Smoothing

Jennifer Wilcock

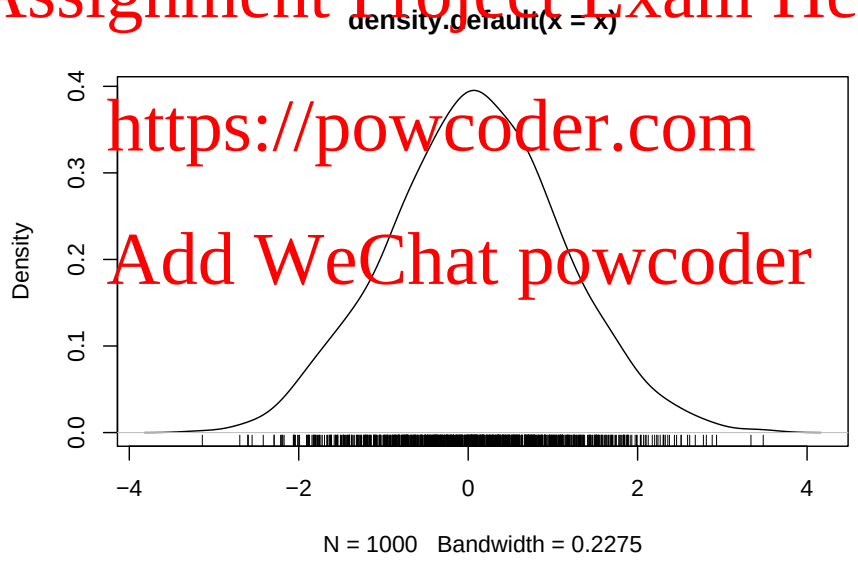
STAT221 2020 S2

Reminder: Density estimation

We focussed on single variables, and looked at two distinct approaches to exploring and understanding the structure of the distribution of the variable being considered:

1. we considered histograms and some methods for 'smoothing' histograms, in particular frequency polygons and ASH plots,
2. we then considered kernel density estimation, which takes a distinctly different approach of fitting a kernel around each individual datapoint, and then summing the contribution to the density of each of these kernels.

```
set.seed(35)
x = rnorm(1000, 0, 1)
k = density(x)  # default is Gaussian/Normal kernel
plot(k)
rug(x)
```

Both a kernel density estimator and a smoothed histogram give a smoothed picture of the density of the variable, using the individual data points in the observed or simulated sample.

The goal of such a smoother is to smooth out the fine details of the sample, which we can assume are the result of sample to sample variability, while allowing the detail of the general structure of the sample to remain. We then assume that this general structure is essentailly that of the population from which the sample was drawn.

Smoothing in the context of regression

Another important context in which smoothing is important is in the context of regression type problems.

In a simple regression type problem, there are two variables:

- an explanatory (or independent) variable, usually called x

- a response (or dependent) variable, usually called y.

In regression we use the explanatory variable x to predict the response variable y, and we usually do this by assuming some form of relationship between x and y.

The simplest relationship is linear, where we assume a known relationship between x and y, for example a straight line:

$$y_i = \alpha + \beta x_i + e_i$$

where α and β are unknown *parameters* that we estimate using the data, and the errors e_i are assumed to come from a Normal distribution. You will have looked at this in STAT101.

In general, however, a regression line can be assumed to be *any* function, which we can write as

$$y_i = g(x_i) + e_i$$

and g is an unknown *function* that we estimate using the data. We can still assume the errors come from a Normal distribution, if we wish - what we will focus on here is that it is the function itself (rather than some parameters of a previously assumed function) that we estimate using the data.

This is called nonparametric regression.

The type of regression considered in STAT101, STAT201, STAT202, STAT318, at school, etc, is called parametric regression and falls under the general side of being based in analytical or mathematical approaches. Nonparametric regression is an essentially computational approach - it wouldn't exist without the existance of computers.

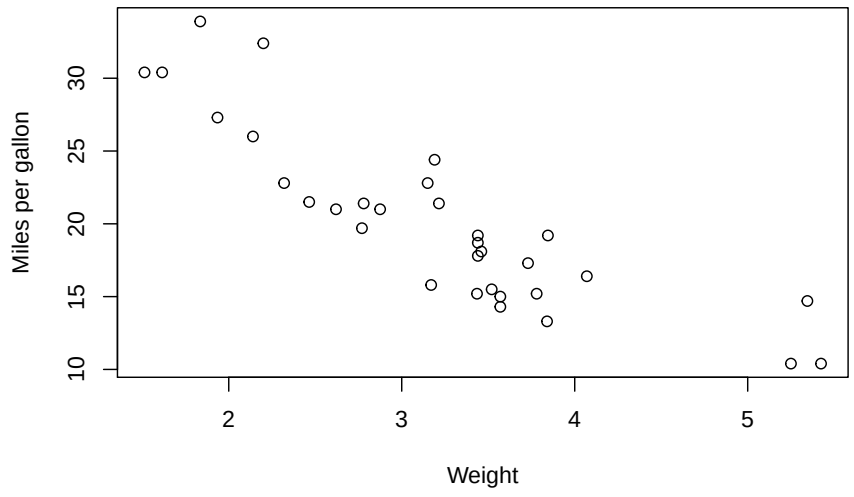
With a smoother, we assume that this unknown regression function is smooth so we assume it is differentiable. However in practice only some smoothing methods use the derivatives, and other methods don't.

