# The University of British Columbia

*Data Science 570 Predictive Modelling*

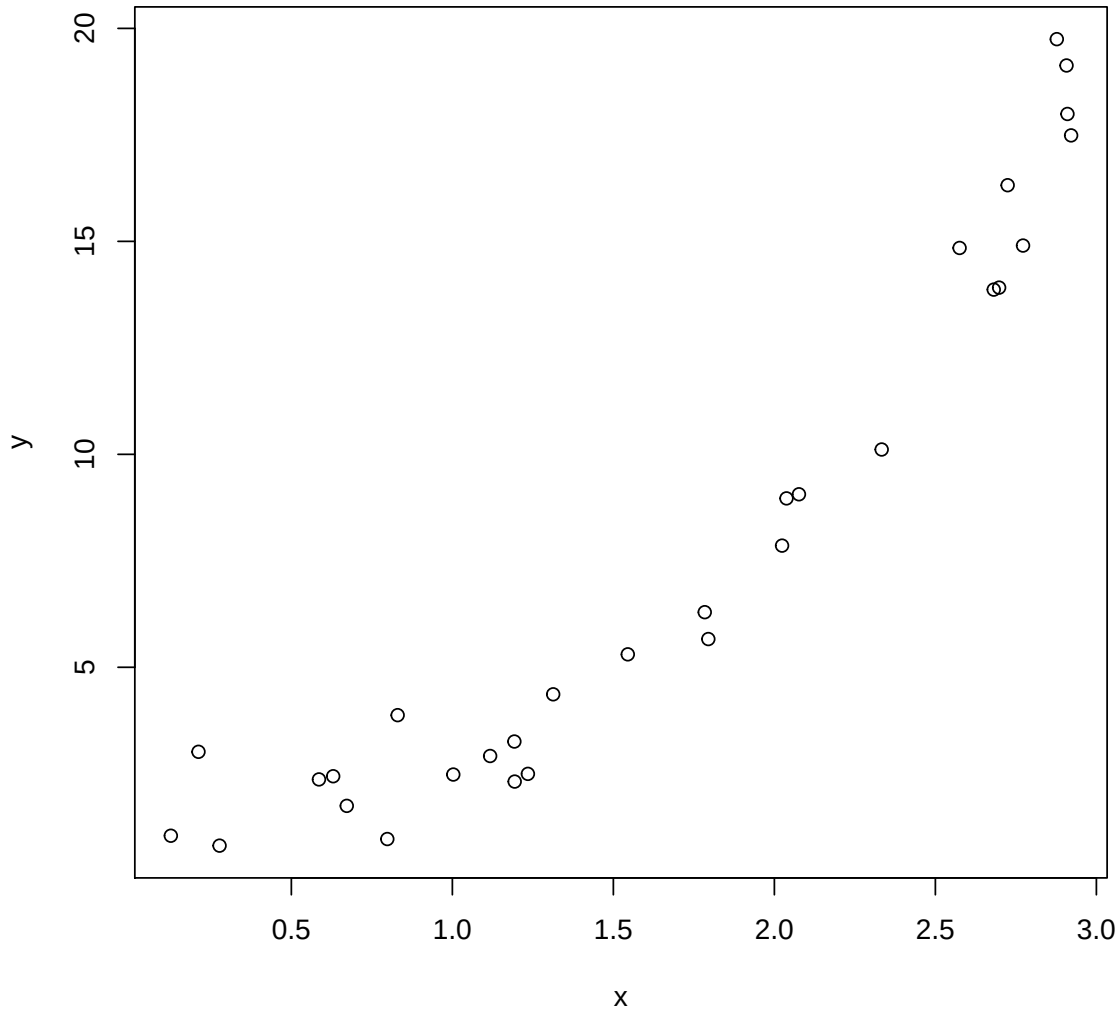
Lab Assignment 1

The TA will demonstrate exercises 1 through 3. You are expected to submit answers to exercises 4 through 7. Instructions: Please use a png, pdf or html file for submission.

1. Assessing Models: MSE

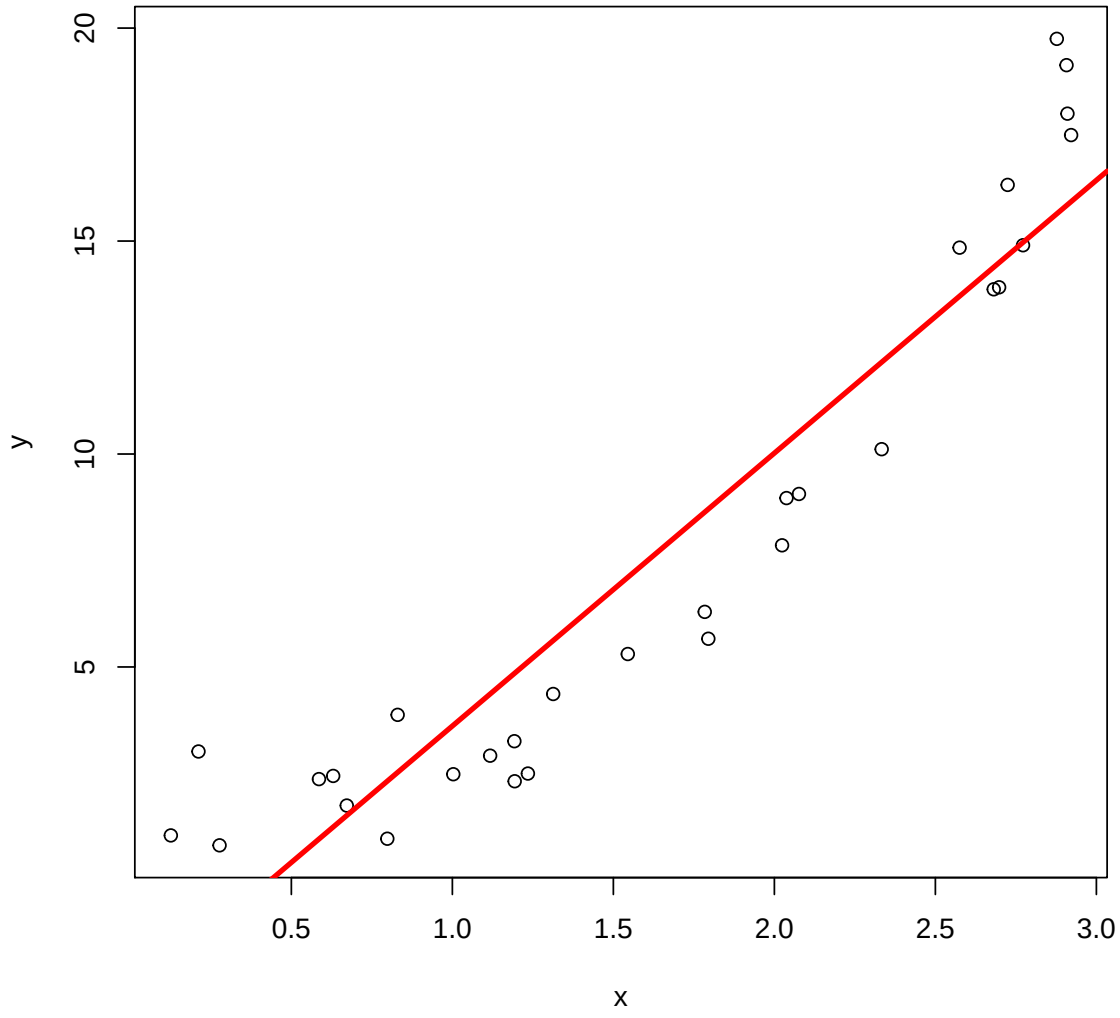
   Let's recreate the regression models shown during Lecture 2. First, simulate our training data by uniformly generating X values, and then assuming an exponential relationship between X and Y with standard normal (mean 0, variance 1) errors.

```
set.seed(23418)
x <- sort(runif(30,0,3))
y <- exp(x) + rnorm(length(x))
plot(x, y)
```



