Machine Learning

Question 1)

General Functions)

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

```
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){
```

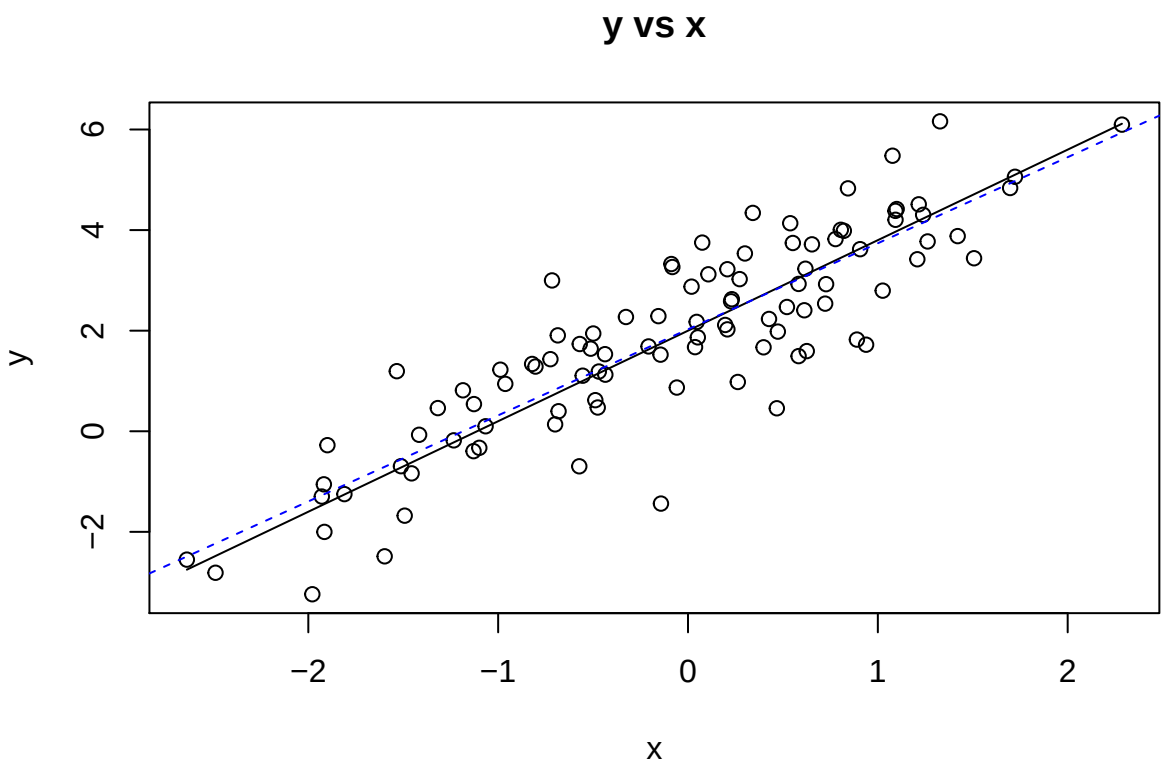
```
  mdl_tr <- lm(y ~ x, data = train)
  plot_regression(f, mdl_tr, train)
  plot_knn(f, train)
  plot_mse(train, test, mdl_tr)
}
```

1: Generating Data)

```
set.seed(98)
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)
```

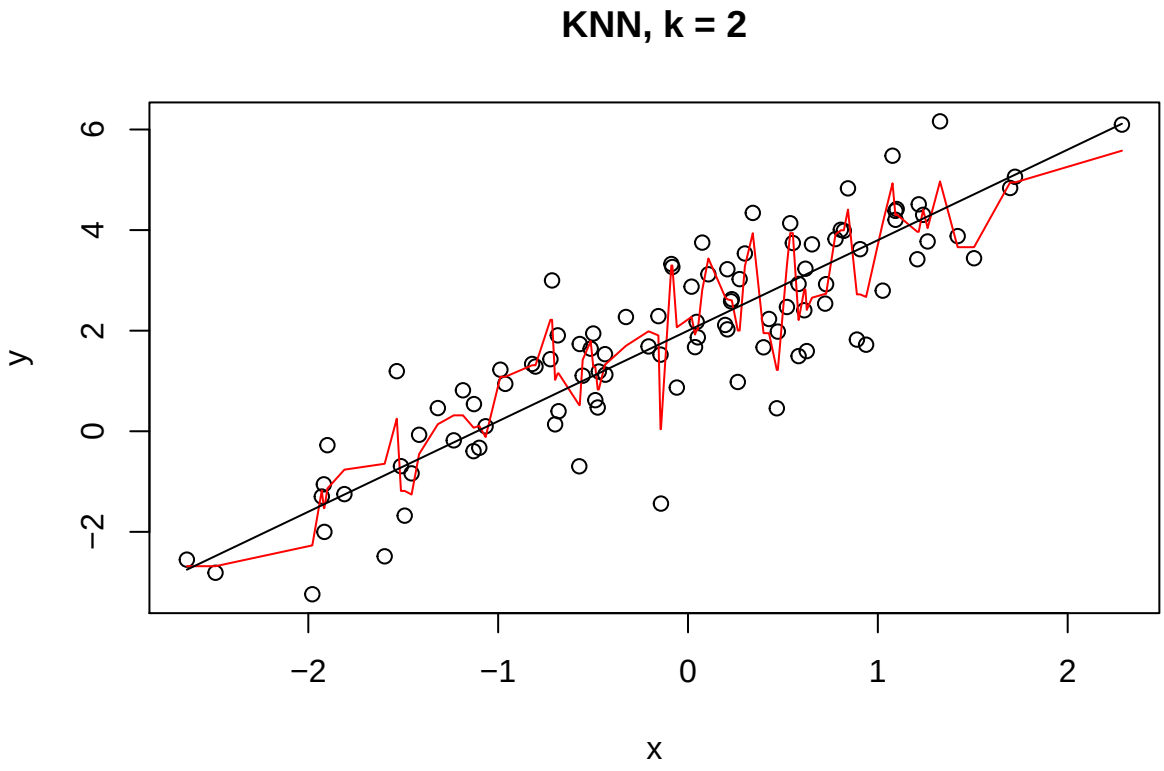
2: Scatterplot and 3) Regression (Train)

