Chapter 6

Multiway data

6.1 Two-way data, and also maps

READ: Cleveland pp. 302–319.

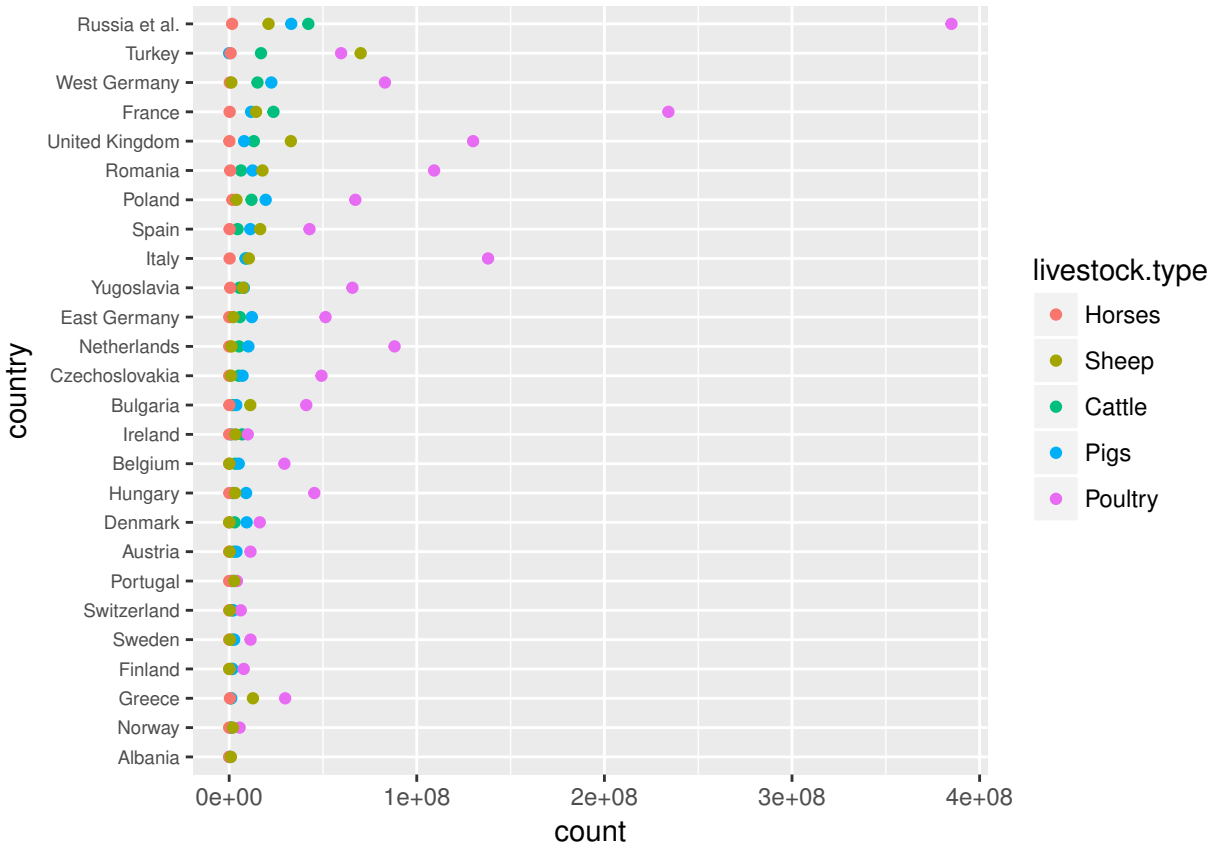
6.1.1 Livestock

Load the data and basics:

```
load("lattice.RData")
library(ggplot2)
library(MASS)
library(tidyr)
```

The data frame `livestock` contains the number of horses, sheep, cattle, pigs, and poultry in 26 European countries in the 1980s. (You can tell it's from the Eighties because some of the countries don't exist anymore.) The data is in long form, with variable names `livestock.type`, `country`, and `count`. Let's first show a dot plot that gives the big picture.

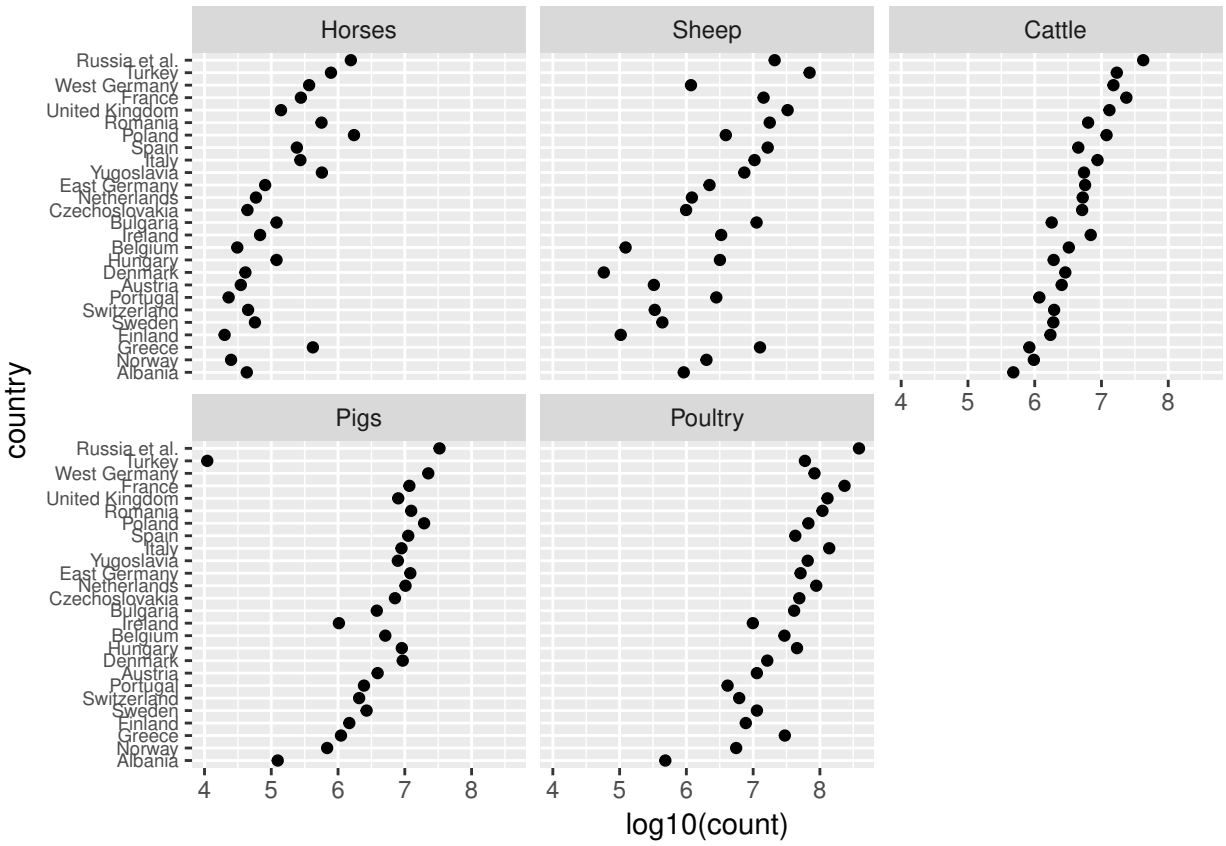
```
ggplot(livestock, aes(x = count, y = country, color = livestock.type)) + geom_point() +
    theme(axis.text.y = element_text(size = 7))
```



The data is extremely skewed, because some countries have a lot of chickens. From hereon out we'll do a $\log_{10}$ transformation of count.

We'll condition on animal and country in turn. First, facet wrap on `livestock.type`:

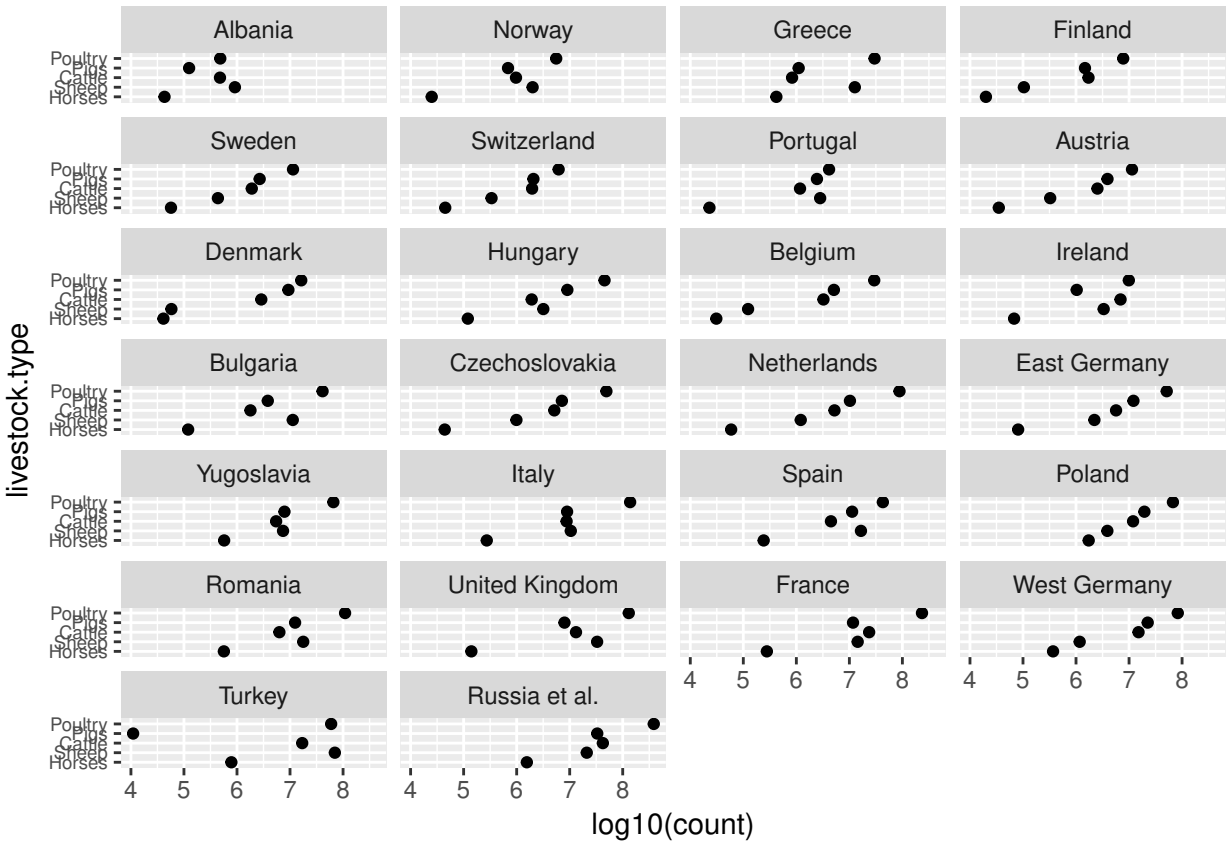
```
ggplot(livestock, aes(x = log10(count), y = country)) + geom_point() + facet_wrap(~livestock.type,
    ncol = 3) + theme(axis.text.y = element_text(size = 7))
```



Note that Turkey has a huge outliers for "pigs" due to being a predominantly Muslim country. The other historically Muslim country in the data set is Albania, which also has a low value for pigs. However, at this stage it's hard to tell if Albania's low pig count is unusual or just because the country is small.

Next, facet wrap on country.

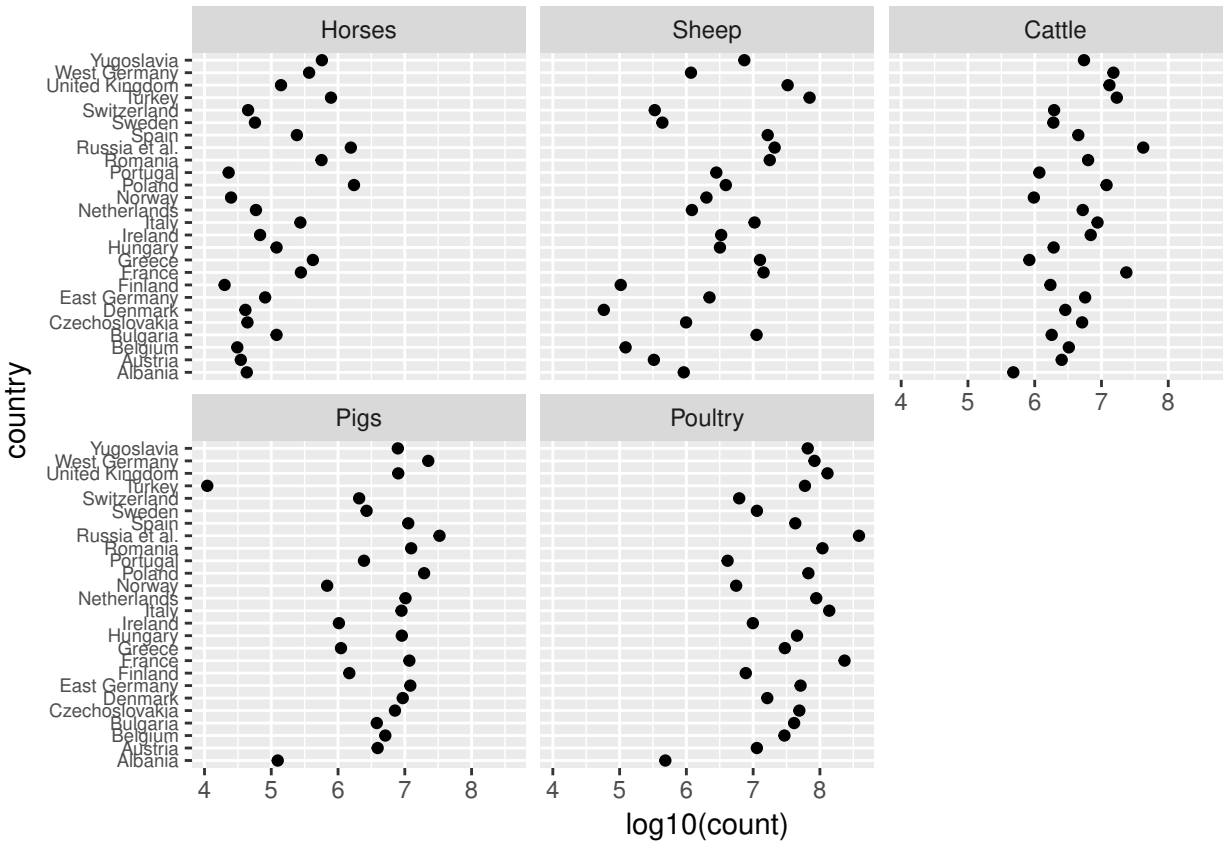
```
ggplot(livestock, aes(x = log10(count), y = livestock.type)) + geom_point() +
    facet_wrap(~country, ncol = 4) + theme(axis.text.y = element_text(size = 7))
```



The majority of the countries have the lowest count for horses and the higher count for poultry. The other three animals are not in a consistent order.

So far, we've just plotted the data in the order that it came in (which was sorted by country median.) To change the order, we specify the order of levels in the underlying factor. For example, we can put the country levels in alphabetical order:

```
country.alpha = factor(livestock$country, sort(levels(livestock$country)))
livestock.alpha = data.frame(livestock.type = livestock$livestock.type, country = country.alpha,
    count = livestock$count)
ggplot(livestock.alpha, aes(x = log10(count), y = country)) + geom_point() +
    facet_wrap(~livestock.type, ncol = 3) + theme(axis.text.y = element_text(size = 7))
```



Alphabetical order isn't very useful though. Usually increasing or decreasing numerical order will make the patterns clearer.

6.1.2 Fitting a model

There are lots of ways of varying complexity to fit a model to multiway data. One basic idea is to break the data down into additive pieces: in our case, the logged count would be an animal effect plus a country effect plus a residual. (Of course, we've taken logs, so it's multiplicative on the original scale.)

