Chapter 8

GLMs

8.1 Count data, mostly

Optional reading: Gelman & Hill pp. 110–116, 118–119.

8.1.1 Probit

(This probably should've been in the logistic section.) Some fields prefer probit regression to logistic regression for count data. Good for them. In logistic regression, the probability that $Y = 1$ is the inverse logit of a linear predictor. In probit regression, the probability is the inverse standard normal CDF (i.e. the standard normal quantile) of a linear predictor:

$$P(Y = 1|x) = \Phi(\beta_0 + \beta_1 x)$$

When you have binary data, you can specify either `logit` or `probit` as your *link* function in your call to `glm()`. `logit` is the default. The coefficients for probit are smaller in magnitude (by about 40%) than in logistic regression:

```
wells = read.table("wells.dat")
glm(switch ~ dist, family = binomial(link = "logit"), data = wells)
```

```
##
## Call:  glm(formula = switch ~ dist, family = binomial(link = "logit"),
##     data = wells)
##
## Coefficients:
## (Intercept)         dist
##    0.605959    -0.006219
##
## Degrees of Freedom: 3019 Total (i.e. Null);  3018 Residual
## Null Deviance:       4118
## Residual Deviance: 4076  AIC: 4080
```

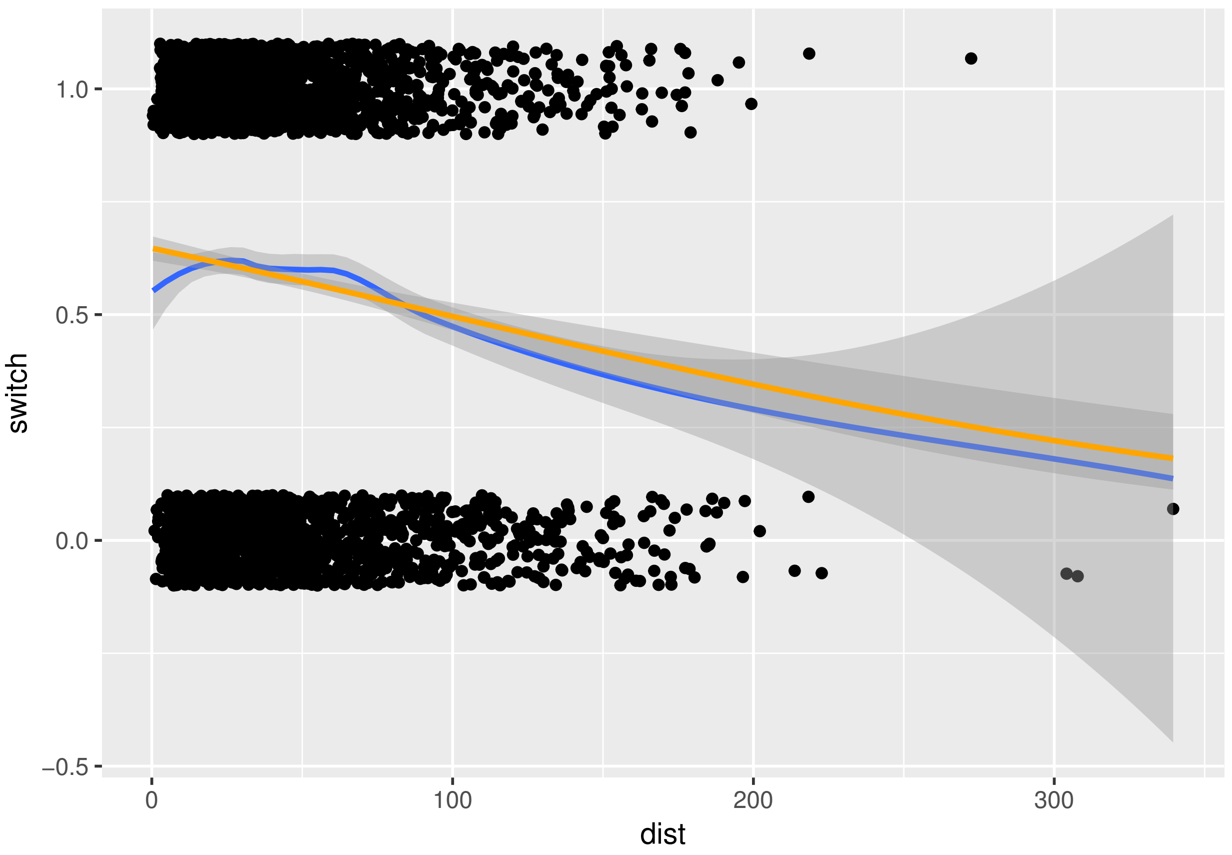
```
glm(switch ~ dist, family = binomial(link = "probit"), data = wells)
```

```
##
## Call:  glm(formula = switch ~ dist, family = binomial(link = "probit"),
##     data = wells)
##
```

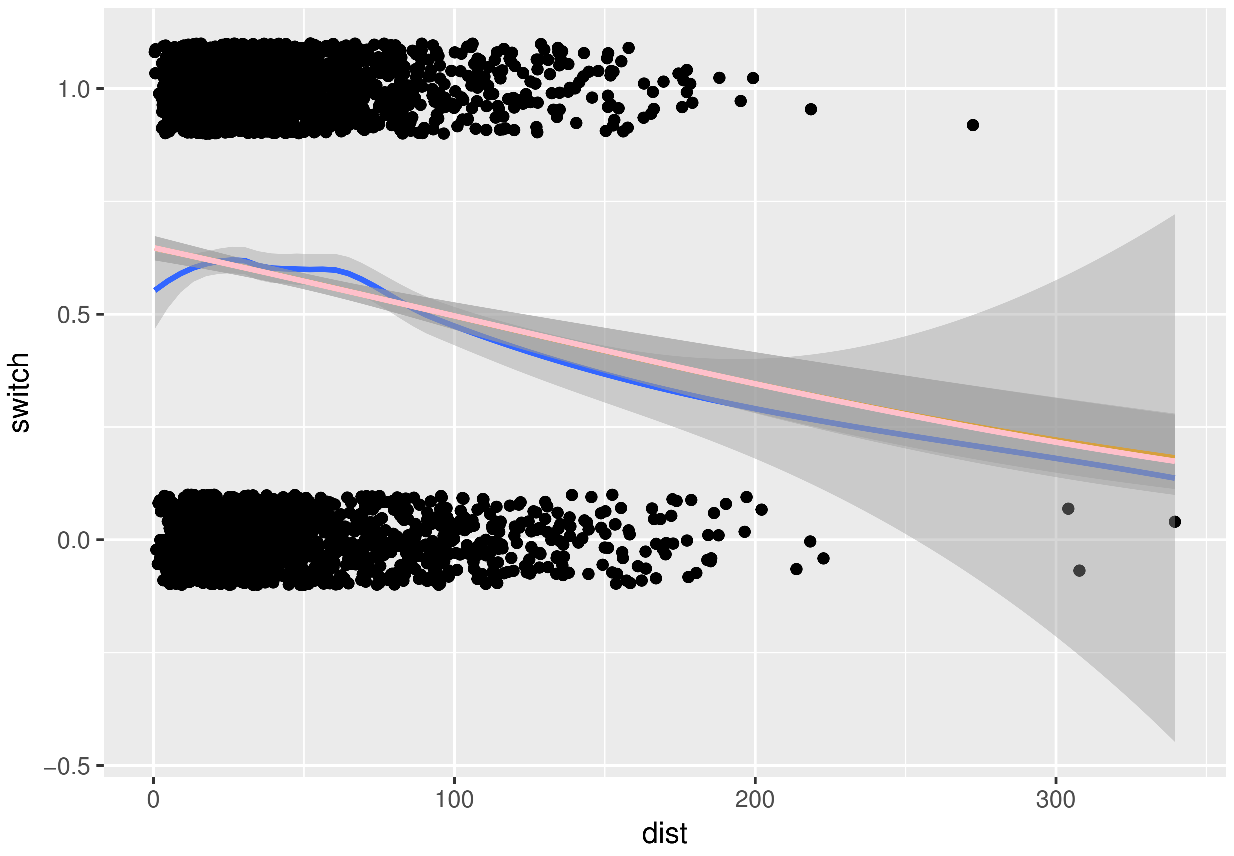
```
## Coefficients:
## (Intercept)         dist
##    0.377808    -0.003874
##
## Degrees of Freedom: 3019 Total (i.e. Null);  3018 Residual
## Null Deviance:       4118
## Residual Deviance: 4076  AIC: 4080
```

In terms of the fit, it rarely makes much difference:

```
# library(tidyverse)
wells.gg = ggplot(wells, aes(x = dist, y = switch)) + geom_jitter(width = 0,
    height = 0.1) + geom_smooth(method = "loess") + geom_smooth(method = "glm",
    method.args = list(family = binomial), color = "orange")
wells.gg
```



```
wells.gg + geom_smooth(method = "glm", method.args = list(family = binomial(link = "probit")),
    color = "pink")
```



If you need to choose between logit and probit, in general you should just use whatever everyone else in your field uses. It doesn't matter much, whereas using a nonparametric model, for example, might make a much more substantive difference.