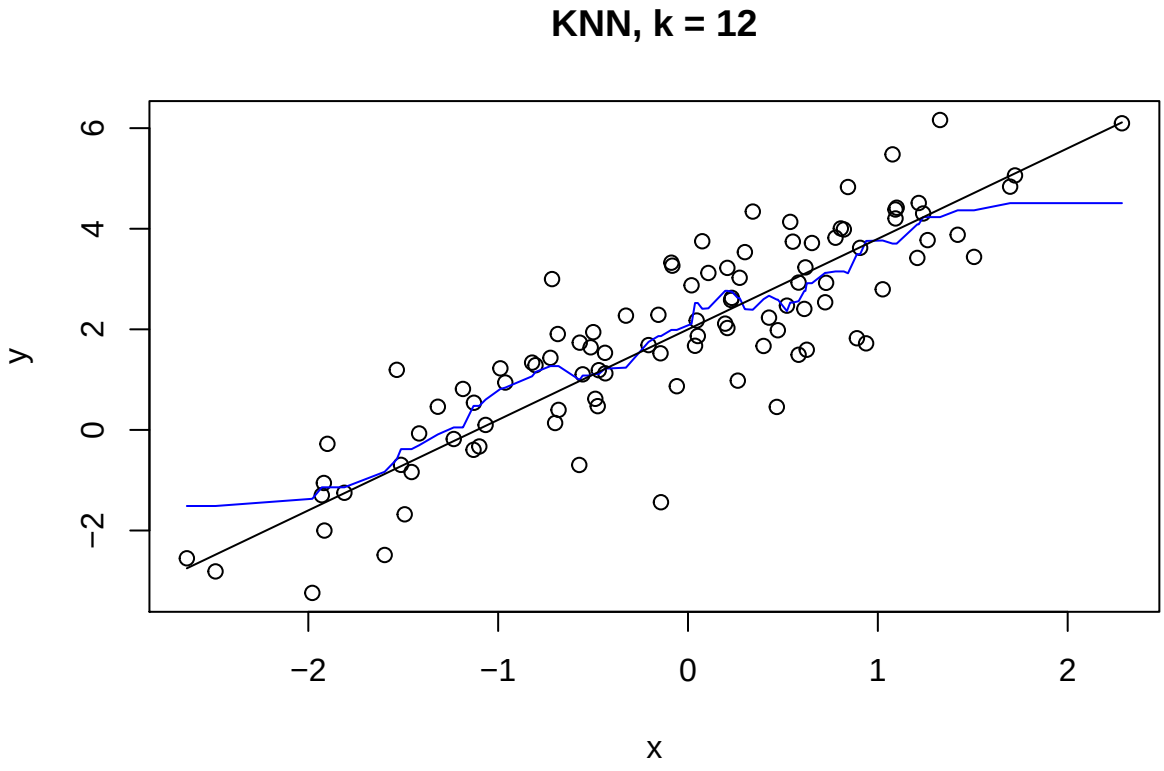
```
mdl_tr <- lm(y ~ x, data = train)
plot_regression(f, mdl_tr, train)
```



4) KNN

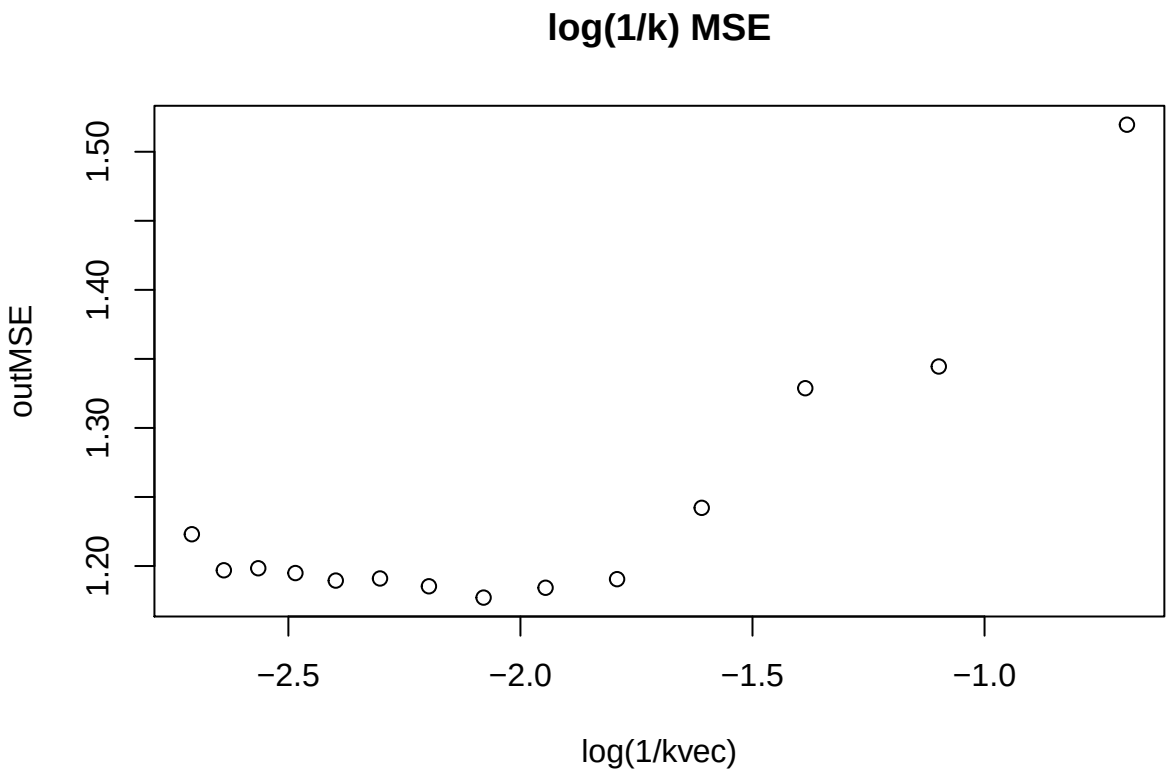
```
plot_knn(f, train)
```


KNN, k = 2
y
x
6
4
2
0
−2
−2
−1
0
1
2


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

5) MSE, Performance

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




```
## best k is  8
## Regression MSE: 1.006323
```

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.

