Chapter 4

Trivariate data

4.1 Burning rubber

READ: Cleveland pp. 180–187, 200–213.

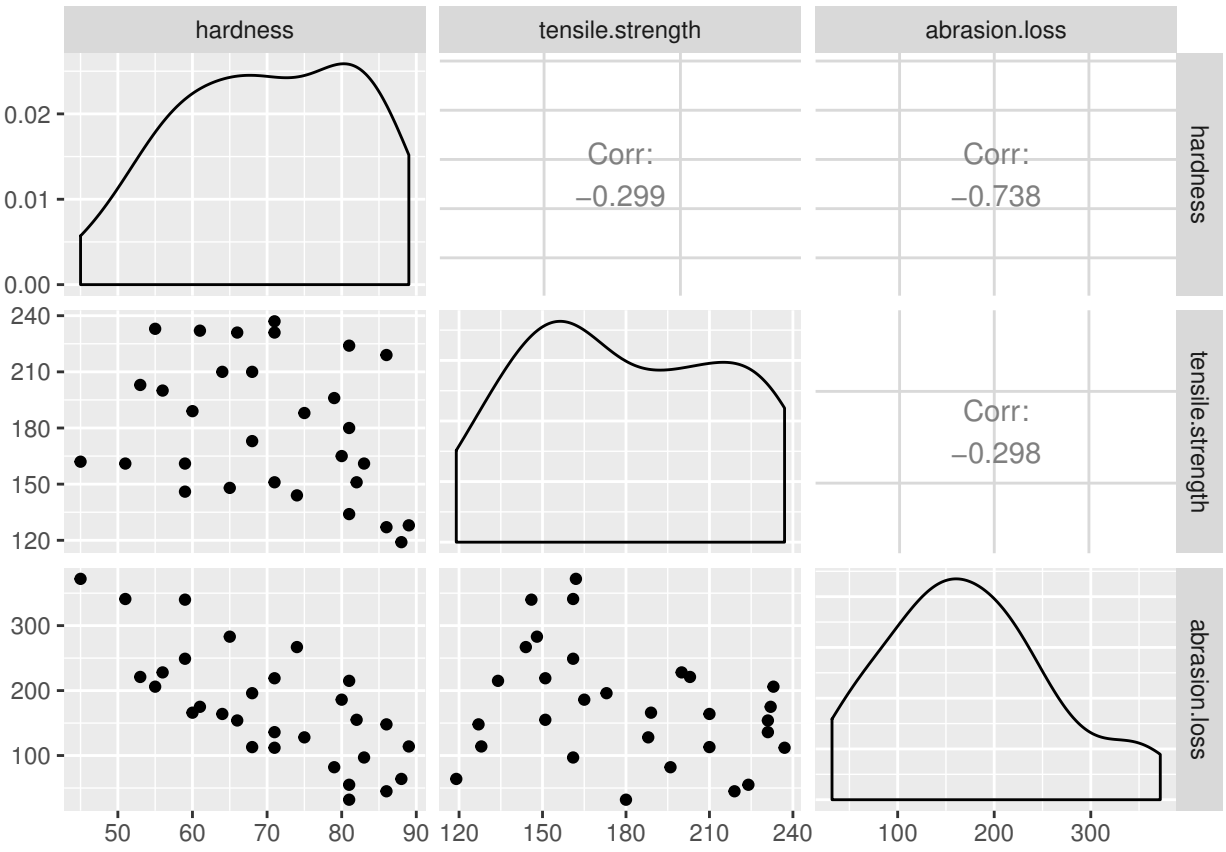
4.1.1 The rubber data

The data frame `rubber` in `lattice.RData` contains three measurements on 30 specimens of tire rubber:

- `hardness`: how much the rubber rebounds after being indented (in Shore degrees.)
- `tensile.strength`: the force per cross-sectional area required to break the rubber (in kg/cm^2.)
- `abrasion.loss`: the amount of material lost to abrasion when rubbing it per unit energy (in grams per hp-hour.) This gives you an idea how fast the tire will wear away when you drive. If we had to choose a "response" variable, it would be this one.

We want to first look at the pairwise relationships between all three variables, so we draw C_2^3 scatterplots. We can do this elegantly through `ggpairs()` in the `GGally` package:

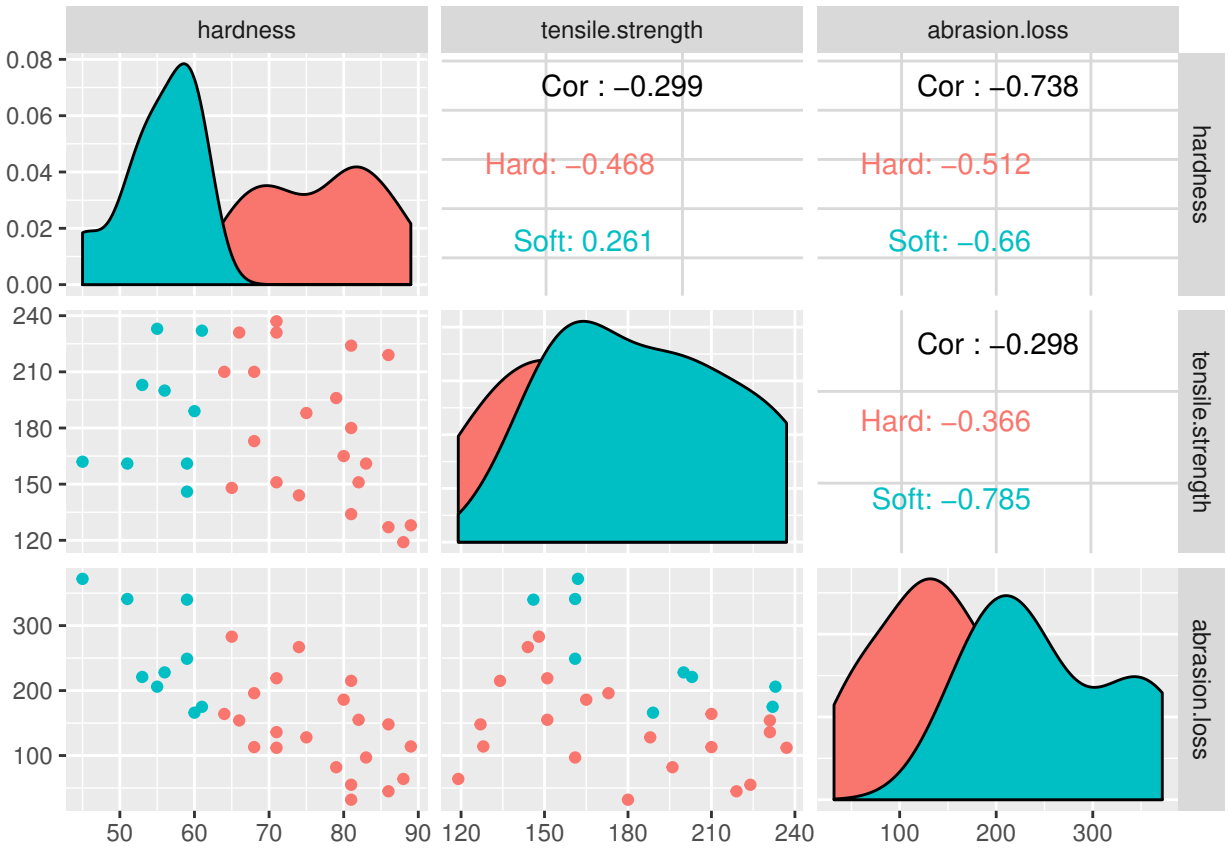
```
load("lattice.RData")
library(ggplot2)
# install.packages('GGally')
library(GGally)
ggpairs(rubber, columns = c("hardness", "tensile.strength", "abrasion.loss"))
```



The diagonal gives density plots. There's no horrible nonlinearity in the data, so the correlations should be good summaries of the strengths of the relationships.

To get at the *trivariate* relationship and not just the bivariate ones, we can use color to indicate the value of one of the three variables. Suppose we count a hardness of over 62 as "hard" and a hardness of 62 or less as "soft". We can create a factor variable that takes these two values, then color hard and soft specimens with different colors.

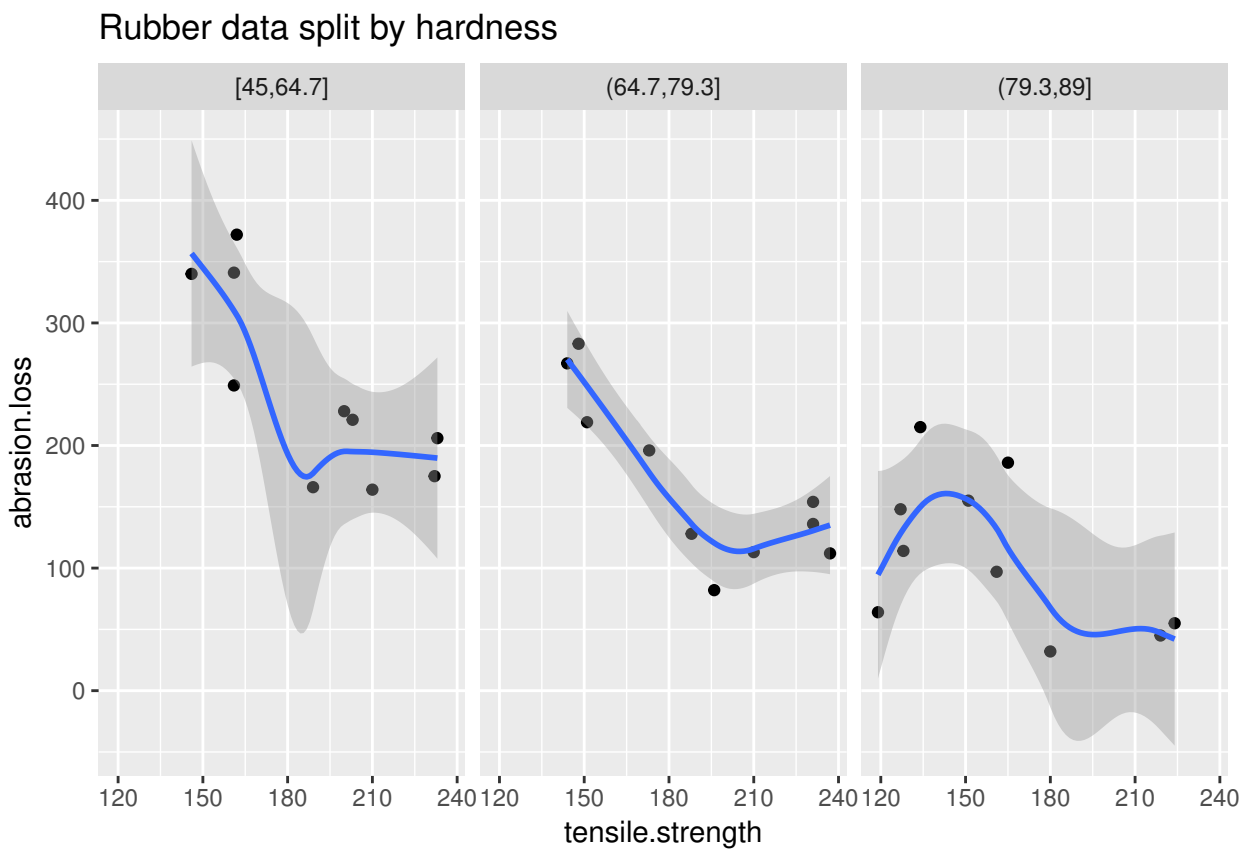
```
hard = rep(NA, nrow(rubber))
hard[rubber$hardness > 62] = "Hard"
hard[rubber$hardness <= 62] = "Soft"
rubber2 = data.frame(rubber, hard)
ggpairs(rubber2, columns = 1:3, aes(colour = hard))
```



The truly trivariate plot is the one in the middle of the bottom row. For a given tensile strength, a soft specimen tends to have more abrasion loss than a hard specimen, which seems to make physical sense.

In the above plot, we colored the points conditional on the binary hardness variable we created. Of course, we could create a hardness variable with three or more levels if we wished. After creating such a categorical variable, we can either color-code or we can draw separate plots for each value of the categorical variable. The latter approach is called a **conditional plot** or **coplot**. We'll cut the hardness variable into three categories, plot three scatterpolots of abrasion loss against tensile strength, and add loess smoothers.

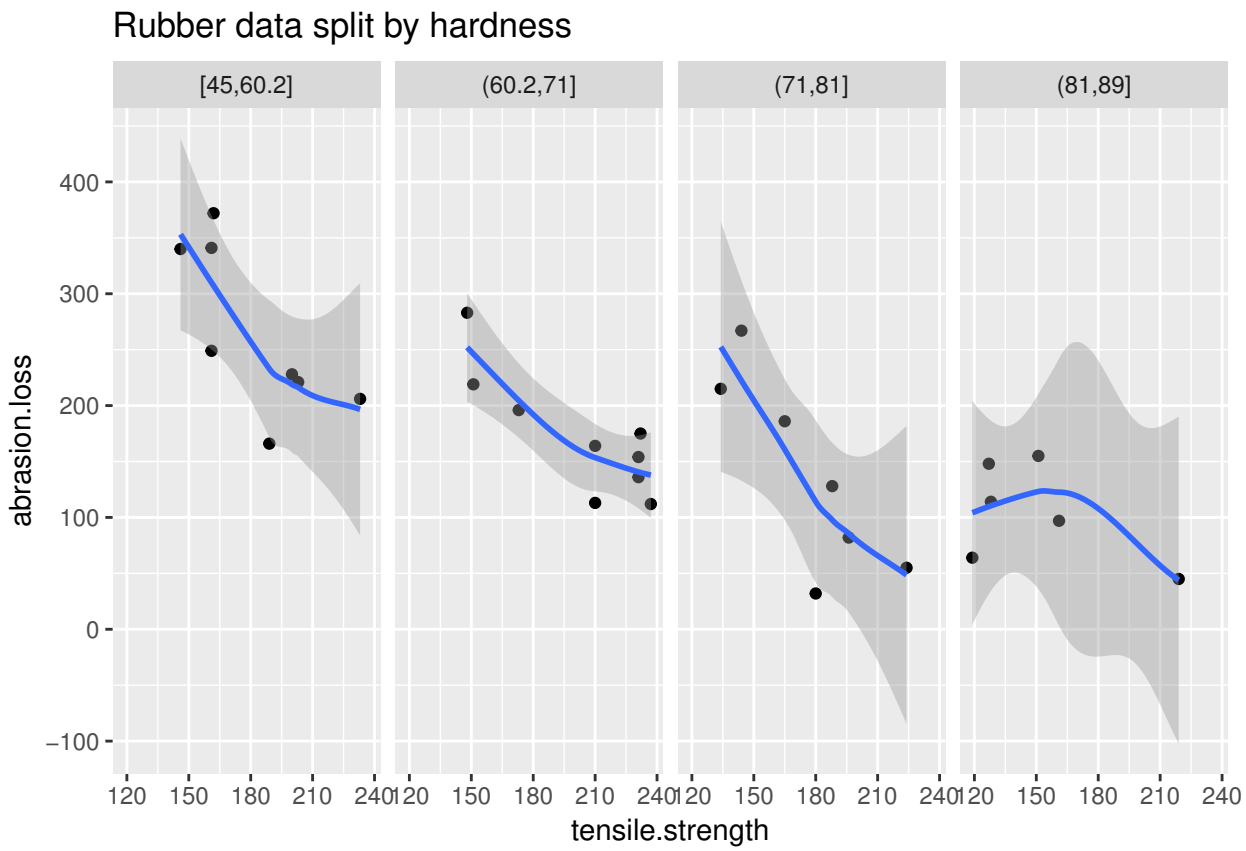
```
ggplot(rubber, aes(x = tensile.strength, y = abrasion.loss)) + geom_point() +
    geom_smooth(method.args = list(degree = 1)) + facet_grid(~cut_number(hardness,
    n = 3)) + labs(title = "Rubber data split by hardness")
```



There's a similar decreasing relationship in each panel, while the level of the curve falls as hardness increases. In the first two plots, the curve seems to flatten out beyond a tensile strength of 180 (in the third it's hard to tell what happens because of lack of data.)

It can be worth trying out multiple cuts and seeing if you get similar results. Try a four categories hardness cut (with a high `span` because of the low number of observations after cutting):

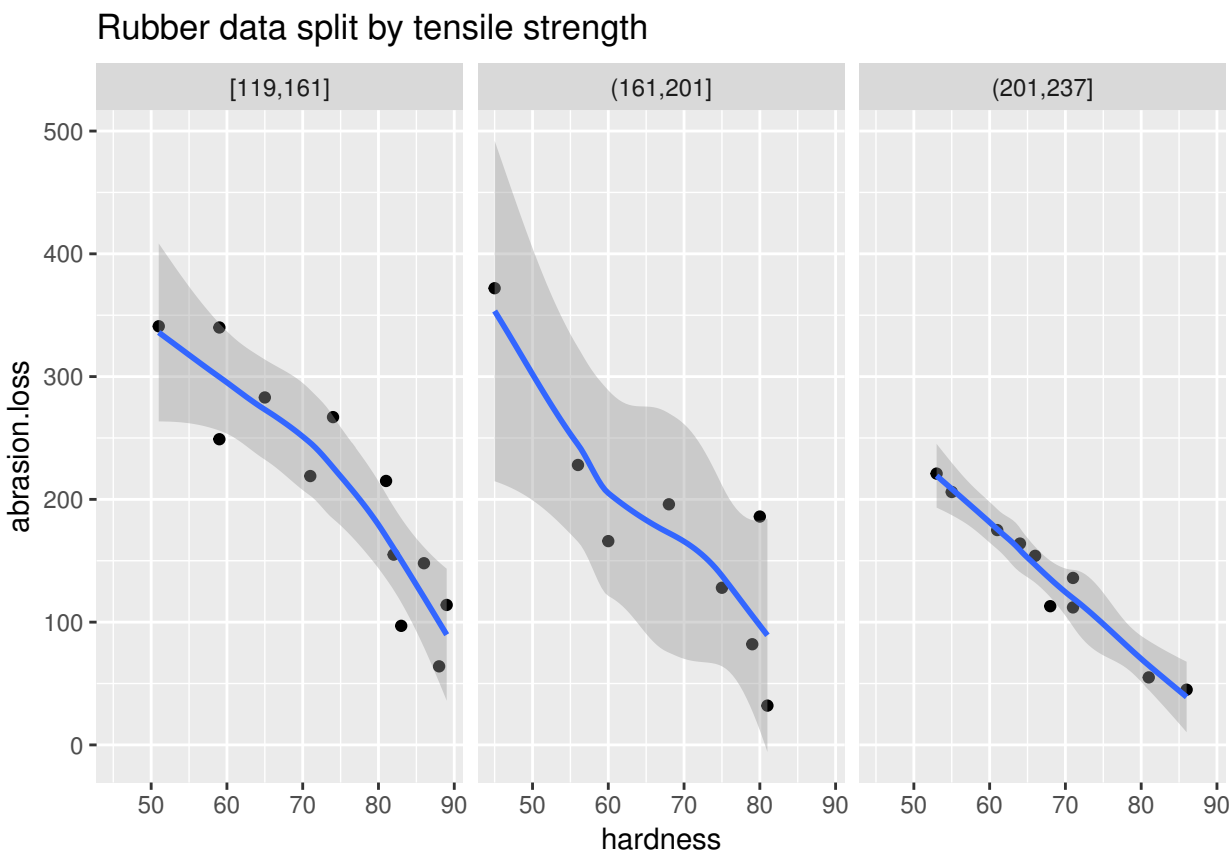
```
ggplot(rubber, aes(x = tensile.strength, y = abrasion.loss)) + geom_point() +
    geom_smooth(span = 1, method.args = list(degree = 1)) + facet_grid(~cut_number(hardness,
    n = 4)) + labs(title = "Rubber data split by hardness")
```



Again we see the relationships tend to flatten out somewhere past a tensile strength of 180.

We can also cut across tensile strength, then draw scatterplots of abrasion loss against hardness for each cut. Try three cuts for tensile strength:

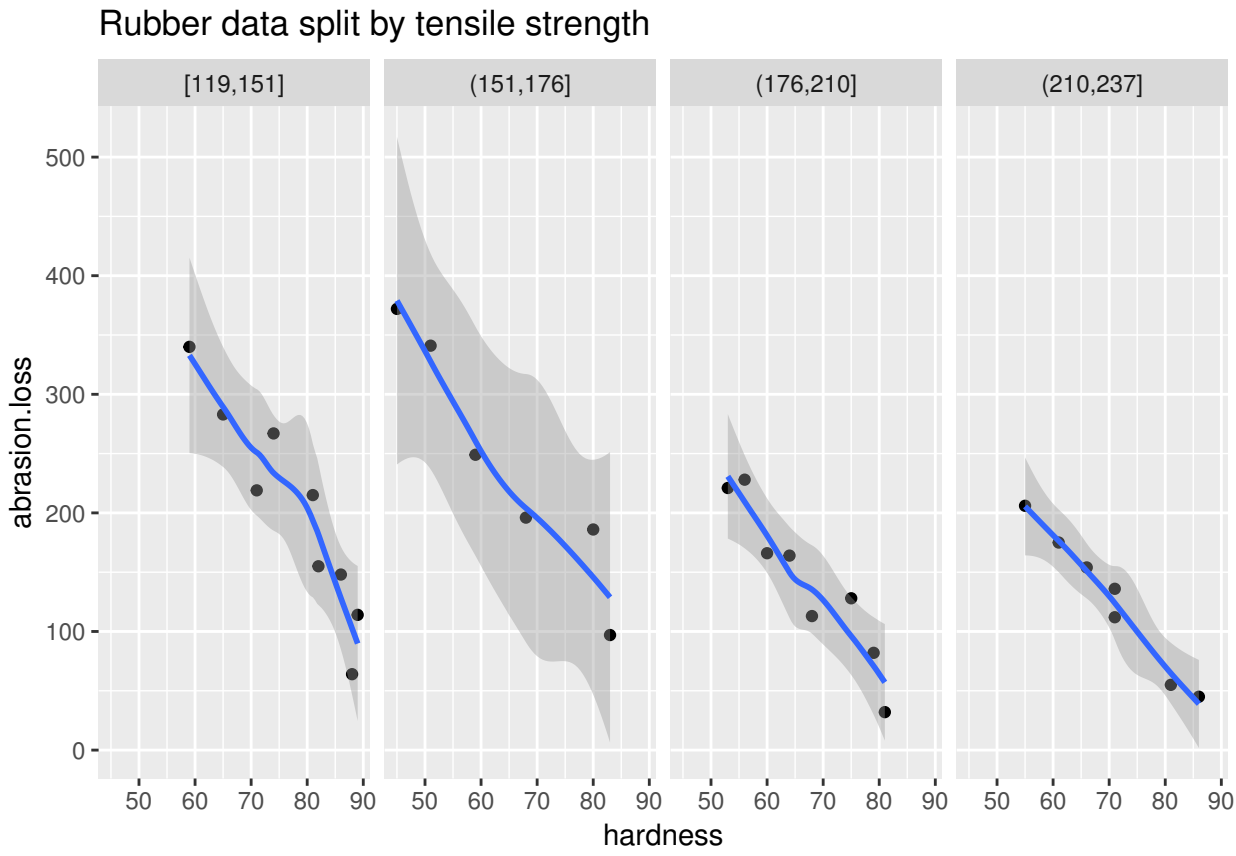
```
ggplot(rubber, aes(x = hardness, y = abrasion.loss)) + geom_point() + geom_smooth(method.args = list(deg
    facet_grid(~cut_number(tensile.strength, n = 3)) + labs(title = "Rubber data split by tensile streng
```



We get an approximately linear relationship in all three plots. The slope is negative and fairly similar in all three.

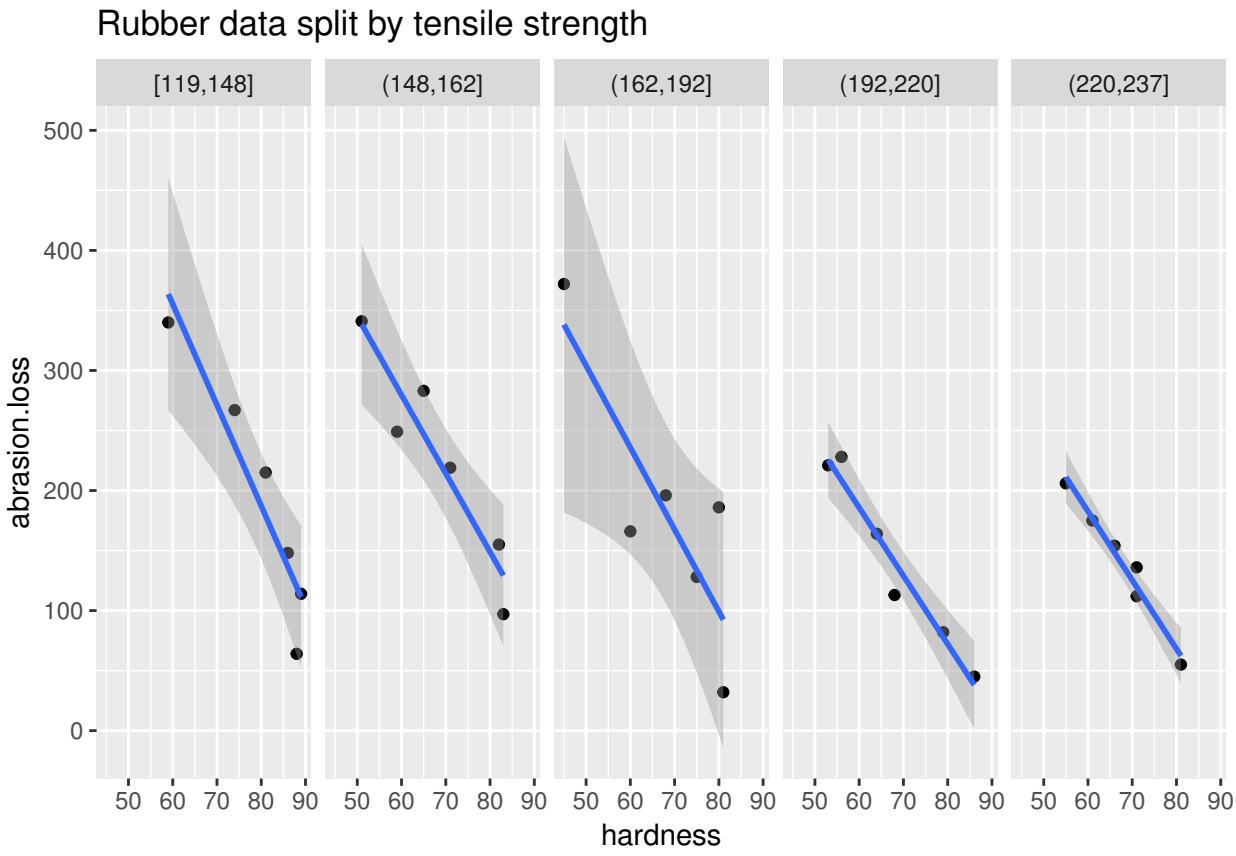
Try four cuts for tensile strength:

```
ggplot(rubber, aes(x = hardness, y = abrasion.loss)) + geom_point() + geom_smooth(method.args = list(de
    facet_grid(~cut_number(tensile.strength, n = 4)) + labs(title = "Rubber data split by tensile streng
```



The differences in level are a bit more apparent now – the two plots for low tensile strength have their lines much higher than for the two plots with low tensile strength. The weird thing is that the relationship isn't strictly decreasing as we go from left to right. We want to investigate further by doing more cuts, but then we start to run out of data. We can alleviate this by just fitting linear models instead of loess curves.

```
ggplot(rubber, aes(x = hardness, y = abrasion.loss)) + geom_point() + geom_smooth(method = "lm") +
    facet_grid(~cut_number(tensile.strength, n = 5)) + labs(title = "Rubber data split by tensile streng
```



There really does seem to be a critical value for tensile strength at which the relationship changes. This is consistent with what we saw in the coplots conditioning on hardness: there was a flattening beyond a tensile strength of about 180.

This exploration leads to some suggestions for fitting a model with abrasion loss as the response.

- The model might have a critical point for tensile strength, such that the relatioship between hardness and abrasion loss is decreasing below that point and flat above that point. We can either try to carefully locate that point or just eyeball it and say it's around 180.
- The nonmonotonicity in tensile strength could be due to outliers. (*Monotonic* means the trend either always goes up or always goes down. *Nonmonotonic* means the trend is upward in some places and downward in others.) So we should strongly consider using a robust/resistant fitting method.
- There's no obvious need for interaction terms. (If the coplots had lines with very different slopes, we'd have to strongly consider them.)
- The model fitting could get pretty complex, so we should probably stick to linear models (and variations thereof) where possible.

4.1.2 Visualizing a fitted model

Let's try fitting a piecewise function in tensile strength, such that the relationship with abrasion loss is linear below 180 and constant above 180. We still want to fit using `lm()` or `rlm()`, so we rewrite as a linear model by transforming tensile strength to "tensile strength below 180." You could do this by applying a function:

```
ts.low = function(x) {
    return((x - 180) * (x < 180))
}
```

(In fact, we didn't need to do this, because we have a variable called `ts.low` in our `rubber` data frame

already.)

Because there appear to be some outlying points in our data set, we prefer to fit using `rlm()` rather than `lm()`.

```
library(MASS)
rubber.rlm = rlm(abrasion.loss ~ hardness + ts.low(tensile.strength), data = rubber,
    psi = psi.bisquare)
```

To visualize the fitted surface, we want to plot a set of predictions for a grid of different values of hardness and tensile strength.

