Machine Learning Homework 1

Adib Ayay, Jarnickae Wilson, Karl Jiang, Chase Packard

January 21, 2017

Question 1

General Functions)

Since the first questions seem to involve doing the same process several times over, I implement these general functions below:

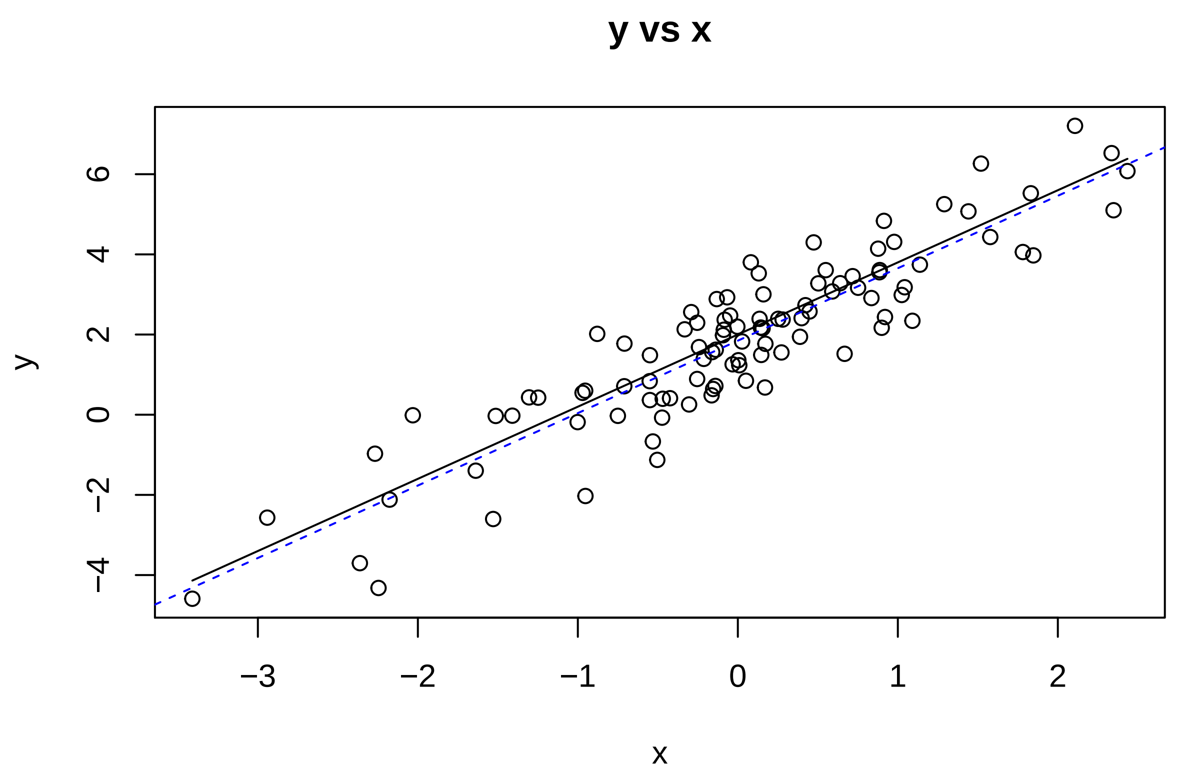
```r
#set randomizer seed
set.seed(77)
#Load library
library(kknn)

plot_regression <- function(f, mdl, train){
  plot(train$y ~ train$x, main = "y vs x", xlab = "x", ylab = "y")
  curve(f, add = TRUE)
  abline(mdl, col = "blue", lty = 2)
}

plot_knn <- function(f, train){
  test <- data.frame(x = sort(train$x))

  knn_2 <- kknn(y ~ x, train, test, k = 2, kernel = "rectangular")
  plot(train$y ~ train$x, main = "KNN, k = 2", xlab = "x", ylab = "y")
  lines(test$x, knn_2$fitted.values, col = "red")
  curve(f, add = TRUE)

  knn_12 <- kknn(y ~ x, train, test, k = 12, kernel = "rectangular")
  plot(train$y ~ train$x, main = "KNN, k = 12", xlab = "x", ylab = "y")
  lines(test$x, knn_12$fitted.values, col = "blue")
  curve(f, add = TRUE)
}

plot_mse <- function(train, test, mdl_tr, p = -1){
  outMSE <- c()
  kvec <- 2:15
  for(k in kvec){
    near = kknn(y ~ ., train, test, k = k, kernel = "rectangular")
    MSE = mean((test$y - near$fitted)^2)
    outMSE <- c(outMSE, MSE)
  }
  #Regression MSE (test)
  y_reg_pred <- predict(mdl_tr, newdata = test)
  mse_regr <- mean((test$y - y_reg_pred)^2)

  title = "log(1/k) MSE"
  if(p > -1){title <- paste(title, "Sine Disturbance", p)}
  plot(outMSE ~ log(1/kvec), main = title)
  abline(a = mse_regr, b = 0, col = "red", lwd = 4)
  imin = which.min(outMSE)
  cat("best k is ",kvec[imin],"\n")
  cat("Regression MSE:", mse_regr)
}

steps_1_5 <- function(f, train, test){
  mdl_tr <- lm(y ~ x, data = train)
  plot_regression(f, mdl_tr, train)
  plot_knn(f, train)
  plot_mse(train, test, mdl_tr)
}
```

1: Generating Data)

```r
f <- function(x){return(1.8*x + 2)}
x_train <- rnorm(100)
y_train <- f(x_train) + rnorm(100)

x_test <- rnorm(10000)
y_test <- f(x_test) + rnorm(10000)

train <- data.frame(x = x_train, y = y_train)
test <- data.frame(x = x_test, y = y_test)
```

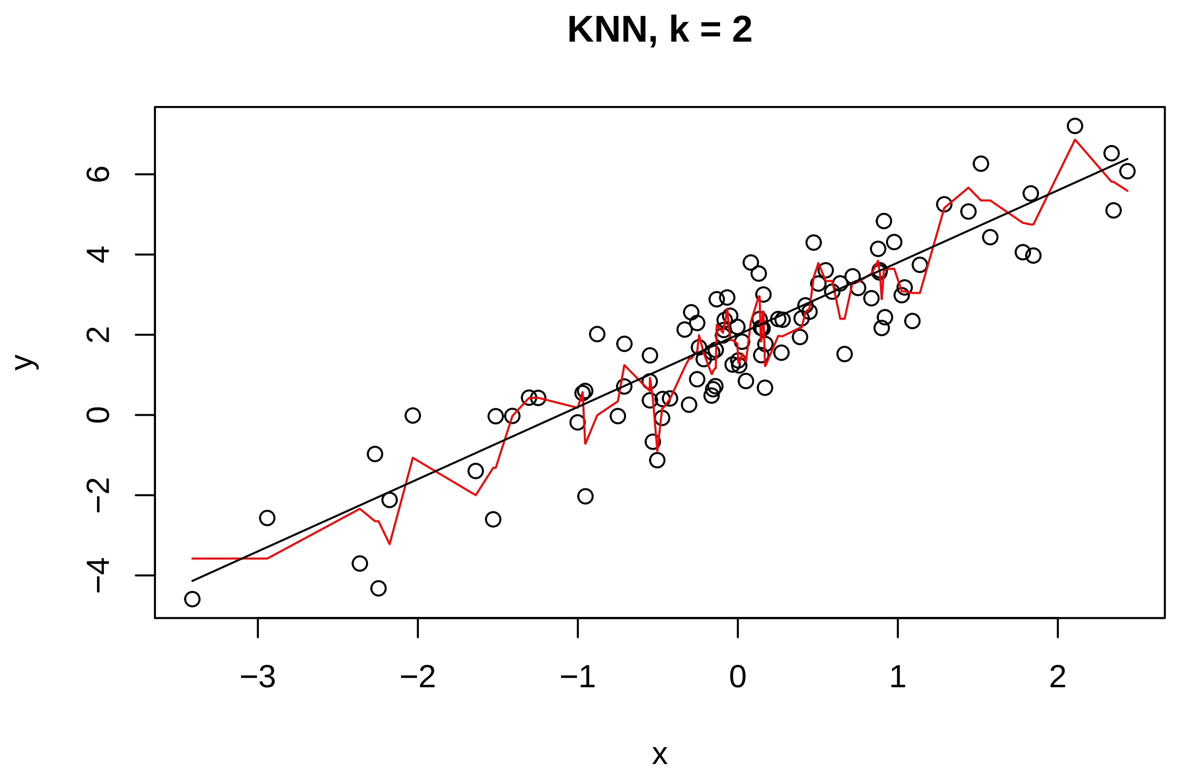
2: Scatterplot and 3) Regression (Train)

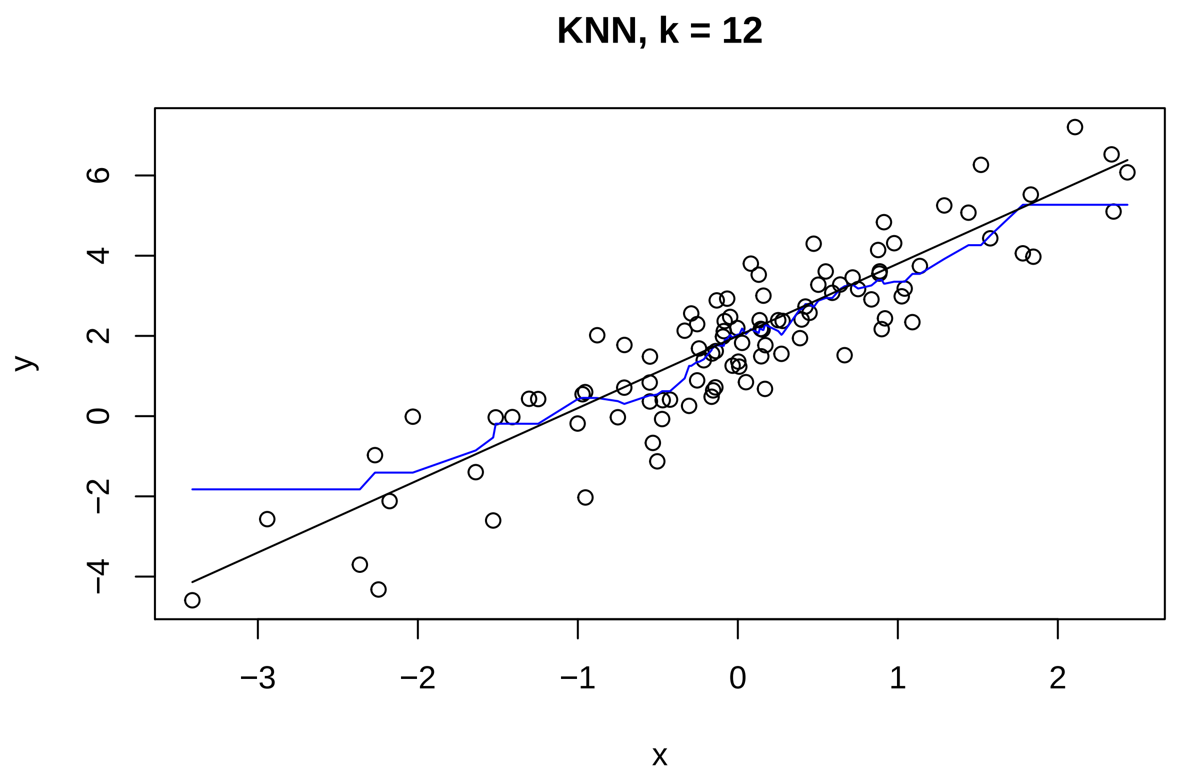
```r
mdl_tr <- lm(y ~ x, data = train)
plot_regression(f, mdl_tr, train)
```