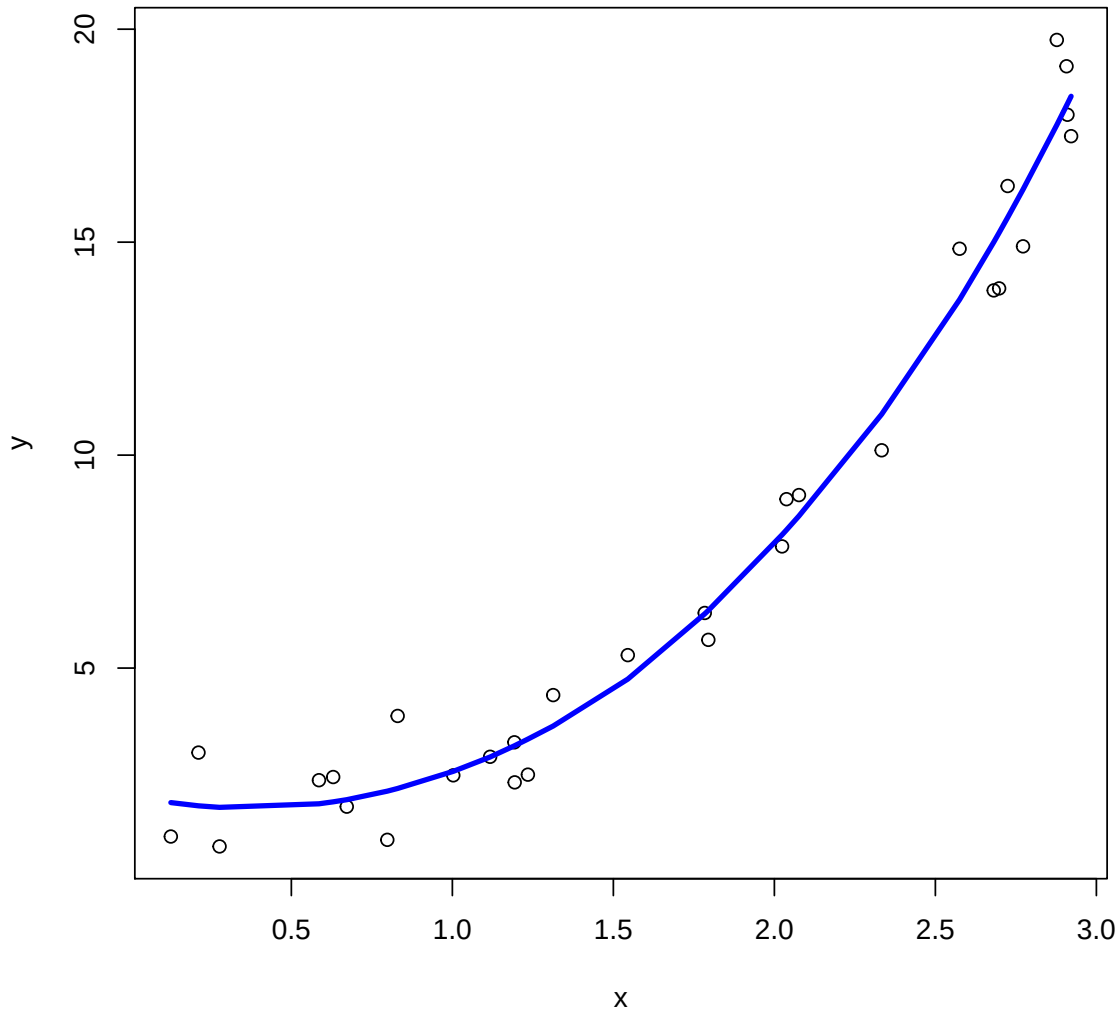
Now fit a standard linear model, and add the line to the plot in red. Note that for markdown, we need to re-plot X and Y in any new code chunk when attempting to add lines.

```
plot(x, y)
linmod <- lm(y ~ x)
abline(linmod, col="red", lwd=3)
```




Next, we fit a 'local polynomial' model and add it in blue. You may see these in more detail in another module, for now just consider it a relatively flexible model.

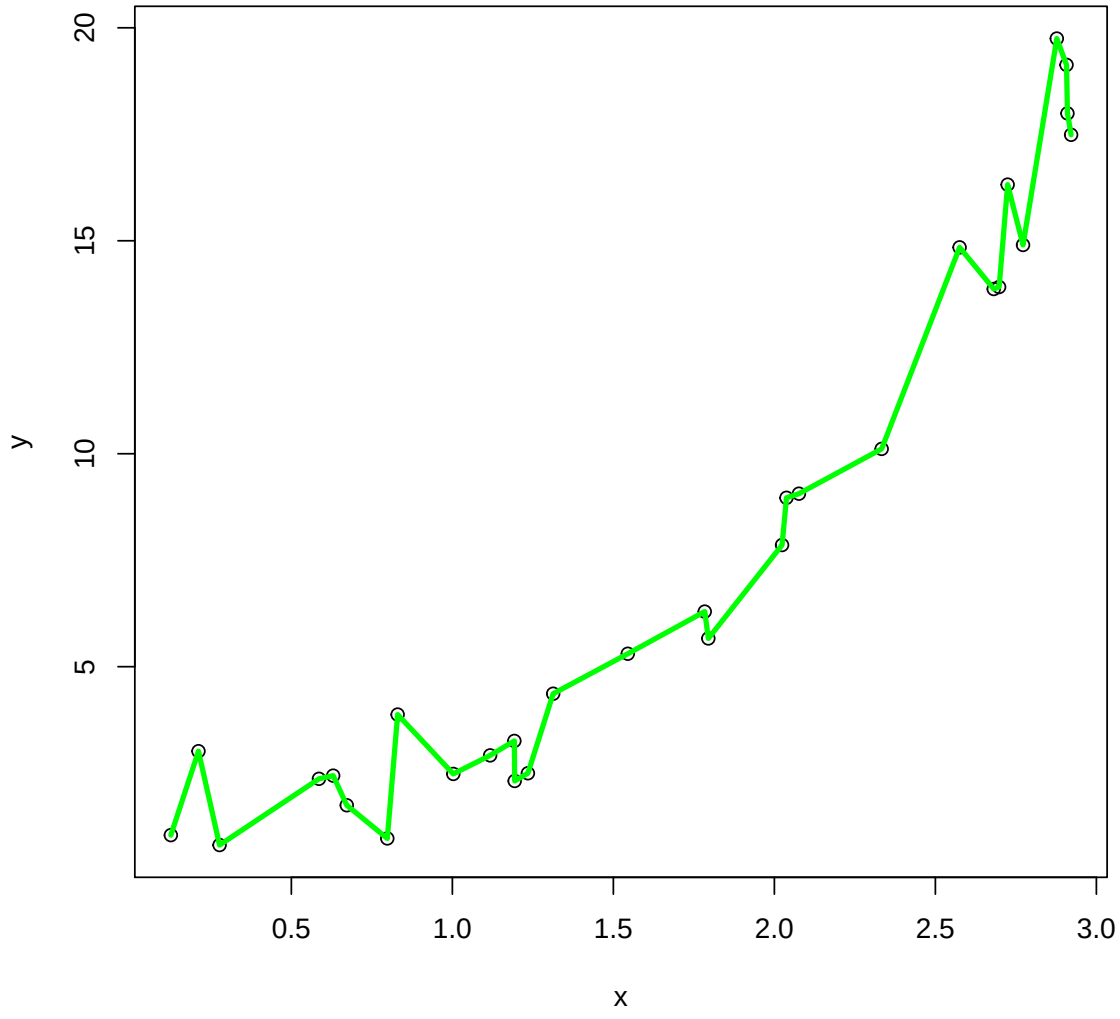
```
plot(x, y)
localpoly1 <- loess(y~x, span=0.75)
lines(x, predict(localpoly1), col="blue", lwd=3)
```