8.1.2 Count data: Stop and frisk

Gelman and Hill have data on police stops in New York City in 1998–1999, during Giuliani's mayoralty. There have been accusations that some ethnic groups have been stopped at rates not justified by either their arrest rate or their location (as measured by precinct.) We'll model this data using (at first) **Poisson regression**, another form of GLM. In a standard Poisson regression, the response has a Poisson distribution with the *log* of the expected value given by a linear function of the predictors. In the single-variable case:

$$\log(E[Y|x]) = \beta_0 + \beta_1 x$$

The data, with noise added for confidentiality, is at

http://www.stat.columbia.edu/~gelman/arm/examples/police/frisk_with_noise.dat

The first few rows of this file are a description, so we tell R to skip these when reading the data.

```
frisk = read.table("http://www.stat.columbia.edu/~gelman/arm/examples/police/frisk_with_noise.dat",
    skip = 6, header = TRUE)
nrow(frisk)
```

```
## [1] 900
```

```
summary(frisk)
```

```
##      stops            pop          past.arrests       precinct       eth
##  Min.   :   0   Min.   :   321   Min.   :   0.0   Min.   : 1   Min.   :1
##  1st Qu.:  26   1st Qu.:  6844   1st Qu.:  53.0   1st Qu.:19   1st Qu.:1
##  Median :  72   Median : 18004   Median : 124.0   Median :38   Median :2
##  Mean   : 146   Mean   : 30105   Mean   : 262.8   Mean   :38   Mean   :2
##  3rd Qu.: 173   3rd Qu.: 46669   3rd Qu.: 287.5   3rd Qu.:57   3rd Qu.:3
##  Max.   :1755   Max.   :184345   Max.   :2655.0   Max.   :75   Max.   :3
##      crime
##  Min.   :1.00
##  1st Qu.:1.75
##  Median :2.50
##  Mean   :2.50
##  3rd Qu.:3.25
##  Max.   :4.00
```

The data gives counts of police stops for all combinations of 75 precincts, three ethnicities of the person stopped (1 = black, 2 = Hispanic, 3 = white), and four types of crime (violent, weapons, property, and drug,) for a total of $75 \times 3 \times 4 = 900$ rows. The other two variables are population of the ethnic group within the precinct and the number of arrests of people in that ethnic group in that precinct for that type of crime in 1997.

To simplify matters, we'll ignore the type of crime, and aggregate the number of stops and past arrests over all four types.

```
frisk.sum = aggregate(cbind(past.arrests, stops) ~ precinct + eth, sum, data = frisk)
nrow(frisk.sum)
```

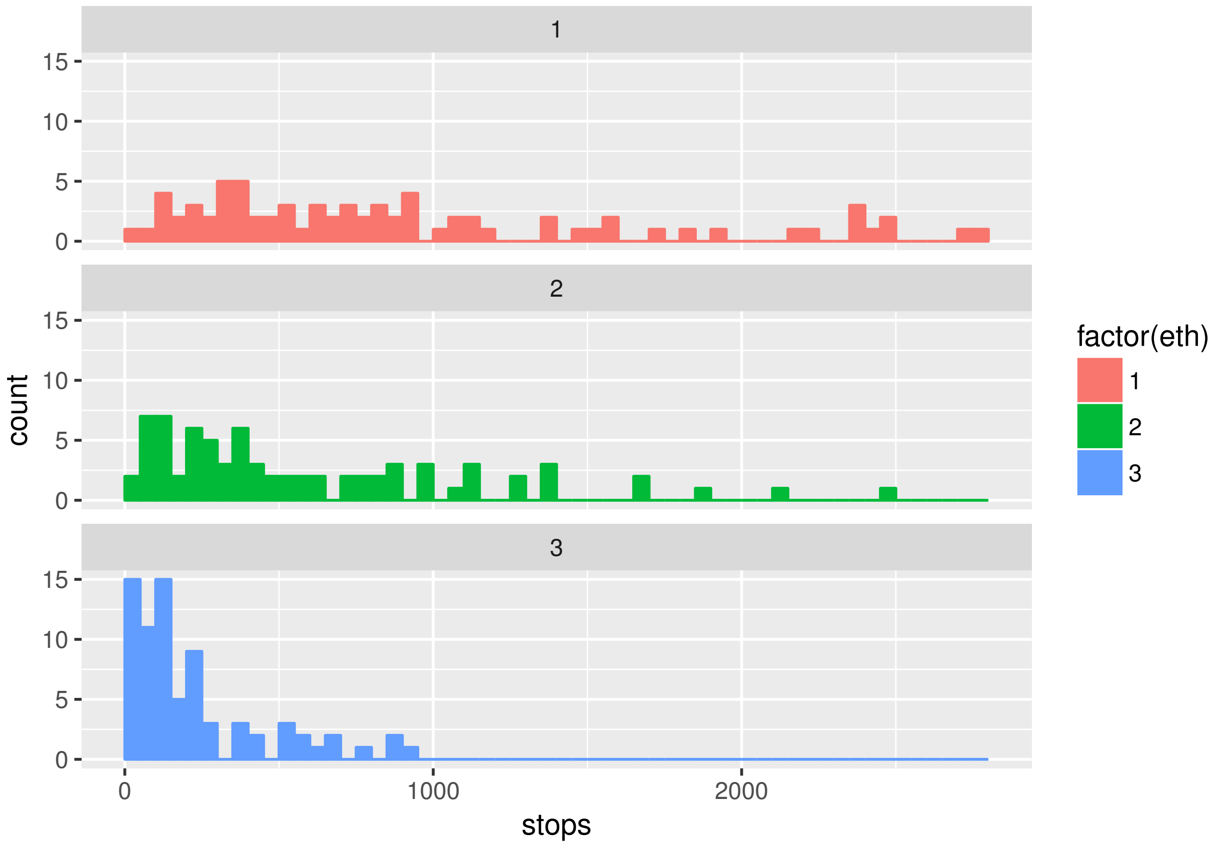
```
## [1] 225
```

```
summary(frisk.sum)
```

```
##     precinct       eth     past.arrests      stops
##  Min.   : 1   Min.   :1   Min.   :  16   Min.   :   7.0
##  1st Qu.:19   1st Qu.:1   1st Qu.: 312   1st Qu.: 133.0
##  Median :38   Median :2   Median : 571   Median : 385.0
##  Mean   :38   Mean   :2   Mean   :1051   Mean   : 584.1
##  3rd Qu.:57   3rd Qu.:3   3rd Qu.:1467   3rd Qu.: 824.0
##  Max.   :75   Max.   :3   Max.   :5667   Max.   :2771.0
```

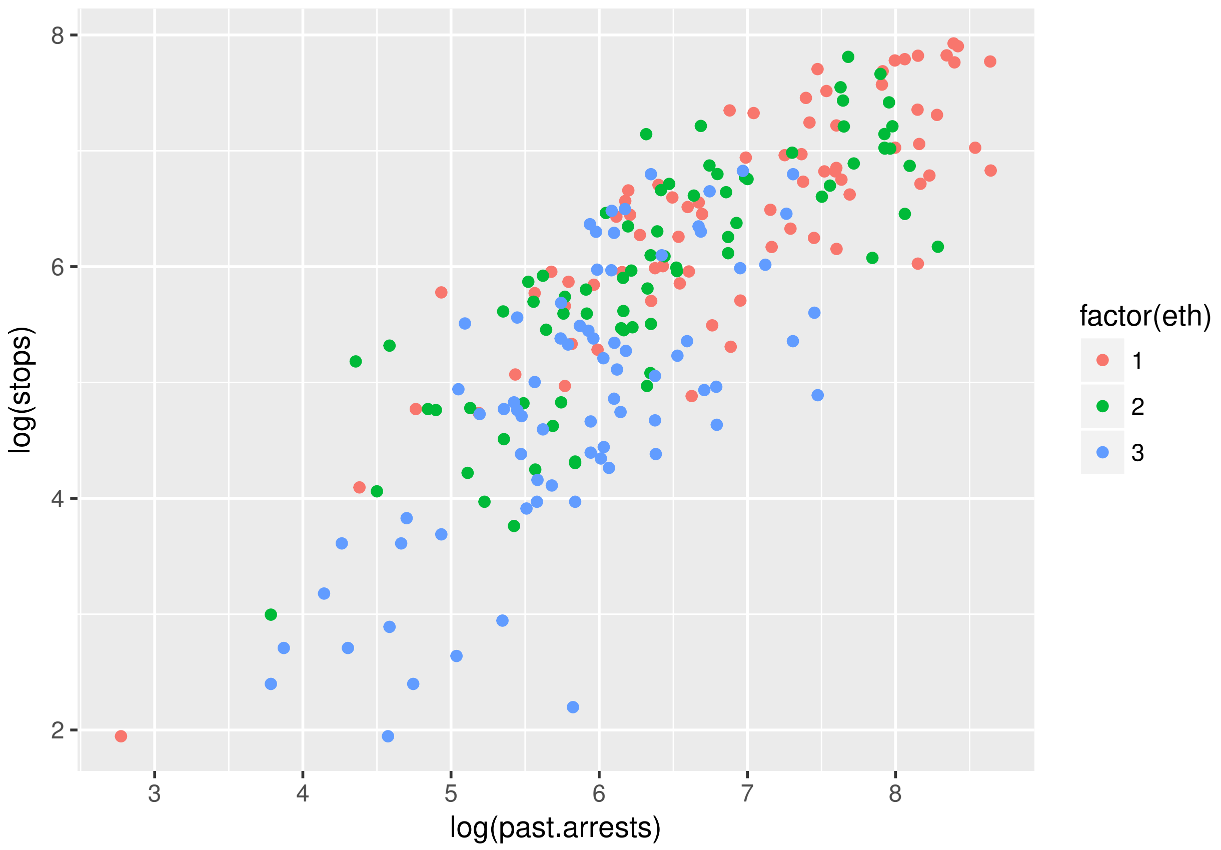
We now have 225 rows (75 precincts $\times$ 3 ethnic groups.) Let's first draw some pictures.

```
ggplot(frisk.sum, aes(x = stops, color = factor(eth), fill = factor(eth))) +
    geom_histogram(breaks = seq(0, 2800, 50)) + facet_wrap(~eth, ncol = 1)
```