Scatter plots in regression

Before doing a regression, we usually plot the two variables against each other in a scatter plot.

For example, we will think about the following data:

```
data(mtcars)

plot(mtcars$wt, mtcars$mpg, xlab = "Weight", ylab = "Miles per gallon")
```



The running mean smoother

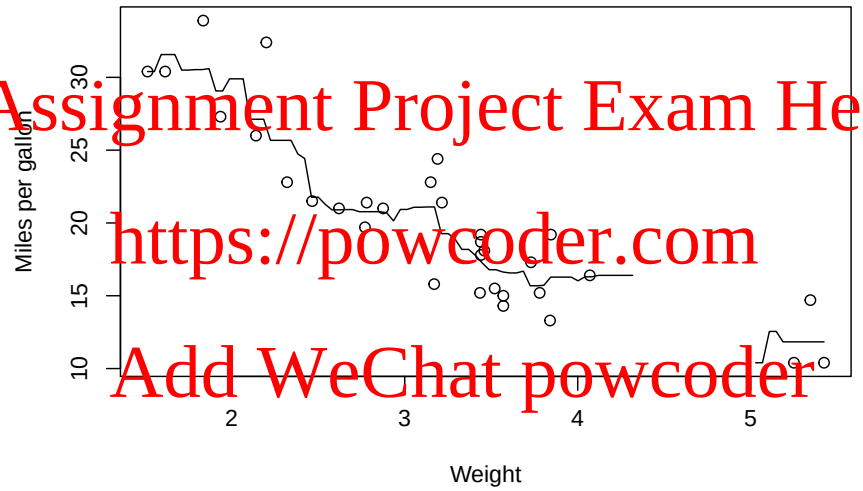
The running mean/running average/moving average is the simplest smoother.

For each given point x, as we move across tha range of the x variable, we average the values of y_i for each x_i that lies within a distance h of x.

As before, h is called the *bandwidth* or *smoothing parameter*. We call the interval $(x - h, x + h)$ the smoothing window.

So we gradually move the smoothing window across the plot, and we take the average y_i value for all of the points that lie within the window. These average values, when plotted, form a 'smoothed' line that we can plot.

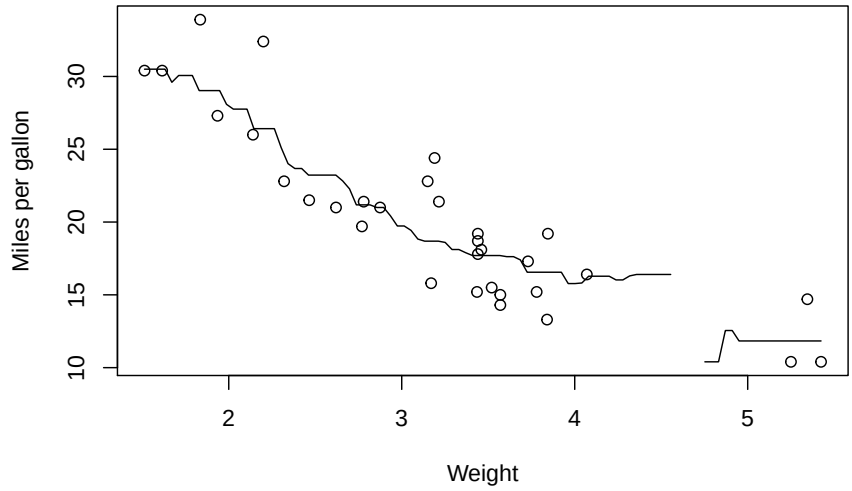
```
data(mtcars)

plot(mtcars$wt, mtcars$mpg, xlab = "Weight", ylab = "Miles per gallon")
k = ksmooth(mtcars$wt, mtcars$mpg, bandwidth = 0.5)
lines(k, col = 1)
```



Here the bandwidth was 0.5, which is quite wide relative to the range of the data for `Weight`. Even so, it doesn't look very smooth.

We could try making the bandwidth wider:

```
data(mtcars)

plot(mtcars$wt, mtcars$mpg, xlab = "Weight", ylab = "Miles per gallon")
k = ksmooth(mtcars$wt, mtcars$mpg, bandwidth = 1)
lines(k, col = 1)
```



It's 'smoother', but still not smooth.

There is a key problem with this smoother, which is why it doesn't really get used in practice.

The reason it isn't smooth is that we are using an all-or-nothing in-or-out decision about which y_i values to use in the average. A given y_i value is either included or not included, and immediately switches from being in to being out of the average calculation (or vice versa). As a result the average value suddenly jumps, as data points either suddenly enter or leave the smoothing window.

This is because the kernel being used here is the 'box' kernel, which is like the rectangular kernel in kernel density estimation - it's never going to be truely smooth.

