsubsets function can apply a formula to the test set for prediction).

```
load("train.JV.RData"); load("test.JV.RData")

test$SalePrice <- rep(1, nrow(test))  # make a "dummy" column for the test SalePrice in order fo
r the prediction to work
```

This code is used to check whether the model matrices for the training set and the test set match up. Any columns in the training model matrix that aren't present in the test model matrix have their variable removed because that variable won't be used to predict in the test set.

```
train_mat <- model.matrix(SalePrice ~.-Id, data=train); test_mat <- model.matrix(SalePrice~.-Id,
data=test)
for (i in 1:length(colnames(train_mat))) {
  if (sum(colnames(train_mat)[i] == colnames(test_mat))!=1) {
    print(colnames(train_mat)[i])
  }
}
```

```
bad_i <- c(1:length(colnames(test)))[colnames(test) %in% c("Condition2", "HouseStyle", "RoofMat
l", "Exterior1st", "Exterior2nd", "Heating", "Electrical", "GarageQual", "PoolQC", "MiscFeature"
)]
train <- train[,-bad_i]
test <- test[,-bad_i]
```

The `leaps` library has a `regsubsets` function that can be used to fit stepwise models and best subset models. The `Metrics` library has a `rmsle` function that can be used to evaluate error, because this is the same metric that Kaggle uses.

```
library(leaps)
library(Metrics)
```

The `regsubsets` function used in the `leaps` library doesn't have a prediction attribute, so this function will serve that purpose.

```r
predict.regsubsets <- function (object, newdata , id, ...) {
  form <- as.formula(object$call[[2]])  # formula of full model
  test.mat <- model.matrix(form, newdata)  # building an X matrix from newdata
  coefi <- coef(object, id = id) # coefficient estimates associated with the object model
  xvars <- names(coefi) # names of the non-zero coefficient estimates

  for (i in 1:length(xvars)) {
    if (sum(xvars[i]==colnames(test.mat))!=1) {
      print(xvars[i])
    }
  }


  return(test.mat[,xvars] %*% coefi) # X[,non-zero variables] * Coefficients[non-zero variables]
}
```

Best Subsets Selection

Best subsets selection tests all the possible models from 0- to p-dimensional models, selecting the best m-dimensional model using the model RSS. Then all of the $p+1$ models are compared to each other using an estimation of the test error (e.g., adjusted R^2), and then the best model is selected.
Best subsets selection examines 2^p different models; in this case, the number of predictors results in more than 10^{20} models, which is computationally very expensive, so I will not perform this method.

Forward Stepwise Selection

Forward stepwise selection starts with a null model ($Y = \beta_0 + \epsilon$), and then tests all models of m dimension and selects the best using the model RSS. Each subsequent model includes all the previously selected predictors plus one more predictor that most improves the model. Once there are $p+1$ models (one for each possible dimension), the best model is selected using some sort of test error estimation criteria (e.g., adjusted R^2).

The following code fits a forward stepwise model to the training data, using a maximum of 200 variables (which would be almost a full model using all qualitative and categorical-dummy variables); a higher number of variables seems excessive.

```r
fwd.reg <- regsubsets(SalePrice~.-Id, data=train, nvmax=200, method="forward")  # perform forwar
d stepwise method
```

```
Reordering variables and trying again:
```

```r
fwd.smry <- summary(fwd.reg)  # store summary
```

Adjusted R^2

The adjusted R^2 metric is used to select “good” models; it is an estimate of a test error. It is an adjustment to the training error that accounts for the fact that the R^2 of a model always increases when there are more predictors.

The following code extracts the dimension of the model that has the best (maximum) adjusted R^2 metric. It then uses that dimension to make a prediction for the `SalePrice` of the test data using the model developed in the beginning of this section. Then the results are exported to a .csv file for submission to Kaggle.

```
fwd.id.adjr2 <- c(0:length(fwd.smry$adjr2)-1)[(fwd.smry$adjr2 == max(fwd.smry$adjr2))==T]  # whi
ch dimension of the forward model is best in terms of adj.R^2

fwd.pred.adjr2 <- predict.regsubsets(object=fwd.reg, newdata=test, id=fwd.id.adjr2)  # predictio
n using forward stepwise adj.R^2  model

fwd.pred.adjr2.df <- data.frame(Id=test$Id, SalePrice=fwd.pred.adjr2); write.csv(fwd.pred.adjr2.
df, "fwd.pred.adjr2.df.csv", row.names=F)  # write to CSV
```

The following code estimates the test error using 5-fold cross-validation from the results above.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train)), breaks=5, labels=FALSE)  # split data into 5 folds

fwd.rmslek.adjr2 <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.k <- train[fold.index != k,]  # fold training set
    test.k <- train[fold.index == k,]  # fold test set
    true.y <- test.k[,"SalePrice"]  # fold test response

    fwd.kpred.adjr2 <- predict.regsubsets(fwd.reg, newdata=test.k, id=fwd.id.adjr2)  # make pred
iction for this test fold
    fwd.rmslek.adjr2 <- c(fwd.rmslek.adjr2, rmsle(actual=true.y, predicted=fwd.kpred.adjr2))  #
 store the RMSLE metric for this test fold
}

fwd.rmsle.adjr2 <- mean(fwd.rmslek.adjr2)  # calculate the average RMSLE
```

Mallow’s C_p

The Mallow’s C_p metric is used to select “good” models; it is an estimate of a test error. It is an adjustment to the training error that penalizes high-dimensional models.

The following code extracts the dimension of the model that has the best (minimum) Mallow’s C_p metric. It then uses that dimension to make a prediction for the `SalePrice` of the test data using the model developed in the beginning of this section. Then the results are exported to a .csv file for submission to Kaggle.

```
fwd.id.cp <- c(0:length(fwd.smry$cp)-1)[(fwd.smry$cp == min(fwd.smry$cp))==T]  # which dimension
 of the forward model is best in terms of Cp

fwd.pred.cp <- predict.regsubsets(object=fwd.reg, newdata=test, id=fwd.id.cp)  # prediction usin
g forward stepwise Cp model

fwd.pred.cp.df <- data.frame(Id=test$Id, SalePrice=fwd.pred.cp); write.csv(fwd.pred.cp.df, "fwd.
pred.cp.df.csv", row.names=F)  # write to CSV
```

The following code estimates the test error using 5-fold cross-validation from the results above.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train)), breaks=5, labels=FALSE)  # split data into 5 folds

fwd.rmslek.cp <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.k <- train[fold.index != k,]  # fold training set
    test.k <- train[fold.index == k,]  # fold test set
    true.y <- test.k[,"SalePrice"]  # fold test response

    fwd.kpred.cp <- predict.regsubsets(fwd.reg, newdata=test.k, id=fwd.id.cp)  # make prediction
 for this test fold
    fwd.rmslek.cp <- c(fwd.rmslek.cp, rmsle(actual=true.y, predicted=fwd.kpred.cp))  # store the
 RMSLE metric for this test fold
}

fwd.rmsle.cp <- mean(fwd.rmslek.cp)  # calculate the average RMSLE
```

Bayes's Information Criterion

The Bayes's Information Criterion ("BIC") is used to select "good" models; it is an estimate of a test error. It is an adjustment to the training error that penalizes high-dimensional models based on the number of points used to fit the model. This method tends to prefer lower-dimensional models than the adjusted R^2 or Mallow's C_p.