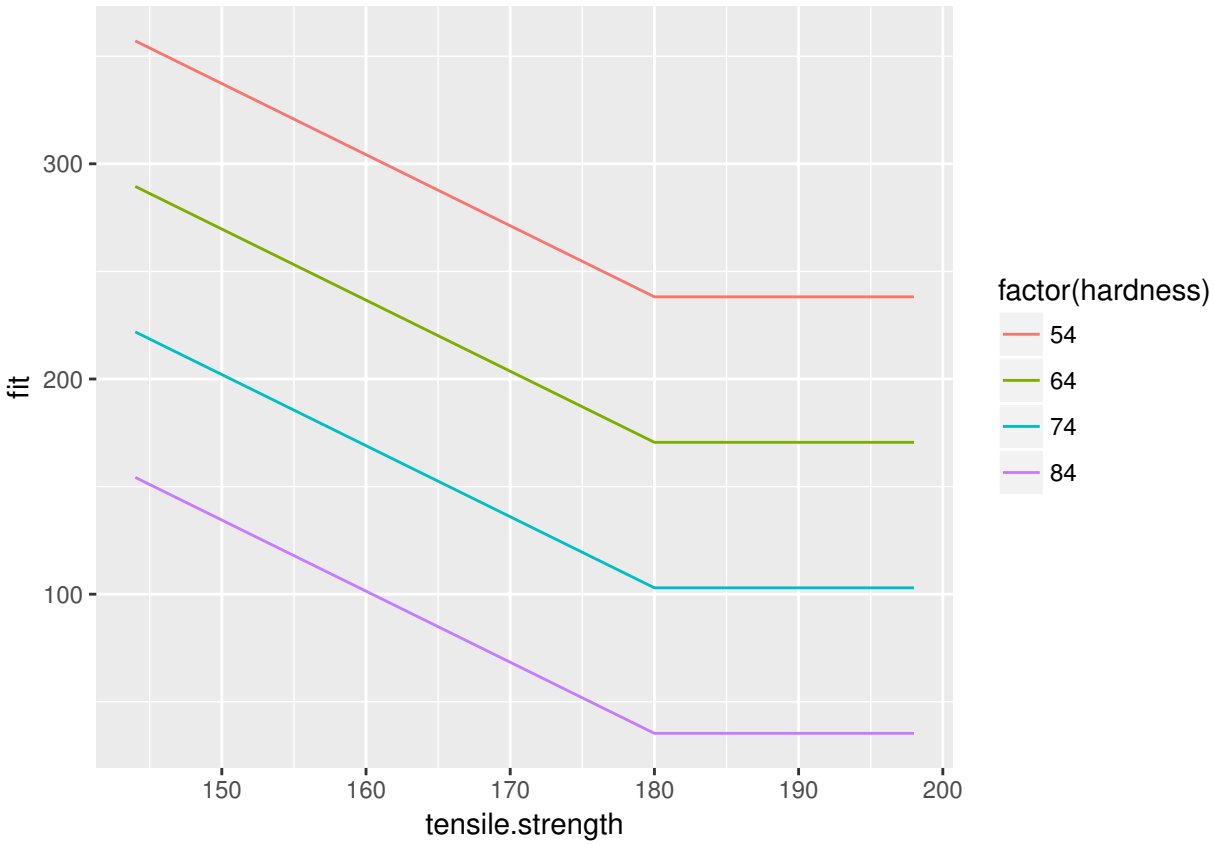
```
rubber.grid = expand.grid(hardness = c(54, 64, 74, 84), tensile.strength = c(144,
    162, 180, 198))
rubber.predict = predict(rubber.rlm, newdata = rubber.grid)
```

Suppose we want to see how the fit depends on tensile strength, conditioning on different values of hardness. One way to do this is to do coplots:

```
ggplot(data.frame(rubber.grid, fit = as.vector(rubber.predict)), aes(x = tensile.strength,
    y = fit)) + geom_line() + facet_grid(~hardness) + labs(title = "Abrasion loss fit conditional on ha
```

Alternatively, we can plot all the lines on the same graph, and distinguish between them by color:

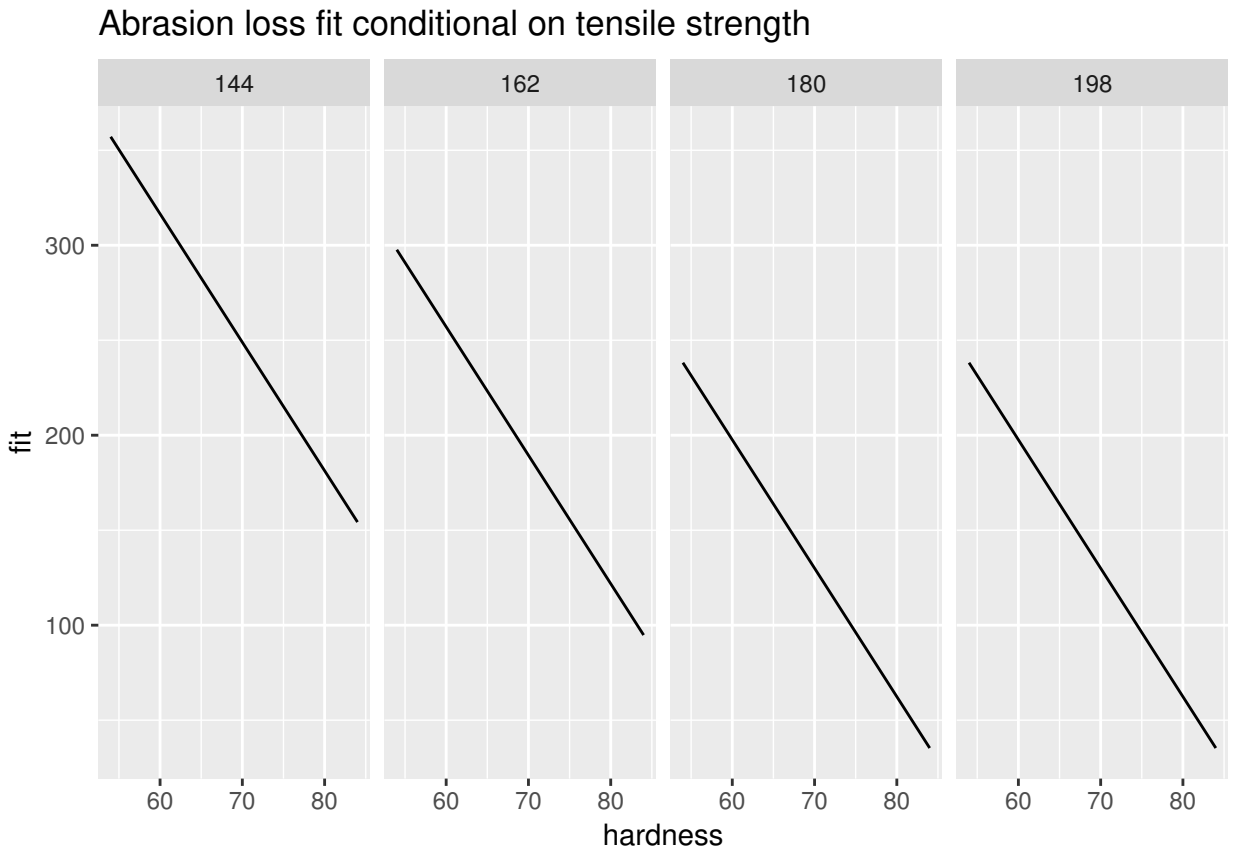
```
ggplot(data.frame(rubber.grid, fit = as.vector(rubber.predict)), aes(x = tensile.strength,
    y = fit, group = hardness, color = factor(hardness))) + geom_line()
```



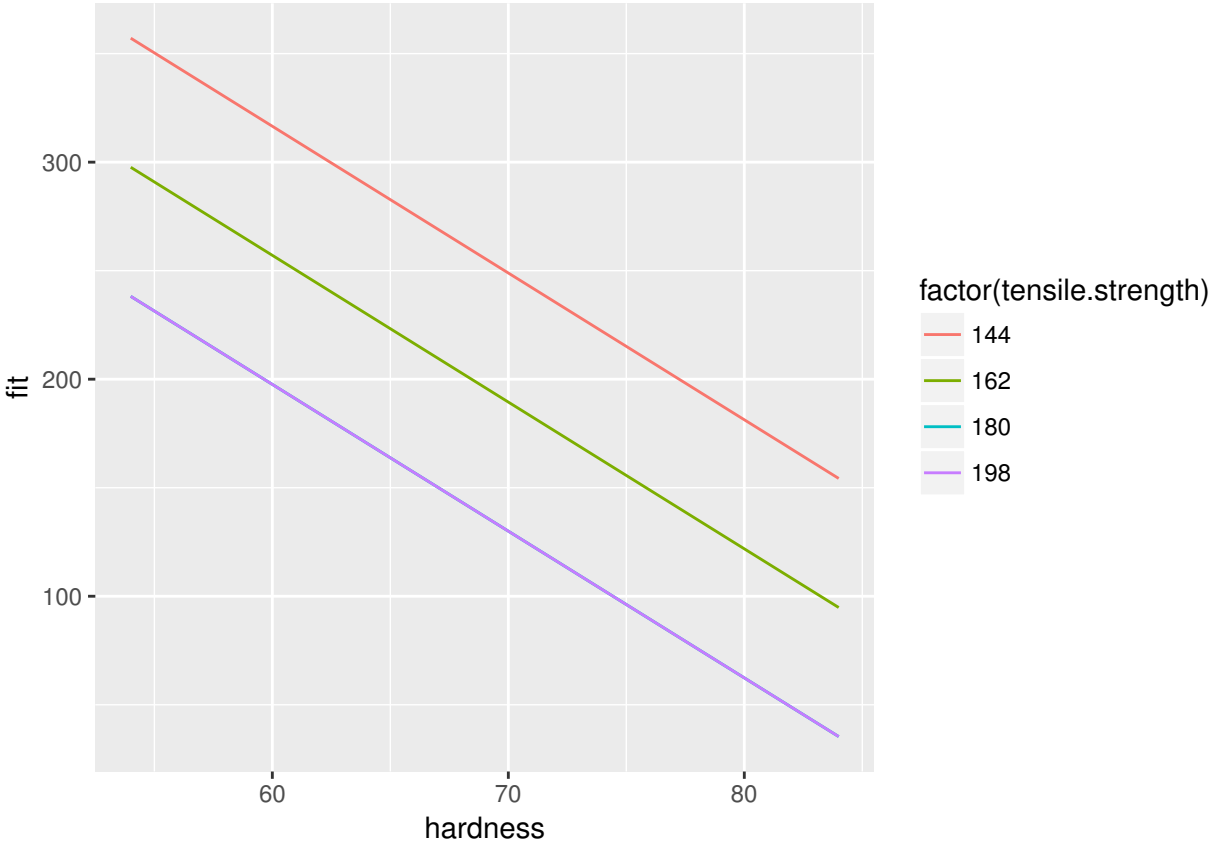
In both plots, we see that the fit becomes horizontal at 180, as we specified. Since there was no interaction in our model, the lines below 180 have the same slope.

We also want to plot the fit as a function of hardness, conditioning on different values of tensile strength. Again, we can use coplots or color.

```
ggplot(data.frame(rubber.grid, fit = as.vector(rubber.predict)), aes(x = hardness,
    y = fit)) + geom_line() + facet_grid(~tensile.strength) + labs(title = "Abrasion loss fit conditiona
```




```
ggplot(data.frame(rubber.grid, fit = as.vector(rubber.predict)), aes(x = hardness,
    y = fit, group = tensile.strength, color = factor(tensile.strength))) +
    geom_line()
```



Note that on the second plot, the line for 198 is right on top of the line for 180 – beyond 180, it doesn't make any difference what tensile strength is (according to our model.) Other than the, we just have parallel lines, which is what you should get from a linear model with no interaction.

Of course, you could always just print out the numerical summary:

```
summary(rubber.rlm)
```

```
##
## Call: rlm(formula = abrasion.loss ~ hardness + ts.low(tensile.strength),
##     data = rubber, psi = psi.bisquare)
## Residuals:
##      Min       1Q   Median       3Q      Max
## -145.817  -22.252   -0.332   13.501   74.061
##
## Coefficients:
##                           Value    Std. Error t value
## (Intercept)              603.2798  33.5405    17.9866
## hardness                  -6.7612   0.4936   -13.6965
## ts.low(tensile.strength)  -3.3036   0.3107   -10.6342
##
## Residual standard error: 28.1 on 27 degrees of freedom
```

4.1.3 Exploring the residuals

Let's refit the same model as above, only this time for convenience, we'll use the `ts.low` variable in `rubber`.

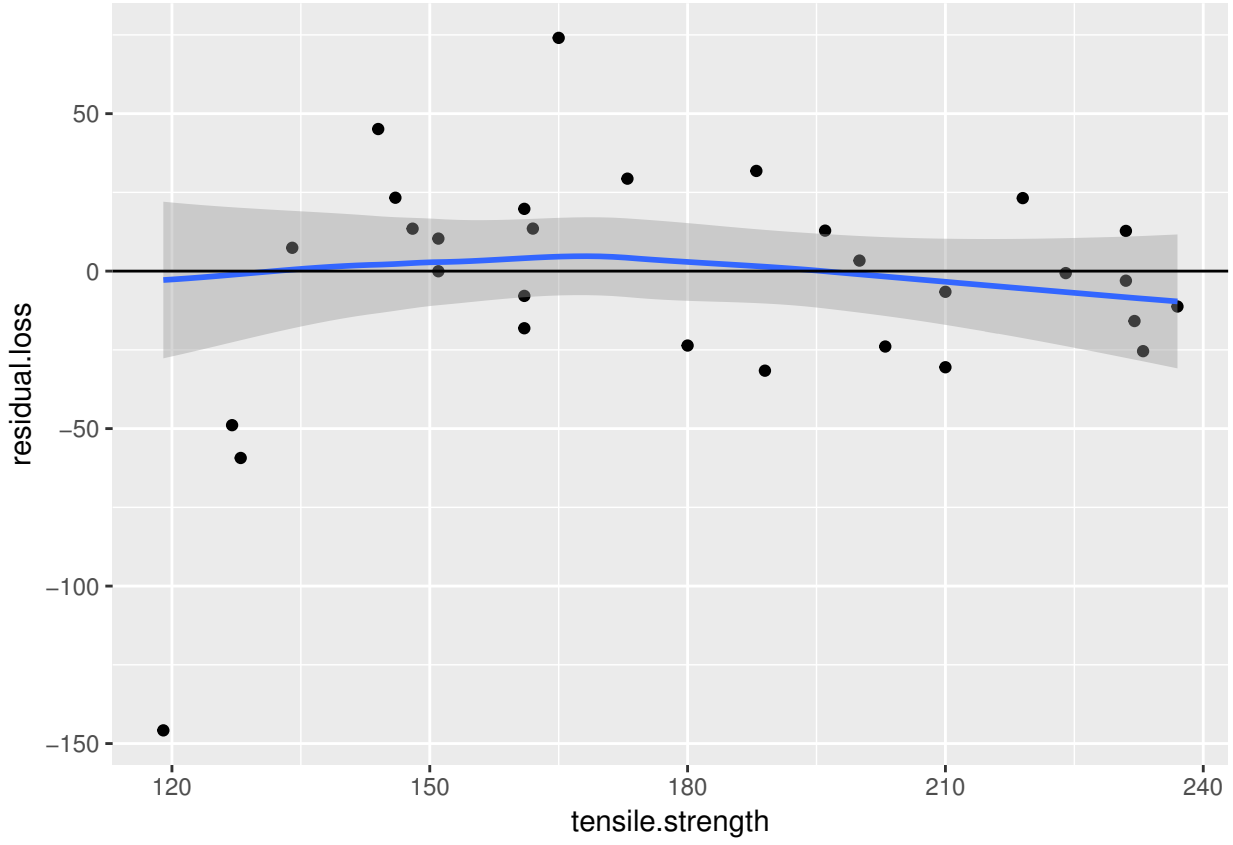
```
rubber.rlm2 = rlm(abrasion.loss ~ hardness + ts.low, data = rubber, psi = psi.bisquare)
```

We now want to collect everything we might want to use in residual plotting – the original variables, cut versions of the variables, as well as the residuals themselves – in one data frame.

```
tensile.strength = rubber$tensile.strength
tensile.cat = cut_number(tensile.strength, n = 3)
hardness = rubber$hardness
hard.cat = cut_number(hardness, n = 3)
residual.loss = residuals(rubber.rlm2)
rubber.rlm2.df = data.frame(tensile.strength, tensile.cat, hardness, hard.cat,
    residual.loss)
```

As always, we examine the residuals to look for weaknesses in our fit. First, plot the residuals as a function of tensile strength, then add a locally linear loess fit (we'll use a robust one because of outliers.)

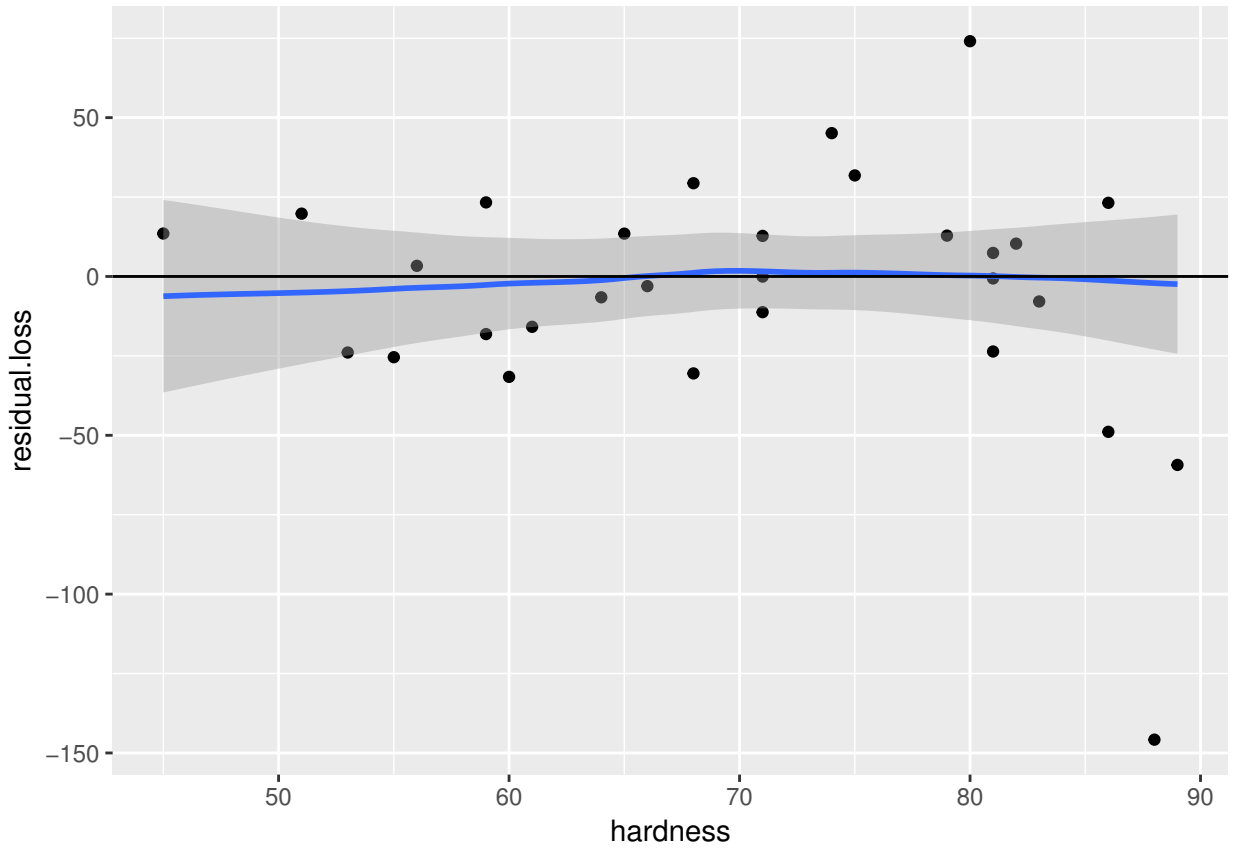
```
ggplot(rubber.rlm2.df, aes(x = tensile.strength, y = residual.loss)) + geom_point() +
    geom_smooth(span = 1, method.args = list(degree = 1, family = "symmetric")) +
    geom_abline(slope = 0)
```



Those outliers on the left are pretty big… Putting those aside, the rest of the residuals seem randomly scattered about the zero line.

We repeat this with hardness as the explanatory variable:

```
ggplot(rubber.rlm2.df, aes(x = hardness, y = residual.loss)) + geom_point() +
    geom_smooth(span = 1, method.args = list(degree = 1, family = "symmetric")) +
    geom_abline(slope = 0)
```



This time the three problematic points have shifted to the right. Once again, the rest of the observations look like random noise. So far, the fit seems to be a good guess despite the handful of outliers.

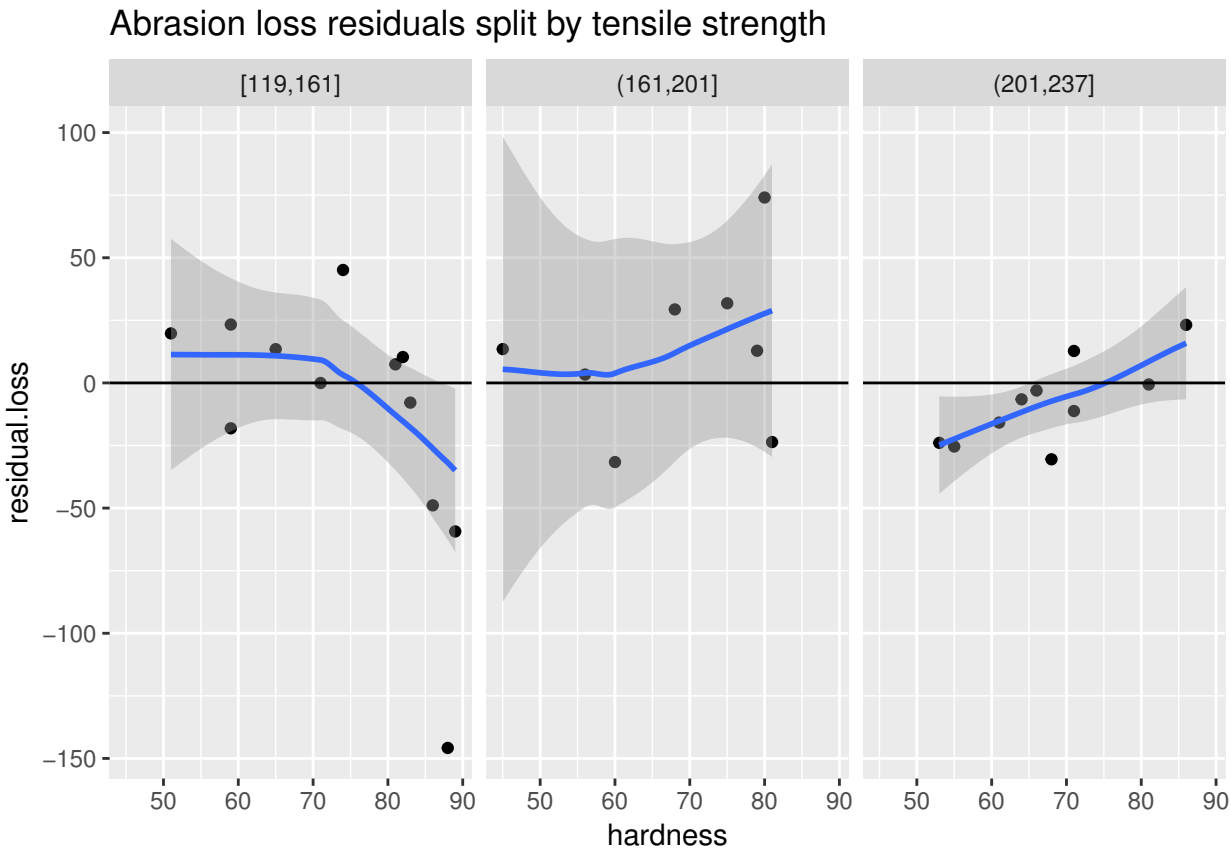
We also want to look at coplots of the residuals. We break up the data into three categories of hardness, then for each category, plot the residuals against tensile strength.

```
ggplot(rubber.rlm2.df, aes(x = tensile.strength, y = residual.loss)) + geom_point() +
    geom_smooth(span = 1, method.args = list(degree = 1, family = "symmetric")) +
    geom_abline(slope = 0) + facet_grid(~hard.cat) + labs(title = "Abrasion loss residuals split by hard
```

This looks less good. The residual coplots for soft and medium-hardness both looks like they slope downward somewhat. The residual coplot for the hardest specimens is bent out of shape by the three outliers on the left.

Now look at coplots that condition on tensile strength.

```
ggplot(rubber.rlm2.df, aes(x = hardness, y = residual.loss)) + geom_point() +
    geom_smooth(span = 1, method.args = list(degree = 1, family = "symmetric")) +
    geom_abline(slope = 0) + facet_grid(~tensile.cat) + labs(title = "Abrasion loss residuals split by
```



We get a similar problem in the opposite direction. The low-strength graph (left) shows the high-hardness outliers distorting the picture. In the other two panels, the trend is upward, and the slope doesn't seem to be the same in the two plots.

There are enough problems that we should try to improve our model.

4.1.4 Improving the fit

The residual plots identified two major problems:

- There are three outlying points with low tensile strength, high hardness, and low abrasion loss that don't seem to fit the same pattern as the rest of the data.
- The residual coplots had different (non-zero) slopes, suggesting that we need an interaction term after all.

The interaction is easy to deal with. The main subjective decision we need to make is whether or not to drop the three outliers. I would lean toward yes, because they really are a long way from the rest of the data. However, dropping them means we should *not* use our model to say anything about rubber specimens with low tensile strength and high hardness (e.g. beyond the observation with ts 151 and hardness 82.)

We fit the model `rubber.rlm3` with an interaction to the data set with the outliers dropped, then throw everything (including fitted values and fitted value with mean removed) into a data frame `rubber.rlm3.df`.

```
rubber.no.outliers = rubber[tensile.strength > 130, ]
rubber.rlm3 = rlm(abrasion.loss ~ hardness * ts.low, data = rubber.no.outliers,
    psi = psi.bisquare)
rubber.rlm3.df = data.frame(tensile.strength = rubber.no.outliers$tensile.strength,
    tensile.cat = cut_number(rubber.no.outliers$tensile.strength, n = 3), hardness = rubber.no.outliers$
    hard.cat = cut_number(rubber.no.outliers$hardness, n = 3), fitted.loss = fitted.values(rubber.rlm3)
```

```
    fit.demeaned = fitted.values(rubber.rlm3) - mean(fitted.values(rubber.rlm3)),
    residual.loss = residuals(rubber.rlm3))
```

Let's try drawing residual coplots again. First, condition on hardness:

```
ggplot(rubber.rlm3.df, aes(x = tensile.strength, y = residual.loss)) + geom_point() +
    geom_smooth(span = 1, method.args = list(degree = 1, family = "symmetric")) +
    geom_abline(slope = 0) + facet_grid(~hard.cat) + labs(title = "Abrasion loss residuals split by hard
```

There's no longer any systematic problem. The small slopes in the added loess smooths could easily be random noise.

Now condition on tensile strength.

```
ggplot(rubber.rlm3.df, aes(x = hardness, y = residual.loss)) + geom_point() +
    geom_smooth(span = 1, method.args = list(degree = 1, family = "symmetric")) +
    geom_abline(slope = 0) + facet_grid(~tensile.cat) + labs(title = "Abrasion loss residuals split by
```

The middle plot has an upward-sloping trend, but this just seems like luck based on the way tensile strength happened to be cut. (We could test out different cuts, but this seems like overkill.) It seems like our model fits the data well.

Now what does the model actually tell us? We could look at the numerical summary:

```
summary(rubber.rlm3)
```

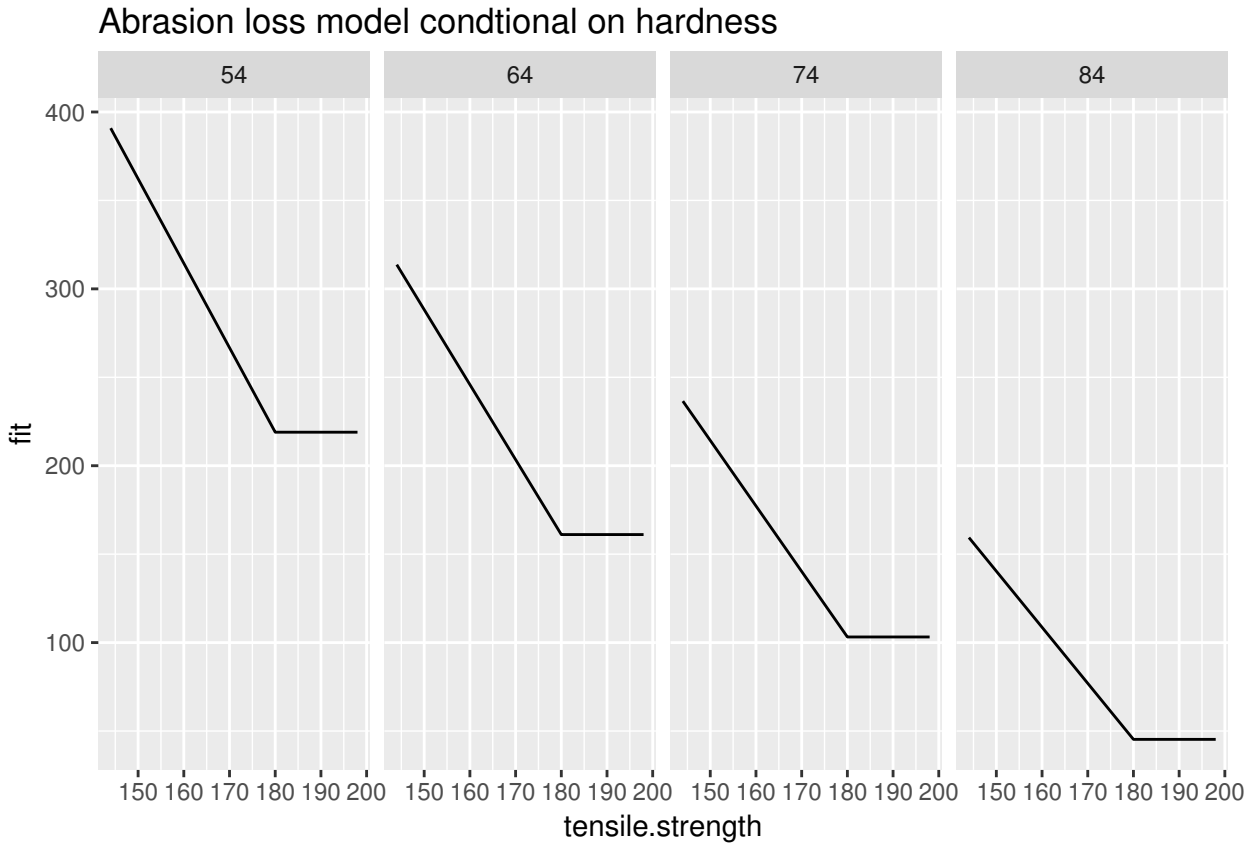
```
##
## Call: rlm(formula = abrasion.loss ~ hardness * ts.low, data = rubber.no.outliers,
##     psi = psi.bisquare)
## Residuals:
##      Min       1Q   Median       3Q      Max
## -30.6476  -8.0941  -0.8085  11.1000  66.8164
##
## Coefficients:
##                 Value    Std. Error t value
## (Intercept)     531.5812  35.2843    15.0656
## hardness         -5.7893   0.5081   -11.3942
## ts.low           -7.6652   2.0844    -3.6775
## hardness:ts.low   0.0535   0.0290     1.8457
##
## Residual standard error: 16.16 on 23 degrees of freedom
```

However, it's hard to understand what the interaction term means just from the numbers. Let's do plots then. Firstly, make predictions on a grid:

```
rubber.grid = data.frame(rubber.grid, ts.low = ts.low(rubber.grid$tensile.strength))
rubber.predict3 = predict(rubber.rlm3, newdata = rubber.grid)
```

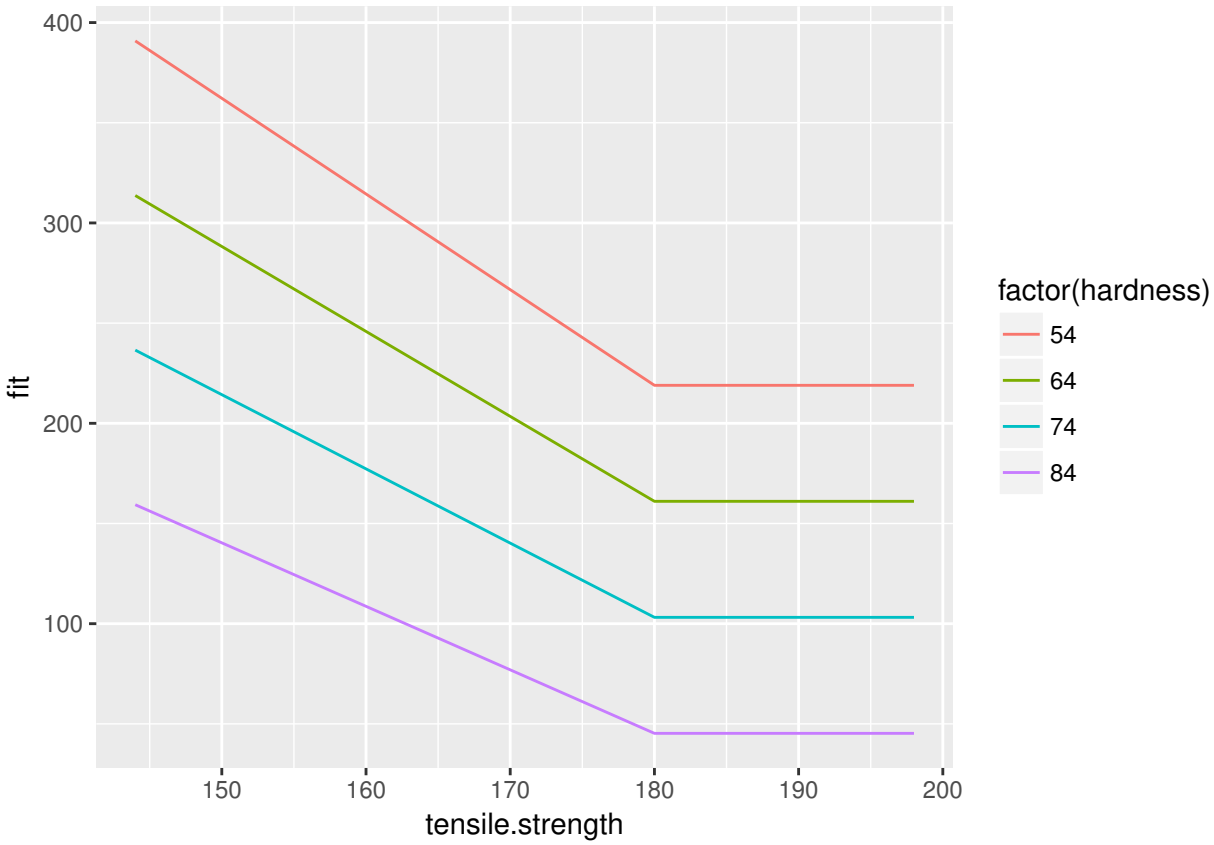
Next, coplots of the model, conditioning on hardness...

```
ggplot(data.frame(rubber.grid, fit = as.vector(rubber.predict3)), aes(x = tensile.strength,
    y = fit)) + geom_line() + facet_grid(~hardness) + labs(title = "Abrasion loss model condtional on ha
```



and/or plot the fits for different hardnesses in different colors.