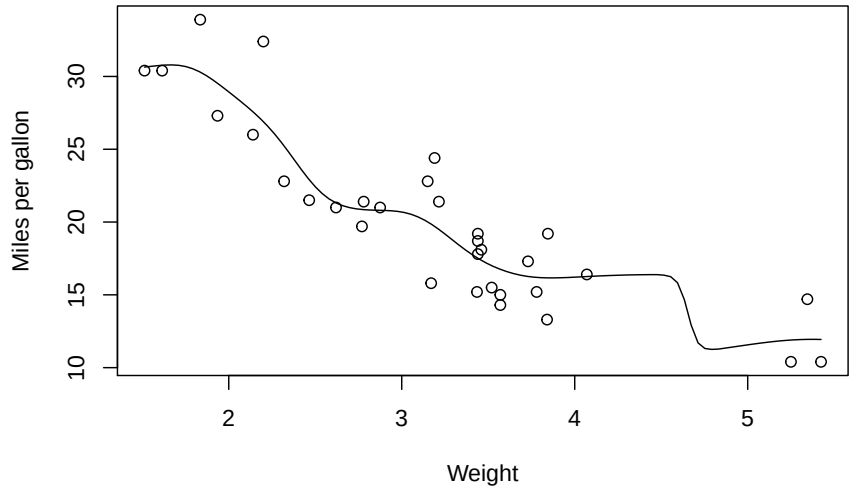
General kernel smoothing

Instead, we use kernel smoothers that are similar to those used in kernel density estimation.

We use these kernels to calculate a 'weighted' average.

For example, we can use a Gaussian kernel with the original bandwidth of 0.5:

```
data(mtcars)

plot(mtcars$wt, mtcars$mpg, xlab = "Weight", ylab = "Miles per gallon")
k = ksmooth(mtcars$wt, mtcars$mpg, "normal", bandwidth = 0.5)
lines(k, col = 1)
```



which is now 'smooth'.

We estimate $g(x)$ by the *kernel regression estimate*

$$\widehat{g}(x) = \frac{\sum_{i=1}^{n} y_i w\left(\frac{x-x_i}{h}\right)}{\sum_{i=1}^{n} w\left(\frac{x-x_i}{h}\right)}$$

where w is a known, fixed non-negative function that is symmetric about zero, called the kernel.

(Recall that to find a *kernel density estimate* we called the kernels K, but they had the same properties.)

How does it work?

This is an example of a *linear smoother*.

Each $\widehat{g}(x)$ is a weighted average of the y_i, which we can write as

$$p_1 y_1 + \cdots + p_n y_n$$

where the p_i sum to one. The p_i are the "weights"

$$p_i = \frac{w\left(\frac{x-x_i}{h}\right)}{\sum_{j=1}^{n} w\left(\frac{x-x_j}{h}\right)}$$

.

When the kernel is the 'box' function, so that the kernel smoother is the running average we looked at earlier, then we define the kernel w as

$$w(x) = \begin{cases} 0, & x < 1 \\ 1, & -1 \leq x \leq 1 \\ 0, & 1 < x. \end{cases}$$

Using the Gaussian/Normal kernel, for example, then

$$w(x) = \exp\left(-x^2/2\right).$$

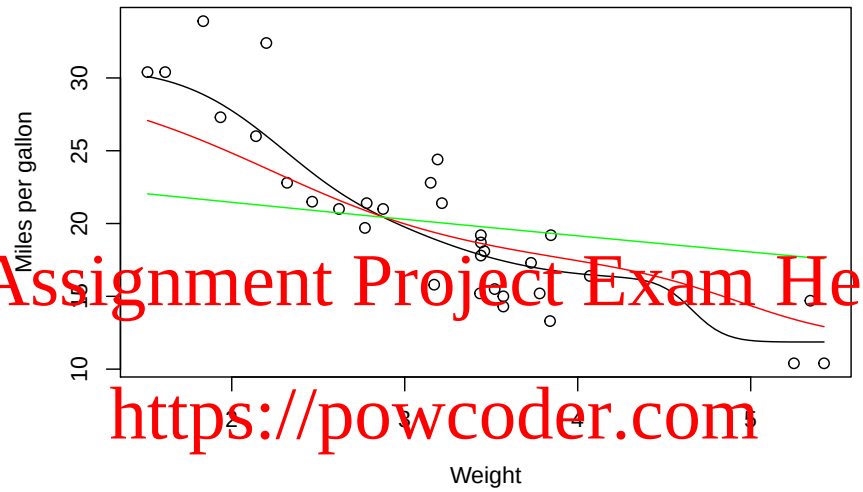
The most common kernels apply small weight to points far away, and relatively large weights to points close to the value of x being considered. Nearby points therefore have the most influence on the weighted estimate, and far away points the least influence.

Changing the bandwidth affects how many points are included in the claculation of the average. The more points that are included, the smoother the curve will be.

Clearly, there will be some optimal bandwidth for a specified kernel with a given dataset.

