Lab 3

Regression Diagnositics and KNN Regression

Dr. Irene Vrbik

2023-09-25

Summary This lab will cover how to make predictions, analyze diagnostic plots, and identify potential problems in multiple linear regression. For more details, consults your textbook (ILSR2 Section 3.3.3. and Lab 3.6). You will also look at KNN regression, a non-parametric method for performing regression.

Learning outcomes

By the end of this lab students will be able to:

-

Prediction

To begin, lets start by fitting a linear regression model to the `Auto` dataset available in the **ISLR2** package. `fit1` attempts to explain gas mileage (in mpg `mpg`) using the predictor variable of the engines horsepower `horsepower`.

```r
library(ISLR2)
attach(Auto)
head(Auto)
```

```
  mpg cylinders displacement horsepower weight acceleration year origin
1  18         8          307        130   3504         12.0   70      1
2  15         8          350        165   3693         11.5   70      1
3  18         8          318        150   3436         11.0   70      1
4  16         8          304        150   3433         12.0   70      1
5  17         8          302        140   3449         10.5   70      1
6  15         8          429        198   4341         10.0   70      1
                       name
1 chevrolet chevelle malibu
2         buick skylark 320
3        plymouth satellite
4             amc rebel sst
5               ford torino
6          ford galaxie 500
```

```
fit1 <- lm(mpg~horsepower)
summary(fit1)
```

```
Call:
lm(formula = mpg ~ horsepower)

Residuals:
     Min       1Q   Median       3Q      Max 
-13.5710  -3.2592  -0.3435   2.7630  16.9240 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) 39.935861   0.717499   55.66   <2e-16 ***
horsepower  -0.157845   0.006446  -24.49   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.906 on 390 degrees of freedom
Multiple R-squared:  0.6059,    Adjusted R-squared:  0.6049 
F-statistic: 599.7 on 1 and 390 DF,  p-value: < 2.2e-16
```

Judging by the very low p-values for β_1 (slope), it appears that there a relationship between the predictor and the response. Since the sign for slope is negative ($\hat{\beta}_1$ =-0.1578) we expect this relationship to be negative, that is, as the horsepower of our engine increase, gas mileage tends to go down. This relationship would also appear to be relatively strong as indicated by the high R^2 value $= 0.6049$ (ie approximately 60% of the variance in the response is being explained by the simple model. To find the predicted mpg associated with a horsepower of say 90, we could use the `predict` function (see `?predict.lm` for the help file; note that the `newdata` argument **must** be a data frame data structure):

```
predict(fit1, data.frame("horsepower"=90))
```

```
       1
25.72984
```

We could have constructed this prediction "by hand", by obtaining the line of best fit and substituting in 90 for `x` but this obviously takes a little bit more work:

```
coef(fit1)
```

```
(Intercept)  horsepower
 39.9358610  -0.1578447
```

```
x <- 90
coef(fit1)["(Intercept)"] + coef(fit1)["horsepower"]*x
```

```
(Intercept)
   25.72984
```

It is also handy to note that we could see all the fitted $\hat{y}_i$ values using the following command:

```
fitted(fit1)
# (output suppressed to save space)
```

To see the corresponding residuals, use:

```
resid(fit1)
# (output suppressed to save space)
```

Confidence and Prediction Intervals

To get a prediction interval we use:

```r
predict(fit1, data.frame("horsepower"=90), interval="predict")
```

```
       fit      lwr      upr
1 25.72984 16.07076 35.38891
```

To get a confidence interval we use:

```r
predict(fit1, data.frame("horsepower"=90), interval="confidence")
```

```
       fit      lwr      upr
1 25.72984 25.20932 26.25035
```

N.B. when using `predict` your data need to be in a data frame and the predictors need to be named in the same way as your training data.