Finally, let's fit another local polynomial that approximates 'connect the dots' and add it in green. We do so by changing the span parameter to an unreasonably small value for the data — this will cause several warnings (which we will ignore for this module).

```
plot(x, y)
# Green line "connect the dots" poly
localpoly2 <- loess(y~x, span=0.1)

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
:  span too small.    fewer data values than degrees of freedom.
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
:  pseudoinverse used at 0.11179
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
:  neighborhood radius 0.16526
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
:  reciprocal condition number  0
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
:  There are other near singularities as well.  0.00079695

lines(x, predict(localpoly2), col="green", lwd=3)
```



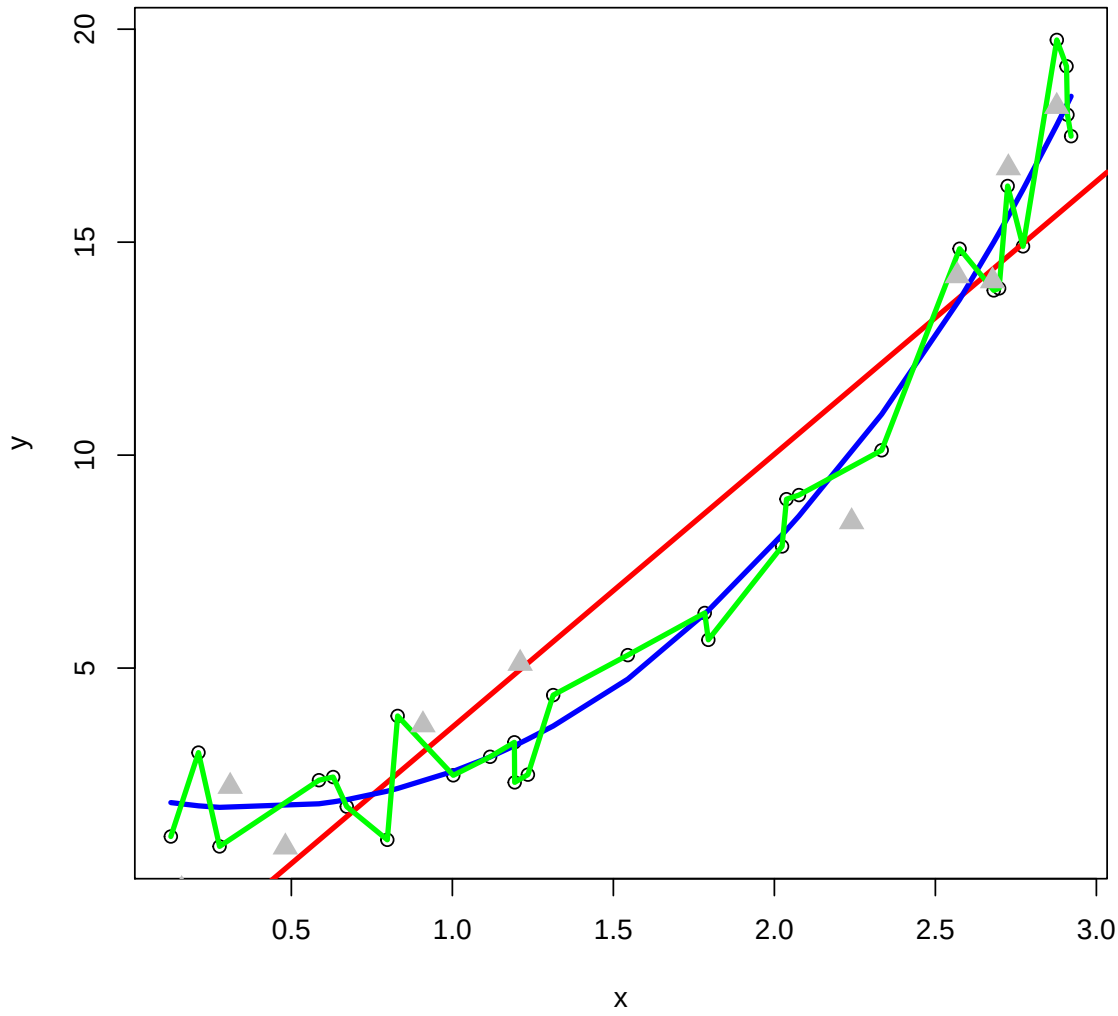

Now we can calculate the training MSE for each model by utilizing the 'predict()' function. By default, 'predict()' simply provides $\hat{y}$ for the previously observed predictors (aka, the training data). We'll use 'round()' to print the MSE with 2 decimal places

```
trmse_red <- mean((y - predict(linmod))^2)
trmse_blue <- mean((y - predict(localpoly1))^2)
trmse_green <- mean((y - predict(localpoly2))^2)
round(trmse_red, 2)
```

```
## [1] 4.37

round(trmse_blue, 2)

## [1] 0.83

round(trmse_green, 2)

## [1] 0
```

Now we will generate 10 new points from the same simulation, and plot alongside our 3 fitted models

```
set.seed(41368)
xnew <- sort(runif(10,0,3))
ynew <- exp(xnew) + rnorm(length(xnew))
plot(x, y)
abline(linmod, col="red", lwd=3)
lines(x, predict(localpoly1), col="blue", lwd=3)
lines(x, predict(localpoly2), col="green", lwd=3)
points(xnew, ynew, pch=17, col="gray", cex=1.5)
```




We can check the MSE of the test set by giving the 'predict' function the new data. Note that 'predict' specifically wants new data as a 'data.frame' structure — we will run into little issues like this throughout the course.

```
temse_red <- mean( (ynew - predict(linmod, data.frame(x = xnew)))^2 )
temse_blue <- mean( (ynew - predict(localpoly1, data.frame(x = xnew)))^2 )
temse_green <- mean((ynew - predict(localpoly2, data.frame(x = xnew)))^2 )
temse_red
```

```
## [1] 3.319594
```

```
temse_blue
```

```
## [1] 1.564691
```

```
temse_green
```