4) KNN

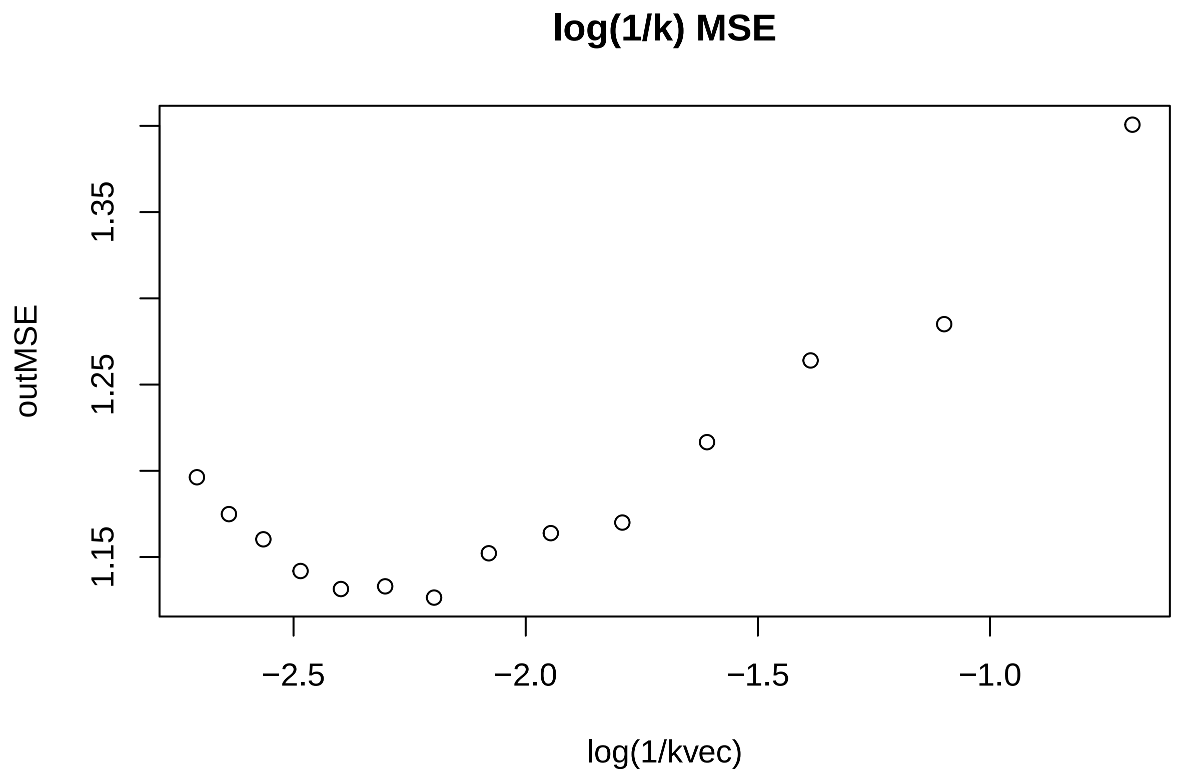
```r
plot_knn(f, train)
```


KNN, k = 2
y
x
6
4
2
0
−2
−4
−3
−2
−1
0
1
2


KNN, k = 12
y
x
6
4
2
0
−2
−4
−3
−2
−1
0
1
2

5) MSE, Performance

```r
plot_mse(train, test, mdl_tr)
```



```
## best k is  9
## Regression MSE: 1.031901
```

The model that performs the best is KNN for k = 8. Linear Regression's out of sample l2 loss was so low that it didn't even show up on the graph, that is, it performed bettr than KNN for all values of k from [2, 15]. Makes sense since the true function is linear.

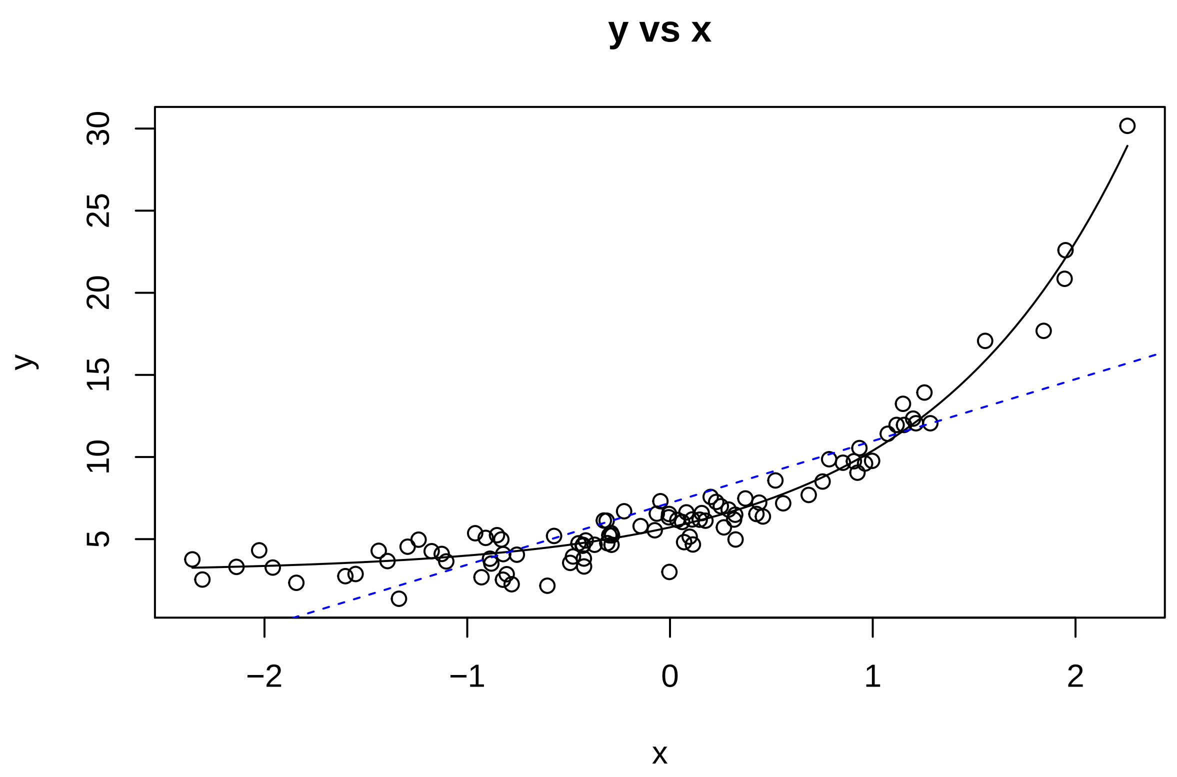
6) Exponential

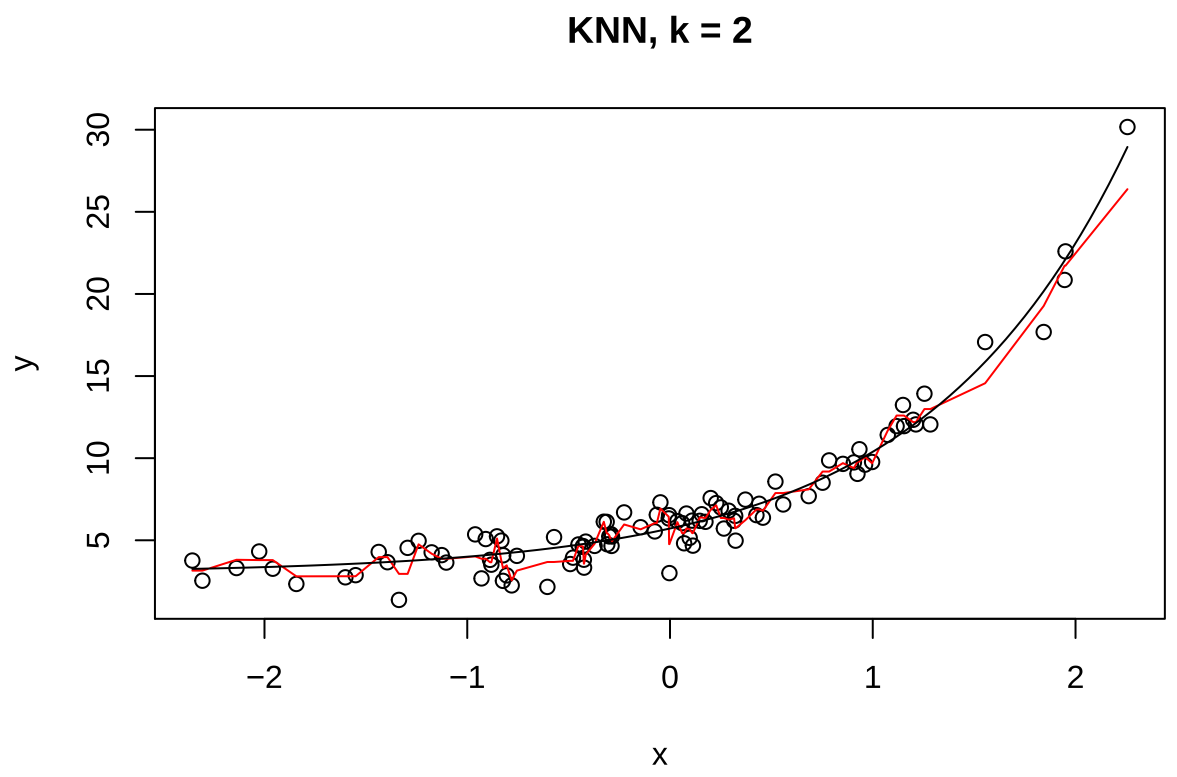
a) Generate Data

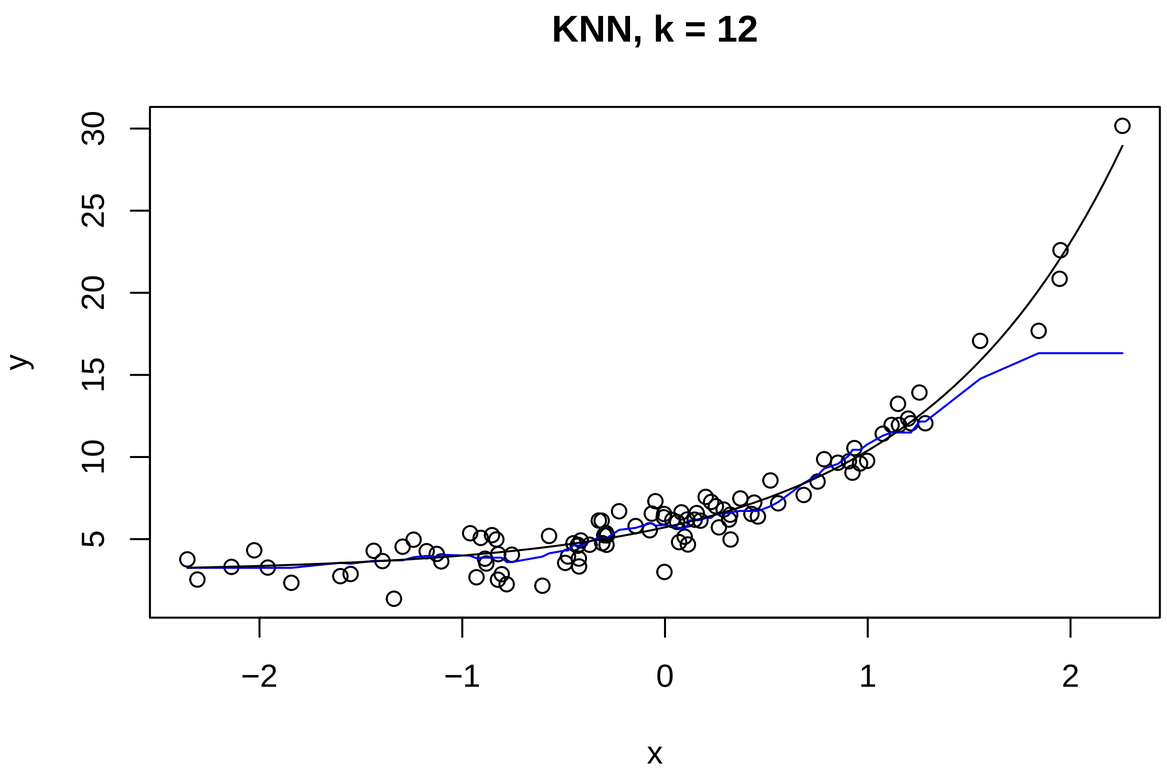
```r
f_exp <- function(x){return(exp(x + 1) + 3)}
x_train <- rnorm(100)
y_train <- f_exp(x_train) + rnorm(100)

x_test <- rnorm(10000)
y_test <- f_exp(x_test) + rnorm(10000)

train <- data.frame(x = x_train, y = y_train)
test <- data.frame(x = x_test, y = y_test)
```