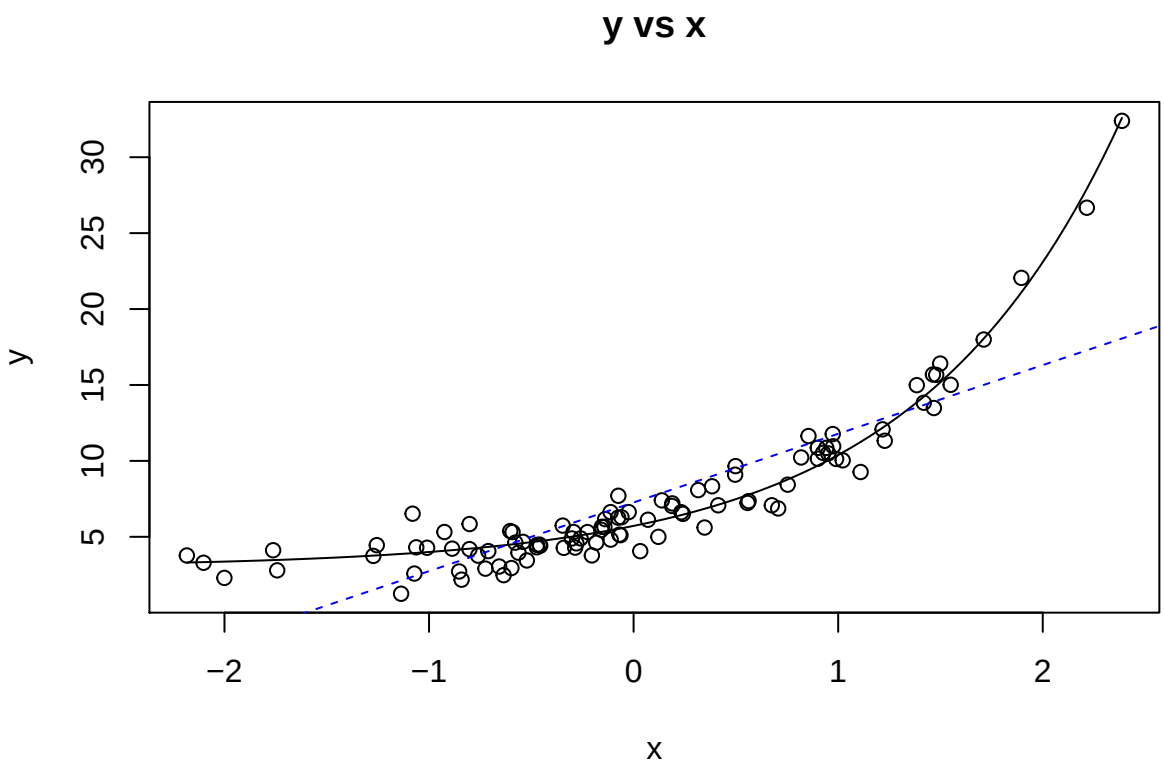
6) Exponential

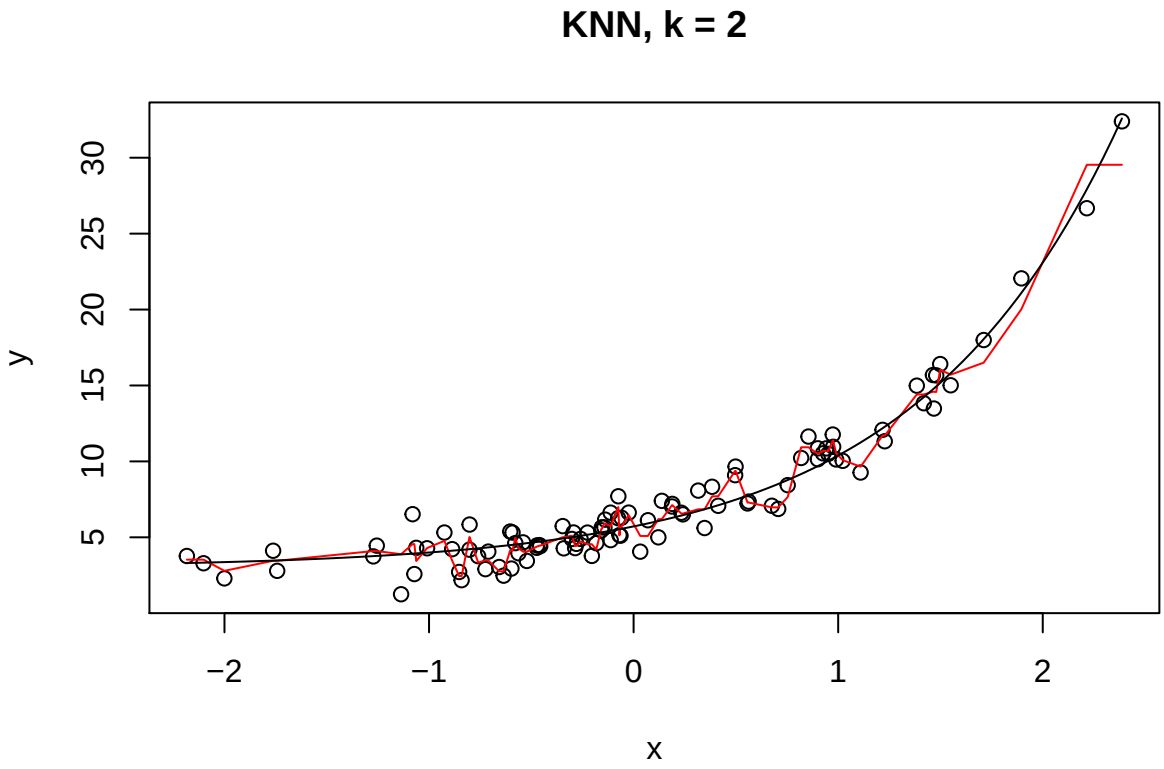
a) Generate Data

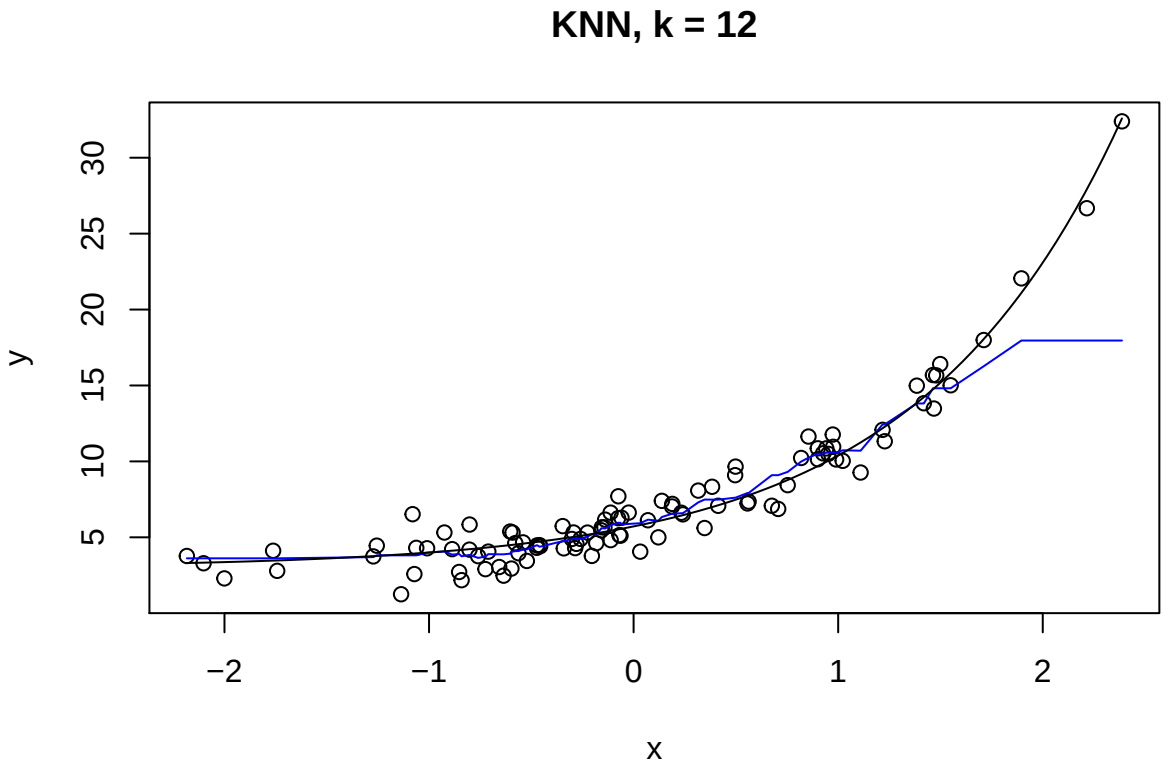
```
set.seed(5)
f_exp <- function(x){return(exp(x + 1) + 3)}
x_train <- rnorm(100)
y_train <- f_exp(x_train) + rnorm(100)

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)
```