The simplest way to fit an additive model is to just use `lm()`, i.e. do an ANOVA. As usual, this fits using least squares:

```
livestock.lm = lm(log10(count) ~ livestock.type + country, data = livestock)
dummy.coef(livestock.lm)$livestock.type
```

```
##      Horses       Sheep      Cattle        Pigs     Poultry
## -1.34532352 -0.02910687  0.17887036  0.14358357  1.05197646
```

However, least squares is (also as usual) potentially misleading when there are outliers. Here, Turkey will throw off least squares pretty badly.

An outlier-resistant alternative is to use `rlm()` with bisquare. Because Turkey is pretty far out there, we need to allow a large number of iterations for the algorithm to converge.

```
livestock.rlm = rlm(log10(count) ~ livestock.type + country, psi = psi.bisquare,
    maxit = 50, data = livestock)
dummy.coef(livestock.rlm)$livestock.type
```

```
##      Horses       Sheep      Cattle        Pigs     Poultry
## -1.43805090  0.07393821  0.11930031  0.23322721  1.01158517
```

The coefficients are now estimating median effects instead of mean effects. The coefficients for horses, sheep, and pigs change a lot.

An alternative is to use **median polish**:

```
livestock.wide = livestock %>% spread(livestock.type, count)
row.names(livestock.wide) = livestock.wide$country
livestock.wide = livestock.wide[, -1]
livestock.mp = medpolish(log10(livestock.wide))
```

```
## 1: 37.85129
## 2: 35.64557
## Final: 35.42729
```

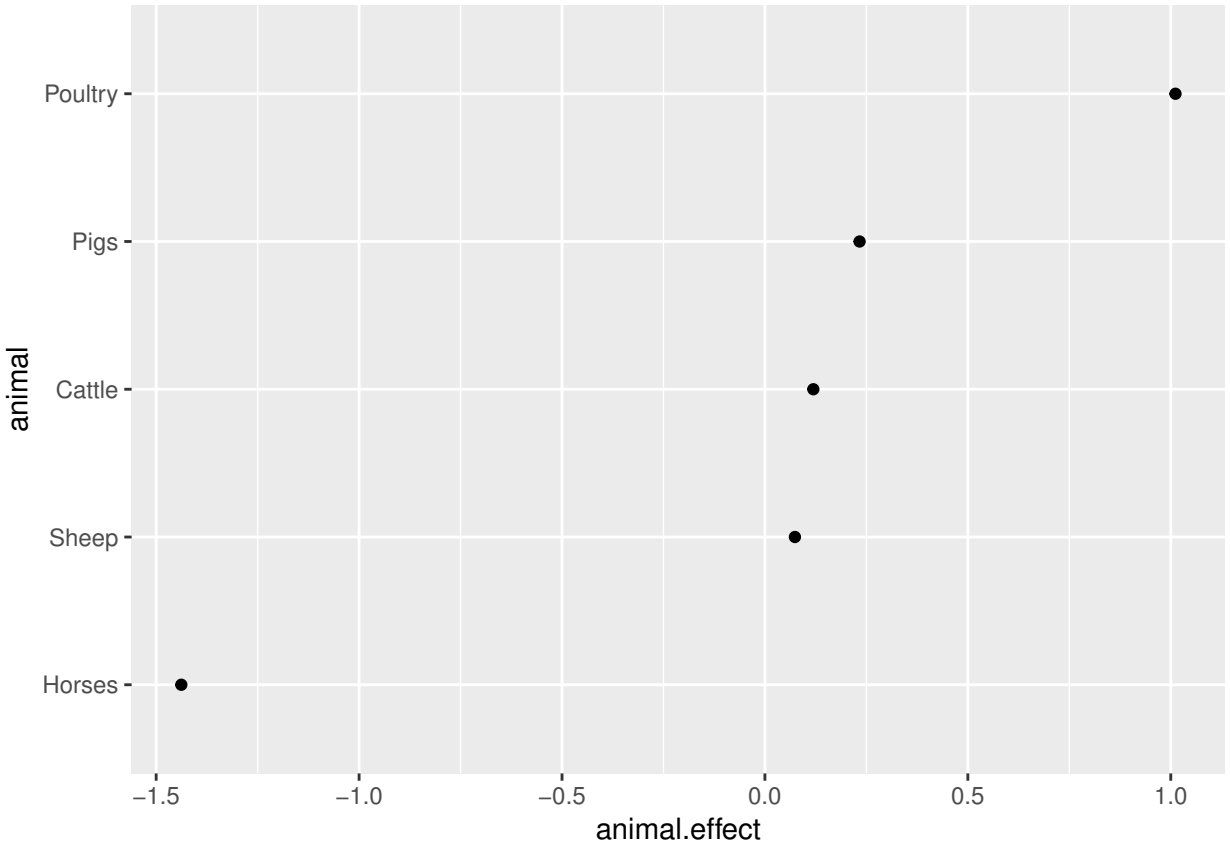
```
livestock.mp$col
```

```
##      Horses       Sheep      Cattle        Pigs     Poultry
## -1.62480255 -0.15773973  0.00000000  0.04181838  0.86008387
```

Note that here the median is set to zero rather than the sum. Anyway, I don't really understand median polish. Jake is the expert on that, so ask him.

Let's stick with the `rlm` for now. We wish to plot the effect sizes using dot plots. We first do the animal effects:

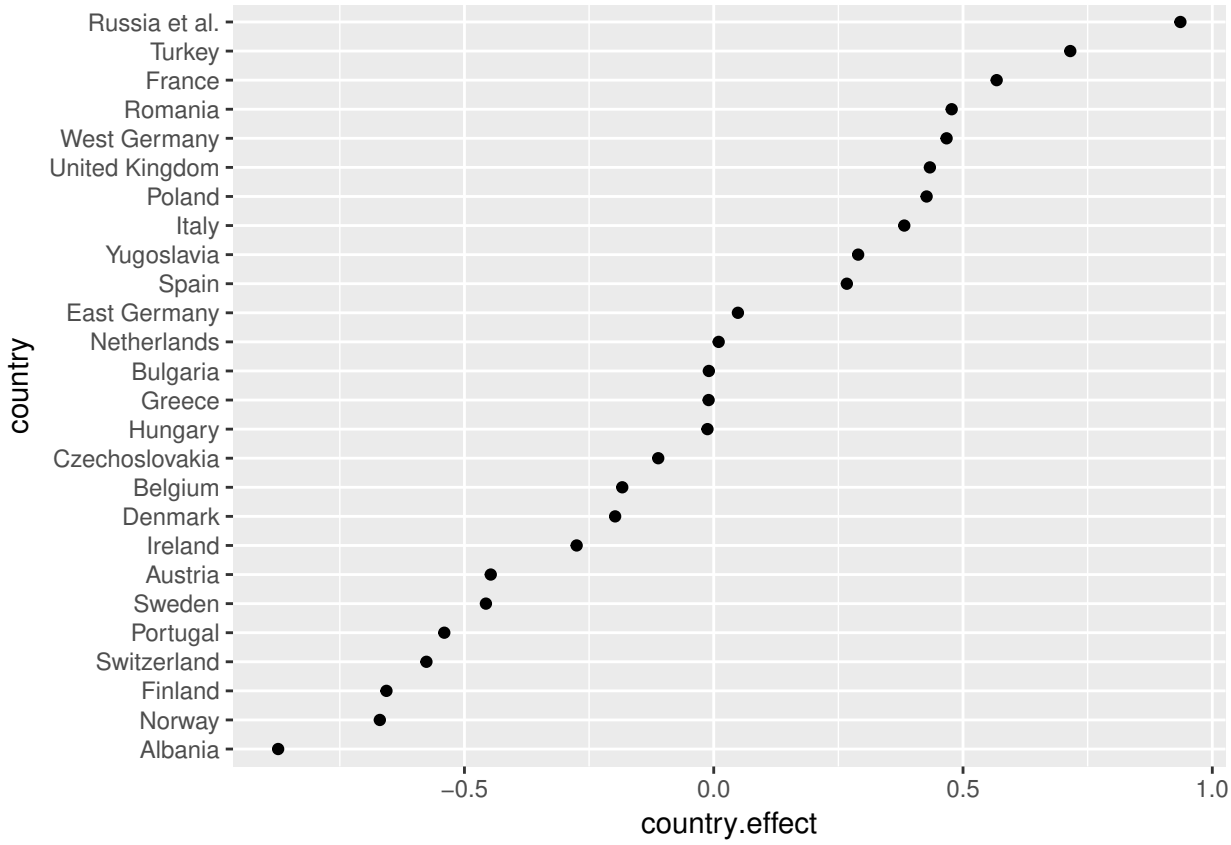
```
animal.effect = sort(dummy.coef(livestock.rlm)$livestock.type)
animal = factor(names(animal.effect), levels = names(animal.effect))
animal.effect.df = data.frame(animal, animal.effect)
ggplot(animal.effect.df, aes(x = animal.effect, y = animal)) + geom_point()
```



As we saw visually, poultry is the highest and horses are the lowest. The other three are all pretty close.

Now do country effects:

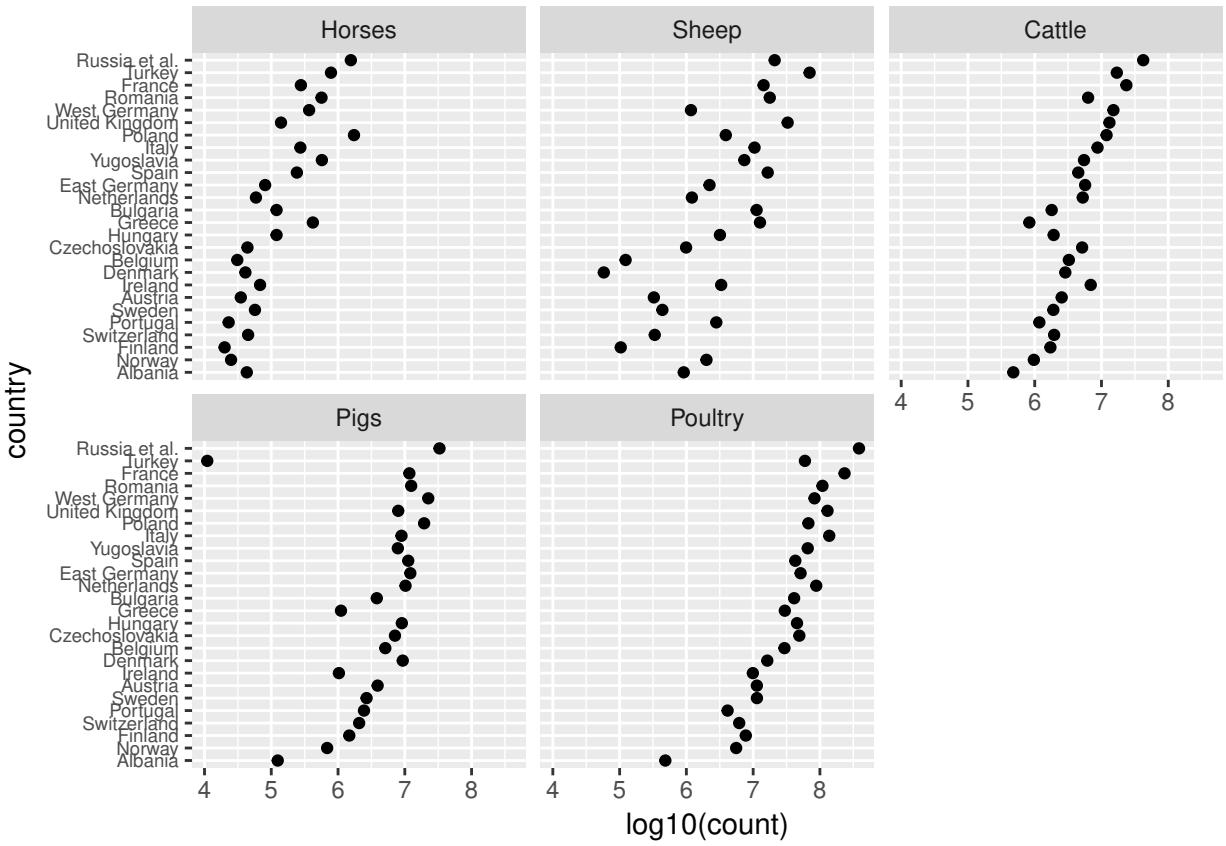
```
country.effect = sort(dummy.coef(livestock.rlm)$country)
country = factor(names(country.effect), levels = names(country.effect))
country.effect.df = data.frame(country, country.effect)
ggplot(country.effect.df, aes(x = country.effect, y = country)) + geom_point()
```



The former U.S.S.R. has the biggest numbers, while Albania has the smallest. It would be interesting to see whether the effects relate more closely to geographic area or population size.

We can now redraw our original faceted plots, reordering the countries by their effect size (the animals are already in the right order.) First, condition on animal:

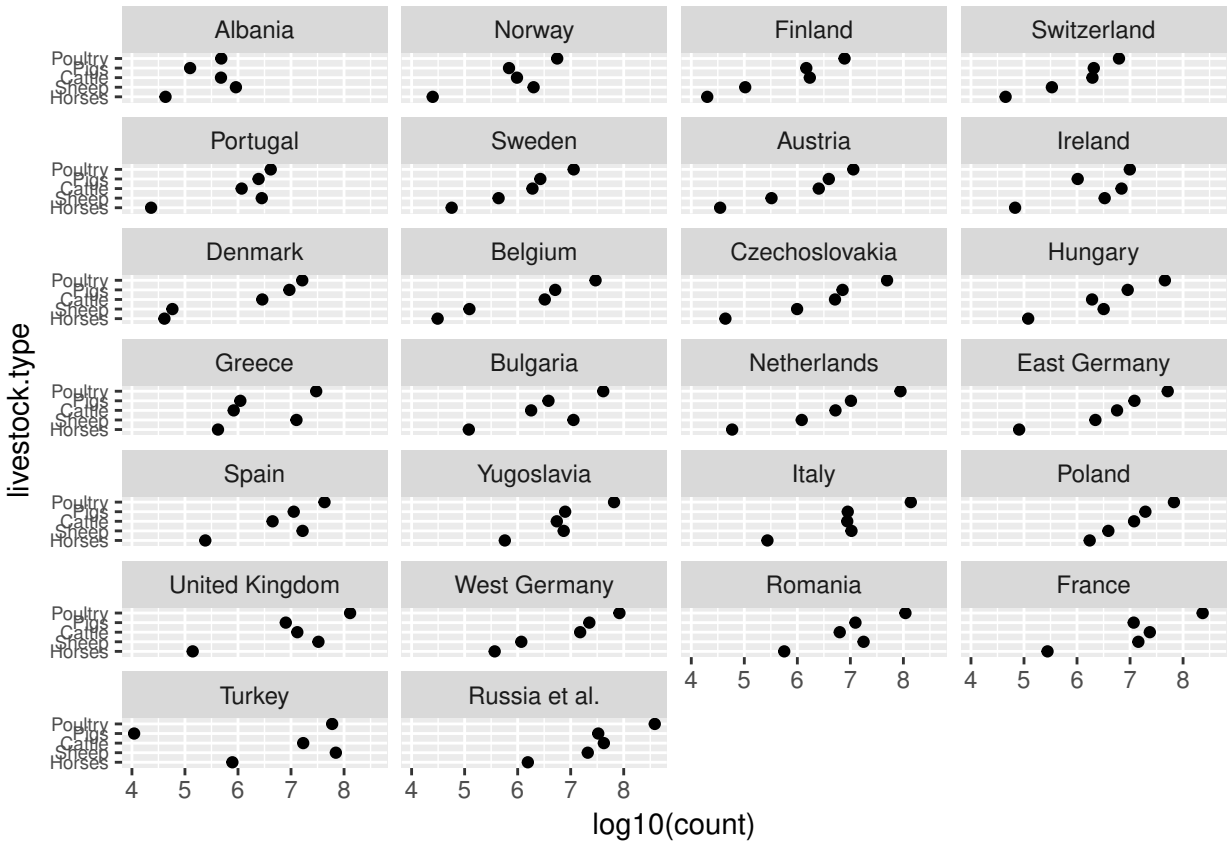
```
livestock.rlm.df = data.frame(livestock, .fitted = fitted.values(livestock.rlm),
    .resid = residuals(livestock.rlm))
livestock.rlm.df$country = factor(livestock$country, levels = names(country.effect))
livestock.rlm.df$livestock.type = factor(livestock$livestock.type, levels = names(animal.effect))
ggplot(livestock.rlm.df, aes(x = log10(count), y = country)) + geom_point() +
    facet_wrap(~livestock.type, ncol = 3) + theme(axis.text.y = element_text(size = 7))
```



The order is slightly different now; France, for example, is up from fourth to third. (Note that our order is different from in Cleveland's book; I think he didn't do enough iterations.)

