# Chapter 3

# Bivariate data

## 3.1 Building and checking a model

**READ: Cleveland pp. 86–119.**

### 3.1.1 Ganglion data

Load the usual:

```r
load("lattice.RData")
library(ggplot2)
```

A ganglion is a nerve cell cluster. The data set `ganglion` contains data on ganglia in the retina of 14 cat fetuses. There are two variables:

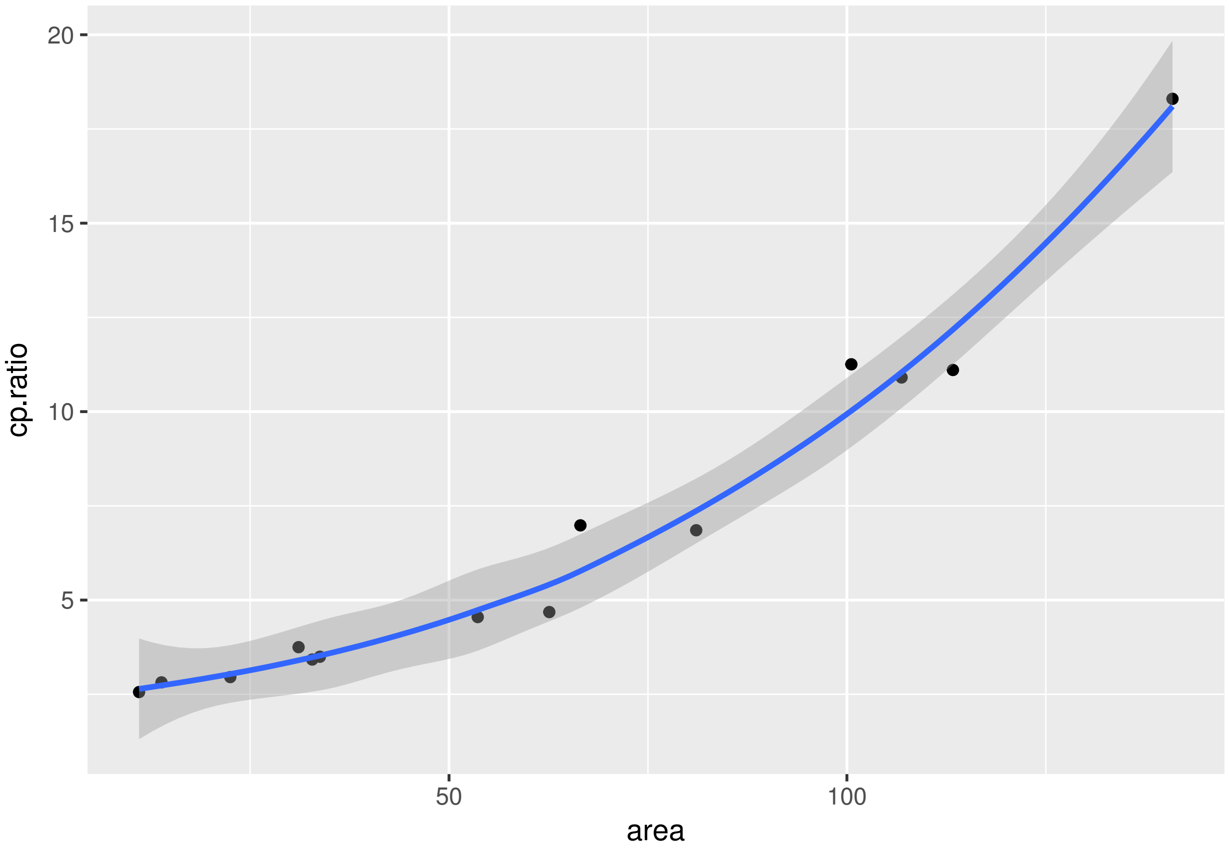
- `area` is retinal area (in $mm^2$)
- `cp.ratio` is the ratio of ganglion cell density in the center of the retina to the ganglion cell density in the periphery (edge) of the retina.

We draw a scatterplot:

```r
ganglion.gg = ggplot(ganglion, aes(x = area, y = cp.ratio)) + geom_point()
```

Let's add a smooth curve to the plot.

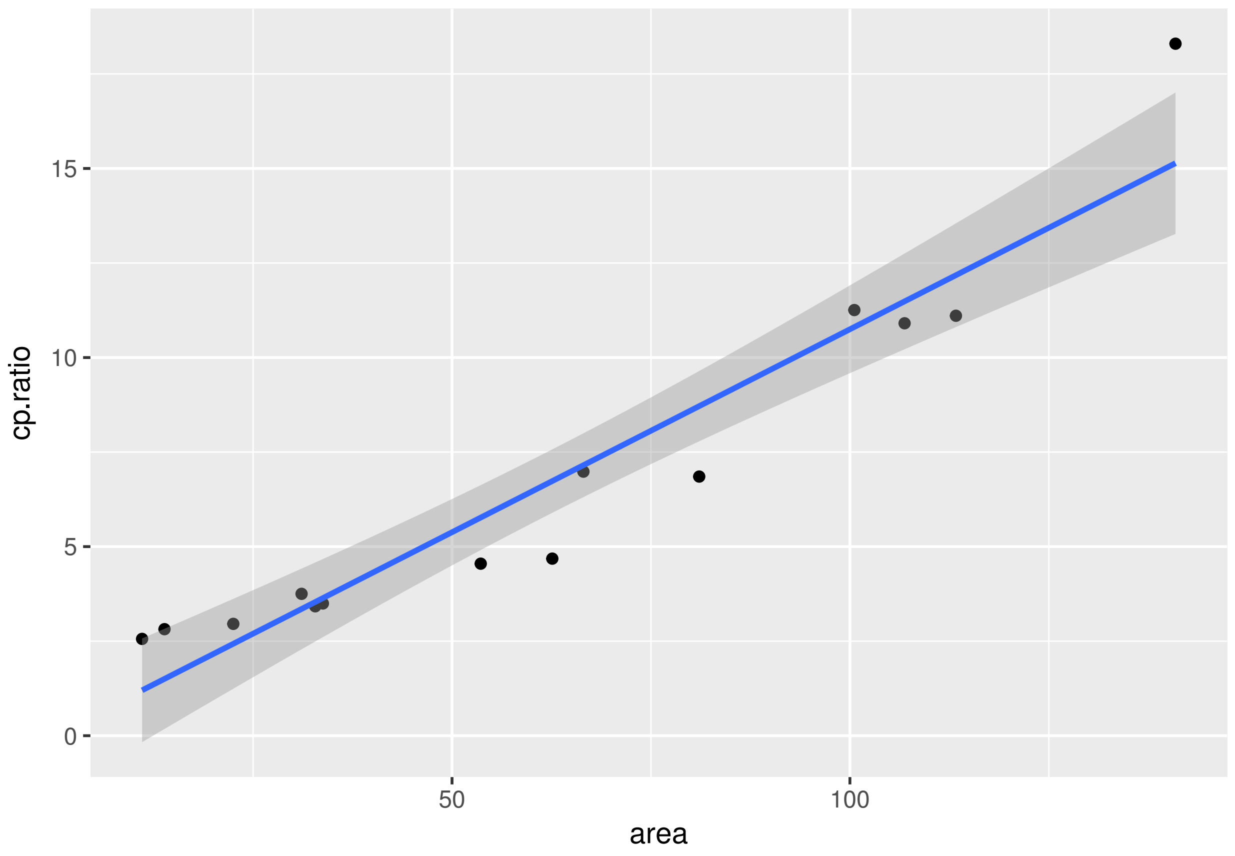
```r
ganglion.gg + geom_smooth()
```



By default, the curve is fitted using the "loess" method; we'll talk more about this method later. The 95% confidence bands are for the fit, not for predictions (so we do not expect 95% of the data to be contained in the shaded area.)

Visually, we see the loess fit is quite curved. How well would a straight line do?

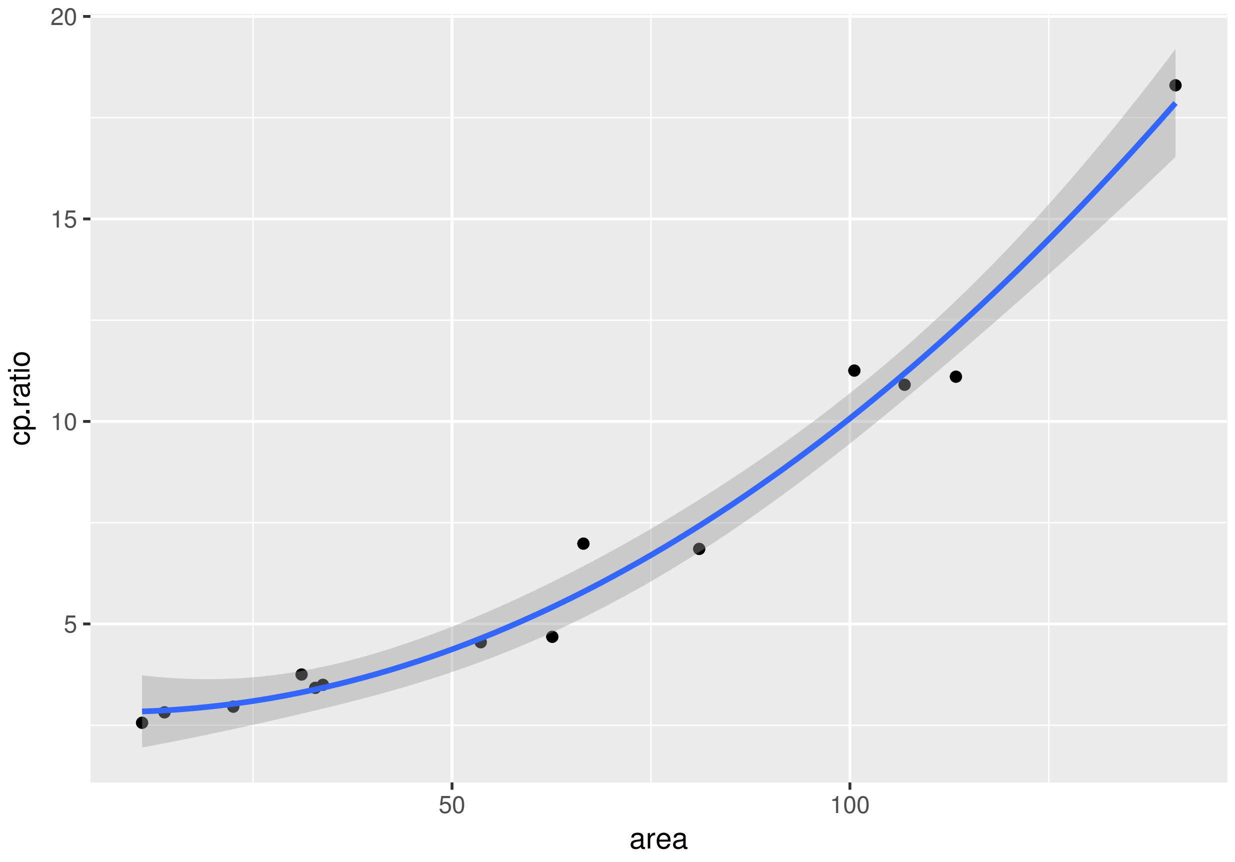
```
ganglion.gg + geom_smooth(method = "lm")
```



We could look more carefully at the residuals, but it seems clear that the loess curve provides a better fit. (Note also there's no straight line that goes through the entire loess confidence band.)

If a straight line doesn't work and you still want a parametric fit, try a quadratic.

```
ganglion.gg + geom_smooth(method = "lm", formula = y ~ x + I(x^2))
```



The quadratic fit looks more or less the same as the loess fit. The confidence bands are much narrower – they probably underestimate the uncertainty unless we have strong reason to believe the parametric model is correct.

### 3.1.2 The broom library and augment()

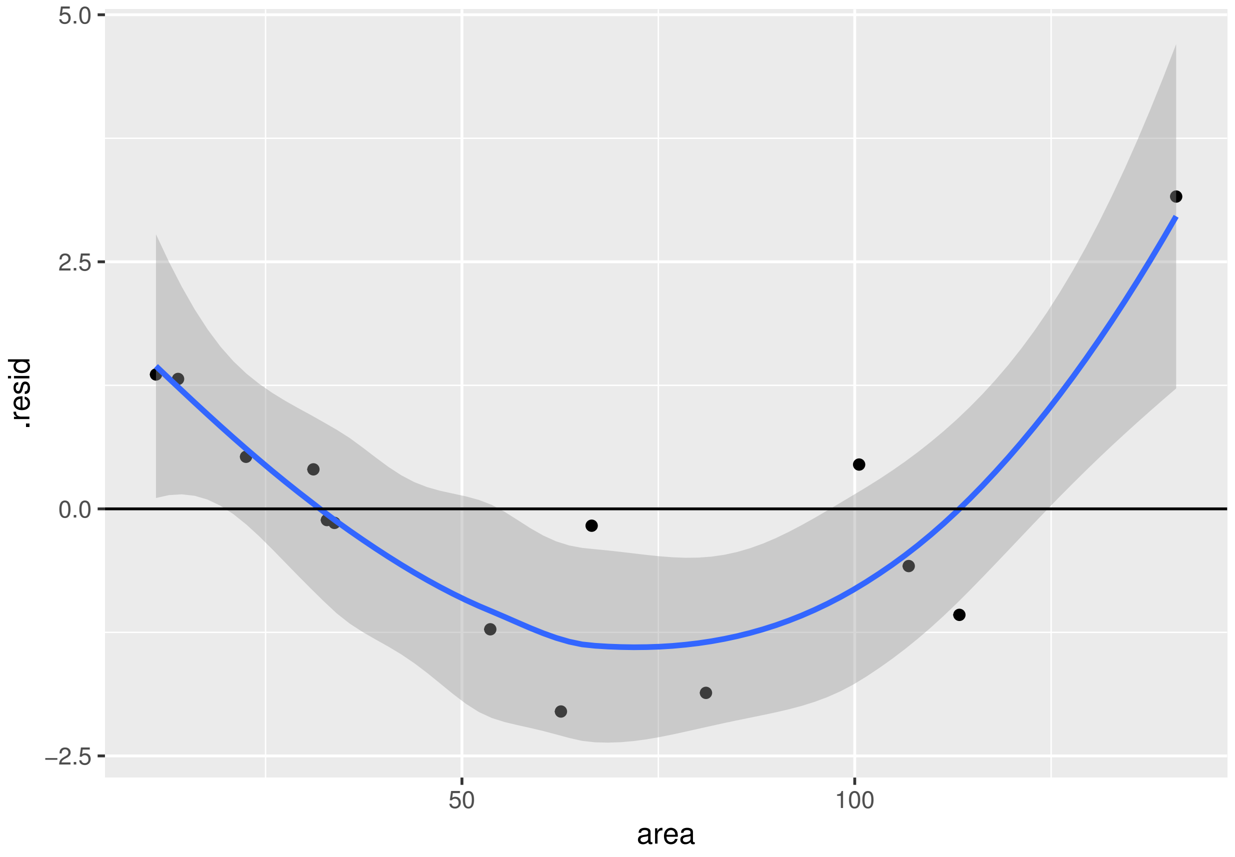
The `augment()` function within the `broom` library (which you should install) puts fitted values and residuals, among other things, into a convenient data frame. Let's fit a linear model and try it out:

```
ganglion.lm = lm(cp.ratio ~ area, data = ganglion)
# install.packages(broom)
library(broom)
gang.lm.df = augment(ganglion.lm)
summary(gang.lm.df)
```

```
##     cp.ratio           area           .fitted          .se.fit
##  Min.   : 2.560   Min.   : 11.05   Min.   : 1.200   Min.   :0.3851
##  1st Qu.: 3.441   1st Qu.: 31.52   1st Qu.: 3.397   1st Qu.:0.4386
##  Median : 4.616   Median : 58.10   Median : 6.251   Median :0.5132
##  Mean   : 6.688   Mean   : 62.18   Mean   : 6.688   Mean   :0.5303
##  3rd Qu.: 9.925   3rd Qu.: 95.68   3rd Qu.:10.284   3rd Qu.:0.6013
##  Max.   :18.300   Max.   :140.92   Max.   :15.139   Max.   :0.8580
##      .resid              .hat              .sigma          .cooksd
##  Min.   :-2.0509   Min.   :0.07144   Min.   :0.9262   Min.   :0.0004336
##  1st Qu.:-0.9493   1st Qu.:0.09286   1st Qu.:1.4356   1st Qu.:0.0065309
##  Median :-0.1277   Median :0.12714   Median :1.4770   Median :0.0249937
##  Mean   : 0.0000   Mean   :0.14286   Mean   :1.4258   Mean   :0.1871098
##  3rd Qu.: 0.5072   3rd Qu.:0.17421   3rd Qu.:1.4990   3rd Qu.:0.0870057
##  Max.   : 3.1603   Max.   :0.35459   Max.   :1.5044   Max.   :2.0477601
##    .std.resid
##  Min.   :-1.47721
##  1st Qu.:-0.73058
##  Median :-0.09395
##  Mean   : 0.03741
##  3rd Qu.: 0.38008
##  Max.   : 2.73029
```

We plot the residuals against the explanatory variable, then add a loess curve. If the confidence band contains the line $y = 0$, then maybe the model is fitting well.

```
ggplot(gang.lm.df, aes(x = area, y = .resid)) + geom_point() + geom_smooth() +
    geom_abline(slope = 0, intercept = 0)
```



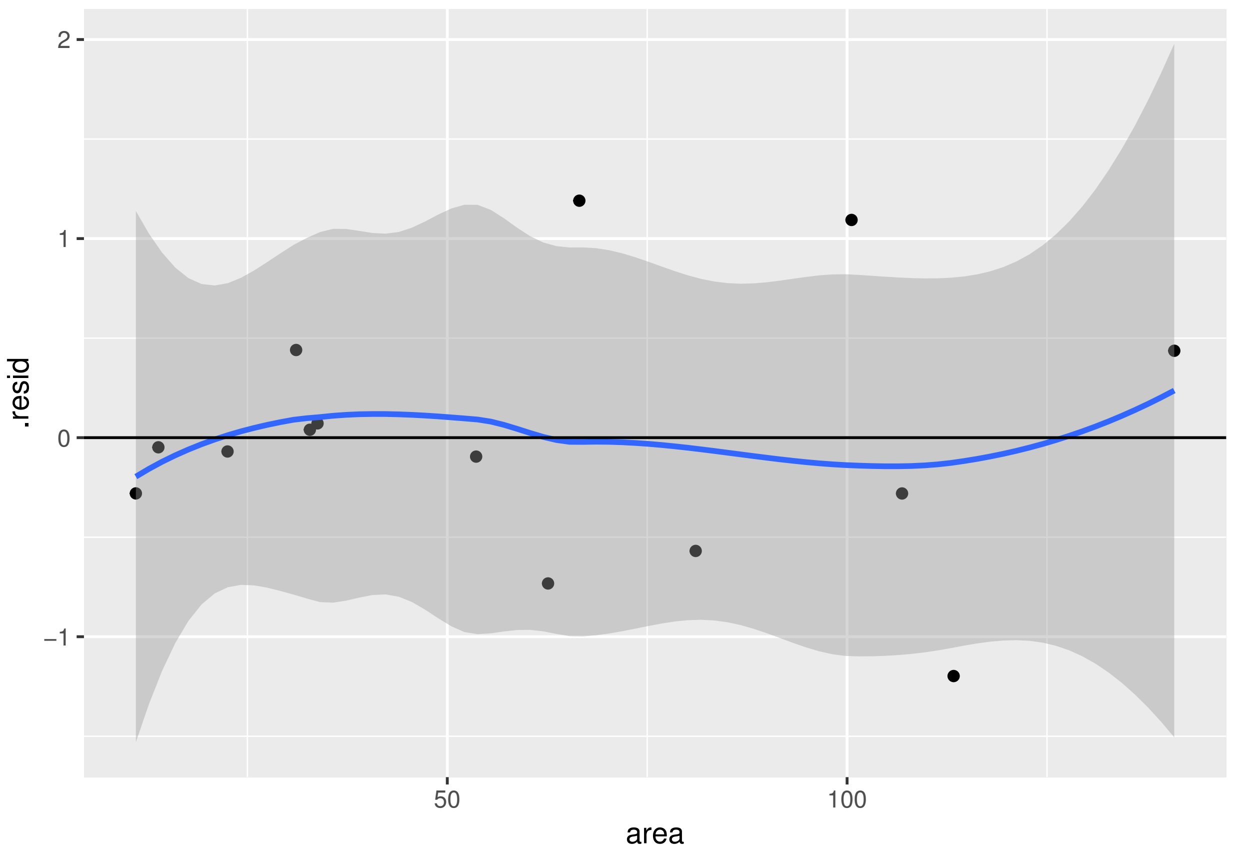
There's a clear curve in the residuals, so the linear model doesn't fit.