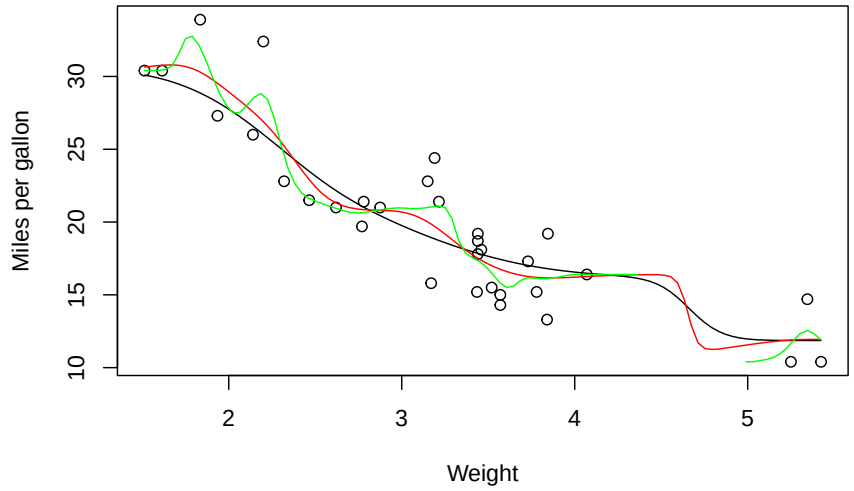
We can look at the effect of increasing the bandwidth on our data, using the Gaussian/normal kernel:

```
data(mtcars)
with(mtcars, {plot(wt, mpg, xlab = "Weight", ylab = "Miles per gallon")})

with(mtcars, {lines(ksmooth(wt, mpg, "normal", bandwidth = 1), col = "black")})
with(mtcars, {lines(ksmooth(wt, mpg, "normal", bandwidth = 2), col = "red")})
with(mtcars, {lines(ksmooth(wt, mpg, "normal", bandwidth = 5), col = "green")})
```



and of decreasing the bandwidth:

```
data(mtcars)
with(mtcars, {plot(wt, mpg, xlab = "Weight", ylab = "Miles per gallon")})

with(mtcars, {lines(ksmooth(wt, mpg, "normal", bandwidth = 1), col = "black")})
with(mtcars, {lines(ksmooth(wt, mpg, "normal", bandwidth = (1/2)), col = "red")})
with(mtcars, {lines(ksmooth(wt, mpg, "normal", bandwidth = (1/5)), col = "green")})
```



Some general points about kernel smoothers

The kernel estimate $\hat{g}(x)$ of the true unknown regression function $g(x)$ will be smooth if the kernel used, $w(x)$ is smooth.

As with kernel density estimators, the bandwidths are not directly comparable between different kernels. For a given kernel, a larger bandwidth always results in a smoother curve, but it is less meaningful to compare the same bandwidth across different kernels.

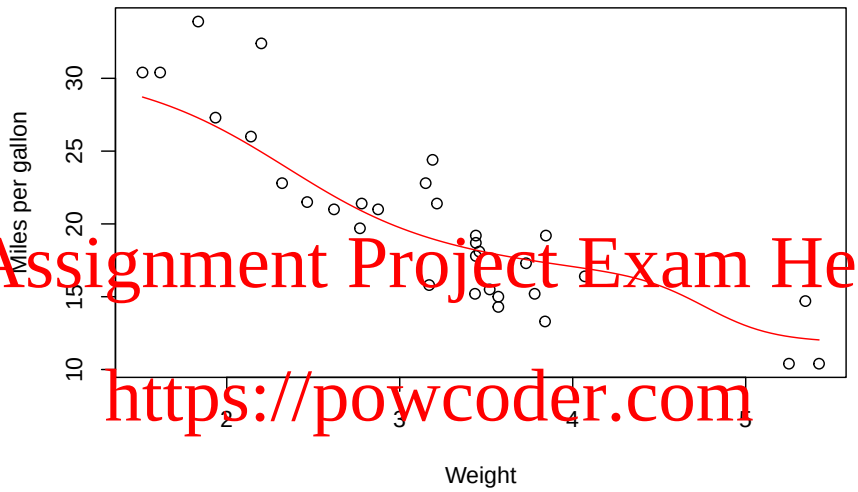
Start of lecture 6

A problem with kernel smoothers

Kernel smoothing is easy to understand and simple to implement. However it has a number of weaknesses, the most important of which is what is called *boundary bias* or *edge effects*, leading to poor behaviour at the edges of a plot.

For example, we can see in this plot:

```
data(mtcars)
with(mtcars, {plot(wt, mpg, xlab = "Weight", ylab = "Miles per gallon")
              lines(ksmooth(wt, mpg, "normal", bandwidth = 1.5), col = "red")})
```



that over most of the range of the data the smoother generally captures the main trend in the data, which is the behaviour that we wish to see.