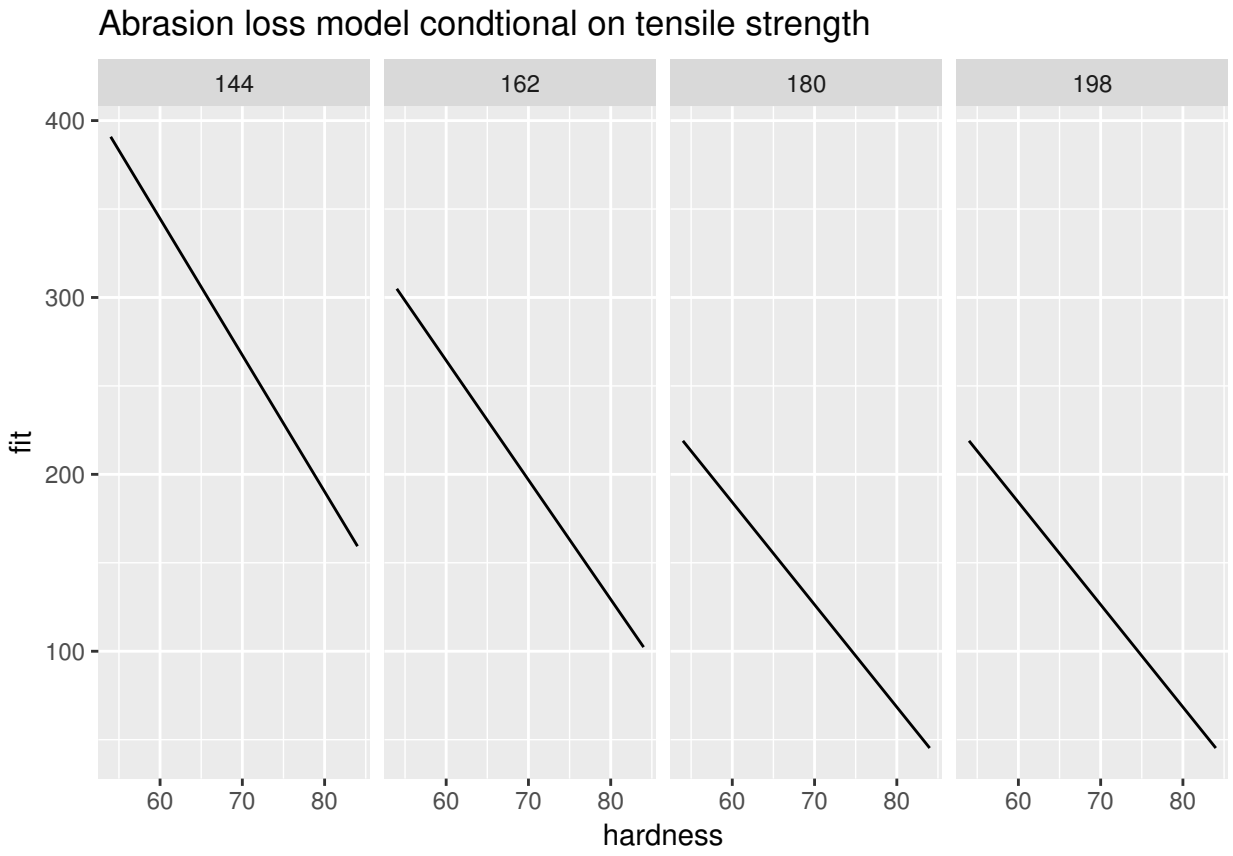
```
ggplot(data.frame(rubber.grid, fit = as.vector(rubber.predict3)), aes(x = tensile.strength,
    y = fit, group = hardness, color = factor(hardness))) + geom_line()
```



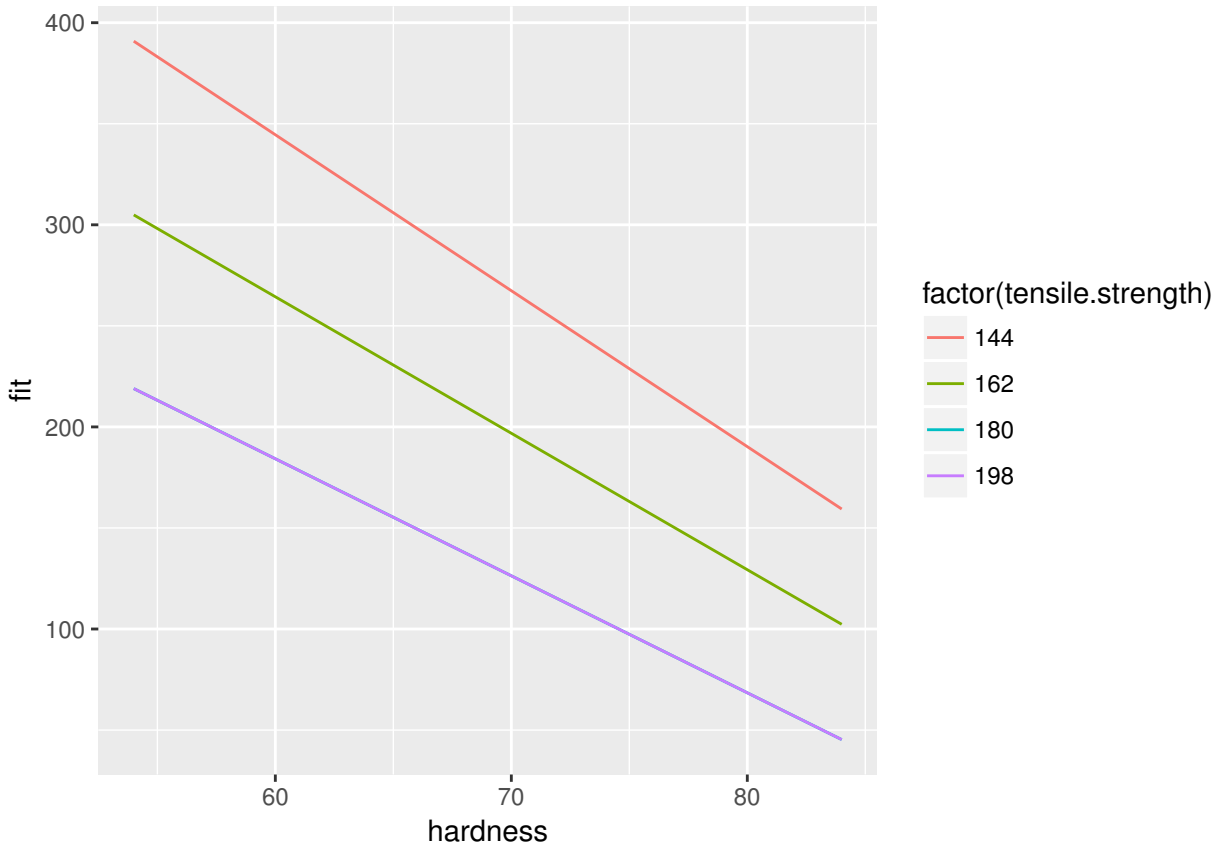
We're still fitting the same kind of function: linear below 180, constant above 180. However, the slopes are now slightly different for different values of hardness – the left-hand lines aren't parallel. The slope gets more shallow as hardness increases.

Now condition on tensile strength. Here are coplots and colored fits:

```
ggplot(data.frame(rubber.grid, fit = as.vector(rubber.predict3)), aes(x = hardness,
    y = fit)) + geom_line() + facet_grid(~tensile.strength) + labs(title = "Abrasion loss model condtio
```




```
ggplot(data.frame(rubber.grid, fit = as.vector(rubber.predict3)), aes(x = hardness,
    y = fit, group = tensile.strength, color = factor(tensile.strength))) +
    geom_line()
```

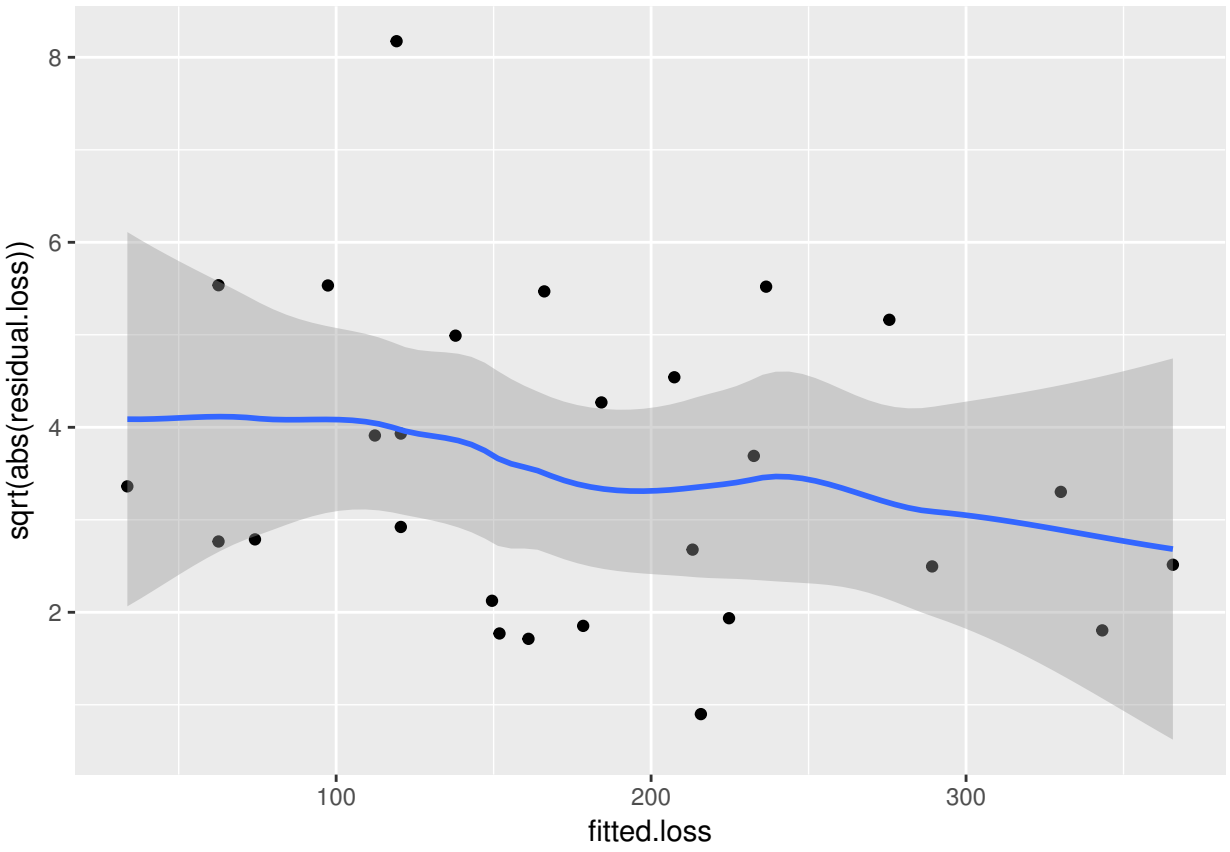


For tensile strength below 180, the slopes of the linear fits become slightly shallower as tensile strength increases. Once tensile strength increases below 180, then by model assumption it makes no difference to either the slope or level of the fit.

4.1.5 Checking the residuals

After all the fudging we did, it seems unlikely the residuals will turn nicely (homoskedastic and normal.) Nevertheless, we should check and hope. First, draw a residual-fit plot, with the y-axis giving the square root absolute residuals.

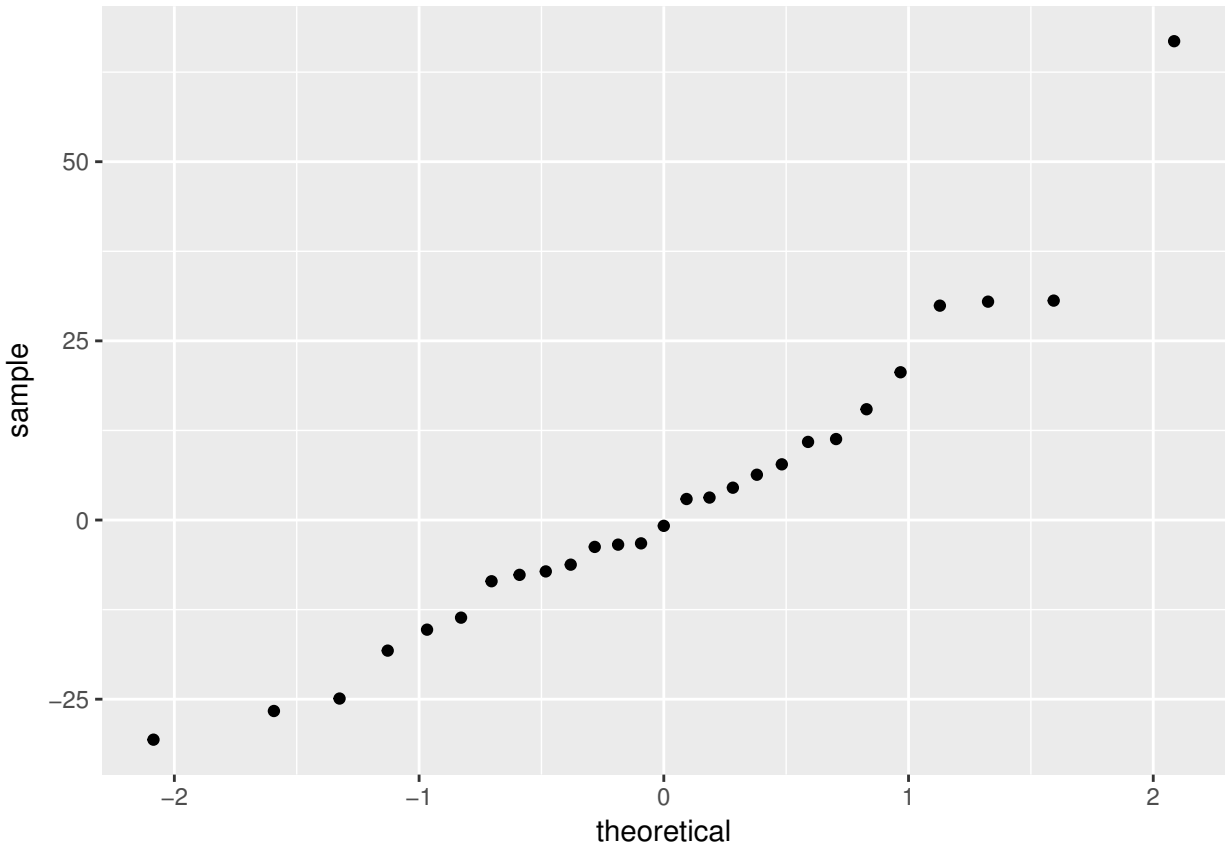
```
ggplot(rubber.rlm3.df, aes(x = fitted.loss, y = sqrt(abs(residual.loss)))) +
    geom_point() + geom_smooth(method.args = list(degree = 1))
```



Recall that if the errors are homoskedastic, the level of the residuals under this transformation should be approximately constant. We see here that the level dropping a little going from left to right, but this might just be random, and even if it wasn't it would be unlikely that going to the effort of fixing this would be worthwhile.

Next, draw a normal QQ plot of the residuals.

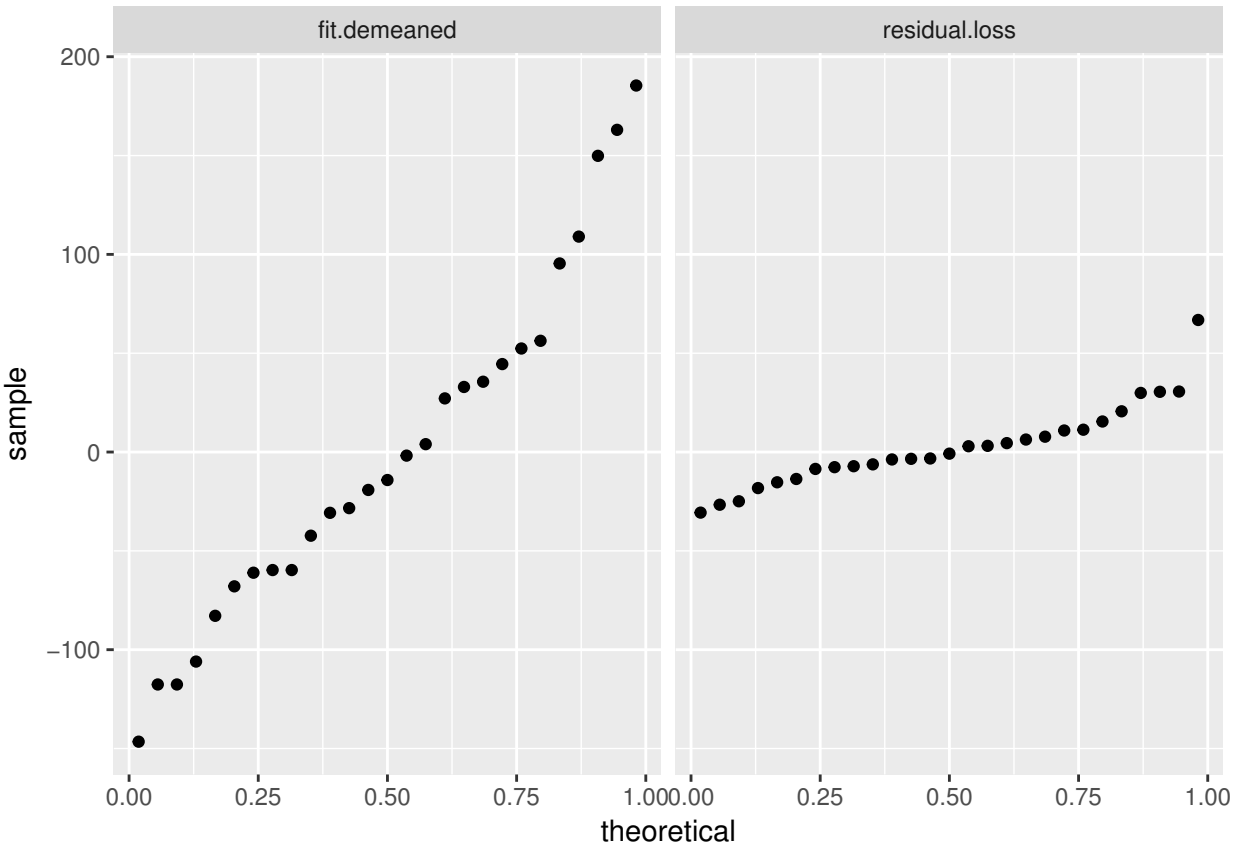
```
ggplot(rubber.rlm3.df, aes(sample = residual.loss)) + stat_qq()
```



There's a little upward curvature in the plot, indicating right-skew. This is heavily driven by the one large positive residual. It shouldn't be a problem – it means that we shouldn't use probabilistic methods on this data, but dropping outliers should have already prevented us from doing this.

Finally, we want to know whether our model acutally explains much. Draw a residual-fit plot:

```
library(tidyr)
rubber.rlm3.long = rubber.rlm3.df %>% gather(component, abrasion.loss, c(fit.demeaned,
    residual.loss))
ggplot(rubber.rlm3.long, aes(sample = abrasion.loss)) + stat_qq(distribution = "qunif") +
    facet_grid(~component)
```



The spread of the fitted values greatly exceeds the fit of the residuals. That means we haven't wasted our time (well, in my opinion.)

In summary, exploration has allowed us to see features of the data that aren't immediately obvious without visualization:

- The decreasing relationship between tensile strength and abrasion loss dies off for a tensile strength above about 180.
- The interaction between hardness and tensile strength is weak, but it's there.
- There are outliers with low tensile strength, high hardness, and low abrasion loss. They're different enough that they arguably should be separated from the rest of the data.

4.2 Ethanol

READ: Cleveland pp. 188–190, 196–199, 214–217, 254–255.

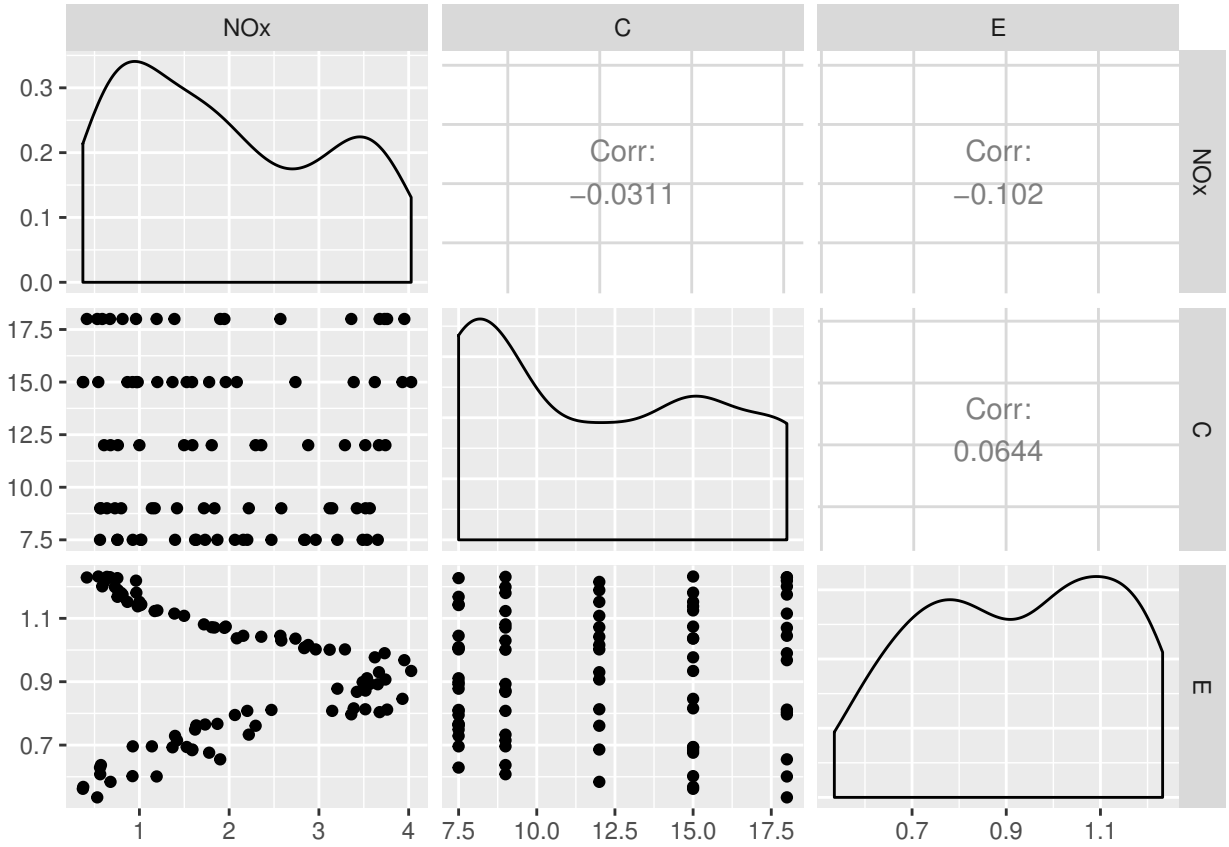
4.2.1 The ethanol data

The data frame `ethanol` in `lattice.RData` contains three measurements from 88 runs of an experiment testing an ethanol-fueled engine:

- `NOx`: the amount of oxides of nitrogen produced by the engine per unit of work, in micrograms per joule (the response.)
- `C`: the compression ratio the engine was set at.
- `E`: the equivalence ratio the engine was set at (a measure of how fuel is in the fuel-air mixture.)

We first plot the pairwise relationships.

```
load("lattice.RData")
library(ggplot2)
library(GGally)
ggpairs(ethanol)
```

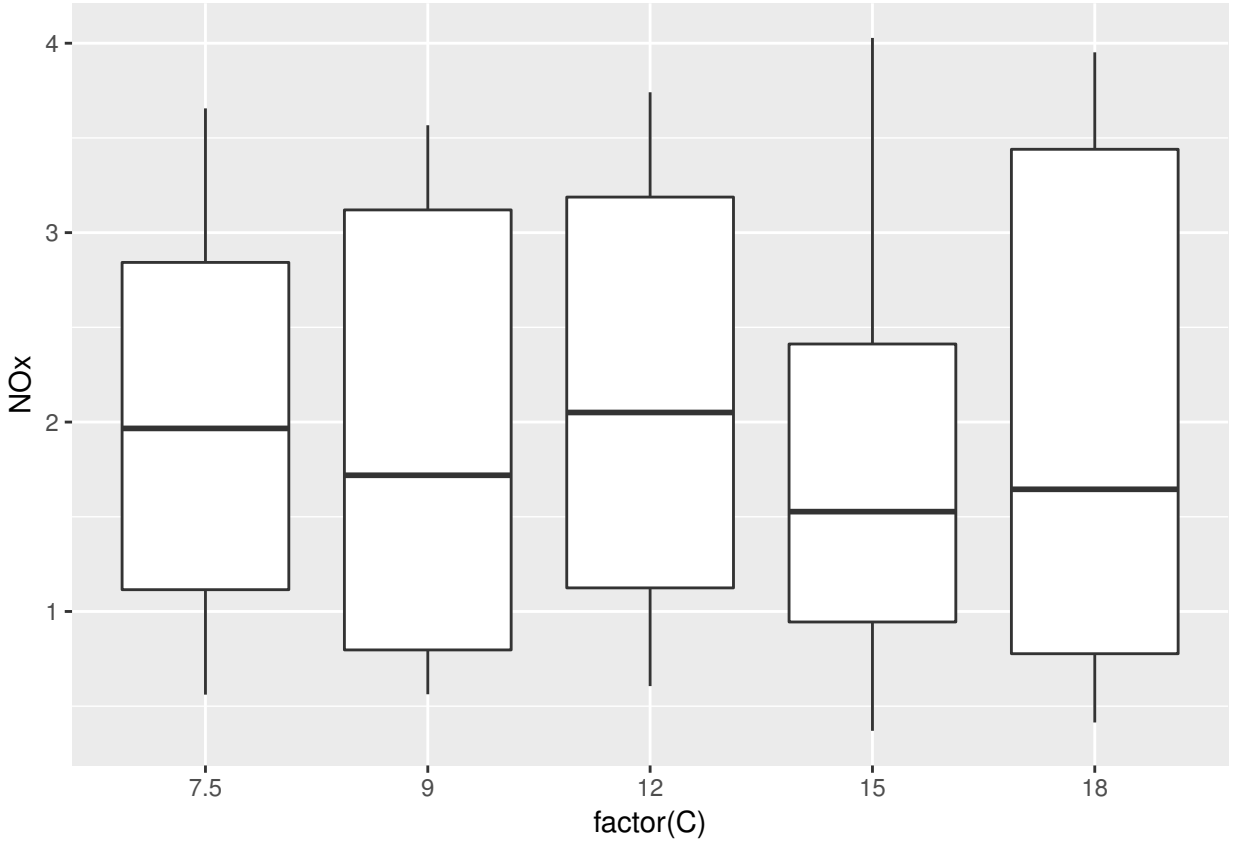


Some features are immediately obvious:

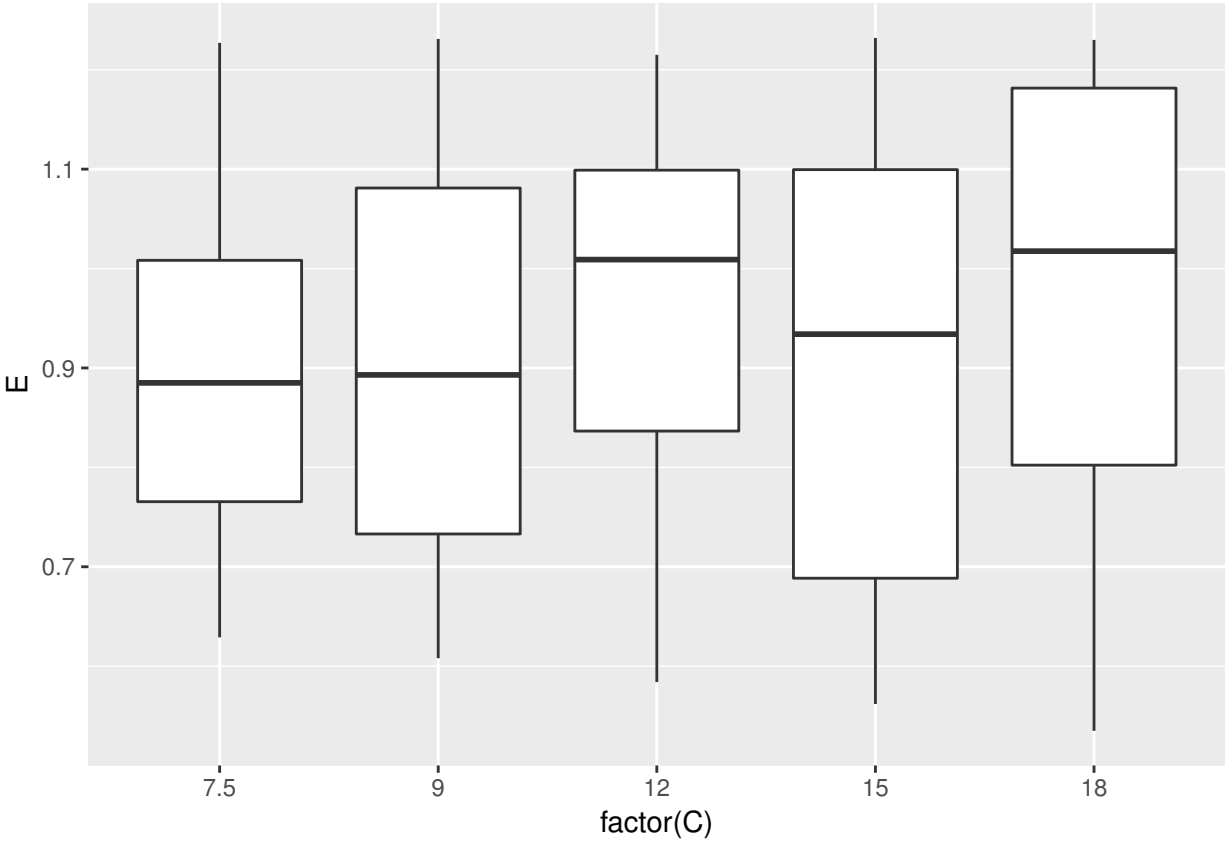
- Compression takes only five different values (it was directly controlled in the experiments.)
- The relationship of NOx as a function of equivalence ratio is non-linear and non-monotonic. (It actually looks kind of like a bell-shaped curve.)