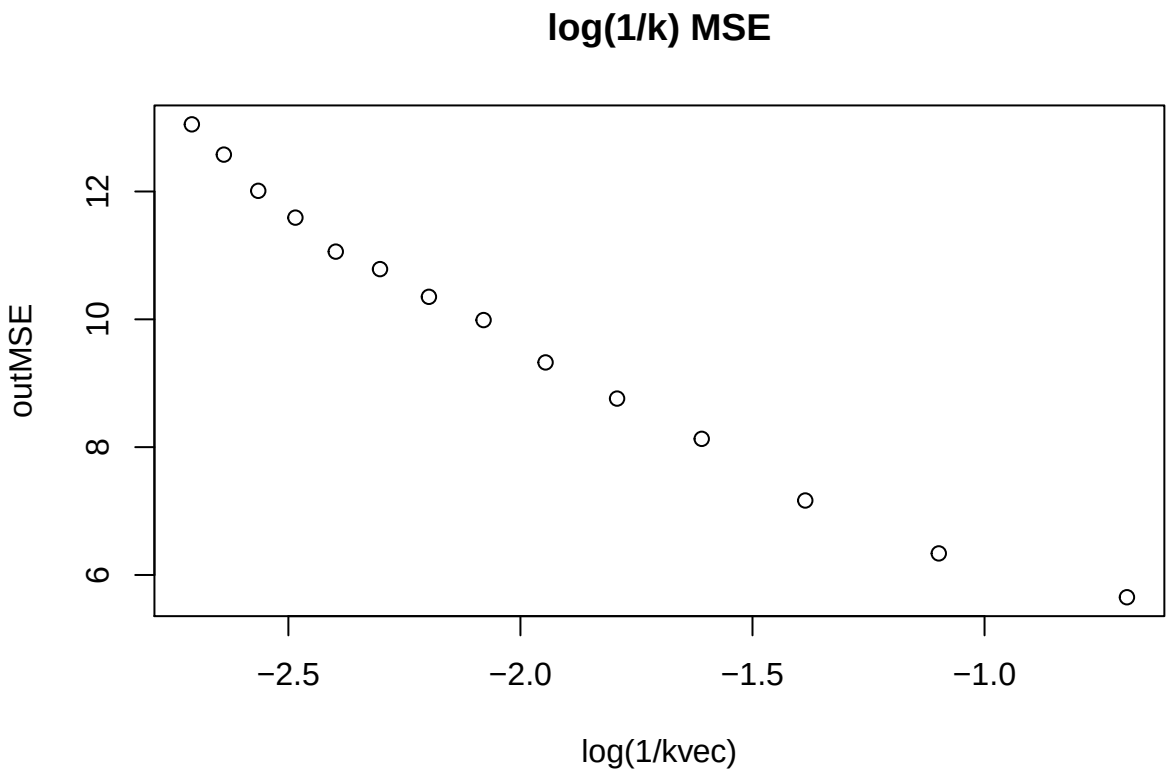
b) the other steps

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




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


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




```
## best k is  2
## Regression MSE: 14.77465
```

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.