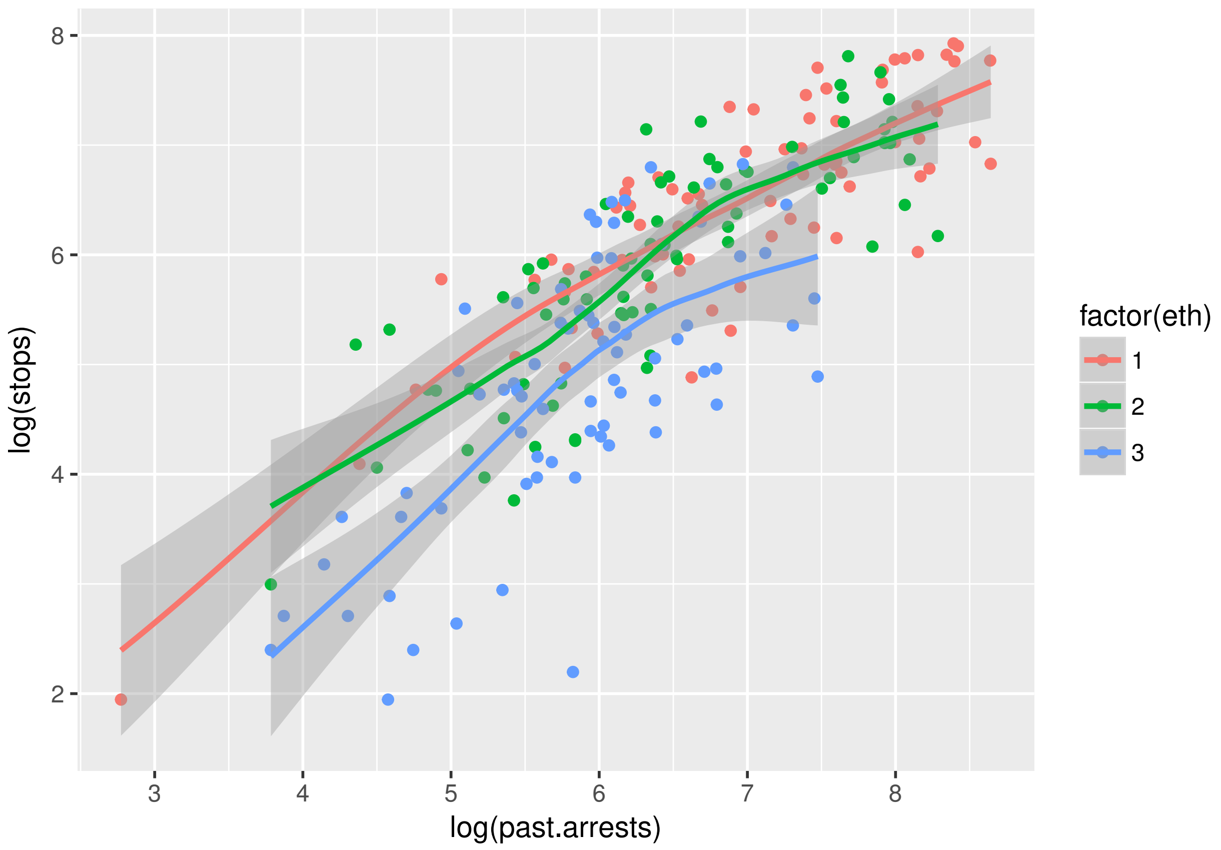
Quite clearly, the distributions of stops for black and Hispanic people are very different from the distribution for white people, though there may be multiple explanations for this. Let's look at the relationship of stops with past arrests. Because of skewness, we log both variables.

```r
ggplot(frisk.sum, aes(x = log(past.arrests), y = log(stops), color = factor(eth))) +
    geom_point()
```



There's certainly a relationship. The question is whether the relationship between the two variables is sufficient to explain the differences between the stops of the three ethnic groups. You could get at this just by adding smoother for the three groups:

```r
ggplot(frisk.sum, aes(x = log(past.arrests), y = log(stops), group = eth, color = factor(eth))) +
    geom_point() + geom_smooth(method.args = list(degree = 1))
```



Since this is an important topic, however, we should be a bit more careful and construct a model.

8.1.3 Poisson regression

We'll start off with a Poission regression model that's much too simple, and build up to a more useful one. The simplest model just treats each number of stops as a realization of a Poisson random variable.

```
constant.glm = glm(stops ~ 1, family = poisson, data = frisk.sum)
summary(constant.glm)
```

```
##
## Call:
## glm(formula = stops ~ 1, family = poisson, data = frisk.sum)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -33.049  -22.552   -8.788    9.343   65.227
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 6.370053   0.002758    2309   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 123333  on 224  degrees of freedom
## Residual deviance: 123333  on 224  degrees of freedom
## AIC: 125041
##
## Number of Fisher Scoring iterations: 5
```

I'm getting sick of all the cruft that gets displayed when we use `summary()` on a GLM. From now on let's use the `display()` function in package `arm` instead.

```
# install.packages(arm)
library(arm)
display(constant.glm)
```

```
## glm(formula = stops ~ 1, family = poisson, data = frisk.sum)
##             coef.est coef.se
## (Intercept) 6.37     0.00
## ---
##   n = 225, k = 1
##   residual deviance = 123332.5, null deviance = 123332.5 (difference = 0.0)
```

This pares away most of the low value information. We see the coefficent estimate (on the log scale) is 6.37, which gives $e^6.37 = 584$ on the original scale. That is, the number of stops for each ethnic group within each precinct is modeled as a random variable with distribution

$$\text{Poisson}(584).$$

The other number to keep track of is the (residual) *deviance*. Low deviance is good, as long as you're not overfitting. In particular, every time you add a degree of freedom, you should expect reduce the deviance by 1 if you're just adding random noise. So if you're not overfitting when you fit a complex model, you should expect to reduce the deviance by more than you increase the degrees of freedom.

Now this model is obviously wrong. We might, for example, think that the number of stops for an ethnic groups in a precinct should be proportional to the number of arrests for that ethnicity-precinct (though this is controversial.) In a GLM, we can model this using an **offset**:

```
offset.glm = glm(stops ~ 1, family = poisson, offset = log(past.arrests), data = frisk.sum)
display(offset.glm)
```

```
## glm(formula = stops ~ 1, family = poisson, data = frisk.sum,
##     offset = log(past.arrests))
##             coef.est coef.se
## (Intercept) -0.59    0.00
## ---
##   n = 225, k = 1
##   residual deviance = 46120.3, null deviance = 46120.3 (difference = 0.0)
```

Since the linear predictor is on the log scale, the offset also has to be logged. This gives the model

$$\log[E(\text{stops}|\text{past arrests})] = -0.59 + \log(\text{past arrests})$$

or (taking the exponential of both sides)

$$E(\text{stops}|\text{past arrests}) = e^{-0.59+\log(\text{past arrests})} = 0.56 \times \text{past arrests}$$

The deviance of this model is much lower than the constant model, so we've improved the fit by a lot.

Now we want to see what happens if we add ethnic group as a predictor. Ethnic group is categorical, so we use it as a factor.

```
eth.glm = glm(stops ~ factor(eth), family = poisson, offset = log(past.arrests),
    data = frisk.sum)
display(eth.glm)
```

```
## glm(formula = stops ~ factor(eth), family = poisson, data = frisk.sum,
##     offset = log(past.arrests))
##               coef.est coef.se
## (Intercept)  -0.59     0.00
## factor(eth)2  0.07     0.01
## factor(eth)3 -0.16     0.01
## ---
##   n = 225, k = 3
##   residual deviance = 45437.4, null deviance = 46120.3 (difference = 682.9)
```

The deviance has dropped substantially again. The model is now

$$E(\text{stops}) = \text{multiplier for ethnic group} \times \text{past arrests}$$

where the multipliers are

```
eth.co = coefficients(eth.glm)
multipliers = exp(c(eth.co[1], eth.co[1] + eth.co[2], eth.co[1] + eth.co[3]))
print(multipliers)
```

```
## (Intercept) (Intercept) (Intercept)
##   0.5553894   0.5957836   0.4725238
```

for black, Hispanic, and white respectively.