To create a band similar to the one we saw in lecture, you will need to create a sequence of plausible `x` values. For our purpose I will just take the observed range of `hosepower` values within our data set. While you can do these plots in base R, the code is greatly facilitated using the `ggplot` function from the **ggplot2** library:

```r
# Load necessary libraries
library(ggplot2)

# Create a new data frame for prediction intervals
new_data <- data.frame("horsepower" = seq(min(Auto$horsepower), max(Auto$horsepower), leng

# Calculate the prediction intervals and confidence intervals
predict_intervals <- predict(fit1, newdata = new_data, interval = "prediction", level = 0.
confidence_intervals <- predict(fit1, newdata = new_data, interval = "confidence", level =

# Combine the intervals with the new data
intervals_data <- cbind(new_data, predict_intervals, confidence_intervals)
# rename column names to avoid duplication
colnames(intervals_data)[c(2:7)] <- c("p_fit","p_low", "p_upr",
                                      "c_fit", "c_lwr", "p_upr")

# Create a plot using ggplot2
ggplot(intervals_data, aes(x = x, y = p_fit)) +
  geom_line(color = "blue") +  # Fitted values
  geom_ribbon(aes(ymin = p_lwr, ymax = upr), fill = "lightblue", alpha = 0.5) +  # Predict
  geom_ribbon(aes(ymin = lwr.ci, ymax = upr.ci), fill = "gray", alpha = 0.5) +  # Confiden
  labs(
    title = "Linear Regression Prediction and Confidence Intervals",
    x = "X-axis",
    y = "Y-axis"
  )
```

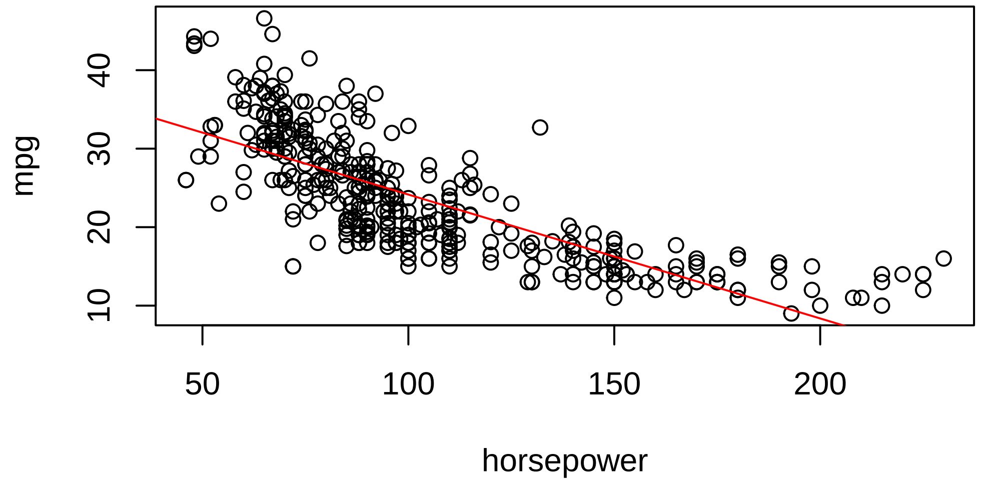
Diagnostics

While the conclusions we made above may seem appropriate based on the R output, we would be amiss to not check the assumptions of this model before we jump to any conclusions. Recall that the assumptions for a linear regression model are that:

1. There exists an (at least approximate) linear relationship between Y and X. %(aka, not some other relationship)
2. The distribution of ϵ_i has constant variance.
3. ϵ_i is normally distributed.
4. ϵ_i are independent of one another. For example, ϵ_2 is not affected by ϵ_1.