Let's try the quadratic:

```
ganglion.lm2 = lm(cp.ratio ~ area + I(area^2), data = ganglion)
gang.lm2.df = augment(ganglion.lm2)
summary(gang.lm2.df)
```

```
##     cp.ratio            area          I.area.2.          .fitted
##  Min.   : 2.560   Min.   : 11.05   Min.   :  122.1   Min.   : 2.840
##  1st Qu.: 3.441   1st Qu.: 31.52   1st Qu.:  994.3   1st Qu.: 3.329
##  Median : 4.616   Median : 58.10   Median : 3396.3   Median : 5.029
##  Mean   : 6.688   Mean   : 62.18   Mean   : 5430.5   Mean   : 6.688
##  3rd Qu.: 9.925   3rd Qu.: 95.68   3rd Qu.: 9226.8   3rd Qu.: 9.477
##  Max.   :18.300   Max.   :140.92   Max.   :19857.9   Max.   :17.863
##     .se.fit           .resid               .hat            .sigma
##  Min.   :0.2346   Min.   :-1.19758   Min.   :0.1107   Min.   :0.6081
##  1st Qu.:0.2683   1st Qu.:-0.28029   1st Qu.:0.1448   1st Qu.:0.6907
##  Median :0.2860   Median :-0.05893   Median :0.1644   Median :0.7283
##  Mean   :0.3130   Mean   : 0.00000   Mean   :0.2143   Mean   :0.7030
##  3rd Qu.:0.3051   3rd Qu.: 0.34532   3rd Qu.:0.1872   3rd Qu.:0.7393
##  Max.   :0.6052   Max.   : 1.19049   Max.   :0.7363   Max.   :0.7396
##     .cooksd             .std.resid
##  Min.   :0.0001503   Min.   :-1.88857
##  1st Qu.:0.0008978   1st Qu.:-0.47249
##  Median :0.0288280   Median :-0.09404
##  Mean   :0.1611383   Mean   : 0.02964
##  3rd Qu.:0.1569547   3rd Qu.: 0.52490
##  Max.   :1.3513989   Max.   : 1.84998
```

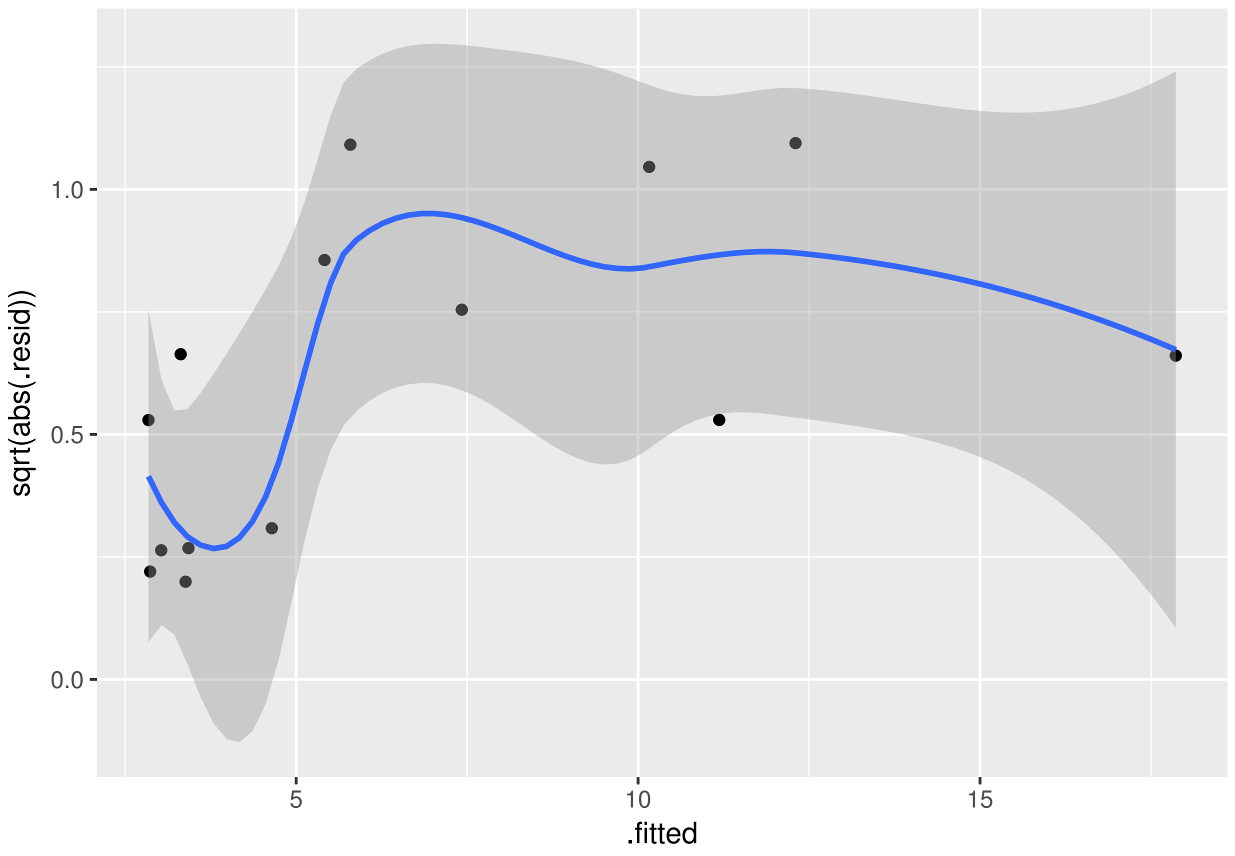
```
ggplot(gang.lm2.df, aes(x = area, y = .resid)) + geom_point() + geom_smooth() +
    geom_abline(slope = 0, intercept = 0)
```



The curve for the residuals just wiggles around 0. It's plausible that the quadratic model is correctly specified.

In addition, we check for homoscedasticity using a spread-location plot.

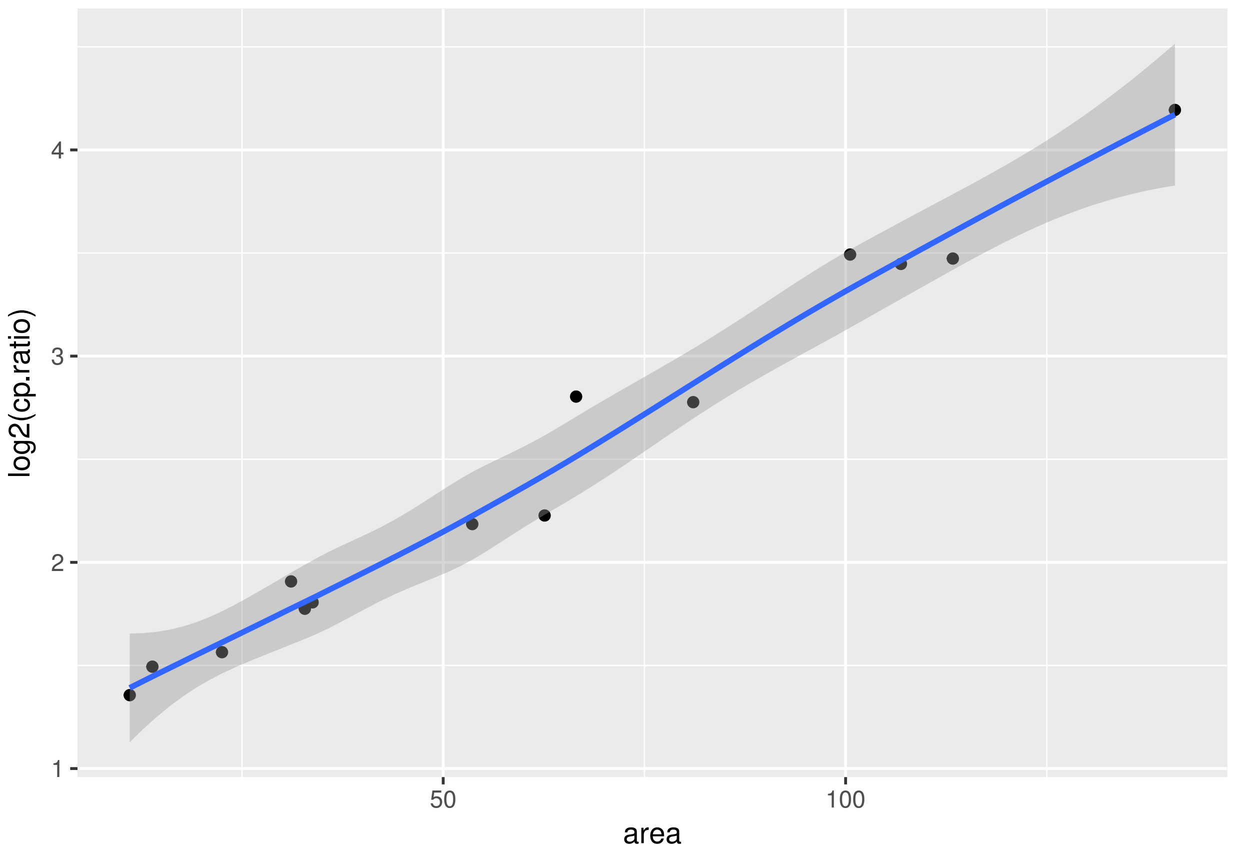
```
ggplot(gang.lm2.df, aes(x = .fitted, y = sqrt(abs(.resid)))) + geom_point() +
    geom_smooth()
```



It doesn't look like a horizontal line describes the transformed residuals well. The spread of the residuals changes with the fitted values, meaning there's heteroscedasticity. This makes the quadratic model less appealing – maybe a transformation would be better.

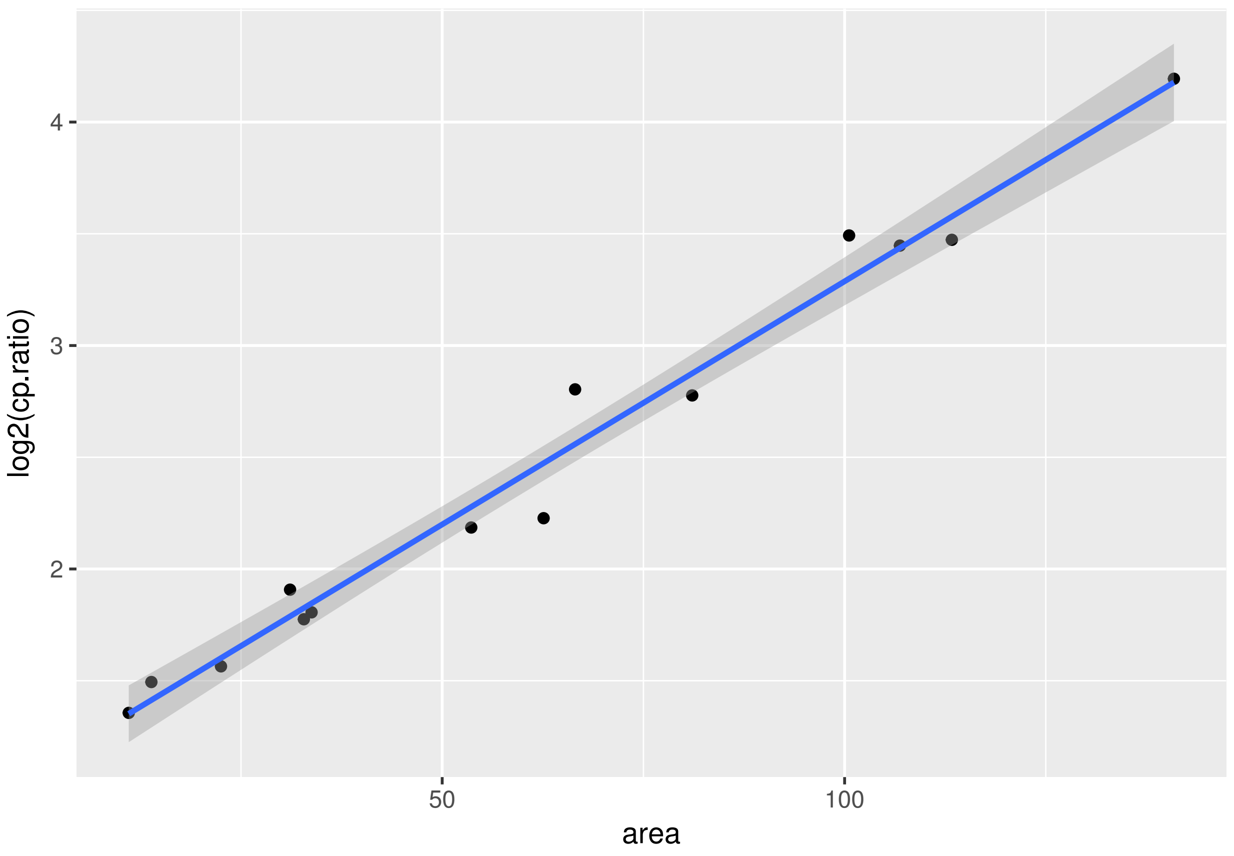
Let's try taking the log of the CP ratio. Add a loess fit:

```r
ggplot(ganglion, aes(x = area, y = log2(cp.ratio))) + geom_point() + geom_smooth()
```



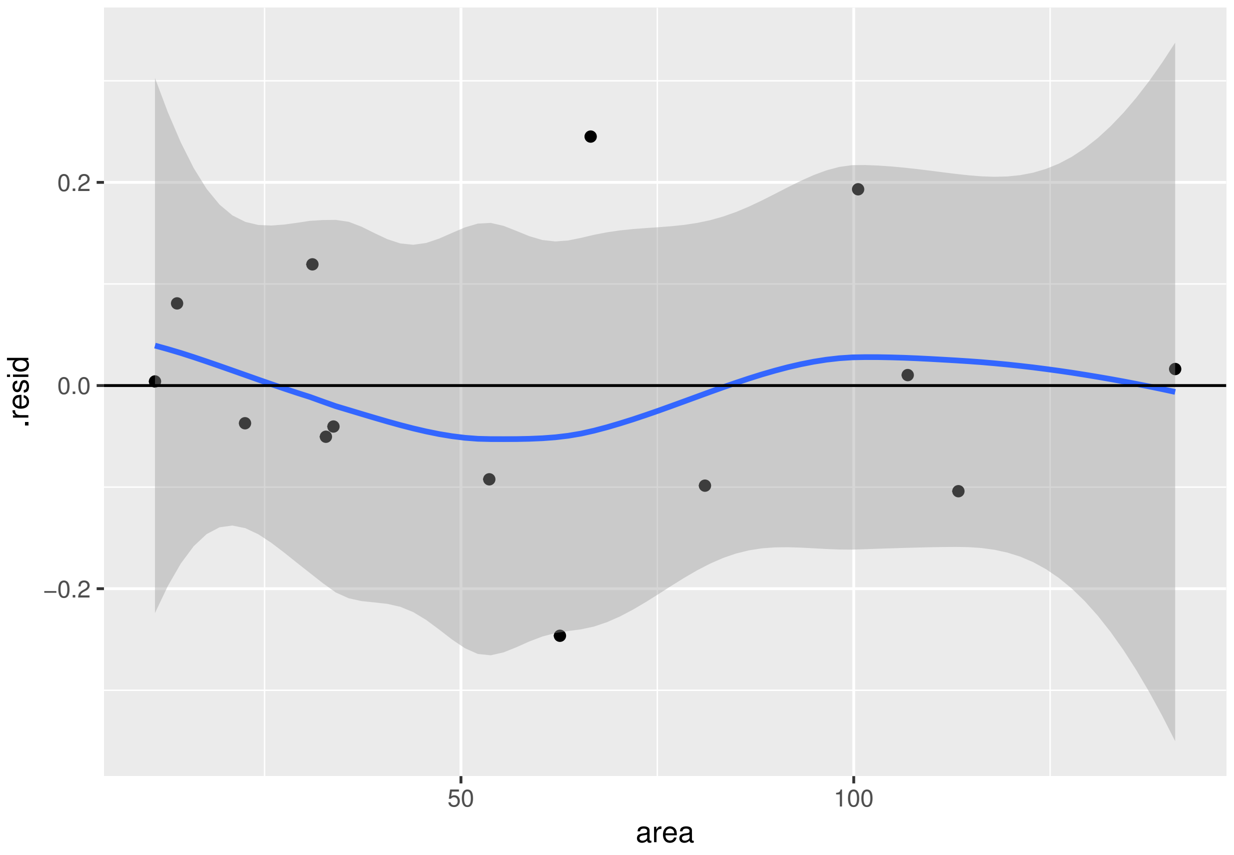
Looks like a straight line would do just as good a job. Let's use `lm()` to fit the model instead.

```r
ggplot(ganglion, aes(x = area, y = log2(cp.ratio))) + geom_point() + geom_smooth(method = "lm")
```



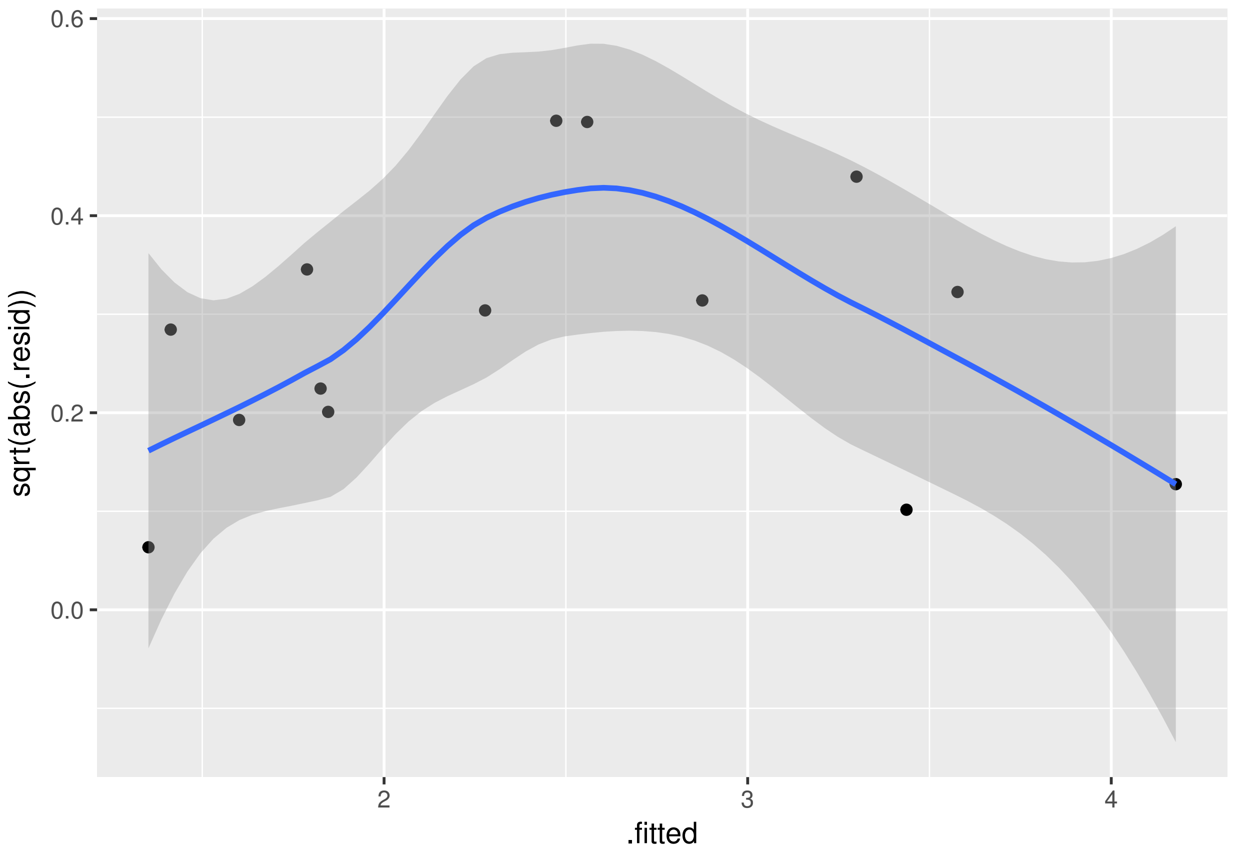
The straight line produced by `lm()` looks more or less the same as the loess fit. So far, so good. To get a better look, plot the residuals agaisnt `area`, the $x$-variable.

```
ganglion.log.lm = lm(log2(cp.ratio) ~ area, data = ganglion)
gang.log.lm.df = augment(ganglion.log.lm)
ggplot(gang.log.lm.df, aes(x = area, y = .resid)) + geom_point() + geom_smooth() +
    geom_abline(slope = 0, intercept = 0)
```



The curve just wiggles around zero. The residuals look like noise. Now check for homoscedasticity using a spread-location plot:

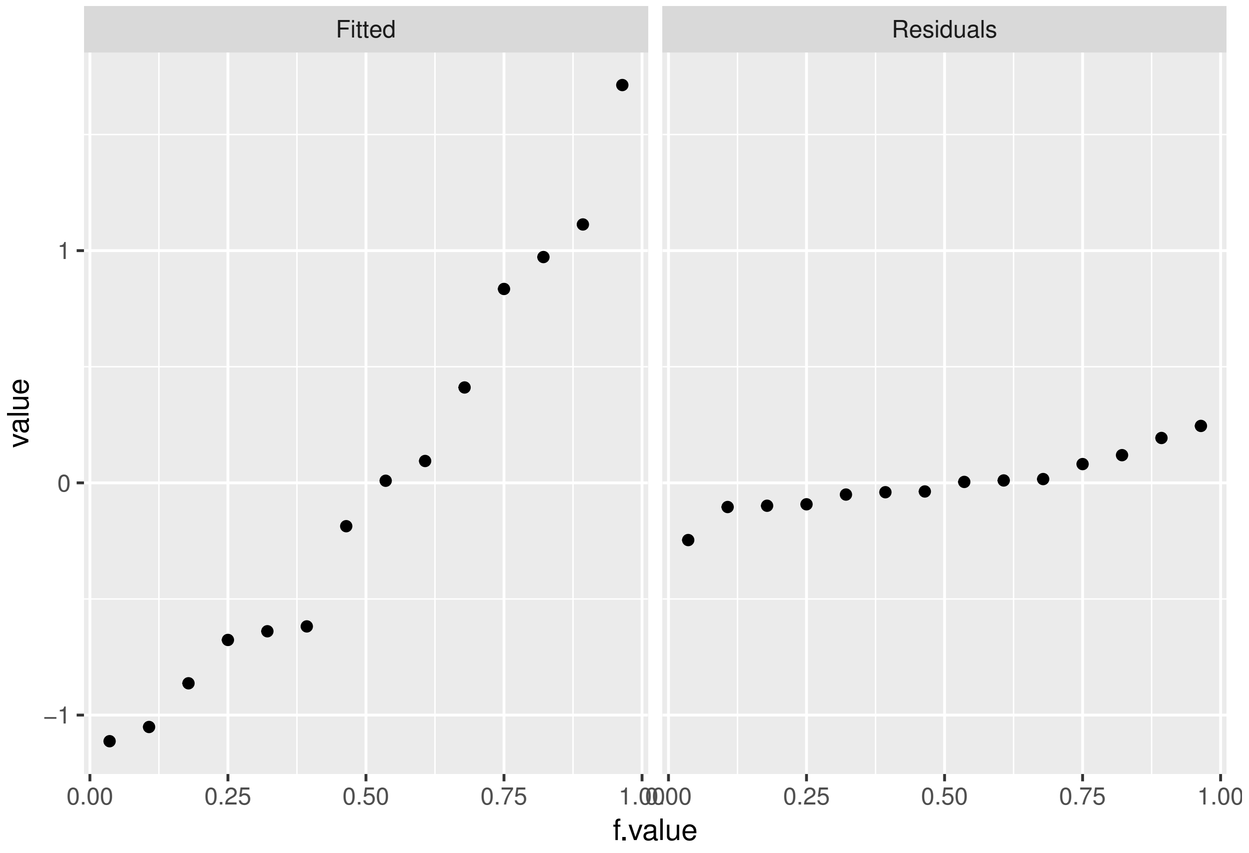
```r
ggplot(gang.log.lm.df, aes(x = .fitted, y = sqrt(abs(.resid)))) + geom_point() +
    geom_smooth()
```



This time we see a horizontal line goes through the confidence band, so homoscedasticity is a reasonable assumption. (The blue line is curved, but that could just be because of the small amount of data.)