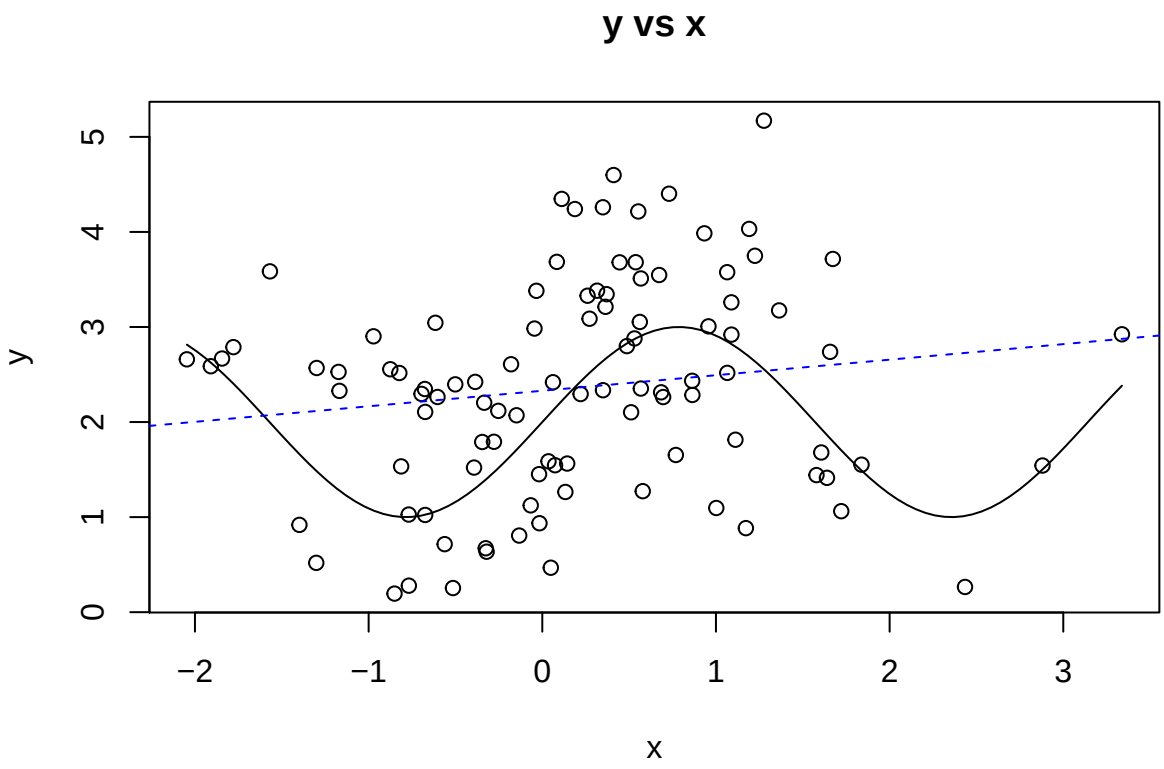
7) Sine

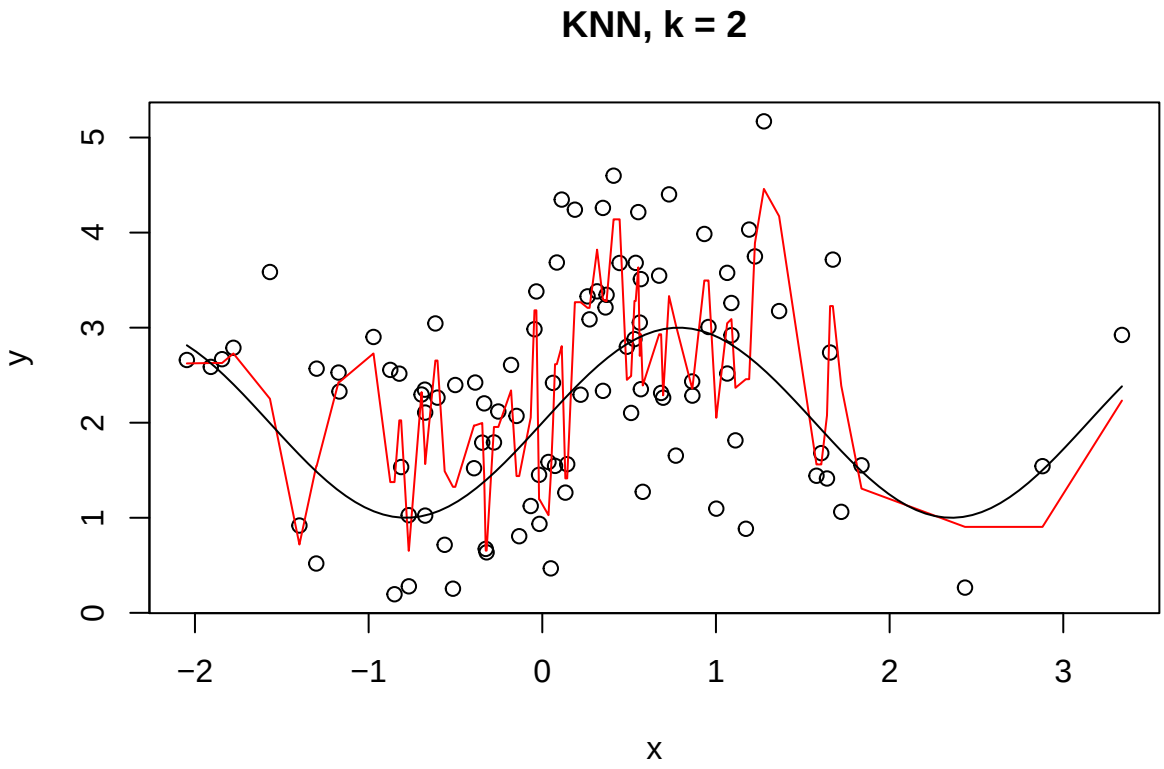
a) Generate Data

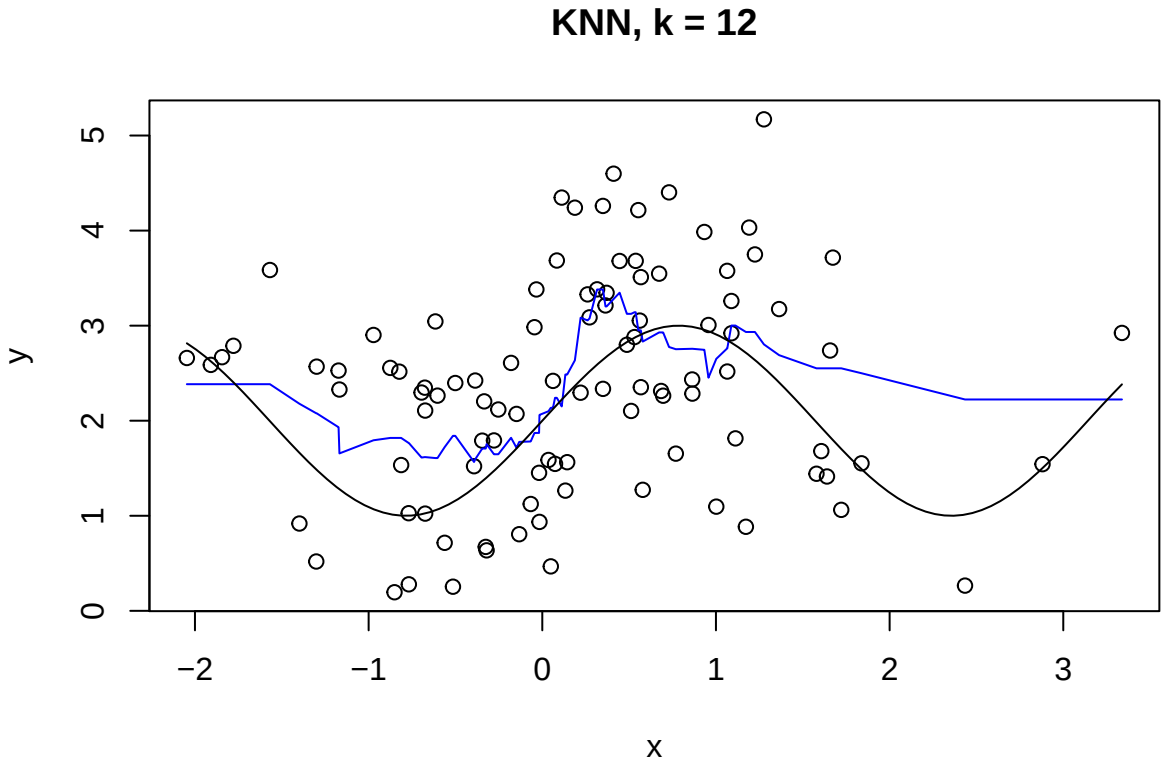
```
set.seed(35)
f_sin <- function(x){return(sin(2*x) + 2)}
x_train <- rnorm(100)
y_train <- f_sin(x_train) + rnorm(100)

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)
```