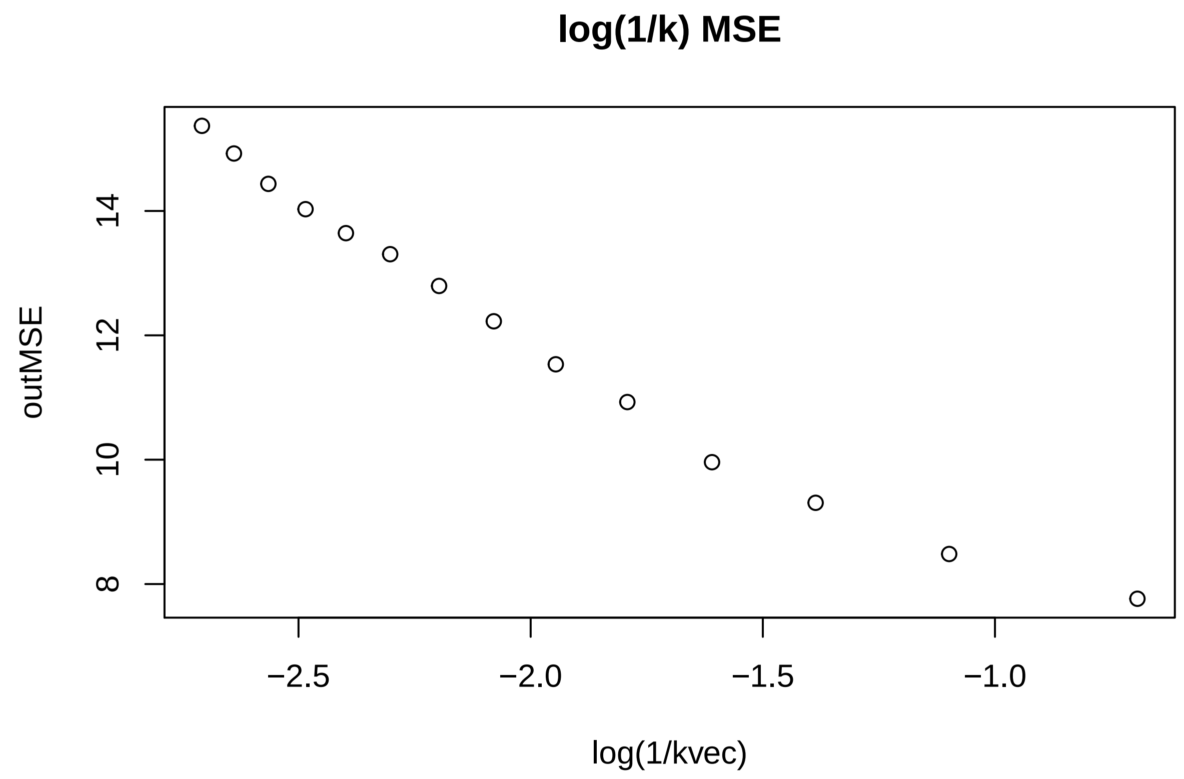
b) the other steps

```
steps_1_5(f = f_exp, train = train, test = test)
```


y vs x
y
x
30
25
20
15
10
5
−2
−1
0
1
2


KNN, k = 2
y
x
30
25
20
15
10
5
−2
−1
0
1
2


KNN, k = 12
y
x
30
25
20
15
10
5
−2
−1
0
1
2



```
## best k is  2
## Regression MSE: 16.12652
```

The best k is k = 2. Most likely came from the boundary bias since the function begins to skyrocket at the right tail. But to be honest, it's probably a stupid bug that I missed. The regression MSE doesn't show up on the plot this time because it was too high. Makes sense since the true function is not linear, which linear regression assumes.

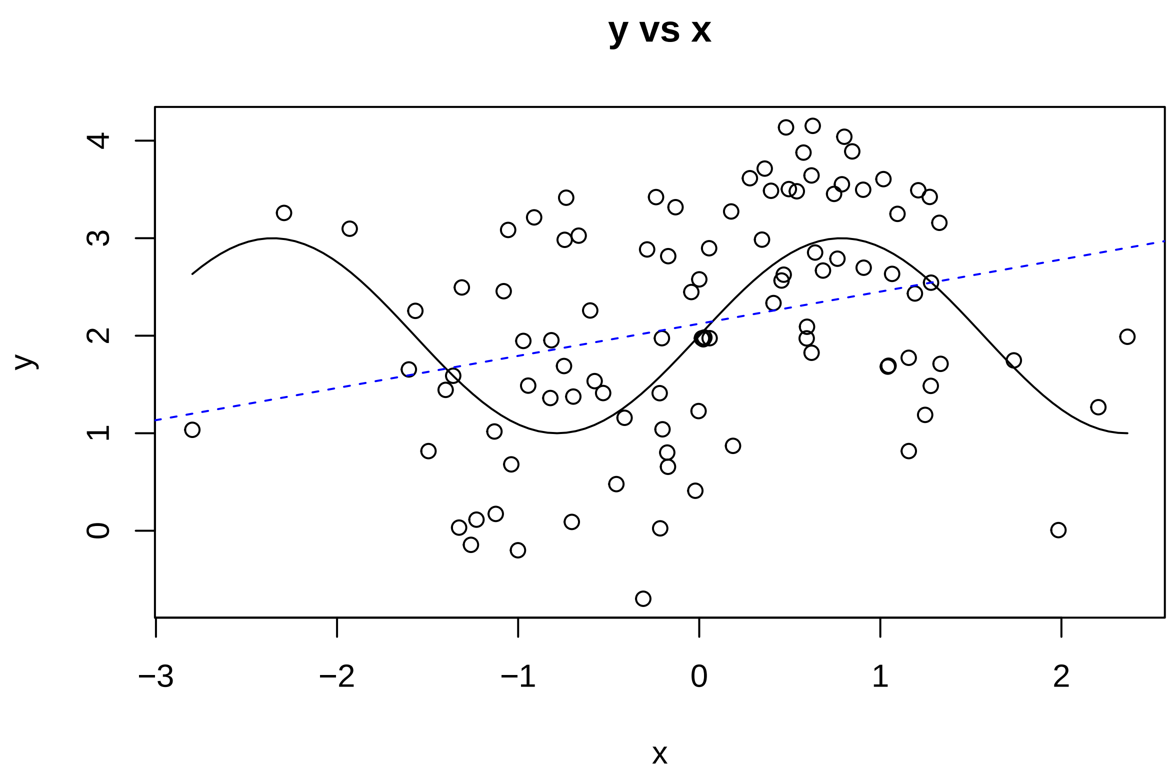
7) Sine

a) Generate Data

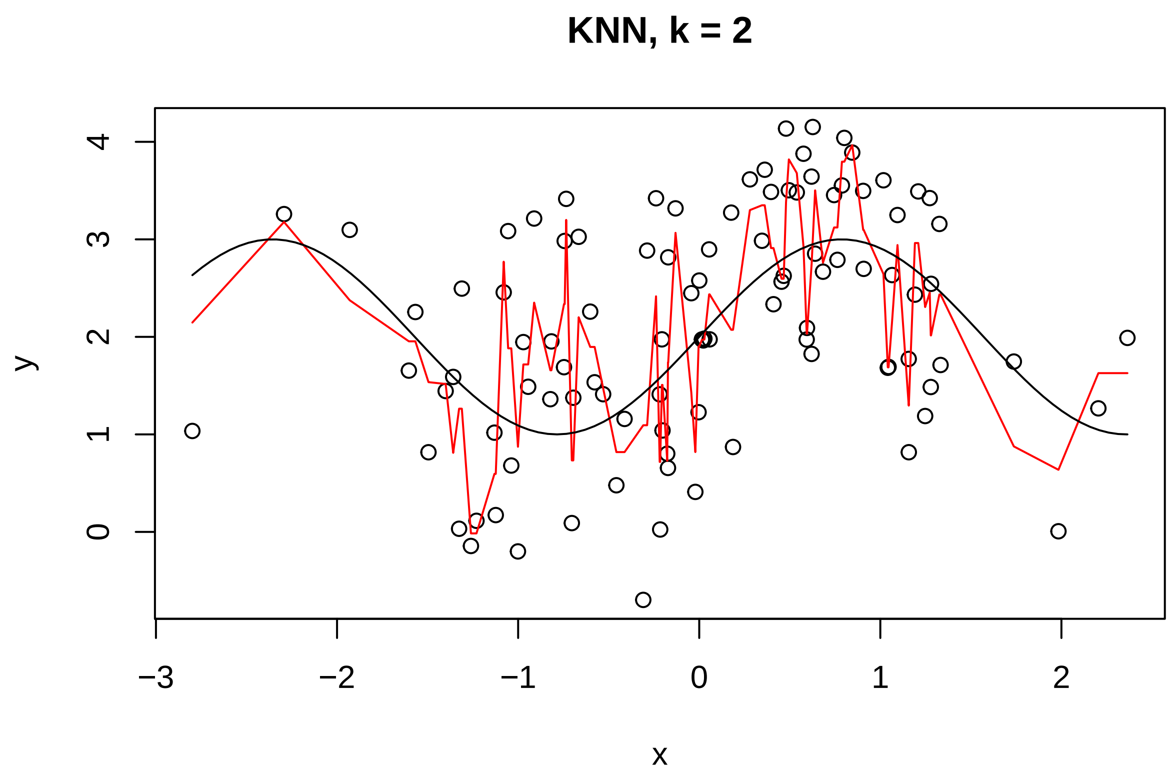
```r
f_sin <- function(x){return(sin(2*x) + 2)}
x_train <- rnorm(100)
y_train <- f_sin(x_train) + rnorm(100)

x_test <- rnorm(10000)
y_test <- f_sin(x_test) + rnorm(10000)

train <- data.frame(x = x_train, y = y_train)
test <- data.frame(x = x_test, y = y_test)
```

b) The rest

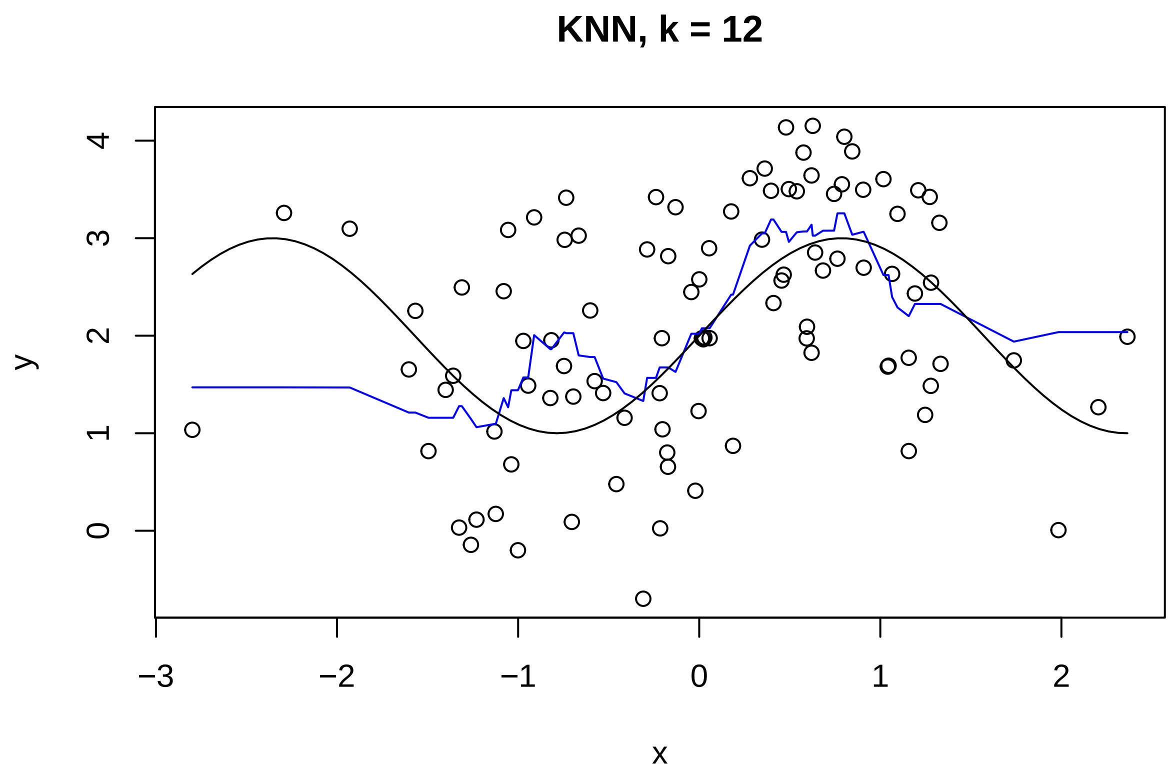
```
steps_1_5(f = f_sin, train = train, test = test)
```

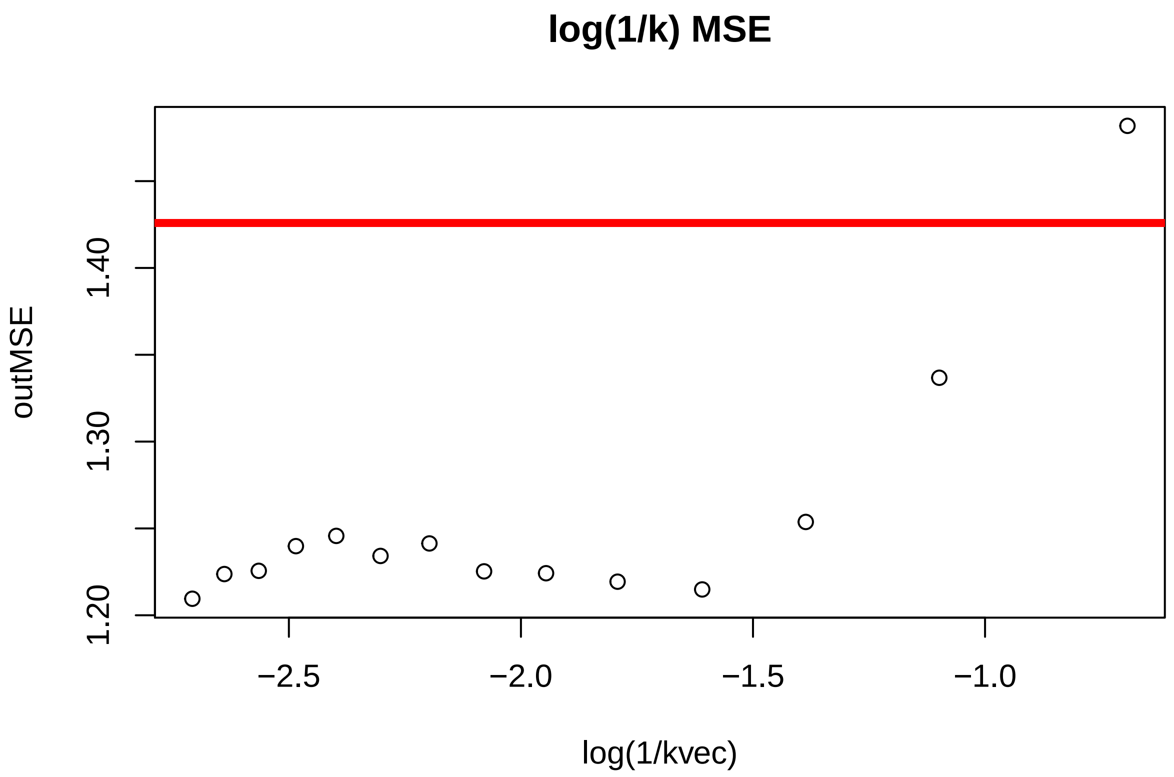


y vs x
y
x
−3
−2
−1
0
1
2
0
1
2
3
4



KNN, k = 2
y
x
−3
−2
−1
0
1
2
0
1
2
3
4


KNN, k = 12
y
x
−3
−2
−1
0
1
2
0
1
2
3
4



```
## best k is  15
## Regression MSE: 1.425892
```

Best k is 9 neighbors. Quite honestly, I expected exponential MSE from the linear regression to perform better than sine, but I guess I either made a typo somewhere / bug.