Now condition on country. Only the order of the panels changes:

```
ggplot(livestock.rlm.df, aes(x = log10(count), y = livestock.type)) + geom_point() +
    facet_wrap(~country, ncol = 4) + theme(axis.text.y = element_text(size = 7))
```

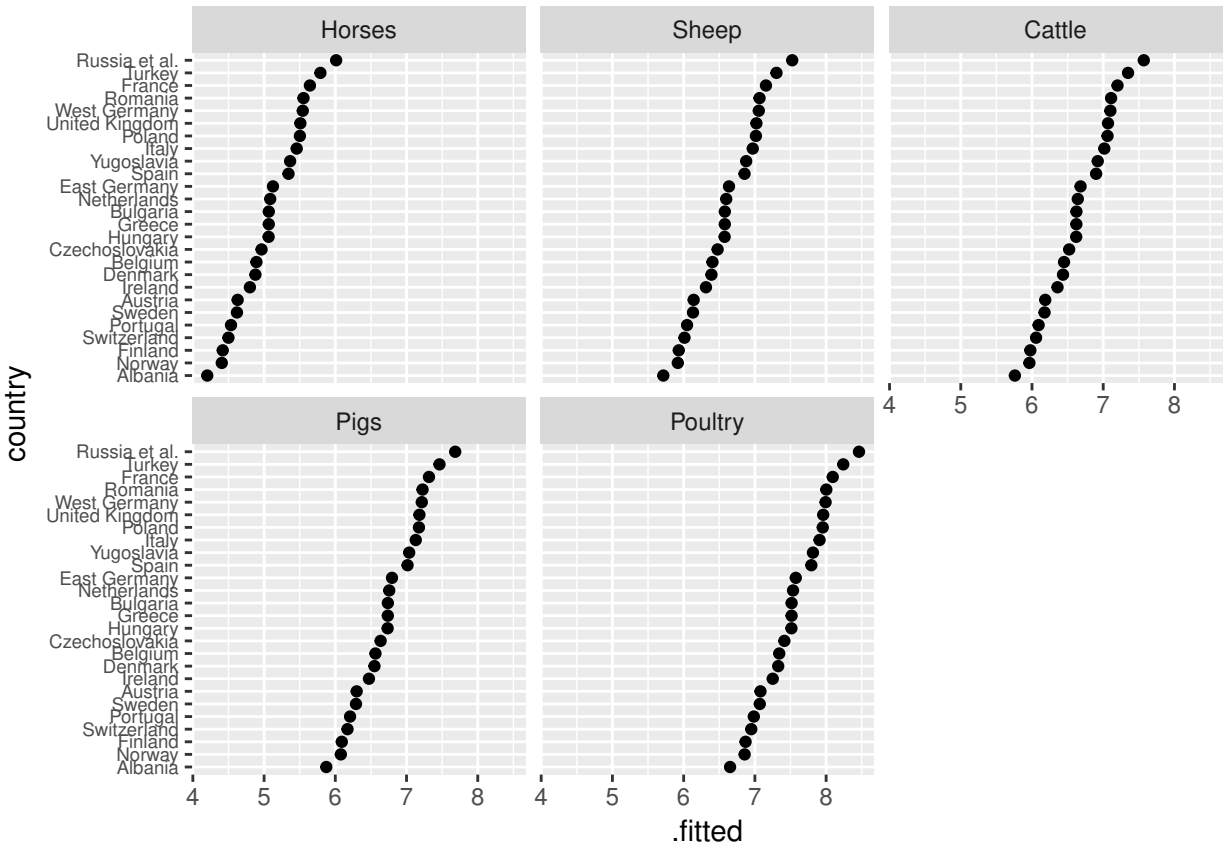


We could also combine the facets on to one plot and distinguish by color. However, that would be a bit busy and disguise the separation into within-country and across-country variation that's the point of the additive model.

6.1.3 Fitted values and residuals

The fitted value consist of center plus animal effect plus country effect. Condition on animal:

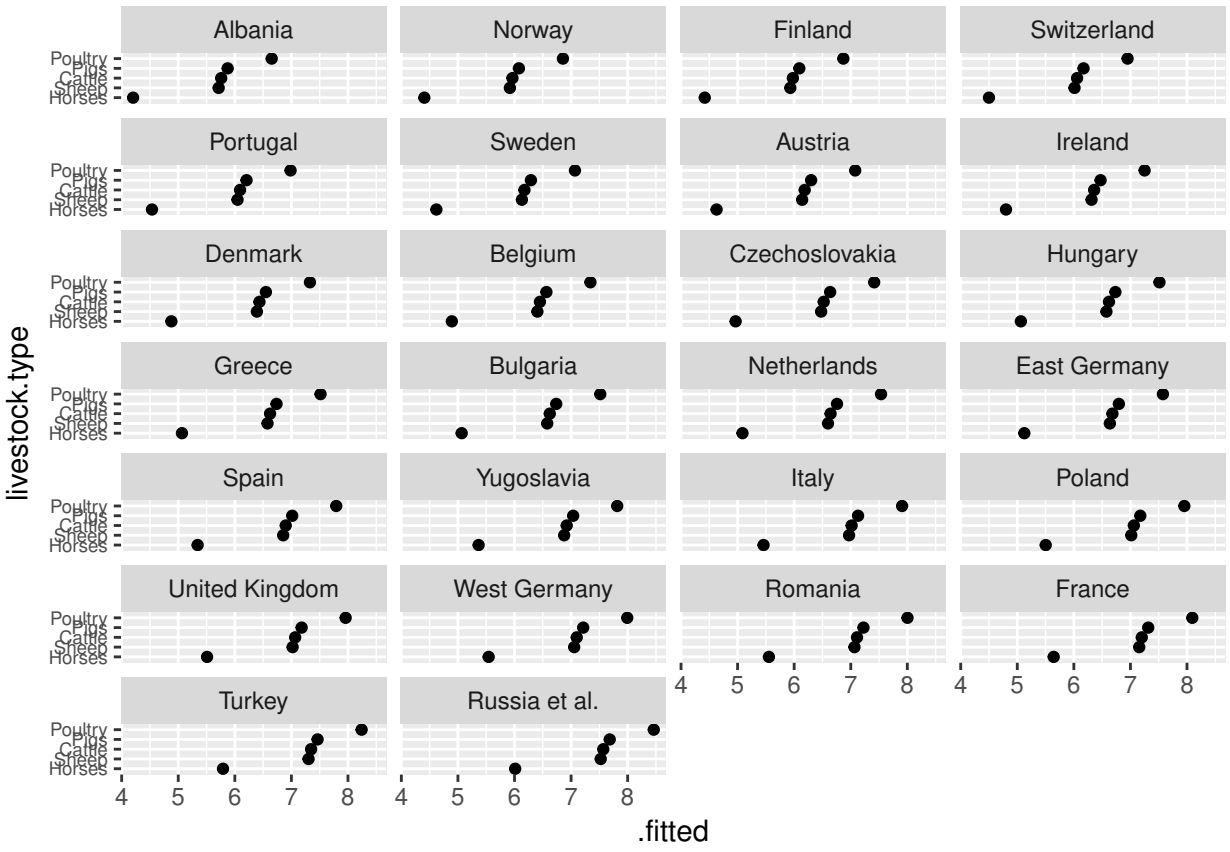
```
ggplot(livestock.rlm.df, aes(x = .fitted, y = country)) + geom_point() + facet_wrap(~livestock.type,
    ncol = 3) + theme(axis.text.y = element_text(size = 7))
```



We see the "curve" just shifts to the right as we get to the more popular animals. The "slope" of the curve is perhaps surprisingly close to vertical: on a log scale, there isn't that much difference between Russia and Albania. Even on the original scale, the Russia multiplier is only about 60 times the Albania multiplier – this might sound like a lot but the former U.S.S.R. was really, really big.

Now condition on country.

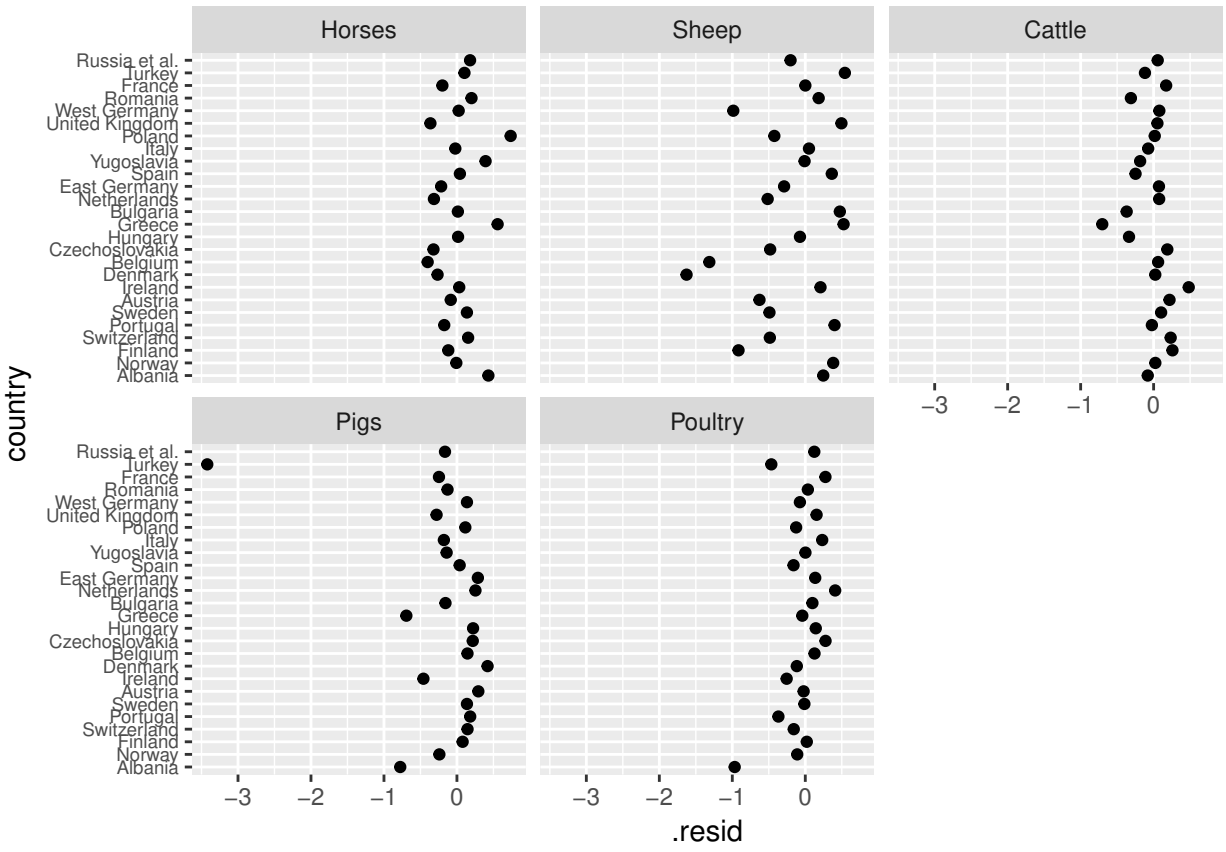
```
ggplot(livestock.rlm.df, aes(x = .fitted, y = livestock.type)) + geom_point() +
    facet_wrap(~country, ncol = 4) + theme(axis.text.y = element_text(size = 7))
```



This is pretty boring – the additive model means we see exactly the same pattern 26 times. The only difference is a very gradual shift to the right (more animals.)

Looking at the residuals should be more interesting. Condition on animal type:

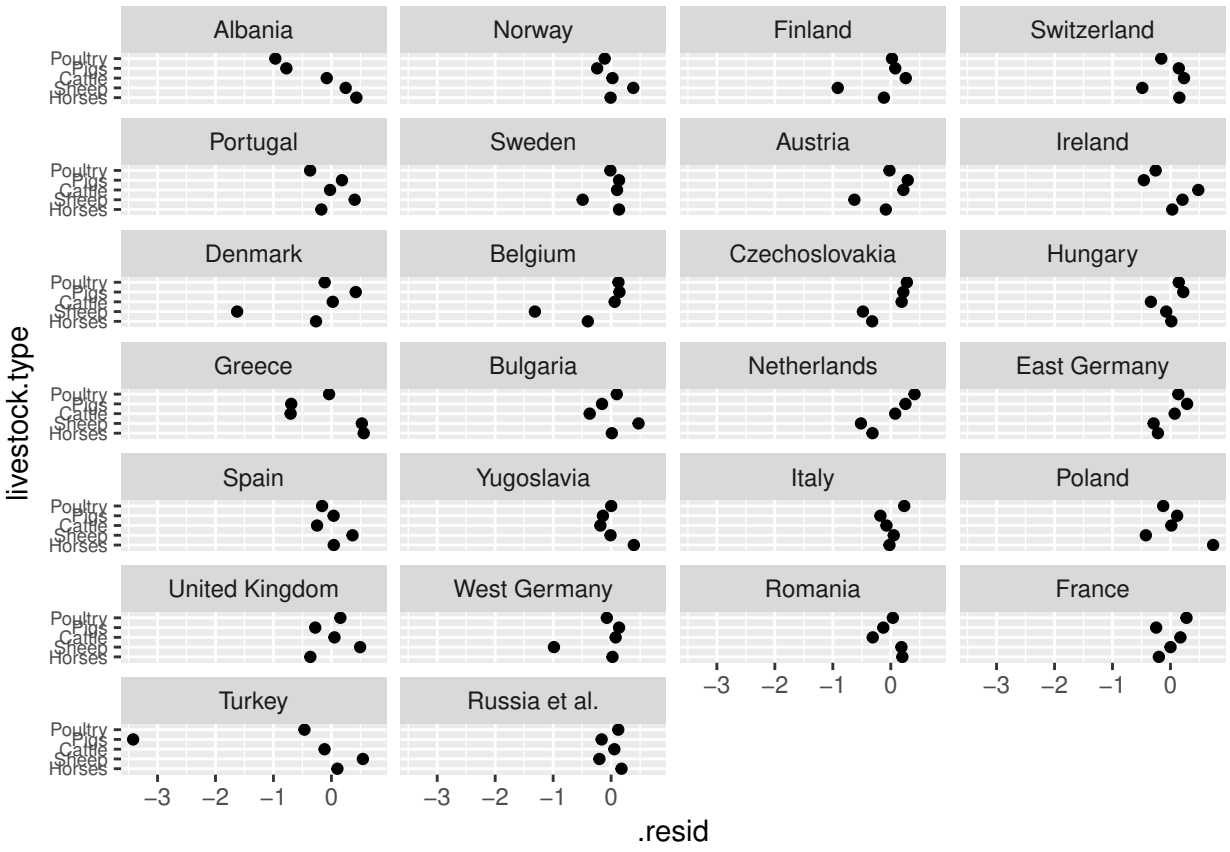
```
ggplot(livestock.rlm.df, aes(x = .resid, y = country)) + geom_point() + facet_wrap(~livestock.type,
    ncol = 3) + theme(axis.text.y = element_text(size = 7))
```



The Turkey-pig outlier is still apparent. We now see that Albania's pig population is somewhat lower than we'd expect from the additive model, but is nowhere near as extreme. The variation in sheep is also notable: some northern European countries just don't like sheep very much.