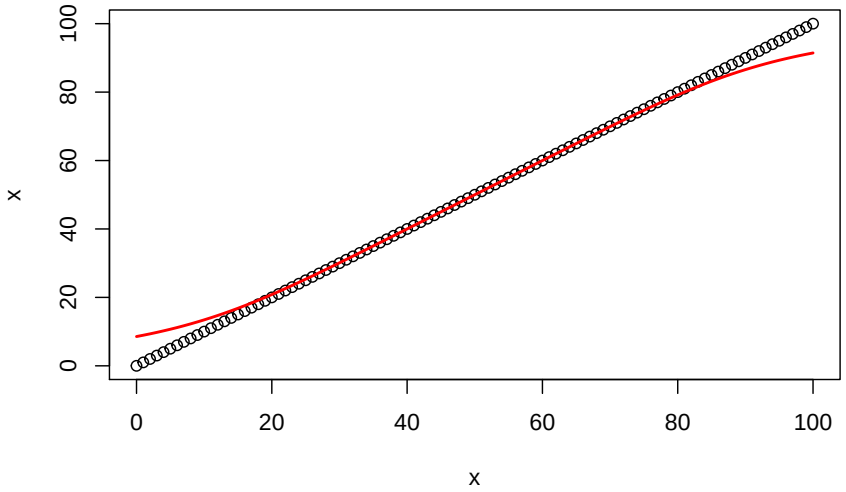
However at the left hand edge of the plot, at values of `Weight` less than about 2.2, we see that all of the datapoints lie above the smoother, rather than the smoother passing roughly through the centre of these points.

If we were to use this smoother to make predictions of the fuel economy of vehicles with weights of less than around 2.2 tons, then these predictions would systematically underestimate the fuel efficiency of these cars, which is a problem.

But ... maybe it's just that we should have used a better smoother ...?

So, as another example, consider using a kernel smoother on some perfectly linear data:

```
x = seq(0, 100)
plot(x, x)
lines(ksmooth(x, x, "normal", bandwidth = 30), col = "red", lwd = 2)
```



The smoother follows the data points closely between x values of about 30 to 70. Between about 20 and 30, and 70 and 80 the smoother starts to slowly drift away from the points, and below 20 and above 80 it definitely moves away from the points it is supposed to be smoothing.

What's going wrong?

We have said that the weights must sum to one.

However if x is within a bandwidth distance h of either edge, then the sum of the weights within the reduced interval is less than one, and $\widehat{g}(x)$ becomes increasingly 'biased', or further from the true $g(x)$ (supposing we know what $g(x)$ is, which we do for the straight line plot).

Various methods of correcting for edge effects have been developed. A simple approach is to re-weight the weights so that they again sum to one, or to use modified ('boundary') kernels in the edge regions of the plot.

Many modified kernel smoothers have been proposed, however alternative approaches to smoothing exist which don't have this problem.

Local regression smoothing

An improvement is to use a method called *local polynomial smoothing*, or *local regression smoothing*, or as *lowess* or *loess smoothing*, and which is a very commonly used approach.

Here, we fit many individual weighted regressions to the datapoints 'locally', centred at each value of x and only using datapoints within the given bandwidth. When there is no weighting applied, we have standard least squares regression. We then find the optimal fit for a regression line by minimising the sum of the squared residuals, where each residual is the distance between the observed y value of a datapoint and the $\widehat{y}$ predicted by the regression.

When a least squares regression is linear, then the equation of the line is given by

$$y_i = \alpha + \beta x_i + e_i$$

and so we need to find the values of α and β that makes

$$\sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2$$

as small as possible. These values are called $\widehat{\alpha}$ and $\widehat{\beta}$.

In a weighted least squares regression we do the same thing except that each residual is weighted in some way.

With local regression the weighting is done using a kernel which, as before, is a function we call w:

$$w_i = w\left(\frac{x - x_i}{h}\right).$$

We therefore now need to find the values of α and β that makes

$$\sum_{i=1}^{n} w\left(\frac{x - x_i}{h}\right)(y_i - \alpha - \beta x_i)^2$$

as small as possible. Again, these values are called $\widehat{\alpha}$ and $\widehat{\beta}$.

The estimate of the true function $g(x)$ is then given by:

$$\widehat{g}(x) = \widehat{\alpha} + \widehat{\beta}x$$

and we do this calculation at every value of x where we want to evaluate $\widehat{g}(x)$.

We can also fit local regressions that are of higher order, such as quadratic and cubic.

In R

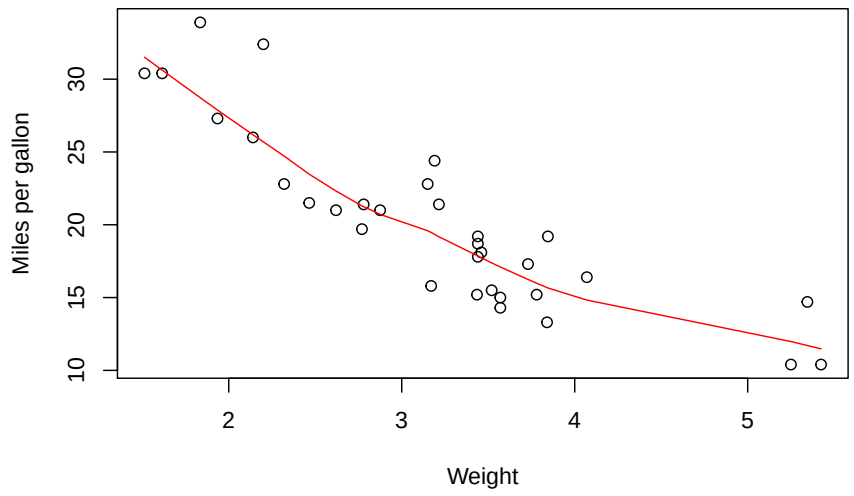
In R there are two well used implementations of local regression smoothers: `lowess` and `loess`. `loess` is more recent and has more features than `lowess`, which is the original implementation. For example, it works with more than one explanatory variable. However, `loess` is much slower than `lowess`, and sometimes fails when `lowess` works, so both are still available to use.