The following code extracts the dimension of the model that has the best (minimum) BIC metric. It then uses that dimension to make a prediction for the `SalePrice` of the test data using the model developed in the beginning of this section. Then the results are exported to a .csv file for submission to Kaggle.

```
fwd.id.bic <- c(0:length(fwd.smry$bic)-1)[(fwd.smry$bic == min(fwd.smry$bic))==T]  # which dimen
sion of the forward model is best in terms of bic

fwd.pred.bic <- predict.regsubsets(object=fwd.reg, newdata=test, id=fwd.id.bic)  # prediction us
ing forward stepwise bic model

fwd.pred.bic.df <- data.frame(Id=test$Id, SalePrice=fwd.pred.bic); write.csv(fwd.pred.bic.df, "f
wd.pred.bic.df.csv", row.names=F)  # write to CSV
```

The following code estimates the test error using 5-fold cross-validation from the results above.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train)), breaks=5, labels=FALSE)  # split data into 5 folds

fwd.rmslek.bic <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.k <- train[fold.index != k,]  # fold training set
    test.k <- train[fold.index == k,]  # fold test set
    true.y <- test.k[,"SalePrice"]  # fold test response

    fwd.kpred.bic <- predict.regsubsets(fwd.reg, newdata=test.k, id=fwd.id.bic)  # make predicti
on for this test fold
    fwd.rmslek.bic <- c(fwd.rmslek.bic, rmsle(actual=true.y, predicted=fwd.kpred.bic))  # store
 the RMSLE metric for this test fold
}

fwd.rmsle.bic <- mean(fwd.rmslek.bic)  # calculate the average RMSLE
```

Cross Validation

The cross-validation approach to model selection is used to select tuning parameters; it estimates the test error, and then selects the tuning parameter with the lowest estimated test error.

The following code uses 5-fold cross-validation to fit a forward stepwise model, finds the “best” dimension ,and finds the estimated RMSLE for that dimension. It then makes a prediction using the full training data set and prints the results to a .csv file for submission.

```
set.seed(1)  # set seed for consistency of fold breaks
fold.index <- cut(sample(1:nrow(train)), breaks=5, labels=FALSE)  # split data into 5 folds

nv <- 200

fwd.rmsleik.cv <- matrix(NA, ncol=nv, nrow=5)

for (k in 1:5) {
  train.k <- train[fold.index != k,]  # k training data
  test.k <- train[fold.index == k,]  # k test data
  true.y <- test.k[,"SalePrice"]  # k test response

  fwd.regk <- regsubsets(SalePrice ~ .-Id, data = train.k, nvmax = nv, method="forward")  # fit
 model using training fold

  for (i in 1:nv) {
    fwd.predk <- predict(fwd.regk, test.k, id = i)  # make predictions using all possible values
 of model dimension
    fwd.rmsleik.cv[k,i] <- rmsle(actual=true.y, predicted=fwd.predk)  # store the prediction err
ors for each possible dimension
  }
}
```

```
Reordering variables and trying again:
Reordering variables and trying again:
Reordering variables and trying again:
Reordering variables and trying again:
Reordering variables and trying again:
```

```
fwd.rmslei.cv <- colMeans(fwd.rmsleik.cv, na.rm=T)  # calculate the estimated prediction error f
or each possible dimension

fwd.id.cv <- which(fwd.rmslei.cv==min(fwd.rmslei.cv))-1  # which dimension is the best dimensio
n?

fwd.pred.cv <- predict.regsubsets(object=fwd.reg, newdata=test, id=fwd.id.cv)  # prediction usin
g forward stepwise CV model

fwd.pred.cv.df <- data.frame(Id=test$Id, SalePrice=fwd.pred.cv); write.csv(fwd.pred.cv.df, "fwd.
pred.cv.df.csv", row.names=F)  # write to CSV
```

Summary of Results

Method	Model	Dimension	Estimated RMSLE	Actual RMSLE
Forward Stepwise	Adjusted R^2	125	0.14826	0.29415
Forward Stepwise	Mallow’s Cp	71	0.18468	0.22928
Forward Stepwise	Bayes’ IC	39	0.24152	0.22727
Forward Stepwise	Cross-Validation	199	0.14963	0.22282

Backward Stepwise Selection

Backward stepwise selection starts with a full model ($Y = \beta_0 + +\beta_1 X_1 + ... + \beta_p X_p + \epsilon$), and then tests all models of m dimension and selects the best using the model RSS. Each subsequent model includes all the previously selected predictors except one less predictor whose removal most improves the model. Once there are $p+1$ models (one for each possible dimension), the best model is selected using some sort of test error estimation criteria.

The following code fits a backward stepwise model to the training data, using a maximum of 200 variables (which would be almost a full model using all qualitative and categorical-dummy variables); a higher number of variables seems excessive.

```
bwd.reg <- regsubsets(SalePrice~.-Id, data=train, nvmax=200, method="backward")  # perform backw
ard stepwise method
```

```
Reordering variables and trying again:
```

```
bwd.smry <- summary(bwd.reg)  # store summary
```

Adjusted R^2

The adjusted R^2 metric is used to select “good” models; it is an estimate of a test error. It is an adjustment to the training error that accounts for the fact that the R^2 of a model always increases when there are more predictors.

The following code extracts the dimension of the model that has the best (maximum) adjusted R^2 metric. It then uses that dimension to make a prediction for the `SalePrice` of the test data using the model developed in the beginning of this section. Then the results are exported to a .csv file for submission to Kaggle.