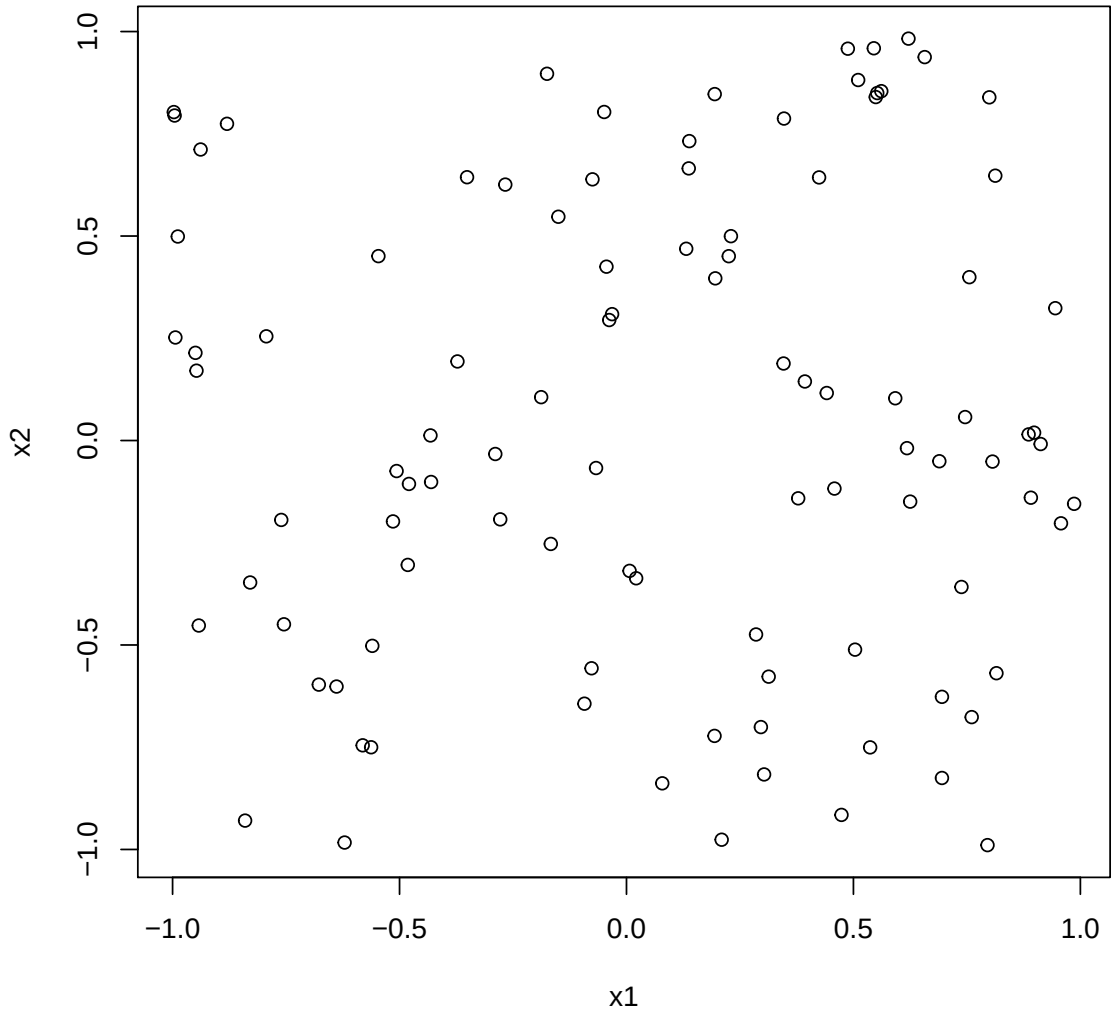
```
## [1] 4.291645
```

As noted in lecture, most of the results follow the general rule that testing MSE's will be larger than training MSE's, though by chance this is not currently holding for the linear model. Feel free to remove the 'set.seed' command and see how the results change with additional simulations.

2. Assessing Models: Classification

   Let's now recreate the classification example shown in lecture 2. Here we'll simulate an underlying classification model where the continuous variables are uniformly distributed...

```
set.seed(4623)
x1 <- runif(100, -1, 1)
x2 <- runif(100, -1, 1)
plot(x1, x2)
```




and the group classification probabilities are associated with the X2 variable. Specifically, the probability of being in "Class 1" is equal to the observation's value at X2 or

0 when X2 is negative. Note that loops can, and should, generally be avoided in ‘R‘. We will often break that rule in lab in the interest of simplicity. First, let’s initialize a classification vector of NAs (missing values)

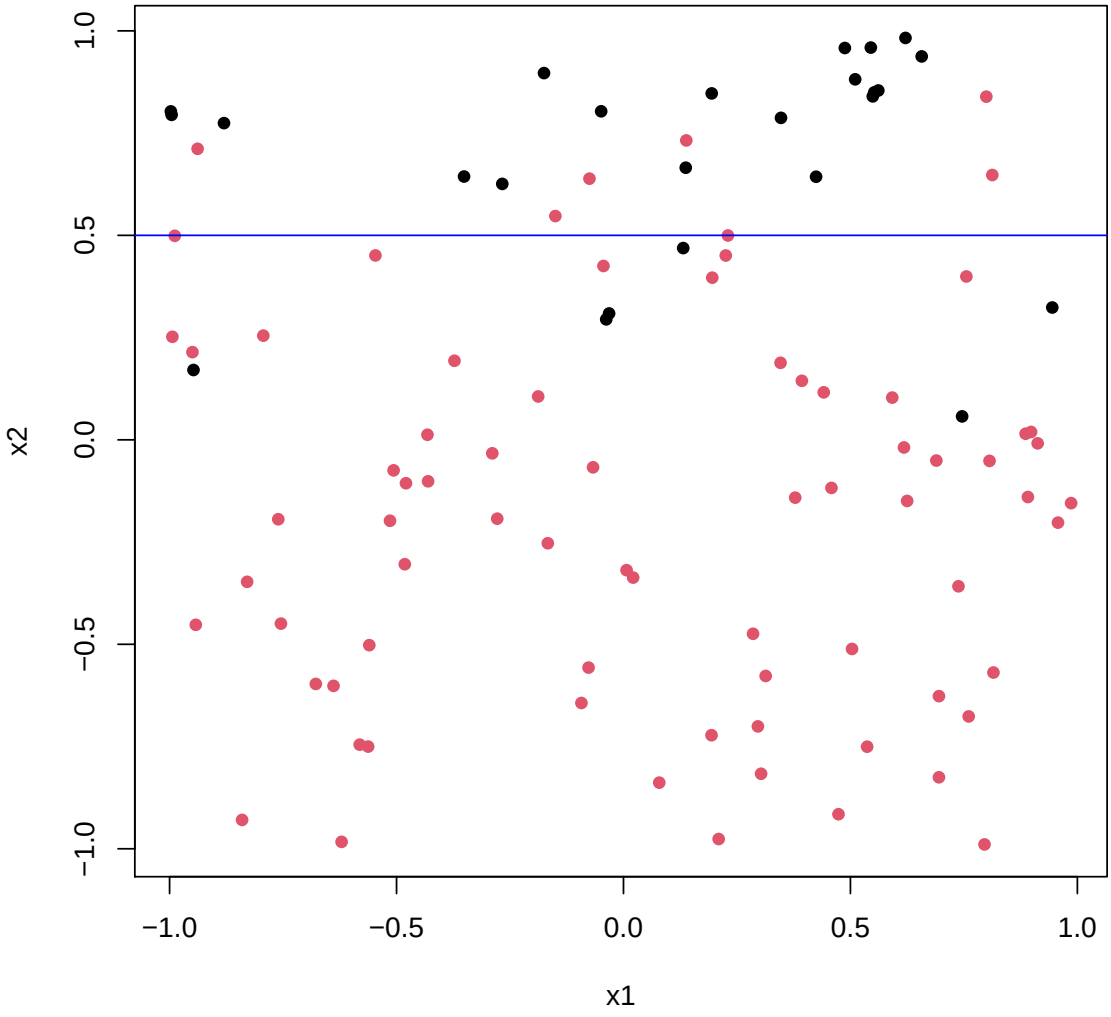
```
clas <- rep(NA, length(x2))
```

Now we will replace each element of that vector with an ”observed” classification sampled from the probability scheme mentioned above.

```
for(i in 1:length(x2)){
  clas[i] <- sample( c(1, 2) , size = 1, prob =c(max(0, x2[i]), min(1 - x2[i], 1)))
}
```

Now we can plot the points according to their observed classification, and include the Bayes Classifier boundary at X2 = 0.5 in blue.

```
plot(x1, x2, col = clas, pch=16)
abline(h = 0.5, col="blue")
```




Furthermore, we can calculate what the observed classification error would be for the Bayes Classifier (that is, the correct classification model). That model would tell us that if X2 ¿ 0.5, we should classify as group "1" (black) and if X2 ¡ 0.5 we should classify as group "2" (red). So we can tabulate that

```
table(x2 > 0.5, clas)

##        clas
##          1  2
##   FALSE  6 69
##   TRUE  19  6
```

And easily pull the misclassifications from that table by looking at the diagonal (in this particular case).

```
sum(diag(table(x2 > 0.5, clas))) / length(x2)

## [1] 0.12
```

This suggests that even using the correct model would result in 13

```
## Installing package into 'C:/Users/xshi/Documents/R/win-library/4.0'
## (as 'lib' is unspecified)
## Error in contrib.url(repos, "source"):  trying to use CRAN without setting
a mirror
```

Now, we'll fit KNN for $k = 15$, $k = 10$, and $k = 1$. Take a look at the help file for 'knn' by entering '?knn' in the console. You can see some discussion surrounding ties in there (which we didn't really cover in lecture). The 'knn' function takes both a training set and a testing set. We'll give it 'x1' and 'x2' for the training set, and the testing set to look at MSE for train.