Taking the log of `cp.ratio` has led to an elegant model: linear and homoscedastic. But how much does the model actually explain? Draw a residual-fit plot:

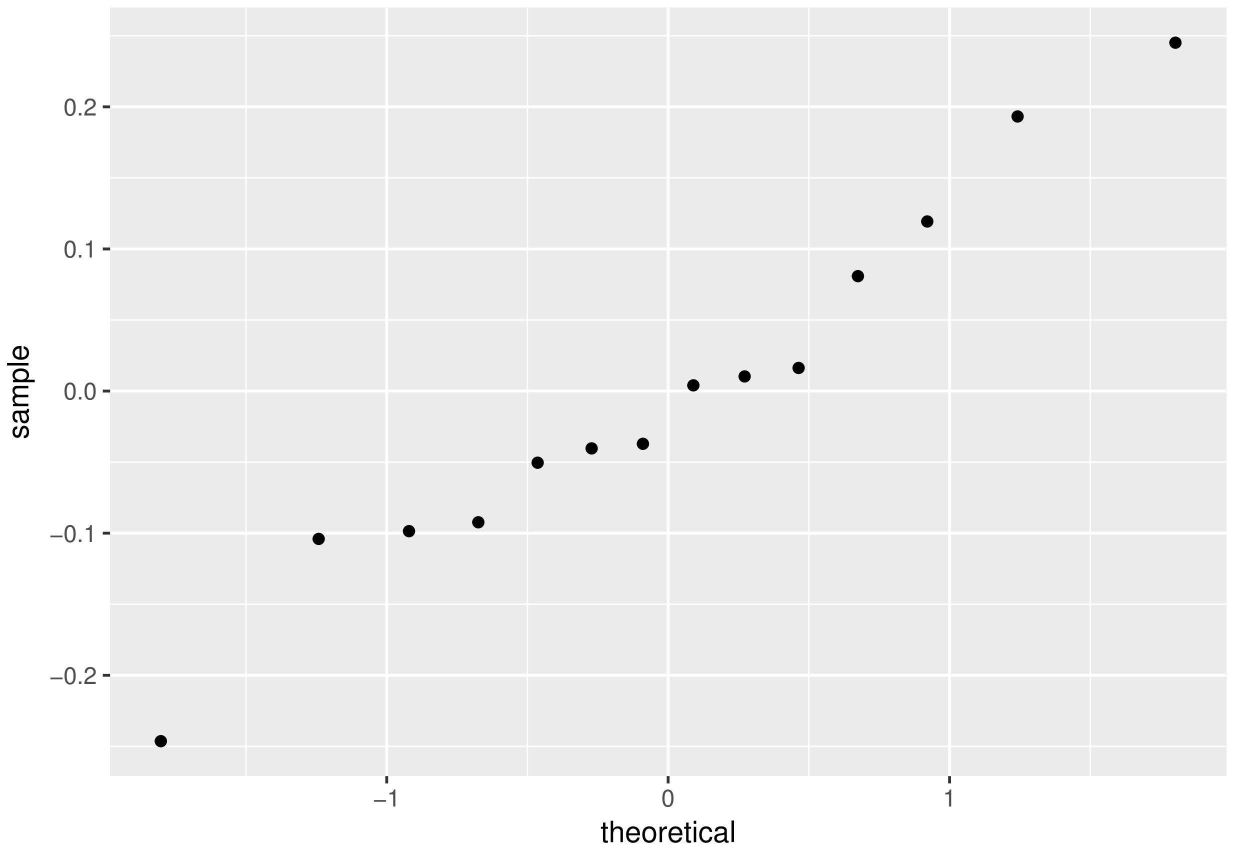
```
n = nrow(gang.log.lm.df)
f.value = (0.5:(n - 0.5))/n
gang.log.fit = data.frame(f.value, Fitted = sort(gang.log.lm.df$.fitted) - mean(gang.log.lm.df$.fitted)
    Residuals = sort(gang.log.lm.df$.resid))
library(tidyr)
gang.log.fit.long = gang.log.fit %>% gather(type, value, Fitted:Residuals)
ggplot(gang.log.fit.long, aes(x = f.value, y = value)) + geom_point() + facet_wrap(~type)
```



The spread of the fitted values far exceeds the spread of the fit. So the model incorporates most of the variation in the data.

Finally, we check if the residuals look normal. If so, this would be good, but it wouldn't necessarily be a dealbreaker if they weren't.

```
ggplot(gang.log.lm.df, aes(sample = .resid)) + stat_qq()
```



They're normal! Great! By taking logs, we get data that fulfills all the usual linear model assumptions, and can do formal inference if we wish.

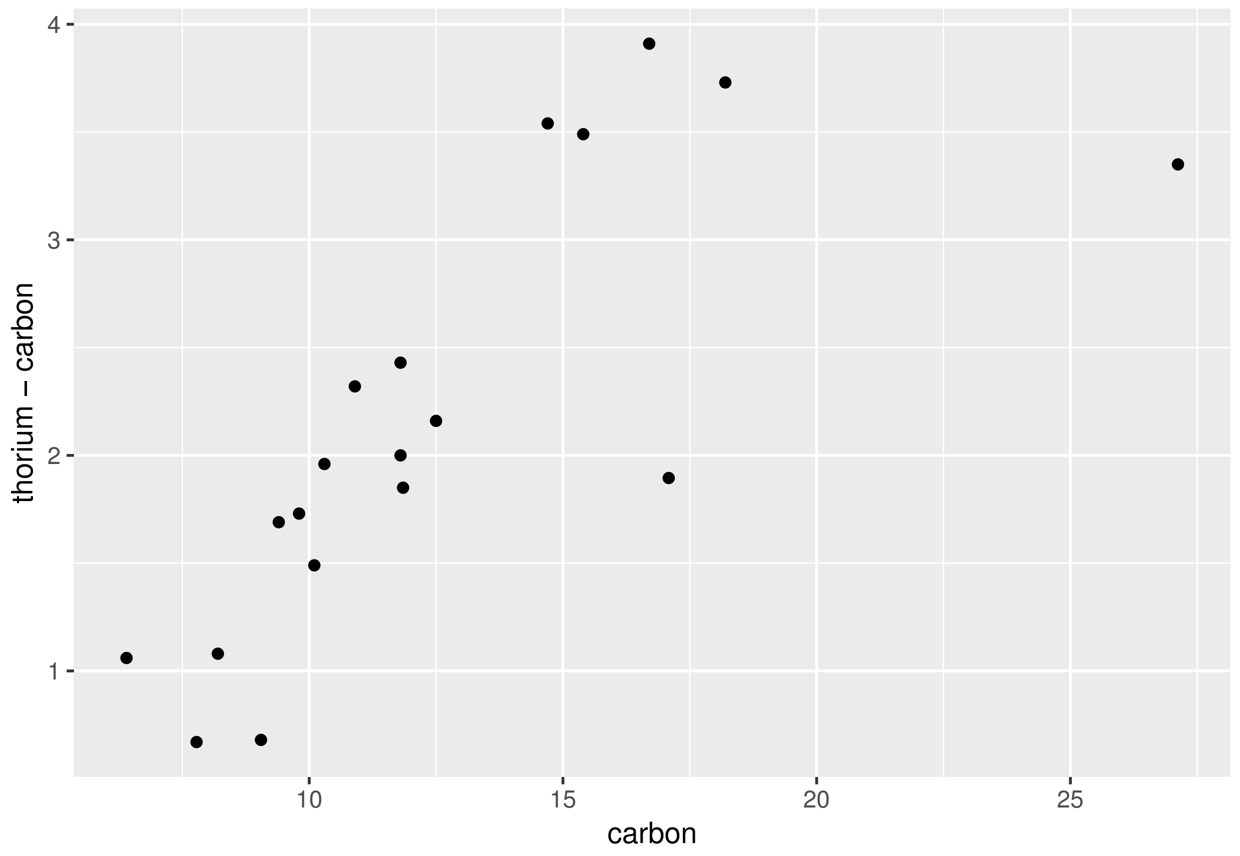
## 3.2 Robust fits and loess details

**READ: Cleveland pp. 110–134.**

### 3.2.1 Example: Carbon dating

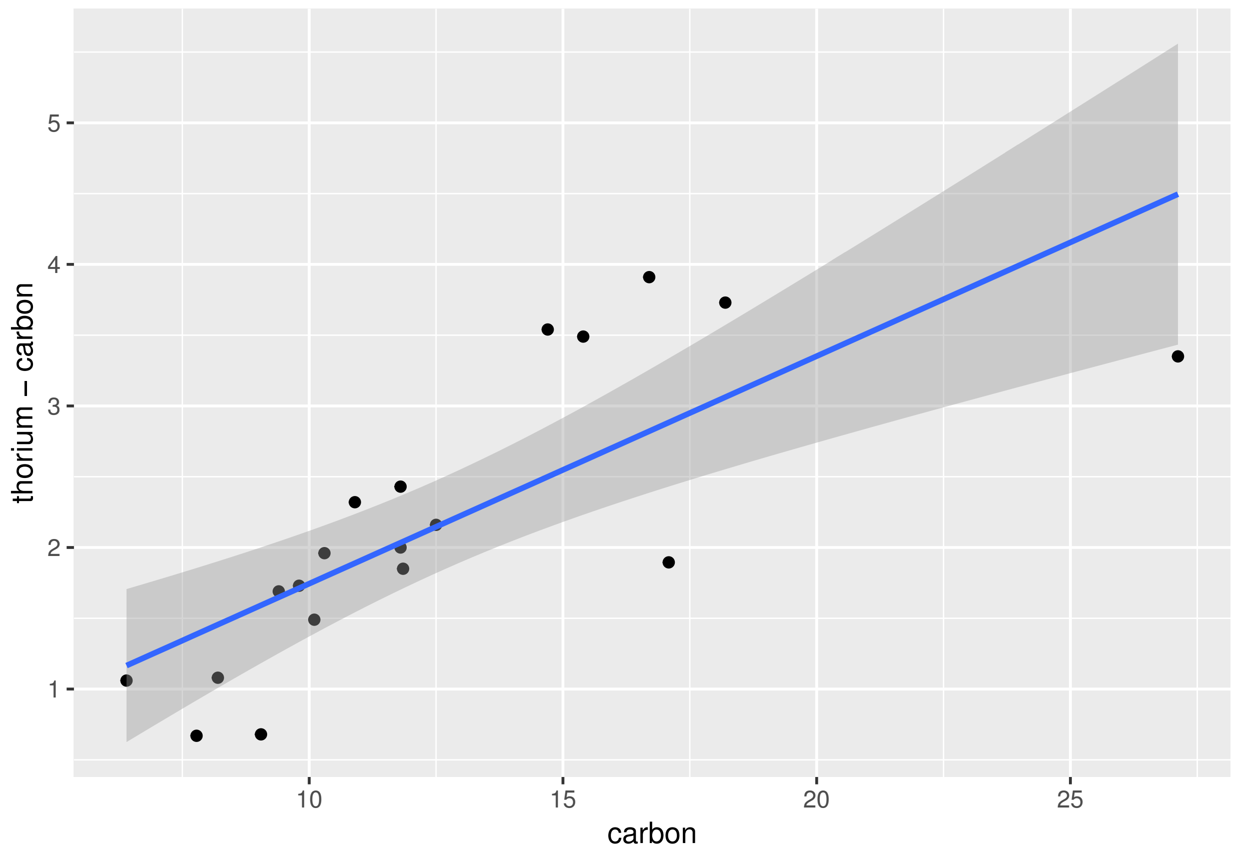
The data set `dating` contains paired observations giving the estimated ages of 19 coral samples in thousands of years using both `carbon` dating (the traditional method) and `thorium` dating (a modern and purportedly more accurate method.) What's the difference between these two methods?

```
load("lattice.RData")
library(ggplot2)
ggplot(dating, aes(x = carbon, y = thorium - carbon)) + geom_point()
```



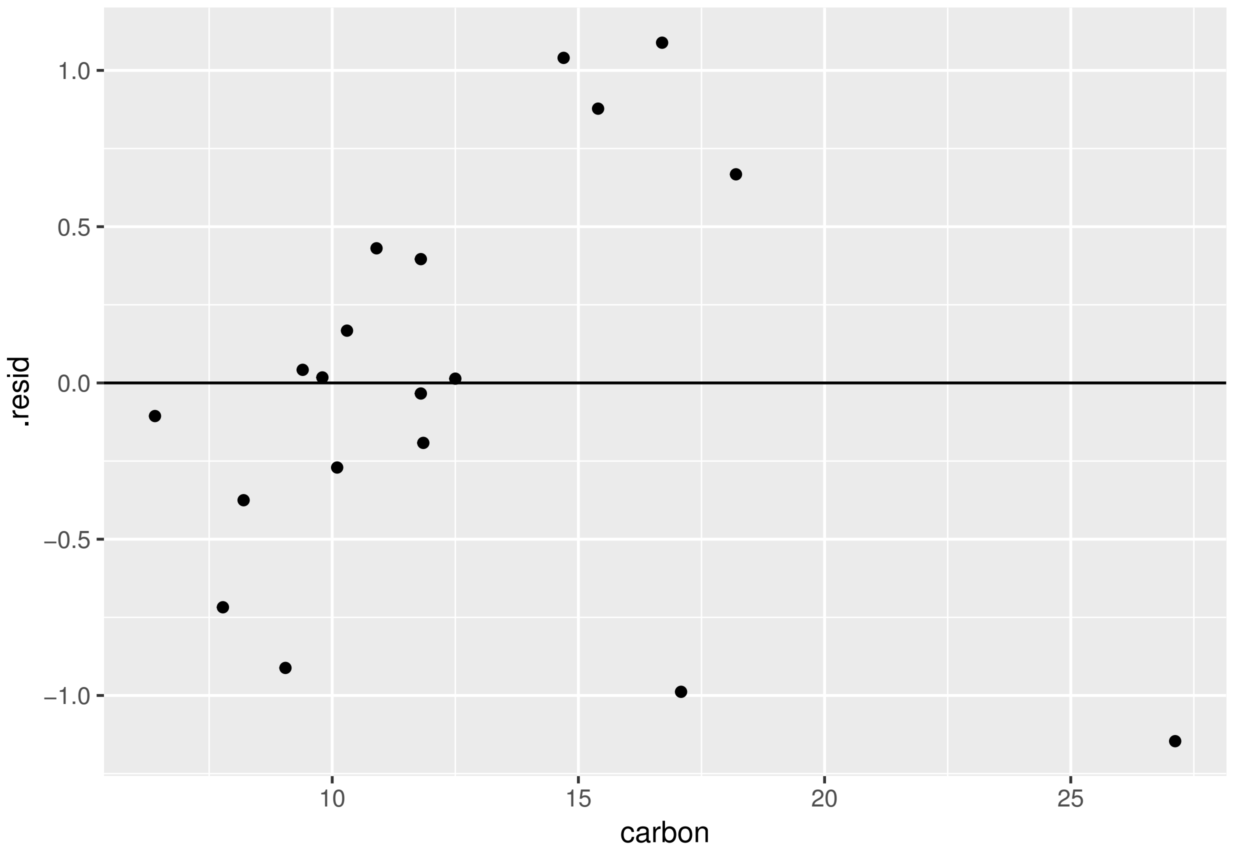
The first thing to note about this graph is that the $y$-values are always above zero, meaning thorium age is always above zero. The second thing to note is this difference increases with carbon age. Try adding a straight line to the plot:

```r
ggplot(dating, aes(x = carbon, y = thorium - carbon)) + geom_point() + geom_smooth(method = "lm")
```



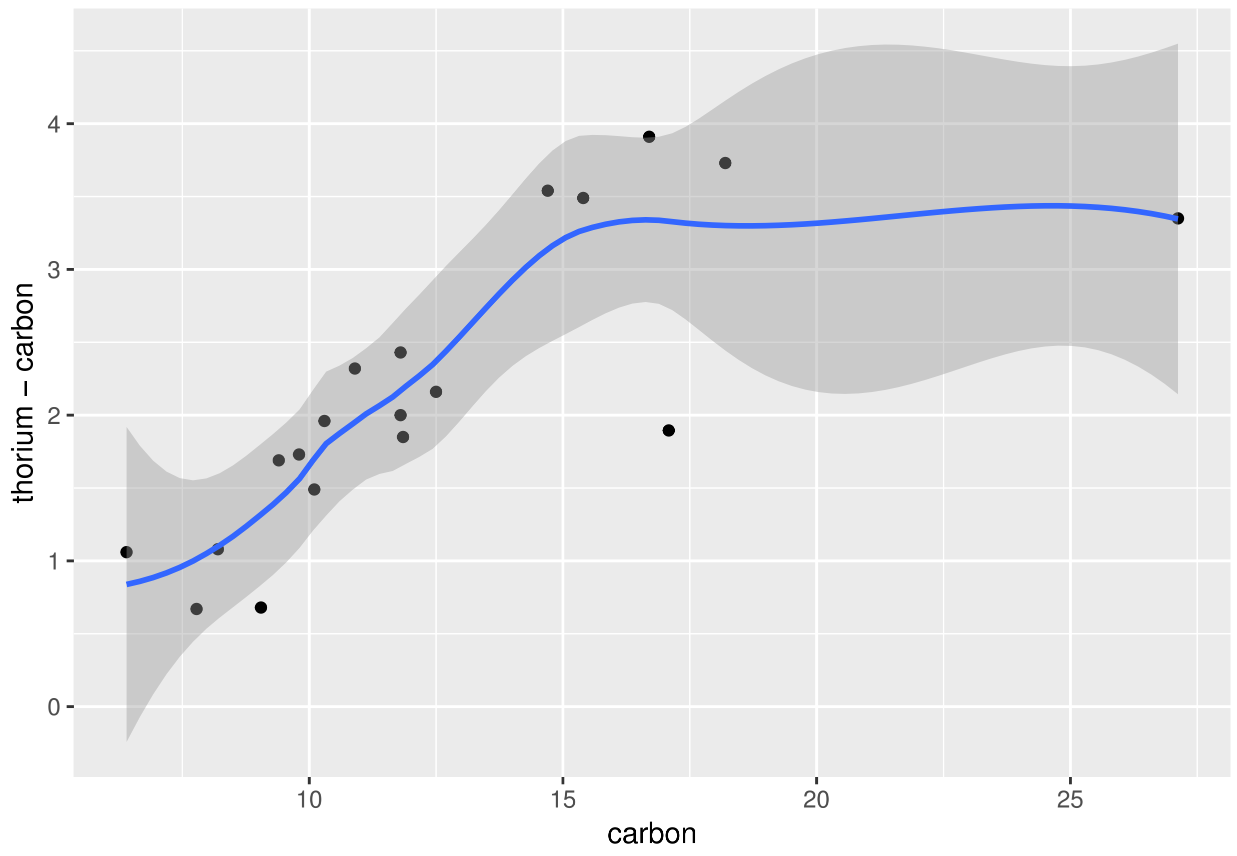
The line doesn't seem like it's doing a good job of describing the bivariate relationship for most of the data – in particular, a lot of points are well above the line. To see why, plot the residuals.

```
dating.lm = lm(thorium - carbon ~ carbon, data = dating)
library(broom)
dating.lm.df = augment(dating.lm)
ggplot(dating.lm.df, aes(x = carbon, y = .resid)) + geom_point() + geom_abline(slope = 0)
```



The issue is that the extreme value on the right-hand side is dragging the whole line down. Recall that outliers can have a large effect on a regression line. We could try using loess instead of a linear model, but this still isn't very satisfactory:

```r
ggplot(dating, aes(x = carbon, y = thorium - carbon)) + geom_point() + geom_smooth()
```

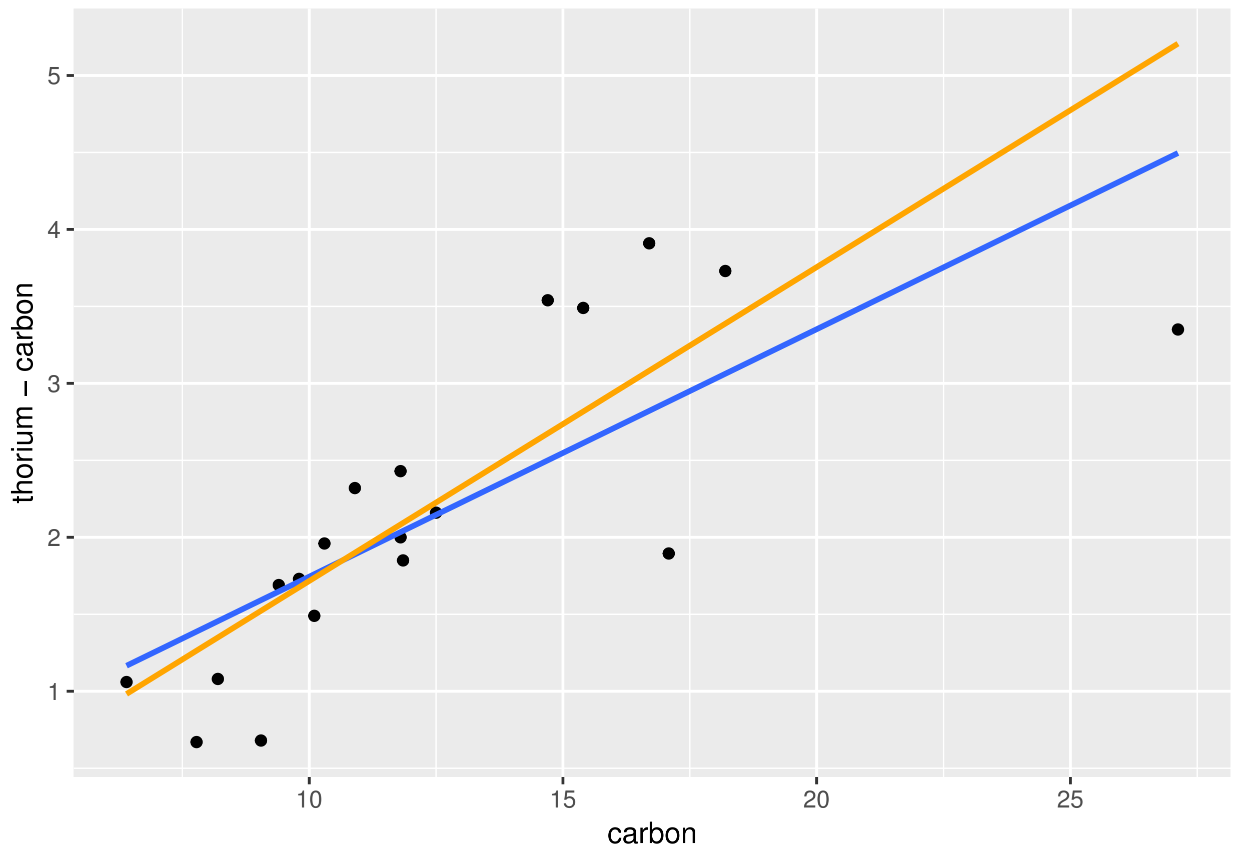


The loess curve is still missing the group of four observations where the difference is about 3.5 to 4 thousand years, possibly because the observation at $(17, 1.9)$ is also something of an outlier.