```
bwd.id.adjr2 <- c(0:length(bwd.smry$adjr2)-1)[(bwd.smry$adjr2 == max(bwd.smry$adjr2))==T]  # whi
ch dimension of the backward model is best in terms of adj.R^2

bwd.pred.adjr2 <- predict.regsubsets(object=bwd.reg, newdata=test, id=bwd.id.adjr2)  # predictio
n using backward stepwise adj.R^2  model

bwd.pred.adjr2.df <- data.frame(Id=test$Id, SalePrice=bwd.pred.adjr2); write.csv(bwd.pred.adjr2.
df, "bwd.pred.adjr2.df.csv", row.names=F)  # write to CSV
```

The following code estimates the test error using 5-fold cross-validation from the results above.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train)), breaks=5, labels=FALSE)  # split data into 5 folds

bwd.rmslek.adjr2 <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.k <- train[fold.index != k,]  # fold training set
    test.k <- train[fold.index == k,]  # fold test set
    true.y <- test.k[,"SalePrice"]  # fold test response

    bwd.kpred.adjr2 <- predict.regsubsets(bwd.reg, newdata=test.k, id=bwd.id.adjr2)  # make pred
iction for this test fold
    bwd.rmslek.adjr2 <- c(bwd.rmslek.adjr2, rmsle(actual=true.y, predicted=bwd.kpred.adjr2))  #
 store the RMSLE metric for this test fold
}

bwd.rmsle.adjr2 <- mean(bwd.rmslek.adjr2)  # calculate the average RMSLE
```

Mallow’s C_p

The Mallow’s C_p metric is used to select “good” models; it is an estimate of a test error. It is an adjustment to the training error that penalizes high-dimensional models.

The following code extracts the dimension of the model that has the best (minimum) Mallow’s C_p metric. It then uses that dimension to make a prediction for the `SalePrice` of the test data using the model developed in the beginning of this section. Then the results are exported to a .csv file for submission to Kaggle.

```
bwd.id.cp <- c(0:length(bwd.smry$cp)-1)[(bwd.smry$cp == min(bwd.smry$cp))==T]  # which dimension
 of the backward model is best in terms of Cp

bwd.pred.cp <- predict.regsubsets(object=bwd.reg, newdata=test, id=bwd.id.cp)  # prediction usin
g backward stepwise Cp model

bwd.pred.cp.df <- data.frame(Id=test$Id, SalePrice=bwd.pred.cp); write.csv(bwd.pred.cp.df, "bwd.
pred.cp.df.csv", row.names=F)  # write to CSV
```

The following code estimates the test error using 5-fold cross-validation from the results above.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train)), breaks=5, labels=FALSE)  # split data into 5 folds

bwd.rmslek.cp <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.k <- train[fold.index != k,]  # fold training set
    test.k <- train[fold.index == k,]  # fold test set
    true.y <- test.k[,"SalePrice"]  # fold test response

    bwd.kpred.cp <- predict.regsubsets(bwd.reg, newdata=test.k, id=bwd.id.cp)  # make prediction
 for this test fold
    bwd.rmslek.cp <- c(bwd.rmslek.cp, rmsle(actual=true.y, predicted=bwd.kpred.cp))  # store the
 RMSLE metric for this test fold
}

bwd.rmsle.cp <- mean(bwd.rmslek.cp)  # calculate the average RMSLE
```

Bayes's Information Criterion

The Bayes's Information Criterion ("BIC") is used to select "good" models; it is an estimate of a test error. It is an adjustment to the training error that penalizes high-dimensional models based on the number of points used to fit the model. This method tends to prefer lower-dimensional models than the adjusted R^2 or Mallow's C_p.

The following code extracts the dimension of the model that has the best (minimum) BIC metric. It then uses that dimension to make a prediction for the `SalePrice` of the test data using the model developed in the beginning of this section. Then the results are exported to a .csv file for submission to Kaggle.

```
bwd.id.bic <- c(0:length(bwd.smry$bic)-1)[(bwd.smry$bic == min(bwd.smry$bic))==T]  # which dimen
sion of the backward model is best in terms of bic

bwd.pred.bic <- predict.regsubsets(object=bwd.reg, newdata=test, id=bwd.id.bic)  # prediction us
ing backward stepwise bic model

bwd.pred.bic.df <- data.frame(Id=test$Id, SalePrice=bwd.pred.bic); write.csv(bwd.pred.bic.df, "b
wd.pred.bic.df.csv", row.names=F)  # write to CSV
```

The following code estimates the test error using 5-fold cross-validation from the results above.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train)), breaks=5, labels=FALSE)  # split data into 5 folds

bwd.rmslek.bic <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.k <- train[fold.index != k,]  # fold training set
    test.k <- train[fold.index == k,]  # fold test set
    true.y <- test.k[,"SalePrice"]  # fold test response

    bwd.kpred.bic <- predict.regsubsets(bwd.reg, newdata=test.k, id=bwd.id.bic)  # make predicti
on for this test fold
    bwd.rmslek.bic <- c(bwd.rmslek.bic, rmsle(actual=true.y, predicted=bwd.kpred.bic))  # store
 the RMSLE metric for this test fold
}

bwd.rmsle.bic <- mean(bwd.rmslek.bic)  # calculate the average RMSLE
```

Cross Validation

The cross-validation approach to model selection is used to select tuning parameters; it estimates the test error, and then selects the tuning parameter with the lowest estimated test error.

The following code uses 5-fold cross-validation to fit a backward stepwise model, finds the “best” dimension ,and finds the estimated RMSLE for that dimension. It then makes a prediction using the full training data set and prints the results to a .csv file for submission.

```
set.seed(1)  # set seed for consistency of fold breaks
fold.index <- cut(sample(1:nrow(train)), breaks=5, labels=FALSE)  # split data into 5 folds

nv <- 200

bwd.rmsleik.cv <- matrix(NA, ncol=nv, nrow=5)

for (k in 1:5) {
  train.k <- train[fold.index != k,]  # k training data
  test.k <- train[fold.index == k,]  # k test data
  true.y <- test.k[,"SalePrice"]  # k test response

  bwd.regk <- regsubsets(SalePrice ~ .-Id, data = train.k, nvmax = nv, method="backward")  # fit
 model using training fold

  for (i in 1:nv) {
    bwd.predk <- predict(bwd.regk, test.k, id = i)  # make predictions using all possible values
 of model dimension
    bwd.rmsleik.cv[k,i] <- rmsle(actual=true.y, predicted=bwd.predk)  # store the prediction err
ors for each possible dimension
  }
}
```

```
Reordering variables and trying again:
Reordering variables and trying again:
Reordering variables and trying again:
Reordering variables and trying again:
Reordering variables and trying again:
```

```
bwd.rmslei.cv <- colMeans(bwd.rmsleik.cv, na.rm=T)  # calculate the estimated prediction error f
or each possible dimension

bwd.id.cv <- which(bwd.rmslei.cv==min(bwd.rmslei.cv))-1  # which dimension is the best dimensio
n?

bwd.pred.cv <- predict.regsubsets(object=bwd.reg, newdata=test, id=bwd.id.cv)  # prediction usin
g forward stepwise CV model

bwd.pred.cv.df <- data.frame(Id=test$Id, SalePrice=bwd.pred.cv); write.csv(bwd.pred.cv.df, "bwd.
pred.cv.df.csv", row.names=F)  # write to CSV
```

Summary of Results

Method	Model	Dimension	Estimated RMSLE	Actual RMSLE
Backward Stepwise	Adjusted R^2	133	0.14081	0.21879
Backward Stepwise	Mallow's Cp	87	0.16272	0.21734
Backward Stepwise	Bayes' IC	43	0.20075	0.23554
Backward Stepwise	Cross-Validation	184	0.14970	0.22234

Mixed Stepwise Selection

Mixed stepwise selection starts with a null model ($Y = \beta_0 + \epsilon$), and then selects the best 1-dimensional method. For all subsequent models, it tries to add or remove variables, and then selects the best $m-1$ or $m+1$ model that improves the existing model.