So far we have shown that black and Hispanic people were stopped at a proportionately higher fraction of their arrest rate compared to white people. However, as the data isn't from a randomized experiment, there may be confounding. For example, black and Hispanic people generally live in precincts with higher stop rates. (Whether this is in itself evidence of bias is again, controversial.) Since this is exploratory work, we won't attempt to prove cause-and-effect, but we'll see what happens if we include precinct as an explanatory variable.

```
precinct.glm = glm(stops ~ factor(eth) + factor(precinct), family = poisson,
    offset = log(past.arrests), data = frisk.sum)
```

We won't print out the full results because we now have a coefficient for each precinct. Let's just first check the deviance has gone down significantly:

```
deviance(eth.glm)
```

```
## [1] 45437.35
```

Now look at the first few coefficients (and their standard errors):

```
coefficients(summary(precinct.glm))[1:6, 1:2]
```

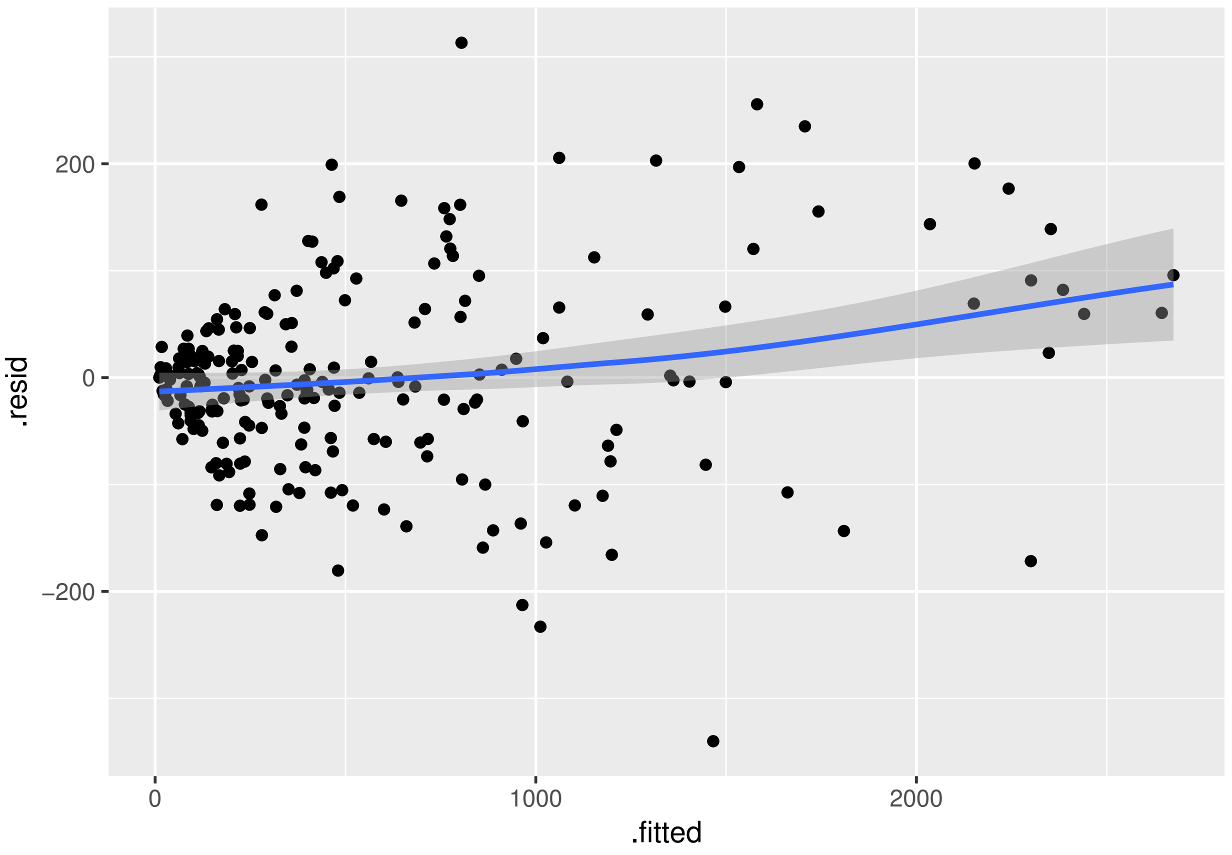
```
##                       Estimate  Std. Error
## (Intercept)        -1.37886803 0.051019006
## factor(eth)2        0.01018798 0.006802045
## factor(eth)3       -0.41900122 0.009434996
## factor(precinct)2 -0.14904964 0.074030344
## factor(precinct)3  0.55995498 0.056758425
## factor(precinct)4  1.21063605 0.057548994
```

After controlling for precinct, the differences between the white and minority coefficients becomes even bigger.

8.1.4 Checking the model

We first plot the residuals against the fitted values on the response (original) scale, and see what happens.

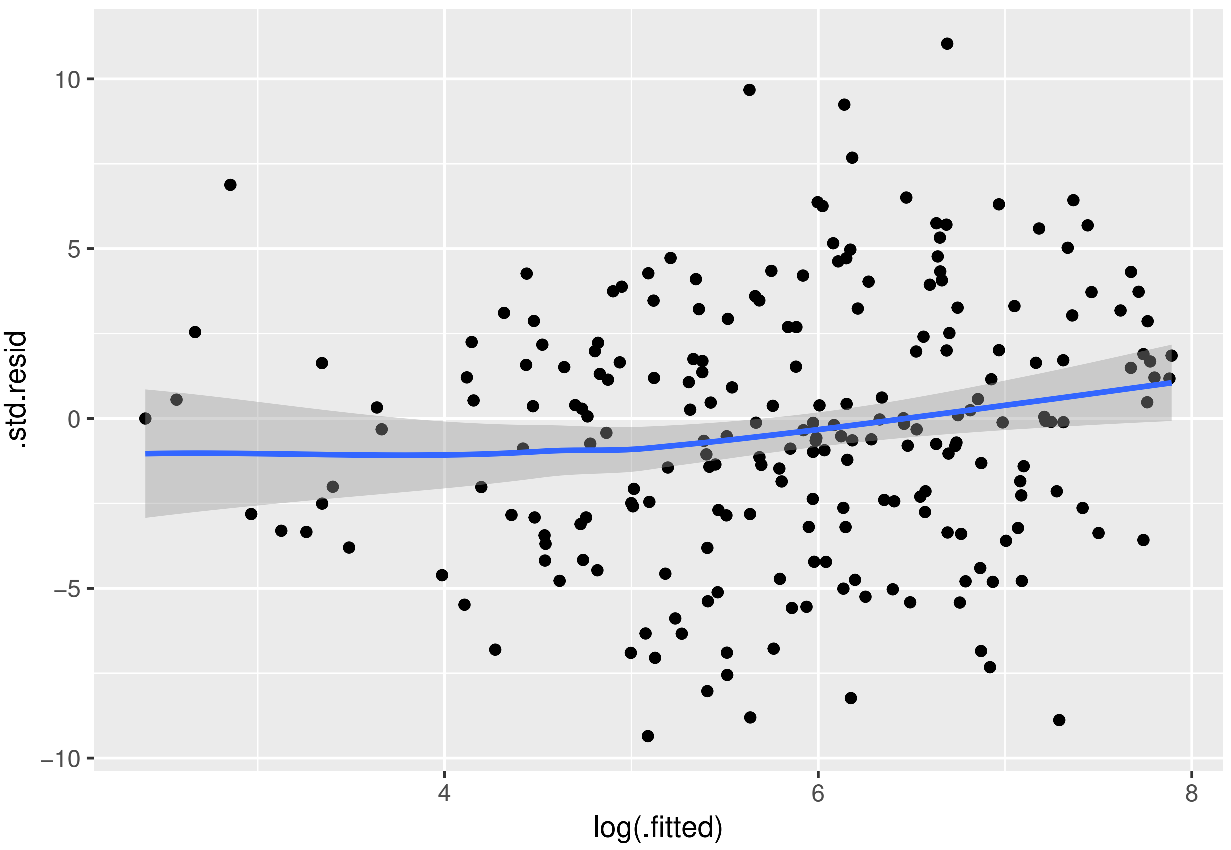
```
precinct.fitted = fitted.values(precinct.glm)
precinct.resid = residuals(precinct.glm, type = "response")
precinct.glm.df = data.frame(frisk.sum, .fitted = precinct.fitted, .resid = precinct.resid)
ggplot(precinct.glm.df, aes(x = .fitted, y = .resid)) + geom_point() + geom_smooth(span = 1,
    method.args = list(degree = 1))
```



The smoother we added isn't flat, but that could just be because the residuals are heteroskedastic: they spread out dramatically. The heteroskedasticity is not a bug: Poissons are supposed to be heteroskedastic. Recall that a Poisson(λ) random variable has variance λ and standard deviation $\sqrt{\lambda}$. So the typical size of the residuals should go up as the square root of the fitted value.

To hopefully remove this effect, we create **standardized residuals** by dividing the raw residuals by the square root of the fitted value. We plot these against the log fitted values to reduce the distortions caused by skewness.

```
precinct.std.resid = precinct.resid/sqrt(precinct.fitted)
precinct.glm.df$.std.resid = precinct.std.resid
ggplot(precinct.glm.df, aes(x = log(.fitted), y = .std.resid)) + geom_point() +
    geom_smooth(span = 1, method.args = list(degree = 1))
```



This is better, though far from perfect. There's still some nonlinearity left in the smoother, though the amount is relatively small. If prediction was the goal, a nonparametric model would probably provide an improvement.

8.1.5 Overdispersion

If you care about more than just the conditional expectation, however, we find a bigger problem. If the Poisson model were correct, the standardized residuals should be on a similar scale to the standard normal – that is, the vast majority should be within ± 2. From the previous graph, that's clearly not the case.

We need to measure the **overdispersion** in the data. We could do a formal χ^2 test for overdispersion, but instead, let's calculate the typical size of the squared residuals. (When we "average", we divide the sum by the residual degrees of freedom.) If the Poisson model is correct, this should be close to 1. If it's much more than 1, we need a better model.

```
overdispersion = sum(precinct.std.resid^2)/df.residual(precinct.glm)
overdispersion
```

```
## [1] 21.88505
```

This is much more than 1. In fact, this happens most of the time with count data – the data is usually more dispersed than the Poisson model.

8.1.6 How bad is it?

We know there are problems with our model. But are they so bad that we can't draw conclusions from it?