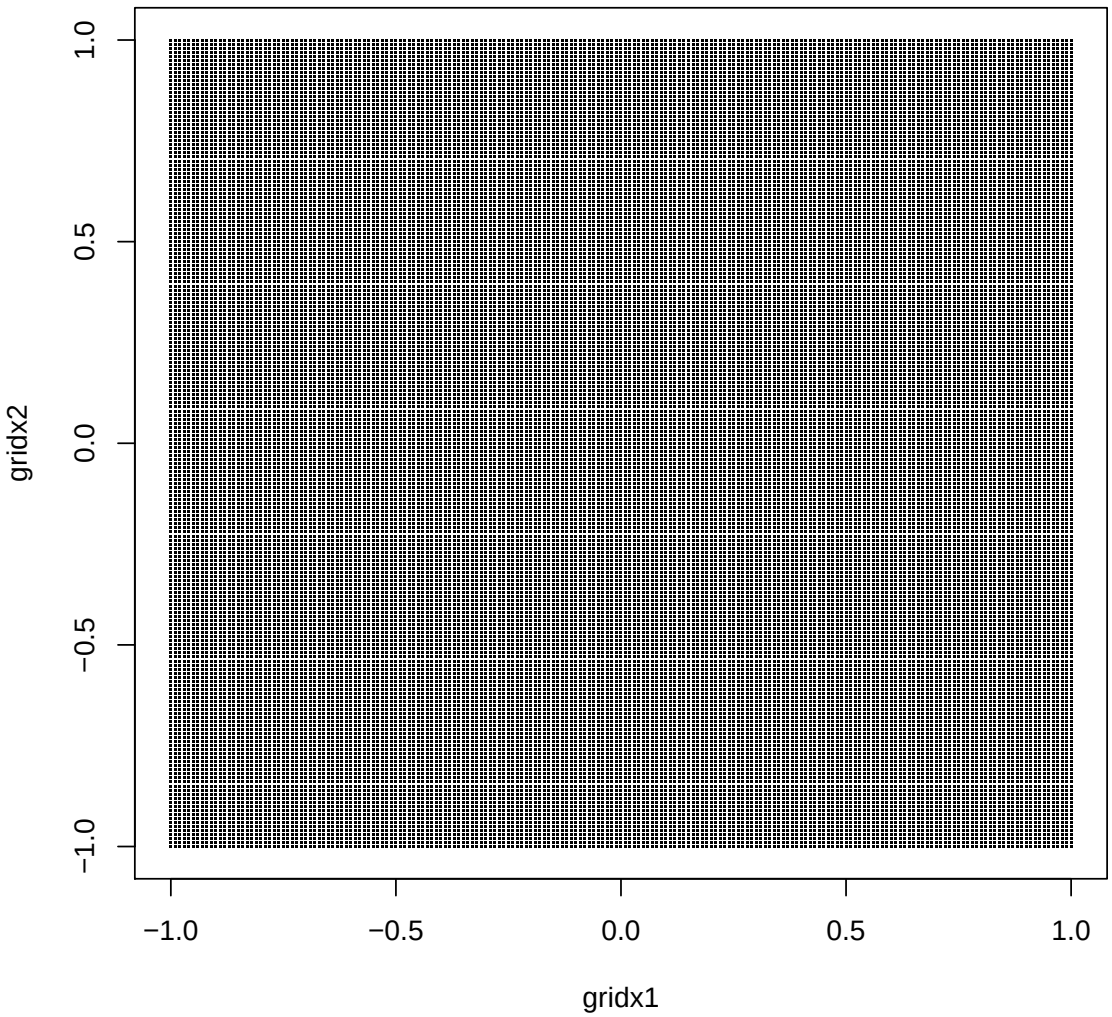
```
library("class")
mod15 <- knn(cbind(x1,x2), cbind(x1, x2), clas, k=15, prob=TRUE)
table(clas, mod15)

##     mod15
## clas  1  2
##    1 16  9
##    2  5 70

#now misclassifications are on the off diagonal, so...
(length(clas)-sum(diag(table(clas, mod15))))/length(clas)

## [1] 0.14

mod10 <- knn(cbind(x1,x2), cbind(x1, x2), clas, k=10, prob=TRUE)
(length(clas)-sum(diag(table(clas, mod10))))/length(clas)

## [1] 0.15

mod1 <- knn(cbind(x1,x2), cbind(x1, x2), clas, k=1, prob=TRUE)
(length(clas)-sum(diag(table(clas, mod1))))/length(clas)

## [1] 0
```

So misclassification rates of 14

Now let's give a big grid of values for the testing set in order to eventually recreate the contours shown in lecture.

```
gridseq <- seq(-1, 1, 0.01)
gridx1 <- rep(gridseq, each=length(gridseq))
gridx2 <- rep(gridseq, length(gridseq))
```

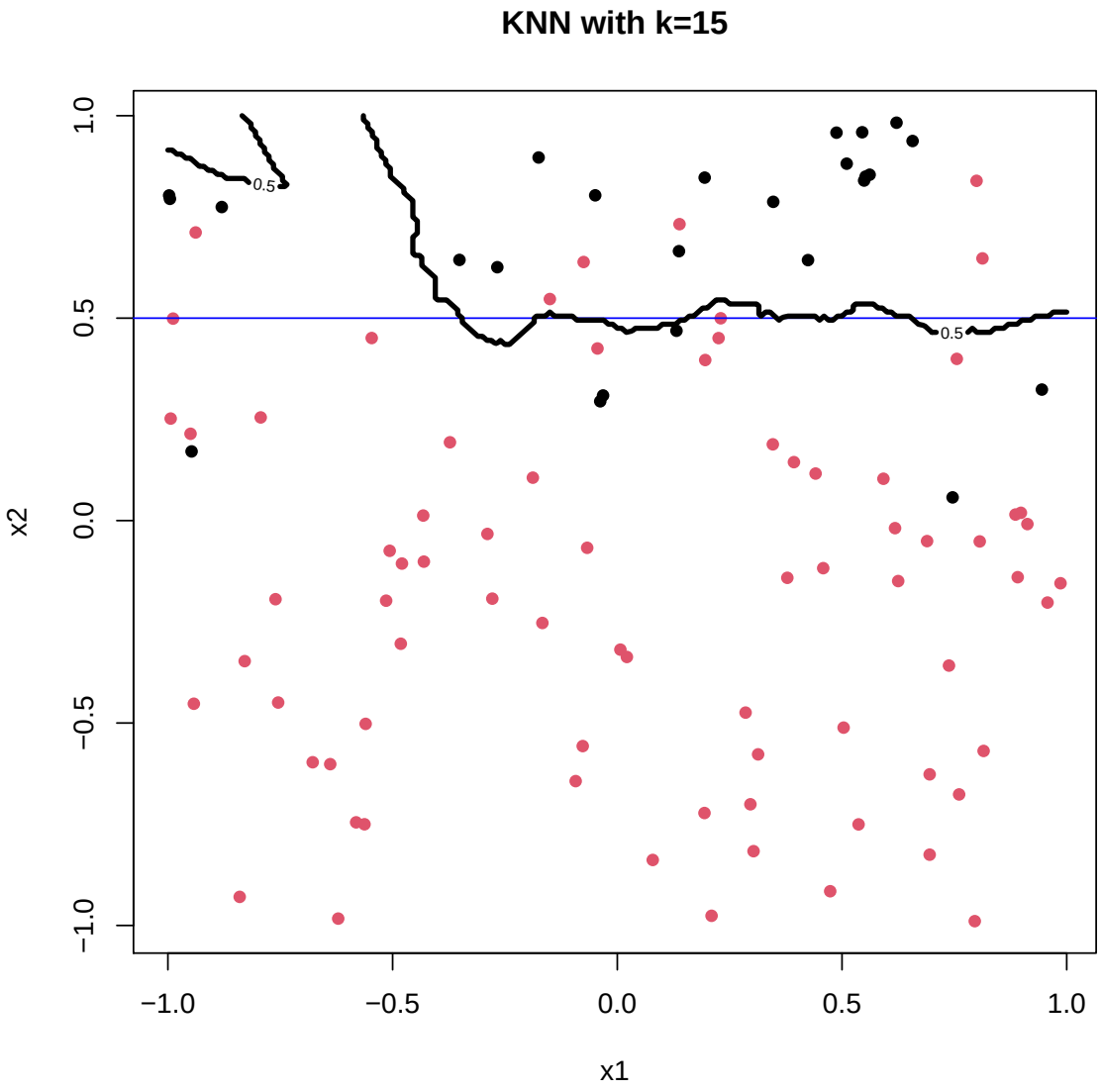
```r
#We are putting a point at each 0.01 increment in both x1 and x2
plot(gridx1, gridx2, pch=".")
```