A better solution is to use a *robust* method to determine the best-fitting line, instead of least squares. The `rlm()` function in the `MASS` library (which should be installed by default) can fit such models, and can be called from within `geom_smooth()`. Let's see if it works just using the defaults:

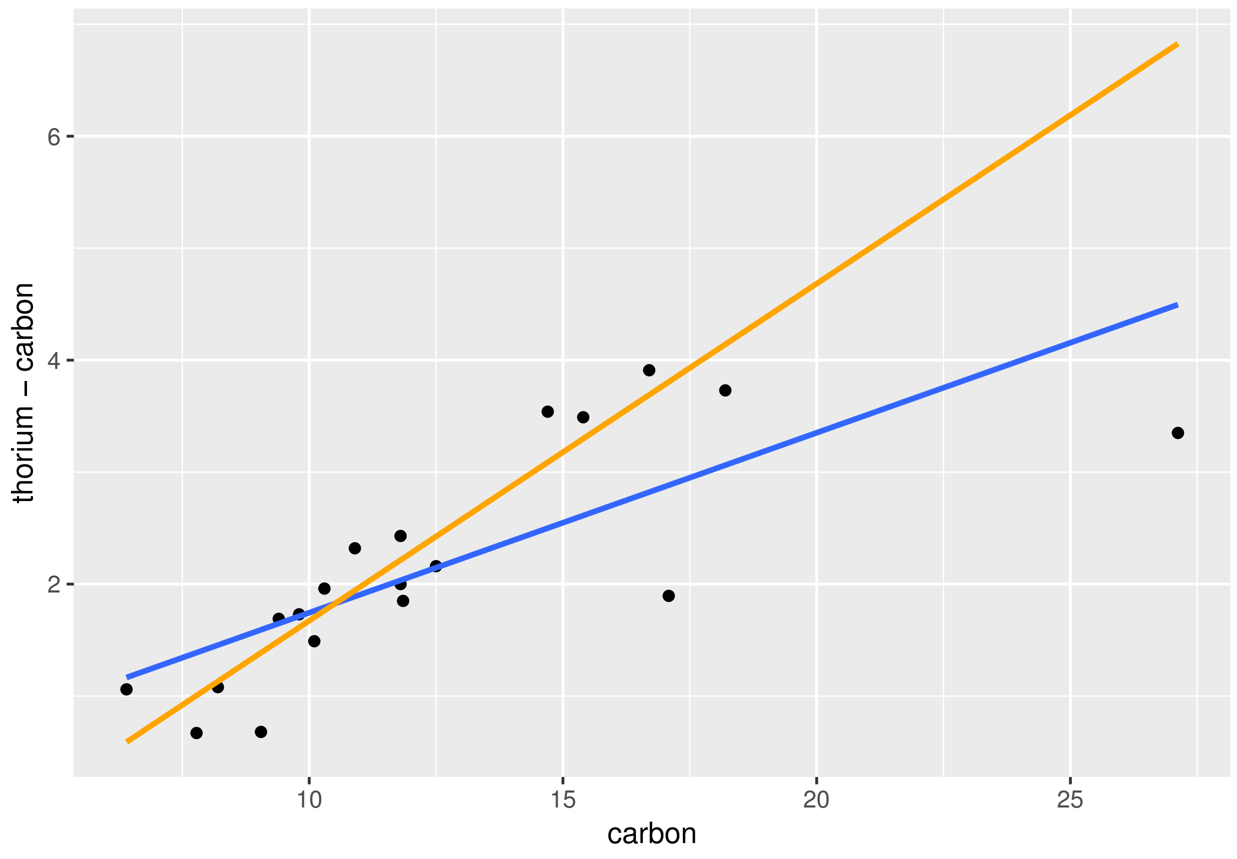
```
library(MASS)
ggplot(dating, aes(x = carbon, y = thorium - carbon)) + geom_point() + geom_smooth(method = "lm",
    se = FALSE) + geom_smooth(method = "rlm", se = FALSE, col = "orange")
```

```
## Warning in rlm.default(x, y, weights, method = method, wt.method =
## wt.method, : 'rlm' failed to converge in 20 steps
```



Not only do we get an error message, it still seems like the robust (orange) line is pulled down a bit too much by the two low-lying points. We can either increase the number of iterations (`maxit`) or try a different kind of fit. Tukey's **bisquare** method (sometimes called **biweight**) can be helpful when there are convergence problems, as it downweights extreme outliers to have zero weight.

```
ggplot(dating, aes(x = carbon, y = thorium - carbon)) + geom_point() + geom_smooth(method = "lm",
    se = FALSE) + geom_smooth(method = "rlm", se = FALSE, col = "orange", method.args = list(psi = psi.
```

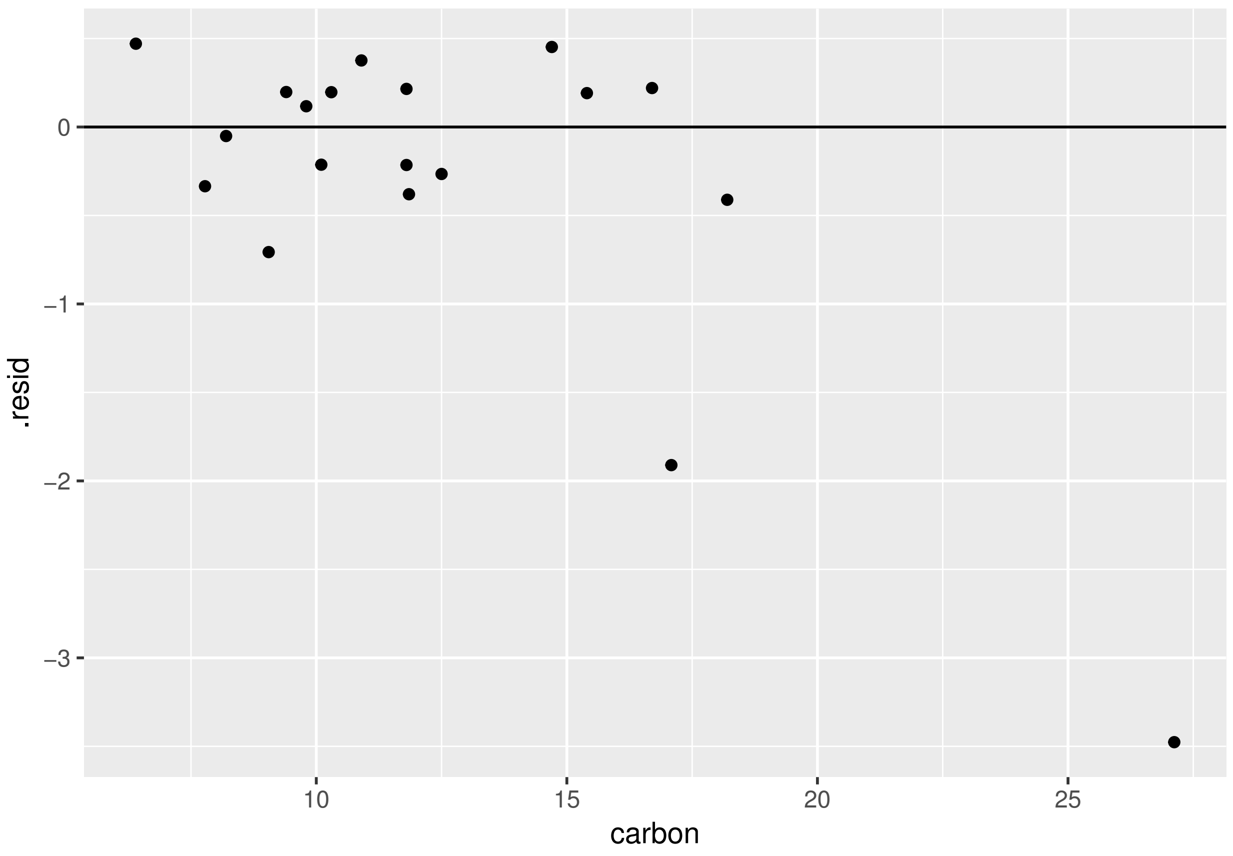


Now the orange line completely misses the two extreme outliers, but it fits all the rest of the data much better. Looking at the residuals makes this obvious:

```
age.diff = dating$thorium - dating$carbon
carbon = dating$carbon
dating.rlm = rlm(age.diff ~ carbon, psi = psi.bisquare)
tidy(dating.rlm)
```

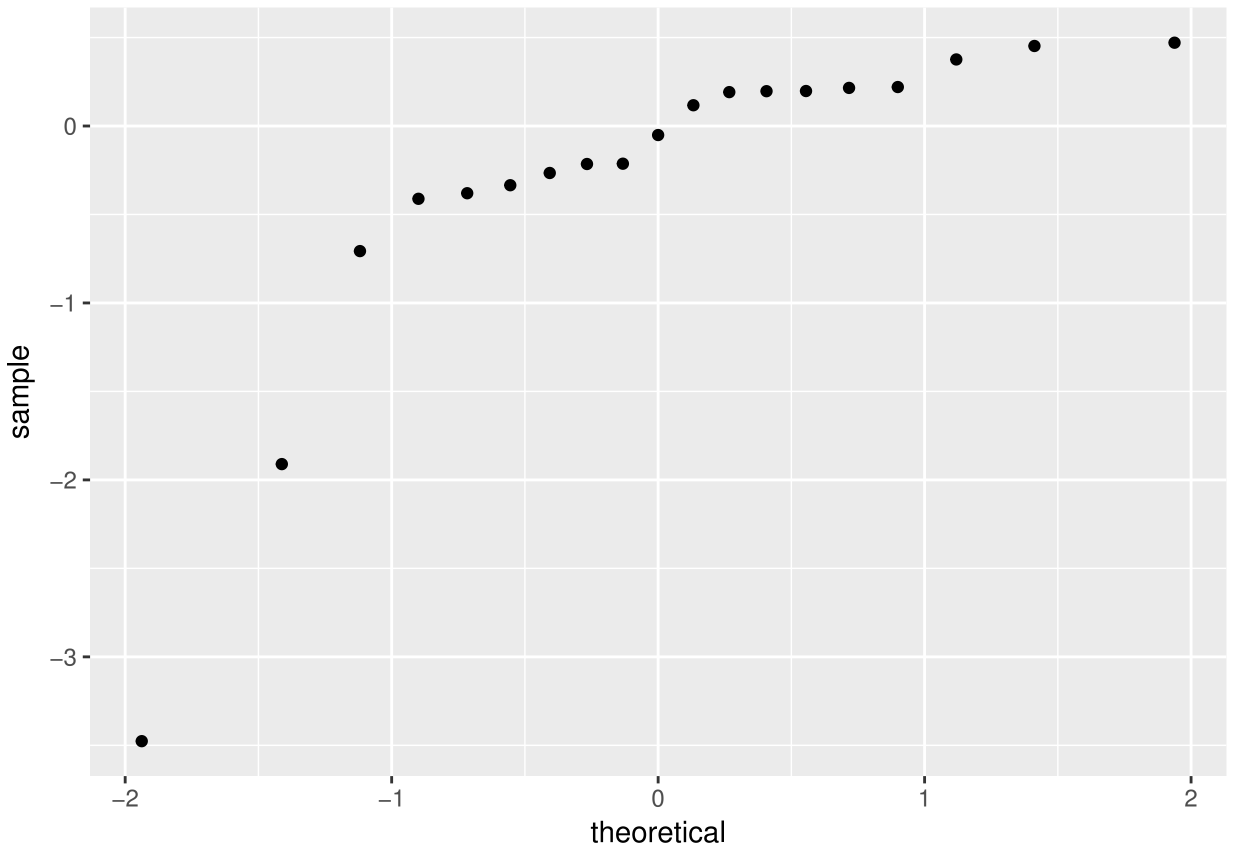
```
##          term   estimate  std.error statistic
## 1 (Intercept) -1.3373450 0.26620549 -5.023732
## 2      carbon  0.3010268 0.01981901 15.188793
```

```
dating.rlm.df = augment(dating.rlm)
ggplot(dating.rlm.df, aes(x = carbon, y = .resid)) + geom_point() + geom_abline(slope = 0)
```



Most of the residuals are scattered around zero, with the two exceptions being very large negative values. By using this robust method, we've managed to get a model that provides a very good fit for 17 out of 19 observations, though we must always keep in mind there will be some observations that don't fit the pattern. In addition, formal inference will be somewhat difficult. Look at the normal QQ plot of the residuals:

```r
ggplot(dating.rlm.df, aes(sample = .resid)) + stat_qq()
```



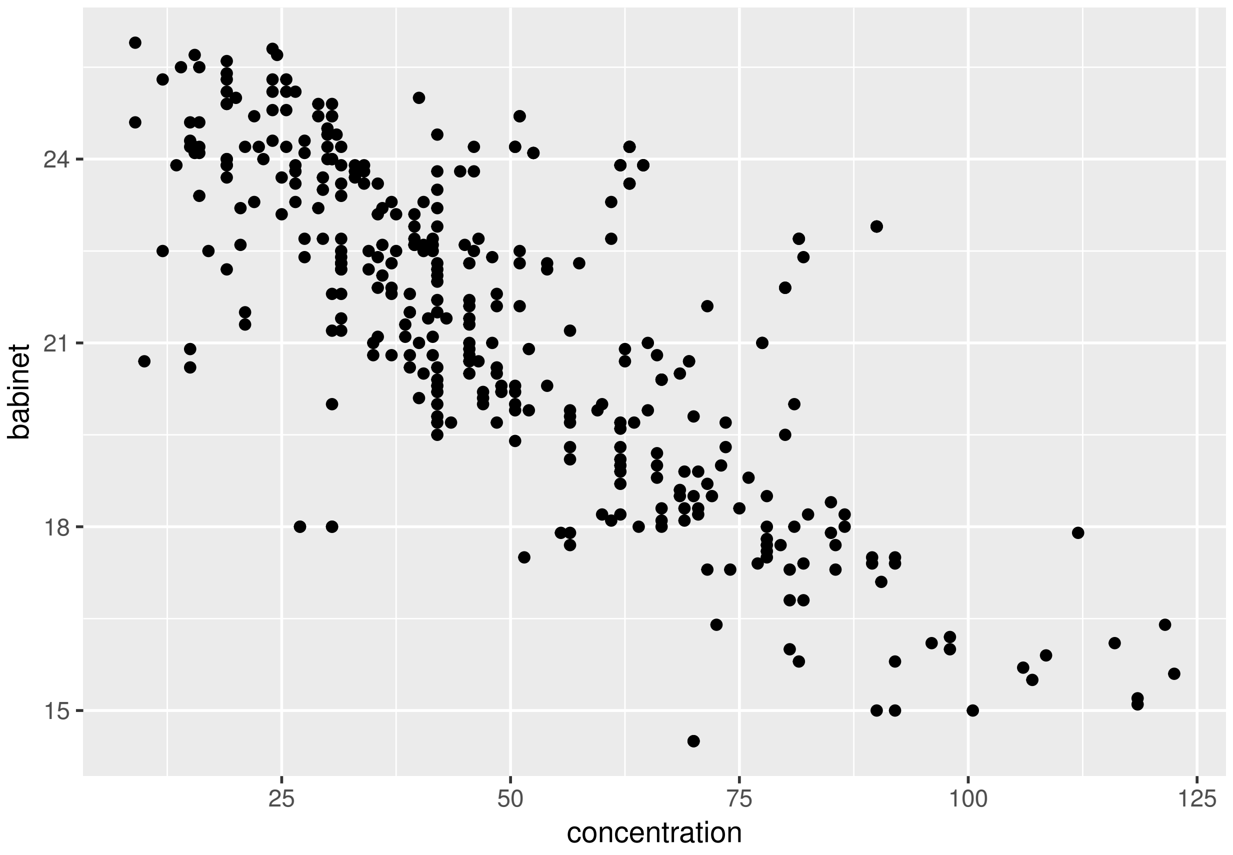
The outliers prevent the residuals from being well-modelled by a normal distribution, or even a symmetric one. So we should be very hesitant to make probabilistic statements about the data or model.

### 3.2.2 Particulates and the Babinet point

Light is *polarized* when the waves oscillate in a plane and *unpolarized* when they vibrate in all directions. Sunlight becomes unpolarized when its angle is greater than the *Babinet point*. The Babinet point depends on the concentration of particulates in the air.

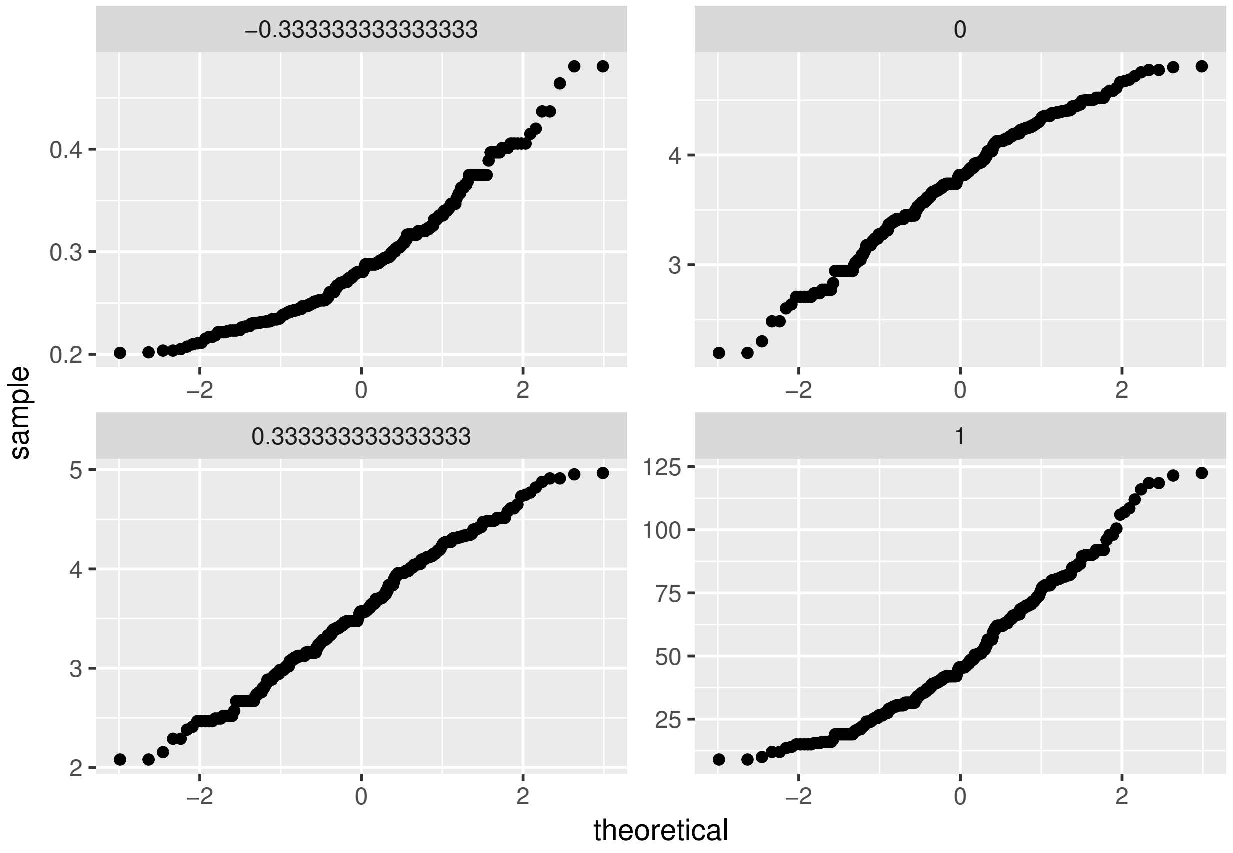
The `polarization` data set in the `lattice.RData` workspace contains 355 observations from an experiment to find the Babinet point (in degrees) while varying the particulate concentration (in micrograms per cubic meter.)

```
ggplot(polarization, aes(x = concentration, y = babinet)) + geom_point()
```



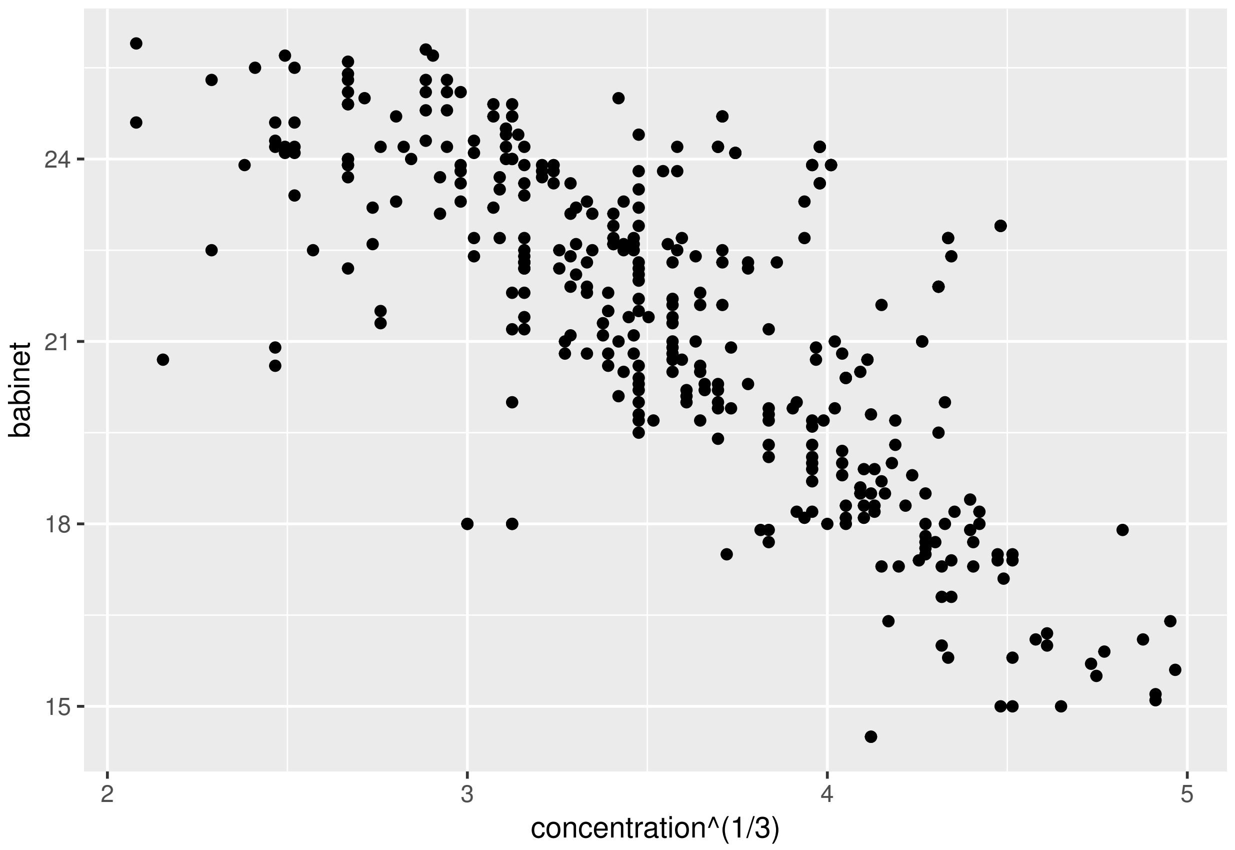
There's clearly a decreasing relationship. Note that because of rounding, some points are plotted on top of each other, so later we'll jitter the points a bit. First, we'll think about transformations. I'm usually against fractional powers, but because concentration has to do with volume it's reasonable to consider cube root transformations. We can compare powers of $1/3, -1/3$, and $-1$ to the untransformed data in terms of normality.

```
n = nrow(polarization)
power = rep(c(1, 1/3, 0, -1/3), each = n)
concentration = polarization$concentration
conc.trans = c(concentration, concentration^(1/3), log(concentration), concentration^(-1/3))
ggplot(data.frame(power, conc.trans), aes(sample = conc.trans)) + stat_qq() +
    facet_wrap(~power, scales = "free")
```