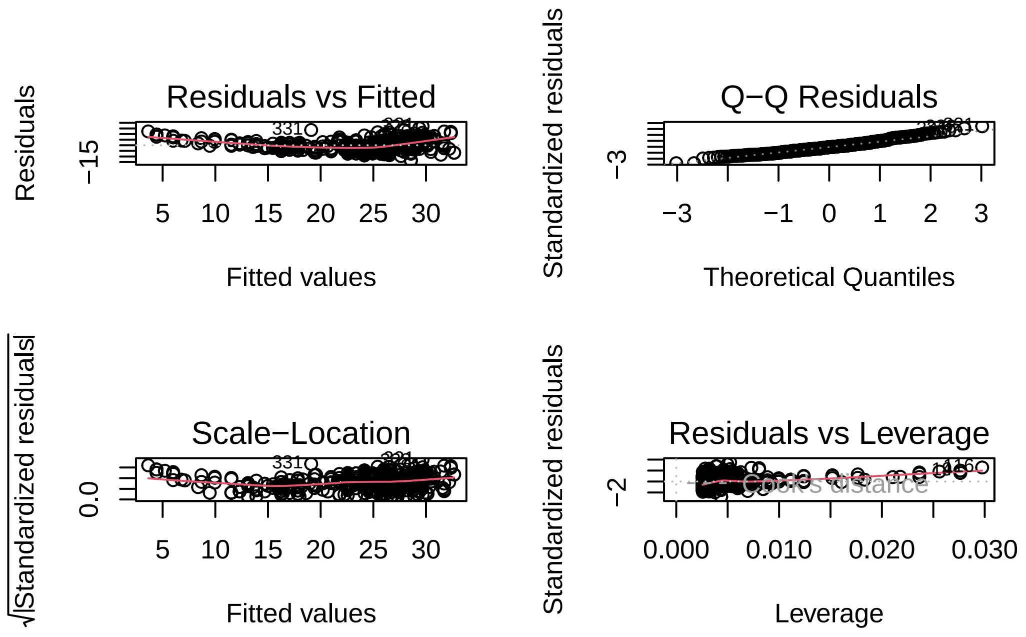
To check the first assumption, we may like to look at the scatterplot with the line of best fit. As demonstrated below, it would appear that the response and predictor variable have a curved relationship rather than a linear one.

```
plot(mpg~horsepower, data=Auto)
abline(fit1, col="red")
```



We could also explore the diagnostic plots produced when call `plot()` on the output of an `lm()` object. To see them all in one plot, let's change the panel layout of our plot window to store 4 figures (in this case in a 2 by 2 matrix) (for more details see `?par`). In other words we could call:

```r
par(mfrow=c(2,2))
plot(fit1)
```



A strong U-shaped pattern can be seen in the residual plot (located in top left corner) of the diagnostic plots below. This U-shape—and in general, any strong pattern—in the residuals indicates non-linearity in the data.

Transformations

We can extend the linear model to accomodate non-linearity by *transforming* predictor variables. For this particular example, the relationship between `mpg` and `horsepower` appears to be curved. Consequently, a modle which includes a *quadratic* `horsepower` term, may provide a better fit.

Note that ~ in R indicates a formula obect; for more details see `?formula`. In formula mode, `^` is used to specify interactions (eg. `^2` denotes all second order interactions from the preceeding term). So in order to use the caret to denote the arithmetic squared function, we need to *insulate* the term using the `I()` function. Alternatively, we could call/define a new variable, say `hp2`, that stores the squared values of `hoursepower` squared.

To fit this linear model—and YES this a linear model (i.e. a multiple linear regression model with $X_1 =$ `horsepower` and $X_2 =$ `horsepower`^{2} as predictors [1]—we have a choice between the following two set-ups:

```
# option 1
hp2 <- Auto$horsepower^2
auto2 <- cbind(Auto, hp2)
fit2a <- lm(mpg~horsepower+hp2)
summary(fit2a)
```

```

Call:
lm(formula = mpg ~ horsepower + hp2)

Residuals:
     Min       1Q   Median       3Q      Max 
-14.7135  -2.5943  -0.0859   2.2868  15.8961 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept) 56.9000997  1.8004268   31.60   <2e-16 ***
horsepower  -0.4661896  0.0311246  -14.98   <2e-16 ***
hp2          0.0012305  0.0001221   10.08   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.374 on 389 degrees of freedom
Multiple R-squared:  0.6876,    Adjusted R-squared:  0.686 
F-statistic:   428 on 2 and 389 DF,  p-value: < 2.2e-16
```

```
# option 2
fit2 <- lm(mpg~horsepower+I(horsepower^2))
summary(fit2)
```