So forgetting about the bug, Regression does a lot worse in terms of out of sample performance.

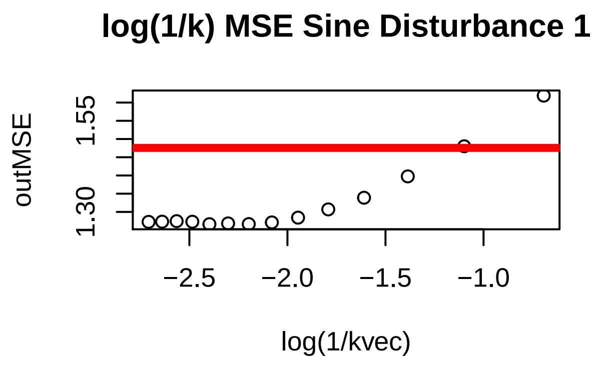
8 Disturbing the neighbors)

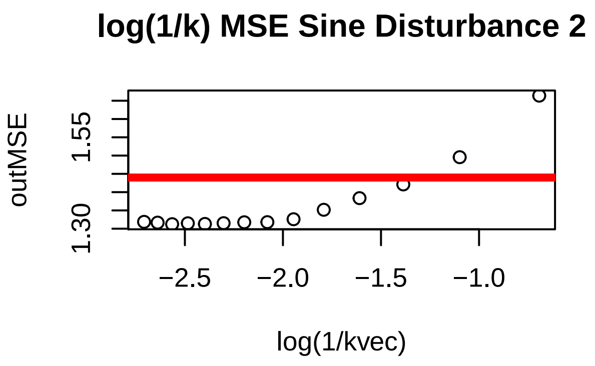
```
steps_psin <- function(p, train, test){
  p_vec <- 1:p
  x_p_train <- matrix(rnorm(100 * length(p_vec)), ncol = length(p_vec))
  colnames(x_p_train) <- p_vec
  x_p_test <- matrix(rnorm(10000 * length(p_vec)), ncol = length(p_vec))
  colnames(x_p_test) <- p_vec

  train_p <- data.frame(train, noise = x_p_train)
  test_p <- data.frame(test, noise = x_p_test)
  mdl_tr <- lm(y ~ ., data = train_p)
  plot_mse(train_p, test_p, mdl_tr, p)
}

par(mfrow = c(2,2))
for(p in 1:20){ steps_psin(p, train, test) }
```

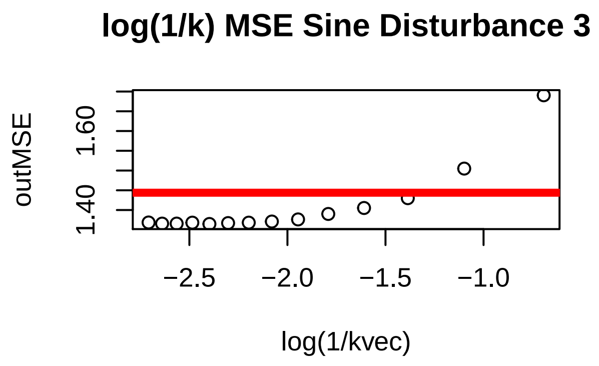
```
## best k is  11
## Regression MSE: 1.475742
```

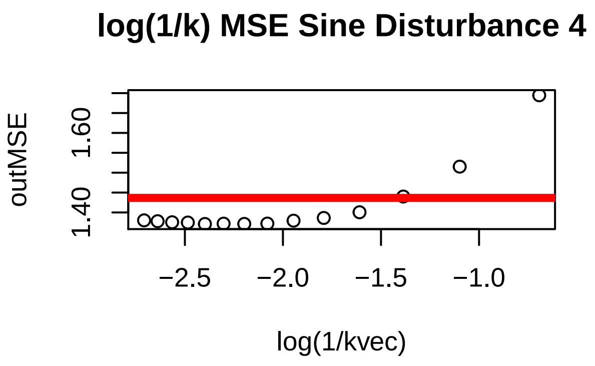
```
## best k is  13
## Regression MSE: 1.439931
```

```
## best k is  11
## Regression MSE: 1.44389
```







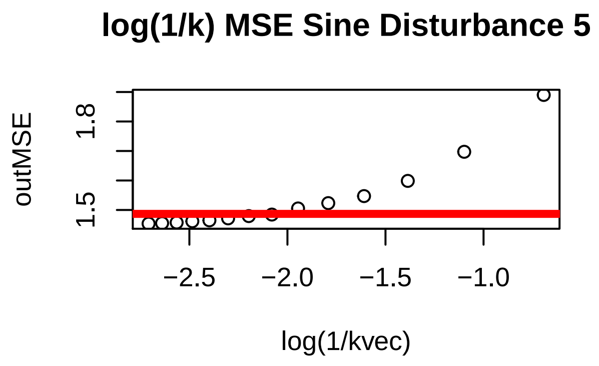


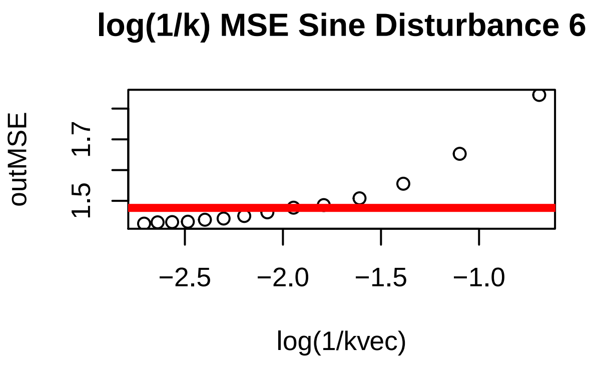
```
## best k is  11
## Regression MSE: 1.436705
```

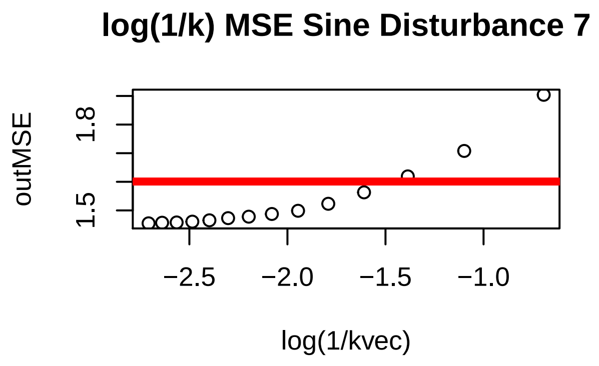
```
## best k is  15
## Regression MSE: 1.486772
```

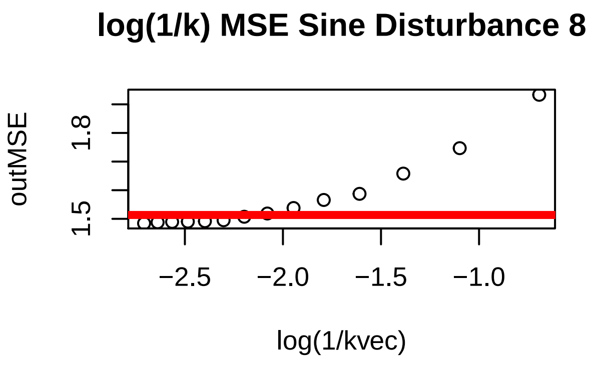
```
## best k is  15
## Regression MSE: 1.477026
```

```
## best k is  15
## Regression MSE: 1.600972
```







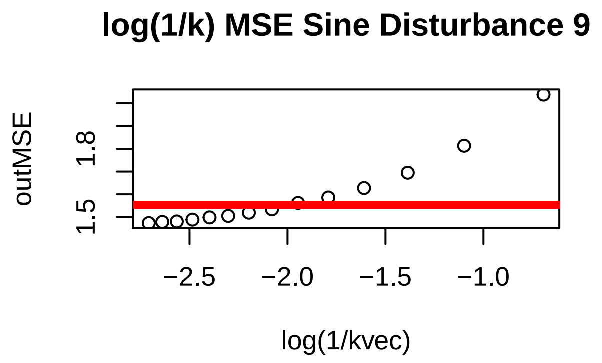


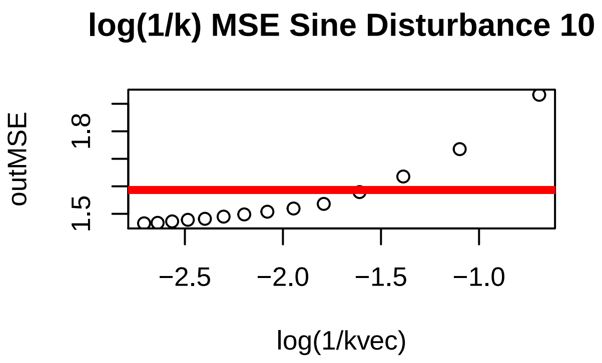
```
## best k is  15
## Regression MSE: 1.513489
```

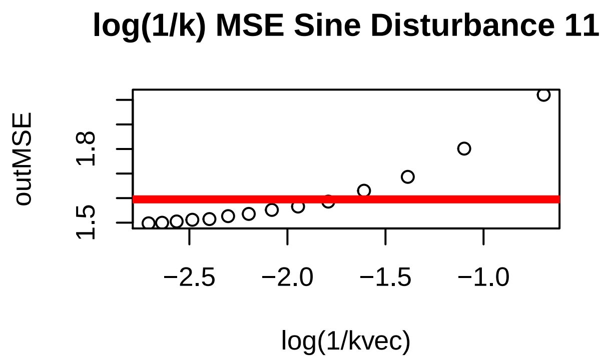
```
## best k is  15
## Regression MSE: 1.553671
```

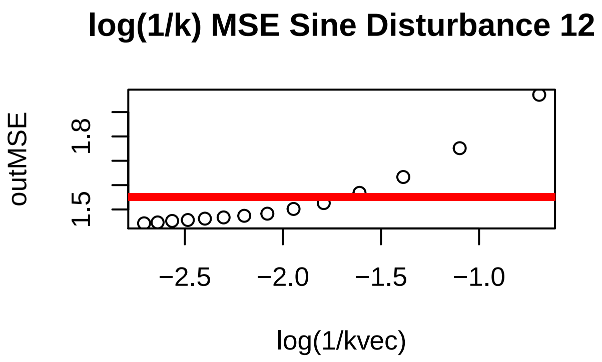
```
## best k is  15
## Regression MSE: 1.586664
```

```
## best k is  15
## Regression MSE: 1.594836
```







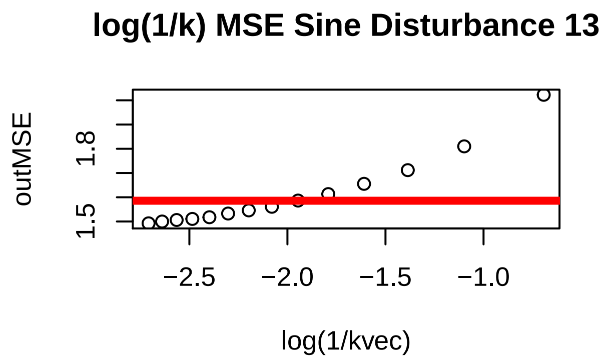


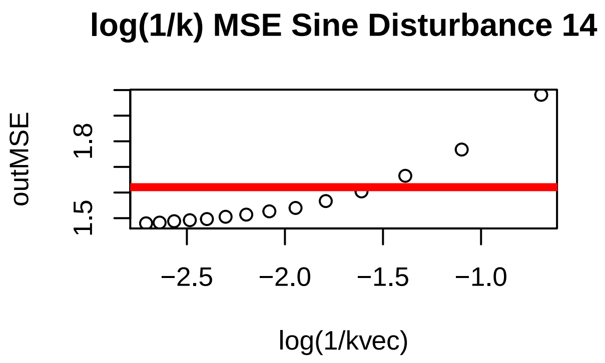
```
## best k is  15
## Regression MSE: 1.550939
```