Condition on country:

```
ggplot(livestock.rlm.df, aes(x = .resid, y = livestock.type)) + geom_point() +
    facet_wrap(~country, ncol = 4) + theme(axis.text.y = element_text(size = 7))
```



Again, we see that sheep are nearly an outlier in countries like Denmark, Belgium, and West Germany. In European countries (in the 1980s), milk and eggs were a consistent part of every culture, while wool and sheep meat were more specialized.

6.1.4 Maps

Maps are overrated. They can be useful when:

- Longitude and latitude are inherently of interest (e.g. when studying climate)
- The audience knows the underlying political geography well (e.g. most Americans know more or less which state goes where.) Even in this case maps can be distracting (because it's not like the shape of the state matters) or misleading (because geographic size might not matter at all.)

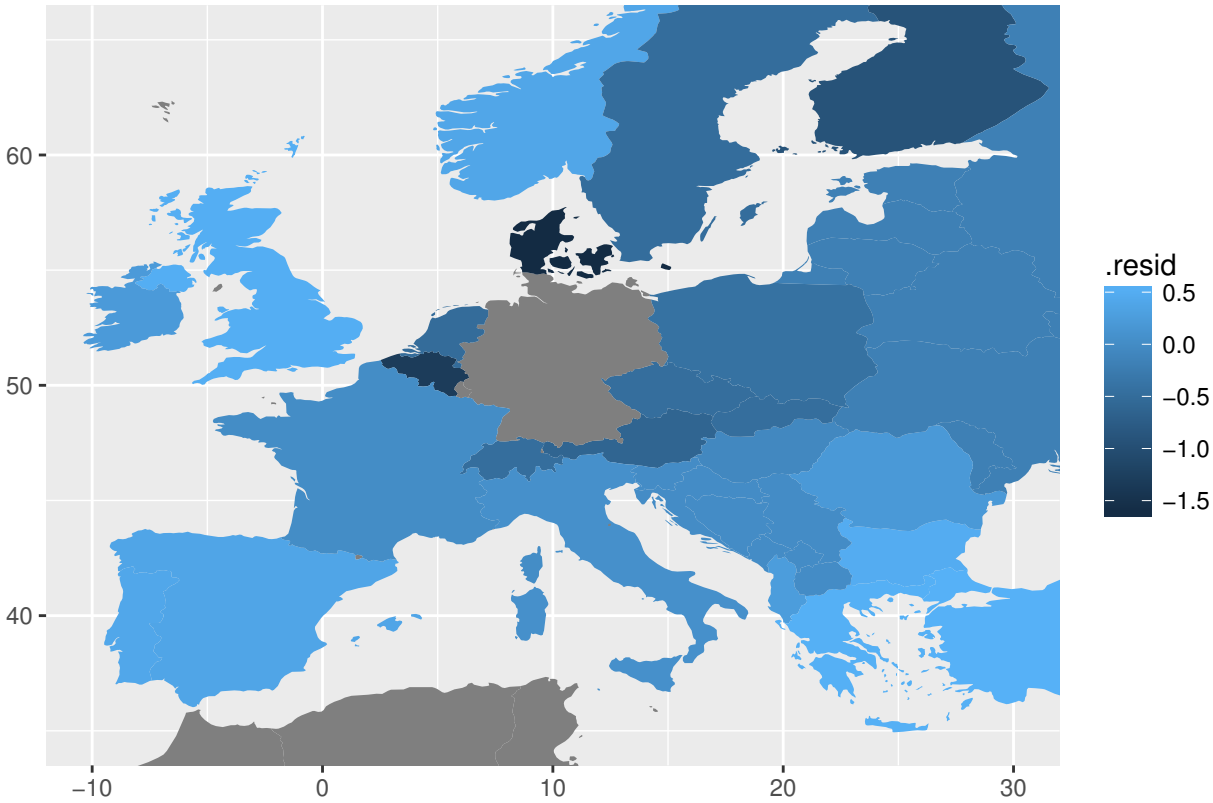
It's quite common that neither of these conditions are met. In our case, most people outside Europe won't know which country is Romania and which is Bulgaria, so we might as well show them dot plots. Nevertheless, maps look cool and knowing how to draw them might impress someone enough that they give you a job, so we'll address them briefly.

One option is to use the `maps` library in conjunction with `geom_map()` in `ggplot2`. We have to fiddle a bit with the country names, e.g. the world map calls the United Kingdom "UK" and doesn't know that these are the same place.

```
sheep.df = subset(livestock.rlm.df, livestock.type == "Sheep")
names(sheep.df)
```

```
## [1] "livestock.type" "country"        "count"          ".fitted"
## [5] ".resid"
```

```
# Change 'country' column name to 'region'
names(sheep.df)[2] = "region"
library(maps)
map.world = map_data(map = "world")
# Pretend it's the Eighties again
map.world$region[map.world$region == "UK"] = "United Kingdom"
map.world$region[map.world$region == "Czech Republic"] = "Czechoslovakia"
map.world$region[map.world$region == "Slovakia"] = "Czechoslovakia"
map.world$region[map.world$region == "Russia"] = "Russia et al."
map.world$region[map.world$region == "Latvia"] = "Russia et al."
map.world$region[map.world$region == "Lithuania"] = "Russia et al."
map.world$region[map.world$region == "Estonia"] = "Russia et al."
map.world$region[map.world$region == "Belarus"] = "Russia et al."
map.world$region[map.world$region == "Ukraine"] = "Russia et al."
map.world$region[map.world$region == "Moldova"] = "Russia et al."
map.world$region[map.world$region == "Serbia"] = "Yugoslavia"
map.world$region[map.world$region == "Montenegro"] = "Yugoslavia"
map.world$region[map.world$region == "Croatia"] = "Yugoslavia"
map.world$region[map.world$region == "Bosnia and Herzegovina"] = "Yugoslavia"
map.world$region[map.world$region == "Kosovo"] = "Yugoslavia"
map.world$region[map.world$region == "Macedonia"] = "Yugoslavia"
map.world$region[map.world$region == "Slovenia"] = "Yugoslavia"
sheep.merge = merge(map.world, sheep.df, by = "region", all.x = TRUE, all.y = FALSE)
ggplot(sheep.merge, aes(fill = .resid)) + geom_map(map = map.world, aes(map_id = region)) +
    coord_equal() + xlim(-10, 30) + ylim(35, 65)
```



The lighter colors are sheepy countries, while the darker ones are less sheepy. In general, the map gets darker as you move north, though there are plenty of exceptions, like the U.K. Note that the one thing I didn't fix

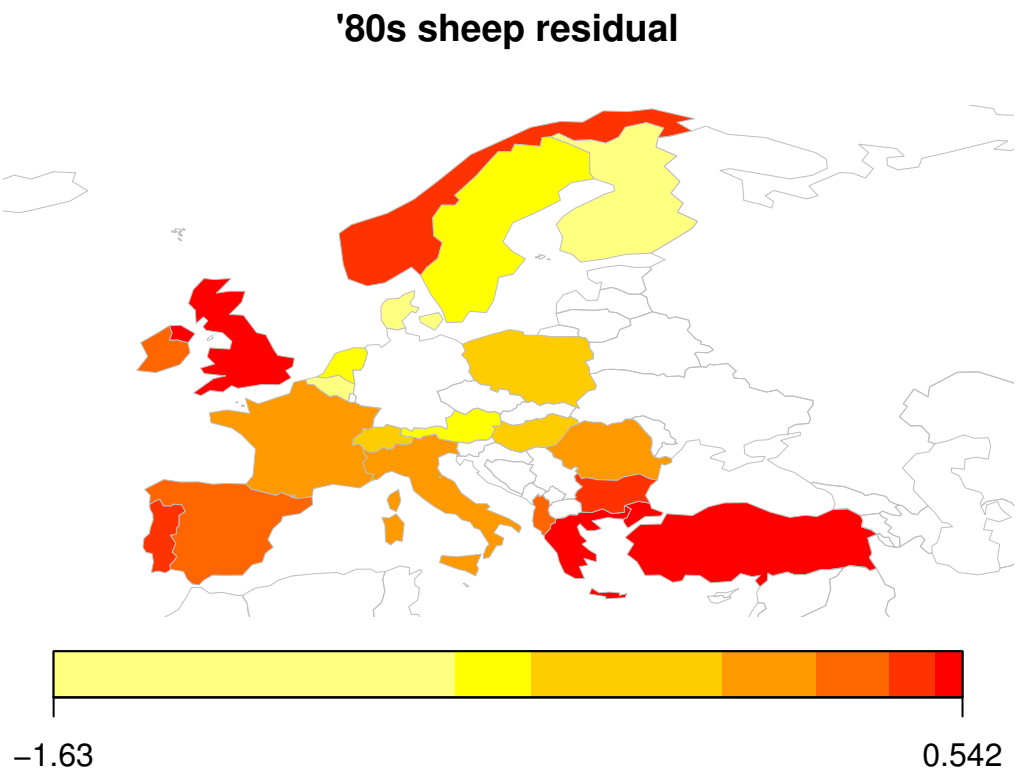
was combining West and East Germany, because I was too lazy to do it properly. (The easiest non-cheating solution would be to combine both Germanies in the original data set and refit the whole model.)

A slightly more user-friendly option is the `rworldmap` package.

```
# install.packages('rworldmap')
library(rworldmap)
sheep.join = joinCountryData2Map(sheep.df, joinCode = "NAME", nameJoinColumn = "region")
```

```
## 21 codes from your data successfully matched countries in the map
## 5 codes from your data failed to match with a country code in the map
## 222 codes from the map weren't represented in your data
```

```
mapCountryData(sheep.join, nameColumnToPlot = ".resid", mapRegion = "europe",
    mapTitle = "'80s sheep residual")
```



Again, you can rename things to fill in some of the blanks for newly split countries. You probably get the point by now, though.

6.2 Multiway data

READ: Cleveland pp. 320–340.

6.2.1 Run time

Load data and libraries:

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

```
library(broom)
library(MASS)
```

The `run.time` data frame contains experimental results on the running time required for three sorting algorithms. The variables are:

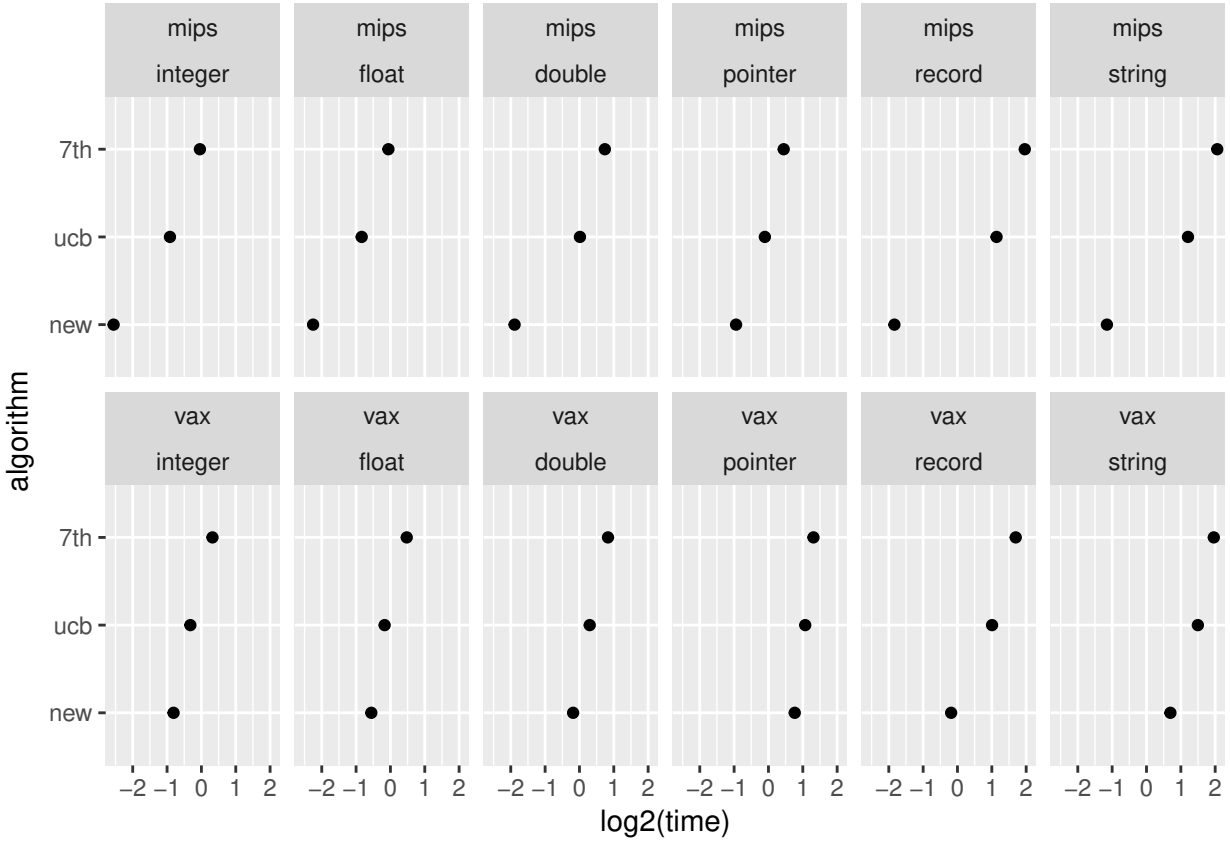
- `time` in seconds. This is strongly skewed, so we'll mostly use $\log_2$ of time;
- `algorithm`: 7th, ucb, or new;
- `machine`: vax or mips;
- `input`: integer, float, double, record, pointer, or string.

This time we have *three-way* data: there are three explanatory factor variables, and one numerical response (time.) To display the data, we can:

- Facet on two factors, and draw dotplots showing the relationship of the response with the third factor; or
- Facet on one factor, distinguish a second factor by color, and draw dotplots showing the relationship of the response with the third factor.

First, we try faceting two ways. The algorithm is the main explanatory variable of interest, so condition on the other two.

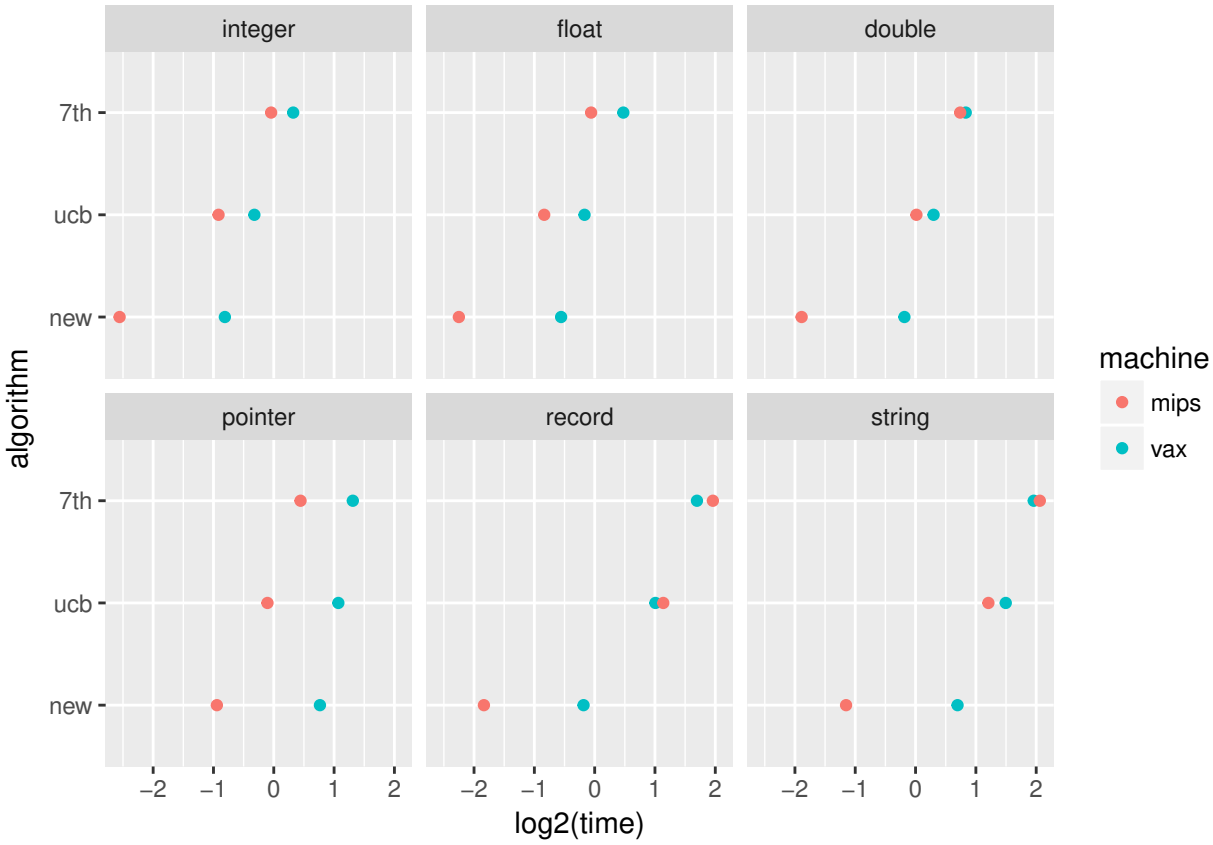
```
ggplot(run.time, aes(x = log2(time), y = algorithm)) + geom_point() + facet_wrap(~machine +
    input, ncol = 6)
```



This shows that the order (from shortest to longest run time) is always new $<$ ucb $<$ 7th. However, the differences aren't constant.