On the other hand, the relationships of equivalence ratio and NOx with compression are not obvious. With five levels of C, we could try boxplots:

```
ggplot(ethanol, aes(x = factor(C), y = NOx)) + geom_boxplot()
```




```
ggplot(ethanol, aes(x = factor(C), y = E)) + geom_boxplot()
```

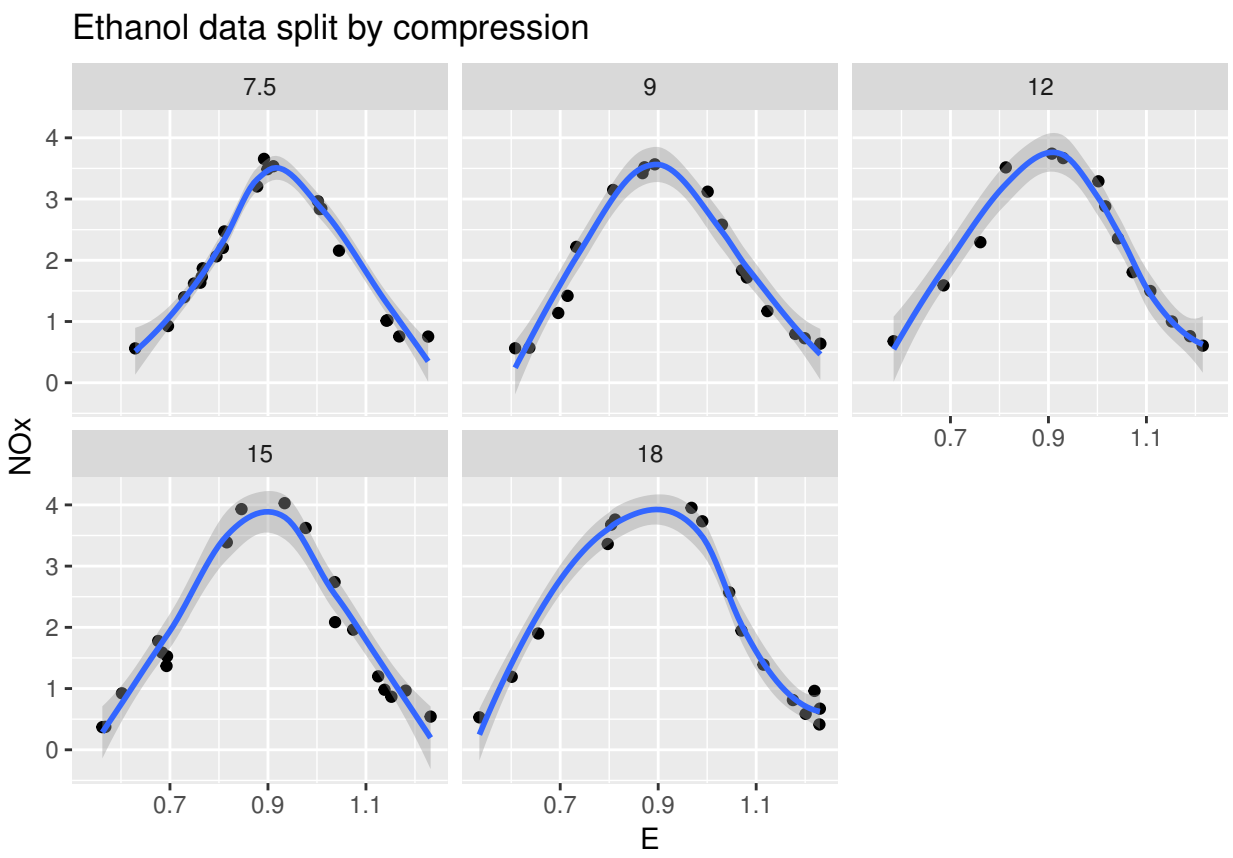


This isn't much clearer. We should try conditional plots of the data.

4.2.2 Coplots and colored plots of the data

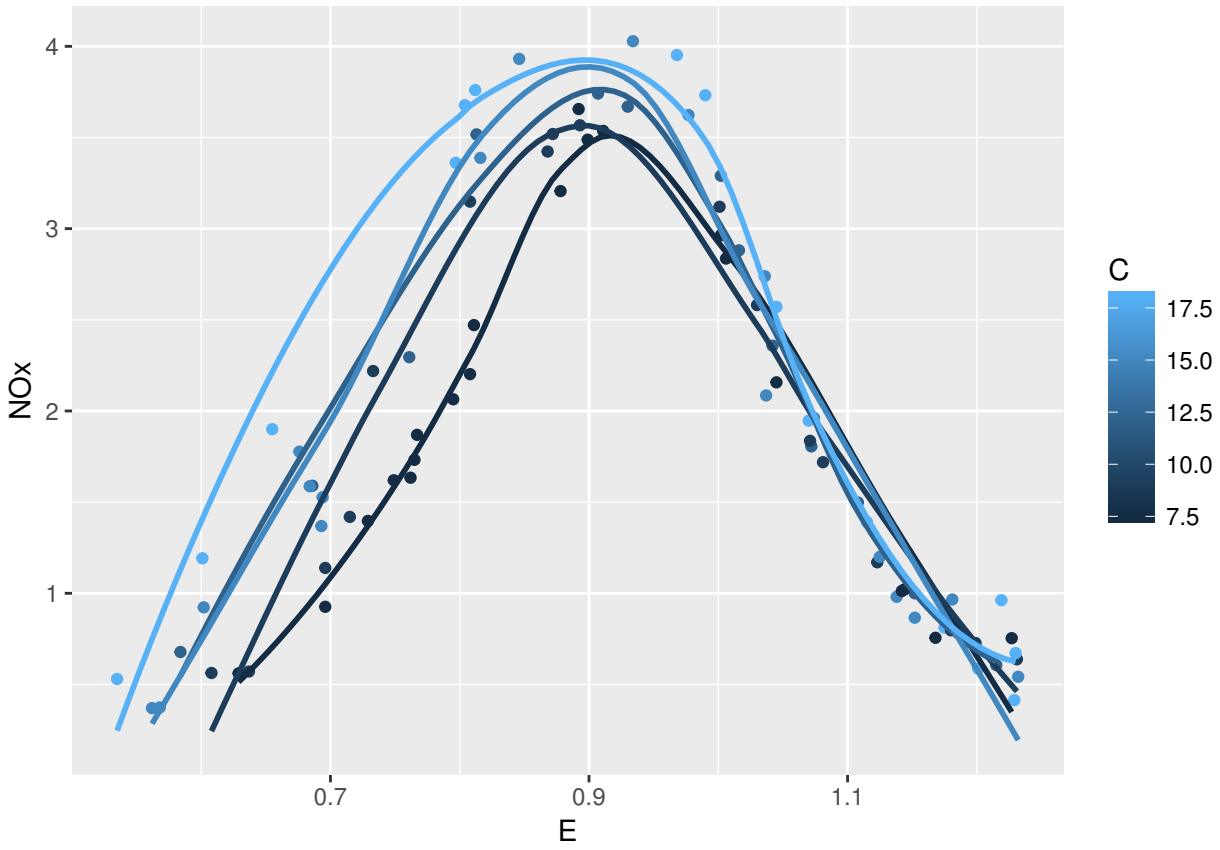
Let's first condition on C: since there are only five values, this is more straightforward. For each C, what's the relationship between E and NOx?

```
ggplot(ethanol, aes(x = E, y = NOx)) + geom_point() + geom_smooth() + facet_wrap(~C,
    ncol = 3) + labs(title = "Ethanol data split by compression")
```



The shape is very similar across all five plots: the trend is increasing up to a peak at an equivalence ratio of about 0.9, then a decline. It's hard to tell the difference in level (for example, which curve peaks the highest) from these graphs. That kind of comparison would be easier if we plotted all five of these curves on one graph. To make the graph clear, we plot each curve in a different color, where the color varies along a gradient with compression ratio.

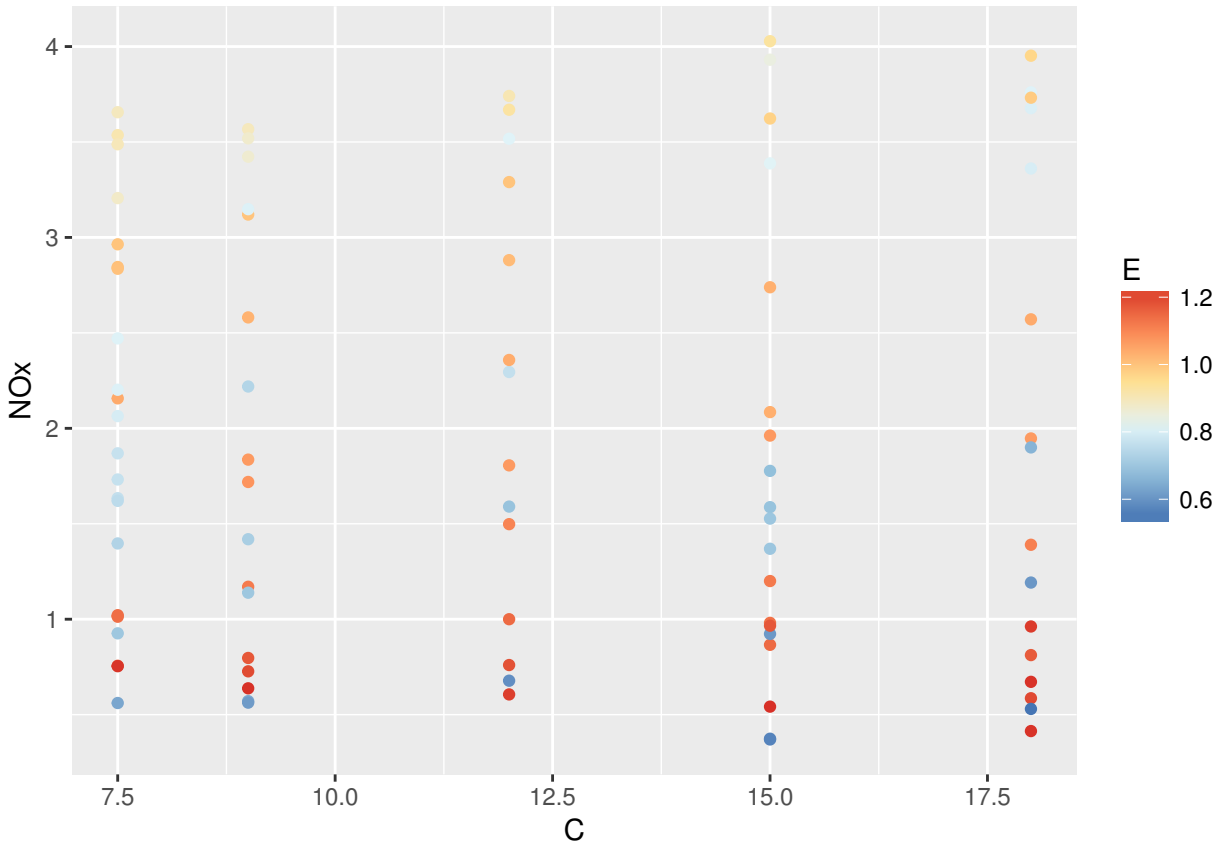
```
ggplot(ethanol, aes(x = E, y = NOx, group = C, color = C)) + geom_point() +
    geom_smooth(se = FALSE)
```



The lower compression ratios are darker. (There are lots of color options to play around with; look up, for instance, the help for `scale_color_distiller()`.) We can now see the highest values of C give the highest peak value of NOx. Note that the curves aren't simply shifted: they're relatively far apart on the left-hand side and relatively close on the right.

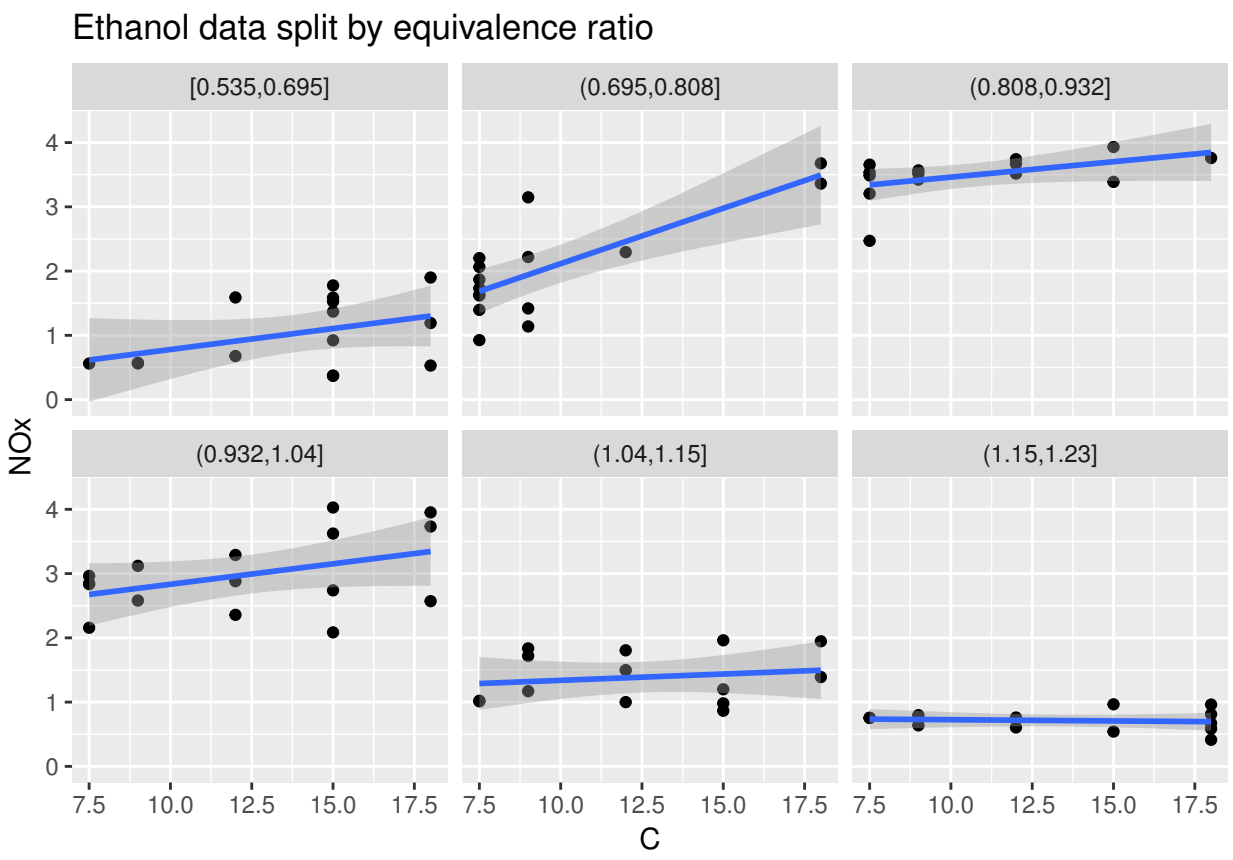
Now let's see how the relationship between C and NOx varies with E. Color the points according to E. A nonlinear relationship gives us an excuse to use a more dramatic color scheme.

```
ggplot(ethanol, aes(x = C, y = NOx, group = E, color = E)) + geom_point() +
    scale_color_distiller(palette = "RdYlBu")
```



As we saw before, for low E (blue), NOx is low; for middling E (yellow), NOx is high; and for high E (red), NOx is low again. However, it's hard to see how the relationship with C changes. Let's cut E into six class and plot six scatterplots of NOx against C. It turns out a linear fit seems adequate:

```
ggplot(ethanol, aes(x = C, y = NOx)) + geom_point() + geom_smooth(method = "lm") +
    facet_wrap(~cut_number(E, n = 6), ncol = 3) + labs(title = "Ethanol data split by equivalence ratio
```



Once again, middling E's give the highest NOx. However, now we can also see that the slope is steepest for a fairly low E, then decreases until the line for the highest E is basically flat.

4.2.3 Fitting and visualizing a model

Conditioning on E gives the simpler structure: then NOx is just a linear function of C. However, we still need an interaction between C and E. `loess(NOx ~ C * E...)` by default would fit a smooth two-dimensional surface to predict NOx, but wouldn't have the conditional linearity. To achieve this, we use the `parametric` argument to specific a parametric model in C, and the `drop.square` argument to give a linear term in C rather than the default quadratic. After some (actually a lot) of trial and error, we find a span of 1/3 looks okay.

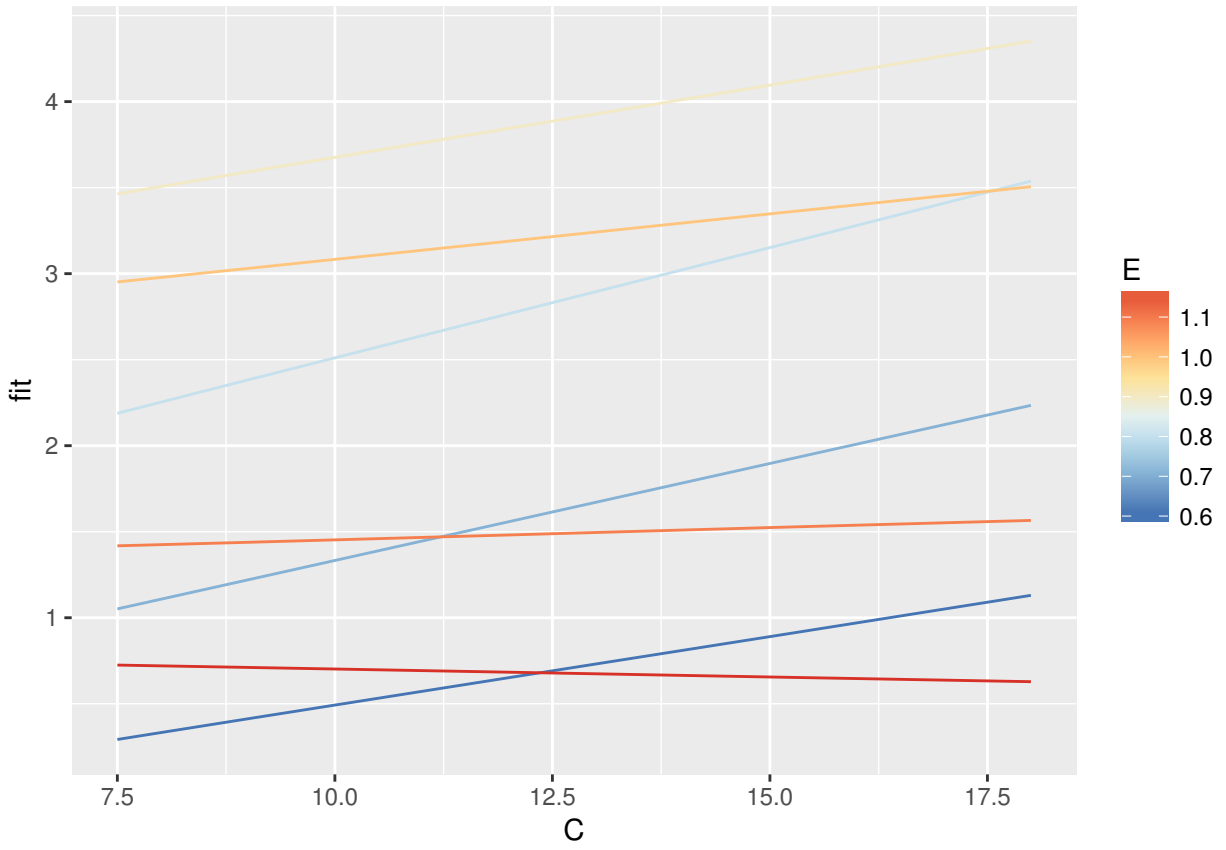
```
ethanol.lo = loess(NOx ~ C * E, data = ethanol, span = 1/3, parametric = "C",
    drop.square = "C", family = "symmetric")
```

For display, create a grid of C and E points, then predict the NOx at each point on this grid.

```
ethanol.grid = expand.grid(C = c(7.5, 9, 12, 15, 18), E = seq(0.6, 1.2, 0.1))
ethanol.predict = predict(ethanol.lo, newdata = ethanol.grid)
```

We can plot lines for different values of E on the same graph in different colors, but it gets a bit crowded.

```
ggplot(data.frame(ethanol.grid, fit = as.vector(ethanol.predict)), aes(x = C,
    y = fit, group = E, color = E)) + geom_line() + scale_color_distiller(palette = "RdYlBu")
```



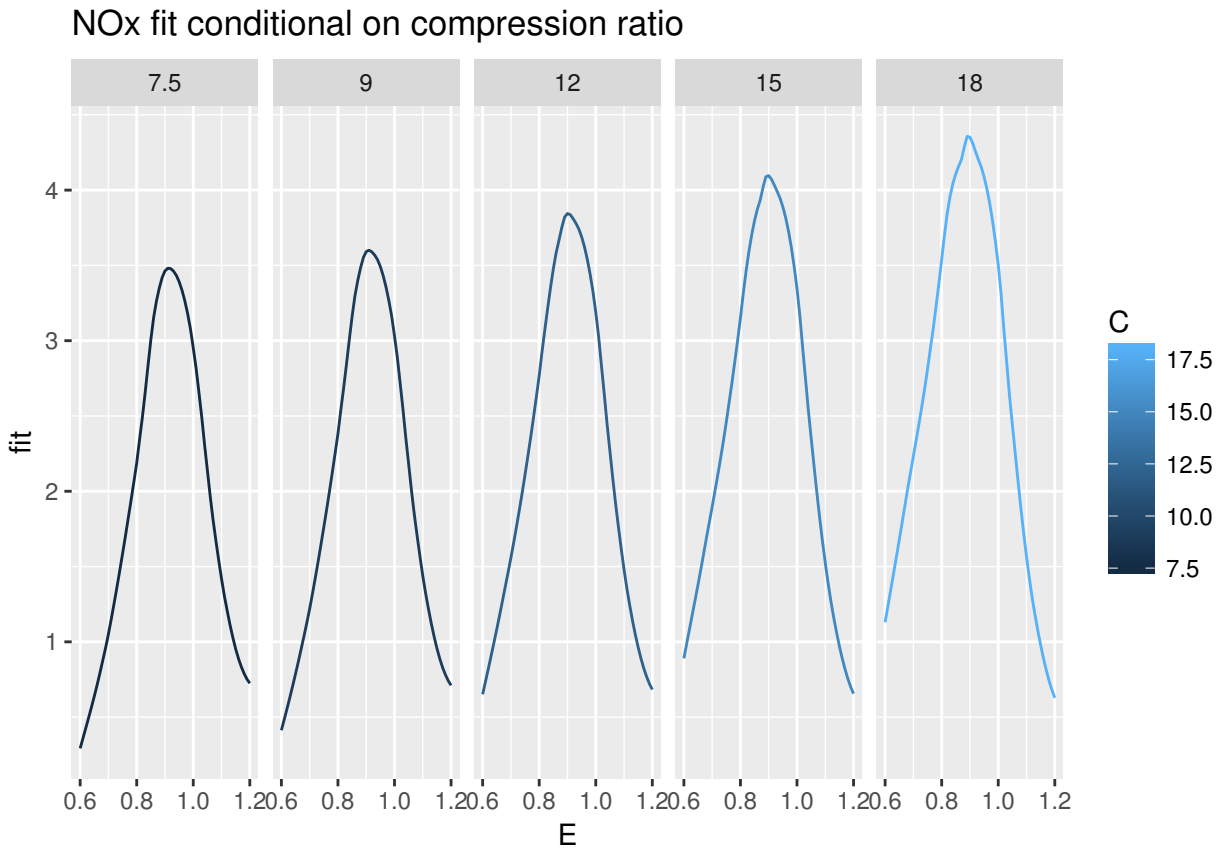
A coplot of the fit faceted by values of E "spreads out" the above graph.

```
ggplot(data.frame(ethanol.grid, fit = as.vector(ethanol.predict)), aes(x = C,
    y = fit, color = E)) + geom_line() + facet_grid(~E) + scale_color_distiller(palette = "RdYlBu") +
    labs(title = "NOx fit conditional on equivalence ratio")
```

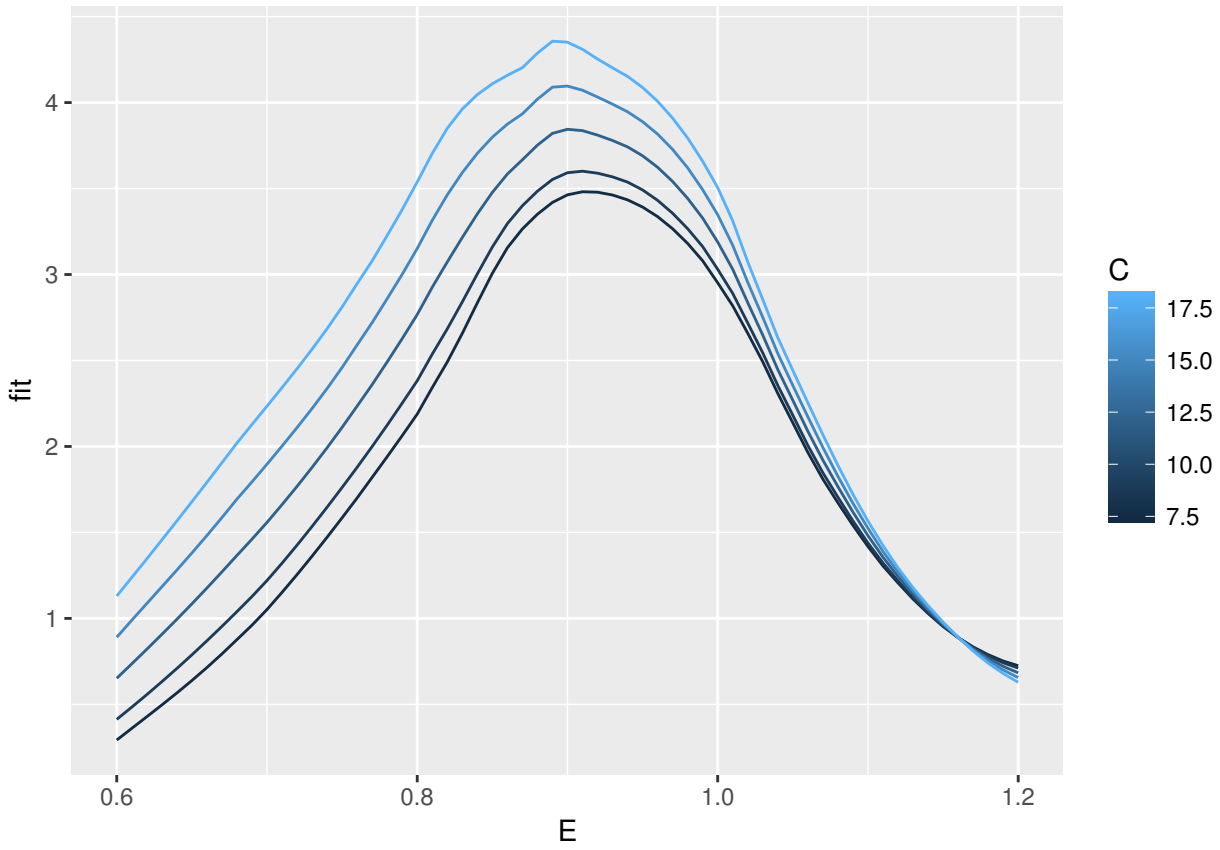
This is much clearer (and makes the graph before this one much easier to understand as well.) In addition to the higher values for E around 0.9, the slope varies with E as well. The steepest slope looks to be at around E = 0.8.

We now draw coplots of the fit conditioning on the compression ratio. Because of the nonlinearity, we need a tightly packed grid in E. It makes sense to use the five values of C from the experiment.

```
ethanol.grid2 = expand.grid(C = c(7.5, 9, 12, 15, 18), E = seq(0.6, 1.2, 0.01))
ethanol.predict2 = predict(ethanol.lo, newdata = ethanol.grid2)
ggplot(data.frame(ethanol.grid2, fit = as.vector(ethanol.predict2)), aes(x = E,
    y = fit, color = C)) + geom_line() + facet_grid(~C) + labs(title = "NOx fit conditional on compressi
```




```
ggplot(data.frame(ethanol.grid2, fit = as.vector(ethanol.predict2)), aes(x = E,
    y = fit, group = C, color = C)) + geom_line()
```



The general shape is similar for all five curves. As with the raw data, the curves are separated for low E (with high C giving higher NOx), but come together for high E.