The following code fits a mixed stepwise model to the training data, using a maximum of 100 variables (this is smaller than the other two methods used above, but is less computationally expensive).

```
mwd.reg <- regsubsets(SalePrice~.-Id, data=train, nvmax=100, method="seqrep")  # perform mixed s
tepwise method
```

```
Reordering variables and trying again:
```

```
mwd.smry <- summary(mwd.reg)  # store summary
```

Adjusted R^2

The adjusted R^2 metric is used to select “good” models; it is an estimate of a test error. It is an adjustment to the training error that accounts for the fact that the R^2 of a model always increases when there are more predictors.

The following code extracts the dimension of the model that has the best (maximum) adjusted R^2 metric. It then uses that dimension to make a prediction for the `SalePrice` of the test data using the model developed in the beginning of this section. Then the results are exported to a .csv file for submission to Kaggle.

```
mwd.id.adjr2 <- c(0:length(mwd.smry$adjr2)-1)[(mwd.smry$adjr2 == max(mwd.smry$adjr2))==T]  # whi
ch dimension of the mixed model is best in terms of adj.R^2

mwd.pred.adjr2 <- predict.regsubsets(object=mwd.reg, newdata=test, id=mwd.id.adjr2)  # predictio
n using mixed stepwise adj.R^2  model

mwd.pred.adjr2.df <- data.frame(Id=test$Id, SalePrice=mwd.pred.adjr2); write.csv(mwd.pred.adjr2.
df, "mwd.pred.adjr2.df.csv", row.names=F)  # write to CSV
```

The following code estimates the test error using 5-fold cross-validation from the results above.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train)), breaks=5, labels=FALSE)  # split data into 5 folds

mwd.rmslek.adjr2 <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.k <- train[fold.index != k,]  # fold training set
    test.k <- train[fold.index == k,]  # fold test set
    true.y <- test.k[,"SalePrice"]  # fold test response

    mwd.kpred.adjr2 <- predict.regsubsets(mwd.reg, newdata=test.k, id=mwd.id.adjr2)  # make pred
iction for this test fold
    mwd.rmslek.adjr2 <- c(mwd.rmslek.adjr2, rmsle(actual=true.y, predicted=mwd.kpred.adjr2))  #
 store the RMSLE metric for this test fold
}

mwd.rmsle.adjr2 <- mean(mwd.rmslek.adjr2)  # calculate the average RMSLE
```

Mallow’s C_p

The Mallow’s C_p metric is used to select “good” models; it is an estimate of a test error. It is an adjustment to the training error that penalizes high-dimensional models.

The following code extracts the dimension of the model that has the best (minimum) Mallow’s C_p metric. It then uses that dimension to make a prediction for the `SalePrice` of the test data using the model developed in the beginning of this section. Then the results are exported to a .csv file for submission to Kaggle.

```
mwd.id.cp <- c(0:length(mwd.smry$cp)-1)[(mwd.smry$cp == min(mwd.smry$cp))==T]  # which dimension
 of the mixed model is best in terms of Cp

mwd.pred.cp <- predict.regsubsets(object=mwd.reg, newdata=test, id=mwd.id.cp)  # prediction usin
g mixed stepwise Cp model

mwd.pred.cp.df <- data.frame(Id=test$Id, SalePrice=mwd.pred.cp); write.csv(mwd.pred.cp.df, "mwd.
pred.cp.df.csv", row.names=F)  # write to CSV
```

The following code estimates the test error using 5-fold cross-validation from the results above.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train)), breaks=5, labels=FALSE)  # split data into 5 folds

mwd.rmslek.cp <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.k <- train[fold.index != k,]  # fold training set
    test.k <- train[fold.index == k,]  # fold test set
    true.y <- test.k[,"SalePrice"]  # fold test response

    mwd.kpred.cp <- predict.regsubsets(mwd.reg, newdata=test.k, id=mwd.id.cp)  # make prediction
 for this test fold
    mwd.rmslek.cp <- c(mwd.rmslek.cp, rmsle(actual=true.y, predicted=mwd.kpred.cp))  # store the
 RMSLE metric for this test fold
}

mwd.rmsle.cp <- mean(mwd.rmslek.cp)  # calculate the average RMSLE
```

Bayes's Information Criterion

The Bayes's Information Criterion ("BIC") is used to select "good" models; it is an estimate of a test error. It is an adjustment to the training error that penalizes high-dimensional models based on the number of points used to fit the model. This method tends to prefer lower-dimensional models than the adjusted R^2 or Mallow's C_p.

The following code extracts the dimension of the model that has the best (minimum) BIC metric. It then uses that dimension to make a prediction for the `SalePrice` of the test data using the model developed in the beginning of this section. Then the results are exported to a .csv file for submission to Kaggle.

```
mwd.id.bic <- c(0:length(mwd.smry$bic)-1)[(mwd.smry$bic == min(mwd.smry$bic))==T]  # which dimen
sion of the mixed model is best in terms of bic

mwd.pred.bic <- predict.regsubsets(object=mwd.reg, newdata=test, id=mwd.id.bic)  # prediction us
ing mixed stepwise bic model

mwd.pred.bic.df <- data.frame(Id=test$Id, SalePrice=mwd.pred.bic); write.csv(mwd.pred.bic.df, "m
wd.pred.bic.df.csv", row.names=F)  # write to CSV
```

The following code estimates the test error using 5-fold cross-validation from the results above.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train)), breaks=5, labels=FALSE)  # split data into 5 folds

mwd.rmslek.bic <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.k <- train[fold.index != k,]  # fold training set
    test.k <- train[fold.index == k,]  # fold test set
    true.y <- test.k[,"SalePrice"]  # fold test response

    mwd.kpred.bic <- predict.regsubsets(mwd.reg, newdata=test.k, id=mwd.id.bic)  # make predicti
on for this test fold
    mwd.rmslek.bic <- c(mwd.rmslek.bic, rmsle(actual=true.y, predicted=mwd.kpred.bic))  # store
 the RMSLE metric for this test fold
}

mwd.rmsle.bic <- mean(mwd.rmslek.bic, na.rm=T)  # calculate the average RMSLE
```

Cross Validation

The cross-validation approach to model selection is used to select tuning parameters; it estimates the test error, and then selects the tuning parameter with the lowest estimated test error.

The following code uses 5-fold cross-validation to fit a mixed stepwise model, finds the “best” dimension ,and finds the estimated RMSLE for that dimension. It then makes a prediction using the full training data set and prints the results to a .csv file for submission. I will use a maximum dimension of 75 for computational efficiency.