Now keep algorithm as the main explanatory, but use color to distinguish one of the others. Since there are only two machines, color by that variable to avoid too many different colors (which requires your eyes going to the legend and back repeatedly and is thus annoying.)

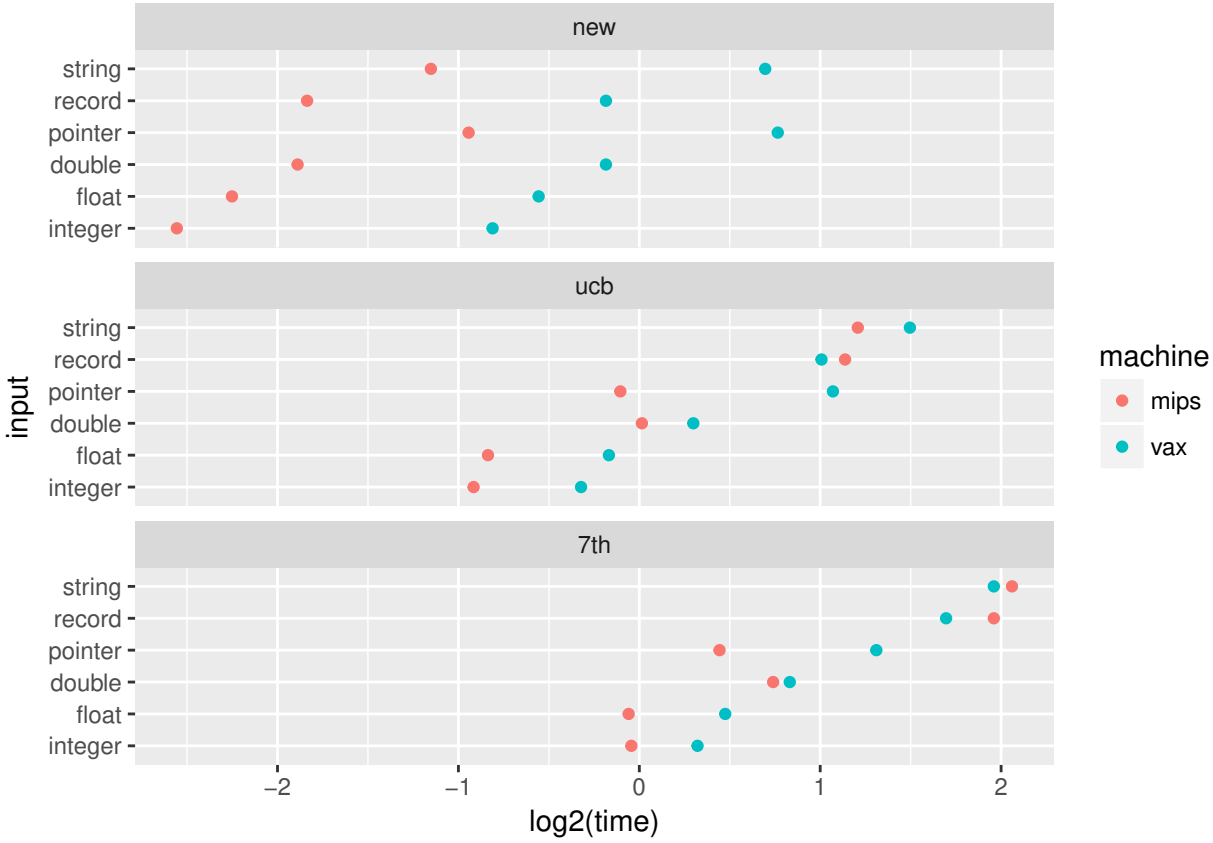
```
ggplot(run.time, aes(x = log2(time), y = algorithm, color = machine)) + geom_point() +
    facet_wrap(~input, ncol = 3)
```



The "slopes" of the red dots are systematically steeper than the "slopes" of the blue dots. That suggest an algorithm-machine interaction.

Now make input the main explanatory. Facet on algorithm and keep machine as color:

```
ggplot(run.time, aes(x = log2(time), y = input, color = machine)) + geom_point() +
    facet_wrap(~algorithm, ncol = 1)
```



Interestingly, the top plot looks like a simple shift from red to blue, the other two plots look much more complicated. To be on the safe side, we should keep an input-machine interaction in the model.

6.2.2 Interlude: The `spread()` function

We know that the new algorithm gives the fastest run times. So what we want to quantify is how much faster it is than the other two algorithms (on the log 2 scale.) In other words, we're interested in the *differences* between the two old algorithms and the new one. With a well-organized data set like this one, it's straightforward to find the differences manually. However, with messier data, it might not be so easy. We thus take this opportunity to learn the `spread()` function in `tidyr`.

`spread()` is nearly the opposite of `gather()`: instead of collapsing columns and making our data long, we break up columns and make our data wide. Run the following line:

```
run.wide = run.time %>% spread(algorithm, time)
```

What this does is create new columns, one for each level of `algorithm`. The numbers that go into these columns are the ones currently in `time`. This is the result:

```
run.wide
```

```
##    machine   input  new  ucb  7th
## 1     mips integer 0.17 0.53 0.97
## 2     mips   float 0.21 0.56 0.96
## 3     mips  double 0.27 1.01 1.67
## 4     mips pointer 0.52 0.93 1.36
## 5     mips  record 0.28 2.20 3.89
## 6     mips  string 0.45 2.31 4.17
## 7      vax integer 0.57 0.80 1.25
```

```
## 8      vax   float 0.68 0.89 1.39
## 9      vax  double 0.88 1.23 1.78
## 10     vax pointer 1.70 2.10 2.48
## 11     vax  record 0.88 2.01 3.24
## 12     vax  string 1.62 2.82 3.89
```

Now create two new variables with the improvements in logged run time of the new algorithm over the two old ones. More positive means more improvement.

```
run.wide$imp.ucb = log2(run.wide$ucb) - log2(run.wide$new)
run.wide$imp.7th = log2(run.wide$"7th") - log2(run.wide$new)
```

Now that we've created these two variables, we can switch back to long form.

```
run.long = run.wide %>% gather(algorithm, improvement, imp.ucb:imp.7th)
```

Let's summarize our new `improvement` variable by finding its mean for each input.

```
improvement.means = aggregate(improvement ~ input, mean, data = run.long)
improvement.means
```

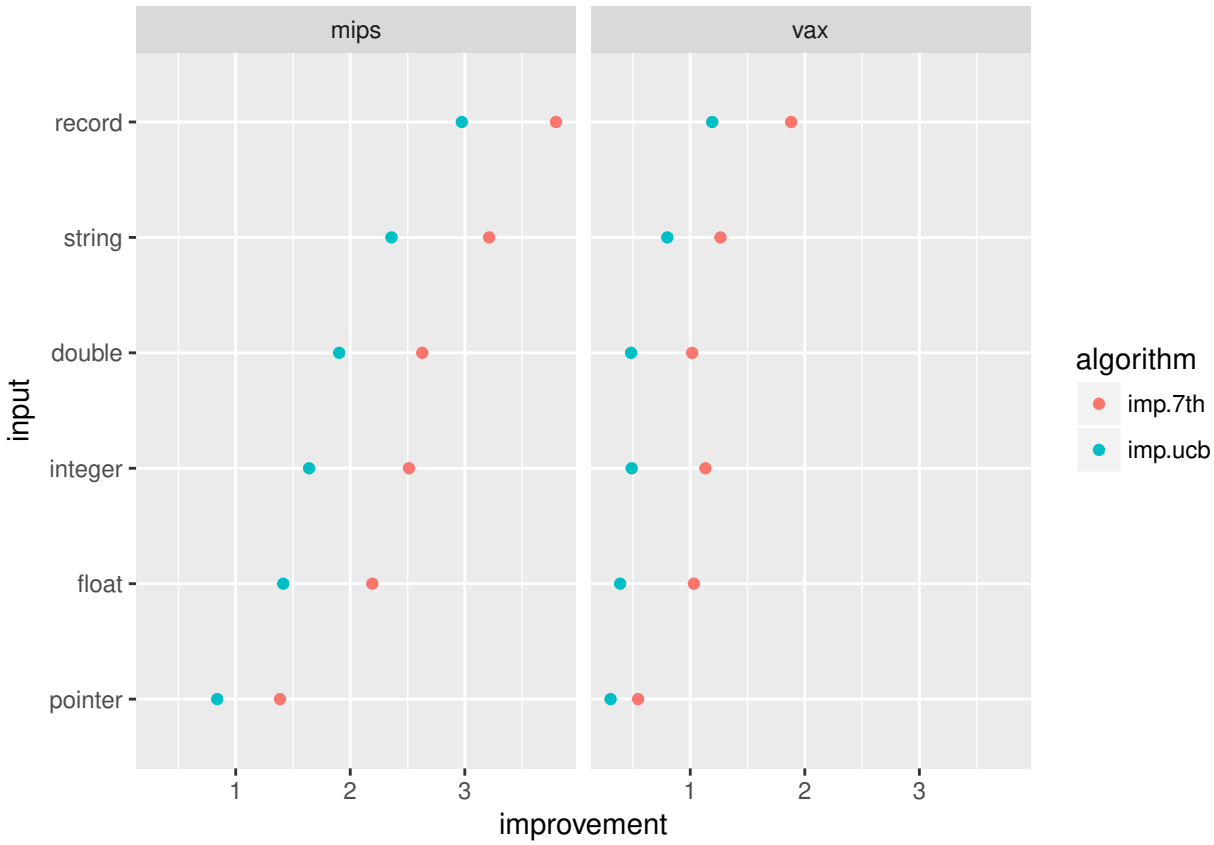
```
##     input improvement
## 1 integer   1.4437100
## 2   float   1.2568578
## 3  double   1.5078814
## 4 pointer   0.7688505
## 5  record   2.4605787
## 6  string   1.9088560
```

For aesthetic purposes, reorder the `input` variable by mean improvement:

```
input.order = improvement.means$input[order(improvement.means$improvement)]
run.long$input = factor(run.long$input, levels = input.order)
```

Plot the data, this time coloring by algorithm:

```
ggplot(run.long, aes(x = improvement, y = input, color = algorithm)) + geom_point() +
    facet_grid(~machine)
```



Interestingly, the difference between blue and red is close to constant on the left panel and close to (a different) constant on the right panel. In other words, we might not need an algorithm-input interaction.

6.2.3 Multiway linear model

Let's say we want to fit input:machine and algorithm:machine interactions, but not one for input:algorithm. There are several ways to specify this. Since there are no outliers, we can just fit an ordinary linear model. We can write out the interactions explicitly:

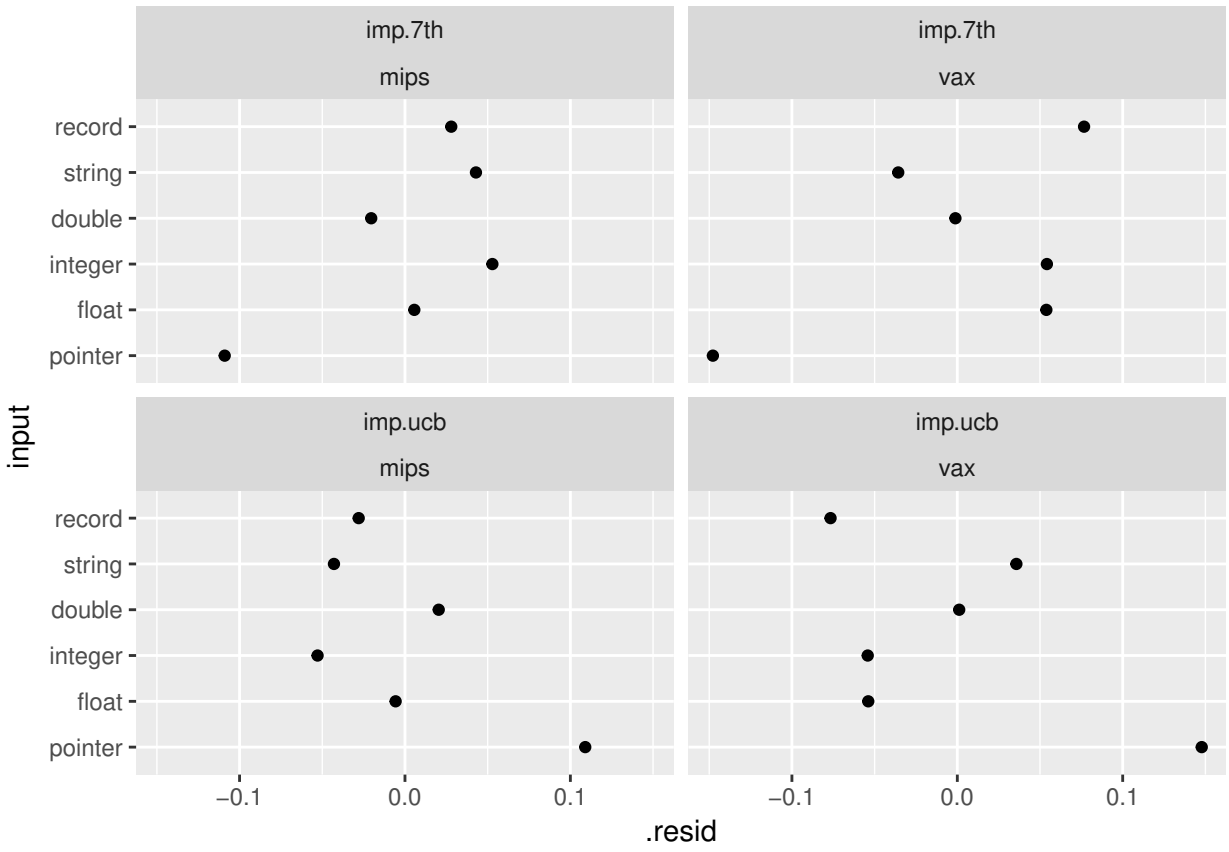
```
runtime.lm = lm(improvement ~ input + algorithm + machine + input:machine +
    algorithm:machine, data = run.long)
```

Or we can think of it the followay way: "Without machine, input and algorithm are addtive. But both interact with machine."

```
runtime.lm = lm(improvement ~ (input + algorithm) * machine, data = run.long)
```

You should get the same thing either way. We check the residuals by plotting them faceted two ways:

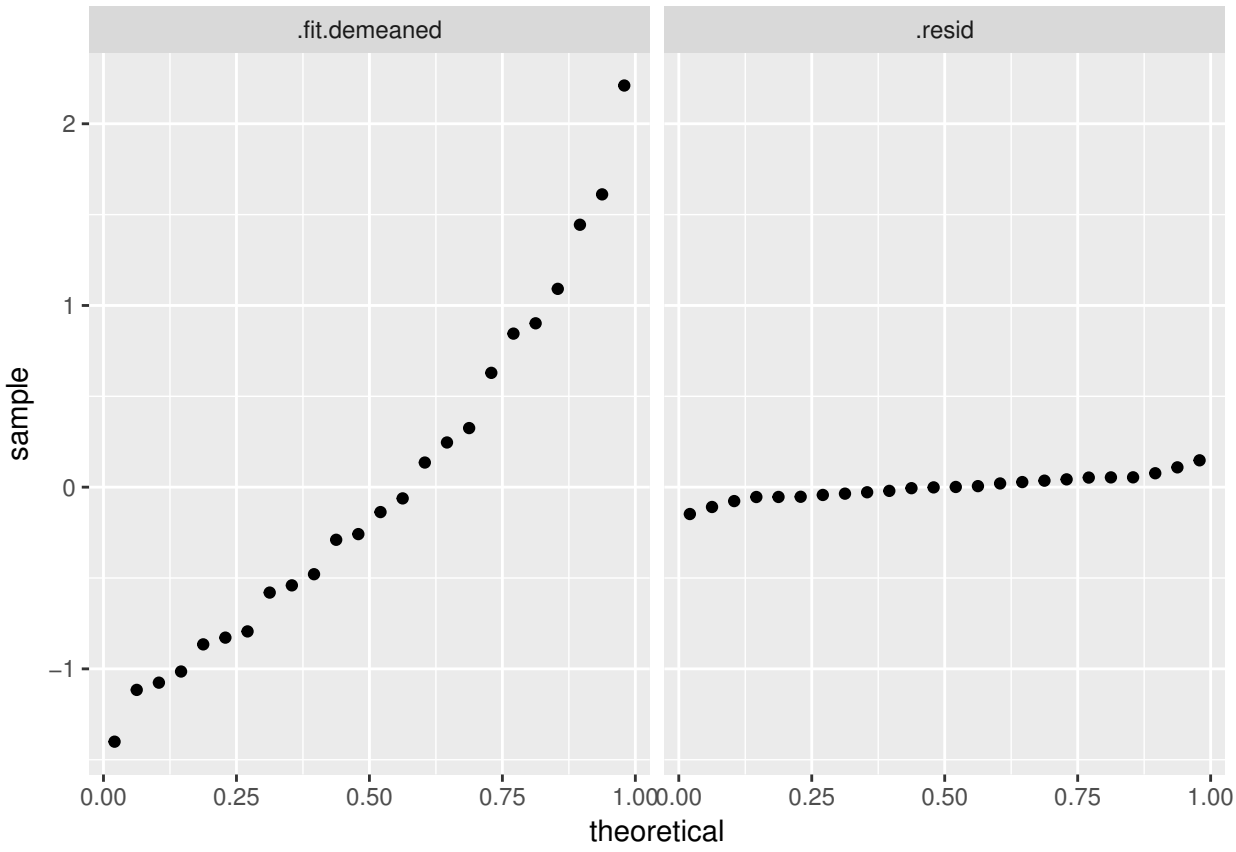
```
runtime.lm.df = augment(runtime.lm)
ggplot(runtime.lm.df, aes(x = .resid, y = input)) + geom_point() + facet_wrap(~algorithm +
    machine)
```



There's not much to note here except that the "pointer" residuals are the largest in magnitude. Do we need an input-algorithm interaction just for pointer? Perhaps, but this could also just be luck.

Now check a residual-fit plot to see how much is left unexplained.

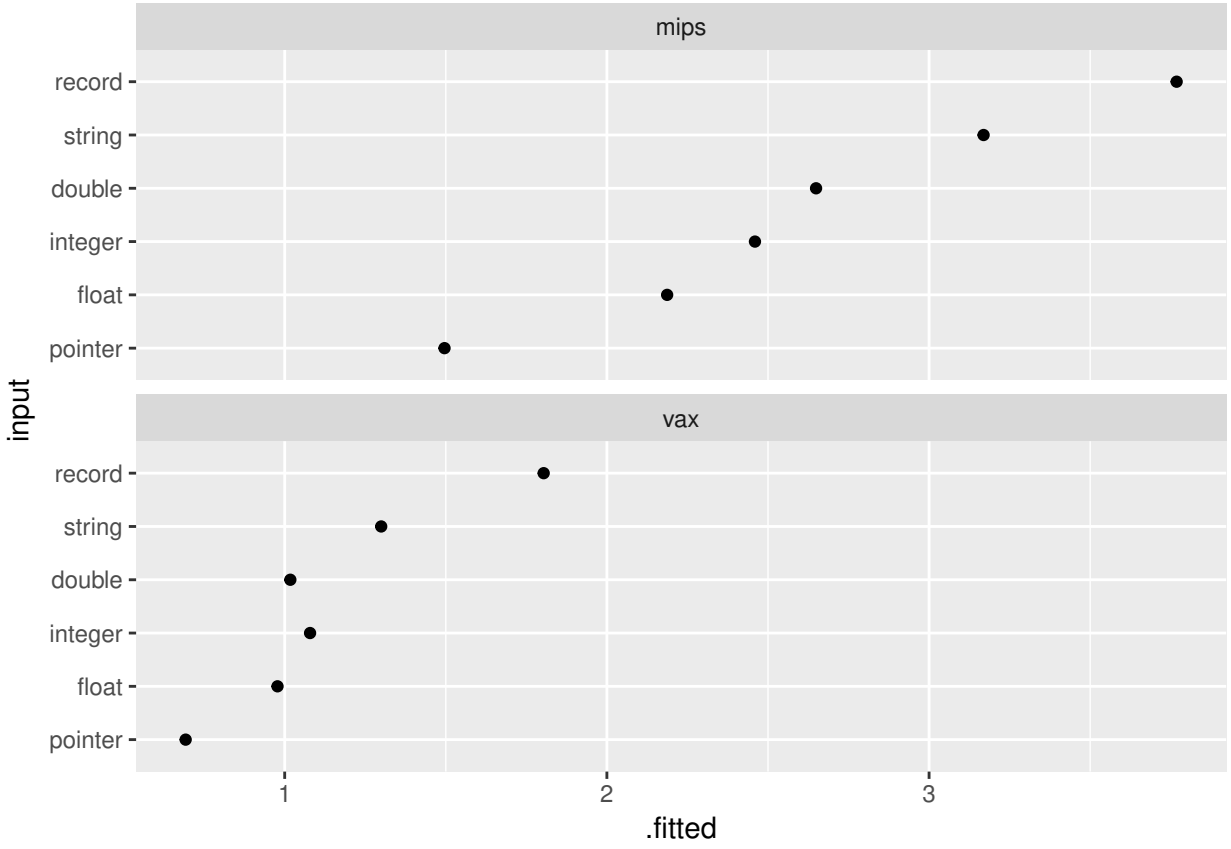
```
runtime.lm.df$.fit.demeaned = runtime.lm.df$.fitted - mean(runtime.lm.df$.fitted)
runtime.lm.long = runtime.lm.df %>% gather(component, value, c(.fit.demeaned,
    .resid))
ggplot(runtime.lm.long, aes(sample = value)) + stat_qq(distribution = "qunif") +
    facet_grid(~component)
```



We've explained almost all of the variation. This may not be as impressive as it seems, since we fitted a 14-parameter model to 24 data points, but we'll take it.

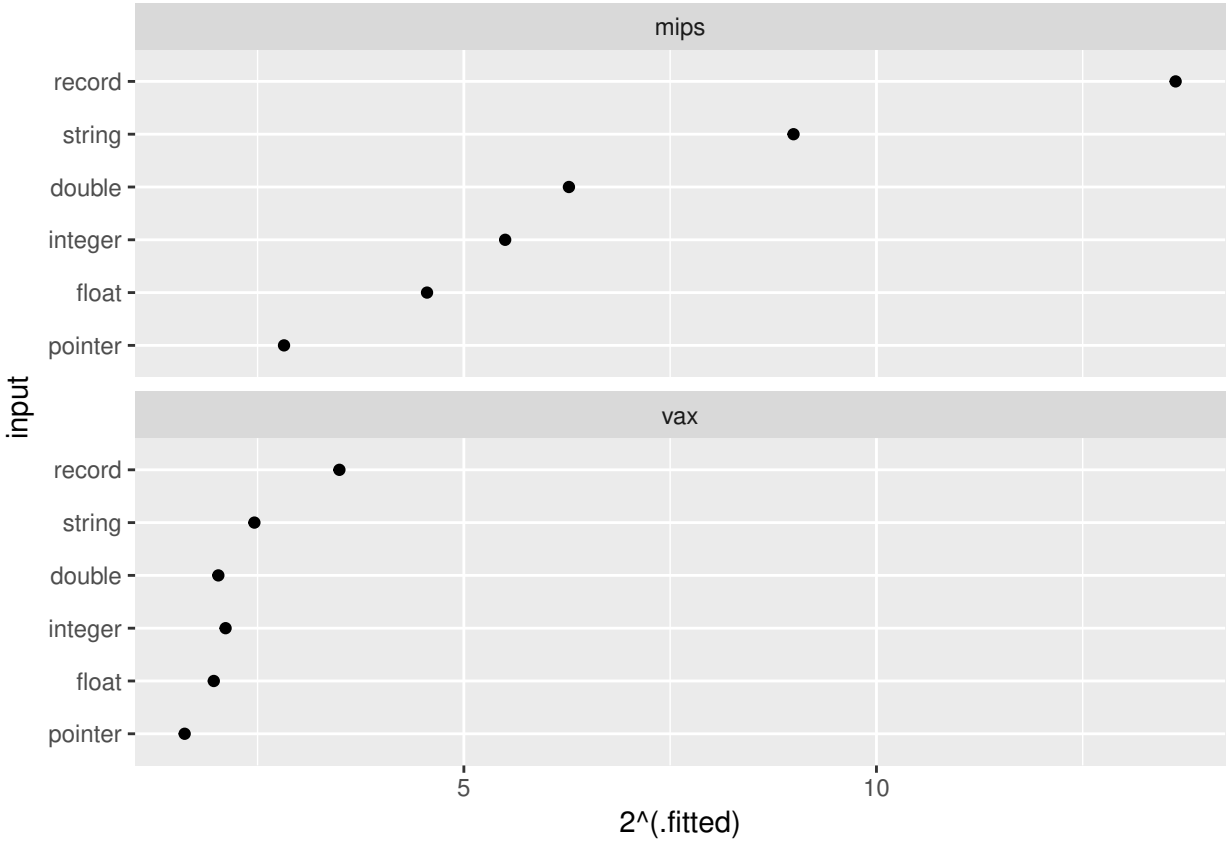
To better understand the input:machine interaction, draw a dot plot of fitted values with those two explanatories.

```
ggplot(subset(runtime.lm.df, algorithm == "imp.7th"), aes(x = .fitted, y = input)) +
    geom_point() + facet_wrap(~machine, ncol = 1)
```



We see the shape of the two sets of dots is quite different. It's not as simple as adding different effects for mips and vax. We can also replot this on the original scale:

```
ggplot(subset(runtime.lm.df, algorithm == "imp.7th"), aes(x = 2^(.fitted), y = input)) +
    geom_point() + facet_wrap(~machine, ncol = 1)
```



The back-transformed fitted values give the number of times longer the old algorithms take compared to the new one. The new algorithm is always faster, but the increase in speed is highly variable. mips looks particularly bad in comparison, usually taking over five times as long.

6.3 The barley controversy

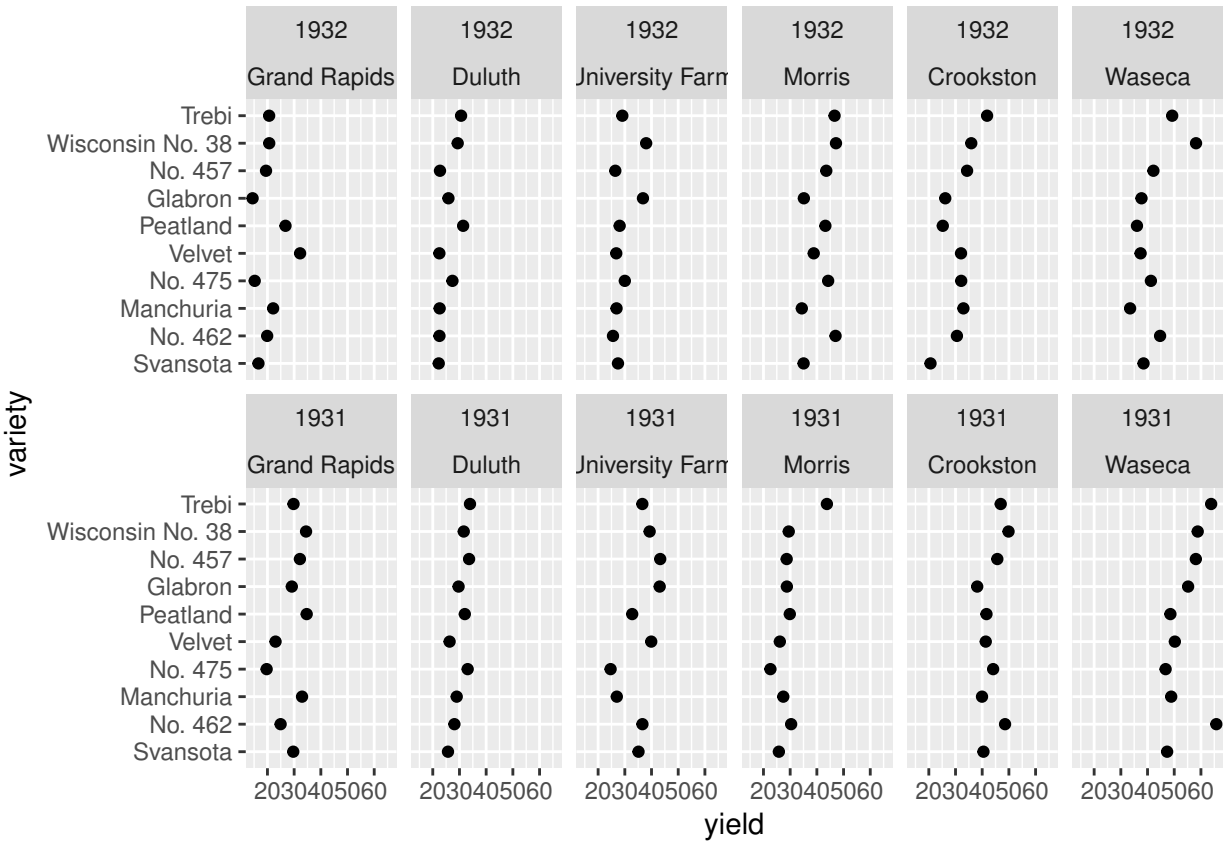
Immer's Minnesota barley data is famous as one of the first real-life applications of analysis of variance. It's also controversial due to a supposed mistake in the data. The variables in the `barley` data frame are:

- `yield`: barley crop yield in bushels per acre (a bushel is four pecks);
- `variety`: one of ten types of barley;
- `year`: 1931 or 1932;
- `site`: one of six locations in Minnesota.

So there are three categorical explanatory variables (we might as well treat year as categorical, since we only have two years) and a quantitative response. The factor variables are presorted by median.