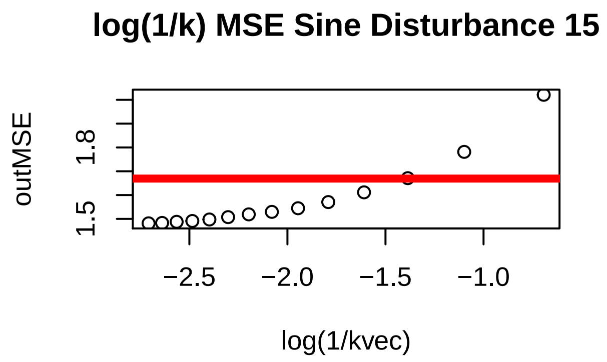
```
## best k is  15
## Regression MSE: 1.585595
```

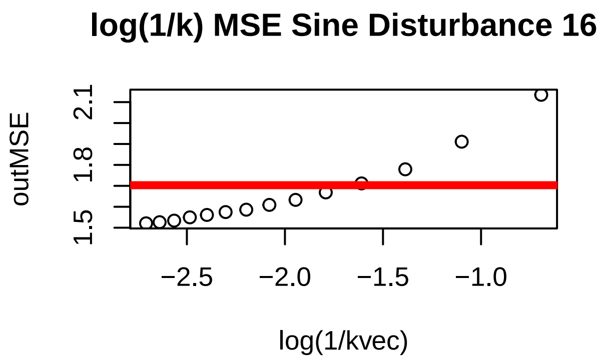
```
## best k is  15
## Regression MSE: 1.620908
```

```
## best k is  15
## Regression MSE: 1.669308
```







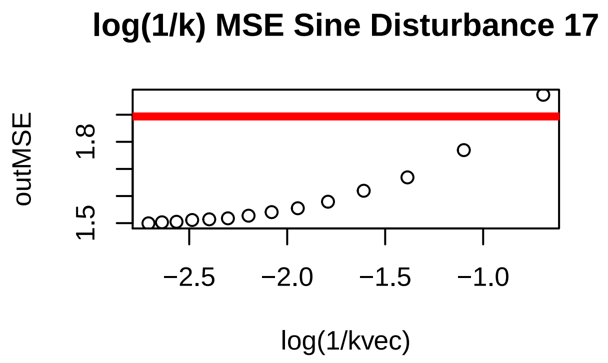


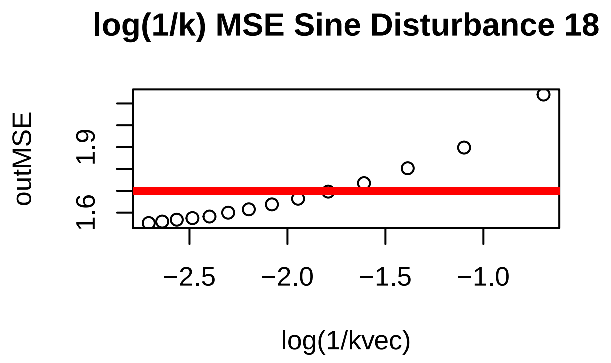
```
## best k is  15
## Regression MSE: 1.703058
```

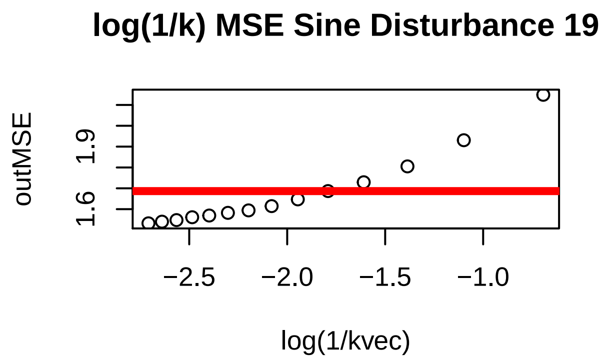
```
## best k is  15
## Regression MSE: 1.893659
```

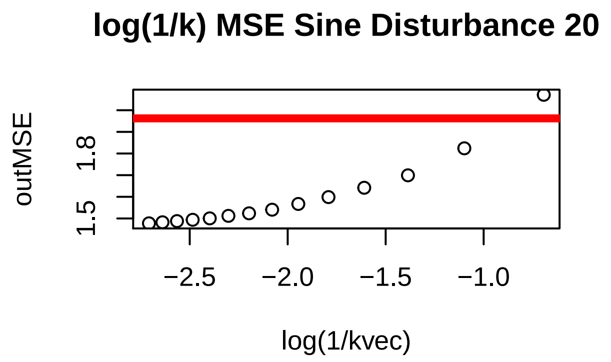
```
## best k is  15
## Regression MSE: 1.698973
```

```
## best k is  15
## Regression MSE: 1.686617
```









```
## best k is  15
## Regression MSE: 1.962412
```

Due to random variation, we sometimes see the regression MSE jump around, but for the most part, it's doing better and better against KNN. I hypothesize that we see this because now, meaningless variations in the euclidian distance (to find the nearest neighbors) are throwing off what the "true" neighbors should be. Also notice that we can no longer see a "bottom" to the MSE curve, which implies that the best #neighbors k may be > 15.

Bonus 1)

I would expect KNN to perform better relative to regression. Regression won't do any better with more data - the true line isn't linear. With more samples, we would expect our tuning parameter k to have a wider range of values (where it outperforms linear regression), since samples mean more information for our KNN model.

Question 2

```r
#load necessary libraries and functions
library(rpart)
library(caret)
```

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

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

```
##
## Attaching package: 'caret'

## The following object is masked from 'package:kknn':
##
##     contr.dummy
```

```r
download.file("https://raw.githubusercontent.com/ChicagoBoothML/HelpR/master/docv.R", "docv.R")
source("docv.R") #this has docvknn used below

#Load data and attach it
url<-'https://raw.githubusercontent.com/ChicagoBoothML/DATA___UsedCars/master/UsedCars.csv'
UsedCars<-read.csv(url)

attach(UsedCars)
#sort by increasing mileage
UsedCars<-UsedCars[order(mileage),]
```

1)

There are a number of possible applications of the Used Car data supplied:

1. A used car dealership can use the data to determine what a reasonable acquisition price is for a used vehicle given a target profit margin, and a description of the vehicle. This type of information would be crucial in a negotiation.
2. A car dealership can also use the data in combination with a predictive model to get a sense of how competitors are likely to price their products. This would allow the business to undercut competitors' prices if so desired.
3. Consumers often choose to enter into lease agreements with manufacturers for vehicles. In these cases, a manufacturer is responsible for reselling the vehicle after the lease term is over, and is therefore concerned about the determinants of a car's residual value. Using this data in conjunction with a predictive model may provide manufacturers with better information so that lease terms and installments will be appropriately determined.

2)

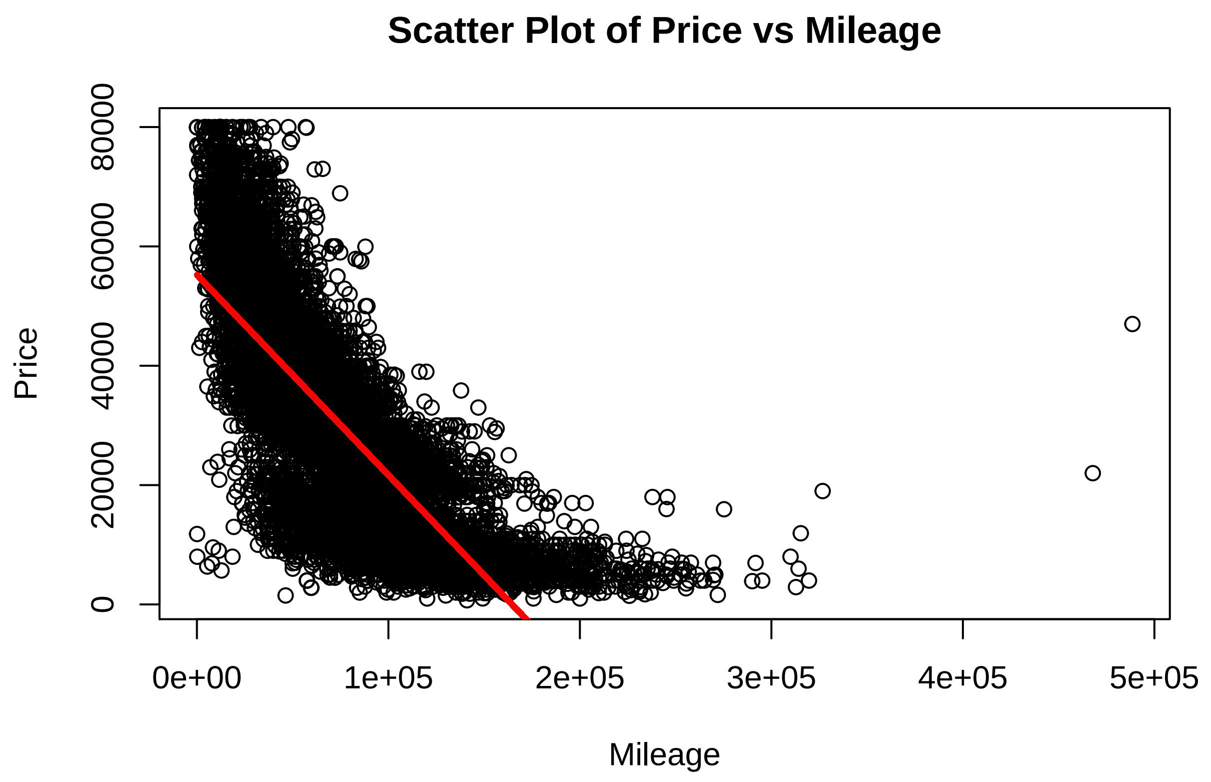
```r
#Splitting in-sample and out-of-sample data
nobserv = nrow(UsedCars) #number of observations
ntrain = nobserv * 0.75 #number of observations for training data
tr = sample(1:nobserv,ntrain) #sampling 75% observations
train = UsedCars[tr,] #training data
test = UsedCars[-tr,] #test data
detach(UsedCars)
```

3)

```r
attach(train)
fit <- lm(price~mileage)
summary(fit)
```

```
##
## Call:
## lm(formula = price ~ mileage)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -47176  -6840     14   6740 155758
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.524e+04  1.792e+02   308.2   <2e-16 ***
## mileage     -3.357e-01  2.112e-03  -158.9   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11160 on 15045 degrees of freedom
## Multiple R-squared:  0.6267, Adjusted R-squared:  0.6267
## F-statistic: 2.526e+04 on 1 and 15045 DF,  p-value: < 2.2e-16
```

```r
plot(mileage, price, xlab="Mileage", ylab="Price",
     main="Scatter Plot of Price vs Mileage")
lines(mileage,fit$fitted.values, col="red", lwd=3)
```

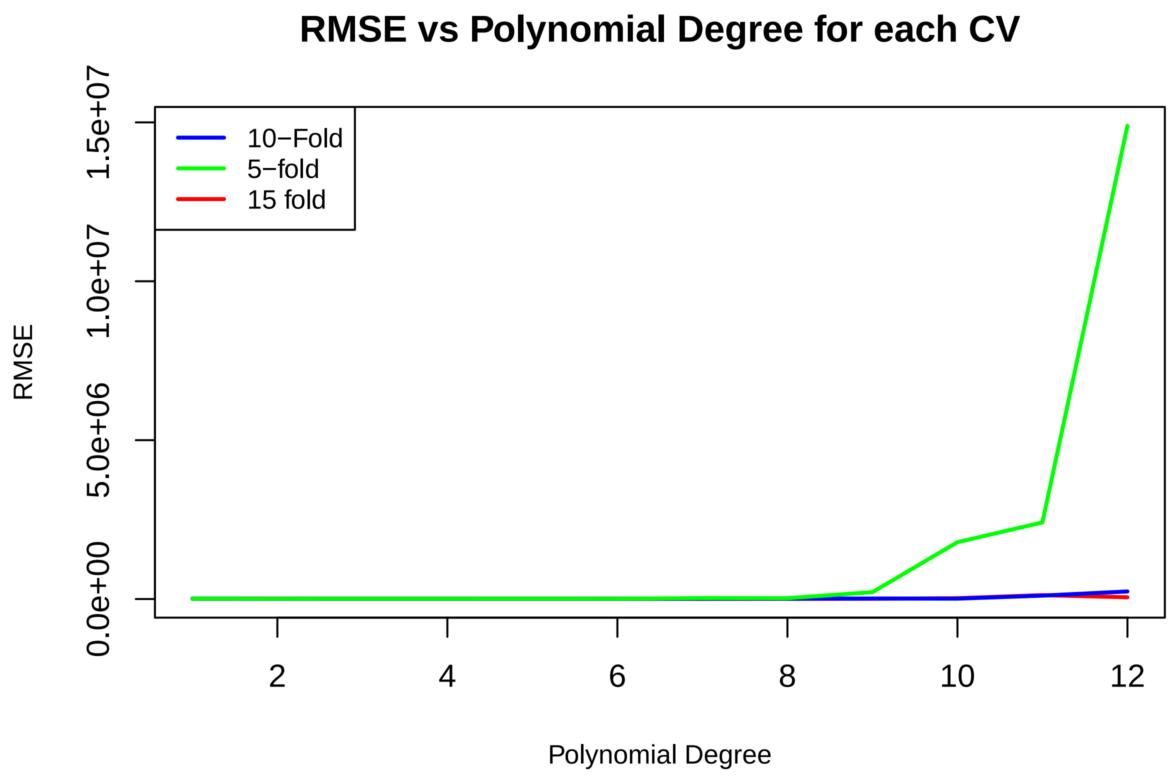


The relationship between Price and Mileage using ordinary linear regression (shown by red line) is:

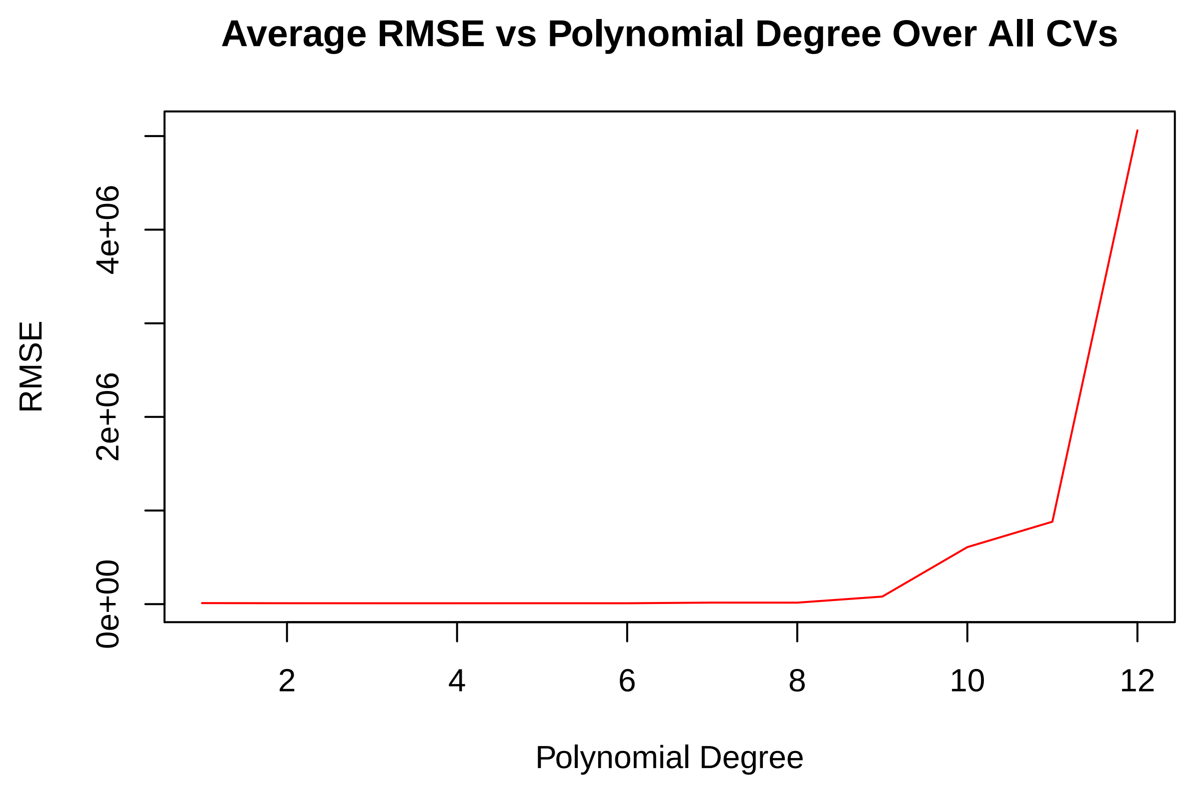
$$price = 5.5235004 \times 10^4 - -0.3357006 * mileage + e$$

4)

```r
#Polynomial Fit of the Data

numFolds <- c(15,10,5)
polyDegree <- c(1:12)
cv <- list()

#Cross Validation
for (n in 1:length(numFolds)) {
  folds <- createFolds(price, k = numFolds[n],list = TRUE, returnTrain = FALSE)
  cvmean=rep(0,length(polyDegree)) #store results for mean
  for (j in folds){
    testIndex <- j
    trainingTemp <- train[-testIndex,]
    testTemp <-  train[testIndex,]
    for (i in polyDegree){
      model <- lm(price ~ poly(mileage,i), data=trainingTemp)
      model_prediction <- predict(model,testTemp)
      residualFold <- (testTemp$price-model_prediction)^2
      msError <- sum(residualFold)
      cvmean[i] <- cvmean[i] + msError
    }
  }
  cv[n] <- list(sqrt(cvmean/length(price)))
}

#Plotting results for Cross Validation
rgy <- range(c(cv[[1]],cv[[2]],cv[[3]]))
plot(polyDegree,cv[[1]],type="l",col="red", main = "RMSE vs Polynomial Degree for each CV",
     ylim = rgy, lwd=2,cex.lab=0.8, xlab="Polynomial Degree", ylab="RMSE")
lines(polyDegree,cv[[2]],col="blue",lwd=2)
lines(polyDegree,cv[[3]],col="green",lwd=2)
legend("topleft",legend=c("10-Fold","5-fold","15 fold"),
       col=c("blue","green","red"),lwd=2,cex=0.8)
```



```
#Plotting results for average Cross Validations
cvFinal <- (cv[[1]]+cv[[2]]+cv[[3]])/3
plot(polyDegree, cvFinal, type="l",col="red",
     main = "Average RMSE vs Polynomial Degree Over All CVs",
     xlab="Polynomial Degree", ylab="RMSE")
```



```r
#Finding best degree
degreeBest <- polyDegree[which.min(cvFinal)]

cat('\n')
```

```r
cat('Optimal Degree is:',degreeBest, '\n')
```

```
## Optimal Degree is: 6
```

```r
#Retraining Using the Entire Training Set for Optimal Polynomial Degree
poly.fit <- lm(price ~ poly(mileage,5), data=train)
print(summary(poly.fit))
```

```
##
## Call:
## lm(formula = price ~ poly(mileage, 5), data = train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -62532  -5378    103   6104  46441
##
## Coefficients:
##                     Estimate Std. Error  t value Pr(>|t|)
## (Intercept)        3.069e+04  7.746e+01  396.185  < 2e-16 ***
## poly(mileage, 5)1 -1.773e+06  9.502e+03 -186.628  < 2e-16 ***
## poly(mileage, 5)2  6.789e+05  9.502e+03   71.441  < 2e-16 ***
## poly(mileage, 5)3 -2.231e+05  9.502e+03  -23.477  < 2e-16 ***
## poly(mileage, 5)4  5.092e+04  9.502e+03    5.359 8.49e-08 ***
## poly(mileage, 5)5  4.129e+04  9.502e+03    4.345 1.40e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9502 on 15041 degrees of freedom
## Multiple R-squared:  0.7293, Adjusted R-squared:  0.7293
## F-statistic:  8107 on 5 and 15041 DF,  p-value: < 2.2e-16
```

```
cat('Correlation between residuals and fitted values:',
    cor(poly.fit$fitted.values,poly.fit$residuals),'\n')
```

```
## Correlation between residuals and fitted values: -2.556859e-17
```

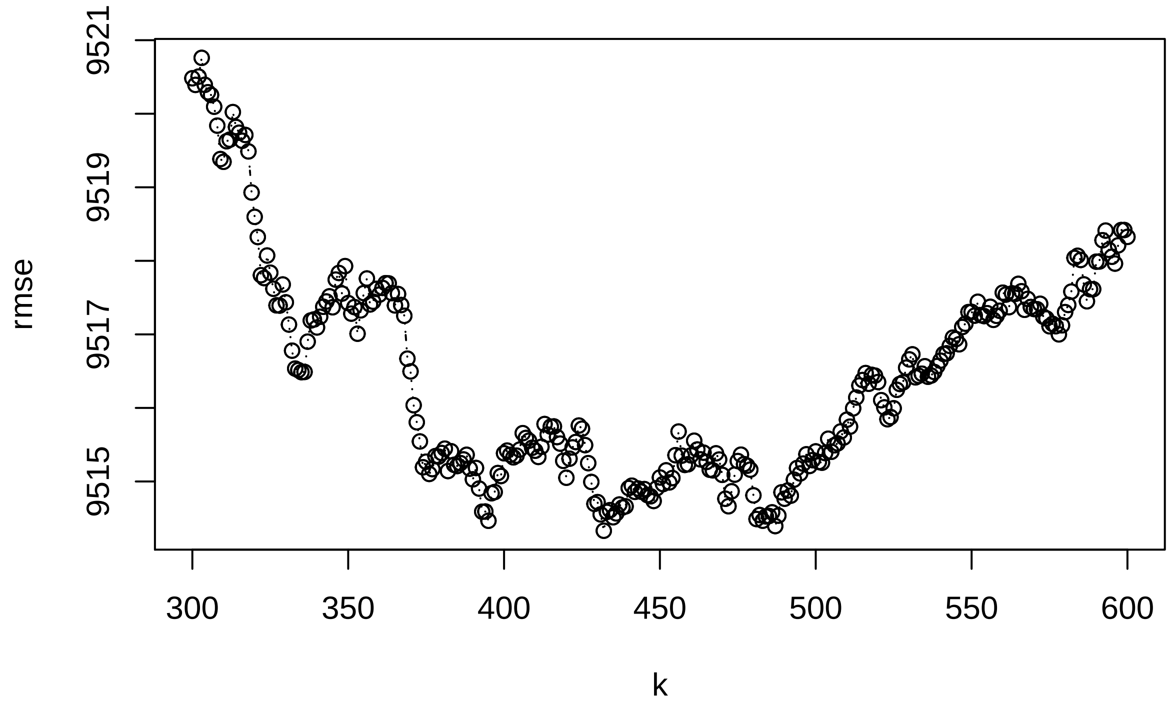
5)

k-NN: Selecting model by Cross Validation

```
CV_FOLDS <- 5 #folds of 5
N_CVS <- 1 #cross validations
k_range = 300 : 600
n=length(price) #number of observations
cv_mean = rep(0,length(k_range)) #store average rmse
cv_matrix = matrix(0,length(k_range),N_CVS) #keep results for each split
for(i in 1:N_CVS) {
    cv_rmse = docvknn(matrix(mileage,ncol=1),price,k_range,CV_FOLDS)
    cv_mean = cv_mean + cv_rmse
    cv_matrix[,i] = sqrt(cv_rmse/n)
}
```

```
## in docv: nset,n,nfold:  301 15047 5
## on fold:  1 , range:  1 : 3009
## on fold:  2 , range:  3010 : 6018
## on fold:  3 , range:  6019 : 9027
## on fold:  4 , range:  9028 : 12036
## on fold:  5 , range:  12037 : 15047
```

```
cv_mean = cv_mean/N_CVS
cv_mean = sqrt(cv_mean/n)
plot(k_range,cv_mean,xlab="k",ylab="rmse")
for(i in 1:N_CVS) lines(k_range,cv_matrix[,i],col=i,lty=3) #plot each result
lines(k_range,cv_mean,type="b",col="black") #plot average result
```



```
detach(train)

#get the min
kbest = k_range[which.min(cv_mean)]
cat("the best k is: ",kbest,"\n")
```

```
## the best k is:  432
```