4.2.4 Exploring the residuals

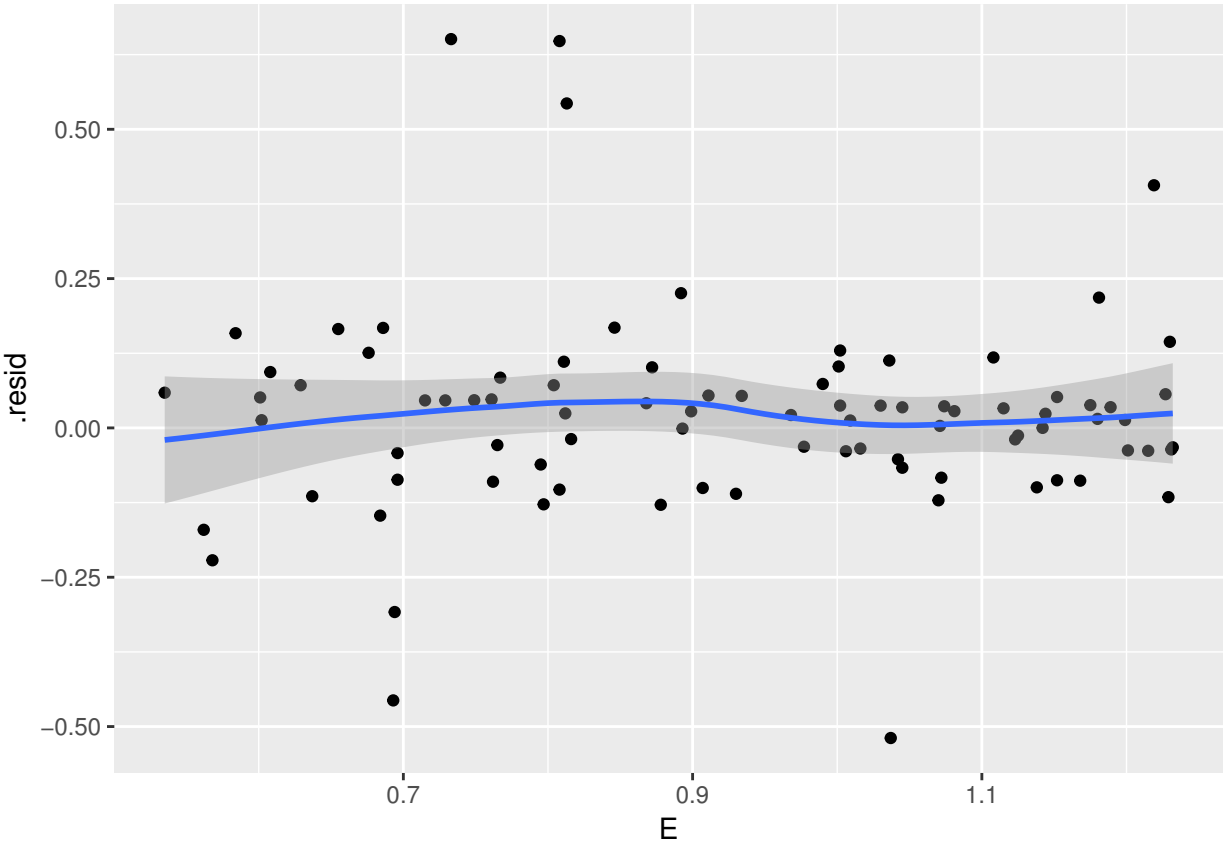
Let's first look at the numerical summary of the model.

```
summary(ethanol.lo)
```

```
## Call:
## loess(formula = NOx ~ C * E, data = ethanol, span = 1/3, parametric = "C",
##     drop.square = "C", family = "symmetric")
##
## Number of Observations: 88
## Equivalent Number of Parameters: 13.59
## Residual Scale Estimate: 0.1186
## Trace of smoother matrix: 16.72  (exact)
##
## Control settings:
##   span     :  0.3333333
##   degree   :  2
##   family   :  symmetric      iterations = 4
##   surface  :  interpolate      cell = 0.2
##   normalize:  TRUE
##  parametric:  TRUE FALSE
## drop.square:  TRUE FALSE
```

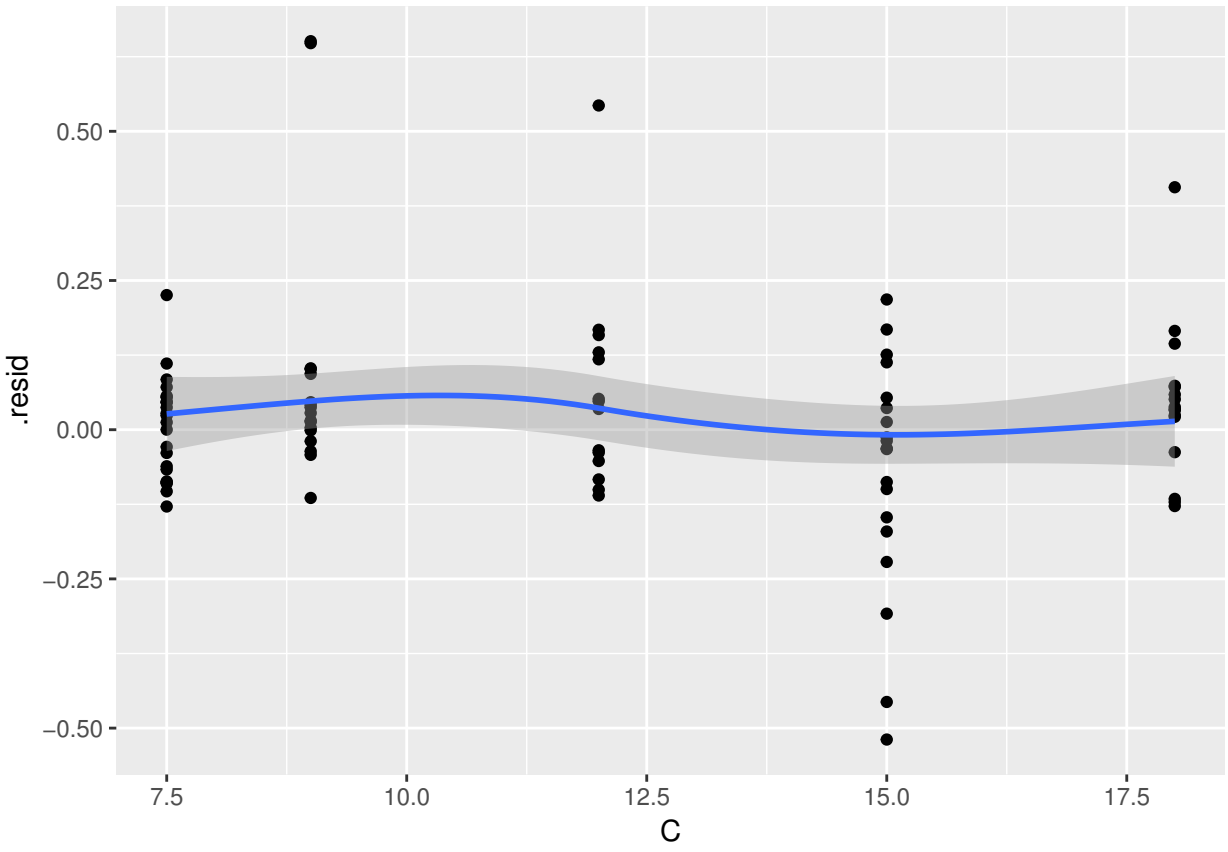
I don't know about you, but that's not very useful to me. Instead, let's construct a data frame with the original variables and the residuals, and getting plotting.

```
library(broom)
ethanol.lo.df = augment(ethanol.lo)
ggplot(ethanol.lo.df, aes(x = E, y = .resid)) + geom_point() + geom_smooth(method.args = list(degree =
```



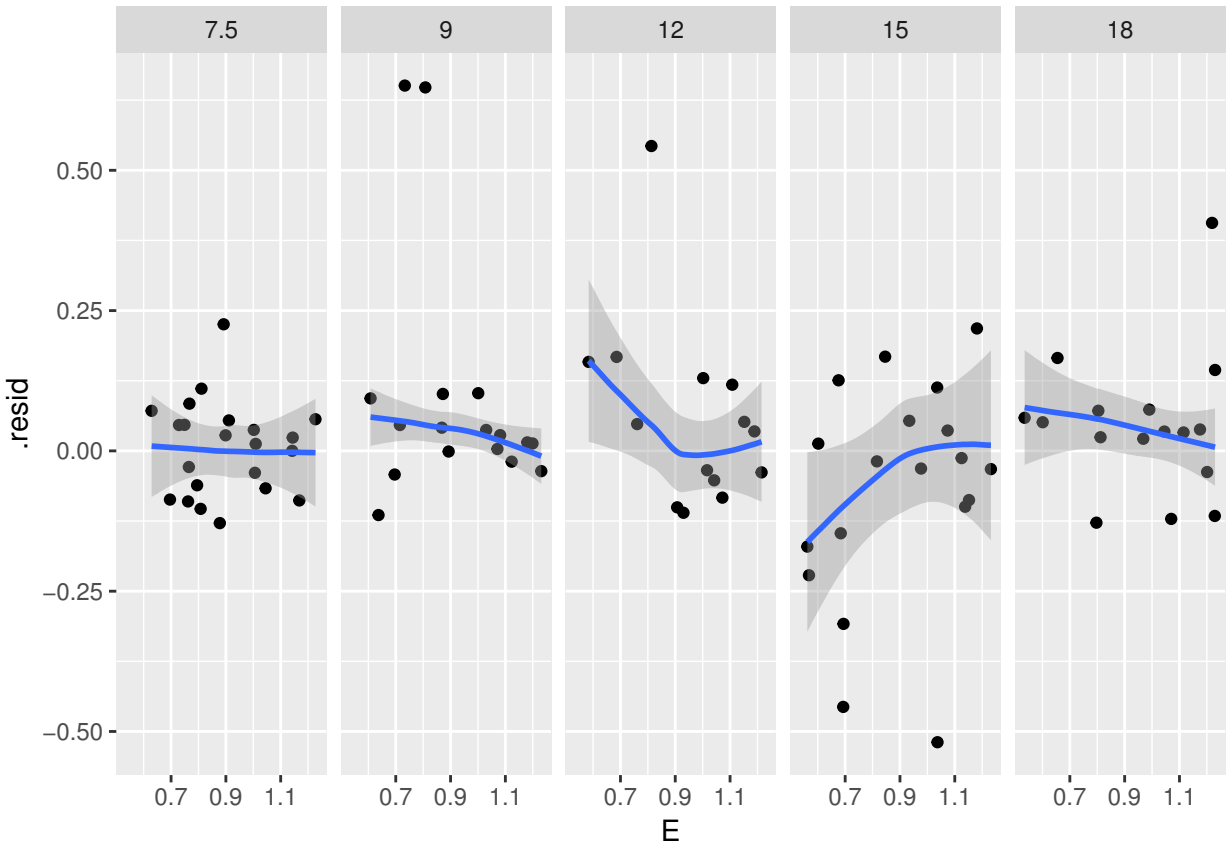
There's no obvious sign of a relationship between the residuals and the equivalence ratio. Now plot the residuals against compression.

```
ggplot(ethanol.lo.df, aes(x = C, y = .resid)) + geom_point() + geom_smooth(method.args = list(degree =
```



The residuals for $C = 9$ are a little concerning: they do seem to be centred a little above zero. While this could be due to nonlinearity, it could also be measurement error or just the one large positive residual. To investigate further, let's condition on C and look at the relationship between the residuals and the equivalence ratio.

```
ggplot(ethanol.lo.df, aes(x = E, y = .resid)) + geom_point() + geom_smooth(span = 1,
    method.args = list(degree = 1, family = "symmetric")) + facet_grid(~C)
```

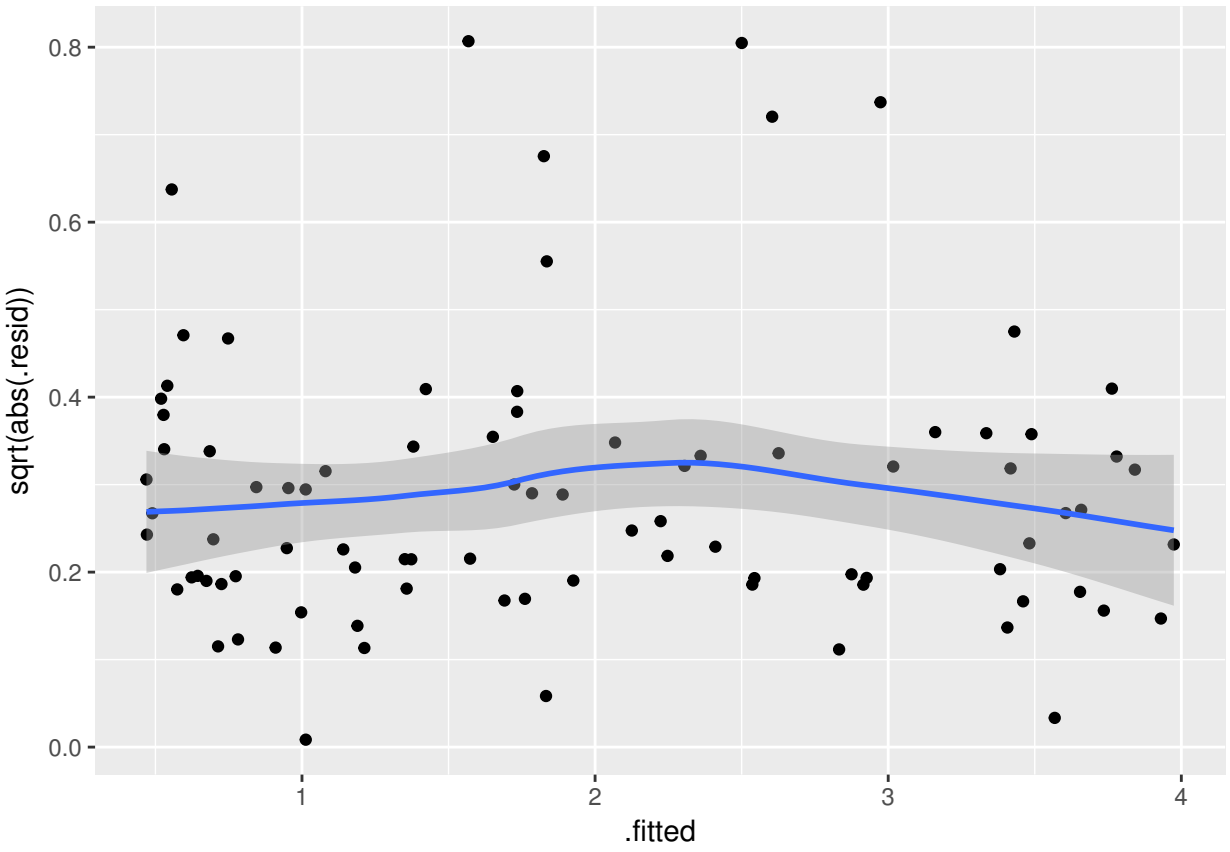


There doesn't seem to be any systematic relationship in the trends of the slopes. The outliers present a minor problem, but if you ignore them there doesn't seem to be too much pattern in the residuals. The fit is probably adequate.

4.2.5 Checking the residuals

We now do the boring stuff. Check for homoskedasticity:

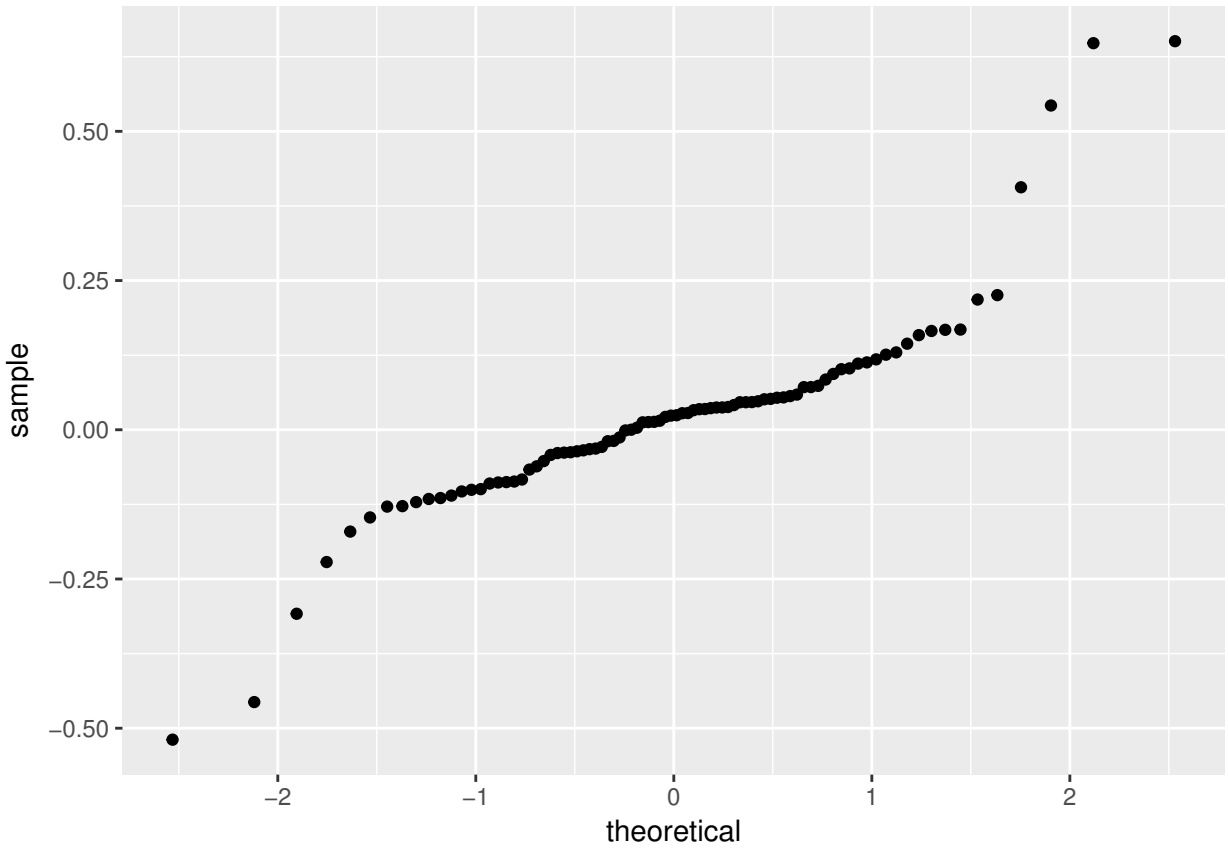
```
ggplot(ethanol.lo.df, aes(x = .fitted, y = sqrt(abs(.resid)))) + geom_point() +
    geom_smooth(method.args = list(degree = 1))
```



The spread-location plot of the transformed residuals is reasonably consistent with a horizontal line. There's negligible evidence of heteroskedasticity.

Next, check normality:

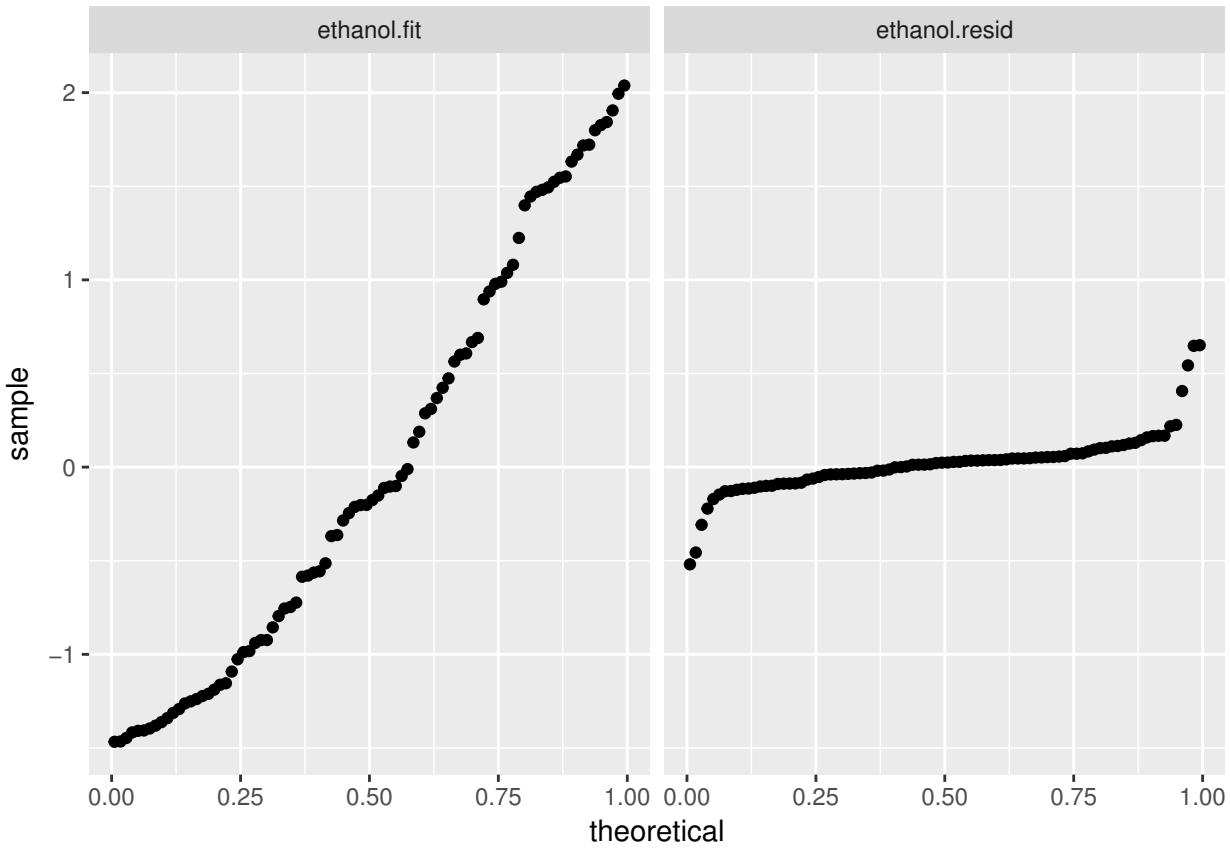
```
ggplot(ethanol.lo.df, aes(sample = .resid)) + stat_qq()
```



The outliers means the residuals aren't normal, so we should hesitate to make probabilistic statements. Oh well.

Finally, did we manage to explain anything?

```
ethanol.fit = ethanol.lo.df$.fitted - mean(ethanol.lo.df$.fitted)
ethanol.resid = ethanol.lo.df$.resid
library(tidyr)
ethanol.lo.long = data.frame(ethanol.fit, ethanol.resid) %>% gather(component,
    NOx)
ggplot(ethanol.lo.long, aes(sample = NOx)) + stat_qq(distribution = "qunif") +
    facet_grid(~component)
```

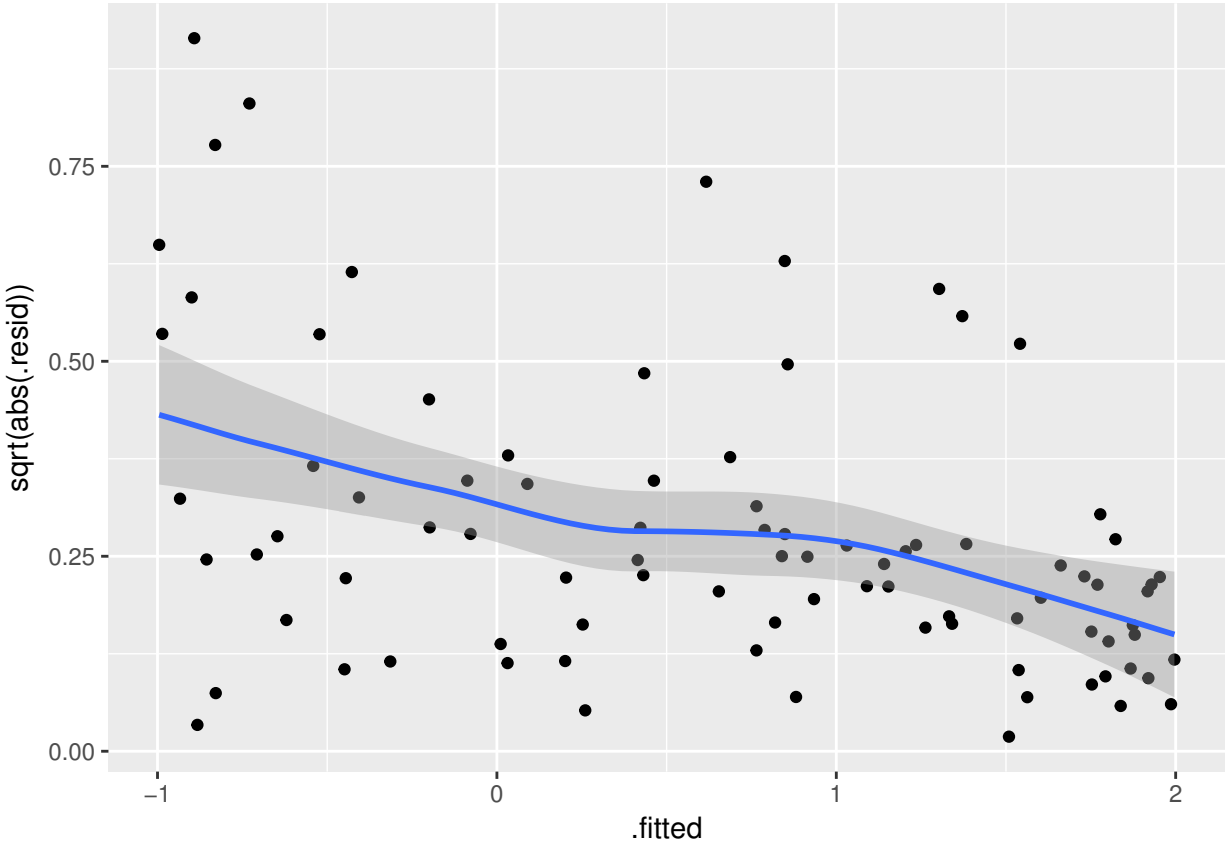


The spread of the (demeaned) fitted values is much greater than the spread of the residuals. So the model is explaining most of the variation.

4.2.6 Should we have transformed?

We can try out a log transformation and see if it does any better. Let's skip to the residuals and check for homoskedasticity:

```
# Compare log NOx fit
ethanol.log.lo = loess(log2(NOx) ~ C * E, span = 1/3, parametric = "C", drop.square = "C",
    family = "symmetric", data = ethanol)
ethanol.log.lo.df = augment(ethanol.log.lo)
ggplot(ethanol.log.lo.df, aes(x = .fitted, y = sqrt(abs(.resid)))) + geom_point() +
    geom_smooth(method.args = list(degree = 1))
```



The spread of the residuals decreases as the fitted values increase. The log fit does worse than the fit on the original scale.

We conclude:

- NOx depends on equivalence ratio in a non-monotonic way.
- Conditional on equivalence ratio, NOx depends on concentration in an approximately linear way.
- The interaction is important: there's no real way to remove it from the data.
- The usual inference based on an assumption of normal errors is inappropriate.
- Transformations don't appear to help and may make things worse.

4.3 Contour plots

READ: Cleveland pp. 228–248.

4.3.1 Velocities of the NGC 7531 galaxy

The data frame `galaxy` contains 323 measurements of velocity for locations in galaxy NGC 7531.

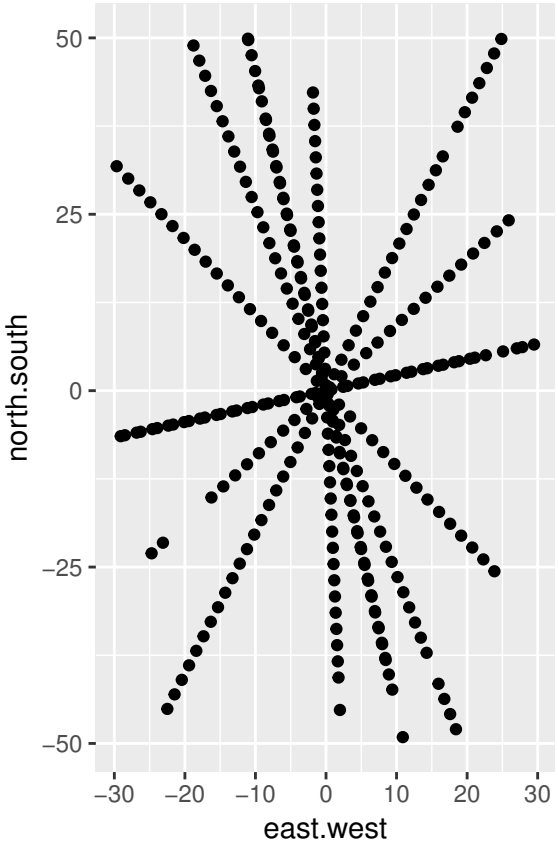
```
load("lattice.RData")
summary(galaxy)
```

```
##    east.west          north.south          angle        radial.position
##  Min.   :-29.66693   Min.   :-49.108   Min.   : 12.50   Min.   :-52.4000
##  1st Qu.: -7.91688   1st Qu.:-13.554   1st Qu.: 63.50   1st Qu.:-21.3500
##  Median : -0.06493   Median :  0.671   Median : 92.50   Median : -0.8000
##  Mean   : -0.33237   Mean   :  1.521   Mean   : 80.89   Mean   : -0.8427
```

```
##  3rd Qu.:  6.95053   3rd Qu.: 18.014   3rd Qu.:102.50   3rd Qu.: 19.6500
##  Max.   : 29.48414   Max.   : 49.889   Max.   :133.00   Max.   : 55.7000
##     velocity
##  Min.   :1409
##  1st Qu.:1523
##  Median :1586
##  Mean   :1594
##  3rd Qu.:1669
##  Max.   :1775
```

`velocity` (in km/second) is the response variables, and the locations along `east.west` and `north.south` coordinates are the main explanatory variables. (North and west are positive – if you lie on your back with your head pointing north, then west is to your right.) First, we simply look where the measurements were taken:

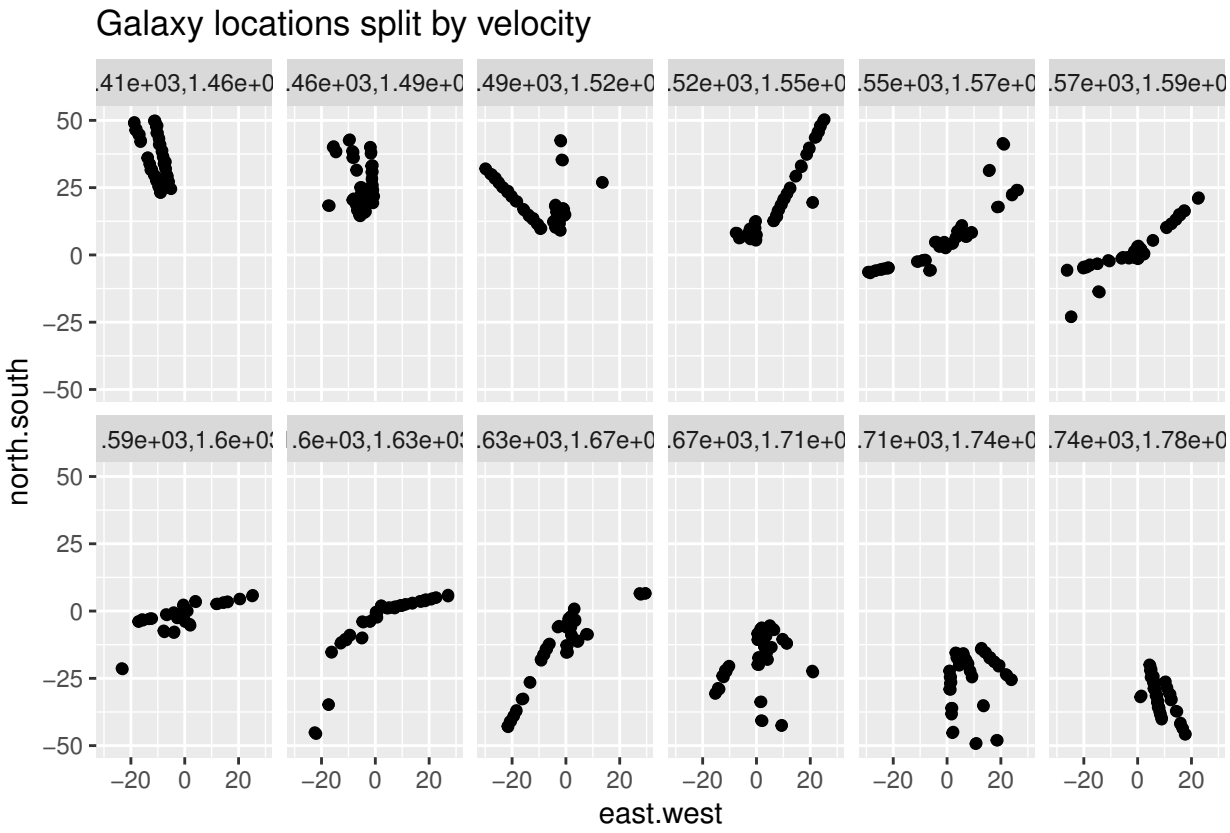
```
library(ggplot2)
ggplot(galaxy, aes(x = east.west, y = north.south)) + geom_point() + coord_fixed()
```



We see the measurements were taken along seven different lines ("slits".) The variables `radial.position` and `angle` give the locations in polar coordinates.