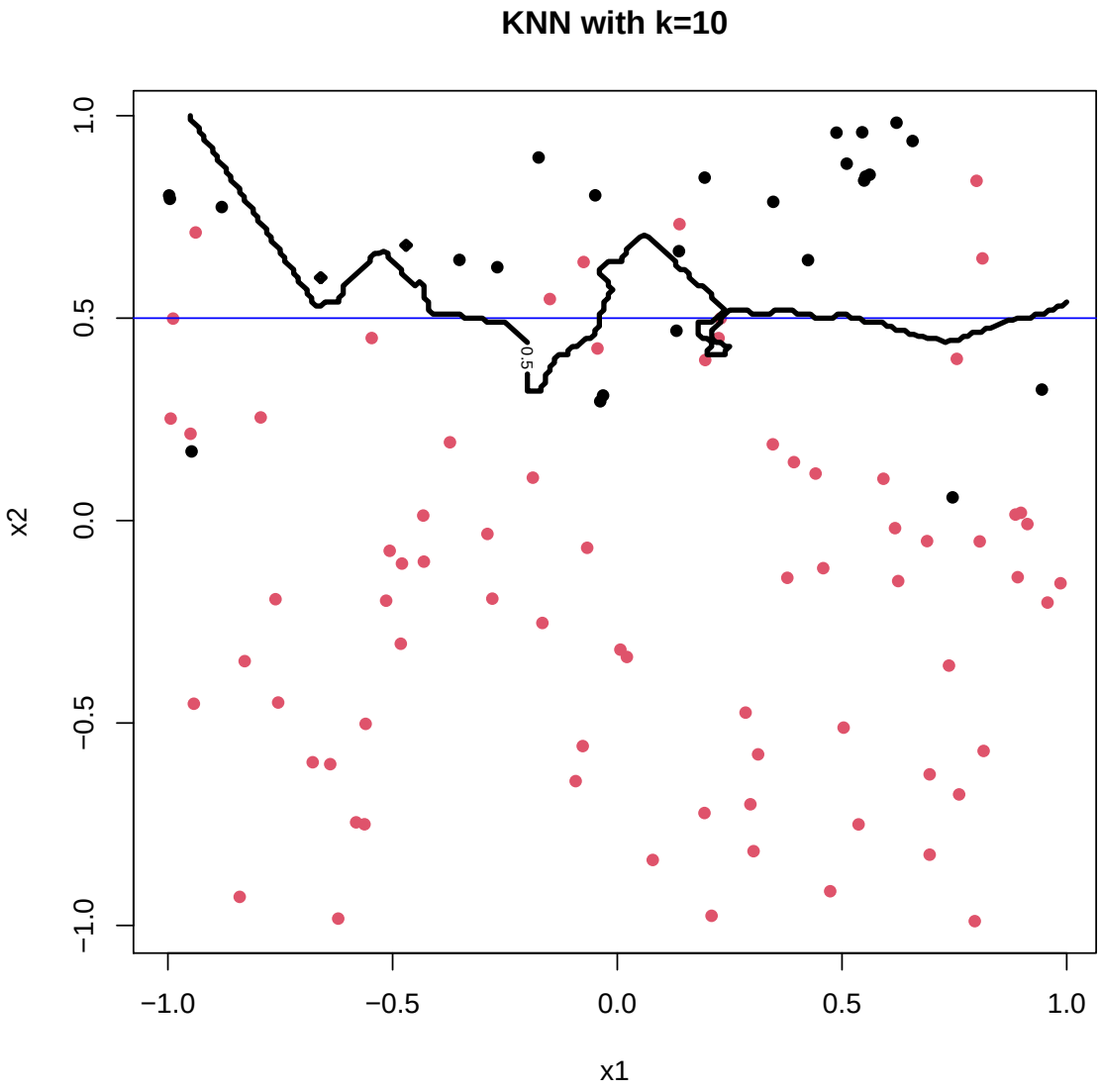

```r
mod15 <- knn(cbind(x1,x2), cbind(gridx1, gridx2), clas, k=15, prob=TRUE)
mod10 <- knn(cbind(x1,x2), cbind(gridx1, gridx2), clas, k=10, prob=TRUE)
mod1 <- knn(cbind(x1,x2), cbind(gridx1, gridx2), clas, k=1, prob=TRUE)
```

The commands for the decision boundaries are up on Connect — copy and paste:

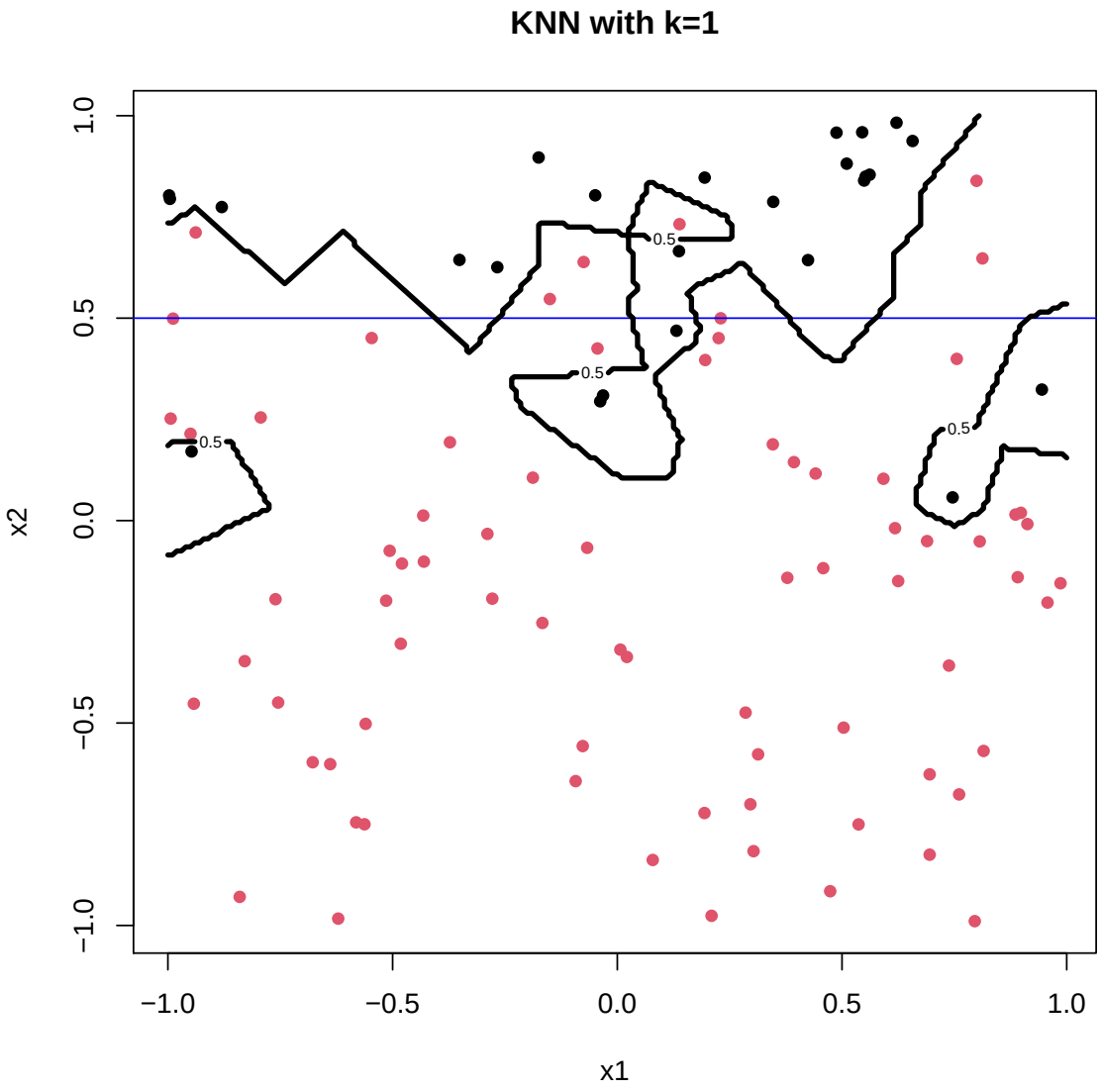
```r
plot(x1, x2, col = clas, pch=16, main="KNN with k=15")
abline(h = 0.5, col="blue")
conprob <- attr(mod15, "prob")
conpoints <- ifelse(mod15==1, conprob, 1-conprob)
conmat <- matrix(conpoints, length(gridseq), length(gridseq))
contour(gridseq, gridseq, t(conmat), levels=0.5, nlevels=1, add=TRUE, col="black", l
```




```
plot(x1, x2, col = clas, pch=16, main="KNN with k=10")
abline(h = 0.5, col="blue")
conprob <- attr(mod10, "prob")
conpoints <- ifelse(mod10==1, conprob, 1-conprob)
conmat <- matrix(conpoints, length(gridseq), length(gridseq))
contour(gridseq, gridseq, t(conmat), levels=0.5, nlevels=1, add=TRUE, col="black", l
```




```
plot(x1, x2, col = clas, pch=16, main="KNN with k=1")
abline(h = 0.5, col="blue")
conprob <- attr(mod1, "prob")
conpoints <- ifelse(mod1==1, conprob, 1-conprob)
conmat <- matrix(conpoints, length(gridseq), length(gridseq))
contour(gridseq, gridseq, t(conmat), levels=0.5, nlevels=1, add=TRUE, col="black", l
```