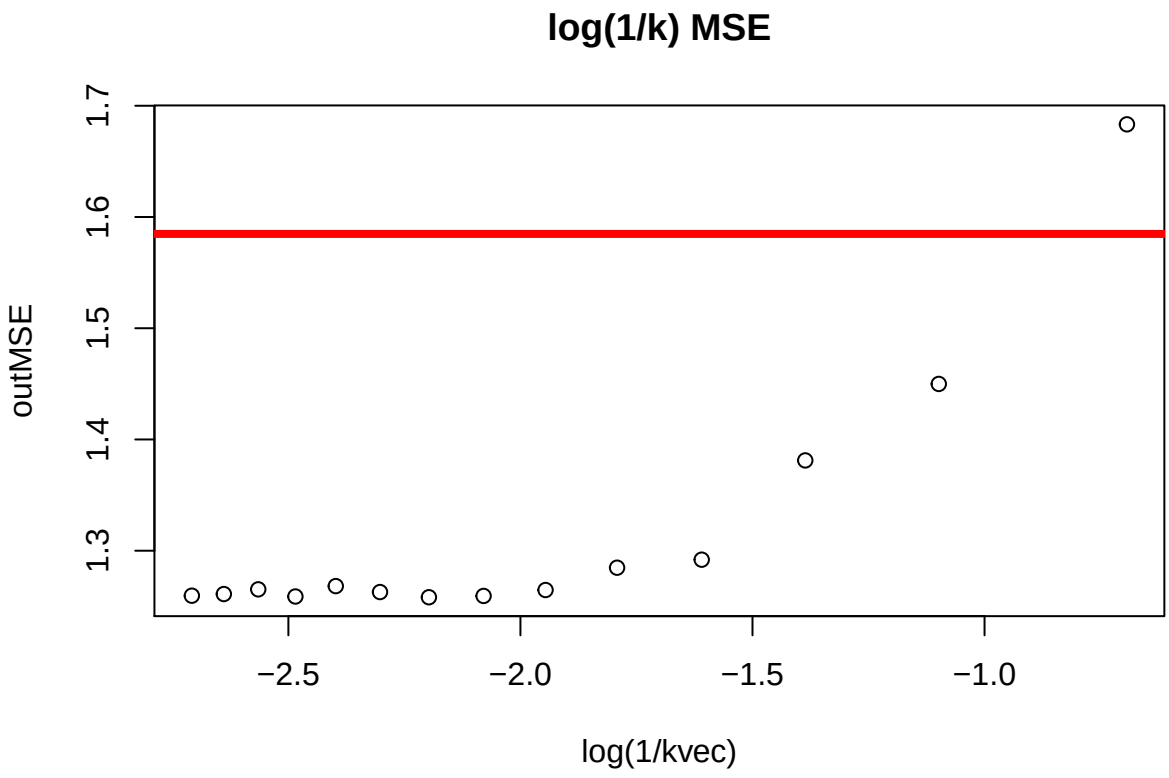
b) The rest

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




KNN, k = 2
y
x
0
1
2
3
4
5
−2
−1
0
1
2
3


KNN, k = 12
y
x
0
1
2
3
4
5
−2
−1
0
1
2
3




```
## best k is  9
## Regression MSE: 1.584797
```

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.

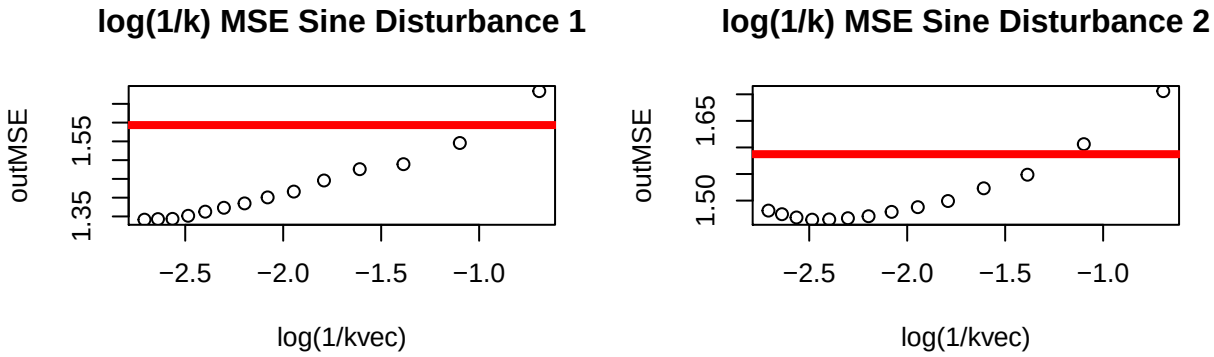
8 Disturbing the neighbors)