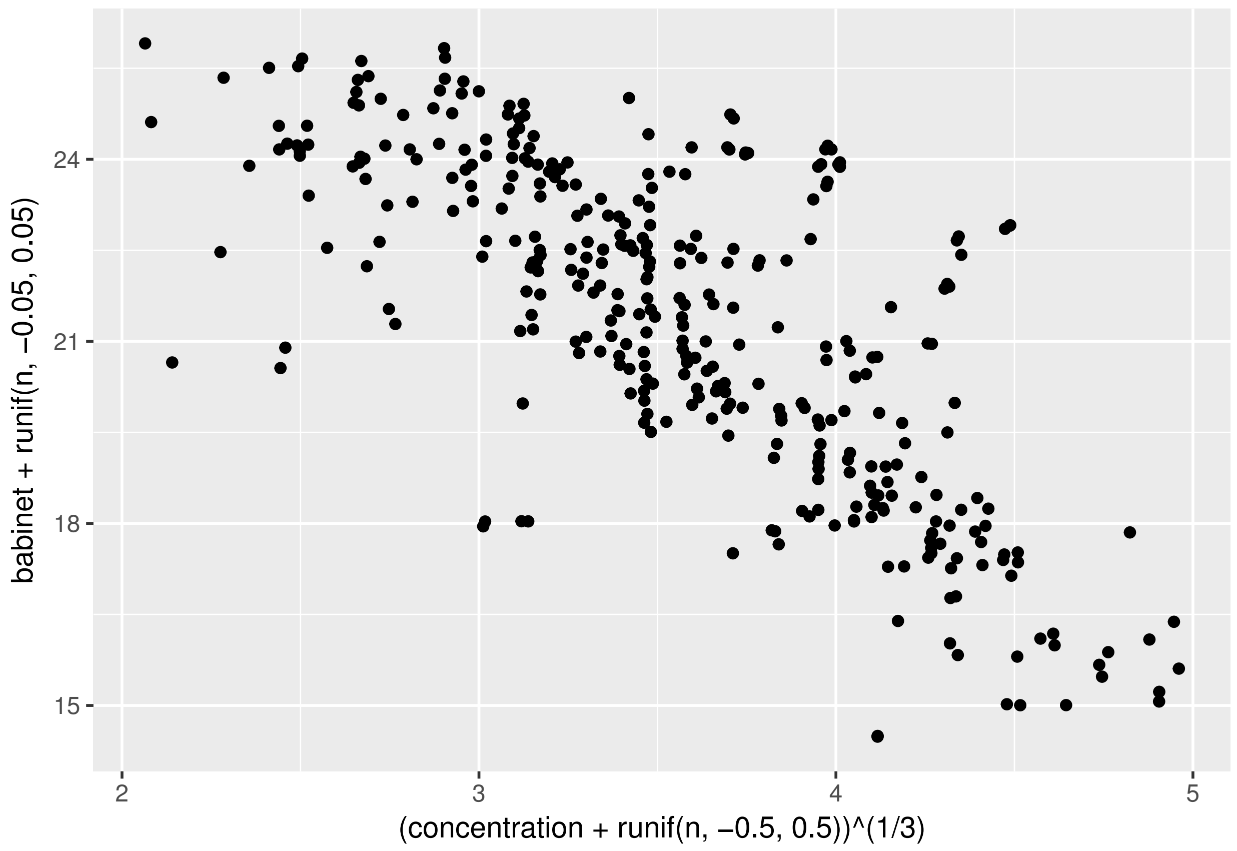
The cube root transformation comes closest to normality. (The tails might be a bit short, but this seems unimportant.) How does this look on a scatterplot?

```
ggplot(polarization, aes(x = concentration^(1/3), y = babinet)) + geom_point()
```



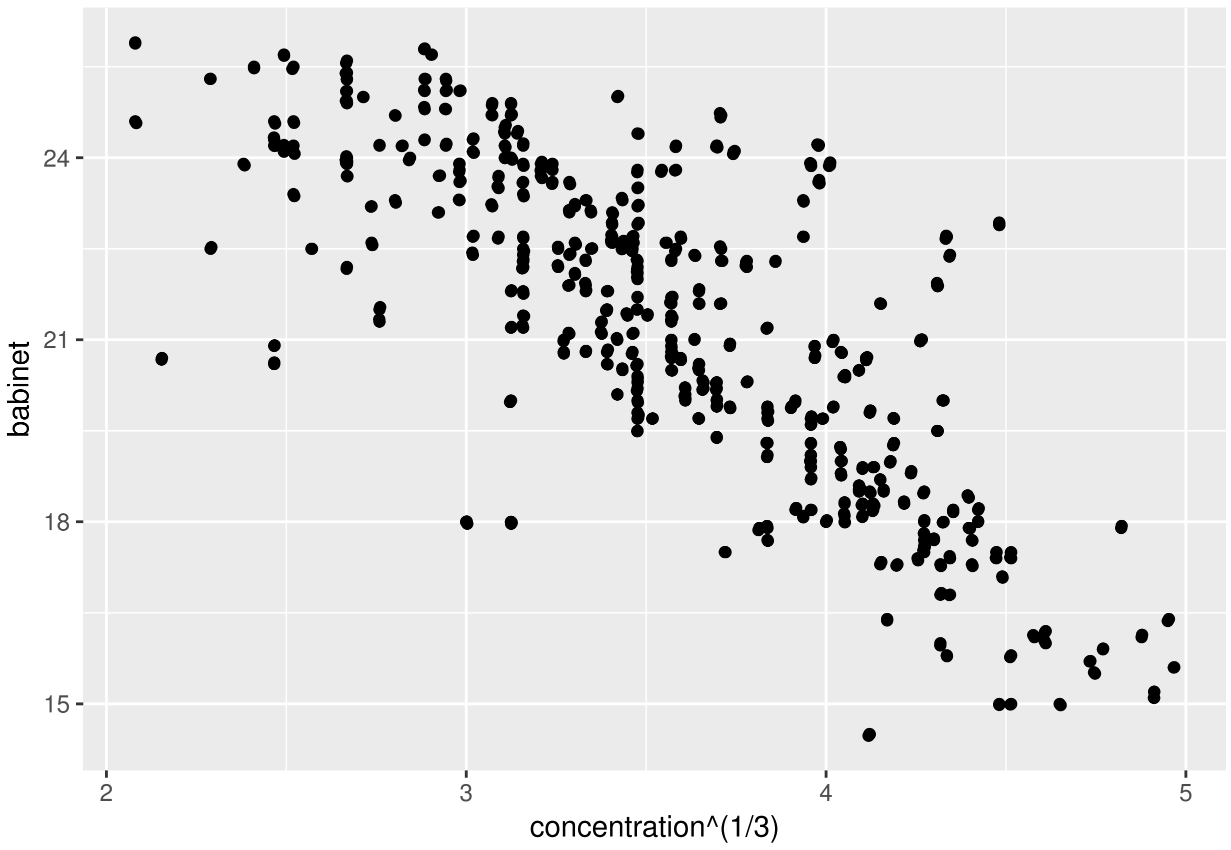
We still have the problem of repeated $x$- and $y$-values, so we jitter. We could "unround" the data by adding random noise manually:

```
ggplot(polarization, aes(x = (concentration + runif(n, -0.5, 0.5))^(1/3), y = babinet +
    runif(n, -0.05, 0.05))) + geom_point()
```



Or we could use the automatic `geom_jitter()` function:

```
ggplot(polarization, aes(x = concentration^(1/3), y = babinet)) + geom_point() +
    geom_jitter()
```



The manual jittering arguably looks a little better, but really it doesn't make much difference.

Let's now fit a **loess** curve to the data. Loess is a form of *local polynomial regression*, meaning that at every $x$-value, it fits a weighted polynomial model "locally": data at nearby $x$-values will be weighted heavily, while data at far away $x$-value will be downweighted or not considered at all if they fall outside a "neighborhood."" See a reliable nonparametric statistics text, or failing that, Luen's *Some of Nonparametric Statistics* for details.

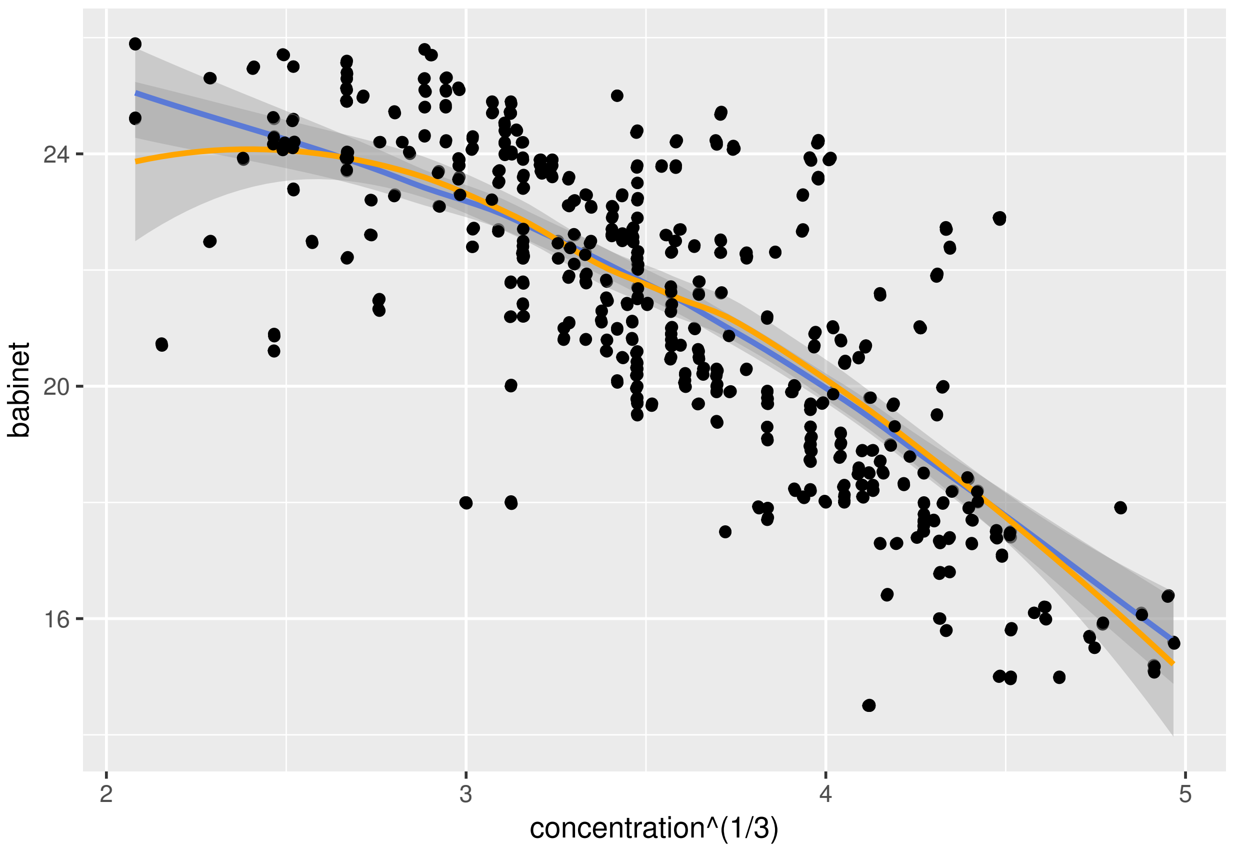
There are three main arguments to play around with:

- `span`: The proportion of the data included in the neighborhood. Default is 0.75.
- `degree`: The degree of polynomial fitted. Default is 2 (locally quadratic) but for data that isn't too wiggly, degree 1 (locally linear) may be less likely to do weird stuff at the extremes.
- `family`: The default, `gaussian`, uses (weighted) least squares to fit the local polynomial. `symmetric` uses bisquare to be more resistant to outliers.

If you're optimizing for prediction, then the tuning parameters should be found using cross-validation or some similar method. If you're doing EDA, playing around with the parameters until you get something that looks good is fine. Generally the default `span` works surprisingly adequately for non-weird data sets, while it's often worth trying both degrees 1 and 2 and visually determining which appears to give the better fit. It's also hard to say if `symmetric` definitely does better than least squares unless there are obvious outlier problems.

Going back to the polarization data, let's compare the degree 1 and 2 fits.

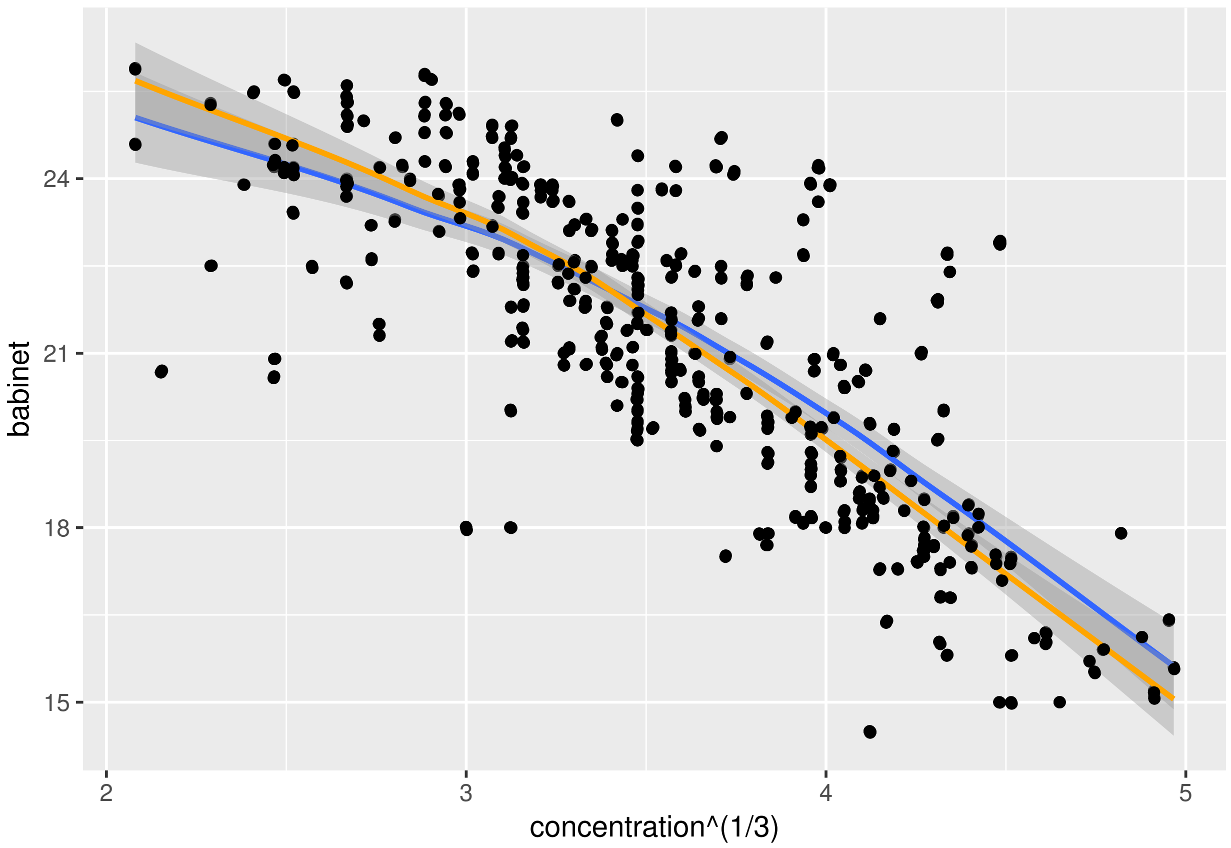
```
ggplot(polarization, aes(x = concentration^(1/3), y = babinet)) + geom_point() +
    geom_smooth(method.args = list(degree = 1)) + geom_smooth(method.args = list(degree = 2),
    col = "orange") + geom_jitter()
```



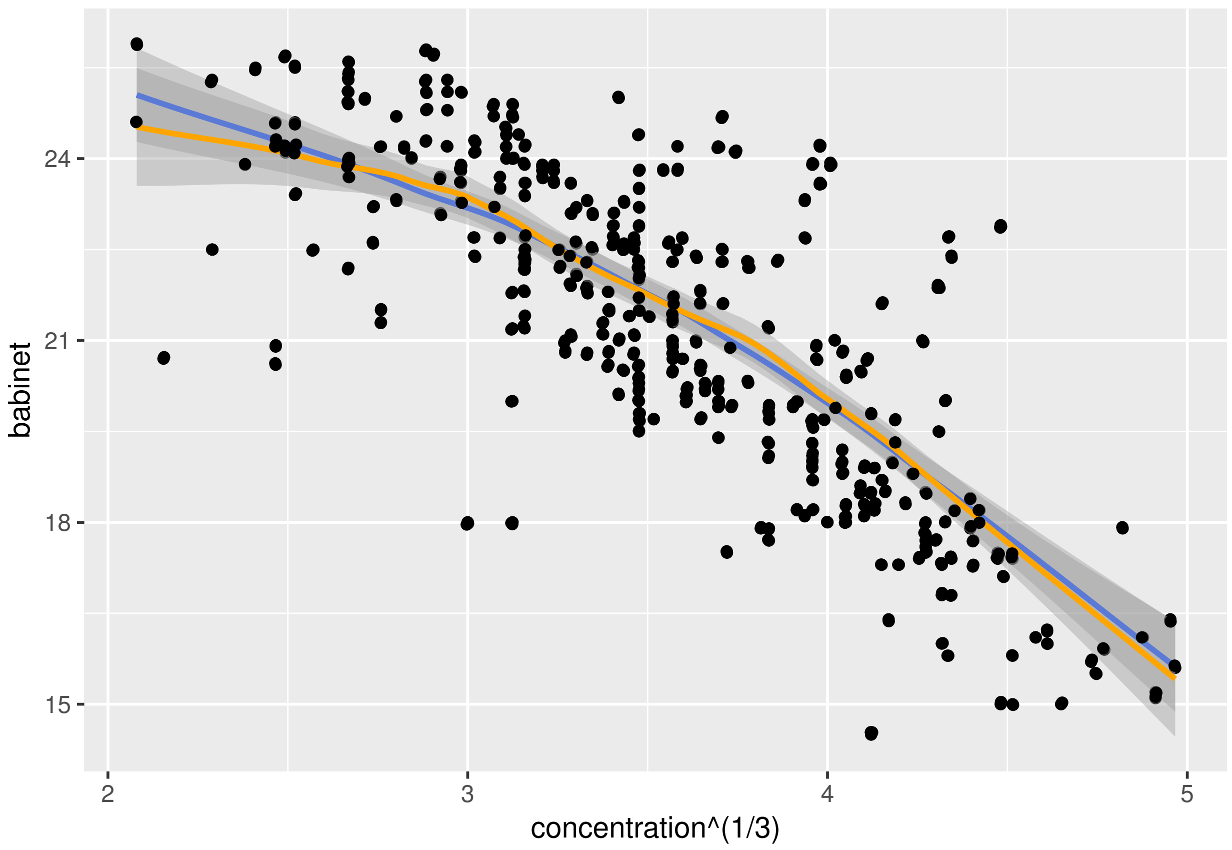
The locally linear fit is in blue, while the locally quadratic fit is in orange. We see the orange fit is bent strong on the left hand side, but there isn't much data to justify this. Subjectively, I'd argue for the degree 1 fit.

Other stuff to try:

```r
ggplot(polarization, aes(x = concentration^(1/3), y = babinet)) + geom_point() +
    geom_smooth(method.args = list(degree = 1)) + geom_smooth(method.args = list(degree = 1,
    family = "symmetric"), col = "orange") + geom_jitter()
```

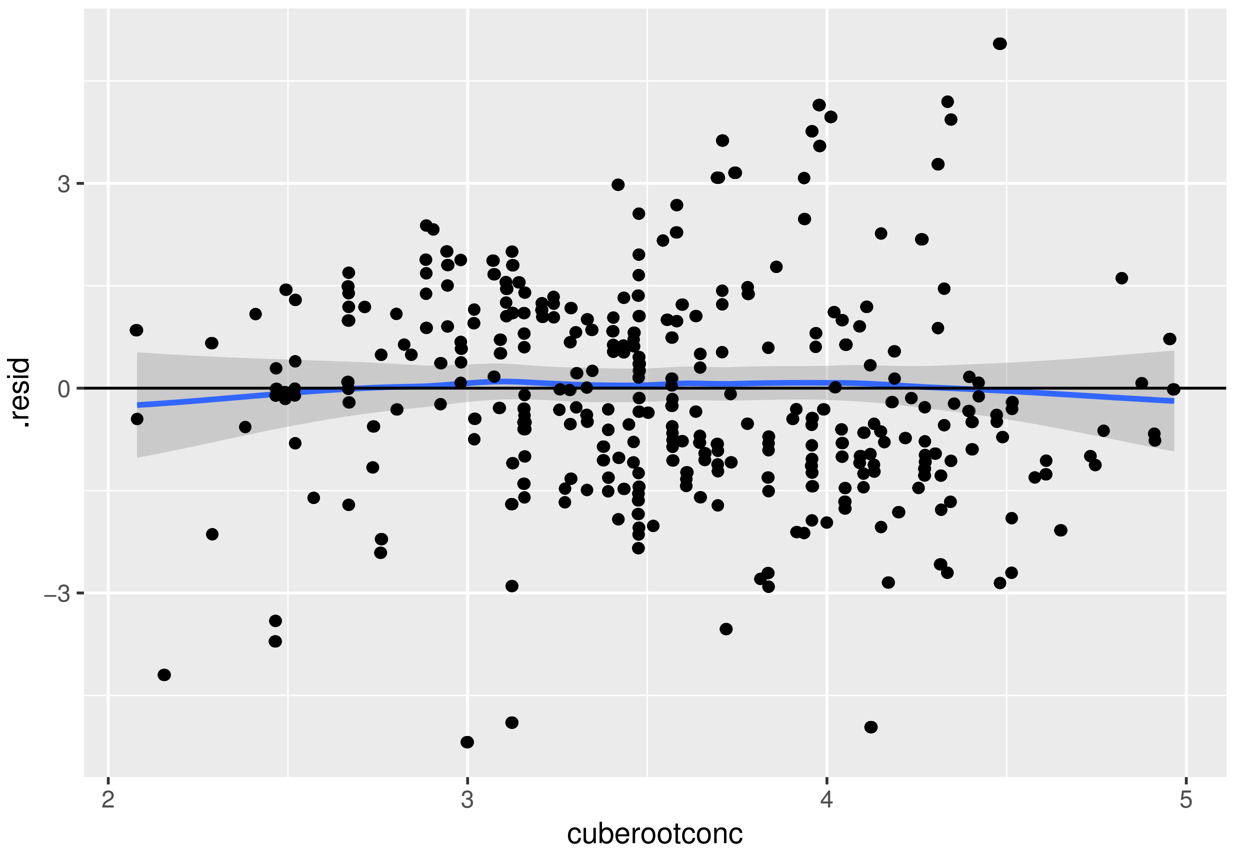


```
ggplot(polarization, aes(x = concentration^(1/3), y = babinet)) + geom_point() +
    geom_smooth(method.args = list(degree = 1)) + geom_smooth(span = 0.5, method.args = list(degree = 1)
    col = "orange") + geom_jitter()
```