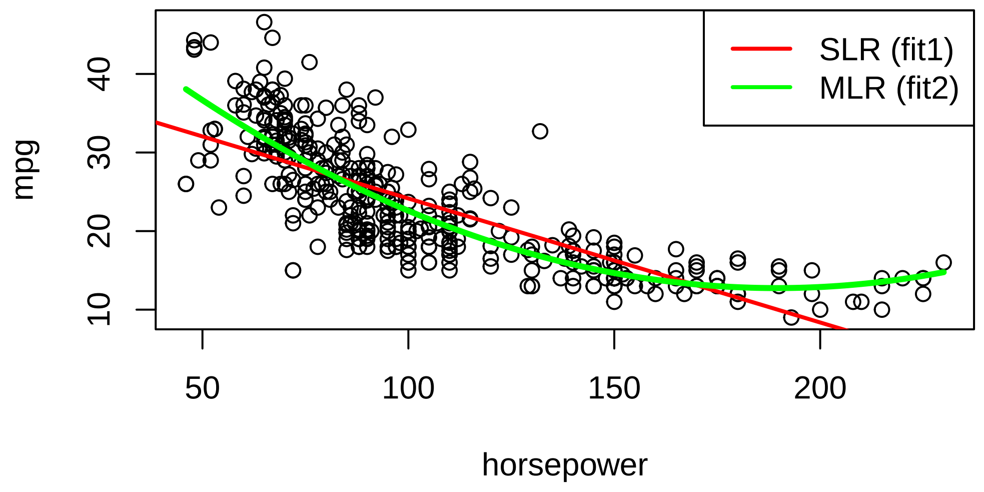
```

Call:
```

[1]recall that we will include all higher and lower order terms because of the hierarchy principal

```
lm(formula = mpg ~ horsepower + I(horsepower^2))

Residuals:
     Min       1Q   Median       3Q      Max
-14.7135  -2.5943  -0.0859   2.2868  15.8961

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)     56.9000997  1.8004268   31.60   <2e-16 ***
horsepower      -0.4661896  0.0311246  -14.98   <2e-16 ***
I(horsepower^2)  0.0012305  0.0001221   10.08   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.374 on 389 degrees of freedom
Multiple R-squared:  0.6876,    Adjusted R-squared:  0.686
F-statistic:   428 on 2 and 389 DF,  p-value: < 2.2e-16
```

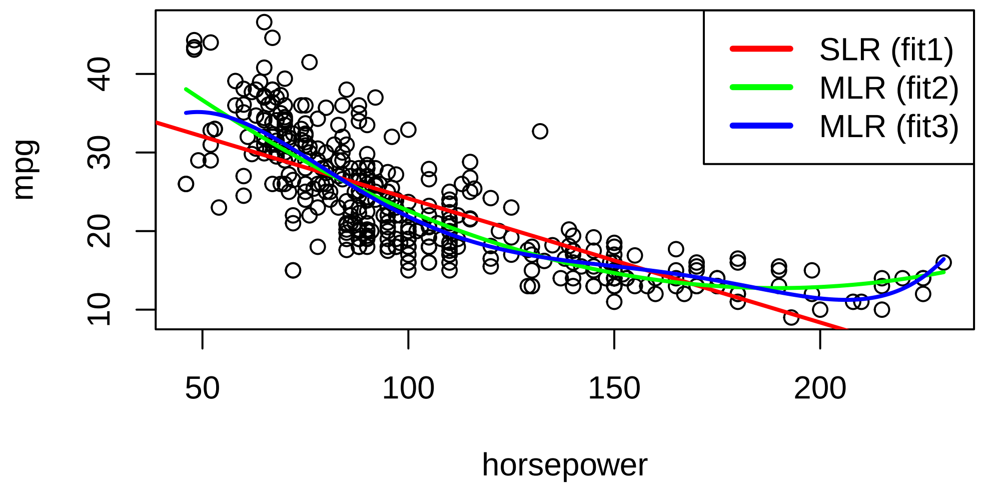
Lets compare the two fits on the scatterplot. It appears the fit with the squared horsepower is fitting the data much better:

```r
newdata <- data.frame(horsepower=seq(min(Auto$horsepower),
                      max(Auto$horsepower), length.out = 1000))
newdata$pred1 <- predict(fit2, newdata)
plot(mpg ~ horsepower, data = Auto)
abline(fit1, col="red", lwd=2)
lines(newdata$horsepower, newdata$pred1, col = "green", lwd=3)
legend('topright', lwd = 2,lty=1, col=c("red","green"),
       legend=c("SLR (fit1)","MLR (fit2)"))
```