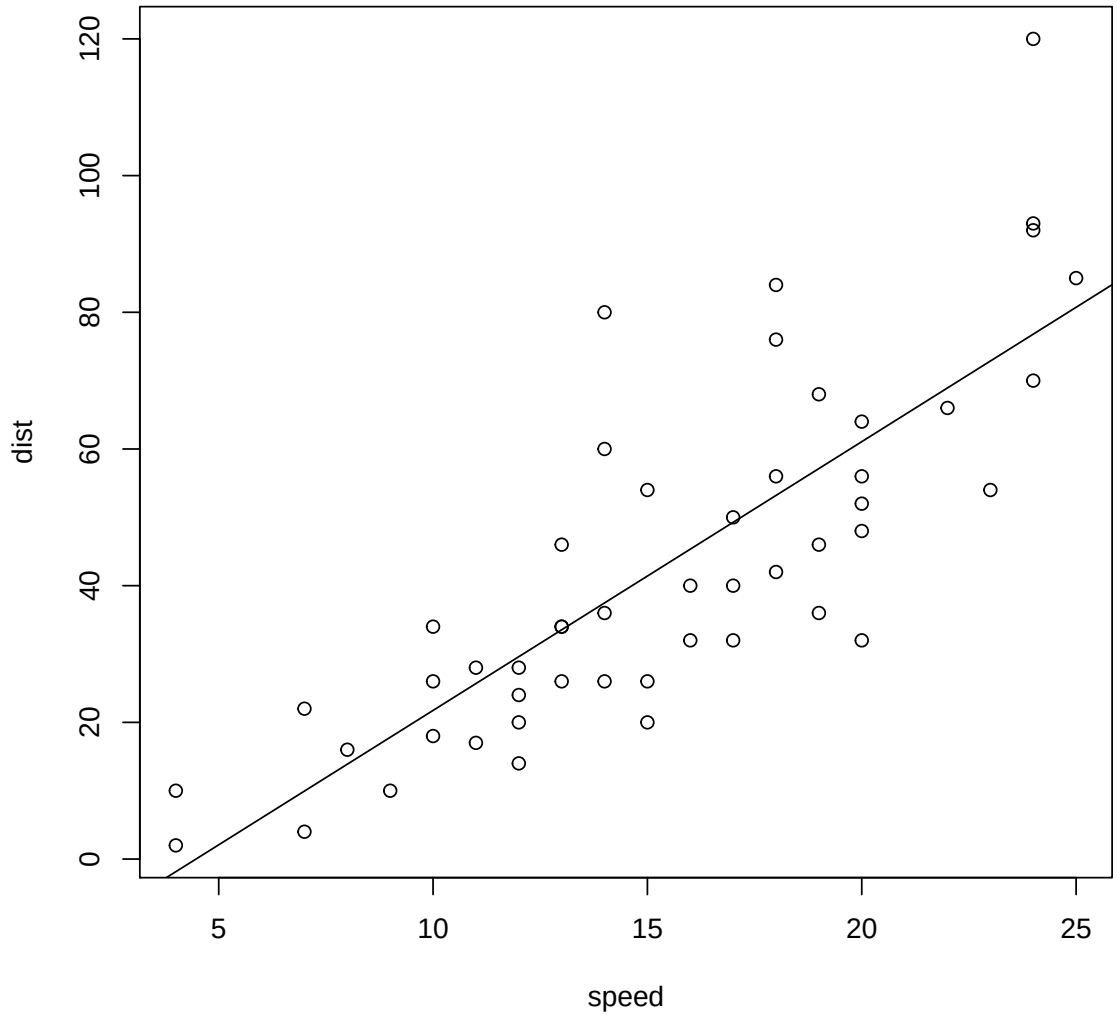



3. Simple Linear Regression

   Commands for the car example from lecture

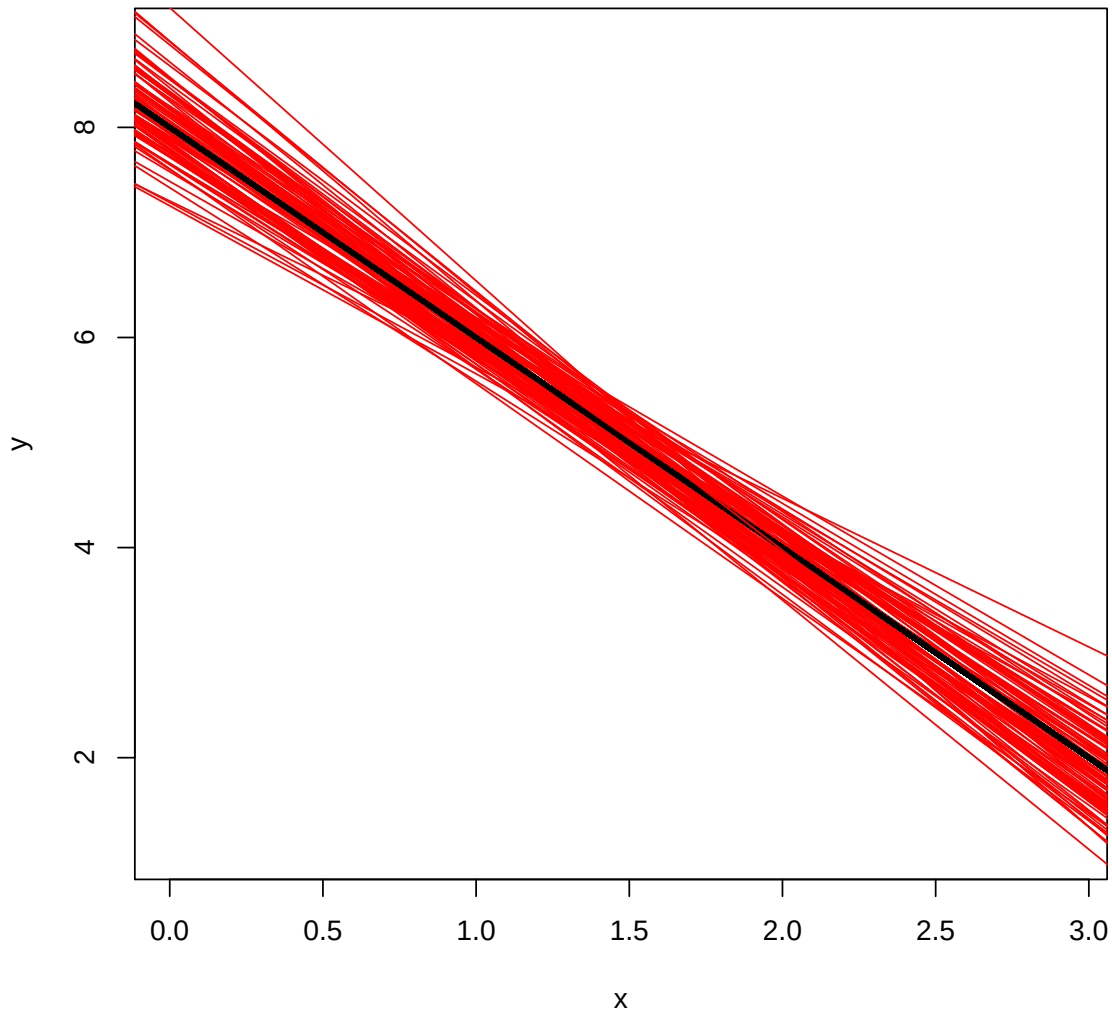
```
data(cars)
plot(cars)
attach(cars)
carlm <- lm(dist~speed)
summary(carlm)

##
## Call:
## lm(formula = dist ~ speed)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511,Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12

abline(carlm)
```



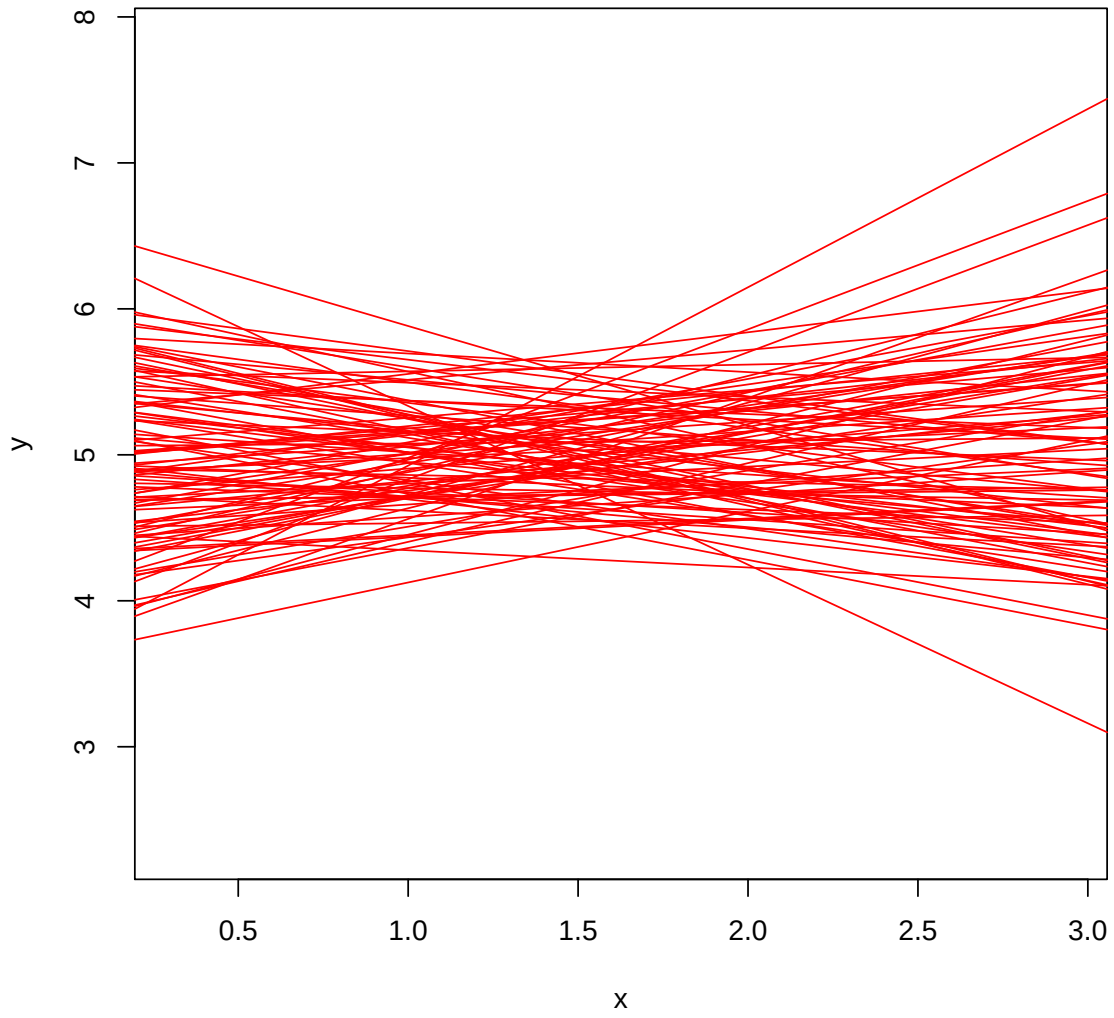

And the simulations to show model variance are on Connect — copy and paste:

```r
for(i in 1:100){
  x <- runif(30, 0,3)
  y <- -2*x+8 + rnorm(length(x))
  if(i==1) plot(x,y,type="n")
  abline(a=8, b=-2, lwd=3)
  simlm <- lm(y~x)
  abline(simlm, col="red")
}
```




```r
for(i in 1:100){
  x <- runif(30, 0, 3)
  y <- runif(30, 2, 8)
  if(i==1) plot(x,y,type="n")
  simlm <- lm(y~x)
  abline(simlm, col="red")
```

```
}
```




4. (2 marks) Provide an example (not used in class or the textbook) where classification would be used as the analysis. Describe the predictor(s), the response variable, and explain whether prediction or inference would be of primary interest.

5. (2 marks) Provide an example (not used in class or the textbook) where regression would be used as the analysis. Describe the predictor(s), the response variable, and explain whether prediction or inference would be of primary interest.