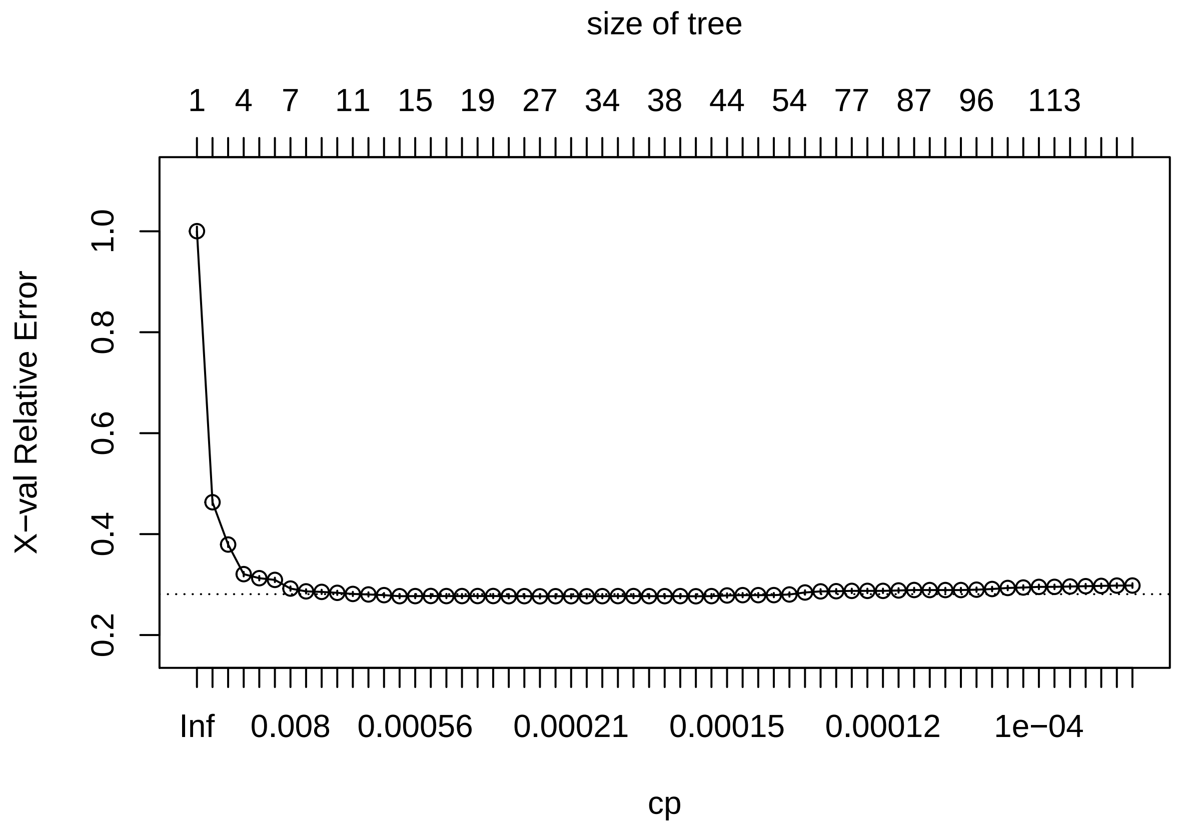
```
#Fit
kfbest = kknn(price~mileage,train,train,k=kbest,kernel = "rectangular") #get training fitted values
```

Regression tree: Selecting model by Cross Validation

```
car.tree = rpart(price~mileage, data=train,
                         control=rpart.control(minsplit=5,
                                               cp=0.0001,
                                               xval=10)
)

#Size big tree
nbig1 = length(unique(car.tree$where))
cat('Size of big tree: ',nbig1,'\n')
```

```
## Size of big tree:  133
```

```r
#Prunning
plotcp(car.tree)
```



```r
bestcp = car.tree$cptable[which.min(car.tree$cptable[, "xerror"]),
                                        "CP"]
best.car.tree <- prune(tree = car.tree,
                       cp = bestcp)
cat('cp corrsponding to the smallest value of xerror is: ',bestcp,'\n')
```

```
## cp corrsponding to the smallest value of xerror is:  0.0002218668
```

```r
#Size pruned tree
nsmall1 = length(unique( best.car.tree$where))
cat('Size of pruned tree: ',nsmall1,'\n')
```

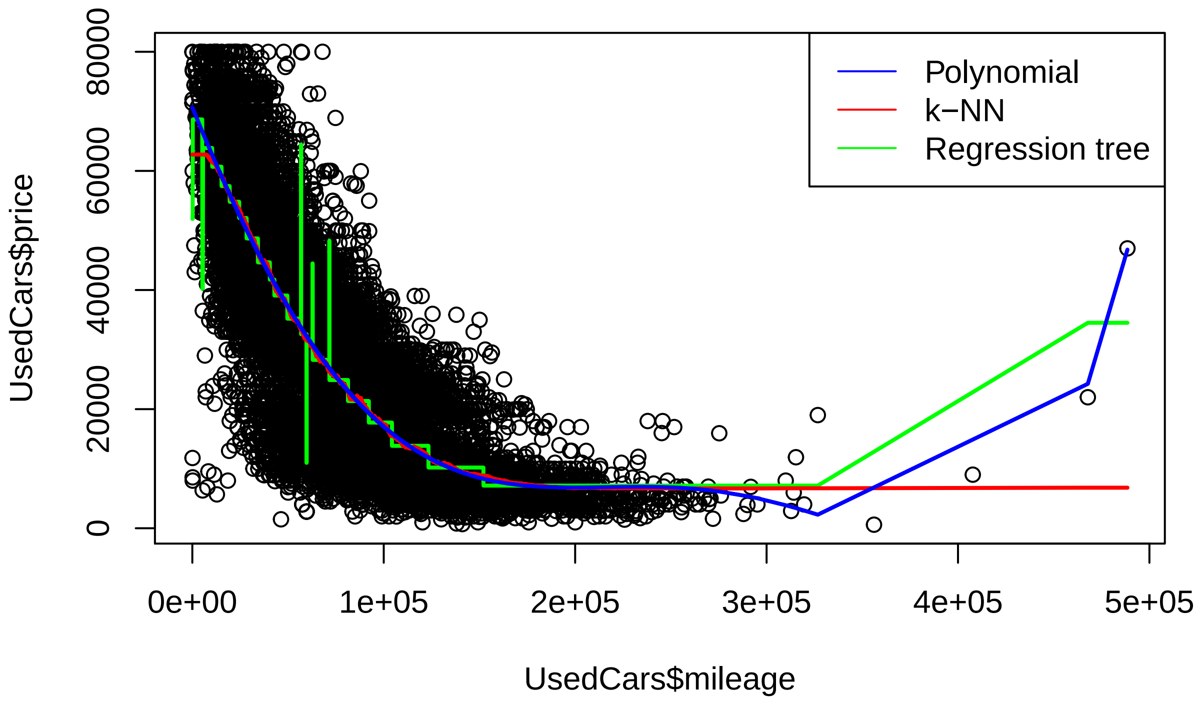
```
## Size of pruned tree:  27
```

```r
#FIT
car.fit = predict(best.car.tree,train) #get training fitted values
```

Plotting polynomial, k-NN and Regression tree

```r
plot(UsedCars$mileage,UsedCars$price,cex.lab=1) #plot price vs mileage for all data

#plot regression tree fit
oo=order(train$mileage)
```

```r
lines(train$mileage[oo],car.fit[oo],col="green",lwd=2) #step function fit

#plot k-NN
lines(train$mileage[oo],kfbest$fitted.values[oo],col="red", lwd=2)

#plot polynomial
lines(train$mileage[oo],predict(poly.fit,train)[oo], col = "blue", lwd = 2)

legend("topright",legend=c("Polynomial","k-NN","Regression tree"),
           col=c("blue","red","green"),lty=c(1,1,1))
```



k-NN model is most adequate

Out of the three models, the Regression tree, as expected, suffers from high variance.The polynomial and k-NN on the other hand seem to have very similar fits for Mileage upto the tail of the graph where the scarce number of observations with high mileage (+300k) unveils the overly flexible nature of the polynomial fit while the k-NN continues smoothly. For these reasons, I would choose the k-NN model to fit Price vs Mileage.

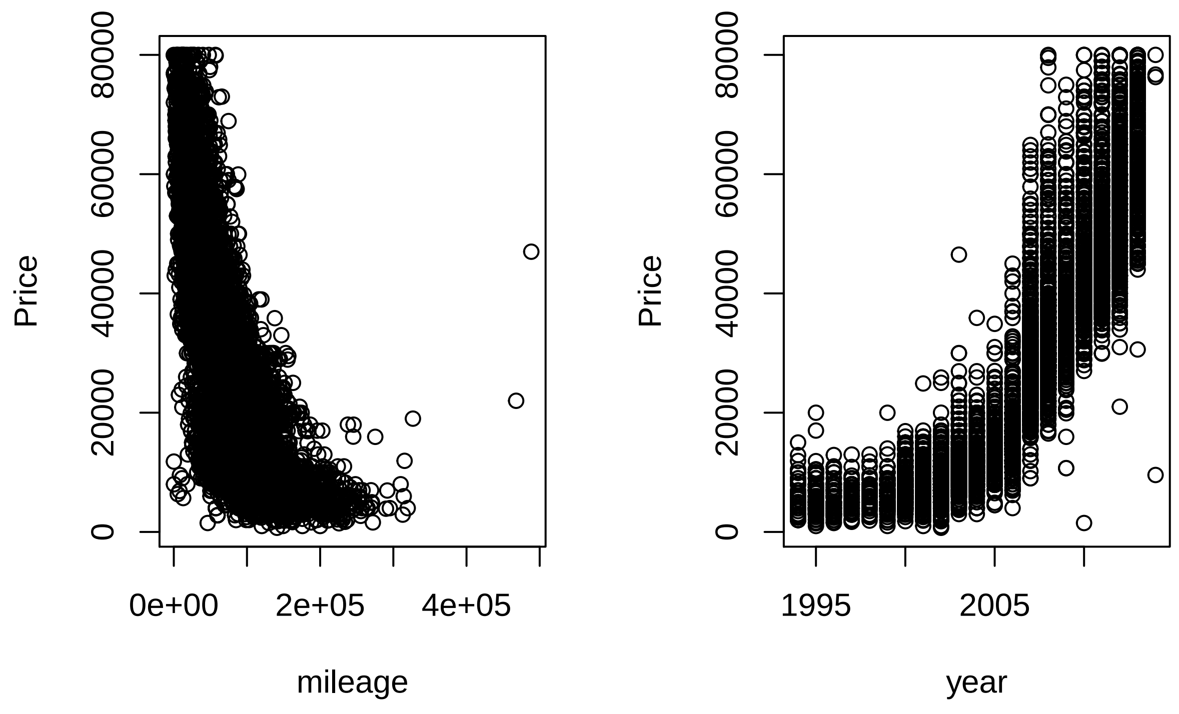
```r
#Fit to test data
kfbest = kknn(price~mileage,train,test,k=kbest,kernel = "rectangular") #get test fitted values
#Test error for k-NN
n = length(test$price)
RMSE_knn1 = sqrt(mse(test$price, kfbest$fitted.values)/n)
cat('The test error (RMSE) for k-NN model is: $',RMSE_knn1,'\n')
```

```
## The test error (RMSE) for k-NN model is: $ 9411.774
```

6)

k-NN: Predicting Price using mileage and year variables

```r
attach(train)
#get variables rescaled
x = cbind(mileage,year)
colnames(x) = c("mileage","year")
y = price
mmsc=function(x) {return((x-min(x))/(max(x)-min(x)))}
xs = apply(x,2,mmsc) #apply scaling function to each column of x
train.rescaled = data.frame(y,xs)

#plot y vs each x
par(mfrow=c(1,2)) #two plot frames
plot(x[,1],y,xlab="mileage",ylab="Price")
plot(x[,2],y,xlab="year",ylab="Price")
```



```r
#Run Cross Validation multiple times
CV_FOLDS <- 5 #folds of 5
N_CVS <- 3 #three cross validations
k_range = 50 : 100
n=length(price) #number of observations
cv_mean = rep(0,length(k_range)) #store average rmse
cv_matrix = matrix(0,length(k_range),N_CVS) #keep results for each split
for(i in 1:N_CVS) {
    cv_rmse = docvknn(xs,price,k_range,CV_FOLDS)
    cv_mean = cv_mean + cv_rmse
    cv_matrix[,i] = sqrt(cv_rmse/n)
}
```