One simple way of checking is to simulate a fake set of data, and see if it closely resembles the actual set. For a Poisson model, this is easy. We know according to the model, each observation is a realization of a Poisson random variable, whose parameter is given by the fitted value. Then we can use `rpois()` to do simulation and do numerical summaries and plots. (Of course, we could repeat the simulation multiple times if necessary, but we'll just do it once.)

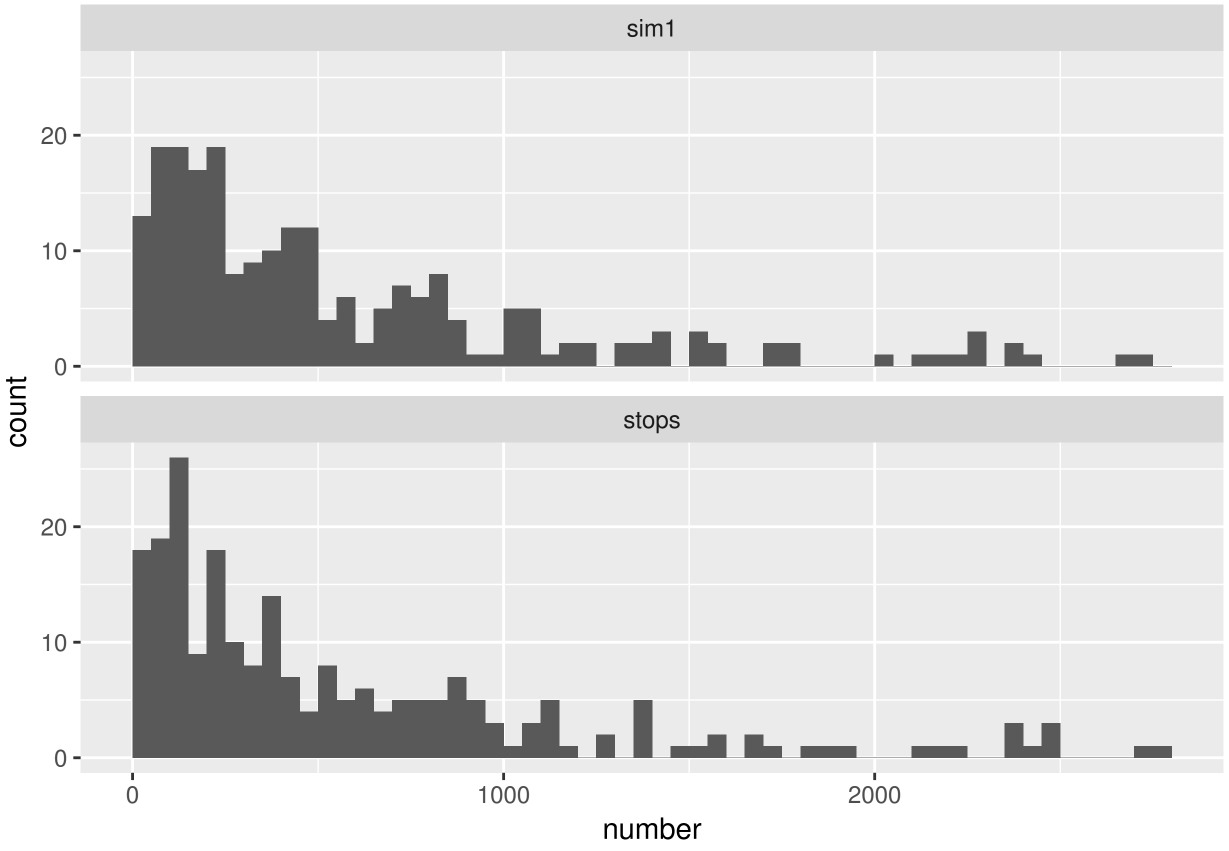
```
sim1 = rpois(nrow(frisk.sum), lambda = fitted.values(precinct.glm))
summary(frisk.sum$stops)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     7.0   133.0   385.0   584.1   824.0  2771.0
```

```
summary(sim1)
```

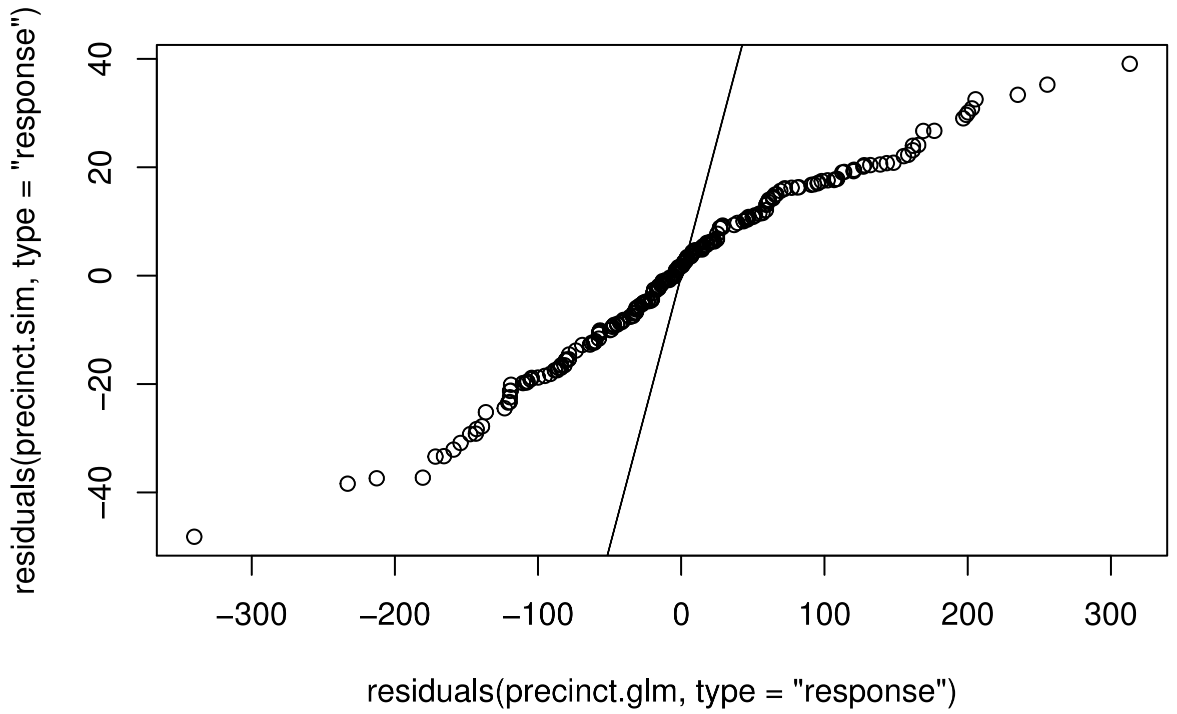
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      13     175     395     584     802    2705
```

```
sim.df = data.frame(frisk.sum, sim1)
sim.long = sim.df %>% gather(type, number, stops:sim1)
ggplot(sim.long, aes(x = number)) + geom_histogram(breaks = seq(0, 2800, 50)) +
    facet_wrap(~type, ncol = 1)
```



If we look at the histograms, there doesn't seem to be much difference. But what happens if we fit a model to the simulated data and look at its residuals? We'll find these and do a two-sample QQ plot of them against the original residuals (out of laziness we'll just draw the plot in base R.)

```
precinct.sim = glm(sim1 ~ factor(eth) + factor(precinct), family = poisson,
    offset = log(past.arrests), data = sim.df)
qqplot(residuals(precinct.glm, type = "response"), residuals(precinct.sim, type = "response"))
abline(0, 1)
```



If the model were correct, this QQ plot should be close to a line through the origin with slope 1. It ain't.

The simulation here is overkill, since we understand the Poisson fairly well and already know the data is overdispersed. However, the more complicated your model gets, the more useful this kind of simulation is as a sanity check.

8.1.7 Fixing overdispersion

The quickest fix is to use the **quasipoisson** family instead of the Poisson.

```
precinct.quasi = glm(stops ~ factor(eth) + factor(precinct), family = quasipoisson,
    offset = log(past.arrests), data = frisk.sum)
coefficients(summary(precinct.quasi))[1:6, 1:2]
```

```
##                       Estimate Std. Error
## (Intercept)        -1.37886803 0.23867441
## factor(eth)2        0.01018798 0.03182097
## factor(eth)3       -0.41900122 0.04413830
## factor(precinct)2 -0.14904964 0.34632483
## factor(precinct)3  0.55995498 0.26552425
## factor(precinct)4  1.21063605 0.26922265
```

Note that the coefficients look the same as they were in the standard Poisson case. However, their standard errors have been inflated by the square root of their overdispersion. We can confirm that the fitted values haven't changed:

```
quasi.fitted = fitted.values(precinct.quasi)
summary(quasi.fitted - precinct.fitted)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0       0       0       0       0       0
```

So the quasipoission doesn't change the fit, only the variance and the standard errors.

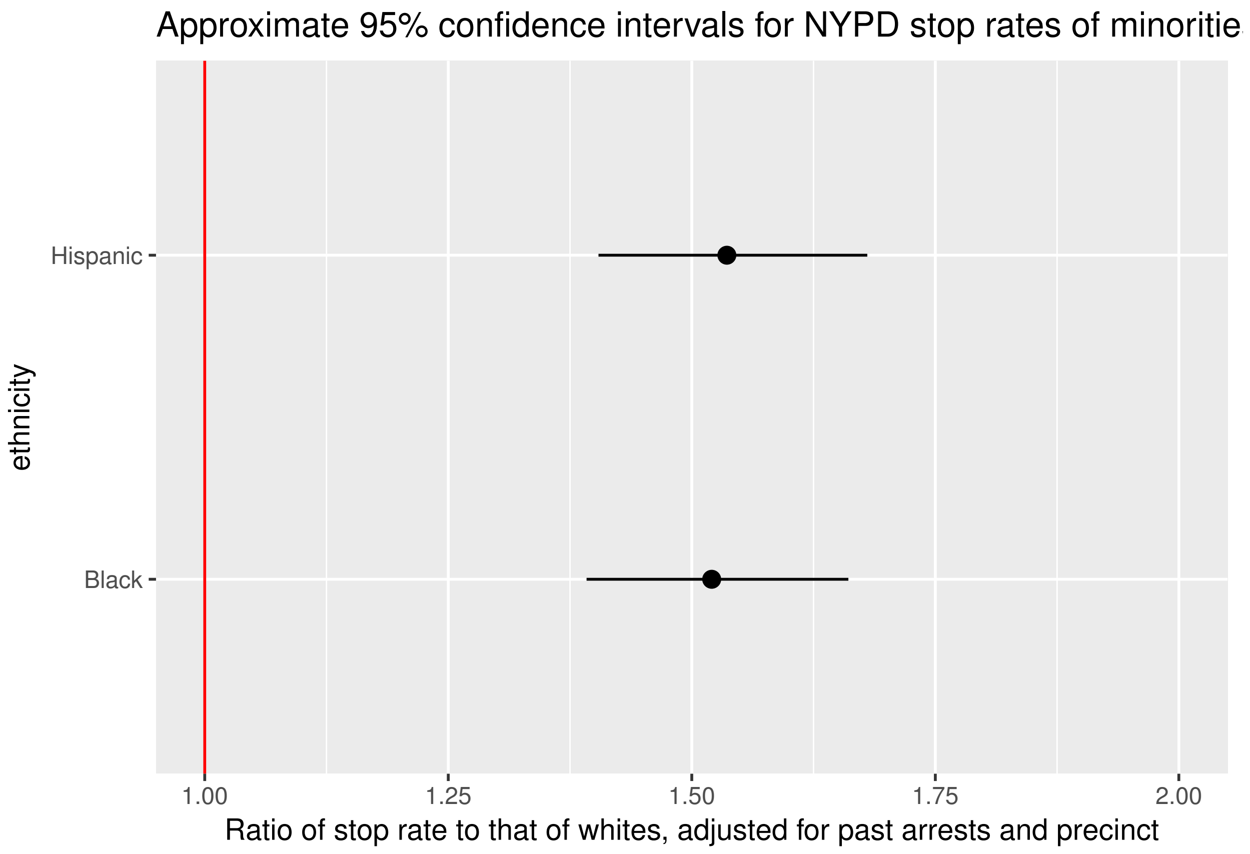
For interpretation, it may be useful to refit the model changing the order of levels in `eth` to use whites as a baseline.

```
precinct.quasi2 = glm(stops ~ factor(eth, levels = c(3, 1, 2)) + factor(precinct),
    family = quasipoisson, offset = log(past.arrests), data = frisk.sum)
coefficients(summary(precinct.quasi2))[1:6, 1:2]
```

```
##                                      Estimate Std. Error
## (Intercept)                        -1.7978692 0.24101846
## factor(eth, levels = c(3, 1, 2))1  0.4190012 0.04413830
## factor(eth, levels = c(3, 1, 2))2  0.4291892 0.04485195
## factor(precinct)2                  -0.1490496 0.34632483
## factor(precinct)3                   0.5599550 0.26552425
## factor(precinct)4                   1.2106361 0.26922265
```

We now back-transform to get intervals for the stop rates of blacks and Hispanics relative to whites, after adjusting for arrest rates and precinct.

```
eth.co = coefficients(summary(precinct.quasi2))[1:3, 1:2]
ethnicity = c("Black", "Hispanic")
estimate = exp(eth.co[2:3, 1])
lower = exp(eth.co[2:3, 1] - 2 * eth.co[2:3, 2])
upper = exp(eth.co[2:3, 1] + 2 * eth.co[2:3, 2])
eth.co.df = data.frame(ethnicity, estimate, lower, upper)
ggplot(eth.co.df, aes(x = ethnicity, y = estimate, ymin = lower, ymax = upper)) +
    geom_pointrange() + ylim(1, 2) + geom_abline(intercept = 1, slope = 0, color = "red") +
    ylab("Ratio of stop rate to that of whites, adjusted for past arrests and precinct") +
    ggtitle("Approximate 95% confidence intervals for NYPD stop rates of minorities") +
    coord_flip()
```



The confidence intervals don't include 1. This would be consistent with a hypothesis of bias against minorities, though we should think very carefully about other confounding variables before drawing a firm conclusion (e.g. type of crime, which we ignored.) You should check your model very thoroughly. A definitive answer here requires subject matter knowledge in conjunction with statistics.

8.1.8 Other fixes

There are lots of alternative approaches:

- **Negative binomial regression** is an alternative to the quasipoisson when the count data is overdispersed.
- Nonparametric approaches like loess and GAM can give you a better fit for the conditional expectation, at the cost of making inference much more complicated.
- A **multilevel model** has appeal here because of the large number of precincts. It can deal with overdispersion as well as regularize the estimates for the precincts. Such models are pretty complicated, though.

8.2 polr(): Ordered categorical regression

Optional reading: Gelman & Hill pp. 119–123.

With categorical regression, the main distinction is between models with **ordered** categories and models with **unordered** categories. Let's start with the ordered case.

8.2.1 Fake data: Grad school

Let's use the (simulated) data on the potential grad school application of college students at

http://stats.idre.ucla.edu/r/dae/ordinal-logistic-regression/

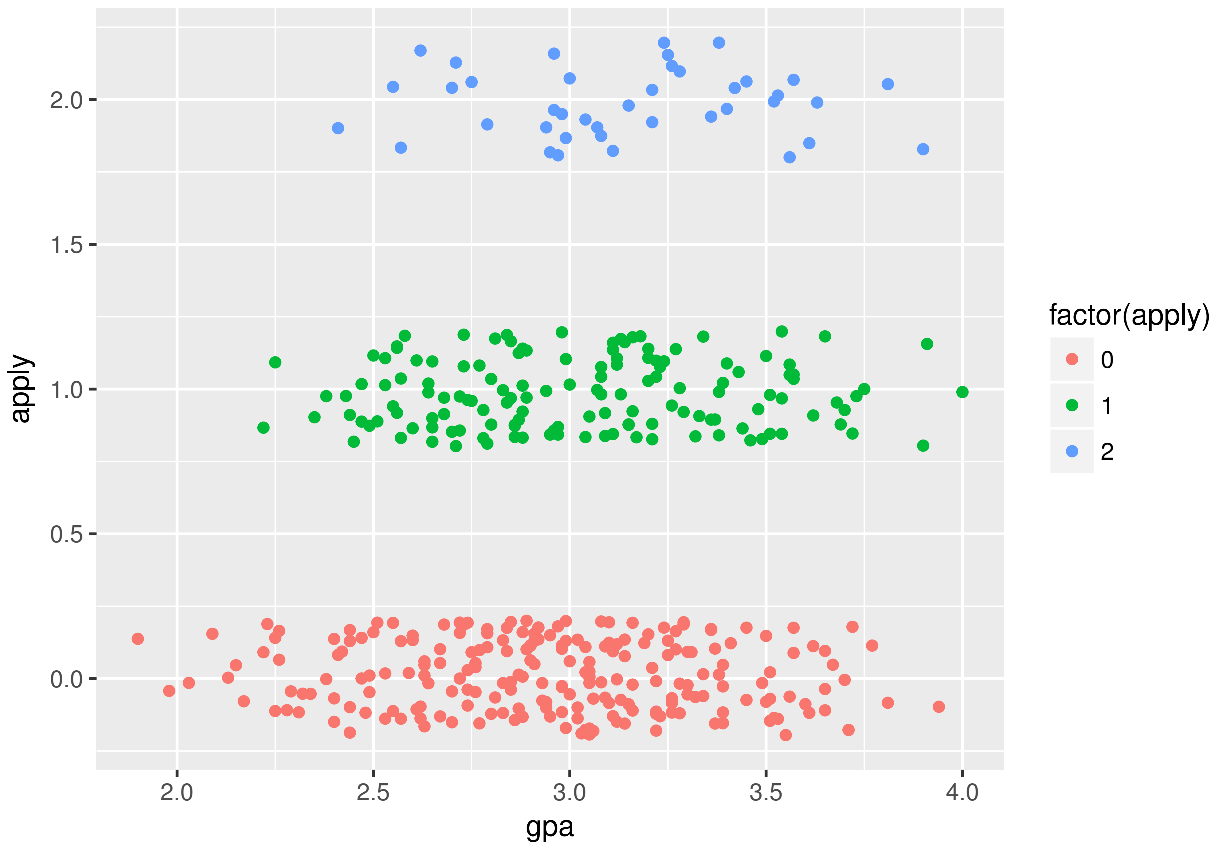
We'll read in the Stata data using `import()` in the `rio` package:

```r
# install.packages('rio')
library(rio)
gradschool = import("https://stats.idre.ucla.edu/stat/data/ologit.dta")
summary(gradschool)
```