6. (2 marks) Suppose we knew that the Bayes' classifier was a boundary that was extremely non-linear. If we were using $K$-nearest neighbours as a classifier, would you expect a larger or smaller value of $K$ to provide a better approximation of the boundary? Explain.

7. (6 marks) You may not use the `knn` or `dist` commands in R for this question. You may perform calculations by hand (and attach to assignment) or manually in R (including commands and output in the document). Suppose we have the following observations:

| Observation | $Y$ | $X_1$ | $X_2$ | $X_3$ |
|---|---|---|---|---|
| 1 | Blue | 0 | 2 | 0 |
| 2 | Yellow | -2 | -1 | 0 |
| 3 | Yellow | -1 | 0 | 1 |
| 4 | Blue | 0 | 1 | 3 |
| 5 | Yellow | 1 | 1 | 1 |
| 6 | Yellow | 0 | 3 | 0 |

(a) (2 marks) Compute the Euclidean distance between all points and a new observation at the origin (0, 0, 0).

(b) (2 marks) Perform $K$-nearest neighbours classification for this new observation with $K$=1. What are the resulting classification probabilities at the origin? How would $K$NN classify the origin point?

(c) (2 marks)Perform $K$-nearest neighbours classification for this new observation with $K$=3. What are the resulting classification probabilities at the origin? How would $K$NN classify the origin point?