With one explanatory variable, they are essentially the same, however they have very different default settings. In particular, by default `lowess` does linear regression, while `loess` does quadratic regression by default.

The implementation of `lowess` in R also adds a second level of weighting that make the smoother more *robust*, which means that it is less sensitive to outliers. Here additional fits of the smoother are found where the weights used are modified based on the size of the residuals in the previous fit of the smoother.

Here's the example data smoothed using `lowess`:

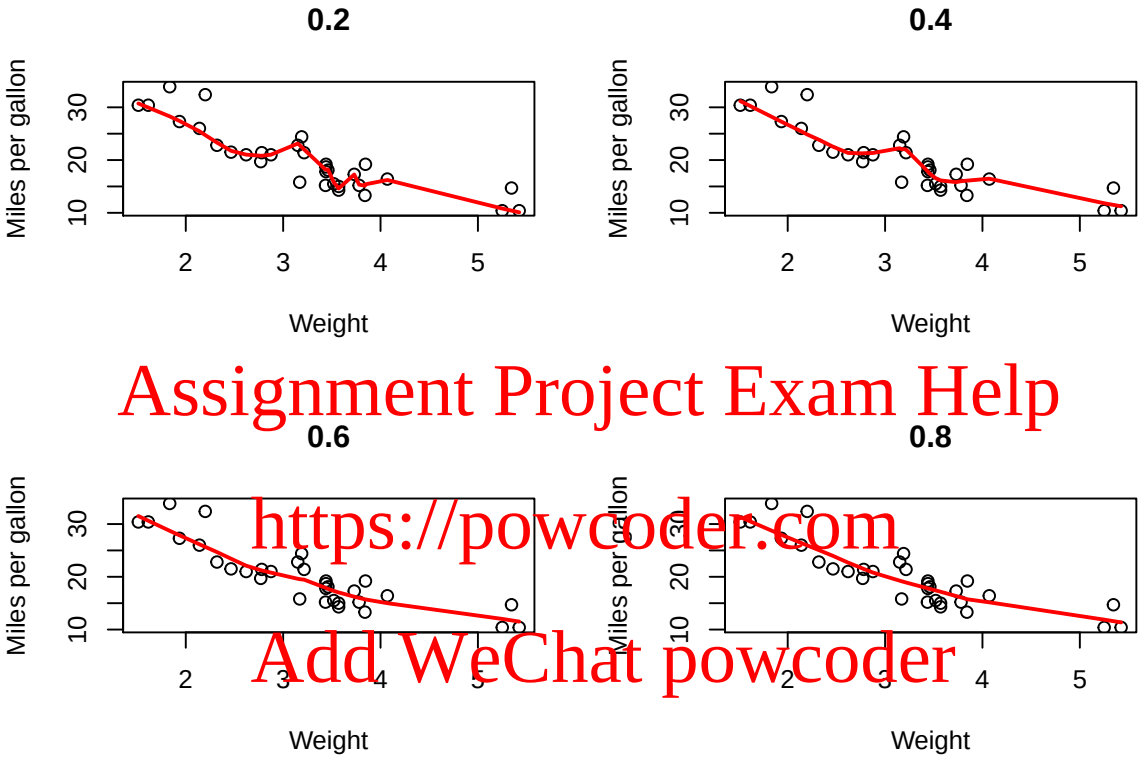
```
data(mtcars)
with(mtcars, {plot(wt, mpg, xlab = "Weight", ylab = "Miles per gallon")
              lines(lowess(wt, mpg), col = "red")})
```



The bandwidth in `lowess` is specified by changing the argument `f`, which varies between 0 and 1, and represents the proportion of points that are included in each local regression.