```
set.seed(1)  # set seed for consistency of fold breaks
fold.index <- cut(sample(1:nrow(train)), breaks=5, labels=FALSE)  # split data into 5 folds

nv <- 75

mwd.rmsleik.cv <- matrix(NA, ncol=nv, nrow=5)

for (k in 1:5) {
  train.k <- train[fold.index != k,]  # k training data
  test.k <- train[fold.index == k,]  # k test data
  true.y <- test.k[,"SalePrice"]  # k test response

  mwd.regk <- regsubsets(SalePrice ~ .-Id, data = train.k, nvmax = nv, method="seqrep")  # fit m
odel using training fold

  for (i in 1:nv) {
    mwd.predk <- predict(mwd.regk, test.k, id = i)  # make predictions using all possible values
 of model dimension
    mwd.rmsleik.cv[k,i] <- rmsle(actual=true.y, predicted=mwd.predk)  # store the prediction err
ors for each possible dimension
  }
}
```

```
Reordering variables and trying again:
Reordering variables and trying again:
Reordering variables and trying again:
Reordering variables and trying again:
Reordering variables and trying again:
```

```
mwd.rmslei.cv <- colMeans(mwd.rmsleik.cv, na.rm=T)  # calculate the estimated prediction error f
or each possible dimension

mwd.id.cv <- which(mwd.rmslei.cv==min(mwd.rmslei.cv))-1  # which dimension is the best dimensio
n?

mwd.pred.cv <- predict.regsubsets(object=mwd.reg, newdata=test, id=mwd.id.cv)  # prediction usin
g mixed stepwise CV model

mwd.pred.cv.df <- data.frame(Id=test$Id, SalePrice=mwd.pred.cv); write.csv(mwd.pred.cv.df, "mwd.
pred.cv.df.csv", row.names=F)  # write to CSV
```

Summary of Results

Method	Model	Dimension	Estimated RMSLE	Actual RMSLE
Mixed Stepwise	Adjusted R^2	99	0.15639	0.27081
Mixed Stepwise	Mallow’s Cp	41	0.17950	0.20876
Mixed Stepwise	Bayes’ IC	79	0.20814	0.23318
Mixed Stepwise	Cross-Validation	71	0.20407	0.22880

Shrinkage

The following code imports the pre-processed data model matrices and responses.

```
load("train.mat.JV.RData"); load("test.mat.JV.RData")
load("train.JV.RData"); load("test.JV.RData")
```

The `glmnet` library has `glmnet` and `cvglmnet` functions that can be used to fit lasso and ridge models. The `Metrics` library has a `rmsle` function that can be used to evaluate error, because this is the same metric that Kaggle uses.

```
library(glmnet)
library(Metrics)
```

Ridge Regression

Ridge regression uses least squares estimation, but with the constraint that the sum of the squared coefficient estimates must be less than a value s (a penalty tuning parameter).

The following code fits a ridge method to the training data. It then uses 5-fold cross validation to select the best penalty tuning parameter λ using the built-in function `cv.glmnet`. Lastly, it makes a prediction using that tuning parameter and writes the results to a .csv file.

```
rdg.reg <- glmnet(train.mat, train$SalePrice, alpha=0)  # fit ridge regression

rdg.s <- cv.glmnet(train.mat, train$SalePrice, alpha=0, nfolds=5)$lambda.min  # extract best pen
alty tuning parameter

rdg.pred <- predict(rdg.reg, s=rdg.s, newx=test.mat)  # make prediction
rdg.pred.df <- data.frame(Id=test$Id, SalePrice=rdg.pred); write.csv(rdg.pred.df, "rdg.pred.csv"
, row.names=F)  # write to CSV
```

The following code estimates the test error using 5-fold cross-validation from the results above.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train.mat)), breaks=5, labels=FALSE)  # split data into 5 folds

rdg.rmslek <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.x <- train.mat[fold.index != k,]  # fold training set
    train.y <- train$SalePrice[fold.index != k]  # fold training response
    test.x <- train.mat[fold.index == k,]  # fold test set
    true.y <- train$SalePrice[fold.index == k]  # fold test response

    rdg.regk <- glmnet(train.x, train.y, alpha=0)  # fit ridge regression using training data
    rdg.predk <- predict(rdg.regk, newx=test.x, s=rdg.s, type="response")  # predict response fo
r test data
    rdg.rmslek <- c(rdg.rmslek, rmsle(actual=true.y, predicted=rdg.predk))  # store the RMSLE me
tric for this test fold
}

rdg.rmsle <- mean(rdg.rmslek)  # calculate the average RMSLE
```

Summary of Results

Method	Lambda	Estimated RMSLE	Actual RMSLE
Ridge Regression	15921.10313	0.13922	0.18548

The Lasso

The lasso uses least squares estimation, but with the constraint that the sum of the absolute value of the coefficient estimates must be less than a value s (a penalty tuning parameter).

The following code fits a lasso method to the training data. It then uses 5-fold cross validation to select the best penalty tuning parameter λ using the built-in function `cv.glmnet` . Lastly, it makes a prediction using that tuning parameter and writes the results to a .csv file.

```
las.reg <- glmnet(train.mat, train$SalePrice, alpha=1)  # fit lasso regression

las.s <- cv.glmnet(train.mat, train$SalePrice, alpha=1, nfolds=5)$lambda.min  # extract best pen
alty tuning parameter

las.pred <- predict(las.reg, s=las.s, newx=test.mat)  # make prediction
las.pred.df <- data.frame(Id=test$Id, SalePrice=las.pred); write.csv(las.pred.df, "las.pred.csv"
, row.names=F)  # write to CSV
```

The following code estimates the test error using 5-fold cross-validation from the results above.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train.mat)), breaks=5, labels=FALSE)  # split data into 5 folds

las.rmslek <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.x <- train.mat[fold.index != k,]  # fold training set
    train.y <- train$SalePrice[fold.index != k]  # fold training response
    test.x <- train.mat[fold.index == k,]  # fold test set
    true.y <- train$SalePrice[fold.index == k]  # fold test response

    las.regk <- glmnet(train.x, train.y, alpha=1)  # fit lasso regression using training data
    las.predk <- predict(las.regk, newx=test.x, s=las.s, type="response")  # predict response fo
r test data
    las.rmslek <- c(las.rmslek, rmsle(actual=true.y, predicted=las.predk))  # store the RMSLE me
tric for this test fold
}

las.rmsle <- mean(las.rmslek)  # calculate the average RMSLE
```

Summary of Results

Method	Lambda	Estimated RMSLE	Actual RMSLE
Lasso Regression	376.45280	0.13168	0.19808

Dimension Reduction

The following code imports the pre-processed data model matrices and responses. It also adds a dummy predictor variable to the test set for prediction function purposes.