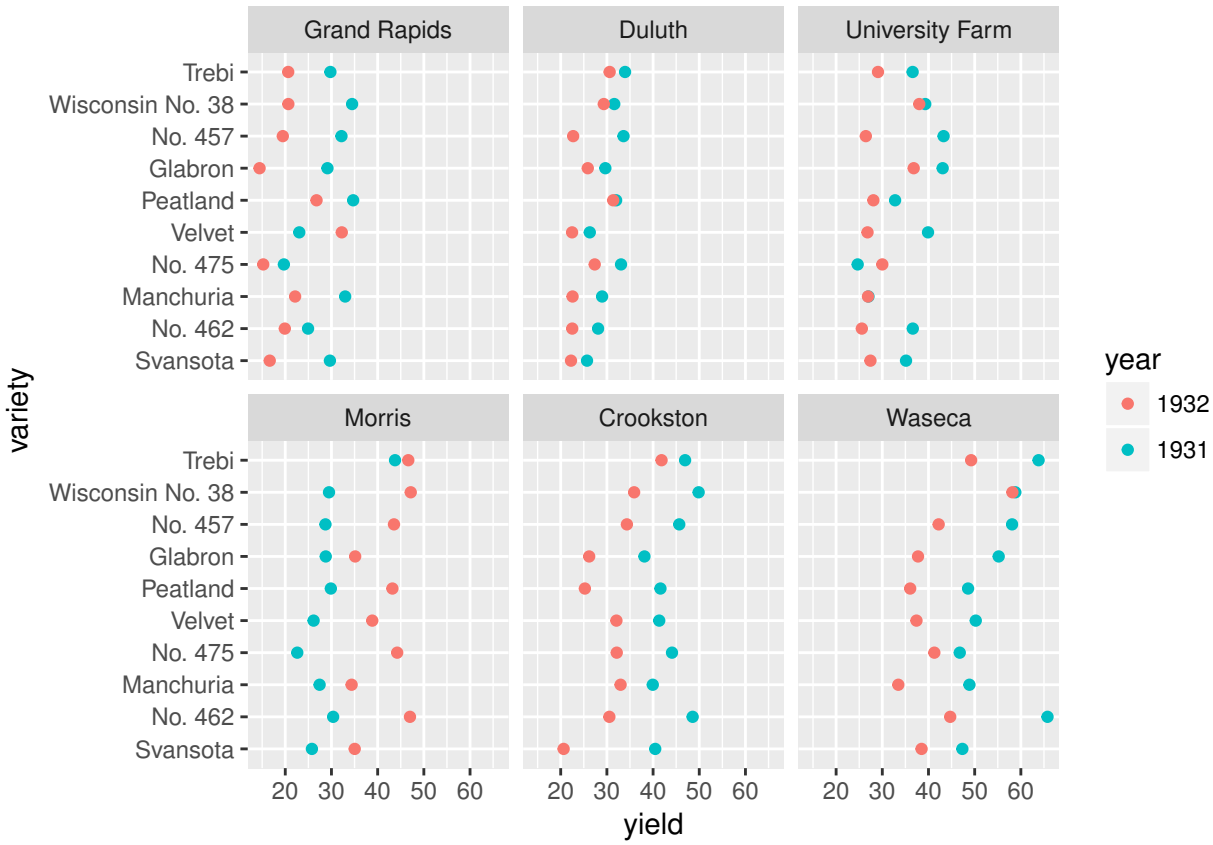
Start with a two-way faceted dot plot, with `variety` as the main explanatory:

```
ggplot(barley, aes(x = yield, y = variety)) + geom_point() + facet_wrap(~year +
    site, ncol = 6)
```



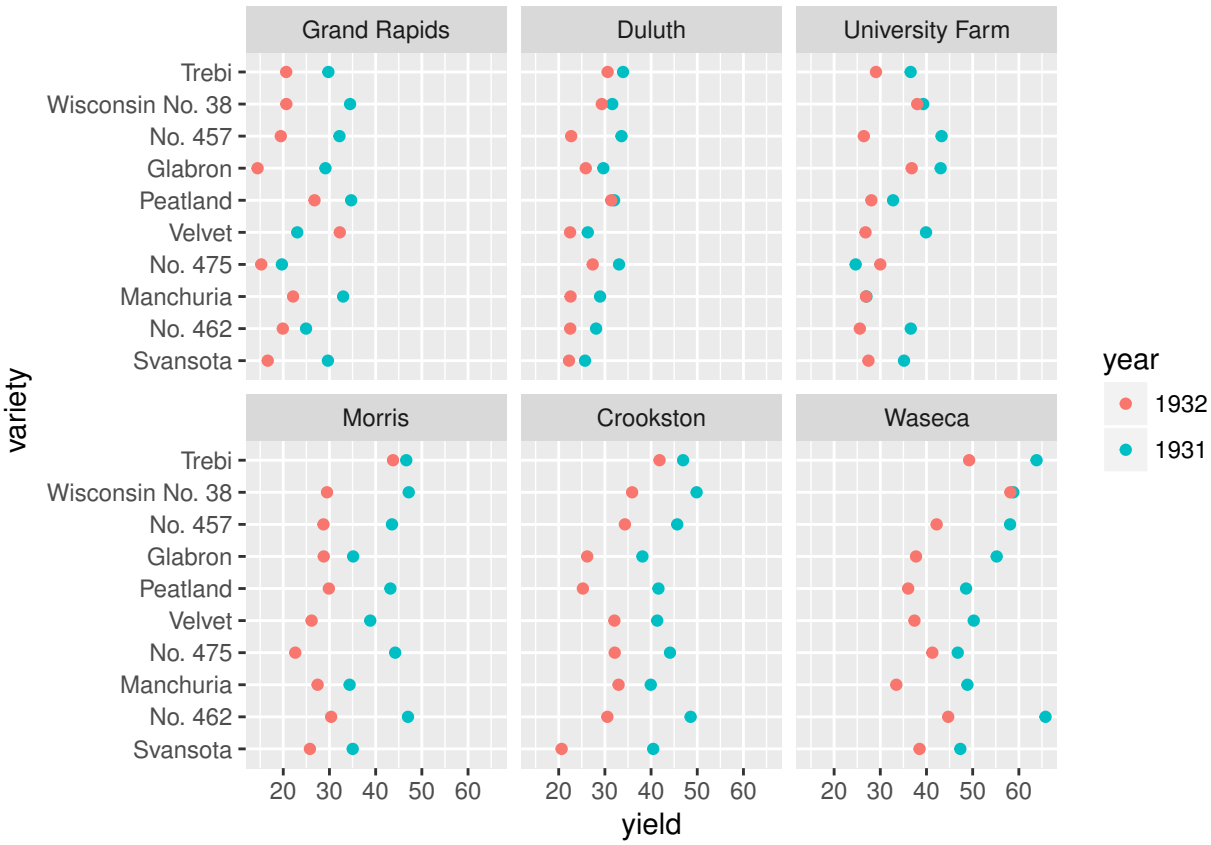
There' are a few too many dots's too much going on here for the patterns to be clear. Instead, let's plot one row on top of the other, distinguishing year by color.

```
ggplot(barley, aes(x = yield, y = variety, color = year)) + geom_point() + facet_wrap(~site)
```



In most cases, the teal dot is to the right of the red dot: that is, the 1931 yield was higher than the 1932 yield. However, the opposite is true at Morris, where every yield increased from 1931 to 1932. Cleveland claims this could have been a mistake in data entry, and we should at least entertain this possibility. Let's swap the 1931 and 1932 numbers for Morris, and see what difference that makes.

```
morris1932fixed = barley$yield[barley$site == "Morris" & barley$year == 1931]
morris1931fixed = barley$yield[barley$site == "Morris" & barley$year == 1932]
barley.fixed = barley
barley.fixed$yield[barley$site == "Morris" & barley$year == 1932] = morris1932fixed
barley.fixed$yield[barley$site == "Morris" & barley$year == 1931] = morris1931fixed
ggplot(barley.fixed, aes(x = yield, y = variety, color = year)) + geom_point() +
    facet_wrap(~site)
```



On one hand, it looks nicer – the pattern is now consistent apart from the odd exception (Grand Rapids–Velvet.) On the other hand, perhaps the pattern is too good to be true: we know that sometimes outliers happen, and maybe Morris was hit by a plague of locusts in 1931 or something. Personally, the data doesn't entirely convince me, so I'll err on the side of sticking with the original data.

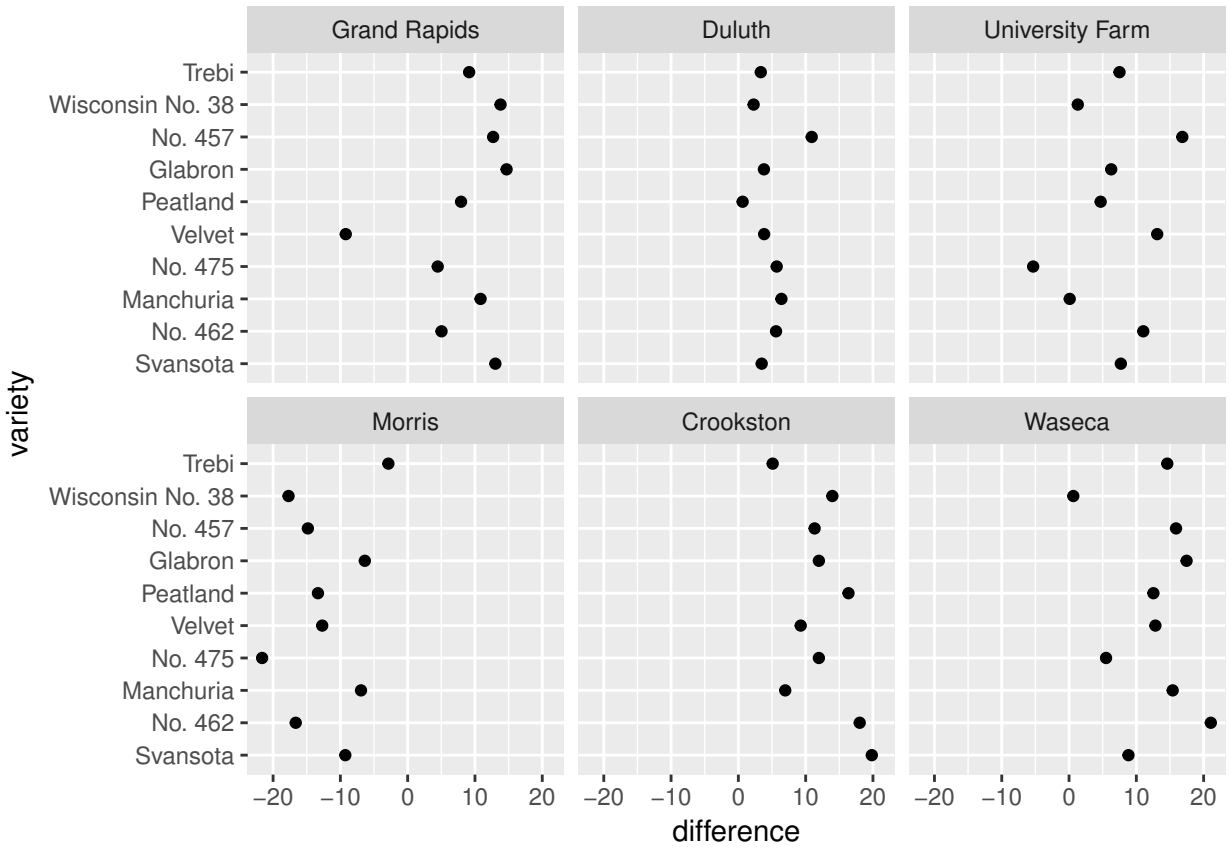
We can simplify the structure by looking at the difference between the 1931 and 1932 yields. To do this, we:

- Put the data into "wide" form by using the `spread()` function in `tidyr`. This will create columns called "1931" and "1932" containing the yields for those years, and halve the number of rows.
- Create a new data frame column called `difference` with the 1931 yield minus the 1932 yield for each variety and site. (We do the subtraction this way around because people like positive numbers.)

```
barley.wide = barley %>% spread(year, yield)
barley.wide$difference = barley.wide$"1931" - barley.wide$"1932"
```

Now we draw a dot plot faceting on site:

```
ggplot(barley.wide, aes(x = difference, y = variety)) + geom_point() + facet_wrap(~site)
```

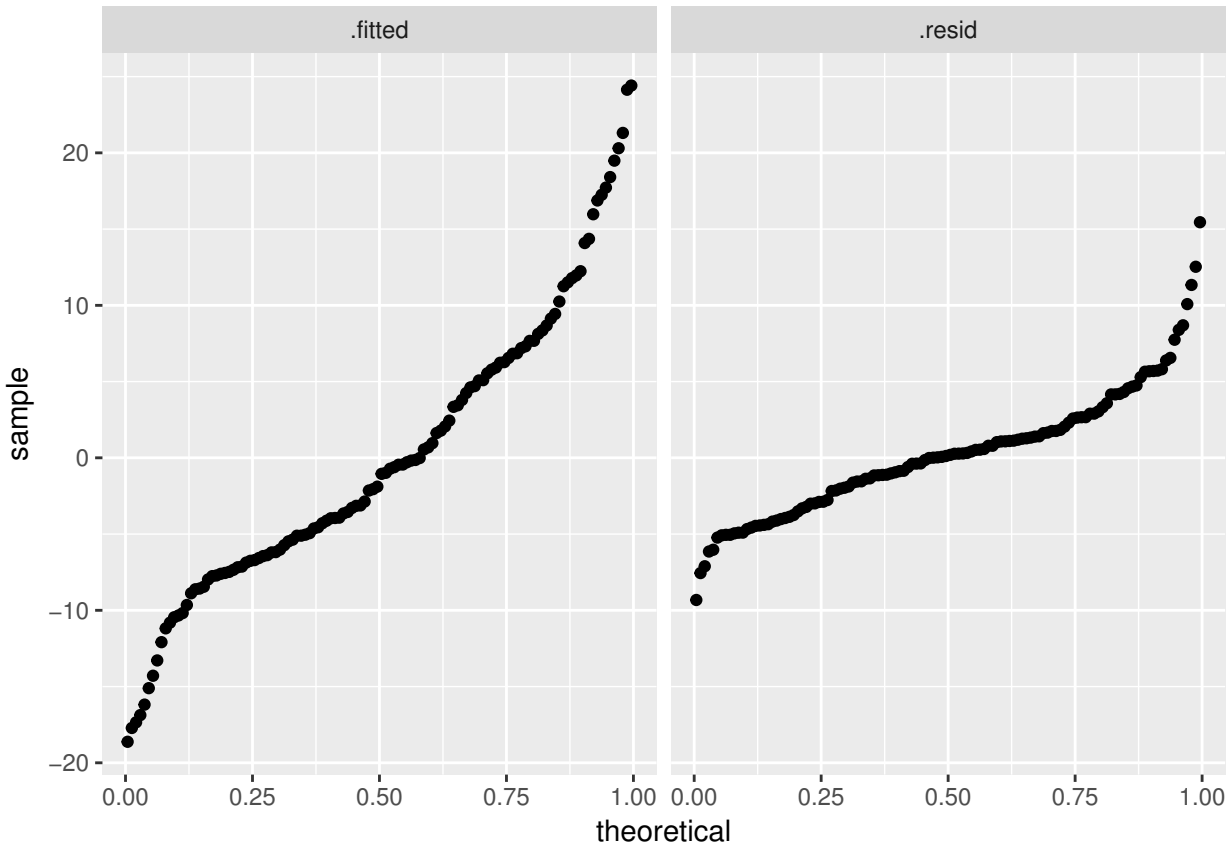


The main modeling question we try to answer here is whether some interaction between variety and the other variables is necessary. If the variety-by-variety variation here is small (i.e. each panel's dot are scattered close to vertical line), then perhaps we can do without such interactions. It's a judgment call, but to me there doesn't seem to be much pattern. On the other hand, we probably need a year:site interaction, for Morris if for no other reason. Because of the outliers, we'll use `rlm()` with bisquare for the fit.

```
barley.rlm = rlm(yield ~ variety + year * site, psi = psi.bisquare, data = barley)
```

Before we move on, let's check the residual-fit plot to ensure we've captured an informative amount of variation.

```
barley.rlm.df = augment(barley.rlm)
barley.rlm.df$.fitted = barley.rlm.df$.fitted - mean(barley.rlm.df$.fitted)
barley.rlm.long = barley.rlm.df %>% gather(component, value, c(.fitted, .resid))
ggplot(barley.rlm.long, aes(sample = value)) + stat_qq(distribution = "qunif") +
    facet_grid(~component)
```