Here are smoothers using `lowess` with a range of smoothing parameter values:

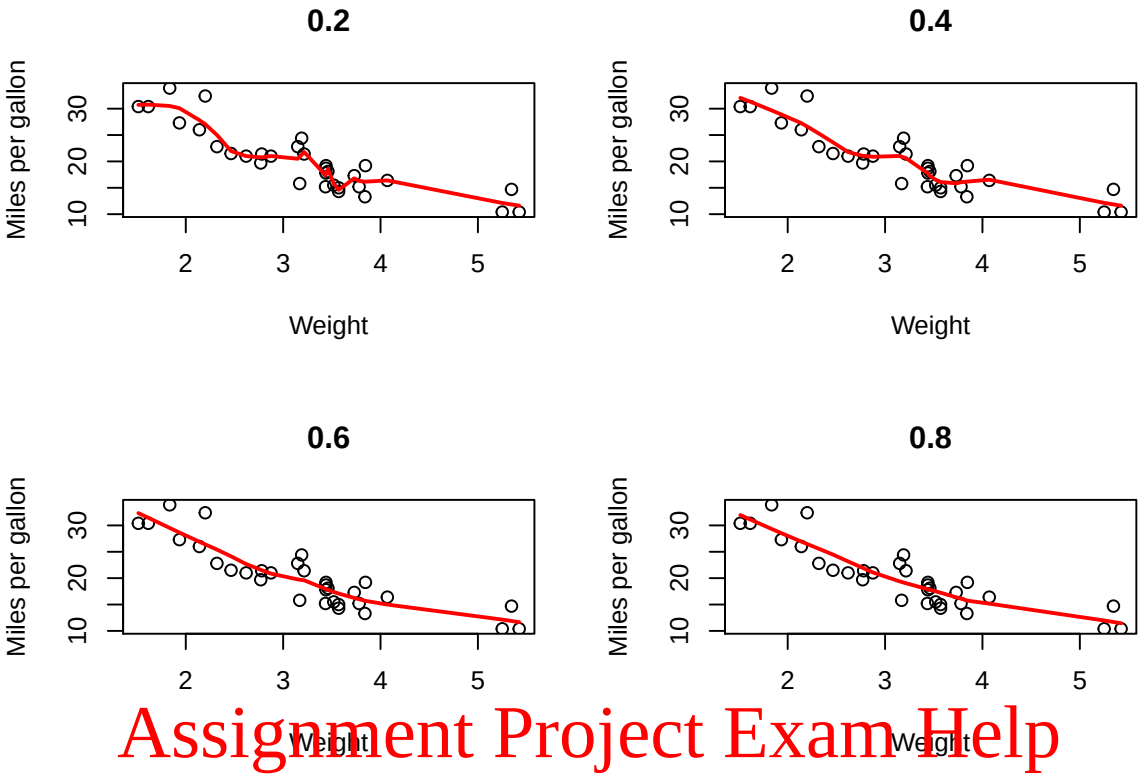
```
data(mtcars)
par(mfrow = c(2,2))
for (i in c(0.2, 0.4, 0.6, 0.8))
  with(mtcars, {plot(wt, mpg, xlab = "Weight",
                     ylab = "Miles per gallon", main = i)
                lines(lowess(wt, mpg, f = i), col = "red", lwd = 2)})
```




```
par(mfrow = c(1,1))
```

We can make this less robust to outliers by setting `iter = 0`:

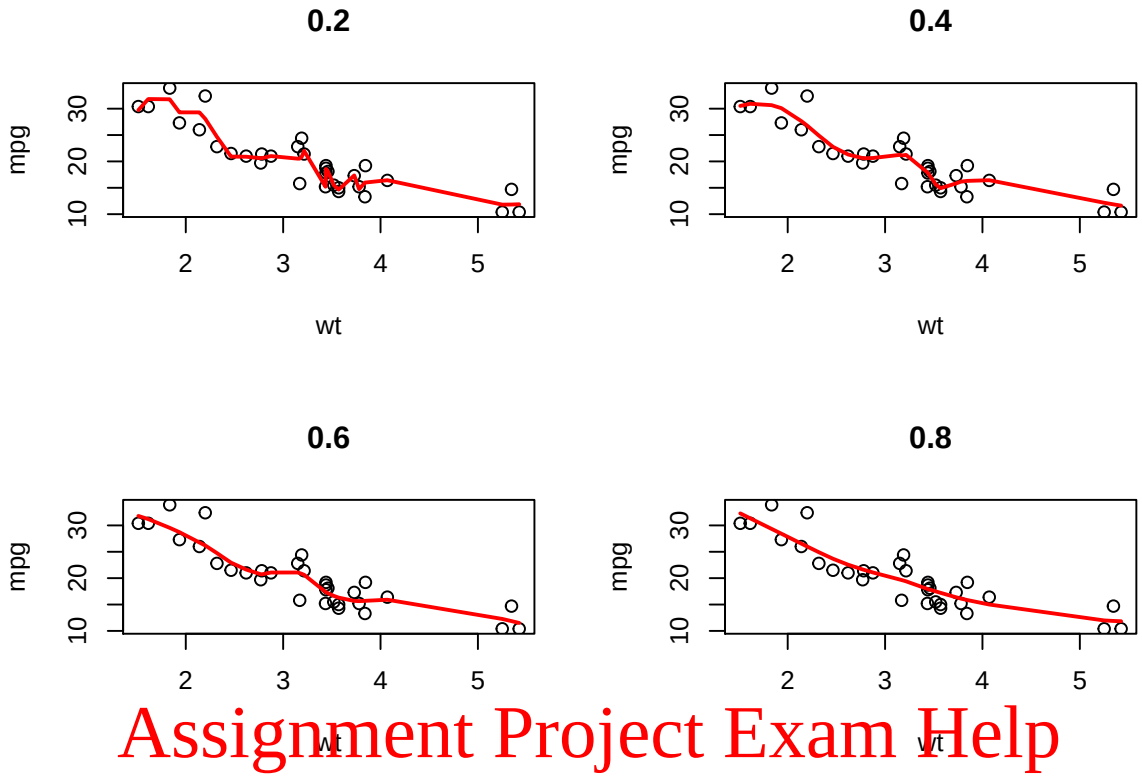
```
data(mtcars)
par(mfrow = c(2,2))
for (i in c(0.2, 0.4, 0.6, 0.8))
  with(mtcars, {plot(wt, mpg, xlab = "Weight",
                     ylab = "Miles per gallon", main = i)
                lines(lowess(wt, mpg, f = i, iter = 0), col = "red", lwd = 2)})
```




```
par(mfrow = c(1,1))
```