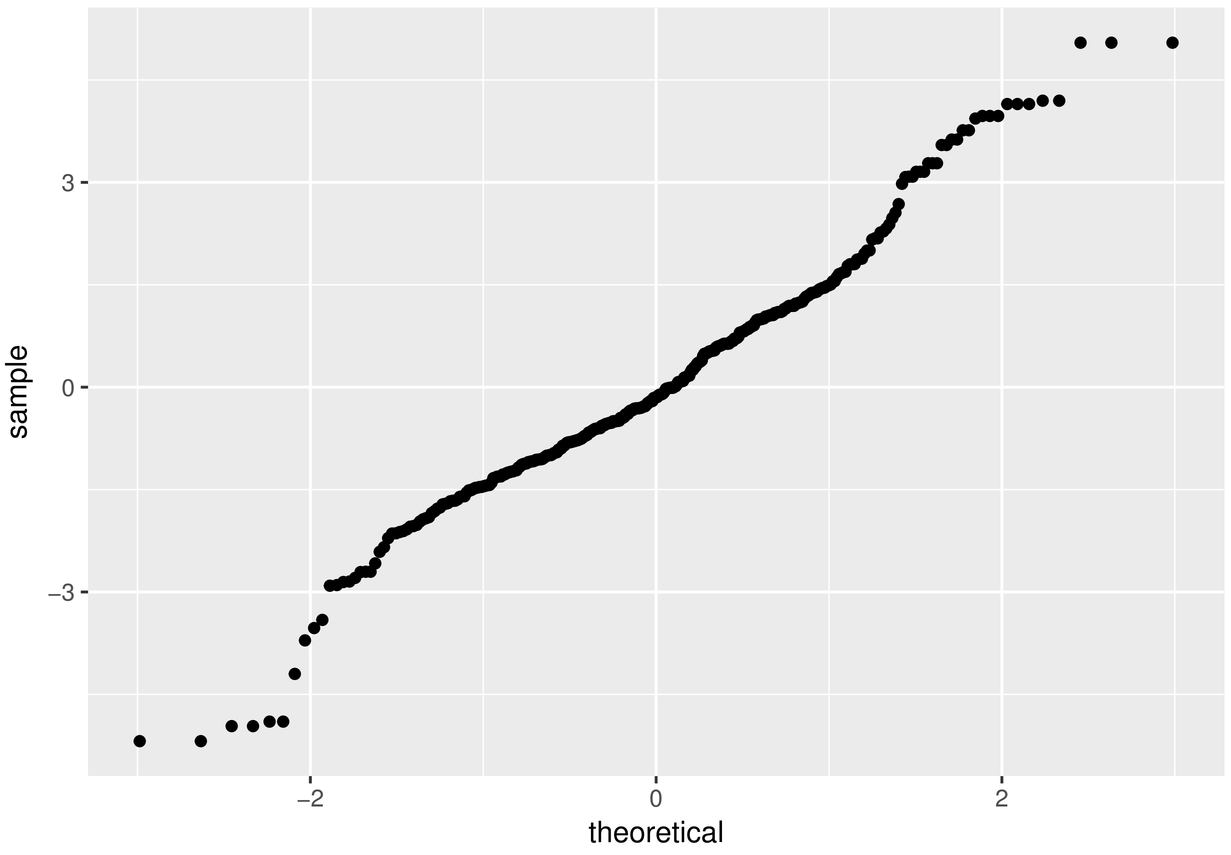
There's no single right answer here. Let's try a degree 1 fit and leave everything else at its default. Now refit the model and plot the residuals with another loess fit. For residual plots, a degree 1 loess fit is often more appropriate than a quadratic:

```
cuberootconc = (polarization$concentration)^(1/3)
babinet = polarization$babinet
polar.lo = loess(babinet ~ cuberootconc, degree = 1)
polar.lo.df = augment(polar.lo)
ggplot(polar.lo.df, aes(x = cuberootconc, y = .resid)) + geom_point() + geom_smooth(method.args = list(
    geom_abline(slope = 0) + geom_jitter(height = 0)
```



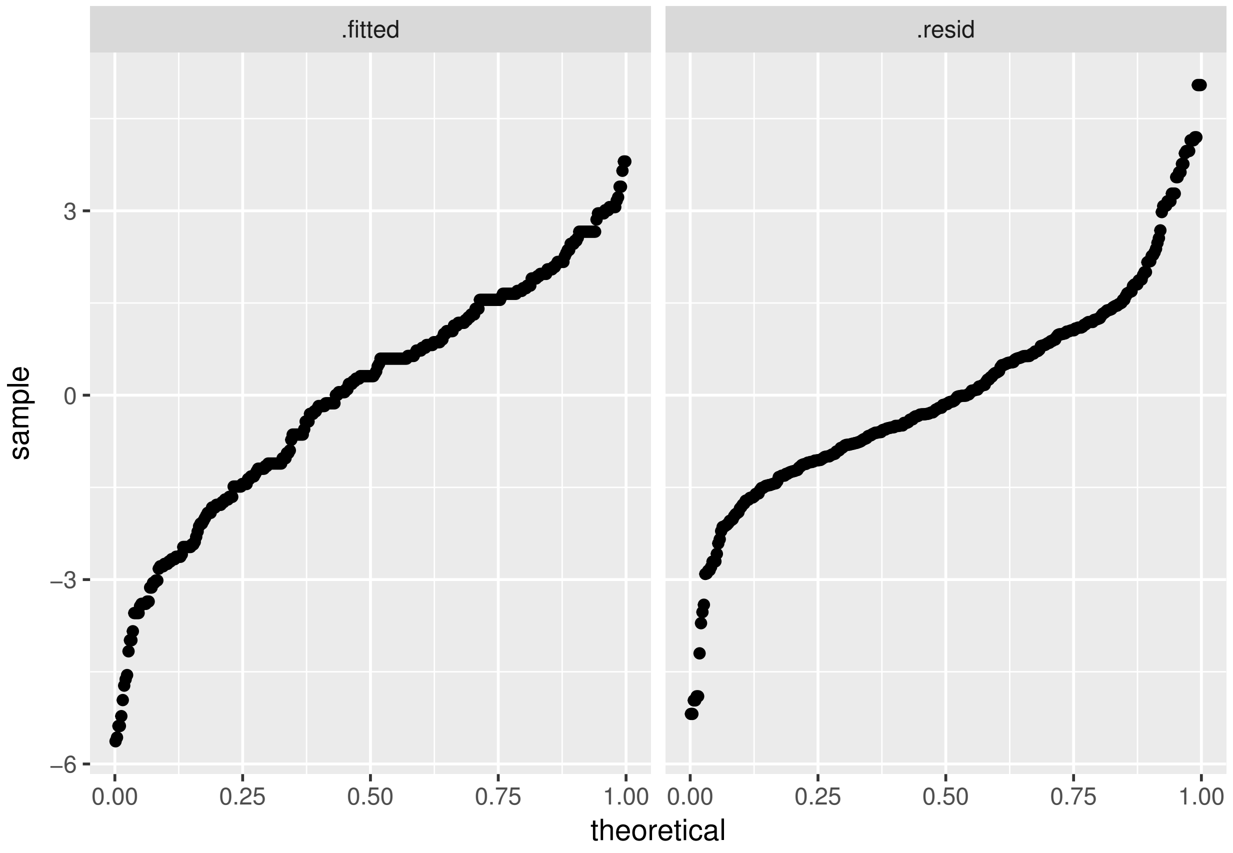
It doesn't look like there's any trend. The fact that there are quite a few large positive residuals for cube root concentration between 3.7 and 4.5 is a little bit worrying; we'll take a closer look at this later. Before that, we'll check normality:

```
ggplot(polar.lo.df, aes(sample = .resid)) + stat_qq()
```



The normal distribution breaks down a bit at the edges. This is unfortunate but not a dealbreaker. To get an idea of how much variation we've explained, look at a residual-fit plot:

```r
polar.lo.df$.fitted = polar.lo.df$.fitted - mean(polar.lo.df$.fitted)
library(tidyr)
polar.lo.df.long = polar.lo.df %>% gather(type, value, c(.fitted, .resid))
ggplot(polar.lo.df.long, aes(sample = value)) + stat_qq(distribution = qunif) +
    facet_grid(~type)
```



Note that we drew this a bit differently this time: we overwrote the fitted values with their mean-removed versions, then changed to long form and faceted by "type" (fitted value or residual.) The variation of the two graphs is comparable. We can also assess this quantitatively:

```
var(polar.lo.df$.fitted)
```

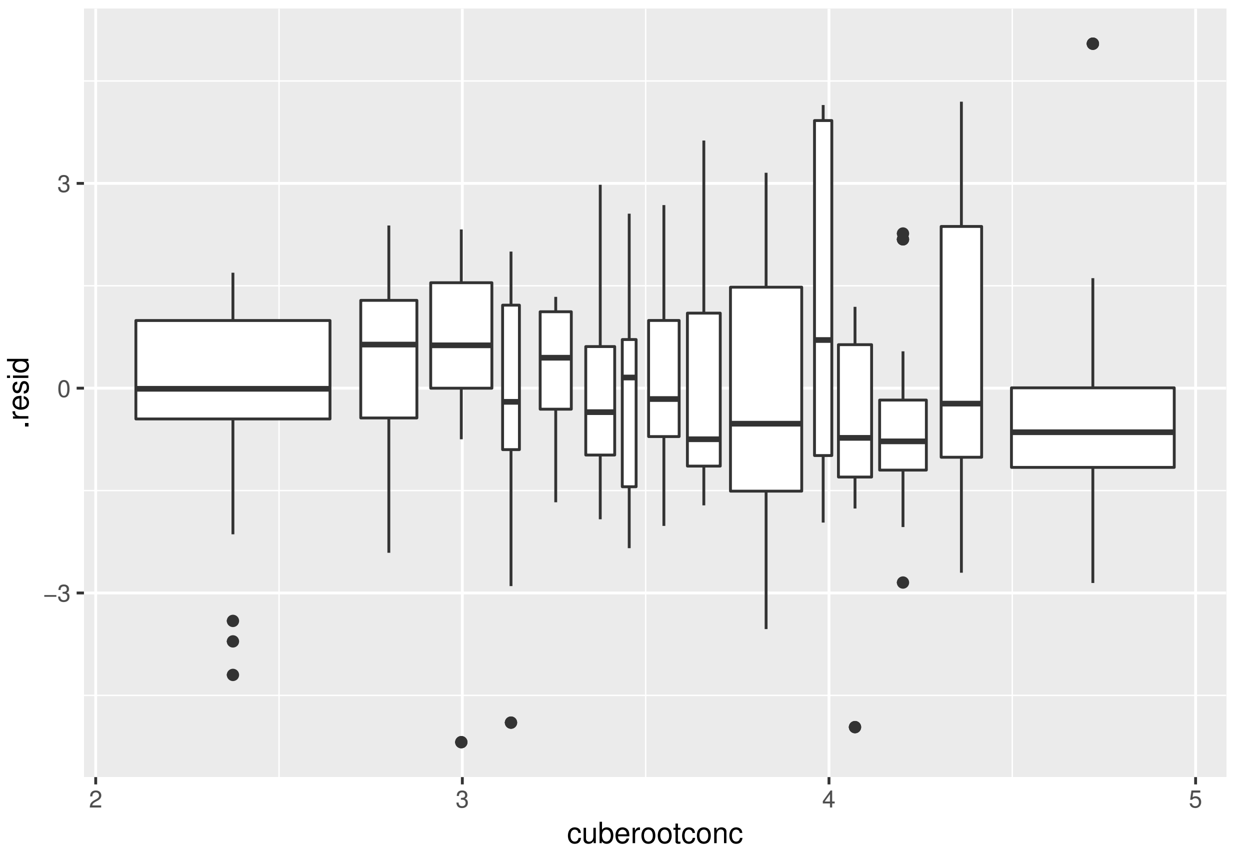
```
## [1] 4.043584
```

```
var(polar.lo.df$.resid)
```

```
## [1] 3.076718
```

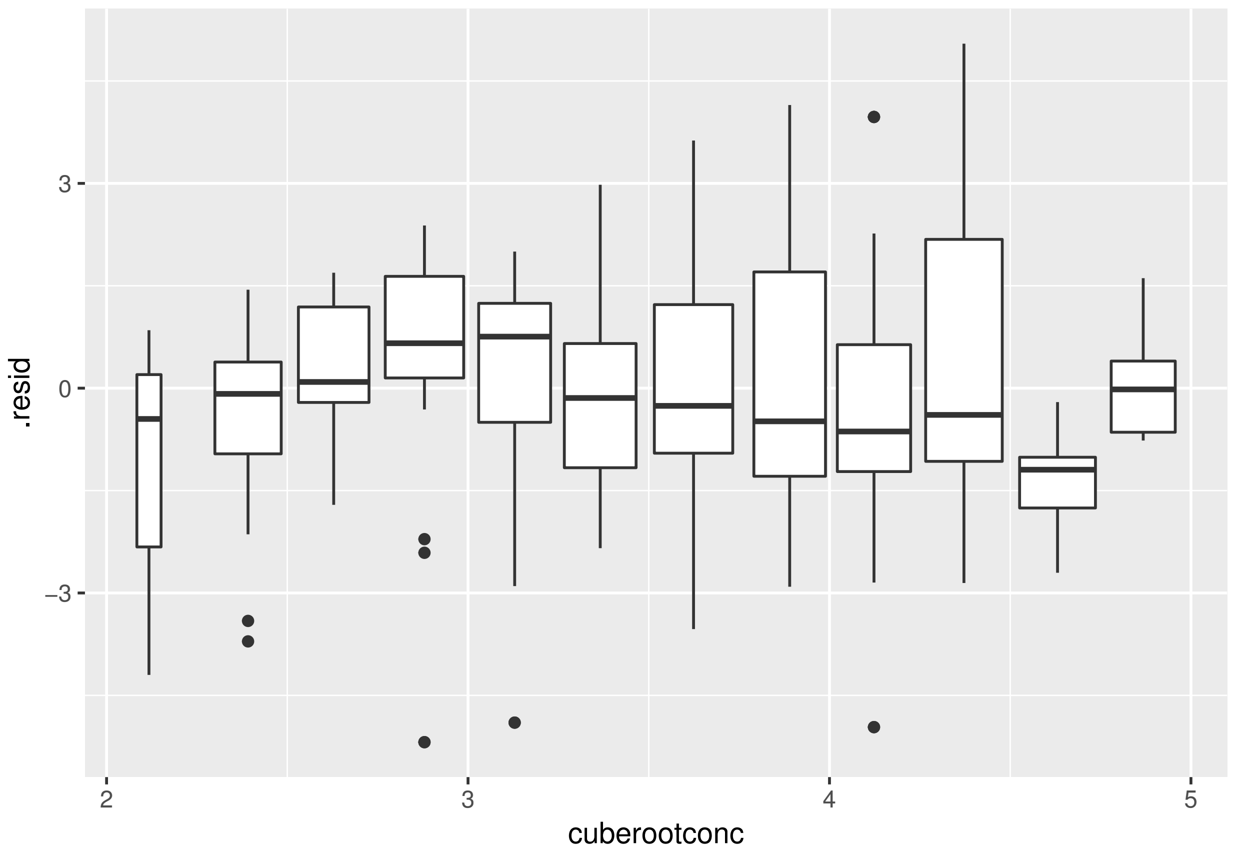
To return to the earlier issue, we want to see if the distribution of the residuals changes with (cube root of) concentration. An easy way to do this is with side-by-side boxplots. We can "cut" the range of $x$-values into a given number of intervals, each with (approximately) equal numbers of observations:

```
ggplot(polar.lo.df, aes(x = cuberootconc, y = .resid)) + geom_boxplot(aes(group = cut_number(cuberootcon
    n = 15)))
```



Or we can specify the $x$-axis bins explicitly:

```
ggplot(polar.lo.df, aes(x = cuberootconc, y = .resid)) + geom_boxplot(aes(group = cut(cuberootconc,
    breaks = seq(2, 5, 0.25))))
```



Either way, it seems like the boxes certainly change as we go from left to right, and the change seems quite complicated (it's not merely heteroskedasticity.) So we've succeeded in modeling the trend, but it's much harder to draw probabilistic inferences. If we wanted to do this, the only good solution with such complex relationships is to get more data.

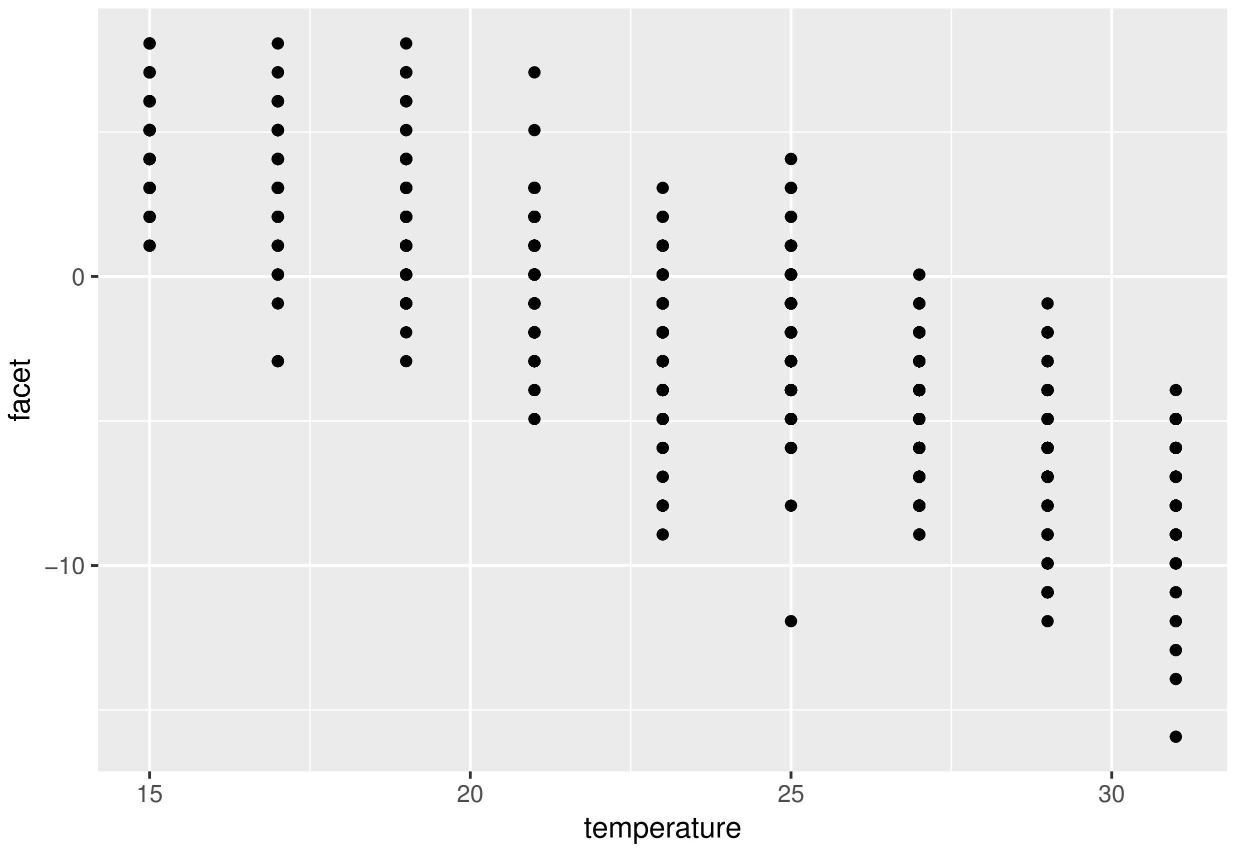
## 3.3 Fixed x-values

### 3.3.1 Fly eyes

The data set `fly` contains two variables from an experiment about breeding flies (see Hersh, The effect of temperature upon the heterozygotes in the series of Drosophilia):

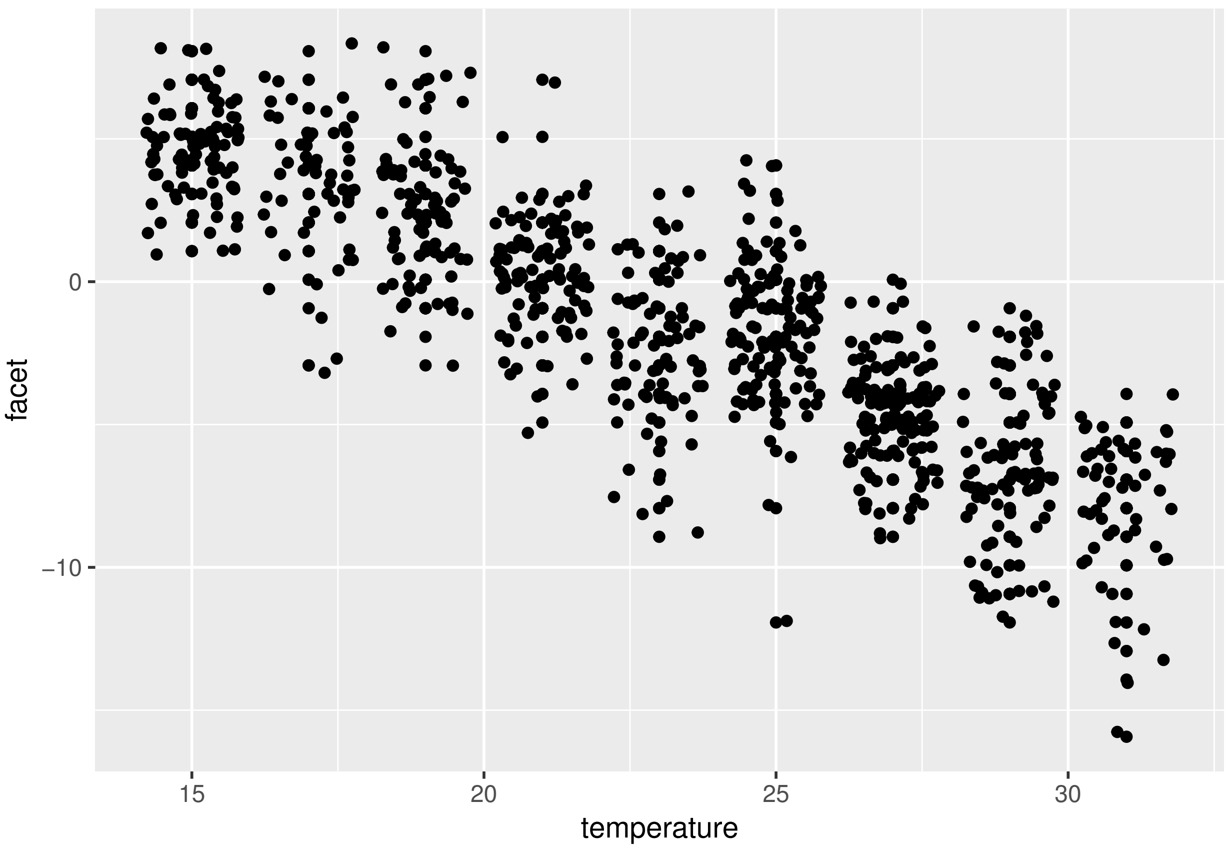
- `temperature` of the aquarium in which flies were hatched (in degrees Celsius);
- `facet`: a measure of the number of facets in a fly's eye (on an "essentially logarithmic" scale.)

```
load("lattice.RData")
library(ggplot2)
ggplot(fly, aes(x = temperature, y = facet)) + geom_point()
```