How does velocity vary with location? We first facet by the `velocity` variable.

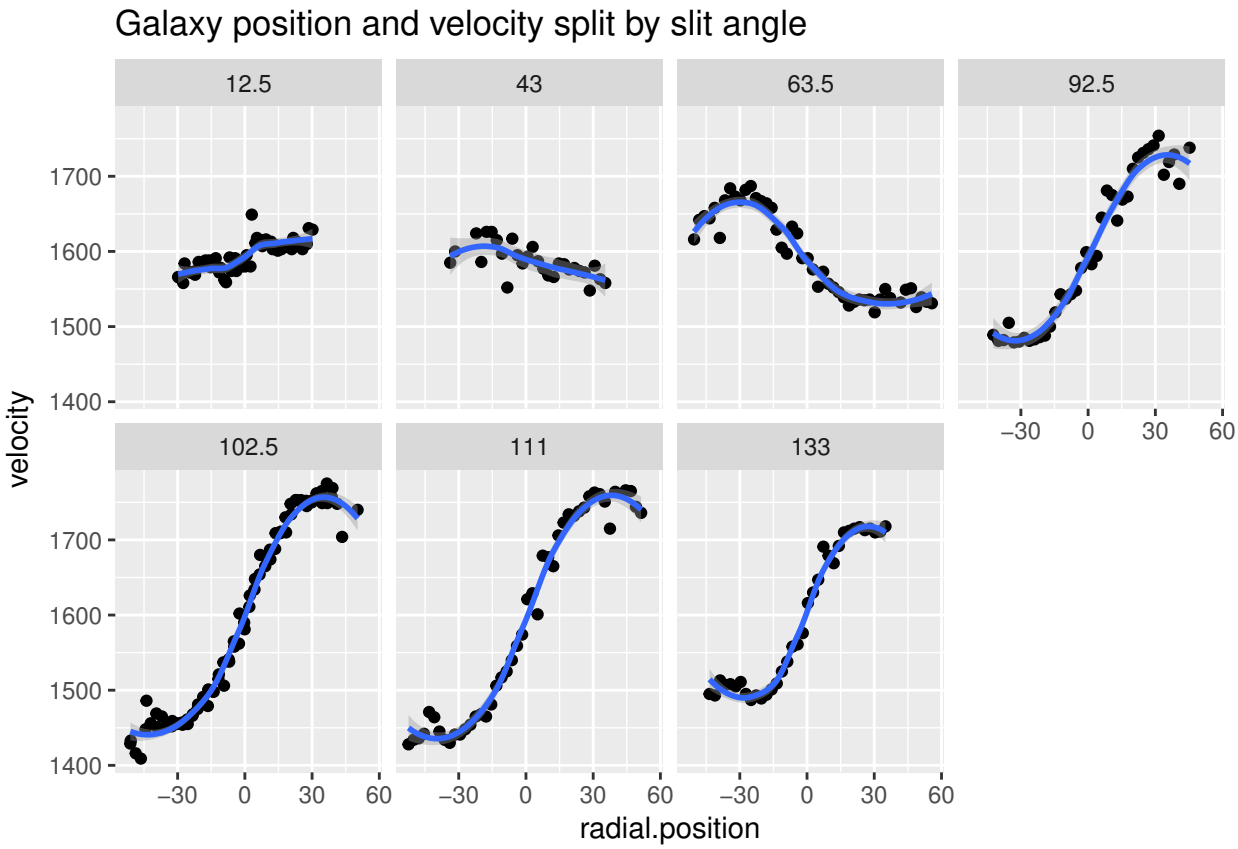
```
ggplot(galaxy, aes(x = east.west, y = north.south)) + geom_point() + geom_jitter(width = 0.5,
    height = 0.5) + facet_wrap(~cut_number(velocity, n = 12), ncol = 6) + coord_fixed() +
    labs(title = "Galaxy locations split by velocity")
```



The slowest locations are in the top left panel – these are all in the northeast. The fastest locations in the bottom right panel – these are in the southwest.

We can also look at the velocities as a function of `radial.position`, faceted by `angle`.

```
ggplot(galaxy, aes(x = radial.position, y = velocity)) + geom_point() + geom_smooth() +
    facet_wrap(~angle, ncol = 4) + labs(title = "Galaxy position and velocity split by slit angle")
```



For each angle, we see a nonmonotonic relationship. The magnitude of the variation in velocities varies a lot – it's big at 92.5 to 111 degrees (NNE to SSW), and small for 12.5 and 43 degrees (WNW to ESE.)

4.3.2 Modeling galaxy velocities

We fit a full loess model with interaction. We add the argument `normalize=FALSE`, which is appropriate when the two explanatory variables are on the same scale and we don't want to standardize them. By trial and error, a smoothing parameter `span = 0.25` seems about right.

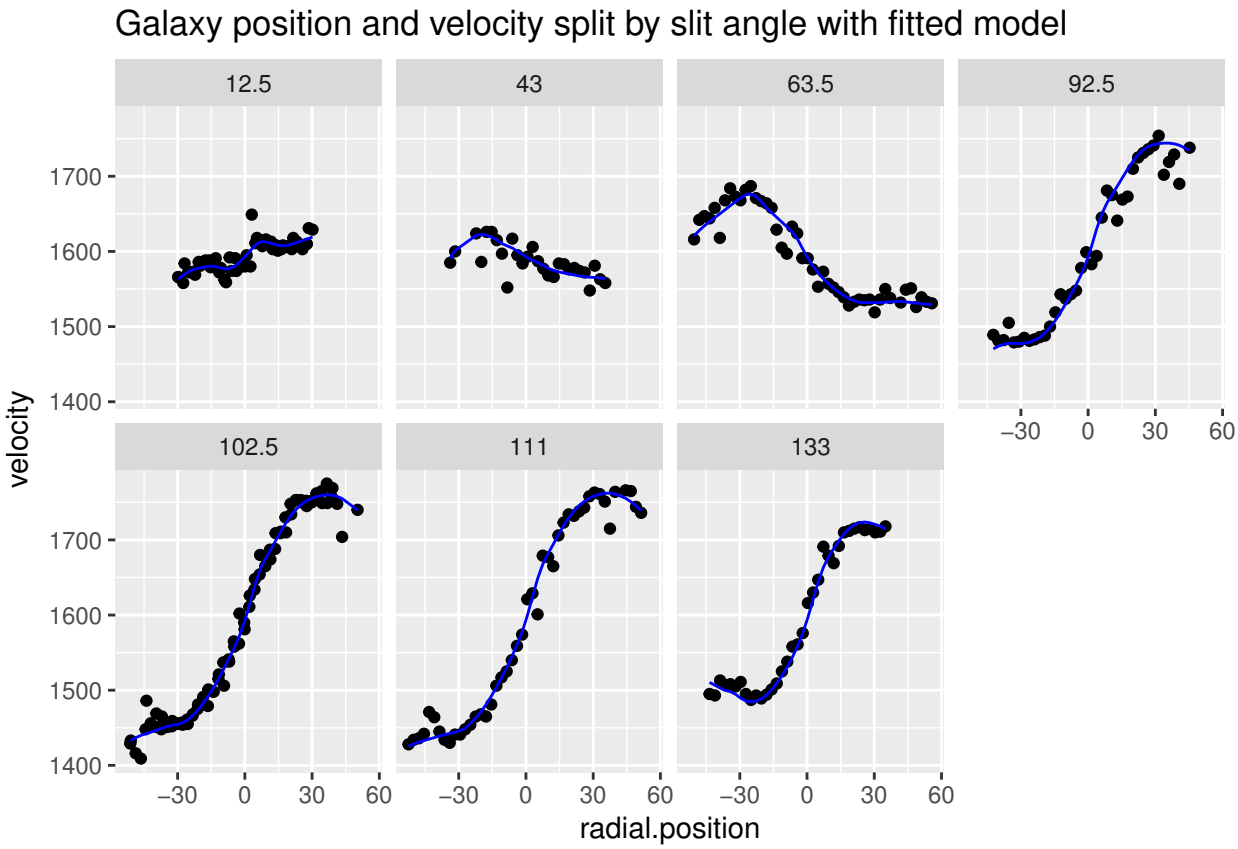
```
galaxy.lo = loess(velocity ~ east.west * north.south, data = galaxy, span = 0.25,
    family = "symmetric", normalize = FALSE)
```

We create a data frame with the original variables as well as the fitted values and residuals:

```
galaxy.lo.df = data.frame(galaxy, .fitted = fitted.values(galaxy.lo), .resid = residuals(galaxy.lo))
```

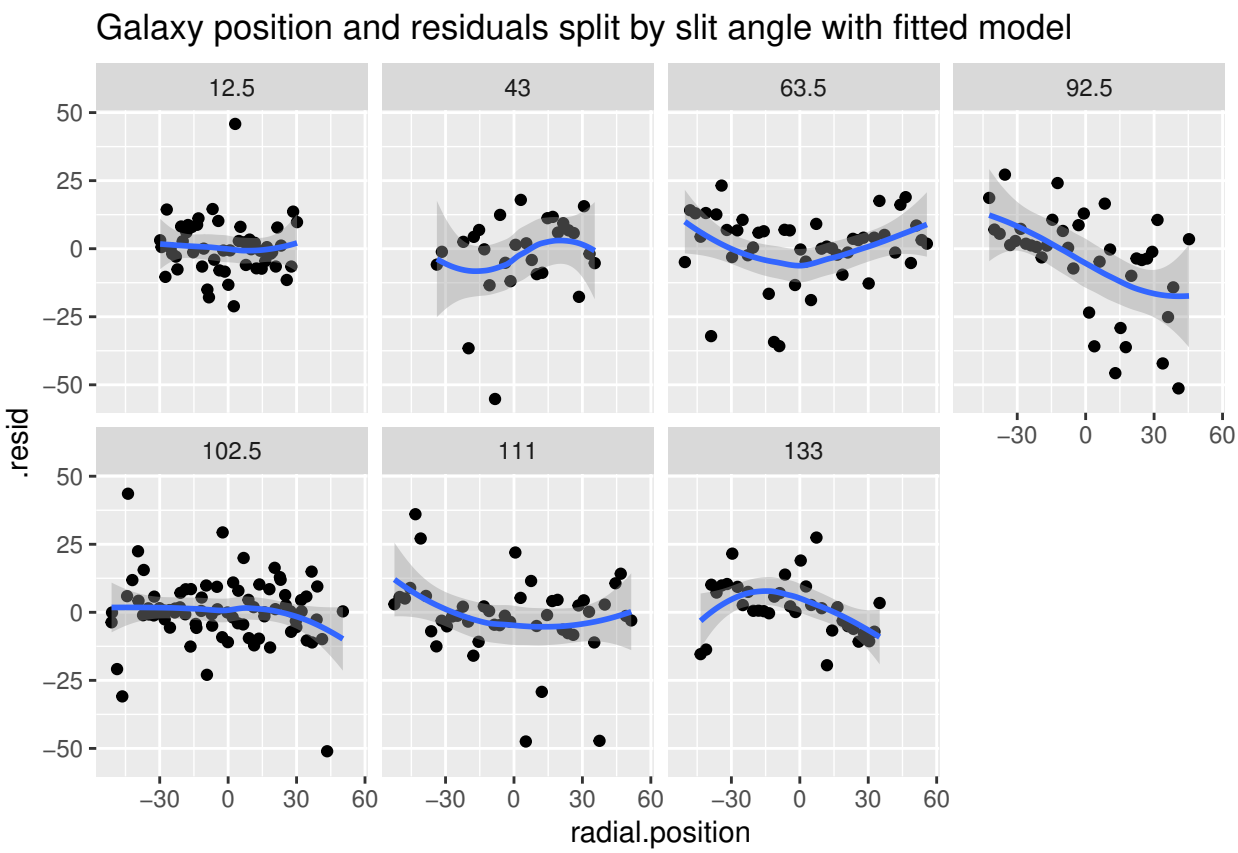
Now redraw the above plot faceted by angle with the fit added.

```
ggplot(galaxy.lo.df, aes(x = radial.position, y = velocity)) + geom_point() +
    geom_line(aes(x = radial.position, y = .fitted), color = "blue") + facet_wrap(~angle,
    ncol = 4) + labs(title = "Galaxy position and velocity split by slit angle with fitted model")
```



The model provides the right general shape to the data in polar coordinates. We take a closer look at the residuals:

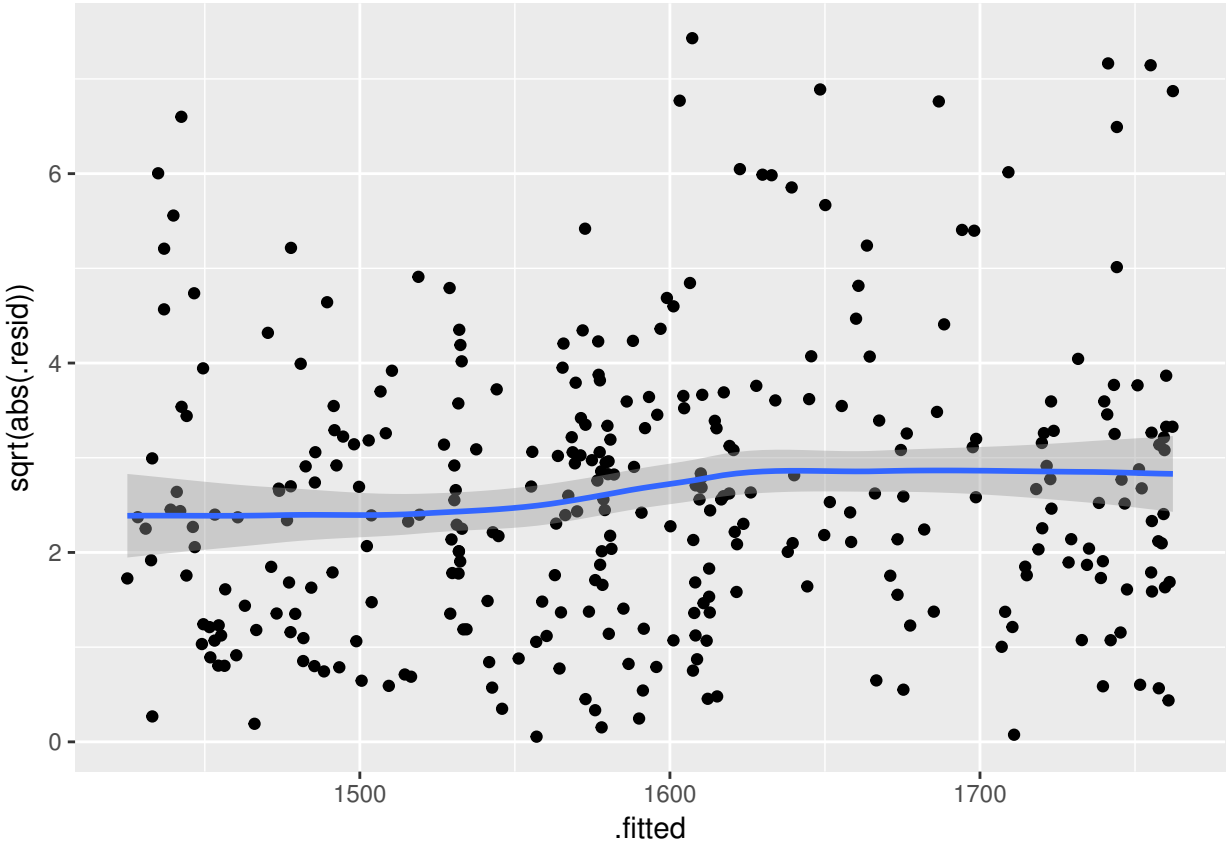
```
ggplot(galaxy.lo.df, aes(x = radial.position, y = .resid)) + geom_point() +
    geom_smooth(span = 1) + facet_wrap(~angle, ncol = 4) + labs(title = "Galaxy position and residuals
```



The 92.5 degrees plot is somewhat bad – it slopes down and it appears heteroskedastic. There's a bit more slope and/or curvature in the others than we would expect under random error as well. Nevertheless, we'll keep our model for simplicity.

We now do our usual residual diagnostics. The spread-location plot of transformed residuals:

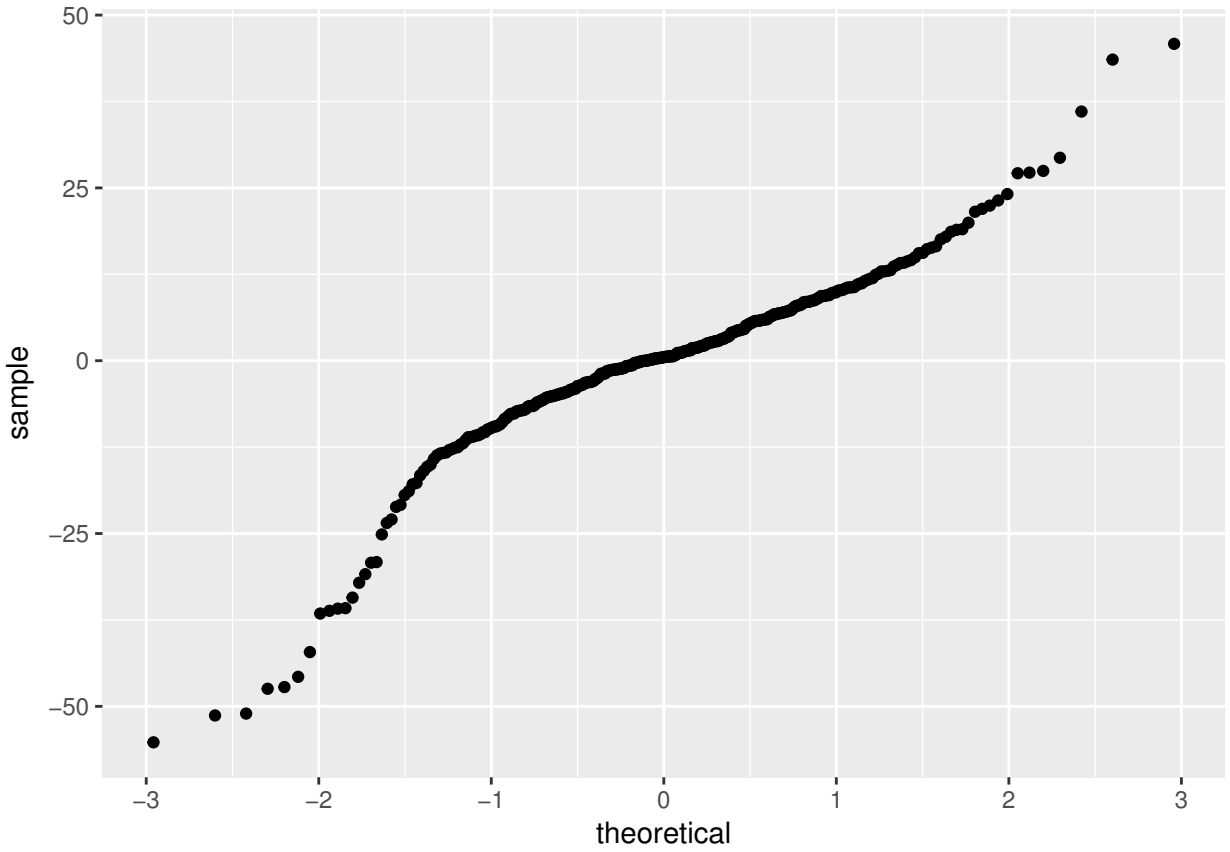
```
ggplot(galaxy.lo.df, aes(x = .fitted, y = sqrt(abs(.resid)))) + geom_point() +
    geom_smooth(method.args = list(degree = 1))
```



The magnitude of the residuals may increase slightly with fitted value, but it's not so bad that it would demand a transformation.

Normal QQ plot of the residuals:

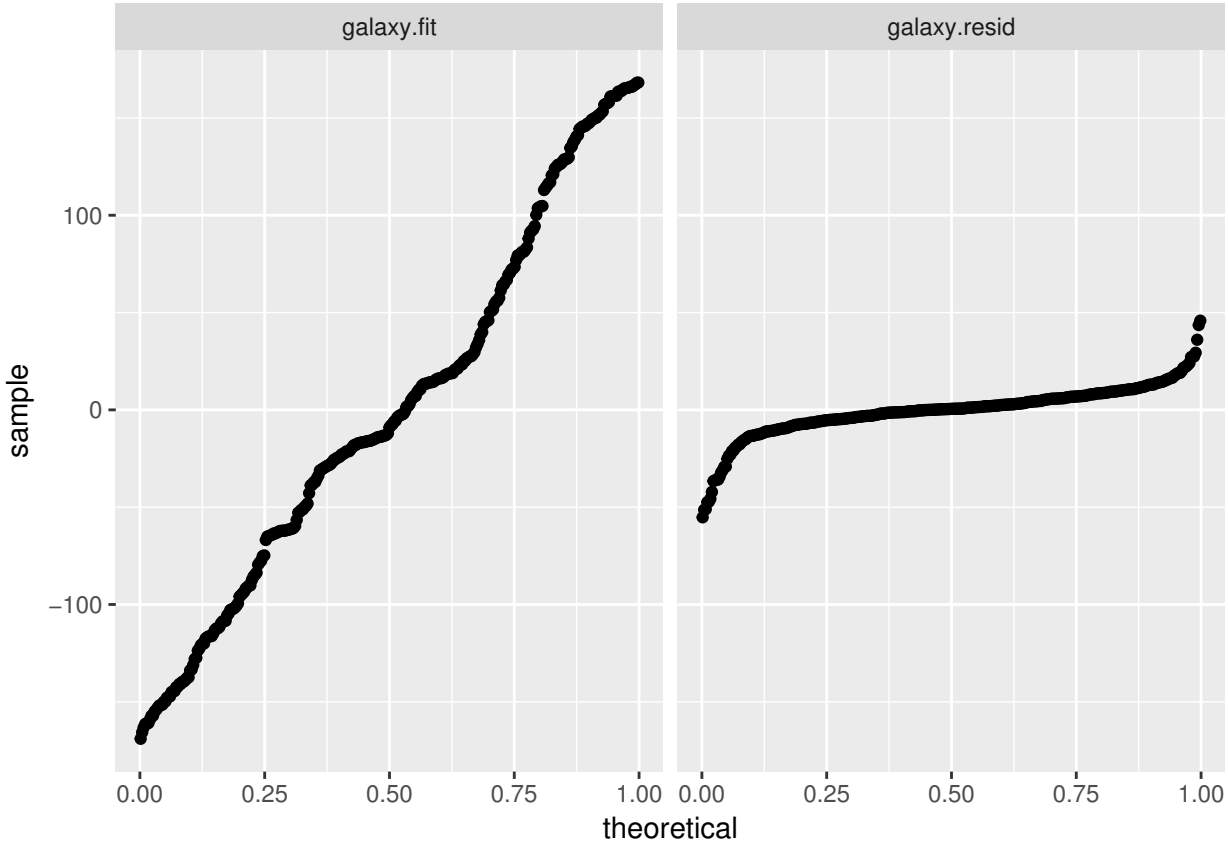
```
ggplot(galaxy.lo.df, aes(sample = .resid)) + stat_qq()
```



The errors aren't normal. The extreme values are too extreme.

Finally, the residual-fit comparsion:

```
galaxy.fit = galaxy.lo.df$.fitted - mean(galaxy.lo.df$.fitted)
galaxy.resid = galaxy.lo.df$.resid
library(tidyr)
galaxy.lo.long = data.frame(galaxy.fit, galaxy.resid) %>% gather(component,
    velocity)
ggplot(galaxy.lo.long, aes(sample = velocity)) + stat_qq(distribution = "qunif") +
    facet_grid(~component)
```



The fitted values are very spread out compared to the residuals. The model explains most of the variation.

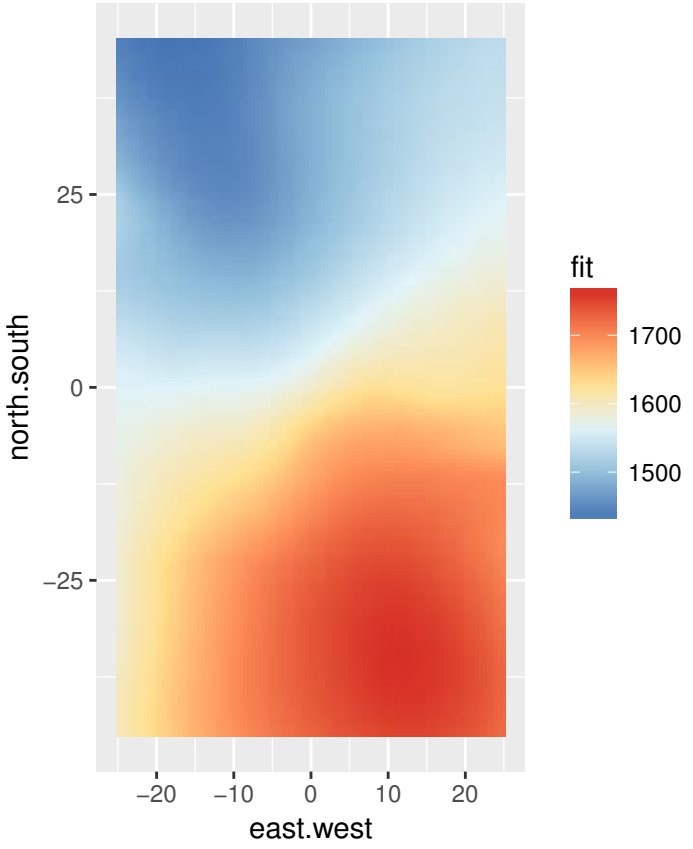
4.3.3 Visualizing the fit: geom_raster and geom_contour

To prepare for plotting, let's make predictions on a rectangular grid, and put the results in a data frame, `galaxy.plot.df`.

```
galaxy.grid = expand.grid(east.west = seq(-25, 25, 0.5), north.south = seq(-45,
    45, 0.5))
galaxy.predict = predict(galaxy.lo, newdata = galaxy.grid)
galaxy.plot.df = data.frame(galaxy.grid, fit = as.vector(galaxy.predict))
```

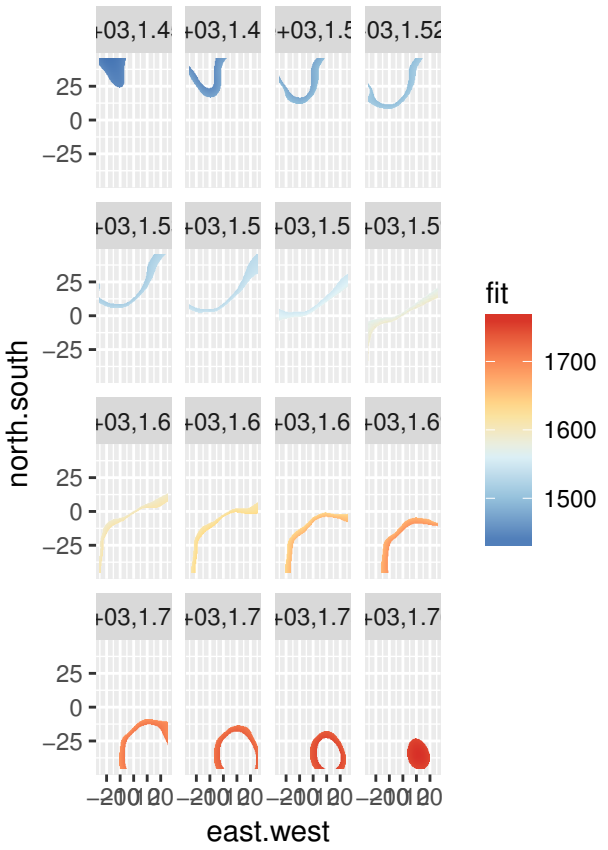
Because of the interaction in the loess, the techinques we've used previously that condition on explanatories aren't going to be sufficient. One option is to use `geom_raster()` to indicate the third variable using color. (`geom_tile()` can be used instead for non-rectangular data.)

```
ggplot(galaxy.plot.df, aes(x = east.west, y = north.south, fill = fit)) + geom_raster() +
    coord_fixed() + scale_fill_distiller(palette = "RdYlBu")
```



It's often quite hard to find a color palette that clearly displays all the features you want to make apparently. One solution is to facet on the fitted value. We cut the above plot into a large number of pieces by fitted value, then draw a grid of plots.