Now smoothers using loess, which by default fits local quadratic models (with the smoothing parameter/bandwidth now specified using the argument span):

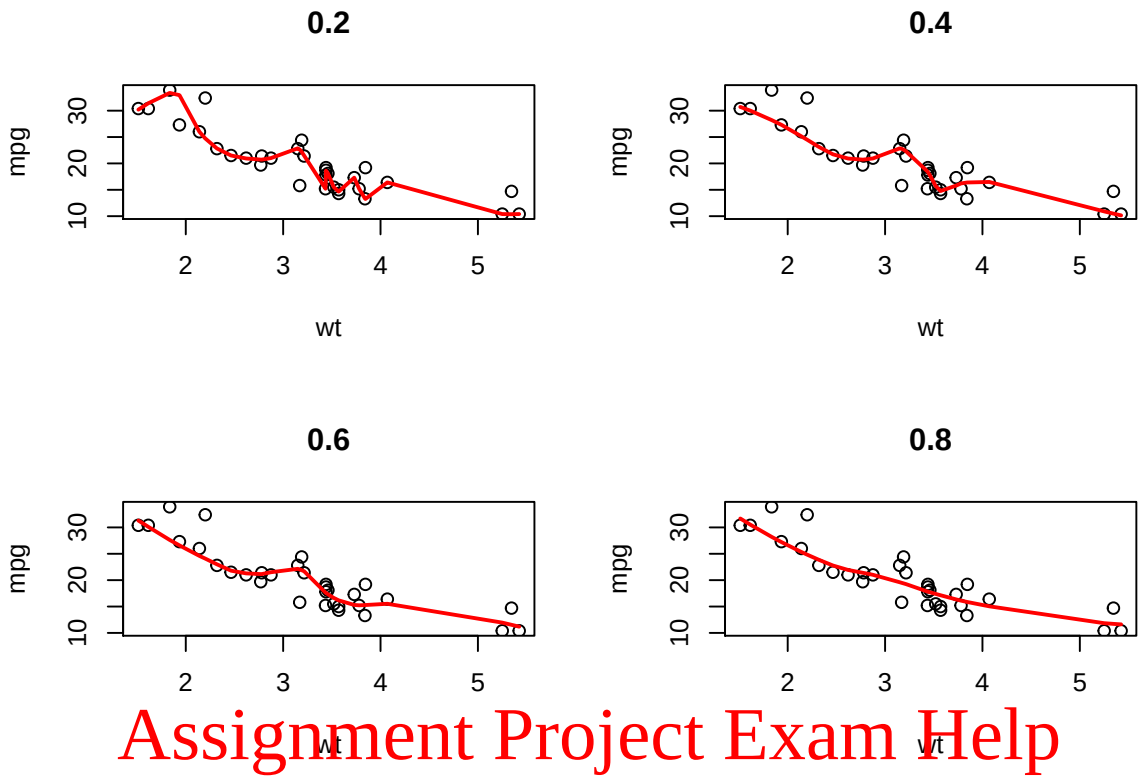

```
wt = sort(mtcars$wt)
mpg = mtcars$mpg[order(mtcars$wt)]
par(mfrow = c(2,2))
for (i in c(0.2, 0.4, 0.6, 0.8)) {
    plot(wt, mpg, main = i)
    loess.fit = loess(mpg ~ wt, span = i)
    lines(wt, predict(loess.fit), col = "red", lwd = 2)
}
```




```
par(mfrow = c(1,1))
```


We can make `loess` more robust to outliers by specifying `family = "symmetric"`:


```
wt = sort(mtcars$wt)
mpg = mtcars$mpg[order(mtcars$wt)]
par(mfrow = c(2,2))
for (i in c(0.2, 0.4, 0.6, 0.8)) {
    plot(wt, mpg, main = i)
    loess.fit = loess(mpg ~ wt, span = i, family = "symmetric")
    lines(wt, predict(loess.fit), col = "red", lwd = 2)
}
```




```
par(mfrow = c(1,1))
```


Other methods of smoothing

There are lots of other methods that we could look at, including

- k-nearest neighbour smoothing
- spline smoothing
- smoothing using orthogonal series