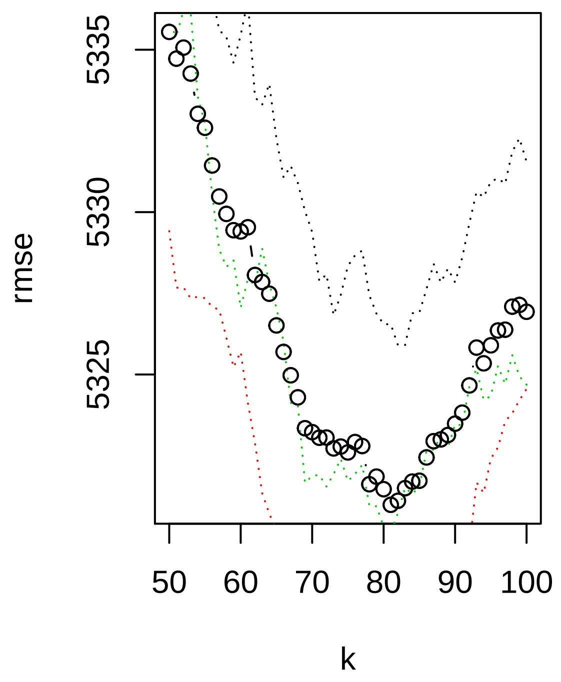
```
## in docv: nset,n,nfold:  51 15047 5
## on fold:  1 , range:  1 : 3009
## on fold:  2 , range:  3010 : 6018
## on fold:  3 , range:  6019 : 9027
## on fold:  4 , range:  9028 : 12036
## on fold:  5 , range:  12037 : 15047
## in docv: nset,n,nfold:  51 15047 5
## on fold:  1 , range:  1 : 3009
## on fold:  2 , range:  3010 : 6018
## on fold:  3 , range:  6019 : 9027
## on fold:  4 , range:  9028 : 12036
## on fold:  5 , range:  12037 : 15047
## in docv: nset,n,nfold:  51 15047 5
## on fold:  1 , range:  1 : 3009
## on fold:  2 , range:  3010 : 6018
## on fold:  3 , range:  6019 : 9027
## on fold:  4 , range:  9028 : 12036
## on fold:  5 , range:  12037 : 15047
```

```
cv_mean = cv_mean/N_CVS
cv_mean = sqrt(cv_mean/n)
plot(k_range,cv_mean,xlab="k",ylab="rmse")
for(i in 1:N_CVS) lines(k_range,cv_matrix[,i],col=i,lty=3) #plot each result
lines(k_range,cv_mean,type="b",col="black") #plot average result

detach(train)

#get the min
kbest2 = k_range[which.min(cv_mean)]
cat("the best k is: ",kbest2,"\n")
```

```
## the best k is:  81
```

```
#Rescale on test data
attach(test)
x = cbind(mileage,year)
colnames(x) = c("mileage","year")
y = price
mmsc=function(x) {return((x-min(x))/(max(x)-min(x)))}
xs = apply(x,2,mmsc) #apply scaling function to each column of x
test.rescaled = data.frame(y,xs)
detach(test)

#Fit to test data
kfbest = kknn(y~.,train.rescaled,test.rescaled,k=kbest2,kernel = "rectangular")

#Test error for k-NN
n = length(test$price)
RMSE_knn2 = sqrt(mse(test$price, kfbest$fitted.values)/n)
cat('The test error (RMSE) for k-NN model Mielage+Year is: $',RMSE_knn2,'\n')
```

```
## The test error (RMSE) for k-NN model Mielage+Year is: $ 5793.938
```

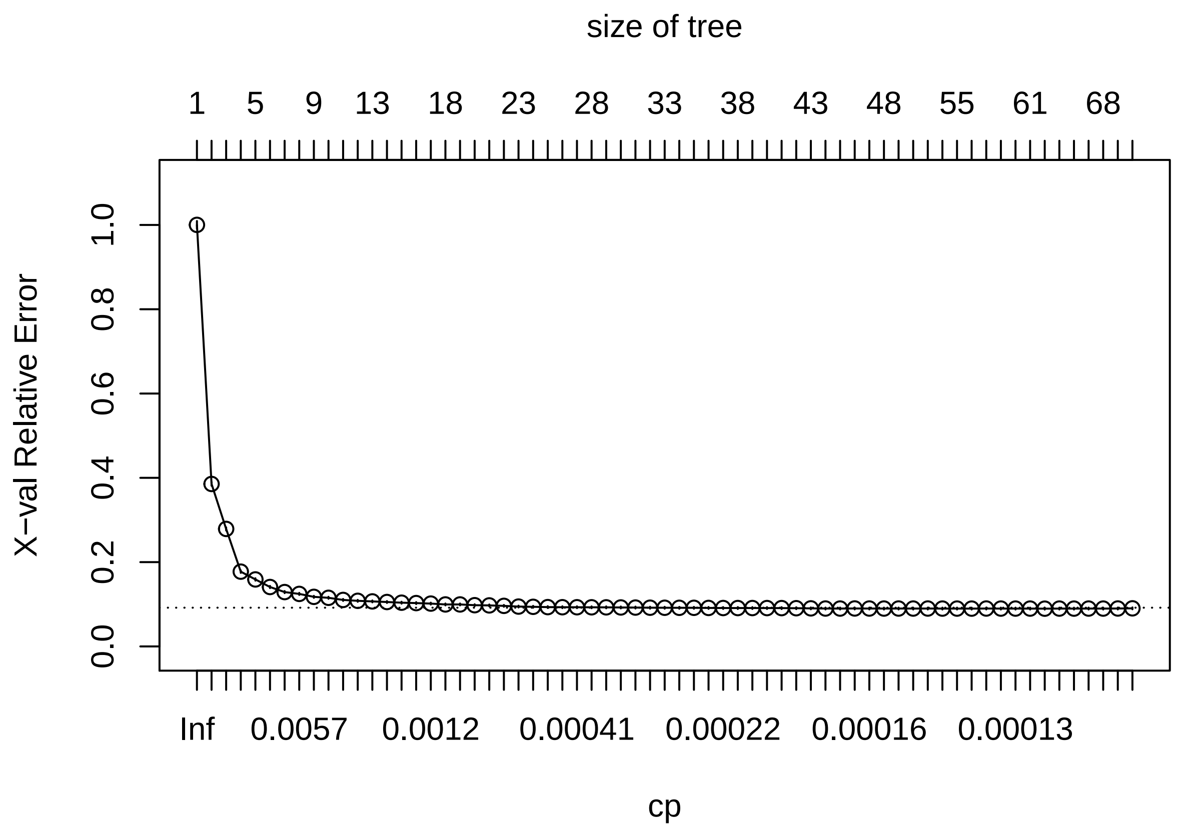


###Regression tree: Predicting Price using mileage and year variables

```r
#Regression
car.year.tree = rpart(price~mileage+year, data=train,
                          control=rpart.control(minsplit=5,
                                                cp=0.0001,
                                                xval=10)
)

#Size big tree
nbig2 = length(unique(car.year.tree$where))
cat('Size of big tree: ',nbig2,'\n')
```

```
## Size of big tree:  70
```

```r
#Prunning
plotcp(car.year.tree)
```



```r
bestcp = car.year.tree$cptable[which.min(car.year.tree$cptable[, "xerror"]),
                                       "CP"]
best.year.tree <- prune(tree = car.year.tree,
                        cp = bestcp)
cat('cp corrsponding to the smallest value of xerror is: ',bestcp,'\n')
```

```
## cp corrsponding to the smallest value of xerror is:  0.0001142549
```

```r
#Size pruned tree
nsmall2 = length(unique( best.year.tree$where))
cat('Size of pruned tree: ',nsmall2,'\n')
```

```
## Size of pruned tree:  62
```

```r
#FIT
car.year.fit = predict(best.year.tree,test) #get test fitted values for Price~Mileage+Year
car.fit = predict(best.car.tree,test) #get test fitted values for Price~Mileage

#Test error for regression tree
n = length(test$price)
RMSE_tree1 = sqrt(mse(test$price, car.fit)/n)
cat('The test error (RMSE) for regression tree model Price~Mileage is: $',RMSE_tree1,'\n')
```

```
## The test error (RMSE) for regression tree model Price~Mileage is: $ 9465.746
```

```r
RMSE_tree2 = sqrt(mse(test$price, car.year.fit)/n)
cat('The test error (RMSE) for regression tree model Price~Mileage+Year is: $',RMSE_tree2,'\n')
```

```
## The test error (RMSE) for regression tree model Price~Mileage+Year is: $ 5583.66
```

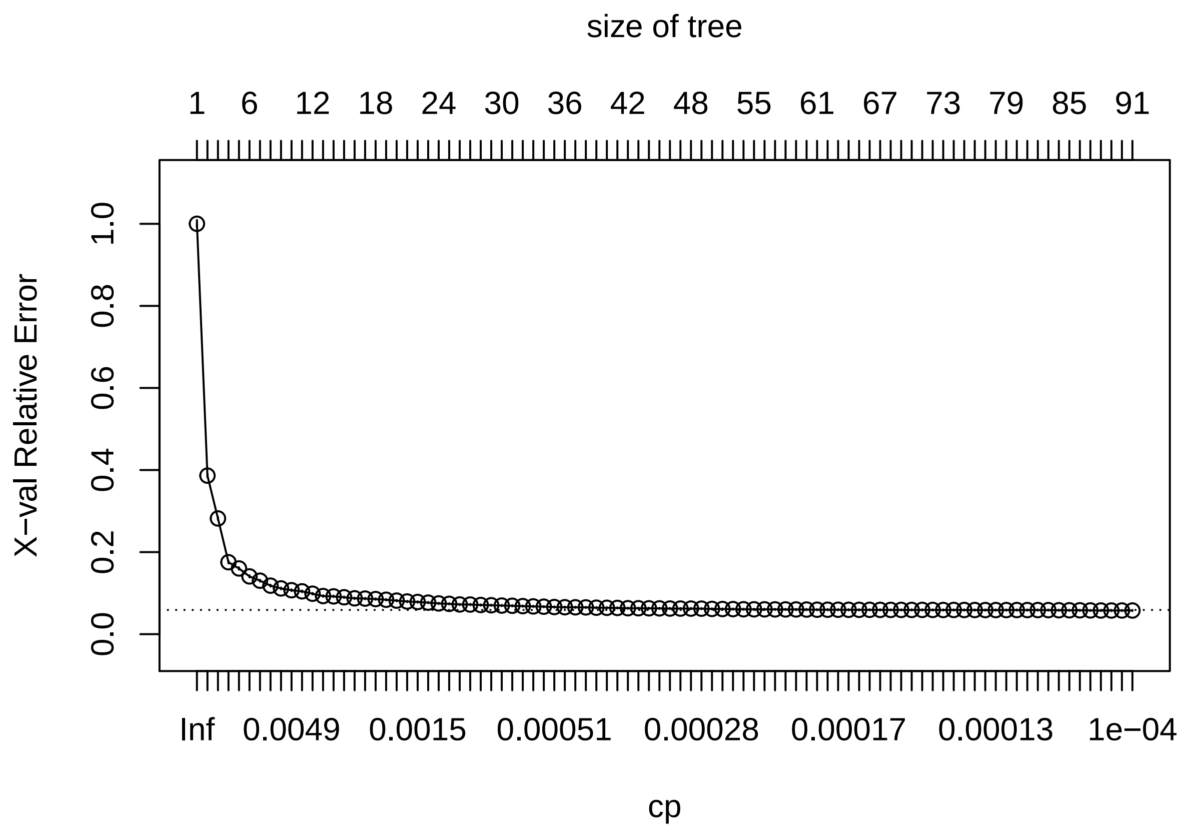
- As expected the optimal k dropped from 432 to 81 when we added the new variable Year. The additional dimension makes it so that observations are farther from each other so a smaller k is expected (the curse of dimensionality).
- The size of the optimal tree increased from 27 to 62 because the new dimension will naturally require more splitting of the data, thus a larger trees.
- To test model performance, we predict test data using the four models and compute the RMSE for each. The models performance for both k-NN and regression tree improved by adding the variable Year. For the k-NN models, the RMSE dropped from 9411.7740841 to 5793.938383; while for regression trees the RMSE dropped similarly from 9465.7457192 to 5583.6595812. Adding the variable Year improved our model's predictive ability.

7)

```r
#Dummy variables
train$isOneOwner = as.factor(train$isOneOwner)
train$color = as.factor(train$color)
train$fuel = as.factor(train$fuel)
train$region = as.factor(train$region)
train$soundSystem = as.factor(train$soundSystem)
train$wheelType = as.factor(train$wheelType)

#Regression tree
car.big.tree = rpart(price~., data=train,
                         control=rpart.control(minsplit=5,
                                               cp=0.0001,
                                               xval=10)
)

#Size big tree
nbig3 = length(unique(car.big.tree$where))
cat('Size of big tree: ',nbig3,'\n')
```

```
## Size of big tree:  91
```

```r
#prunning
plotcp(car.big.tree)
```



```r
bestcp = car.big.tree$cptable[which.min(car.big.tree$cptable[, "xerror"]),
                                        "CP"]
best.all.tree <- prune(tree = car.big.tree,
                       cp = bestcp)
cat('cp corrsponding to the smallest value of xerror is: ',bestcp,'\n')
```

```
## cp corrsponding to the smallest value of xerror is:  0.0001001118
```

```r
#Size pruned tree
nsmall3 = length(unique(best.all.tree$where))
cat('Size of big tree: ',nsmall3,'\n')
```

```
## Size of big tree:  89
```

```r
#FIT
car.big.fit = predict(best.all.tree,test) #get test fitted values

#Test error for regression tree
n = length(test$price)
RMSE = sqrt(mse(test$price, car.big.fit)/n)
cat('The test error (RMSE) for regression tree model on all variables is: $',RMSE,'\n')
```

```
## The test error (RMSE) for regression tree model on all variables is: $ 4565.161
```

Bonus Question 2

The following approach may be used for variable selection. It mirrors some methods that we have discussed briefly such as bagging and random forests:

1. Randomly separate the data into different segments or folds.
2. For each fold, fit the regression tree (for constant α)
3. At each node, calculate a weighted impurity decrease e.g. Mean Decrease Gini function for the predictor p_i used.
4. Add up the weighted impurity decreases across the tree for each predictor. This gives us a way of ranking the predictive power of each variable.
5. Repeat this process for the other folds, and take an average for each predictor to determine the overall result.
6. We can choose the most influential predictor using the average weighted impurity decreases across all folds.
7. The process can be modified to randomize the predictor choices at each node (as in a Random forest). We can then average results across all random forests for our mean impurity decrease function. This would probably be particularly helpful with the issue of correlated predictors.