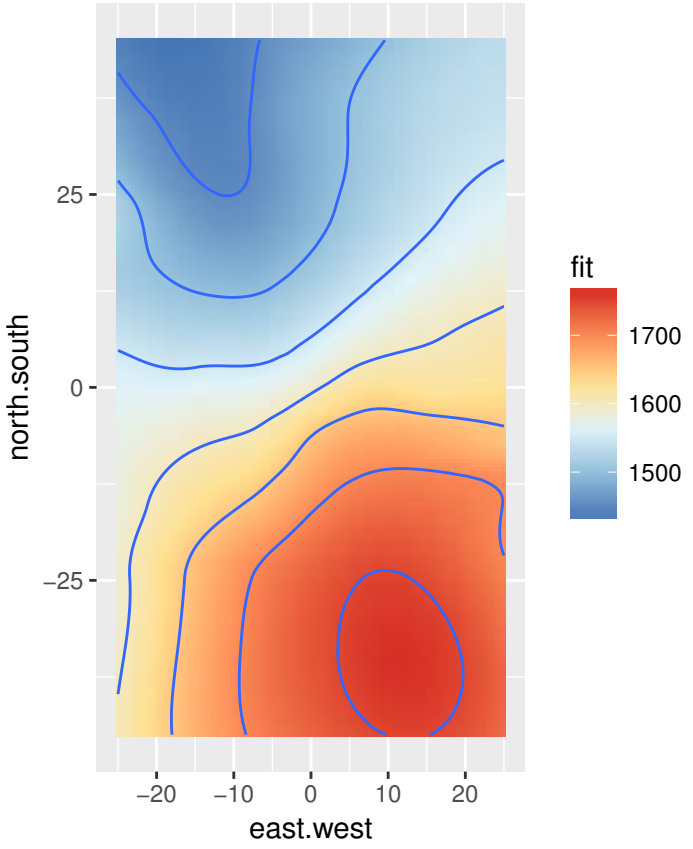
```
ggplot(galaxy.plot.df, aes(x = east.west, y = north.south, fill = fit)) + geom_raster() +
    coord_fixed() + scale_fill_distiller(palette = "RdYlBu") + facet_wrap(~cut_number(fit,
    n = 16), ncol = 4)
```



The basic pattern is clear: the fitted velocities are small in the northeast and generally increase as you move toward the southwest, though the details are complicated.

Alternatively, we can make the shapes clear on the original graph by adding **contours**. A contour is a curve that joins together points that have the same value of the z-variable. `geom_contour()` gives the basic contour plot:

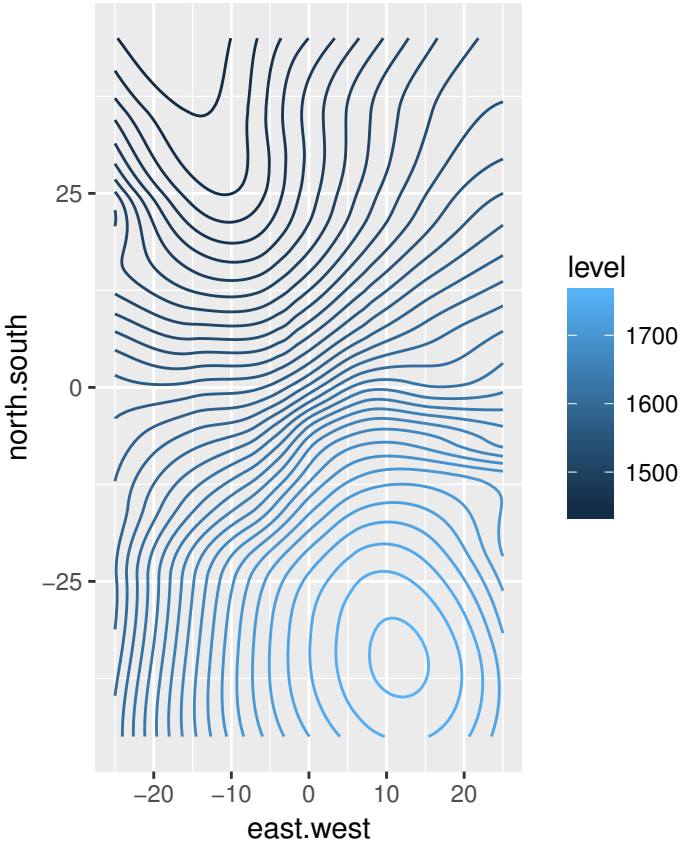
```
ggplot(galaxy.plot.df, aes(x = east.west, y = north.south, z = fit)) + geom_raster(aes(fill = fit)) +
    coord_fixed() + scale_fill_distiller(palette = "RdYlBu") + geom_contour()
```



We can see, for example, a circle of points in the southwest all have the same fitted value. The coloring makes it clear this is a high value (velocity over 1700 km/second.) The northeast has the lowest fitted values.

A final choice is to skip the raster part and go straight to the contours.

```
ggplot(data.frame(galaxy.grid, fit = as.vector(galaxy.predict)), aes(x = east.west,
    y = north.south, z = fit)) + geom_contour(binwidth = 10, aes(color = ..level..)) +
    coord_fixed()
```

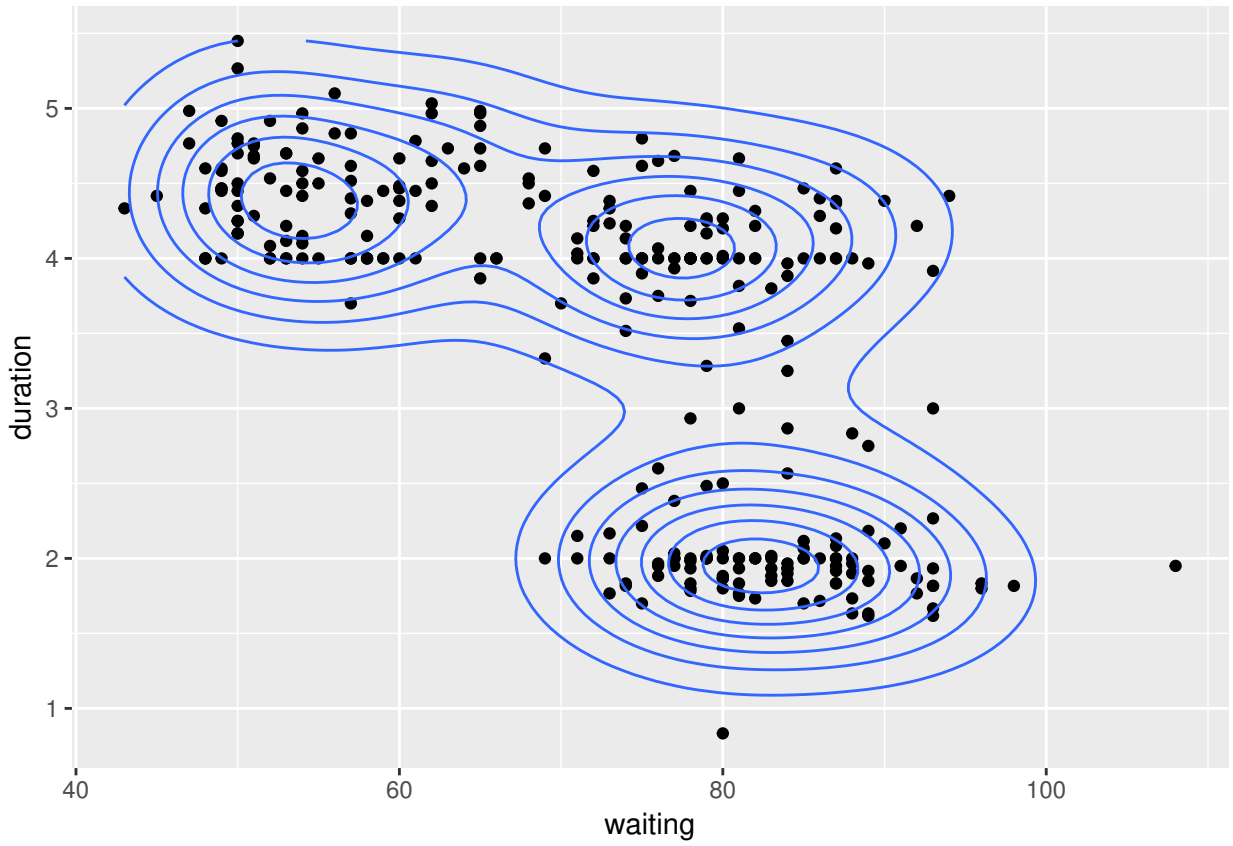


If relying on contours alone, they need to be fairly dense to get the details of the distribution across. Note that labeling the contours by putting numbers directly on the graph is very hard to do within `ggplot` (if you must, use `contourplot` in the `lattice` library instead.)

4.3.4 Contour plots for bivariate densities

The other major use for contour plots is to display density estimates for bivariate data. The ggplot function `stat_density_2d` does bivariate density estimation. We apply it to the data set `geyser`, which contains bivariate data on the waiting time until eruption and the duration of the subsequent eruption (both in minutes) for 299 successive eruptions of Old Faithful in 1985:

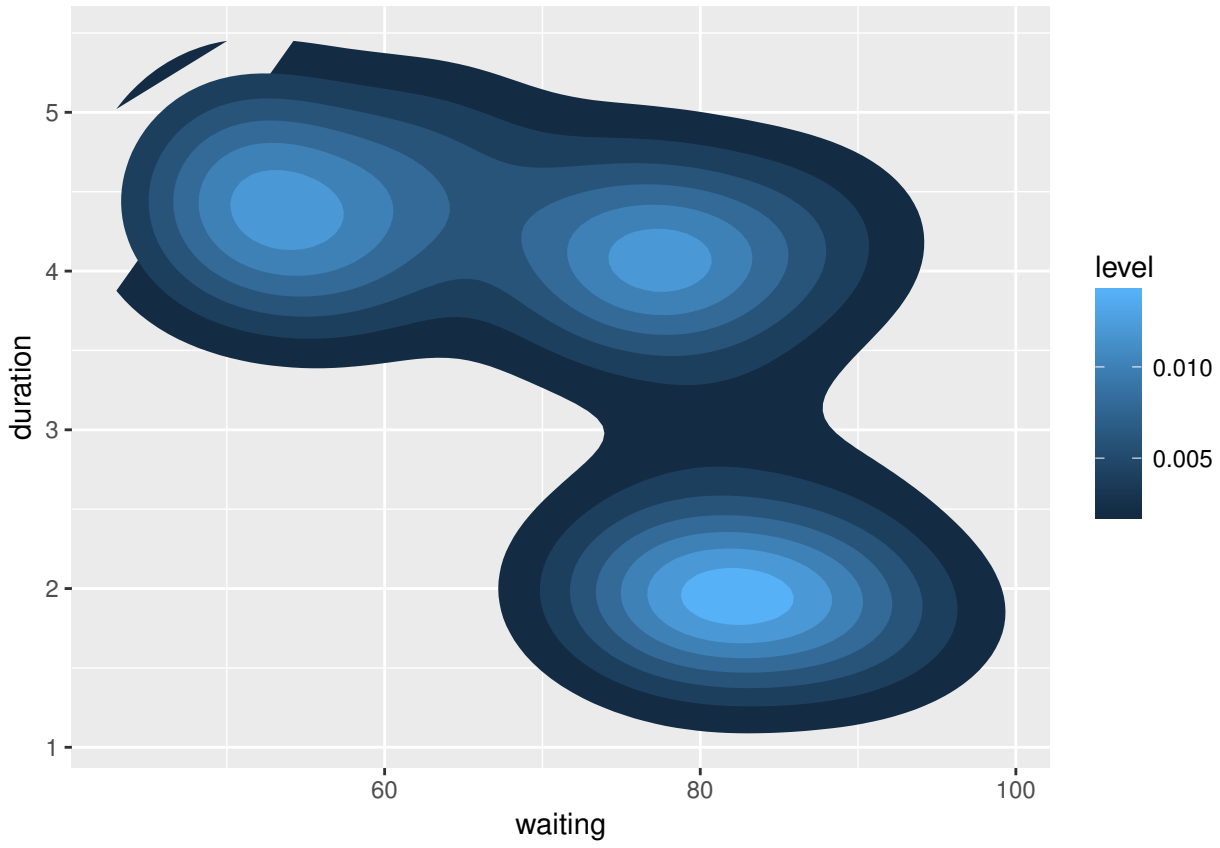
```
library(MASS)
ggplot(geyser, aes(x = waiting, y = duration)) + geom_point() + stat_density_2d()
```



The data appears trimodal: we get short wait/long eruption, long wait/long eruption, and long wait/short eruption. We don't get short wait/short eruption.

We could also look at this by coloring in between the contours to get a solid surface:

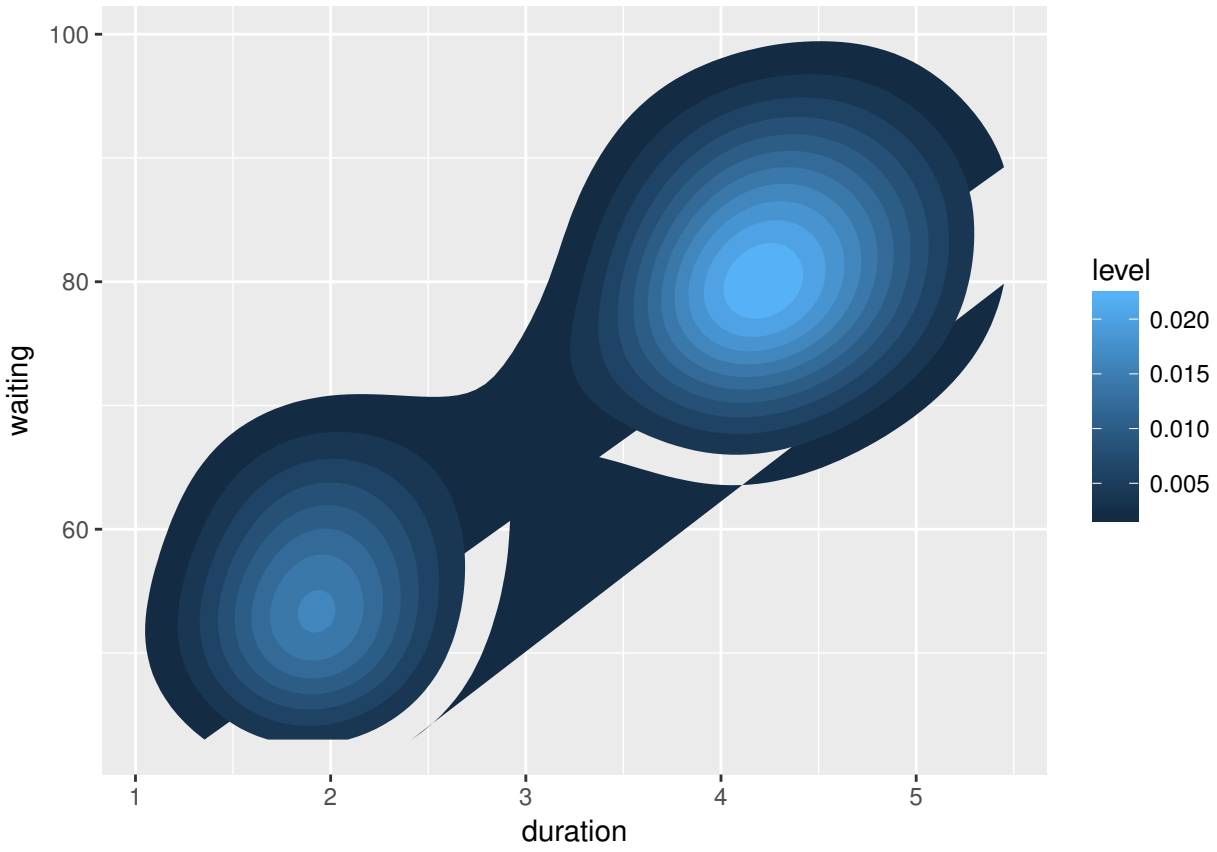
```
ggplot(geyser, aes(x = waiting, y = duration)) + stat_density_2d(aes(fill = ..level..),
    geom = "polygon")
```



The general message is the same, but now we have quantitative information on the density. The highest peak is for long wait/long eruption, at around 0.015 (per minute per minute.) You can explore the 2D density further by studying things like the conditional distribution of eruption length given waiting time, etc.

If you're sitting around at Yellowstone, you might be more interested in the joint density of the eruption duration and the waiting time to the *next* eruption.

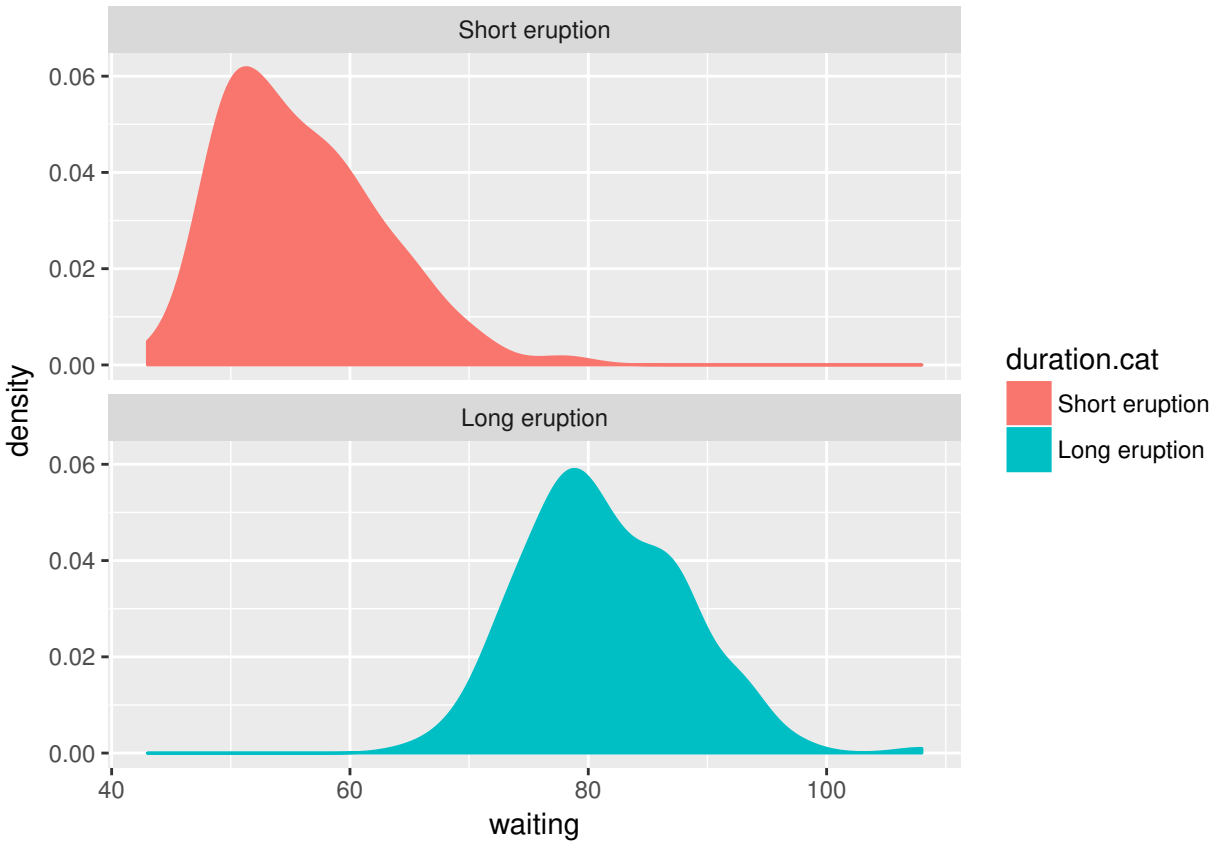
```
n = nrow(geyser)
next.geyser = data.frame(duration = geyser$duration[-n], waiting = geyser$waiting[-1])
ggplot(next.geyser, aes(x = duration, y = waiting)) + stat_density_2d(aes(fill = ..level..),
    geom = "polygon")
```



This time the data looks bimodal: short eruption followed by short wait, or long eruption followed by long wait. The long/long peak is substantially higher.

Finally, we want to study the **conditional** distributions: given an eruption duration, how long will we have to wait for the next eruption? For simplicity, we only split `duration` into two categories.

```
duration.cat = rep(NA, nrow(next.geyser))
duration.cat[next.geyser$duration <= 3] = "Short eruption"
duration.cat[next.geyser$duration > 3] = "Long eruption"
duration.cat = factor(duration.cat, levels = c("Short eruption", "Long eruption"))
ggplot(data.frame(next.geyser, duration.cat), aes(x = waiting, fill = duration.cat,
    color = duration.cat)) + stat_density() + facet_wrap(~duration.cat, ncol = 1)
```



The modal waiting time after a short eruption is a bit over 50 minutes, while the modal waiting time after a long eruption is a bit under 80 minutes. Note that this graph does not attempt to show that long eruptions are more common than short eruptions.

4.4 Truly 3D plots: Wireframes

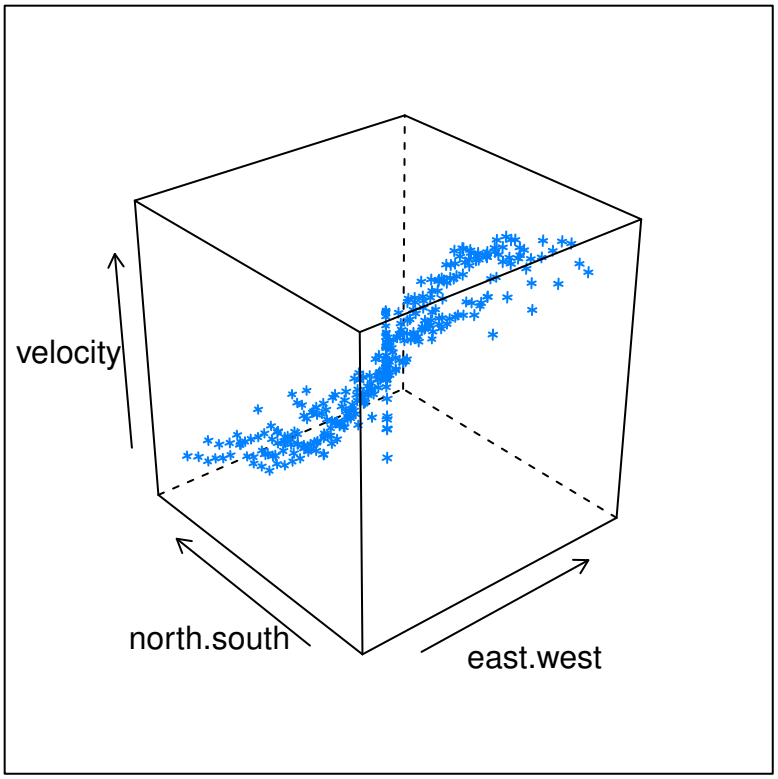
READ: Cleveland pp. 249–267.

It's the lecturer's opinion that 3D plots are kind of overrated, but some people like them so here they are. `ggplot()` doesn't really do 3D plots, so we go to the `lattice` library for this section.

4.4.1 Galaxy data

Let's return to the loess fit to the galaxy data from last time. Recall that north and west are positive; putting north on top, the slow parts of the galaxy were in the top left (northeast) and the fast parts of the galaxy were in the bottom right (southwest.) Firstly, we can use the `cloud()` function to plot the raw data in three dimensions.

```
load("lattice.RData")
library(lattice)
cloud(velocity ~ east.west * north.south, data = galaxy)
```



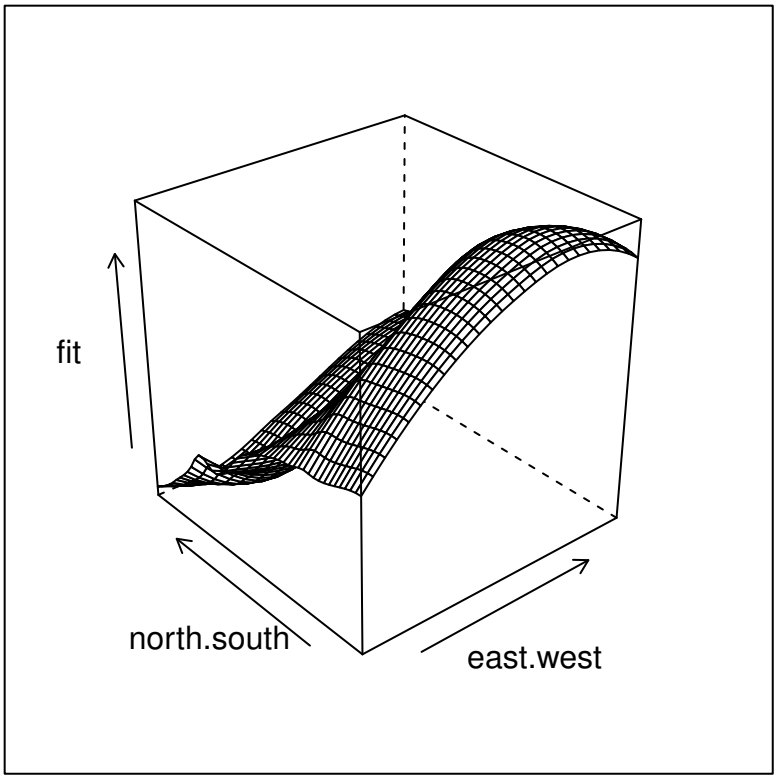
The arrows point toward more positive values of the variables: north, west, and high velocity respectively. (If you wanted numeric scales on the axes instead of just arrows, you could use `scatterplot3d()` in the library of the same name, but I find it near-impossible to accurately read numbers off 3D plots so I don't bother including them.) Now that we know what to look for, we see the data is consistent with low velocities in the northeast and high velocities in the southwest. This would be possible but somewhat hard to see if we didn't know what we were looking for.

Now re-fit the loess model we chose last time, and make predictions on a grid.

```
galaxy.lo = loess(velocity ~ east.west * north.south, data = galaxy, span = 0.25,
    family = "symmetric", normalize = FALSE)
galaxy.wf.grid = expand.grid(east.west = seq(-25, 25, 2), north.south = seq(-45,
    45, 2))
galaxy.wf.predict = predict(galaxy.lo, newdata = galaxy.wf.grid)
galaxy.wf.df = data.frame(galaxy.wf.grid, fit = as.vector(galaxy.wf.predict))
```

To draw a truly 3D plot, we use `wireframe()`.

```
wireframe(fit ~ east.west * north.south, data = galaxy.wf.df)
```



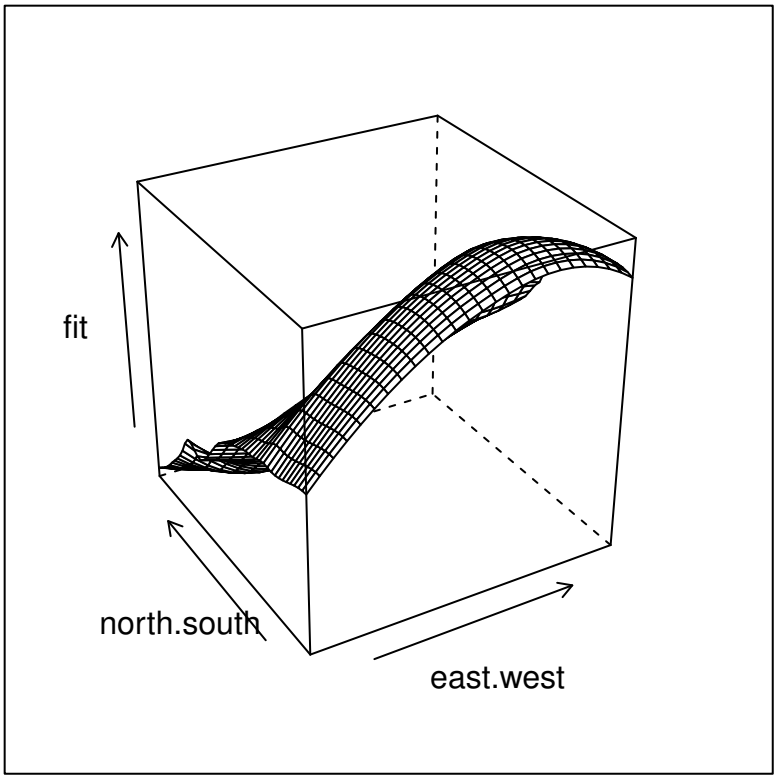
This is quite a bit clearer than the `cloud()` plot. In particular, the curvature of the fitted surface is apparent. The estimated velocity goes down a bit in the extreme southwest corner.

With a wireframe plot, it's rare that one angle lets you see all relevant details of the fit. If you're just exploring the data on your own, the `rotate.wireframe()` function in the `TeachingDemos` library can be fun to play with (if a bit buggy.)

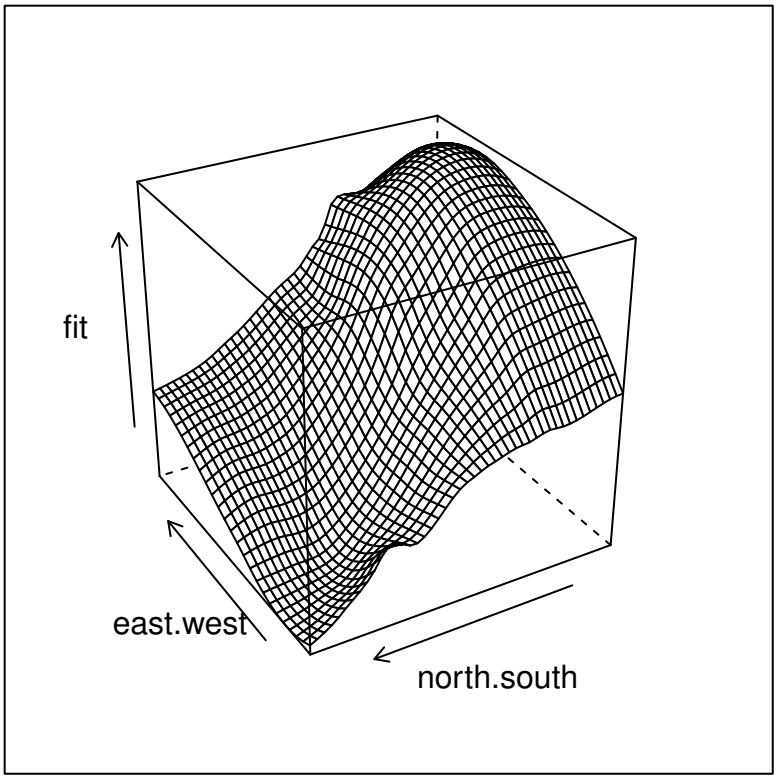
```
# install.packages('TeachingDemos')
library(TeachingDemos)
rotate.wireframe(fit ~ east.west * north.south, data = galaxy.wf.df)
```

Of course, this doesn't fly if you're trying to prepare a document. You could build a Shiny app but that probably isn't going to be worth the effort. Instead, pick a few different angles, and show the wireframe from those angles.