```
load("train.mat.JV.RData"); load("test.mat.JV.RData")
load("train.JV.RData"); load("test.JV.RData")

test.mat <- cbind(test.mat, rep(1, nrow(test.mat)))  # create dummy column
colnames(test.mat) <- c(colnames(test.mat)[-1], "SalePrice")
```

The `pls` library has `pcr` and `plsr` functions that can be used to fit principal components regression and partial least squares models. The `Metrics` library has a `rmsle` function that can be used to evaluate error (the same metric that Kaggle uses).

```
library(pls)
library(Metrics)
```

The following code scales the quantitative predictors for use in the dimension reduction functions.

```
train.mat.scale <- train.mat  # initialize object

for (i in 1:ncol(train.mat.scale)) {
  if (is.numeric(train.mat.scale[i])) {
    train.mat.scale[i] <- scale(train.mat.scale[i])  # scale non-factors
  }
}

train.mat.scale <- cbind(train.mat.scale, train$SalePrice)  # need a model matrix that includes
 the response for PCR/PLS
colnames(train.mat.scale) <- c(colnames(train.mat), "SalePrice")
train.mat.scale <- train.mat.scale[,-1]
```

Principal Components Regression

Principal components regression is a combination of using least squares estimation with principal components analysis, which uses linear combinations of the original predictors to reduce the number of predictors (yielding a less variable model).

The following code uses cross-validation to find a good M parameter for PCR, and then writes the prediction to a csv file for submission.

```
pcr.reg <- pcr(SalePrice~., data=as.data.frame(train.mat.scale), scale=F, validation="CV")  # fi
t PCR regression and use CV
pcr.id <- c(0:ncol(train.mat.scale))[which.min(RMSEP(pcr.reg)$val[1,1,])]  # store value of m to
 minimize estimated RMSE
pcr.pred <- predict(pcr.reg, newdata=as.data.frame(test.mat), ncomp=pcr.id)  # make prediction

pcr.pred.df <- cbind(Id=test$Id, SalePrice=pcr.pred)
write.csv(pcr.pred.df, "pcr.pred.csv", row.names=F)
```

The following code estimates the RMSLE using 5-fold cross-validation for PCR.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train.mat.scale)), breaks=5, labels=FALSE)  # split data into 5
 folds

pcr.rmslek <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.x <- train.mat.scale[fold.index != k,]  # fold training set
    test.x <- train.mat.scale[fold.index == k,]  # fold test set
    true.y <- train$SalePrice[fold.index == k]  # fold test response

    pcr.regk <- pcr(SalePrice~., data=as.data.frame(train.x), scale=F, validation="CV")  # fit P
C regression using training data
    pcr.idk <- c(0:ncol(train.mat.scale))[which.min(RMSEP(pcr.reg)$val[1,1,])]  # extract number
 of components to use
    pcr.predk <- predict(pcr.regk, newx=test.x, M=pcr.idk)  # predict response for test data
    pcr.rmslek <- c(pcr.rmslek, rmsle(actual=true.y, predicted=pcr.predk))  # store the RMSLE me
tric for this test fold
}

pcr.rmsle <- mean(pcr.rmslek, na.rm=T)  # calculate the average RMSLE
```

Summary of Results

Method	Components	Estimated RMSLE	Actual RMSLE
PCR	161	0.55844	0.19129

Partial Least Squares

Partial least squares is a combination of using least squares estimation with supervised principal components analysis, which uses linear combinations of the original predictors as they relate to the response to reduce the number of predictors (yielding a less variable model).

The following code uses cross-validation to find a good M parameter for PLS, and then writes the prediction to a csv file for submission.

```
pls.reg <- plsr(SalePrice~., data=as.data.frame(train.mat.scale), scale=F, validation="CV")  # f
it pls regression and use CV
pls.id <- c(0:ncol(train.mat.scale))[which.min(RMSEP(pls.reg)$val[1,1,])]  # store value of m to
 minimize estimated RMSE
pls.pred <- predict(pls.reg, newdata=as.data.frame(test.mat), ncomp=pls.id)  # make prediction

pls.pred.df <- cbind(Id=test$Id, SalePrice=pls.pred)
write.csv(pls.pred.df, "pls.pred.csv", row.names=F)
```

The following code estimates the RMSLE using 5-fold cross-validation for PLS.

```
set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train.mat.scale)), breaks=5, labels=FALSE)  # split data into 5
 folds

pls.rmslek <- c()  # initialize storage of the k RMSLE's
for (k in 1:5) {
    train.x <- train.mat.scale[fold.index != k,]  # fold training set
    test.x <- train.mat.scale[fold.index == k,]  # fold test set
    true.y <- train$SalePrice[fold.index == k]  # fold test response

    pls.regk <- plsr(SalePrice~., data=as.data.frame(train.x), scale=F, validation="CV")  # fit
 PLS regression using training data
    pls.idk <- c(0:ncol(train.mat.scale))[which.min(RMSEP(pls.reg)$val[1,1,])]  # extract number
 of components to use
    pls.predk <- predict(pls.regk, newx=test.x, M=pls.idk)  # predict response for test data
    pls.rmslek <- c(pls.rmslek, rmsle(actual=true.y, predicted=pls.predk))  # store the RMSLE me
tric for this test fold
}

pls.rmsle <- mean(pls.rmslek, na.rm=T)  # calculate the average RMSLE
```

Summary of Results

Method	Components	Estimated RMSLE	Actual RMSLE
PLS	26	0.56133	0.18627

K-Nearest Neighbors

The following code imports the pre-processed data model matrices and responses.

```
load("train.mat.JV.RData"); load("test.mat.JV.RData")
load("train.JV.RData"); load("test.JV.RData")
```

The `FNN` library has a `knn.reg` function that can be used to do KNN regression. The `Metrics` library has a `rmsle` function that can be used to evaluate error, because this is the same metric that Kaggle uses. The `leaps` library has a `regsubsets` function that will be used to choose 1,2,3,4 “best” variables.

```
library(FNN)
library(Metrics)
library(leaps)
```

KNN

KNN uses averages to estimate a response based on the response of the neighbors of the test point in the training set. Generally, KNN is not good for $p > 4$, but it is worth a shot.