```
##      apply          pared            public             gpa
##  Min.   :0.00   Min.   :0.0000   Min.   :0.0000   Min.   :1.900
##  1st Qu.:0.00   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:2.720
##  Median :0.00   Median :0.0000   Median :0.0000   Median :2.990
##  Mean   :0.55   Mean   :0.1575   Mean   :0.1425   Mean   :2.999
##  3rd Qu.:1.00   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:3.270
##  Max.   :2.00   Max.   :1.0000   Max.   :1.0000   Max.   :4.000
```

`apply` gives a student's intention to apply to grad school, where 0 means unlikely, 1 means somewhat likely, and 2 means very likely. That is, `apply` is an ordered categorical response. We'll use `gpa` as our initial explanatory variable. Draw a jittered dot plot:

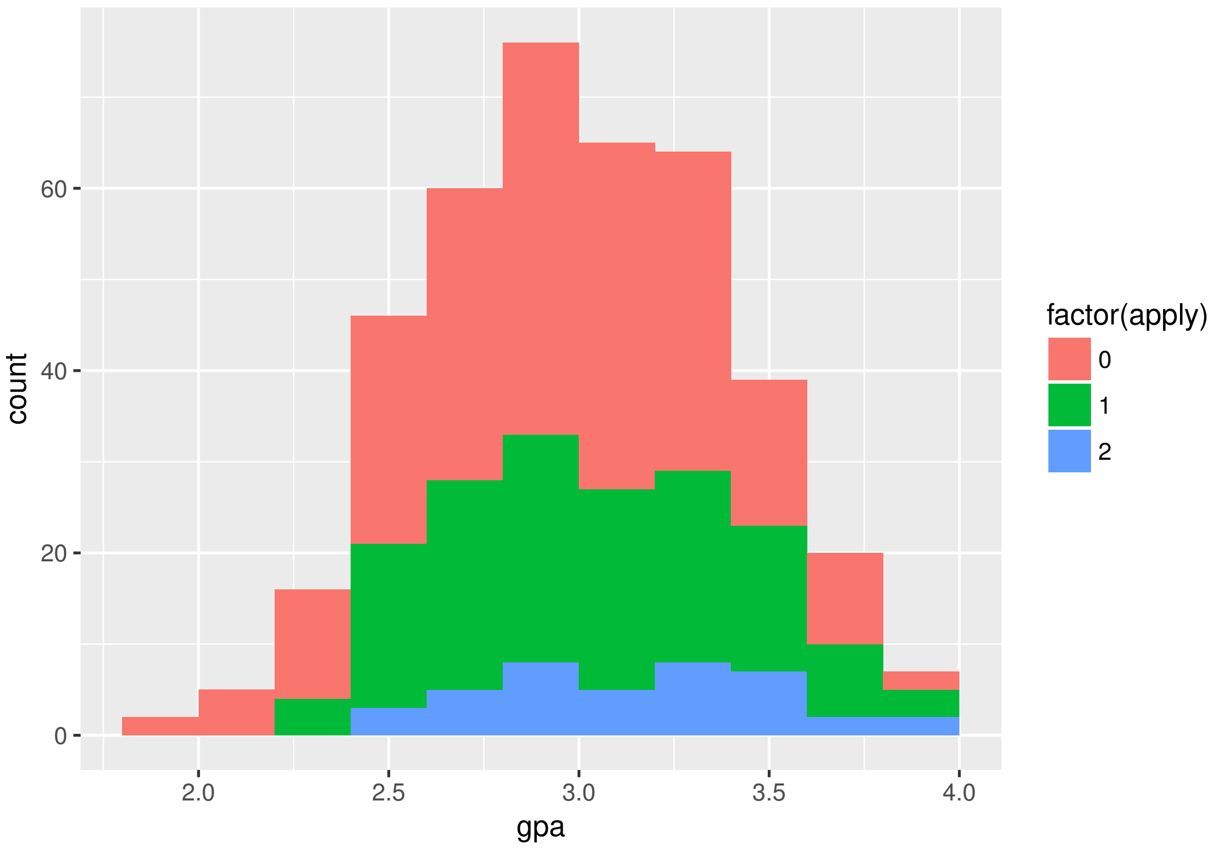
```r
# library(tidyverse)
ggplot(gradschool, aes(x = gpa, y = apply, color = factor(apply))) + geom_jitter(width = 0,
    height = 0.2)
```



Most students are unlikely to apply to grad school. However, a higher GPA does mean a student is more likely to apply.

It's easy to get a sense of the conditional distribution of `gpa` given `apply`. What we want, however, is the conditional distribution of `apply` given `gpa`. We can get at this by discretizing `gpa`, drawing a histogram, then coloring it by the levels of `apply`:

```r
ggplot(gradschool, aes(x = gpa, fill = factor(apply))) + geom_histogram(breaks = seq(1.8,
    4, 0.2))
```



However, discretizing is wasteful and arbitrary, so we're better off fitting a model.

There are lots of options for modeling this kind of data. The one we'll pursue is **proportional odds logistic regression**, fitted in R using the `polr()` function in `MASS`. Let's first fit the model, then explain what we got.

```
library(MASS)
gpa.polr = polr(factor(apply) ~ gpa, data = gradschool)
library(arm)
display(gpa.polr)
```

```
## polr(formula = factor(apply) ~ gpa, data = gradschool)
##     coef.est coef.se
## gpa 0.72     0.25
## 0|1 2.37     0.76
## 1|2 4.40     0.78
## ---
## n = 400, k = 3 (including 2 intercepts)
## residual deviance = 732.6, null deviance is not computed by polr
```

The model gives us both a **linear predictor** (on a logit scale) and **cutpoints**. The linear predictor is

$$0.72 \times \text{GPA}$$

(Note that the form of the model fitted by `polr()` has no intercept.) To get deterministic predictions, we compare the linear predictor to the cutpoints. The boundary between group 0 (unlikely) and group 1 (somewhat likely) is 2.37, while the boundary between group 1 and group 2 (very likely) is 4.4. (These can be extracted with `gpa.polr$zeta`.) Division tells us that this means a GPA below 3.28 gives a deterministic prediction of “unlikely”, a GPA between 3.28 and 6.07 gives a prediction of “somewhat likely”, and a GPA of above 6.07 gives a prediction of “very likely.” Of course, no one has a GPA above 6.07.

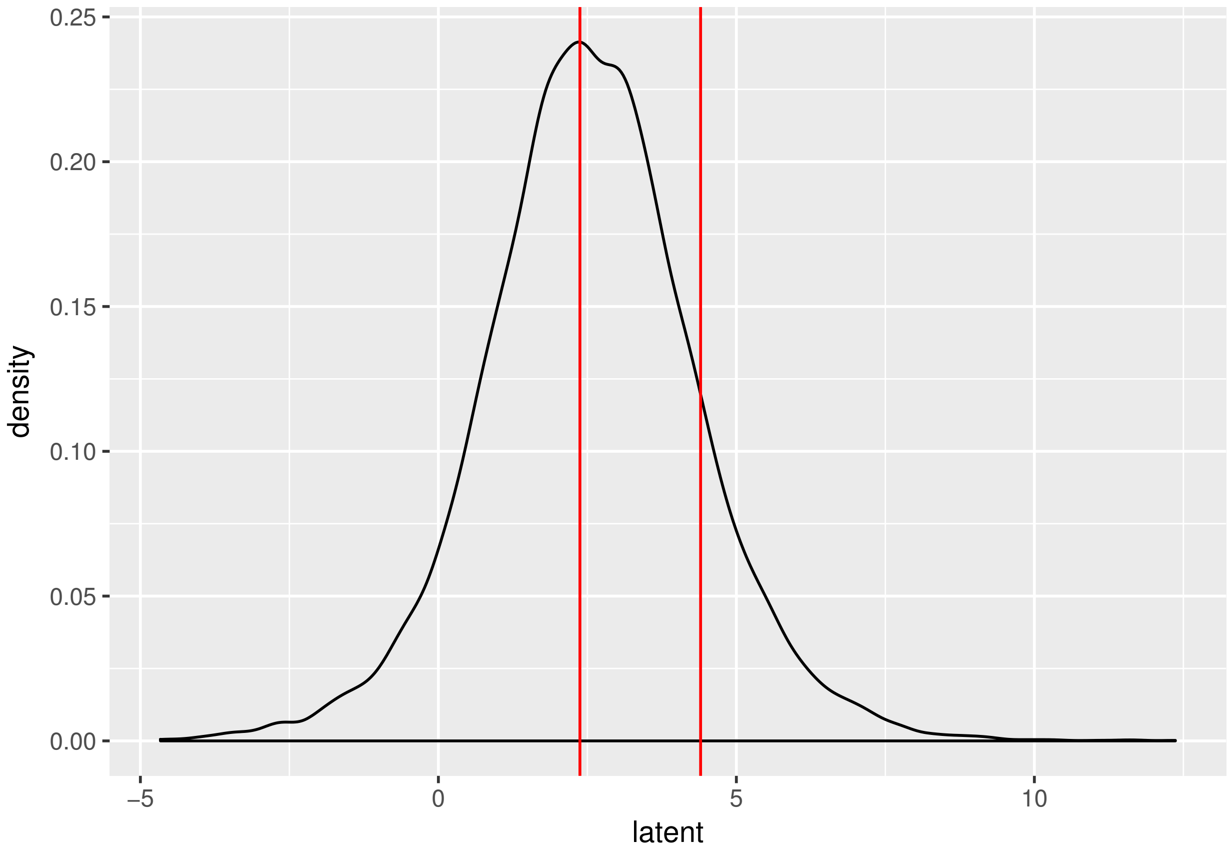
8.2.2 polr() and probability

Let’s move on to probabilistic predictions. The errors in this model have a standard logistic distribution (i.e. with mean zero and scale parameter 1.) To make a prediction for an individual:

- Find the linear predictor based on their GPA;
- Add random logistic noise;
- Compare this “latent” variable to the cutpoints;
- Repeat lots of times.

Because we might not be used to the logistic distribution, let’s first use simulation to estimate the distribution of the latent variable for a person with a 3.5 GPA. Their linear predictor is $0.725 \times 3.5 = 2.54$. We add logistic noise and see how often they fall in each cutpoint range.