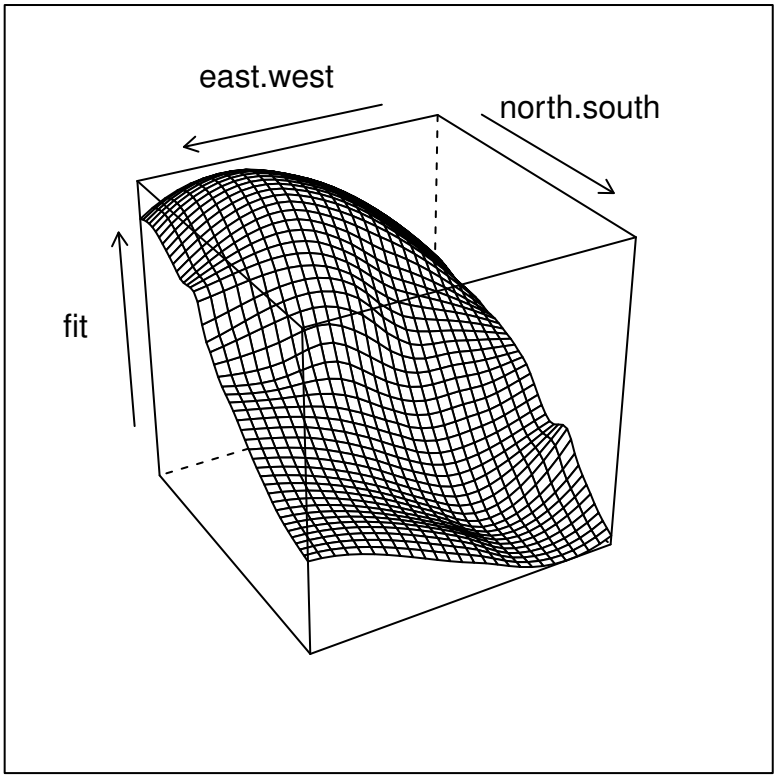
```
wireframe(fit ~ east.west * north.south, data = galaxy.wf.df, screen = list(z = 30,
    x = -60, y = 0))
```



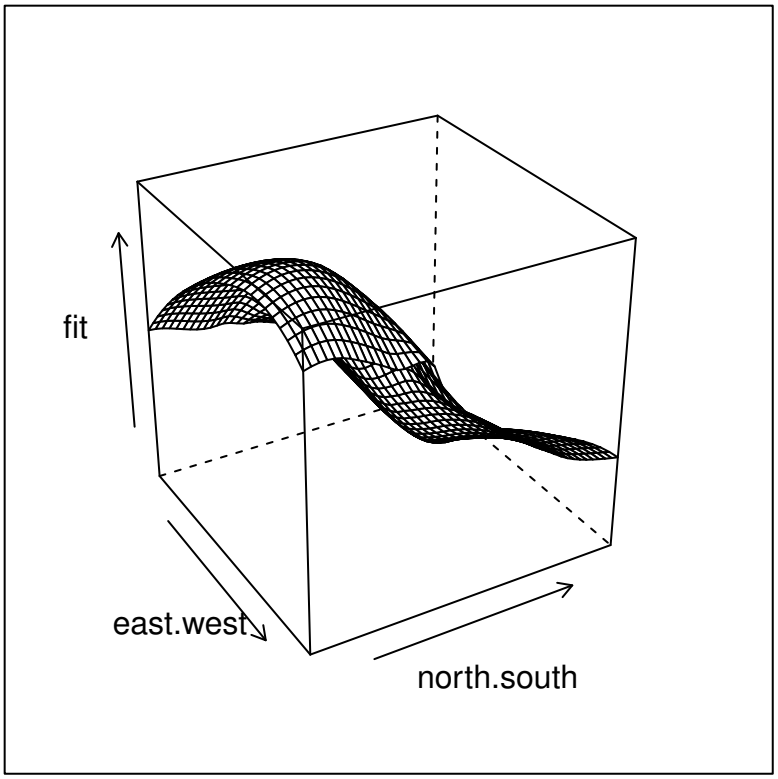

```
wireframe(fit ~ east.west * north.south, data = galaxy.wf.df, screen = list(z = 120,
    x = -60, y = 0))
```



```
wireframe(fit ~ east.west * north.south, data = galaxy.wf.df, screen = list(z = 210,
    x = -60, y = 0))
```



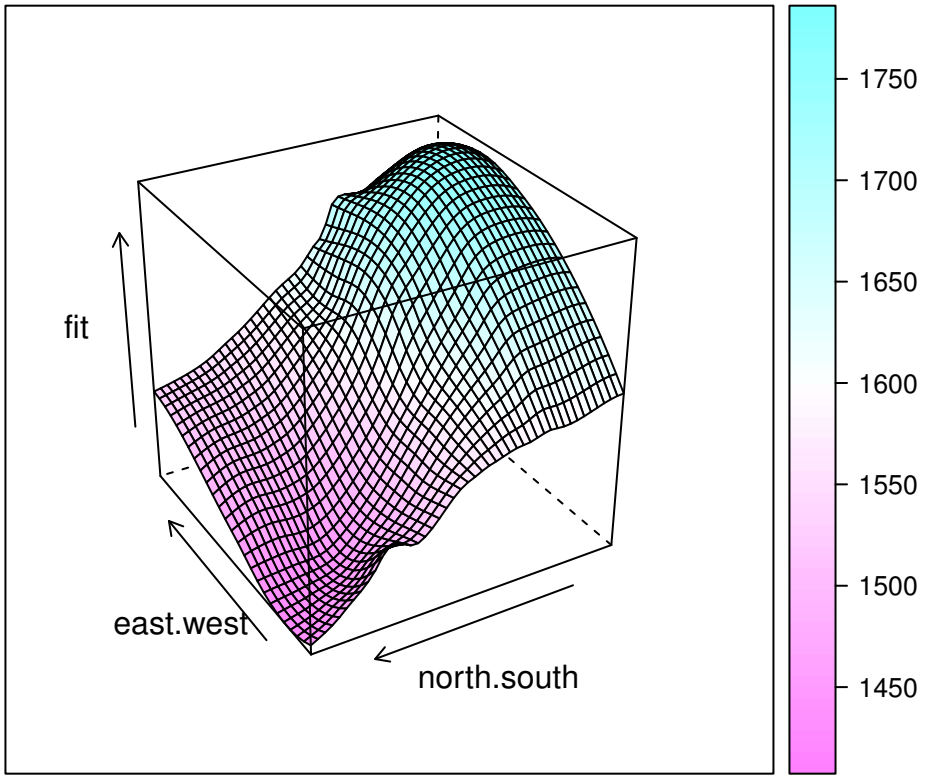

```
wireframe(fit ~ east.west * north.south, data = galaxy.wf.df, screen = list(z = 300,
    x = -60, y = 0))
```



Here we kept the x and y angles fixed, while rotating z by 90 degrees each time. The effect is to "spin" the surface around while keeping the "camera" fixed.

Finally, we can "fill in" in the wireframe using the `drape` argument.

```
wireframe(fit ~ east.west * north.south, data = galaxy.wf.df, screen = list(z = 120,
    x = -60, y = 0), drape = TRUE)
```

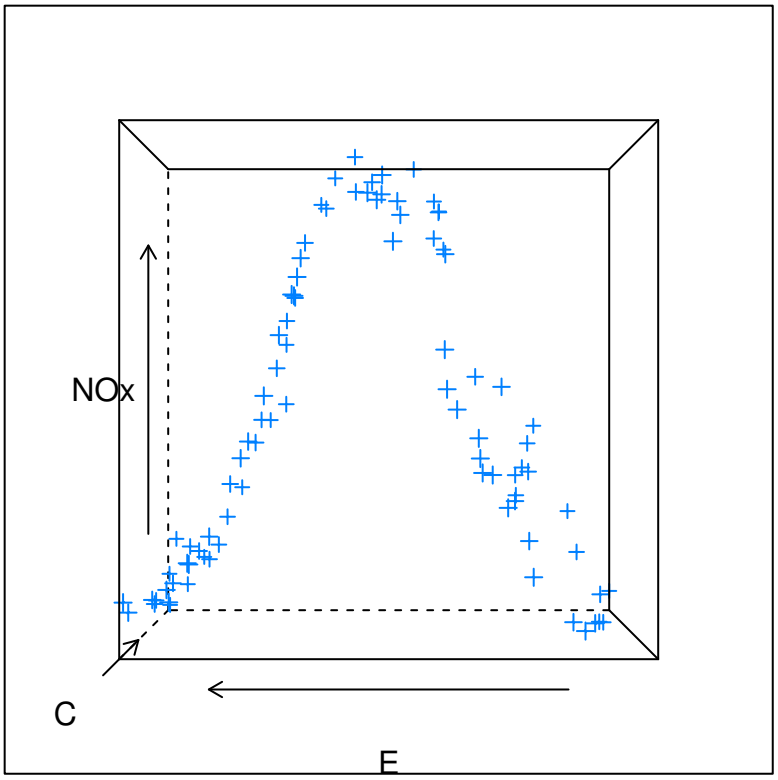


You can play around with the color scheme using the `col.regions` argument. All this is getting dangerously close to chartjunk territory, though.

4.4.2 Ethanol data

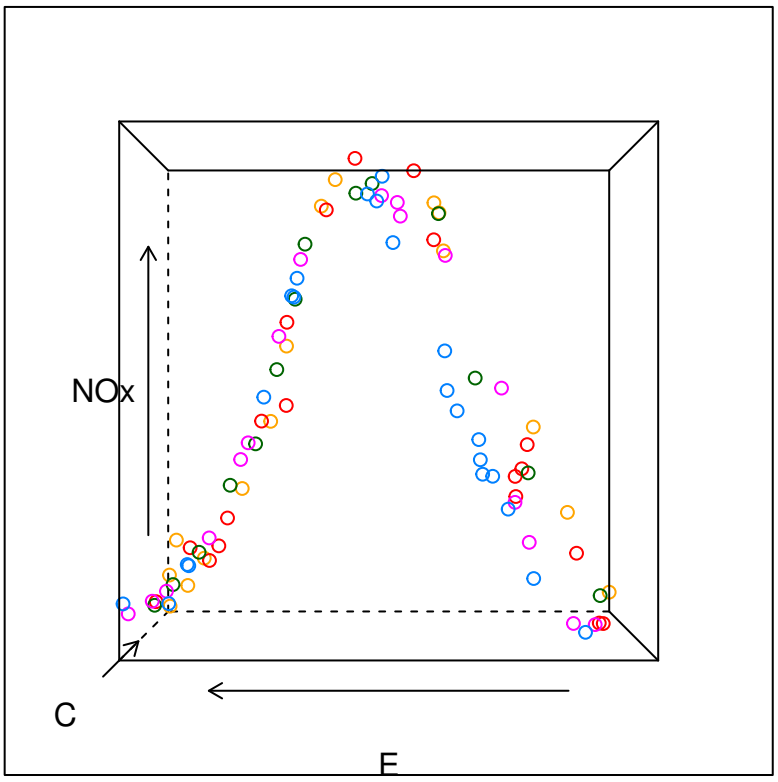
We now return to the ethanol engine data. Recall that the curves relating NOX to equivalence ratio (E) were close in shape but not identical for all five values of compression ratio (C). If we cloud plot the data and look at it "front-on," i.e. with C going into the screen, we get something that's almost a 2D scatterplot of NOx against E.

```
cloud(NOx ~ C * E, data = ethanol, screen = list(z = 90, x = -90, y = 0))
```



We can try coloring the plot by levels of C:

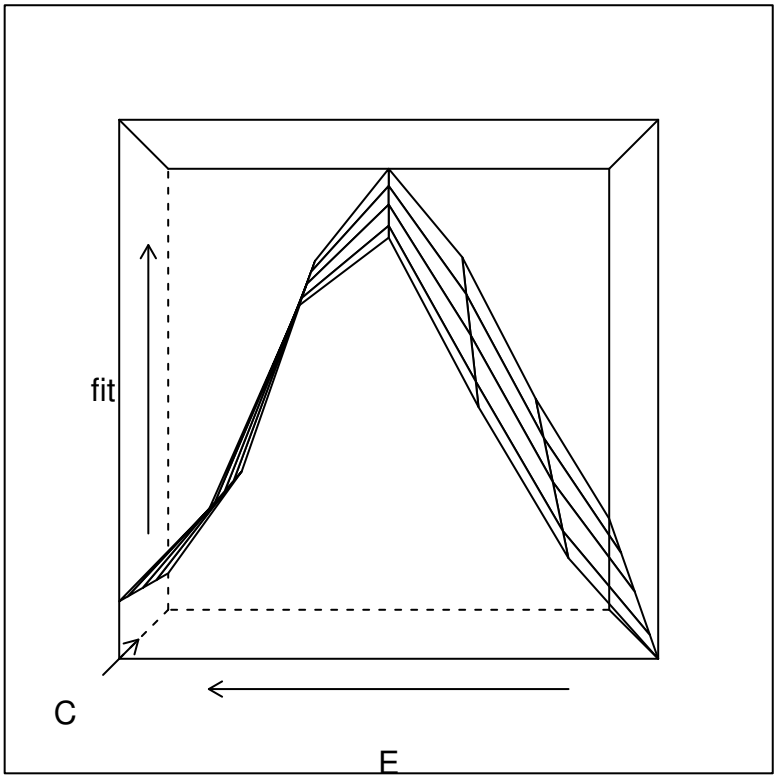
```
cloud(NOx ~ C * E, data = ethanol, screen = list(z = 90, x = -90, y = 0), groups = C)
```



Now we can see the blue points (for example) are lower than the others on the right hand side, while they're

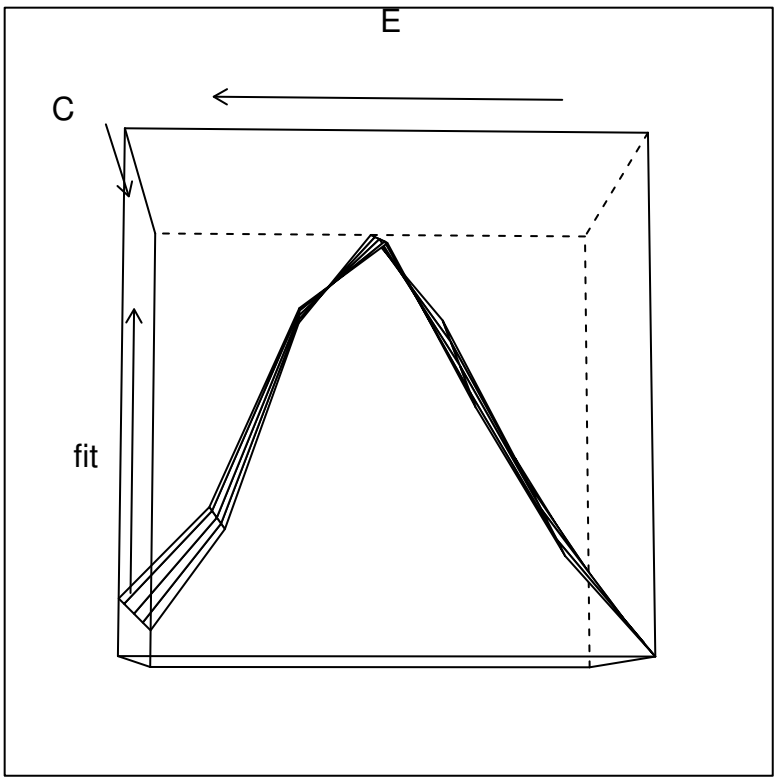
similar to the other colors on the left. (Again, this is easier to see in retrospect after having studied the data.) We could try to make this plot look nicer by fiddling with the color scheme or adding a legend, but we'll be better off instead looking at the loess surface we fitted to the data. Re-fit the model and look at it head-on:

```
ethanol.lo = loess(NOx ~ C * E, data = ethanol, span = 1/3, parametric = "C",
    drop.square = "C", family = "symmetric")
ethanol.grid = expand.grid(C = c(7.5, 9, 12, 15, 18), E = seq(0.6, 1.2, 0.1))
ethanol.predict = predict(ethanol.lo, newdata = ethanol.grid)
ethanol.df = data.frame(ethanol.grid, fit = as.vector(ethanol.predict))
wireframe(fit ~ C * E, data = ethanol.df, screen = list(z = 90, x = -90, y = 0))
```



We see that the height of the surface (the fitted value) gets taller as C increases. Note, however, that changing the angles by just a few degrees make this seem to disappear.

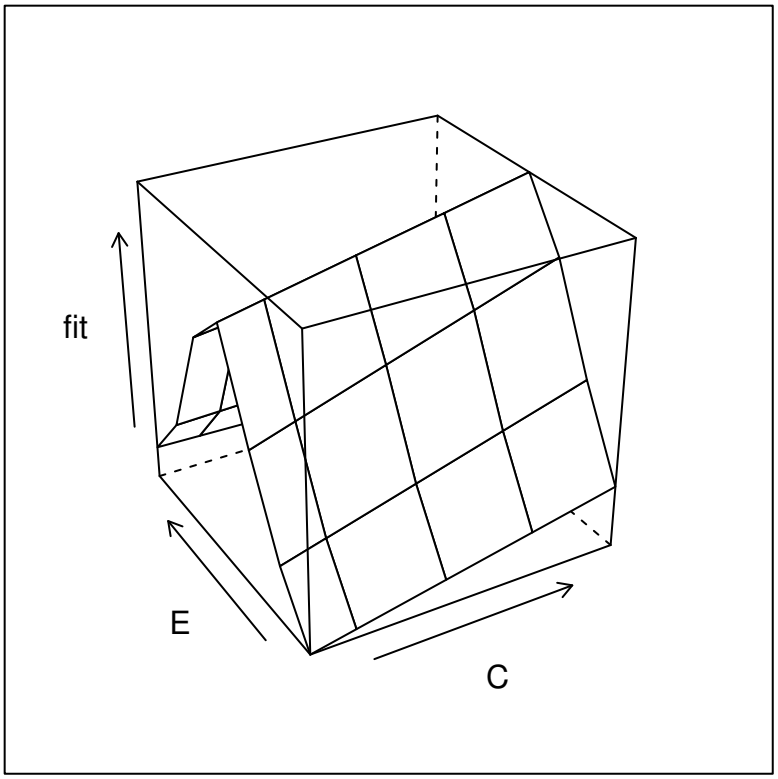
```
wireframe(fit ~ C * E, data = ethanol.df, screen = list(z = 92, x = -97, y = 0))
```



The issue is *foreshortening* – things that are further away look smaller. To better understand foreshortening, look at a bunch of Italian Renaissance paintings. For our purposes, it's enough to remember that the choice of angle is important, so make sure to try out a few.

For this fit, looking somewhat from the side makes the change in height clearer:

```
wireframe(fit ~ C * E, data = ethanol.df, screen = list(z = 30, x = -60, y = 0))
```

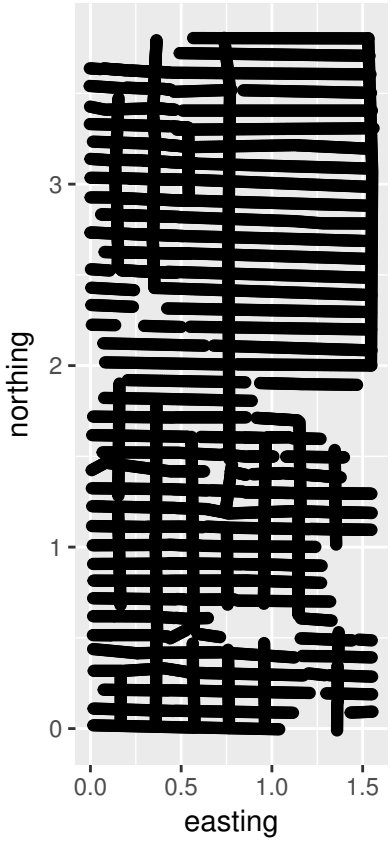


Now it's obvious that higher C generally means somewhat higher NOx. Again, you need more than one angle to see all of what's going on.

4.4.3 Soil

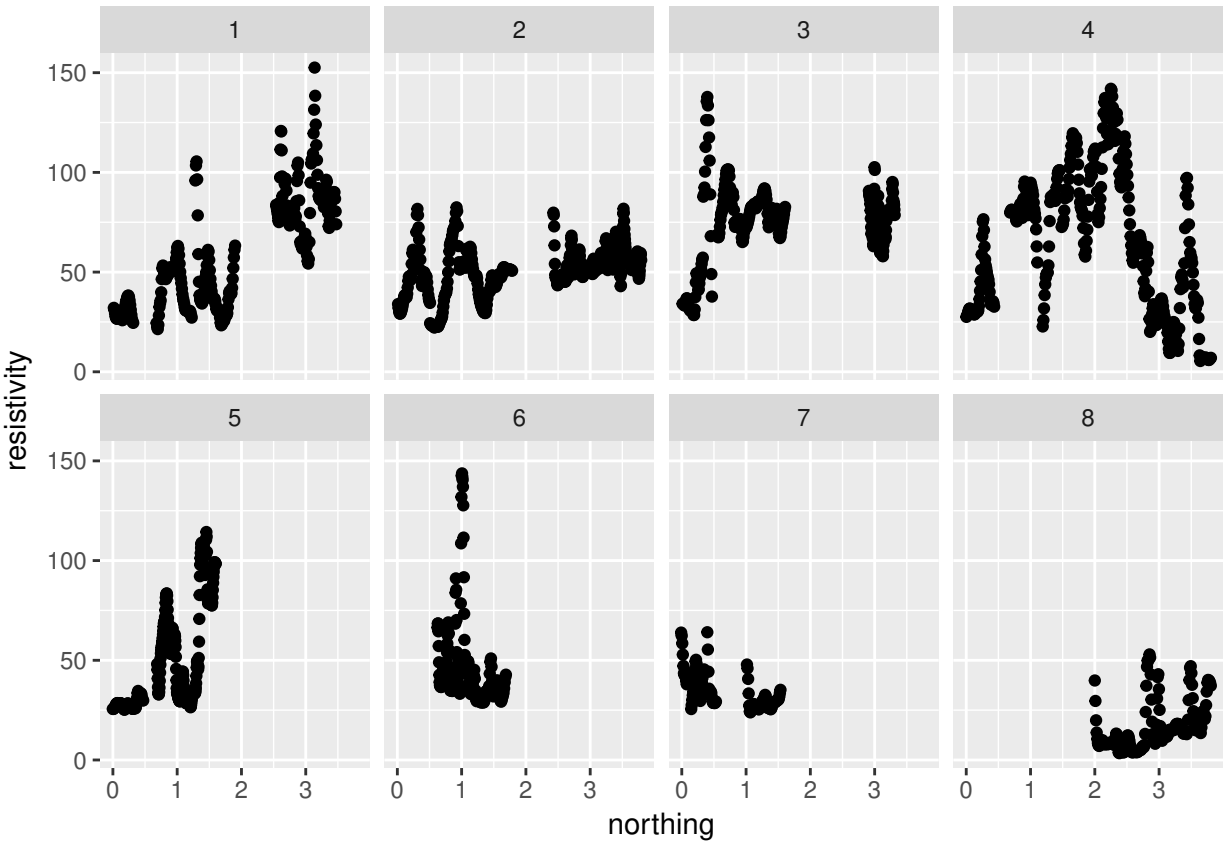
The Cleveland data set `soil` contains measurements on resistivity (in ohm cm) in a field in Western Australia. The locations are given by "easting" and "northing" coordinates, which just measure distance from an origin in kilometers.

```
library(ggplot2)
ggplot(soil, aes(x = easting, y = northing)) + geom_point() + coord_fixed()
```



We see the locations occur along a number of "tracks", which are recorded in the `track` variable. Some tracks are north-south while others are east-west; this is recorded in the variable `is.ns`. To see how the resistivity varies by coordinate, we'll first subset by track direction, then facet by track.

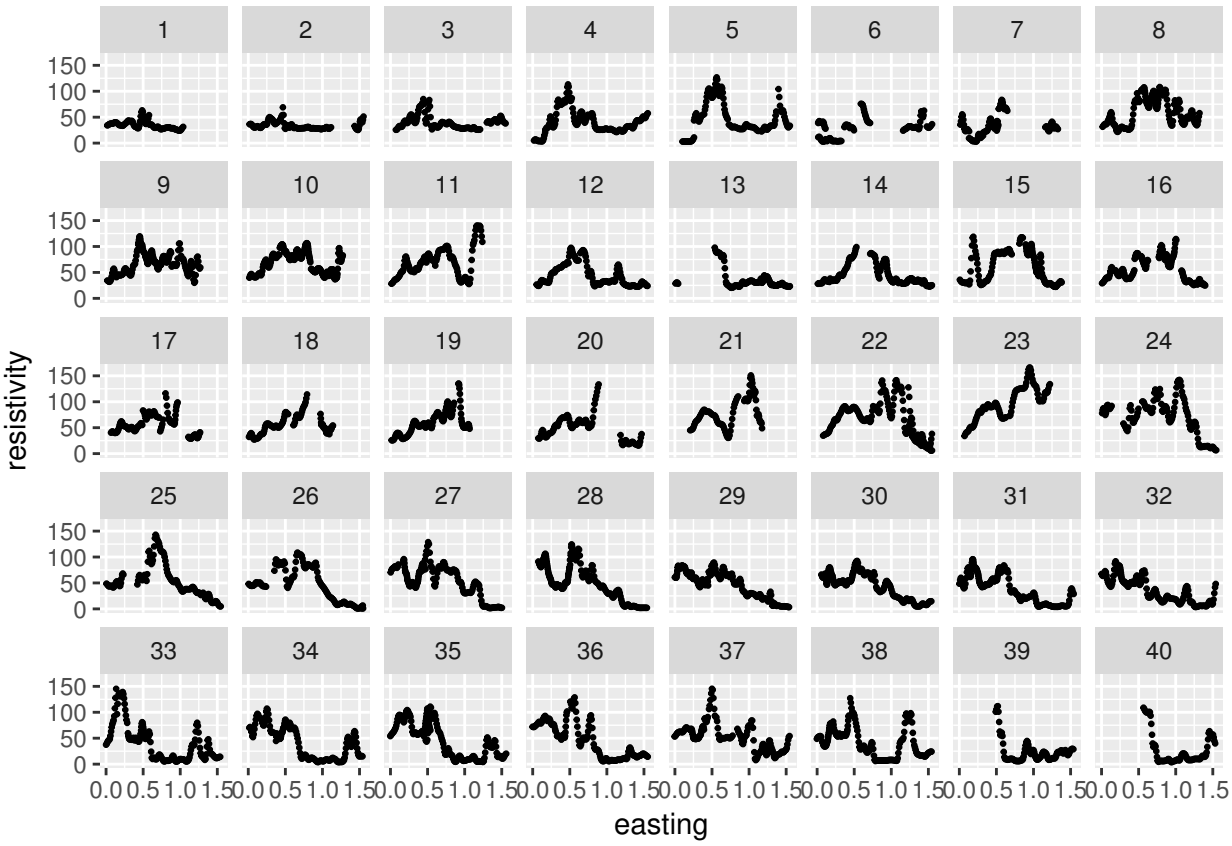
```
ggplot(subset(soil, is.ns == TRUE), aes(x = northing, y = resistivity)) + geom_point() +
    facet_wrap(~track, ncol = 4)
```



Looking at the north-south tracks, the patterns aren't very consistent. There are a bunch of spikes occurring at seemingly random locations.

Now try the east-west tracks:

```
ggplot(subset(soil, is.ns == FALSE), aes(x = easting, y = resistivity)) + geom_point(size = 0.5) +
    facet_wrap(~track, ncol = 8)
```



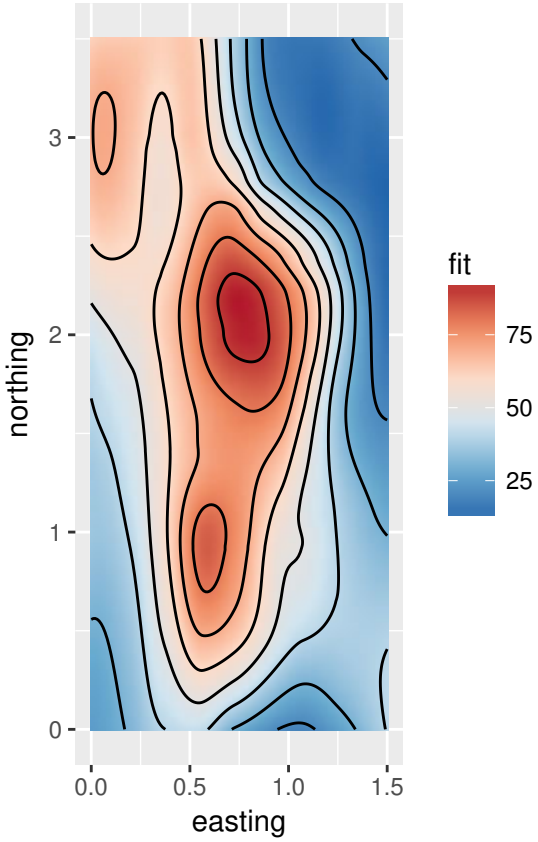
We see that there's usually a downward trend in resistivity as the easting coordinate increases. However, this doesn't show up in all the plots.

We fit a loess model to the data, predicting resistivity from the easting and northing coordinates (with an interaction.)

```
soil.lo = loess(resistivity ~ easting * northing, span = 0.25, data = soil)
```

Now predict on a grid and plot the fit using color and contours.

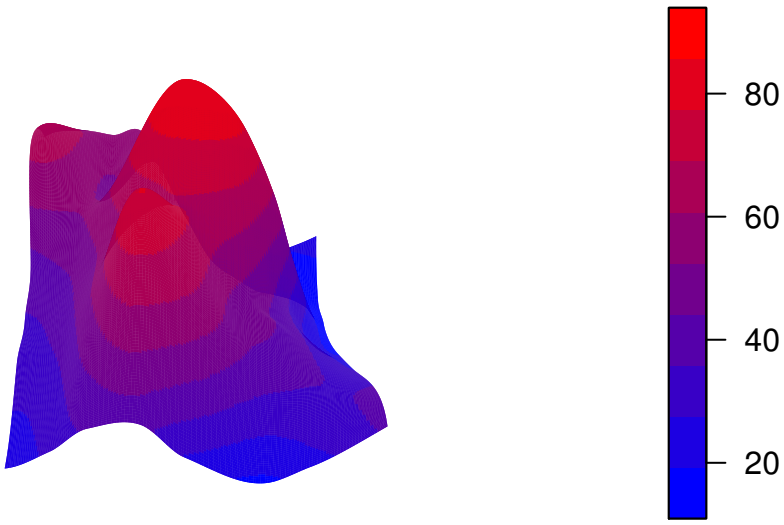
```
soil.grid = expand.grid(easting = seq(0, 1.5, 0.01), northing = seq(0, 3.5,
    0.01))
soil.predict = predict(soil.lo, newdata = soil.grid)
soil.df = data.frame(soil.grid, fit = as.vector(soil.predict))
ggplot(soil.df, aes(x = easting, y = northing, z = fit, fill = fit)) + geom_raster() +
    geom_contour(binwidth = 10, color = "black") + scale_fill_distiller(palette = "RdBu") +
    coord_fixed()
```



There's a clear peak around $(0.75, 2.1)$, along with a smaller peak near $(0.6, 0.9)$.

The complexity of the surface means that a wireframe plot isn't well-suited to displaying the fit. You can use `surf3D()` in the `plot3D` library:

```
# install.packages('plot3D')
library(plot3D)
east.grid = seq(0, 1.5, 0.01)
north.grid = seq(0, 3.5, 0.01)
mesh.grid = mesh(east.grid, north.grid)
fit.grid = matrix(soil.predict, nrow = length(east.grid))
surf3D(mesh.grid$x, mesh.grid$y, fit.grid, theta = 0, col = ramp.col(col = c("blue",
    "red"), n = 10))
```



It's unclear whether this is of any real data analytic value. It looks cool, I guess.