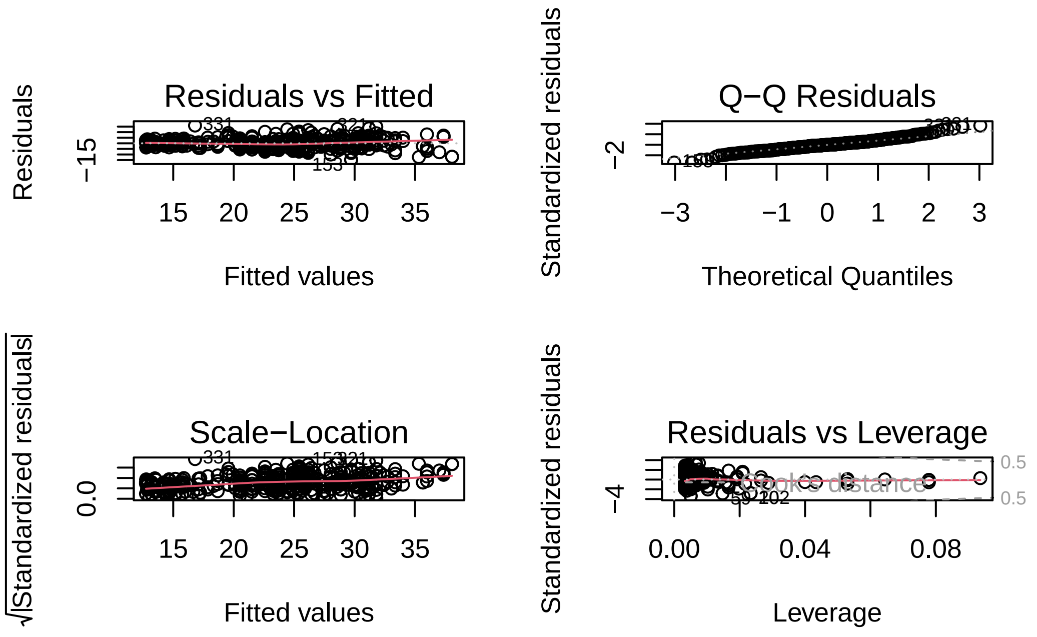
We could now fit higher order terms (eg. `horsepower`^{3}, `horsepower`^{4}, …), however, we run the risk of *overfitting*. For instance, `fit3` found below appears to be too wiggly. (This relates back to our bias and variance tradeoff).

```
fit3 <- lm(mpg~horsepower+I(horsepower^2)+I(horsepower^3)+I(horsepower^4)+I(horsepower^5))
newdata <- data.frame(horsepower=seq(min(Auto$horsepower),
                      max(Auto$horsepower), length.out = 1000))
newdata$pred1 <- predict(fit2, newdata)
newdata$pred2 <- predict(fit3, newdata)
plot(mpg ~ horsepower, data = Auto)
abline(fit1, col="red", lwd=2)
lines(newdata$horsepower, newdata$pred1, col = "green", lwd=2)
lines(newdata$horsepower, newdata$pred2, col = "blue", lwd=2)
legend('topright', lwd = 3,lty=1, col=c("red","green", "blue"),
       legend=c("SLR (fit1)","MLR (fit2)", "MLR (fit3)"))
```



Sticking with `fit2`, we can now replot our diagnostic plots to investigate assumptions. The first assumption of linearity can be checked in the residual plots. Note that the 3d plotting of this data in R is not as straight forward as their 2d counterparts. But we visualize the linear assumption in MLR with 2 predictors as having points lying approximately on a 2D plane in our 3D space (having x, y, and z variables equal to `horsepower`, `mpg` and `horsepower`^{2}, respectively).

```
par(mfrow=c(2,2))
plot(fit2)
```



While there is no more curvature in our residual plots, we are prehaps seeing some evidence of heteroscedasticity In other words, assumption 2 may be violated. In addition, our QQ-plot suggests that the residuals are not normally distributed (we are seeing high amounts of deviation at the extremities).

Outliers and high leverage points

As seen in the Scale-Location graph, it appears there may also been some outliers in our data. Using the ISLR recommendation, we can check the observations whose studentized residuals and observe which ones are greater than 3.

```
which(rstudent(fit2)>3)
```

```
321 328 331
321 328 331
```

It appears from the Residuals vs Leverage graph that a number of points could be high leverage points. To investigate the top, say 5, observations having the highest cooks distance, we could do the following:

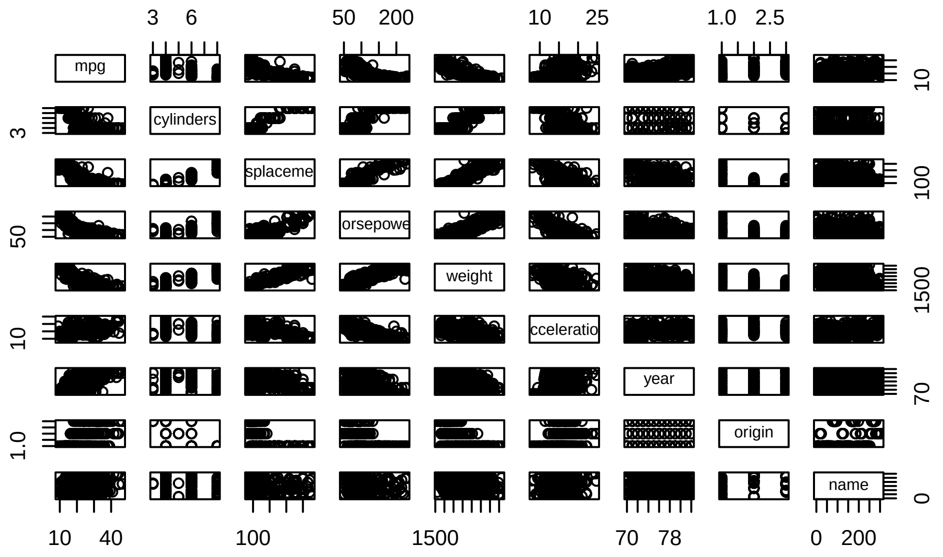
```
order(cooks.distance(fit2), decreasing = TRUE)[1:5]
```

```
[1]  20 102  59 331 321
```

Collinearity

Now lets consider a MLR model using more predictors from the `Auto` data from above. We discussed in lecture how some of these predictor variables might be correlated with one another. To investigate this, lets have a look at a scatterplot matrix which includes all of the variables in the data set using the `pairs()` function.

```
pairs(Auto)
```



We can see that `horespower` and `displacement` for example are highly correlated. We might also decide to look at the matrix of correlations between the variables using the function `cor()`.

```
# remove "name" (since it is not numeric)
cor(Auto[,names(Auto)!="name"])
```

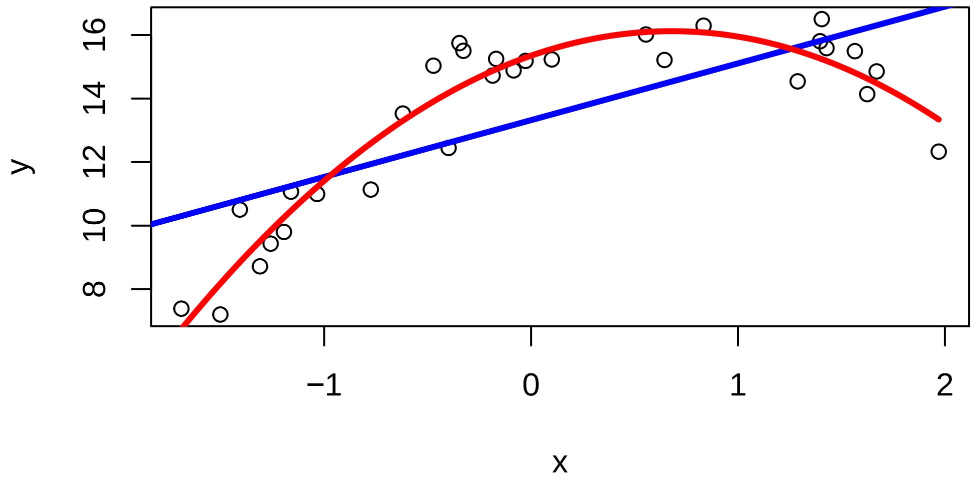
```
                    mpg  cylinders displacement horsepower     weight
mpg           1.0000000 -0.7776175   -0.8051269 -0.7784268 -0.8322442
cylinders    -0.7776175  1.0000000    0.9508233  0.8429834  0.8975273
displacement -0.8051269  0.9508233    1.0000000  0.8972570  0.9329944
horsepower   -0.7784268  0.8429834    0.8972570  1.0000000  0.8645377
weight       -0.8322442  0.8975273    0.9329944  0.8645377  1.0000000
acceleration  0.4233285 -0.5046834   -0.5438005 -0.6891955 -0.4168392
year          0.5805410 -0.3456474   -0.3698552 -0.4163615 -0.3091199
origin        0.5652088 -0.5689316   -0.6145351 -0.4551715 -0.5850054
             acceleration       year     origin
mpg             0.4233285  0.5805410  0.5652088
cylinders      -0.5046834 -0.3456474 -0.5689316
displacement   -0.5438005 -0.3698552 -0.6145351
horsepower     -0.6891955 -0.4163615 -0.4551715
weight         -0.4168392 -0.3091199 -0.5850054
acceleration    1.0000000  0.2903161  0.2127458
year            0.2903161  1.0000000  0.1815277
origin          0.2127458  0.1815277  1.0000000
```