With only nine $x$-values, many observations are on top of each other. Jitter the data:

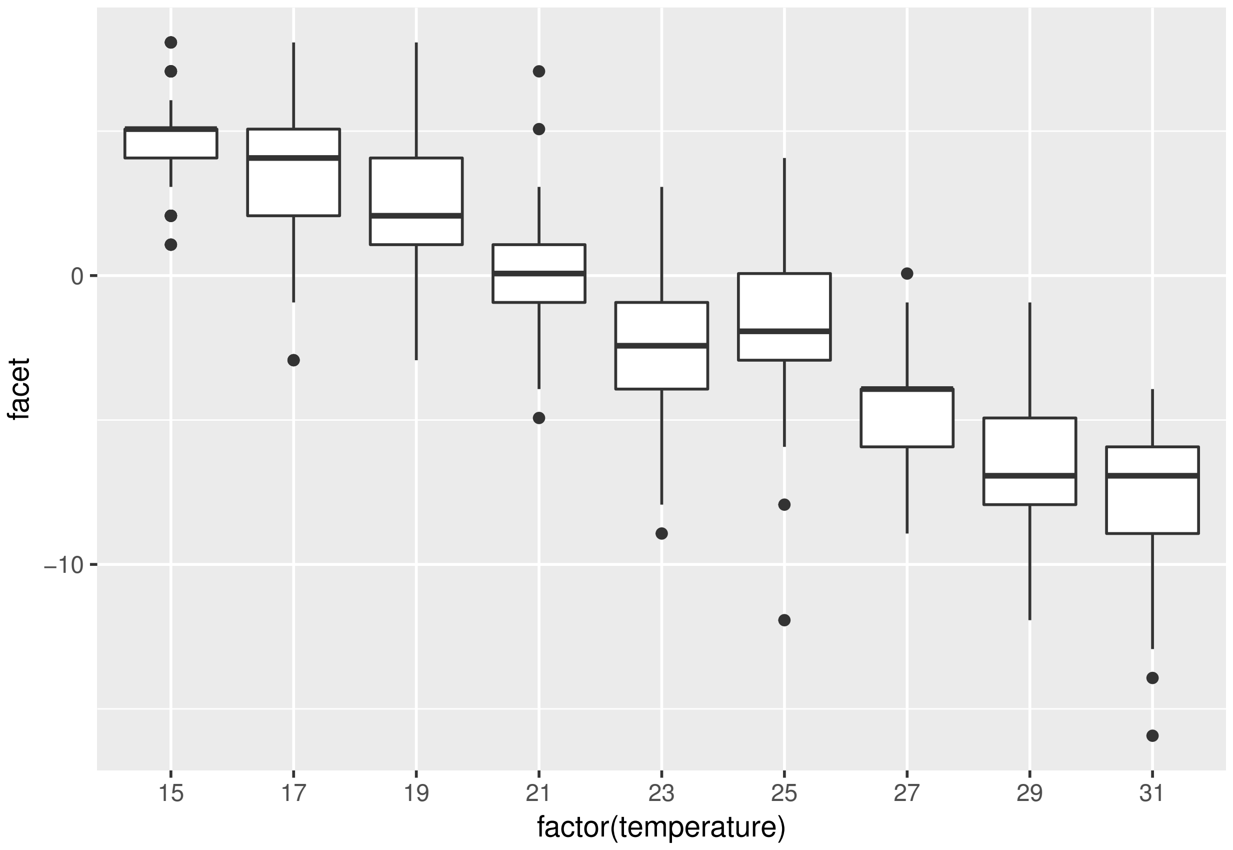
```
ggplot(fly, aes(x = temperature, y = facet)) + geom_point() + geom_jitter()
```



The facet variable looks like it decreases approixmately linearly with temperature. However, the data at $x = 23$ seems to be a bit lower than we'd expect given the rest of the data. R.A. Fisher did an analysis of variance that found the deviations from the line were statistically significant, but that doesn't tell you much other than the probability model isn't literally true.

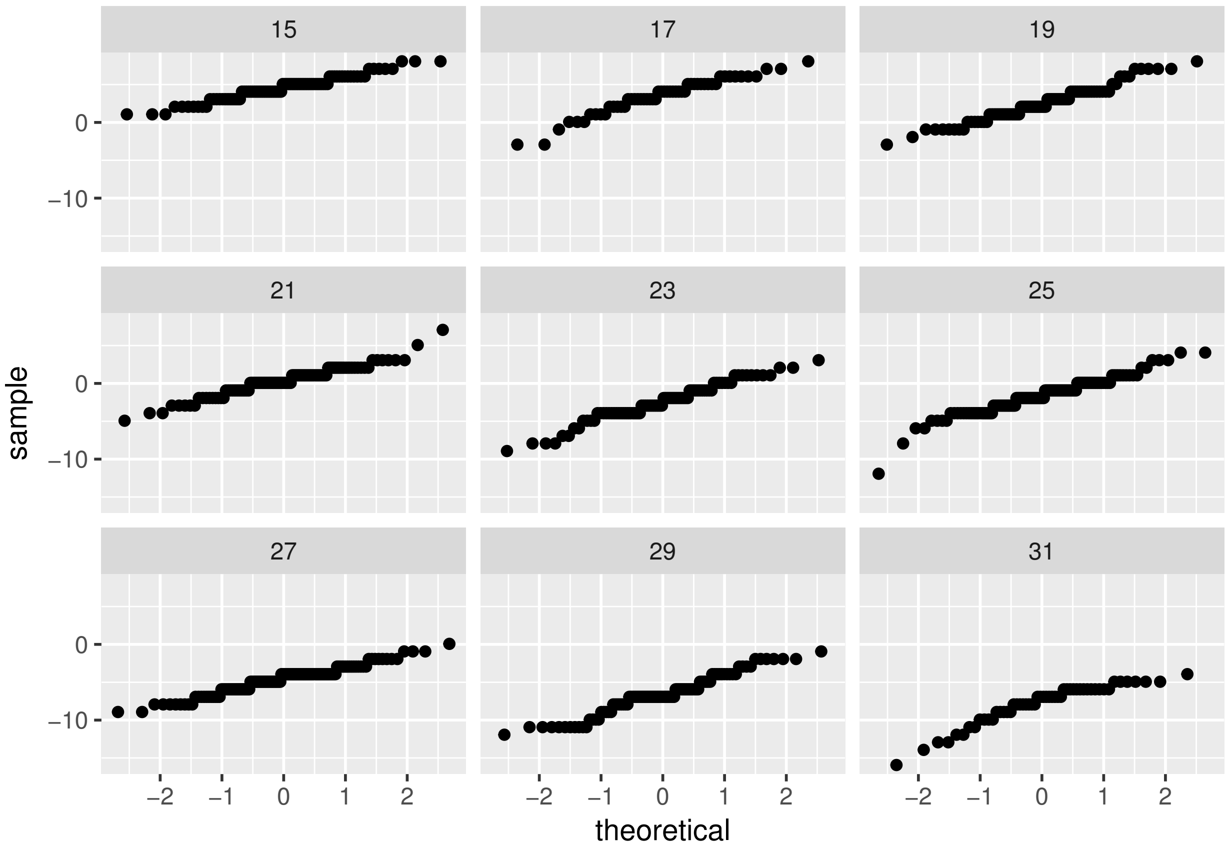
Another way to visualize this is through side-by-side box plots:

```
ggplot(fly, aes(x = factor(temperature), y = facet)) + geom_boxplot()
```



The decreasing trend is again apparent. Can the data be well-modeled by a linear model with normal errors? Boxplots aren't a good guide to the shape of distributions, so let's draw normal QQ plots.

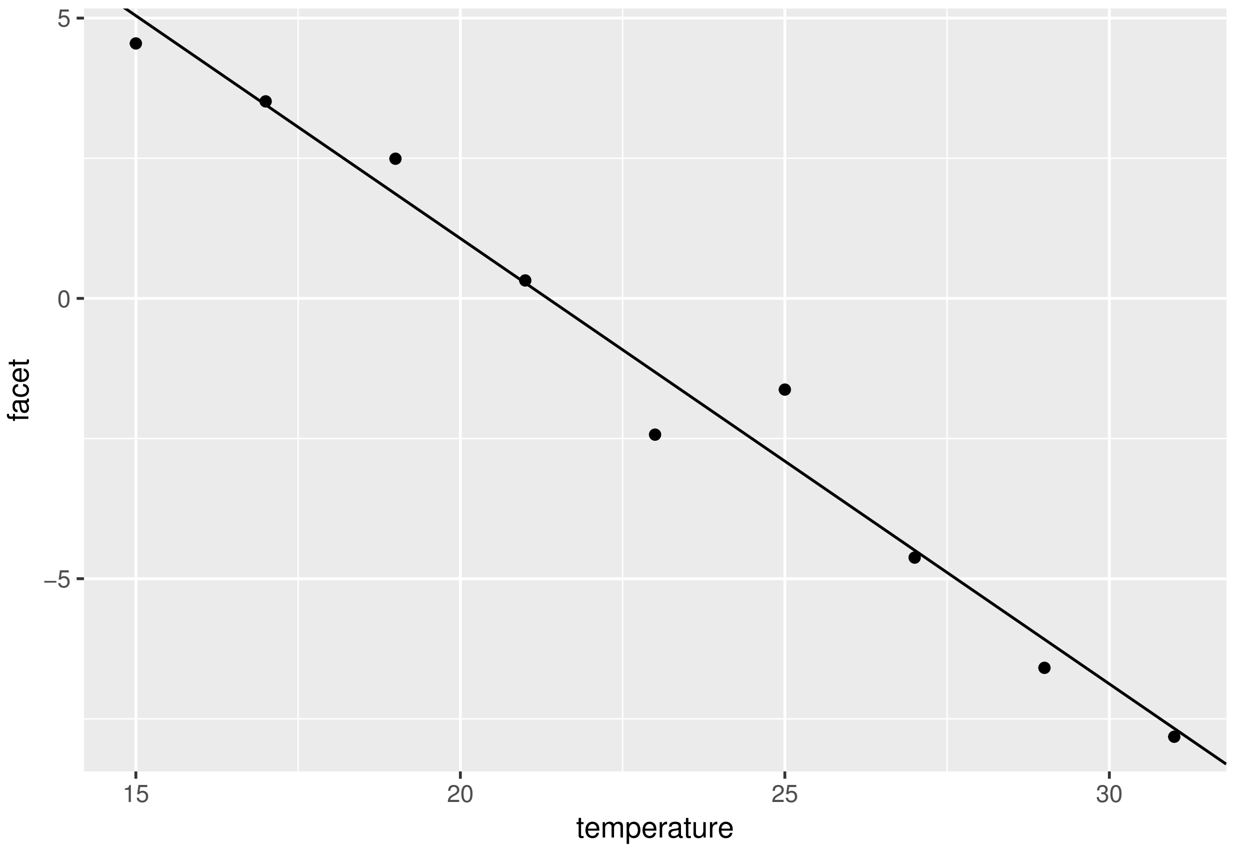
```r
ggplot(fly, aes(sample = facet)) + stat_qq() + facet_wrap(~temperature, ncol = 3)
```



The plots generally look like straight lines: the last one (for 31 degrees) might be a bit off, but that's a small concern. Furthermore, the slopes look similar for each plot, which is consistent with homoscedasticity.

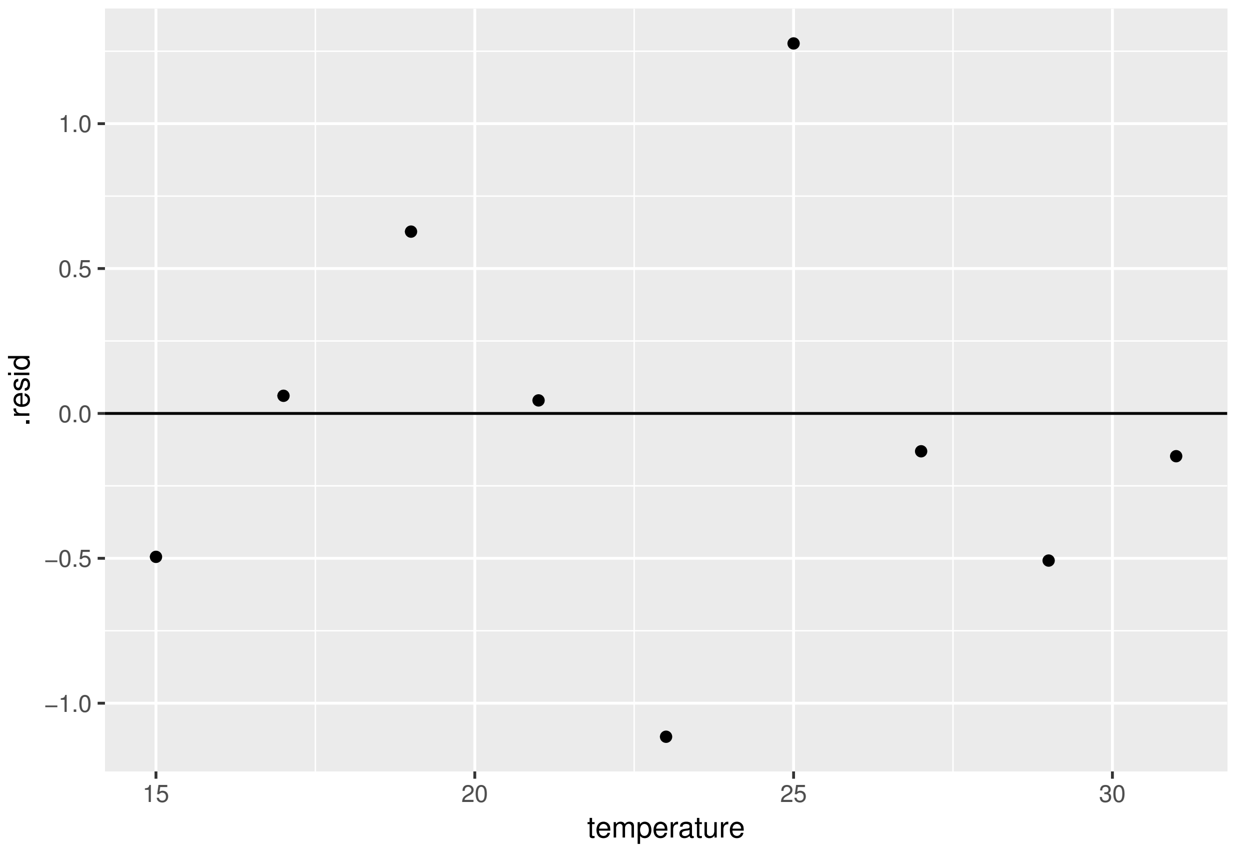
We now try fitting the linear model. Instead of drawing the line over all the data (which is messy due to the repeated $x$-values,) we plot the mean for each $x$-value.

```
facet.means = aggregate(facet ~ temperature, mean, data = fly)
facet.lm = lm(facet ~ temperature, data = fly)
ggplot(facet.means, aes(x = temperature, y = facet)) + geom_point() + geom_abline(intercept = facet.lm$
    slope = facet.lm$coe[2])
```



The line fits the means well, and the mean for 23 degrees doesn't seem like it's too far from the line. To look at this more clearly, plot the mean residual for each $x$:

```
library(broom)
facet.lm.df = augment(facet.lm)
facet.resid.means = aggregate(.resid ~ temperature, mean, data = facet.lm.df)
ggplot(facet.resid.means, aes(x = temperature, y = .resid)) + geom_point() +
    geom_abline(slope = 0)
```



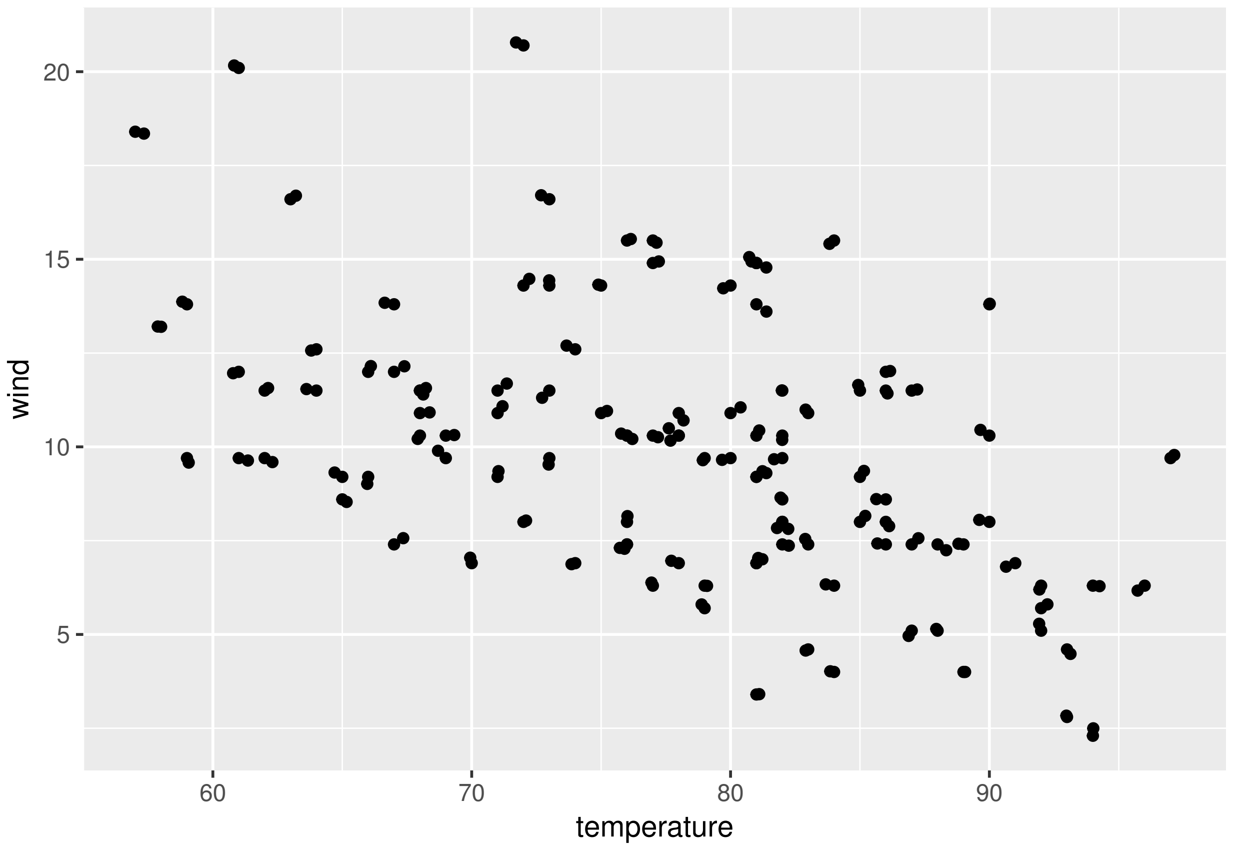
The largest residuals are a bit over 1 degree in magnitude. According to the original paper, when the experiment was performed, the temperature in the 23 aquarium varied by up to 3 degrees. Looking at the last two residual plots, this would be a sufficient explanation for the large residual for that $x$-value.