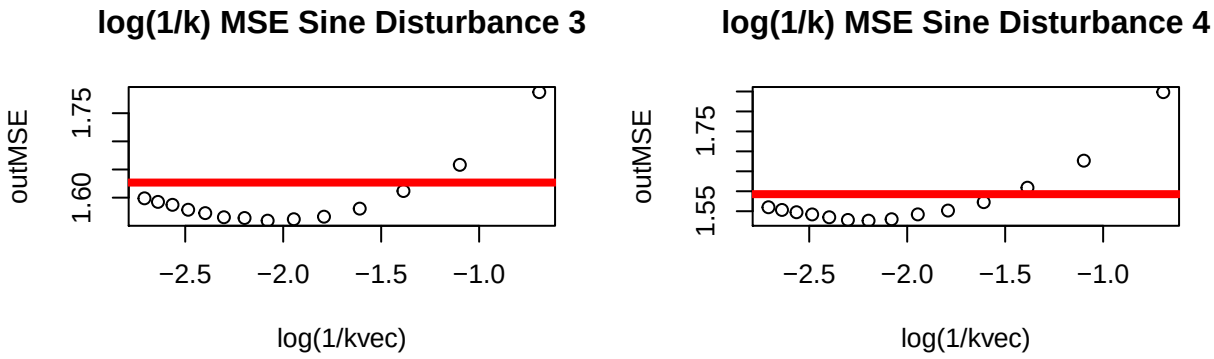
```
steps_psin <- function(p, train, test){
  p_vec <- 1:p
  x_p_train <- matrix(rnorm(100 * length(p_vec)), ncol = length(p_vec))
  colnames(x_p_train) <- p_vec
  x_p_test <- matrix(rnorm(10000 * length(p_vec)), ncol = length(p_vec))
  colnames(x_p_test) <- p_vec

  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  15
## Regression MSE: 1.593403
```

```
## best k is  12
## Regression MSE: 1.587342
```

```
## best k is  8
## Regression MSE: 1.626781
```



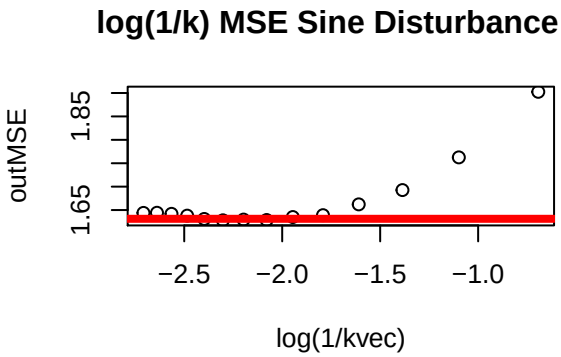


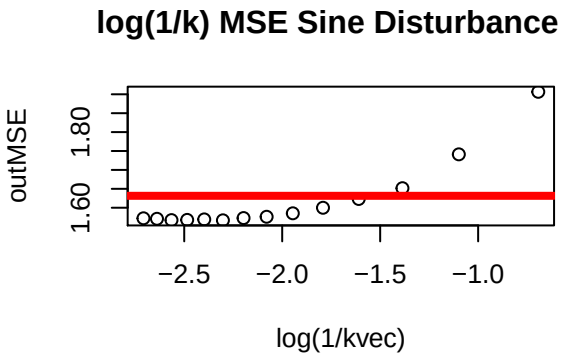

```
## best k is  9
## Regression MSE: 1.592729
```

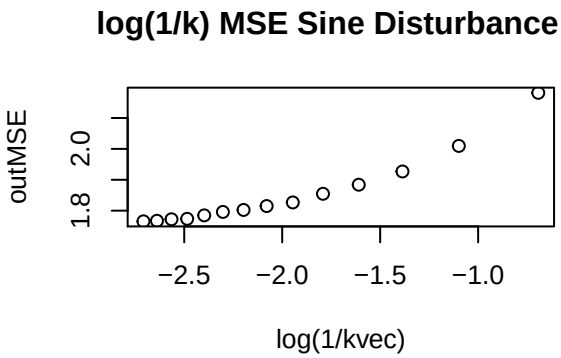
```
## best k is  10
## Regression MSE: 1.631321
```

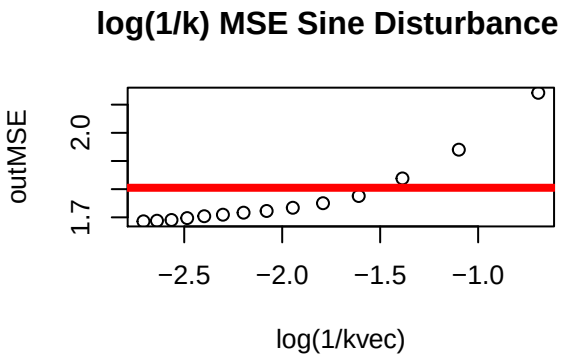
```
## best k is  10
## Regression MSE: 1.631525
```

```
## best k is  15
## Regression MSE: 1.623652
```

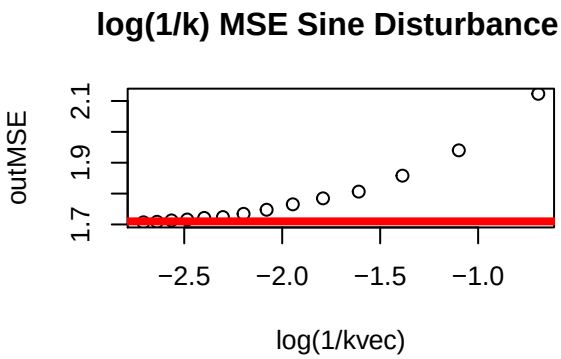


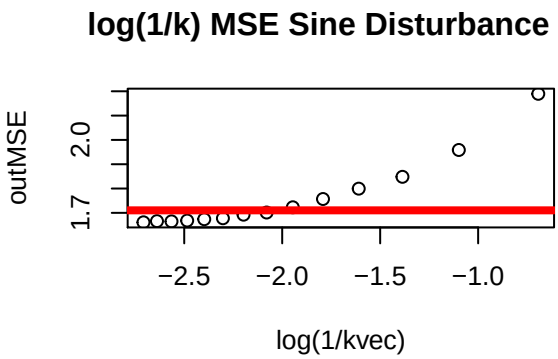


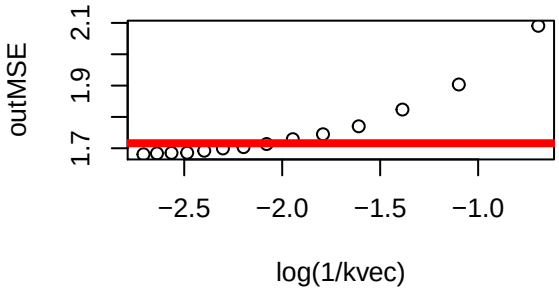


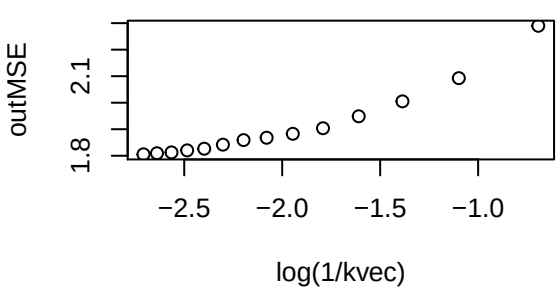



```
## best k is  15
## Regression MSE: 1.804985

## best k is  15
## Regression MSE: 1.709953

## best k is  15
## Regression MSE: 1.71035

## best k is  15
## Regression MSE: 1.715966
```