The following code tests different numbers of neighbors to find the one that yields the lowest estimated error using 5-fold cross-validation. It then uses that number of neighbors to make a prediction on the test set. This prediction is then written to a .csv file for submission to Kaggle.

```
K.max <- 100  # maximum possible number of neighbors within reasonable computation time

set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train.mat)), breaks=5, labels=FALSE)  # split data into 5 folds

knn.rmsleki.cv <- matrix(NA, ncol=K.max, nrow=5)  # initialize storage of the k RMSLE's
for (j in 1:5) {
    train.x <- train.mat[fold.index != j,]  # fold training set
    train.y <- train$SalePrice[fold.index != j]  # fold training response
    test.x <- train.mat[fold.index == j,]  # fold test set
    true.y <- train$SalePrice[fold.index == j]  # fold test response

    for (Ki in 1:K.max) {
      knn.predk.cv <- knn.reg(train=train.x, test=test.x, y=train.y, k=Ki)$pred  # make KNN pred
ictions using Ki neighbors
      knn.rmsleki.cv[j, Ki] <- rmsle(actual=true.y, predicted=knn.predk.cv)  # calculate estimat
ed RMSLE
    }
}

knn.rmslei.cv <- colMeans(knn.rmsleki.cv)  # calculate the average RMSLE for each possible numbe
r of neighbors

knn.id.cv <- which(knn.rmslei.cv==min(knn.rmslei.cv))  # extract right number of neighbors

knn.pred.cv <- knn.reg(train=train.mat, test=test.mat, y=train$SalePrice, k=knn.id.cv)$pred  # m
ake prediction
knn.pred.cv.df <- data.frame(Id=test$Id, SalePrice=knn.pred.cv); write.csv(knn.pred.cv.df, "knn.
pred.cv.csv", row.names=F)  # write to CSV
```

Best Subset

The following code notes the best predictors for 1, 2, and 3 predictors.

```
train.mat.full <- cbind(train.mat, train$SalePrice)  # make a model matrix with the response
bst.reg <- regsubsets(V277~., data=as.data.frame(train.mat.full), nvmax=3, method="exhaustive",
 really.big=T)  # fit best subsets
```

```
Reordering variables and trying again:
```

```
names(coef(bst.reg, id=3))  # extract variable names
```

```
[1] "(Intercept)"       "`RoofMatlTar&Grv`" "BedroomAbvGr"
[4] "FireplaceQuPo"
```

```
names(coef(bst.reg, id=2))
```

```
[1] "(Intercept)"       "`RoofMatlTar&Grv`" "FireplaceQuPo"
```

```
names(coef(bst.reg, id=1))
```

```
[1] "(Intercept)"   "FireplaceQuPo"
```

The following code tests different numbers of neighbors to find the one that yields the lowest estimated error using 5-fold cross-validation using 1, 2, and 3 best subset predictors.

```r
K.max <- 100  # maximum possible number of neighbors within reasonable computation time

set.seed(1)  # consistency of k-fold validation breaks
fold.index <- cut(sample(1:nrow(train.mat)), breaks=5, labels=FALSE)  # split data into 5 folds

knn.rmsleki.1 <- matrix(NA, ncol=K.max, nrow=5); knn.rmsleki.2 <- matrix(NA, ncol=K.max, nrow=5
); knn.rmsleki.3 <- matrix(NA, ncol=K.max, nrow=5)  # initialize storage of the k RMSLE's
for (j in 1:5) {
  train.x <- train.mat[fold.index != j,]  # fold training set
  train.x.1 <- as.data.frame(train.x[,215])
  train.x.2 <- train.x[,c(225,106)]
  train.x.3 <- train.x[,c(225,106,199)]
  train.y <- train$SalePrice[fold.index != j]  # fold training response
  test.x <- train.mat[fold.index == j,]  # fold test set
  test.x.1 <- as.data.frame(test.x[,215])
  test.x.2 <- test.x[,c(225,106)]
  test.x.3 <- test.x[,c(225,106,199)]
  true.y <- train$SalePrice[fold.index == j]  # fold test response

  for (Ki in 1:K.max) {
    knn.predk.1 <- knn.reg(train=train.x.1, test=test.x.1, y=train.y, k=Ki)$pred  # make KNN pre
dictions using Ki neighbors
    knn.rmsleki.1[j, Ki] <- rmsle(actual=true.y, predicted=knn.predk.1)  # calculate estimated R
MSLE
    knn.predk.2 <- knn.reg(train=train.x.2, test=test.x.2, y=train.y, k=Ki)$pred  # make KNN pre
dictions using Ki neighbors
    knn.rmsleki.2[j, Ki] <- rmsle(actual=true.y, predicted=knn.predk.2)  # calculate estimated R
MSLE
    knn.predk.3 <- knn.reg(train=train.x.3, test=test.x.3, y=train.y, k=Ki)$pred  # make KNN pre
dictions using Ki neighbors
    knn.rmsleki.3[j, Ki] <- rmsle(actual=true.y, predicted=knn.predk.3)  # calculate estimated R
MSLE
  }
}

knn.rmslei.1 <- colMeans(knn.rmsleki.1); knn.rmslei.2 <- colMeans(knn.rmsleki.2); knn.rmslei.3 <
- colMeans(knn.rmsleki.3)  # calculate the average RMSLE for each possible number of neighbors

knn.id.1 <- which(knn.rmslei.1==min(knn.rmslei.1)); knn.id.2 <- which(knn.rmslei.2==min(knn.rmsl
ei.2)); knn.id.3 <- which(knn.rmslei.3==min(knn.rmslei.3))  # extract right number of neighbors

knn.pred.1 <- knn.reg(train=train.mat, test=test.mat, y=train$SalePrice, k=knn.id.1)$pred  # mak
e prediction
knn.pred.1.df <- data.frame(Id=test$Id, SalePrice=knn.pred.1); write.csv(knn.pred.1.df, "knn.pre
d.1.csv", row.names=F)  # write to CSV
knn.pred.2 <- knn.reg(train=train.mat, test=test.mat, y=train$SalePrice, k=knn.id.2)$pred  # mak
e prediction
knn.pred.2.df <- data.frame(Id=test$Id, SalePrice=knn.pred.2); write.csv(knn.pred.2.df, "knn.pre
d.2.csv", row.names=F)  # write to CSV
knn.pred.3 <- knn.reg(train=train.mat, test=test.mat, y=train$SalePrice, k=knn.id.3)$pred  # mak
e prediction
knn.pred.3.df <- data.frame(Id=test$Id, SalePrice=knn.pred.3); write.csv(knn.pred.3.df, "knn.pre
d.3.csv", row.names=F)  # write to CSV
```

It turns out this was a poor idea, but it was worth a shot.