Notice that the correlation between `horespower` and `displacement` is very high (0.8972570) as is the correlation between `cylinders` and `displacement` is very high (0.9508233).

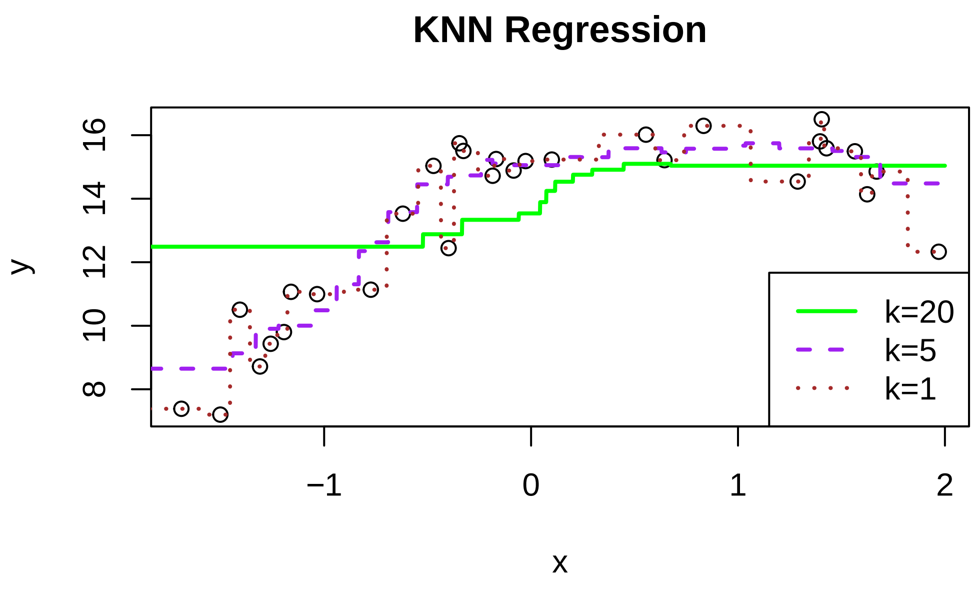
KNN Regression

Below is the example from our KNN Regression example. There was some discussion in class about the best way we might choose K. In practice the most common way to do this is to use *cross-validation*. While this will be covered later on in the course, for now, one solution we might use is to simply plot the predicted values for each value of K and choose the model which is not too jagged or too bias. While this solution may be unsatisfactory to some, any other method we could dream up we simply be set aside once we learn about cross-validation.

```r
# Non-linearity example
set.seed(3531)
x <- runif(30,-2,2)
y <- 15+2.3*x-1.5*x^2+rnorm(30)
plot(y~x)
linmod <- lm(y~x)
abline(linmod, col="blue", lwd=3)

x2 <- x^2
quadmod <- lm(y~x+x2)

# Manually plot curve using coefficients
curve(15.35+2.27*x-1.67*x^2, add=TRUE, col="red", lwd=3)
```



```
library(FNN)
seqx <- seq(from=-2, to=2, by=0.001)
plot(y~x, main="KNN Regression")
knnr <- knn.reg(x, y=y, test=cbind(seqx), k=20)
lines(seqx, knnr$pred, col="green", lwd=2)
knnr <- knn.reg(x, y=y, test=cbind(seqx), k=5)
lines(seqx, knnr$pred, col="purple", lwd=2, lty=2)
knnr <- knn.reg(x, y=y, test=cbind(seqx), k=1)
lines(seqx, knnr$pred, col="brown", lwd=2, lty=3)
legend("bottomright", legend = c("k=20", "k=5", "k=1"), lwd=2, lty=1:3, col=c("green", "pu
```


KNN Regression
y
x
k=20
k=5
k=1