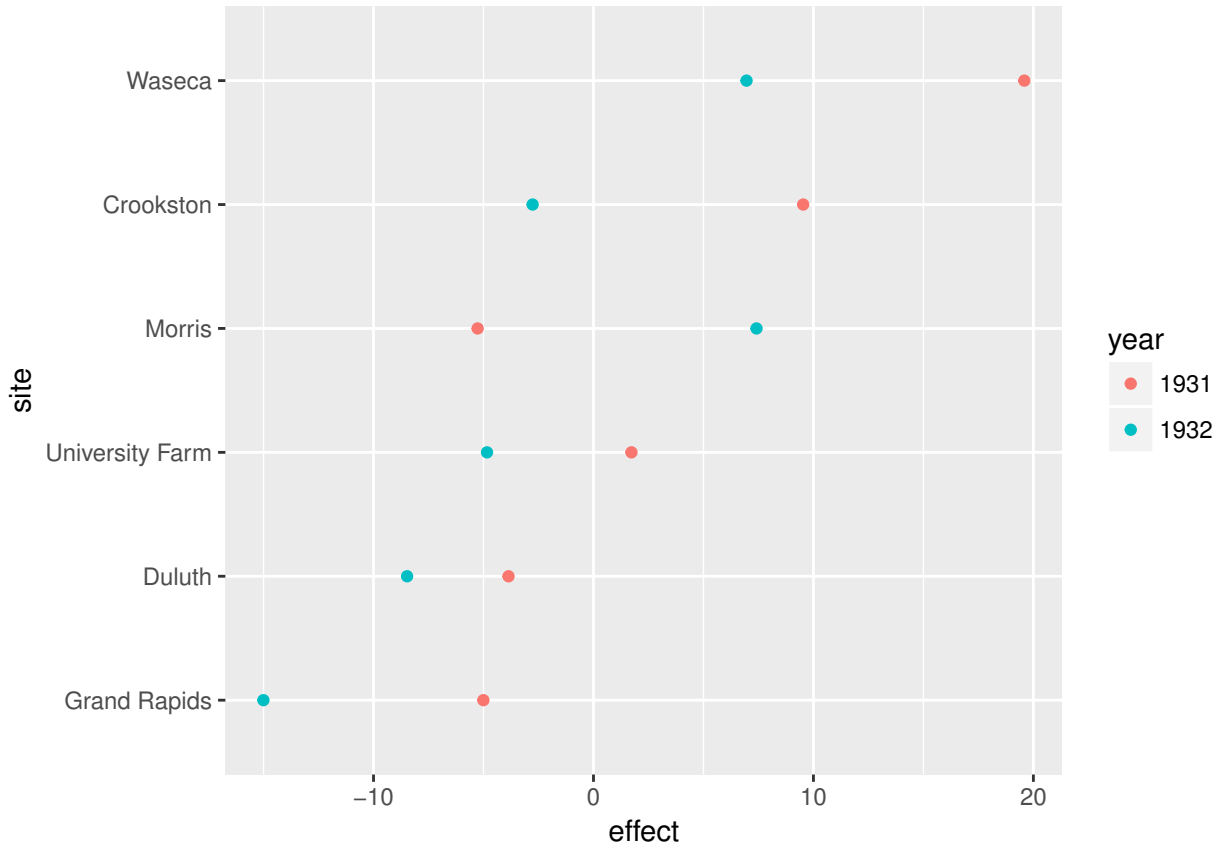


It seems like we've explained the majority of the variation, so let's proceed.

6.3.1 Viewing the model

Because we have a year:site interaction, we should view the effects of these two variables in tandem. Draw a dotplot of the effects, with color indicating year:

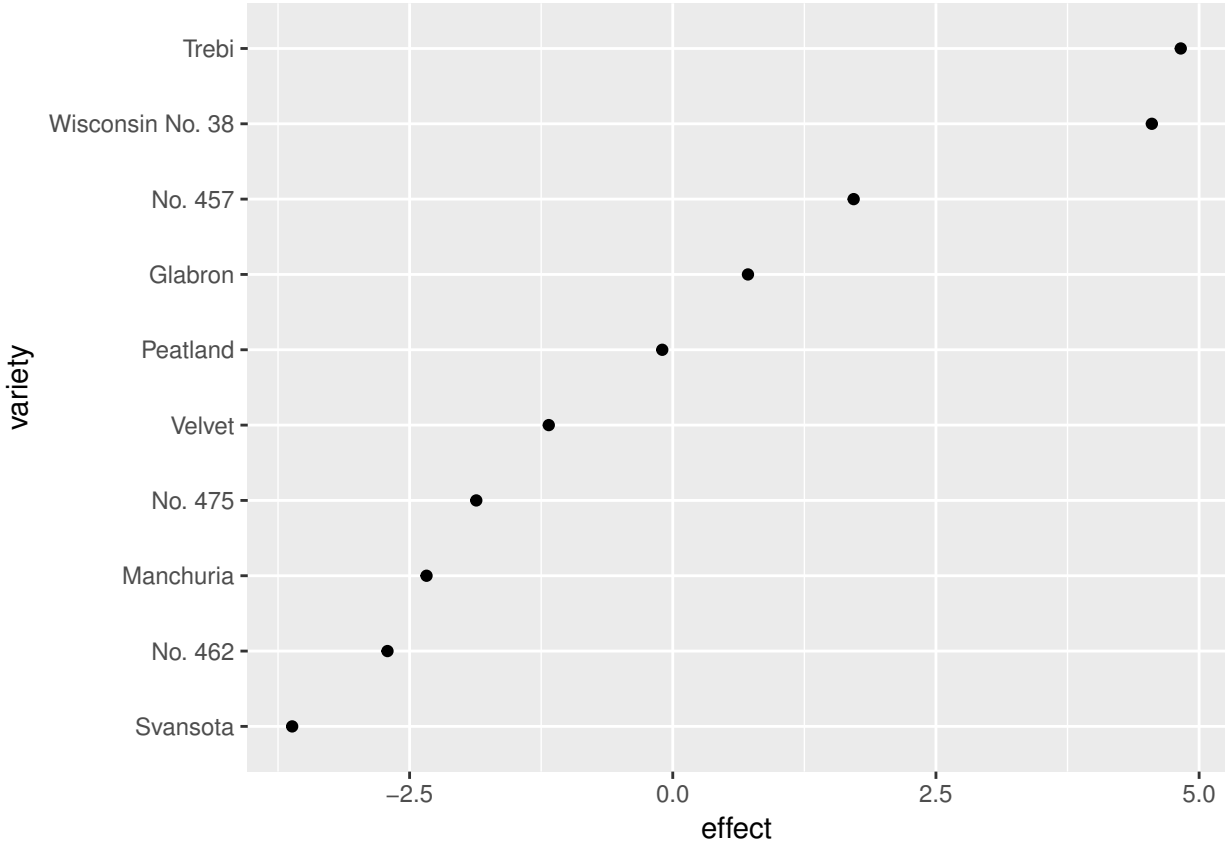
```
barley.effects = dummy.coef(barley.rlm)
year.site.main = outer(barley.effects$year, barley.effects$site, "+")
year.site.inter = barley.effects$"year:site"
year.site.effect = year.site.inter + as.vector(year.site.main)
years = rep(row.names(year.site.main), 6)
sites = rep(colnames(year.site.main), each = 2)
sites = factor(sites, levels = names(barley.effects$site))
year.site.df = data.frame(year = years, site = sites, effect = year.site.effect)
ggplot(year.site.df, aes(x = effect, y = site, col = year)) + geom_point()
```



As we know, Morris doesn't fit the pattern. Even excluding Morris, there's still quite a bit of variation between sites, both within years and in the difference from 1932 to 1931.

Let's now look at the variety effects:

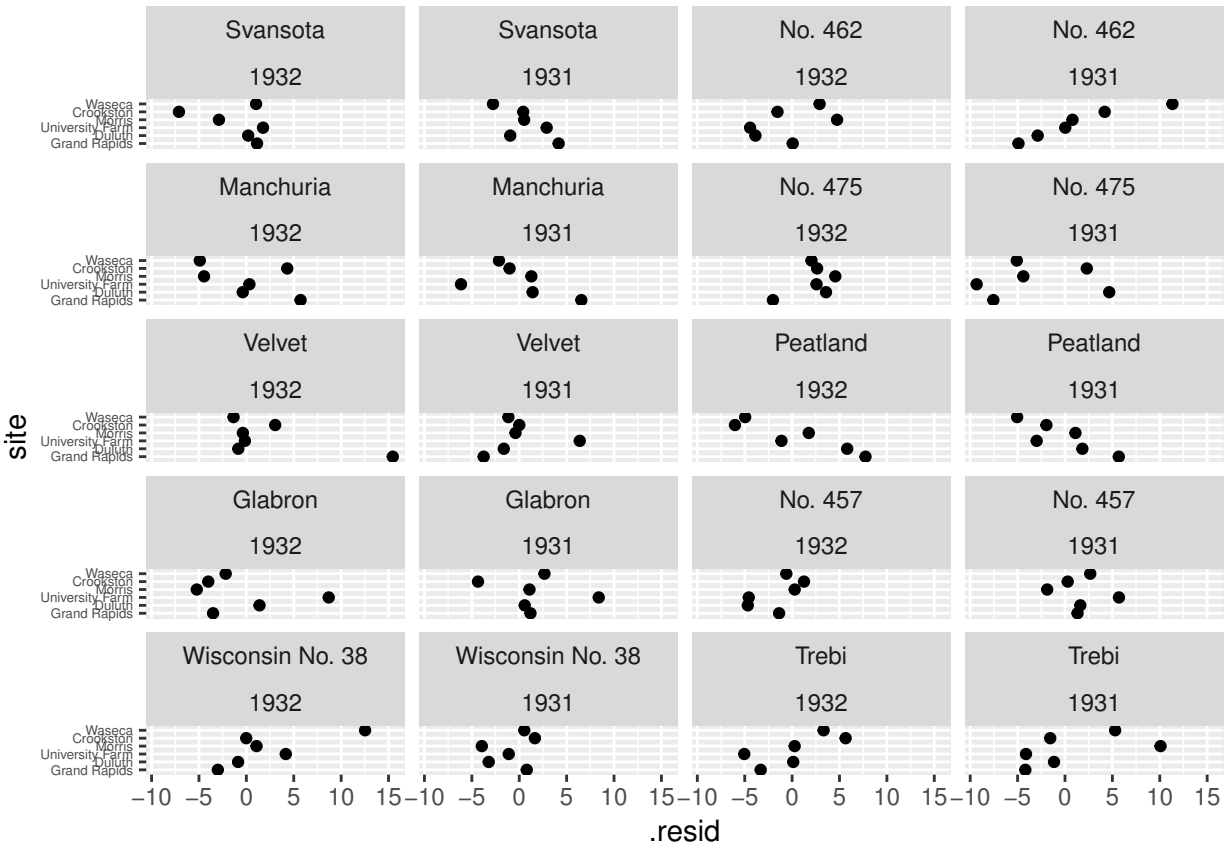
```
variety.effects = sort(barley.effects$variety)
varieties = factor(names(barley.effects$variety), levels = names(barley.effects$variety))
variety.df = data.frame(effect = variety.effects, variety = varieties)
ggplot(variety.df, aes(x = effect, y = variety)) + geom_point()
```



You'd need to know more about pre-World War II strains of barley than I do to get the most out of this graph. Generally though we see that the effects are fairly well-behaved. (You might at least think about fitting a multilevel model, though there's not much data and robust multilevel models are a pain in R.)

Let's now check the residuals for anomalies. Facet by variety and year:

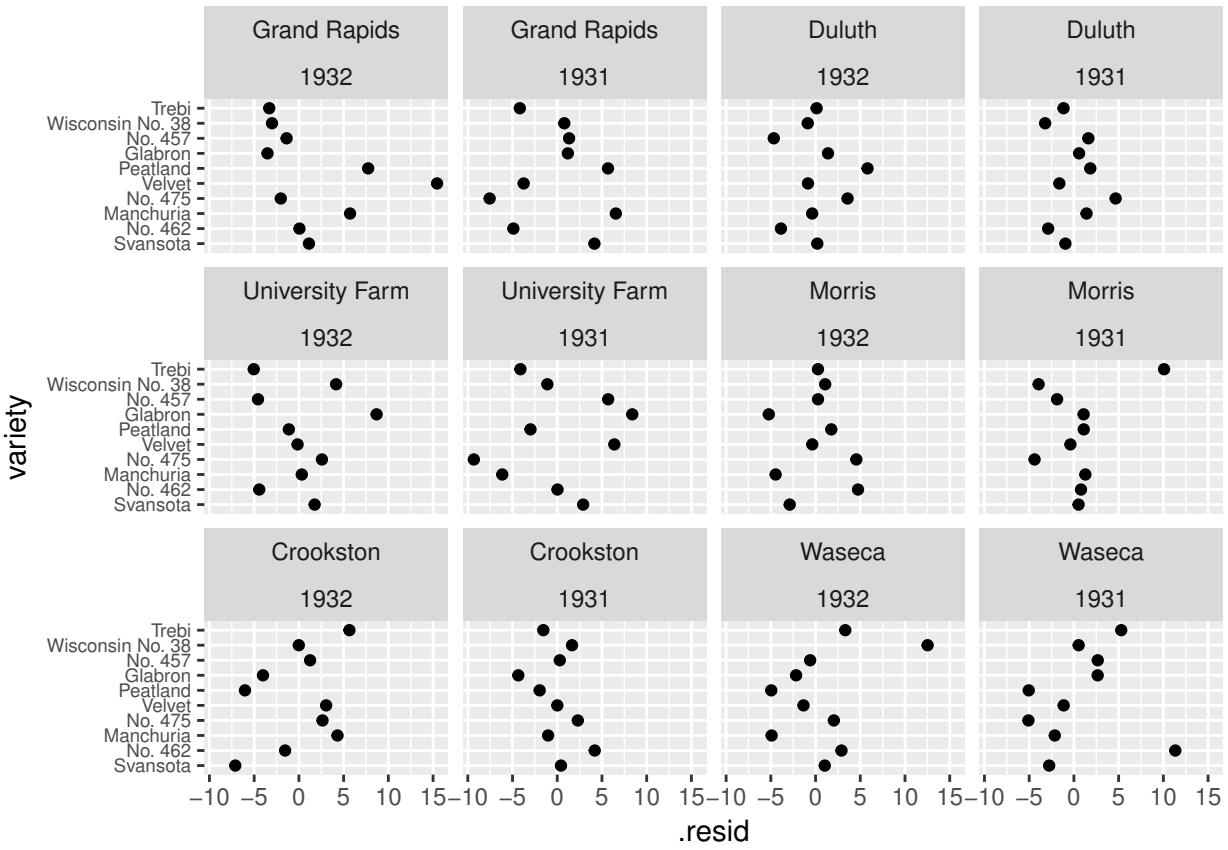
```
ggplot(barley.rlm.df, aes(x = .resid, y = site)) + geom_point() + facet_wrap(~variety +
    year, ncol = 4) + theme(axis.text.y = element_text(size = 5))
```



Looking carefully, there probably is a bit of a site:variety interaction: the Peatland plots look fairly similar for both 1931 and 1932, for example. But this effect is pretty small, so we ignore it. Otherwise, there's not much of note here aside from an outlier or two.

Now facet by year and location:

```
ggplot(barley.rlm.df, aes(x = .resid, y = variety)) + geom_point() + facet_wrap(~site +
    year) + theme(axis.text.y = element_text(size = 7))
```



Again, not much pattern, which is good. Note that Morris looks unremarkable now: including the interaction gets rid of the anomaly.

6.3.2 Was Morris a mistake?

The best way to find out is to look at more data. In fact, (incomplete) data from 1927 to 1936 is available in the file `minnesota.barley.yield.txt`. Sounds like a good topic for a problem set, doesn't it…