Summary of Results

Method	**K**	**Estimated RMSLE**	**Actual RMSLE**	**Notes**
KNN Regression	6	0.22252	0.38399	Using all predictors
KNN Regression	14	0.39730	0.37080	Using FireplaceQuPo
KNN Regression	16	0.39528	0.36765	Using FireplaceQuPo, RoofMatlTar&Grv
KNN Regression	100	0.39357	0.35547	Using FireplaceQuPo, RoofMatlTar&Grv, BedroomAbvGr

Conclusion

Method	Est. RMSLE	Actual RMSLE	Notes
Forward stepwise	0.14826	0.29415	125 dimension; selected using adjusted R^2
Forward stepwise	0.18468	0.22928	71 dimension; selected using Mallow's Cp
Forward stepwise	0.24152	0.22727	39 dimension; selected using Bayes' IC
Forward stepwise	0.14963	0.22282	199 dimension; selected using cross validation
Backward stepwise	0.14081	0.21879	133 dimension; selected using adjusted R^2
Backward stepwise	0.16272	0.21734	87 dimension; selected using Mallow's Cp
Backward stepwise	0.20075	0.23554	43 dimension; selected using Bayes' IC
Backward stepwise	0.14970	0.22234	184 dimension; selected using cross validation
Mixed stepwise	0.15639	0.27081	99 dimension; selected using adjusted R^2
Mixed stepwise	0.17950	0.20876	41 dimension; selected using Mallow's Cp
Mixed stepwise	0.20814	0.23318	79 dimension; selected using Bayes' IC
Mixed stepwise	0.20407	0.22880	71 dimension; selected using cross validation
Ridge regression	0.13922	0.18548	Lambda=15921.10313
Lasso	0.13168	0.19808	Lambda=376.45280
Principal components regression	0.55844	0.19129	161 components
Partial least squares	0.56133	0.18627	26 components
K-nearest neighbors	0.22252	0.38399	K=6; using all predictors
K-nearest neighbors	0.39730	0.37080	K=14; best predictor
K-nearest neighbors	0.39528	0.36765	K=16; best 2 predictors
K-nearest neighbors	0.39357	0.35547	K=100; best 3 predictors

The following table is sorted by estimated test error. Recall that all estimated test errors were calculated using 5-fold cross validation.

Method	Est. RMSLE	Actual RMSLE	Notes
Lasso	0.13168	0.19808	Lambda=376.45280
Ridge regression	0.13922	0.18548	Lambda=15921.10313
Backward stepwise	0.14081	0.21879	133 dimension; selected using adjusted R^2

Method	Est. RMSLE	Actual RMSLE	Notes
Forward stepwise	0.14826	0.29415	125 dimension; selected using adjusted R^2
Forward stepwise	0.14963	0.22282	199 dimension; selected using cross validation
Backward stepwise	0.14970	0.22234	184 dimension; selected using cross validation
Mixed stepwise	0.15639	0.27081	99 dimension; selected using adjusted R^2
Backward stepwise	0.16272	0.21734	87 dimension; selected using Mallow’s Cp
Mixed stepwise	0.17950	0.20876	41 dimension; selected using Mallow’s Cp
Forward stepwise	0.18468	0.22928	71 dimension; selected using Mallow’s Cp
Backward stepwise	0.20075	0.23554	43 dimension; selected using Bayes’ IC
Mixed stepwise	0.20407	0.22880	71 dimension; selected using cross validation
Mixed stepwise	0.20814	0.23318	79 dimension; selected using Bayes’ IC
K-nearest neighbors	0.22252	0.38399	K=6; using all predictors
Forward stepwise	0.24152	0.22727	39 dimension; selected using Bayes’ IC
K-nearest neighbors	0.39357	0.35547	K=100; best 3 predictors
K-nearest neighbors	0.39528	0.36765	K=16; best 2 predictors
K-nearest neighbors	0.39730	0.37080	K=14; best predictor
Principal components regression	0.55844	0.19129	161 components
Partial least squares	0.56133	0.18627	26 components

The following table is sorted by actual test error.

Method	Est. RMSLE	Actual RMSLE	Notes
Ridge regression	0.13922	0.18548	Lambda=15921.10313
Partial least squares	0.56133	0.18627	26 components
Principal components regression	0.55844	0.19129	161 components
Lasso	0.13168	0.19808	Lambda=376.45280
Mixed stepwise	0.17950	0.20876	41 dimension; selected using Mallow’s Cp
Backward stepwise	0.16272	0.21734	87 dimension; selected using Mallow’s Cp
Backward stepwise	0.14081	0.21879	133 dimension; selected using adjusted R^2
Backward stepwise	0.14970	0.22234	184 dimension; selected using cross validation
Forward stepwise	0.14963	0.22282	199 dimension; selected using cross validation

Method	Est. RMSLE	Actual RMSLE	Notes
Forward stepwise	0.24152	0.22727	39 dimension; selected using Bayes' IC
Mixed stepwise	0.20407	0.22880	71 dimension; selected using cross validation
Forward stepwise	0.18468	0.22928	71 dimension; selected using Mallow's Cp
Mixed stepwise	0.20814	0.23318	79 dimension; selected using Bayes' IC
Backward stepwise	0.20075	0.23554	43 dimension; selected using Bayes' IC
Mixed stepwise	0.15639	0.27081	99 dimension; selected using adjusted R^2
Forward stepwise	0.14826	0.29415	125 dimension; selected using adjusted R^2
K-nearest neighbors	0.39357	0.35547	K=100; best 3 predictors
K-nearest neighbors	0.39528	0.36765	K=16; best 2 predictors
K-nearest neighbors	0.39730	0.37080	K=14; best predictor
K-nearest neighbors	0.22252	0.38399	K=6; using all predictors

Shrinkage

The Lasso method had the lowest estimated test error and the 4th lowest actual test error. This model wasn't as heavily penalized as the ridge method (it had a lower λ), meaning it had fewer coefficients that were equal to or near zero. The ridge regression had the second lowest estimated test error and the lowest actual test error. It was much more heavily penalized than the Lasso, but because its coefficients never reach zero, it was still a high dimensional model. Overall, the shrinkage methods performed the best in terms of estimated test error and best or second-best in terms of actual test error. In the case where I wouldn't be able to see the true test error, I would be highly inclined to pick either of these models for their interpretability and low estimated test error.

Subset Selection

The subset selection methods all performed moderately well in terms of both estimated and actual test error. It is notable that the higher dimensional methods had lower estimated test errors but there isn't a very clear pattern between dimensionality and actual test error. This method performed the second best in terms of estimated test error and the third best in terms of actual test error. These methods have good interpretability and are easy to run, making them good contenders as final models.

Dimension Reduction

The dimension reduction methods performed very poorly in terms of estimated test error but were the 2nd and 3rd best in terms of actual test error; I am not sure why this happened. Dimension reduction can be hard to interpret and they showed poor estimated errors, so I would be disinclined to use these methods as a final model if I didn't know that their true test errors were so good.

K-Nearest Neighbors

While the estimated test errors for the KNN methods were mediocre, their actual test errors were abysmal; for the K=6 case with all predictors, this is expected because $p > 4$. However, even with 1, 2, or 3 predictors, this method still performed poorly.