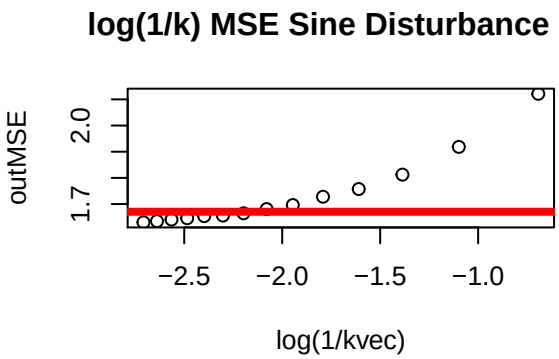


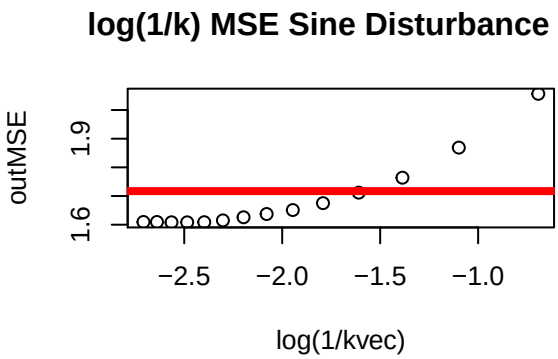

```
## best k is  15
## Regression MSE: 1.621578
```

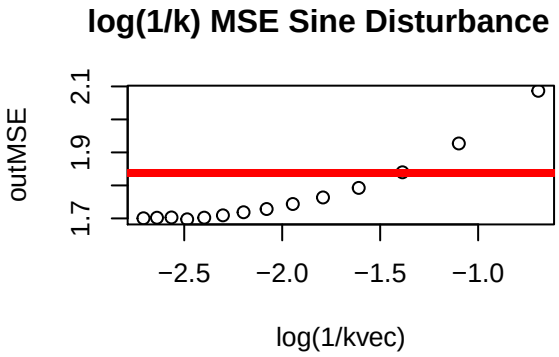
```
## best k is  15
## Regression MSE: 1.670572
```

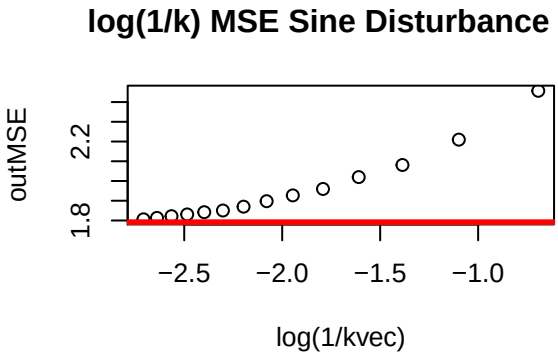
```
## best k is  12
## Regression MSE: 1.717137
```

```
## best k is  12
## Regression MSE: 1.837632
```

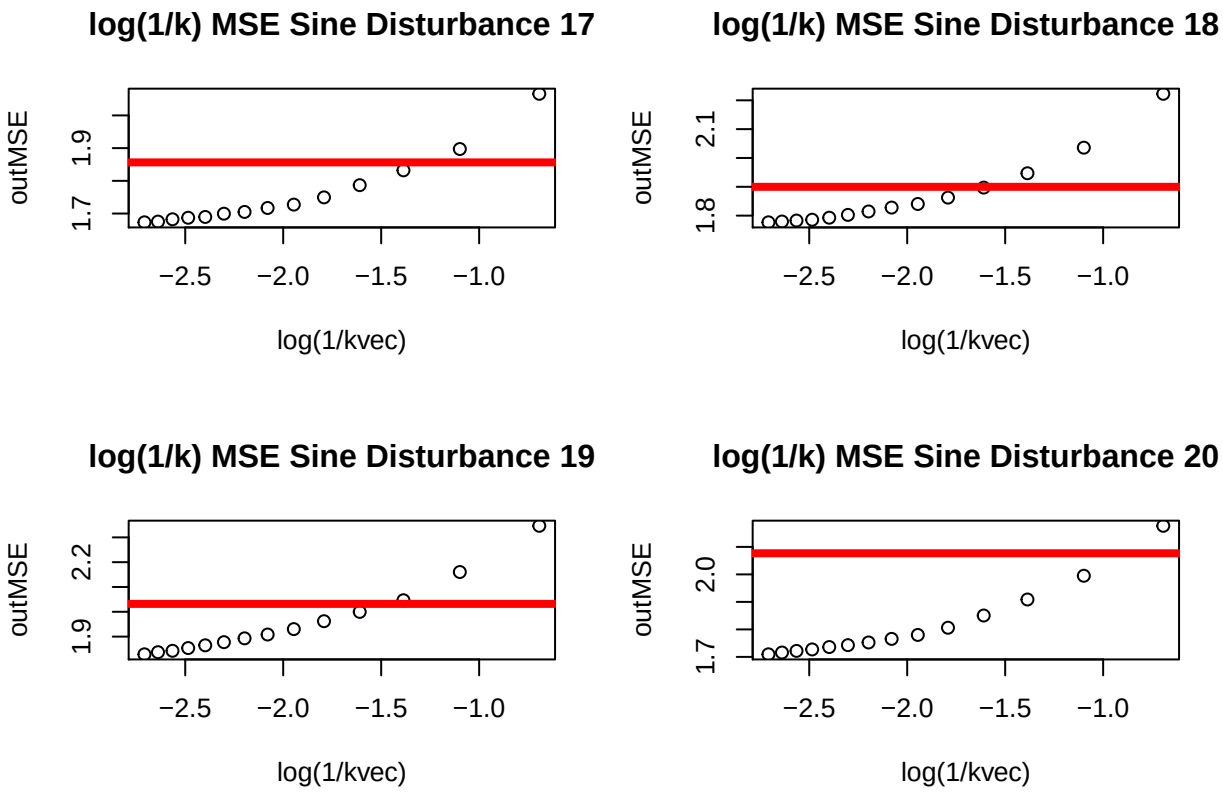









```
## best k is  15
## Regression MSE: 1.785456

## best k is  15
## Regression MSE: 1.856361

## best k is  15
## Regression MSE: 1.899913

## best k is  15
## Regression MSE: 2.031901
```




```
## best k is  15
## Regression MSE: 2.076809
```

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.

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.