## 3.4 Truly bivariate data

### 3.4.1 Wind and temperature

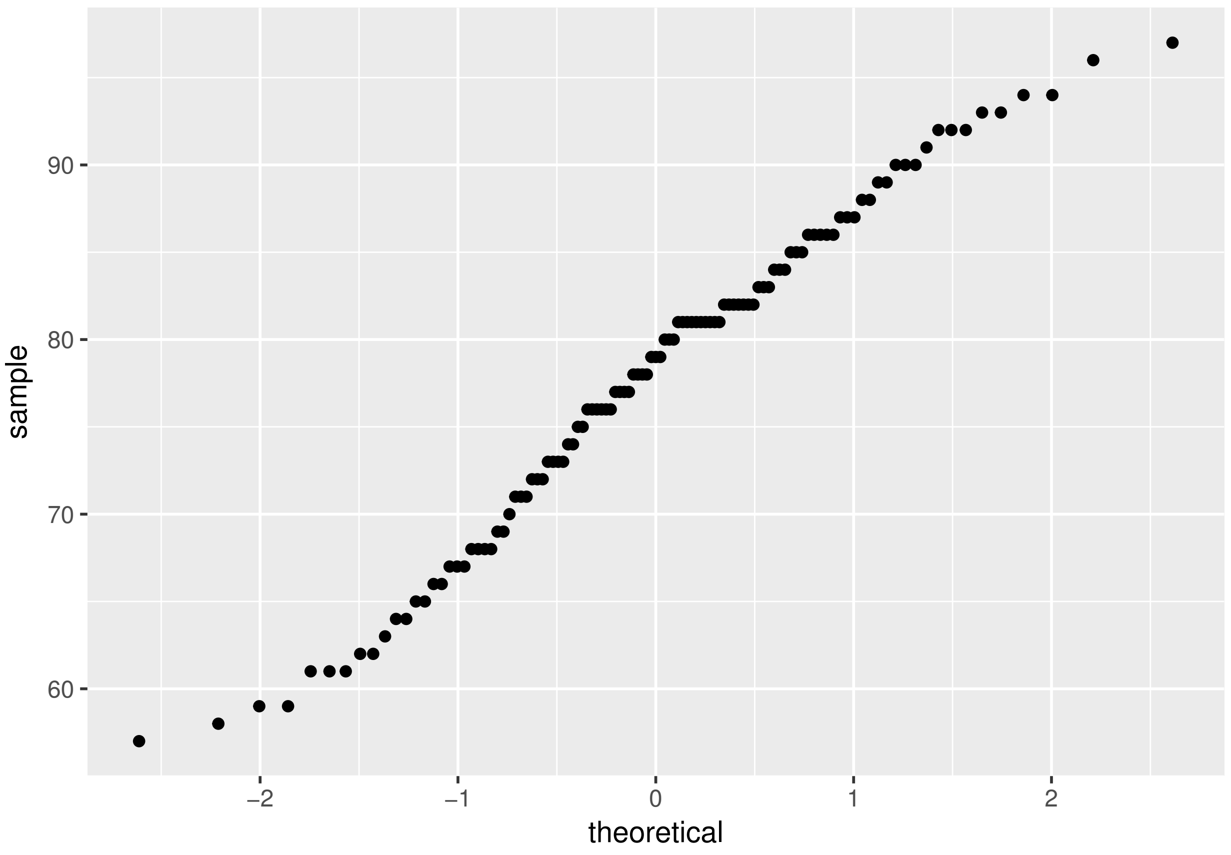
The data set `environmental` contains `temperature` and `wind` measurements on 111 days from May to September 1973. This data is "truly" bivariate, in that it's not appropriate to declare one variable as explanatory and one as the response: both variables depend on each other in some sense. We still start by drawing a (jittered) scatterplot, arbitrarily taking `temperature` as our $x$-variable:

```r
ggplot(environmental, aes(x = temperature, y = wind)) + geom_point() + geom_jitter()
```



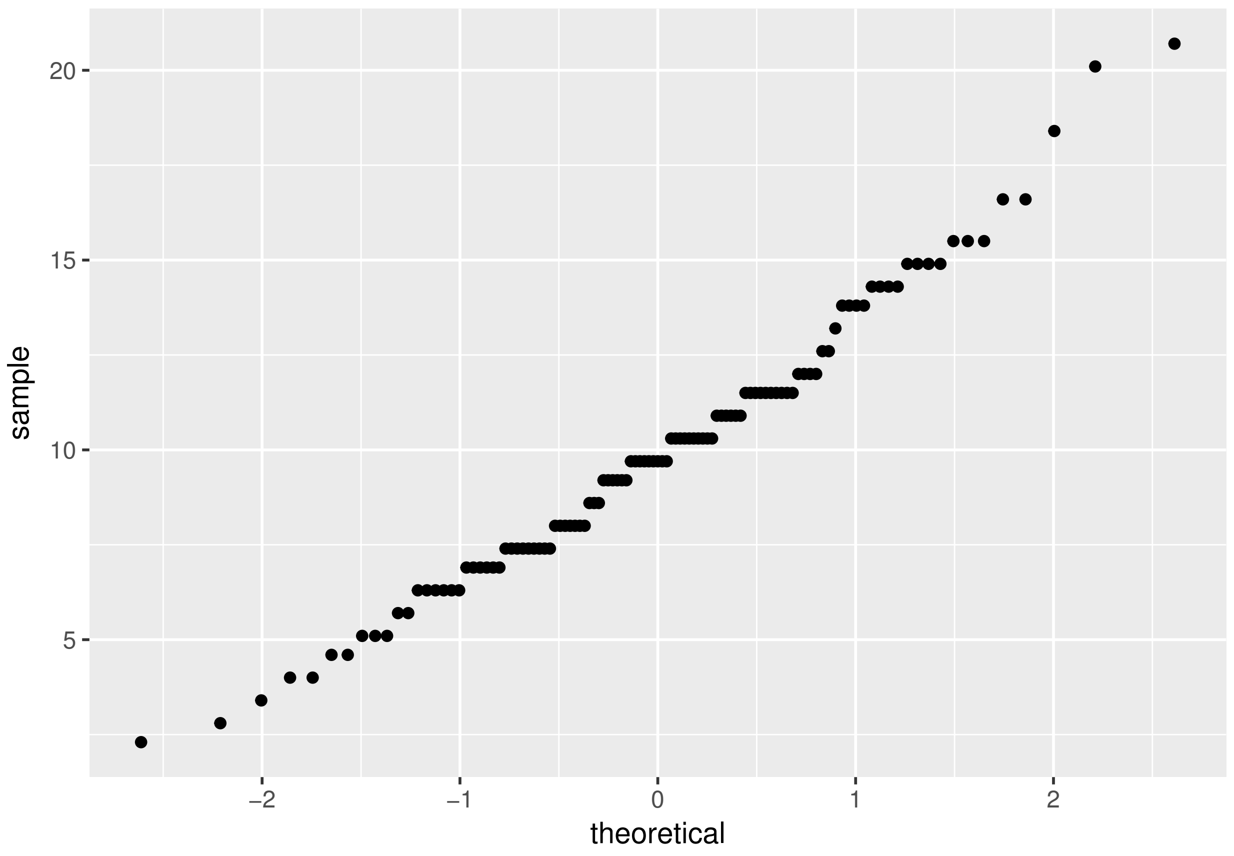
This doesn't too far from the ellipse shape that we see with bivariate normal data. To examine normality more carefully, draw QQ plots. First, temperature:

```
ggplot(environmental, aes(sample = temperature)) + stat_qq()
```



Next, wind:

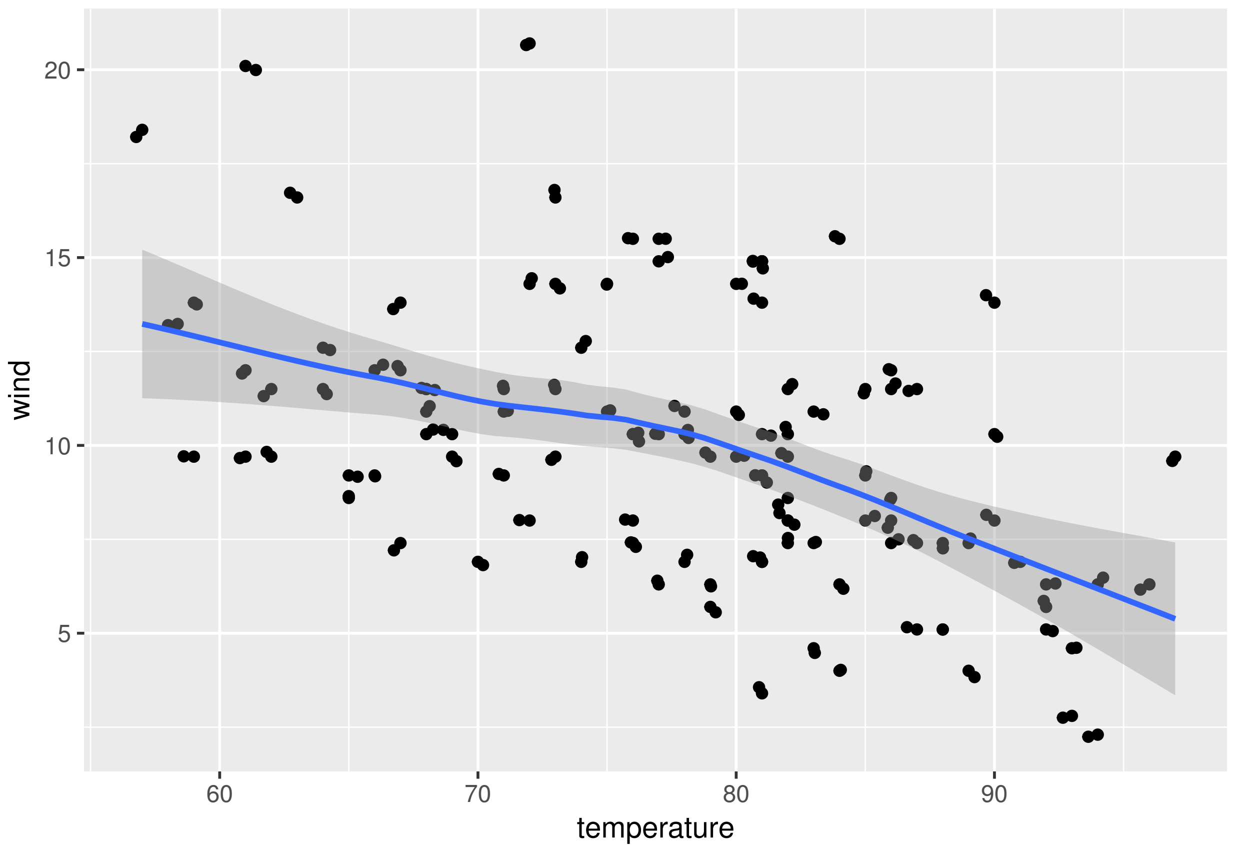
```
ggplot(environmental, aes(sample = wind)) + stat_qq()
```



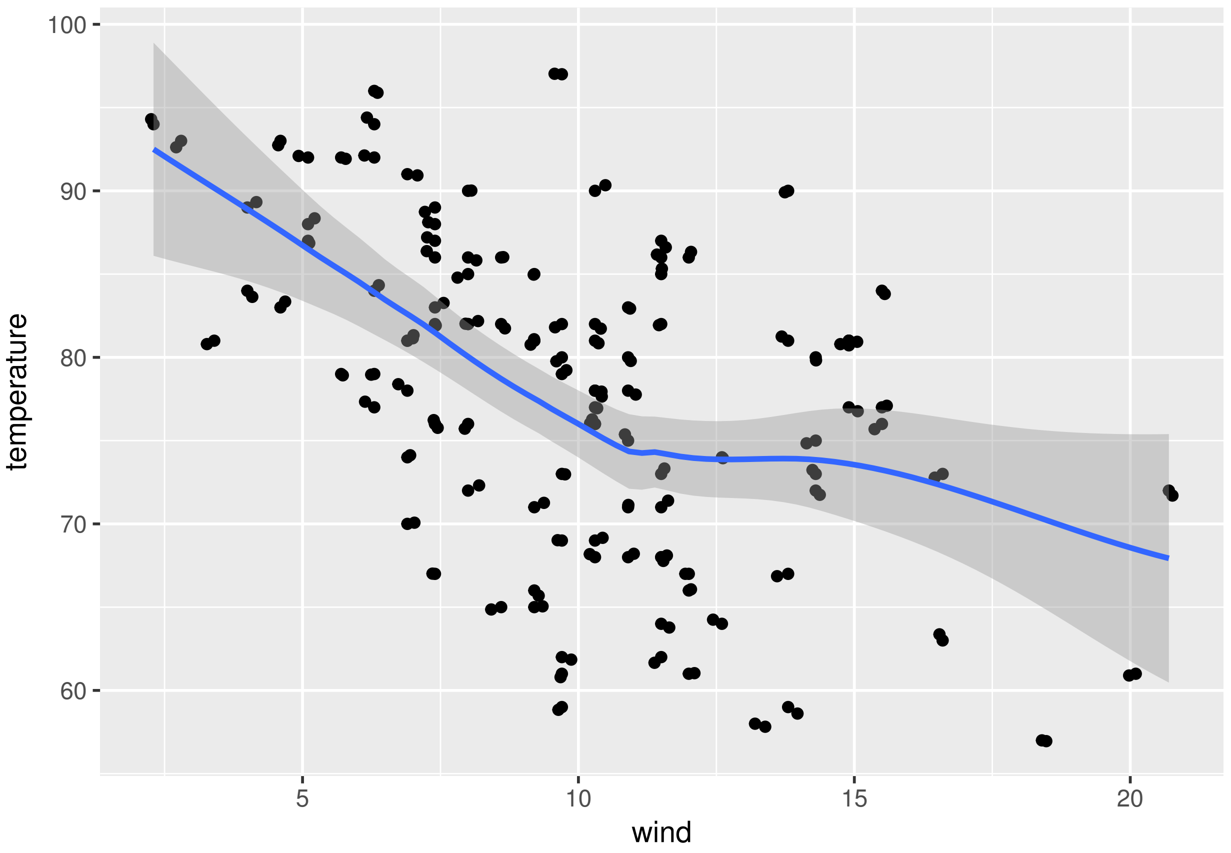
There's a little bit of non-normality in the tails of both distributions, especially temperature. Too bad.

We can model the trend both ways: wind as a function of temperature, and temperature as a wind a function of wind. We could plot both fits on one graph but really that's too much effort, so we'll just do the two separately.

```
ggplot(environmental, aes(x = temperature, y = wind)) + geom_point() + geom_jitter() +
    geom_smooth(method.args = list(degree = 1))
```



```
ggplot(environmental, aes(x = wind, y = temperature)) + geom_point() + geom_jitter() +
    geom_smooth(method.args = list(degree = 1))
```

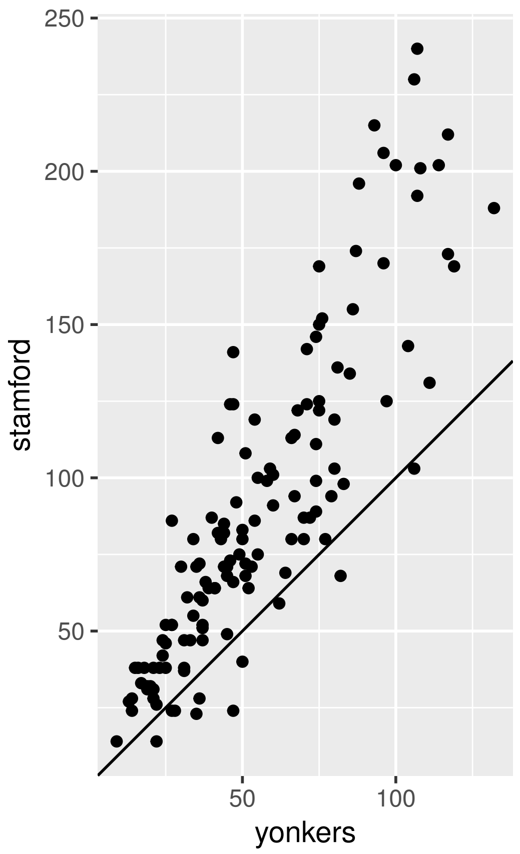


The average wind decreases just about linearly in temperature, while the average temperature has a curved decreasing relationship with wind. This isn't a paradox: when we leave the comfortable world of bivariate normal data, relationships may not be nice and symmetric when you interchange the variables.

### 3.4.2 Equal scales: Ozone

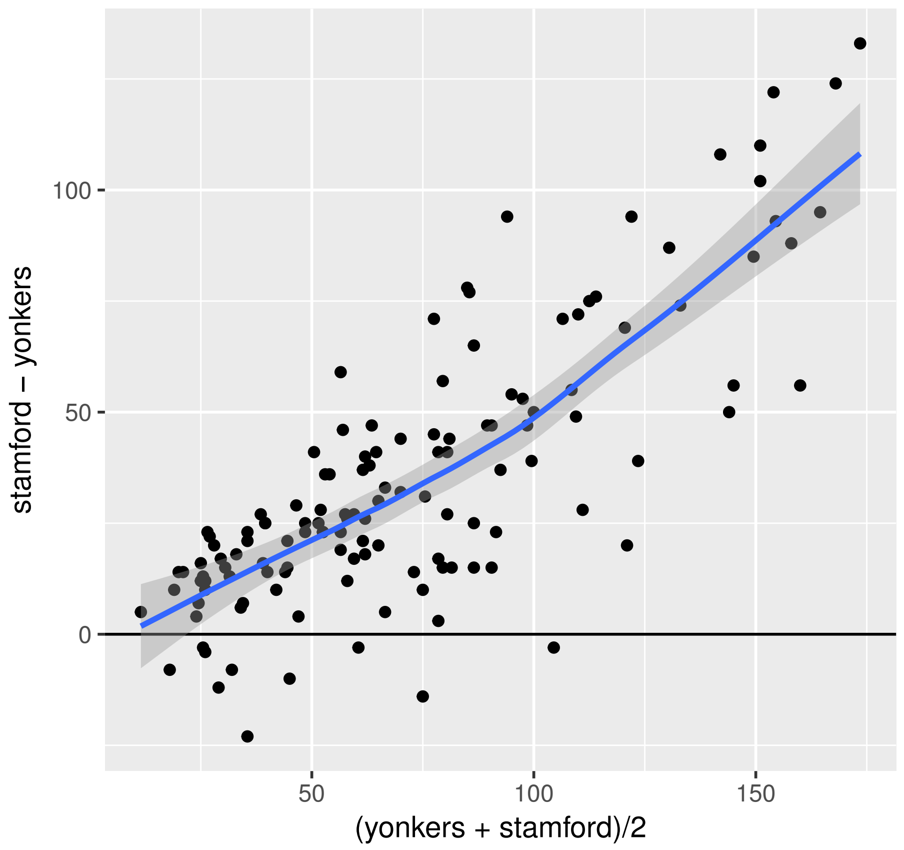
The `ozone` data set contains paired measurements of ozone concentration (in parts per billion) at Yonkers, NY and Stamford, CT. Since we've measuring the same variable at the two locations, it makes sense to use the same scales for both axes. The `coord_fixed()` function takes care of this.

```r
ggplot(ozone, aes(x = yonkers, y = stamford)) + geom_point() + geom_abline() +
    coord_fixed()
```



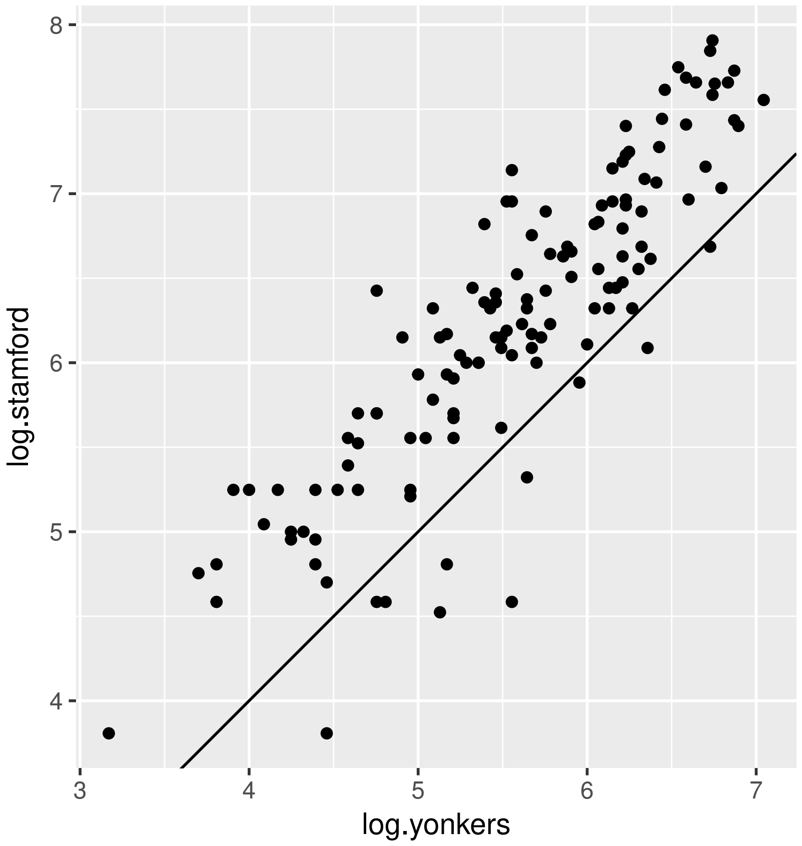
The points are almost all above the line $y = x$, so ozone is typically higher at Stamford. The data looks like it could be well-modeled by a straight line. To investigate further, we draw a Tukey mean-difference plot.

```
ggplot(ozone, aes(x = (yonkers + stamford)/2, y = stamford - yonkers)) + geom_point() +
    geom_abline(slope = 0) + geom_smooth(method.args = list(degree = 1)) + coord_fixed()
```



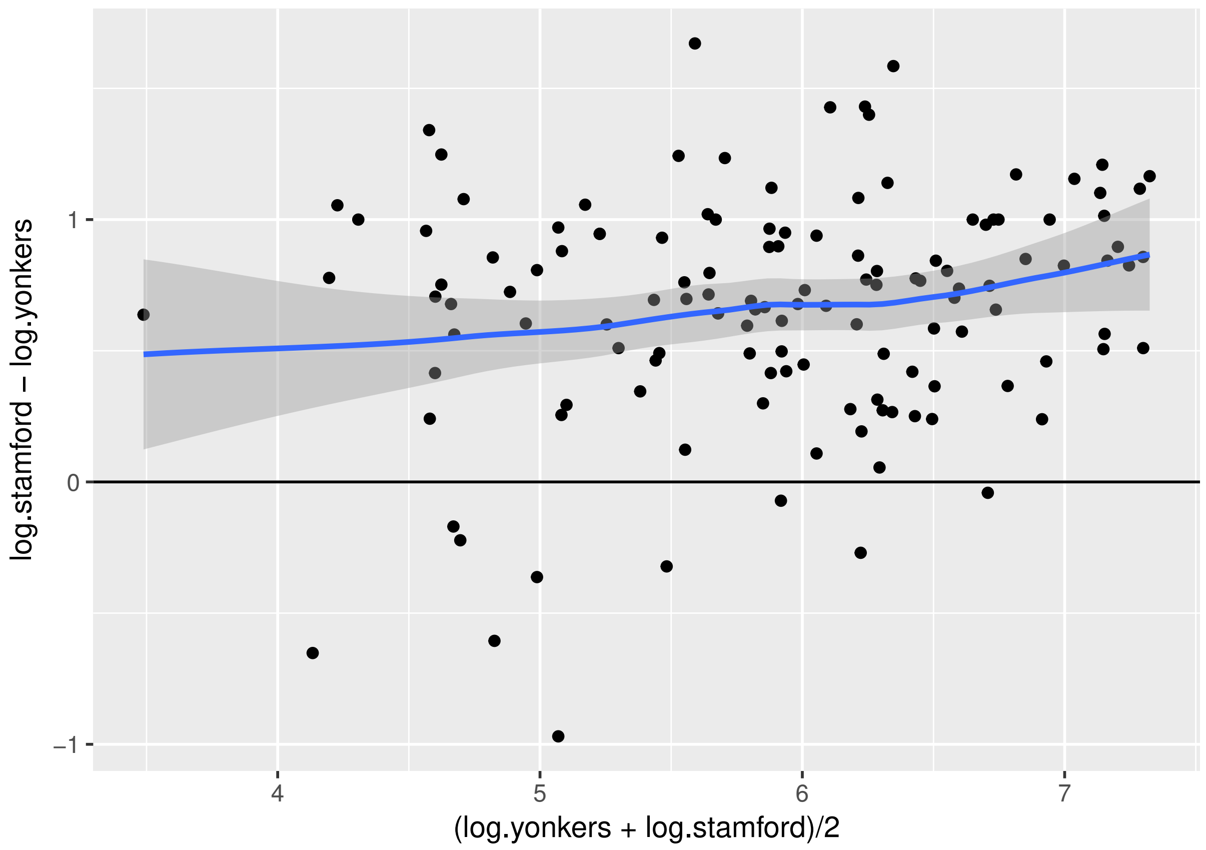
The loess fit is roughly consistent with an upward-sloping straight line. Since the line isn't horizontal, an additive shift isn't appropriate; perhaps a multiplicative shift would be better. Redraw the scatterplot with both variables logged:

```
log.ozone = data.frame(log.yonkers = log2(ozone$yonkers), log.stamford = log2(ozone$stamford))
ggplot(log.ozone, aes(x = log.yonkers, y = log.stamford)) + geom_point() + geom_abline() +
    coord_fixed()
```



It looks like a line parallel to $y = x$ might be a good fit. To get a better idea, draw the mean-difference plot.

```
ggplot(log.ozone, aes(x = (log.yonkers + log.stamford)/2, y = log.stamford -
    log.yonkers)) + geom_point() + geom_abline(slope = 0) + geom_smooth(method.args = list(degree = 1))
    coord_fixed()
```



We see that a multiplicative shift is nearly right, but we can do a bit better. There's a little bit of an upward slope in the fitted curve, going from about 0.5 on the left to 0.9 on the right. Back-transformed, that means on low-ozone days in Yonkers, the Stamford ozone concentration is typically about 40% higher; while on high-ozone days in Yonkers, the Stamford ozone concentration is typically about 90% higher.