```
prediction = coefficients(gpa.polr) * 3.5
latent = prediction + rlogis(10000)
ggplot(as.data.frame(latent), aes(x = latent)) + geom_density() + geom_vline(xintercept = gpa.polr$zeta
    color = "red")
```



We see that the left and middle areas are bigger than the right area. This means that “unlikely” and “somewhat likely” are more probable than “very likely.” Let’s find the exact probabilities. The probability of being “unlikely”

$$P(\beta x + \epsilon < z_{0|1})$$

where x is GPA, ϵ is standard logistic noise, and $z_{0|1}$ is the lower cutpoint. This is the same as

$$P(\epsilon < z_{0|1} - \beta x)$$

i.e., the probabilistic a standard logistic random variable is less than $z_{0|1} - \beta x$. We find logistic probabilities using the `inv.logit()` function in `boot`.

```
beta = coefficients(gpa.polr)
zeta = gpa.polr$zeta
library(boot)
inv.logit(zeta[1] - beta * 3.5)
```

```
##       0|1
## 0.4595418
```

There's a 46% chance a person with a 3.5 GPA is "unlikely" to apply to grad school. Similarly, the probability they're "very likely" to apply to grad school is the probability a standard logistic random variable is *greater* than the difference between the second cutpoint and the linear predictor:

```
1 - inv.logit(zeta[2] - beta * 3.5)
```

```
##       1|2
## 0.1343714
```

There's a 13% chance they're "very likely." That leaves a 41% chance they're "somewhat likely."

Now that we know what we're doing, we can just get these probabilities using `predict()`:

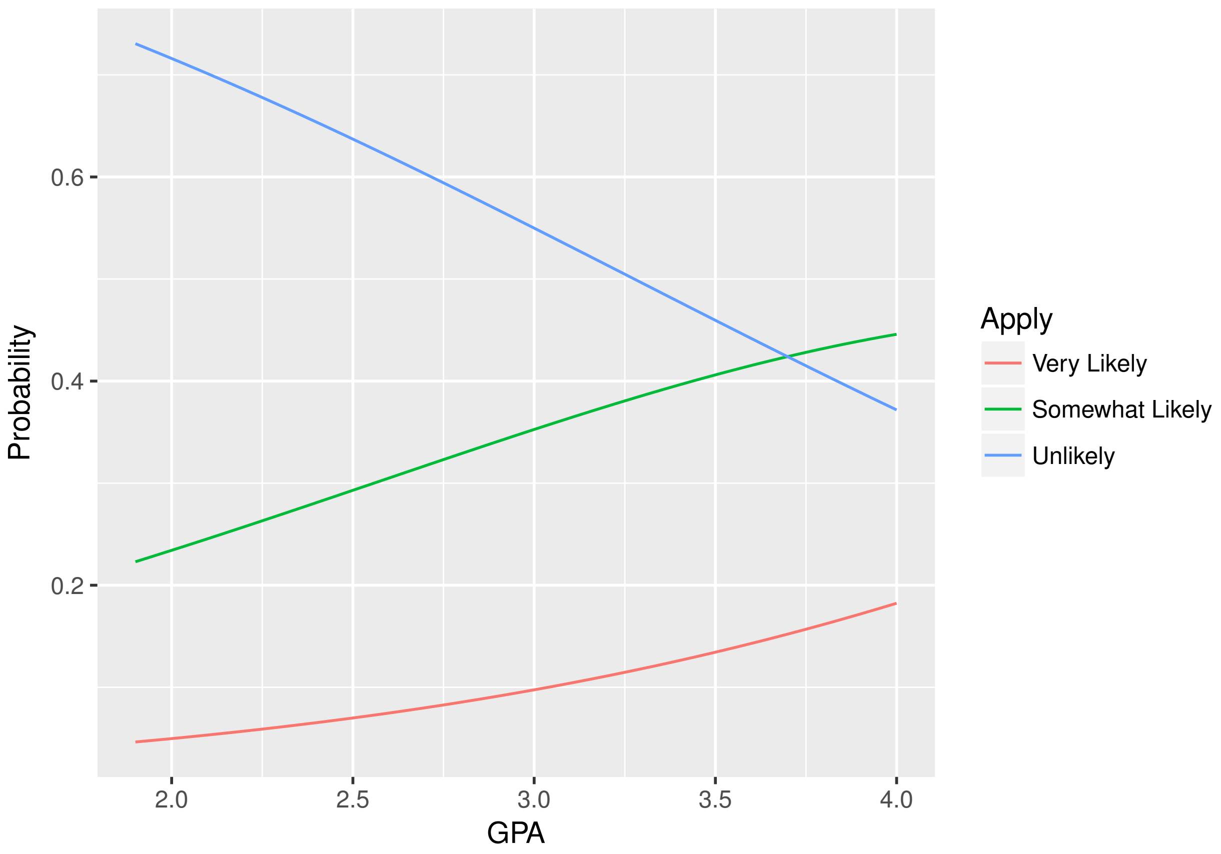
```
predict(gpa.polr, newdata = data.frame(gpa = 3.5), type = "probs")
```

```
##         0         1         2
## 0.4595418 0.4060868 0.1343714
```

8.2.3 Graphing and checking the model

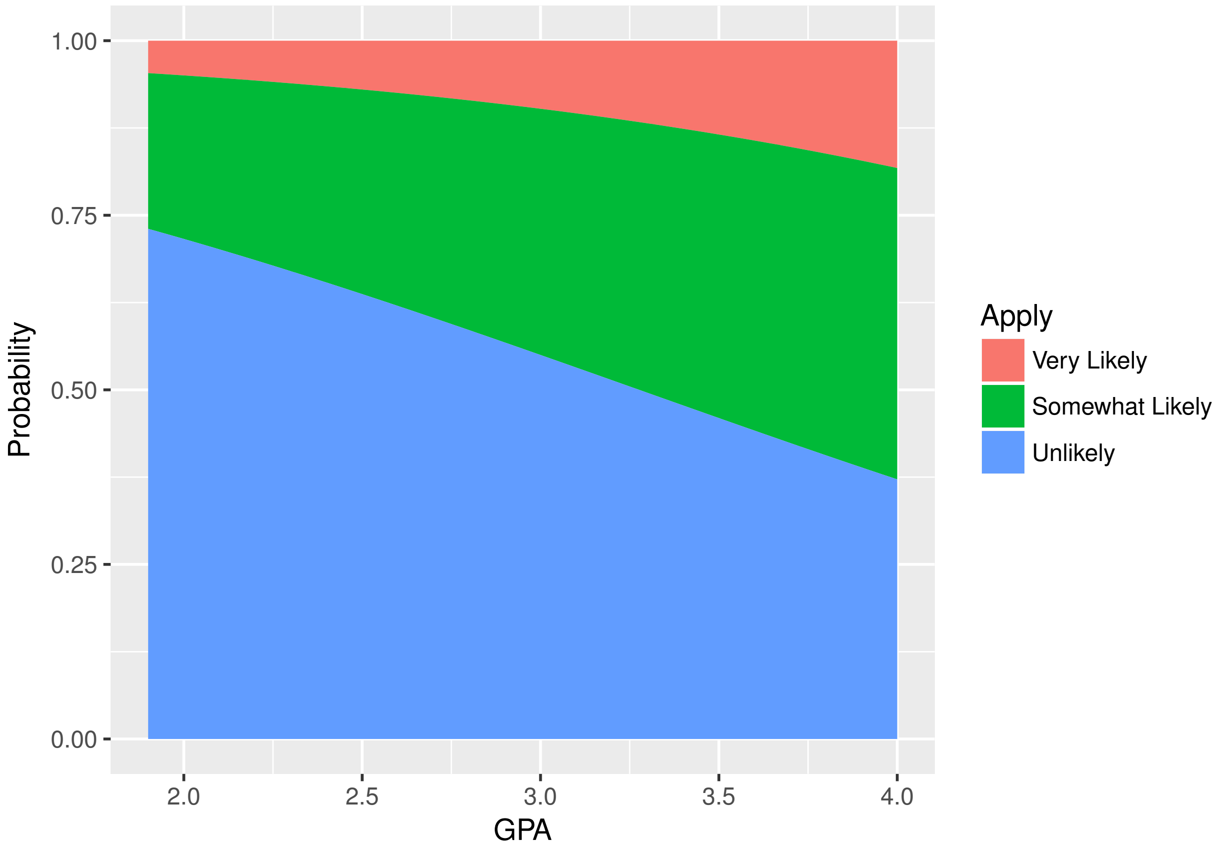
Let's display the fit as a function of GPA.

```
gpa = seq(min(gradschool$gpa), max(gradschool$gpa), 0.01)
grad.probs = predict(gpa.polr, newdata = data.frame(gpa), type = "prob")
grad.probs.df = data.frame(gpa, grad.probs)
names(grad.probs.df) = c("GPA", "Unlikely", "Somewhat Likely", "Very Likely")
grad.probs.long = grad.probs.df %>% gather(Apply, Probability, 2:4)
grad.probs.long$Apply = factor(grad.probs.long$Apply, levels = c("Very Likely",
    "Somewhat Likely", "Unlikely"))
ggplot(grad.probs.long, aes(x = GPA, y = Probability, group = Apply, color = Apply)) +
    geom_line()
```



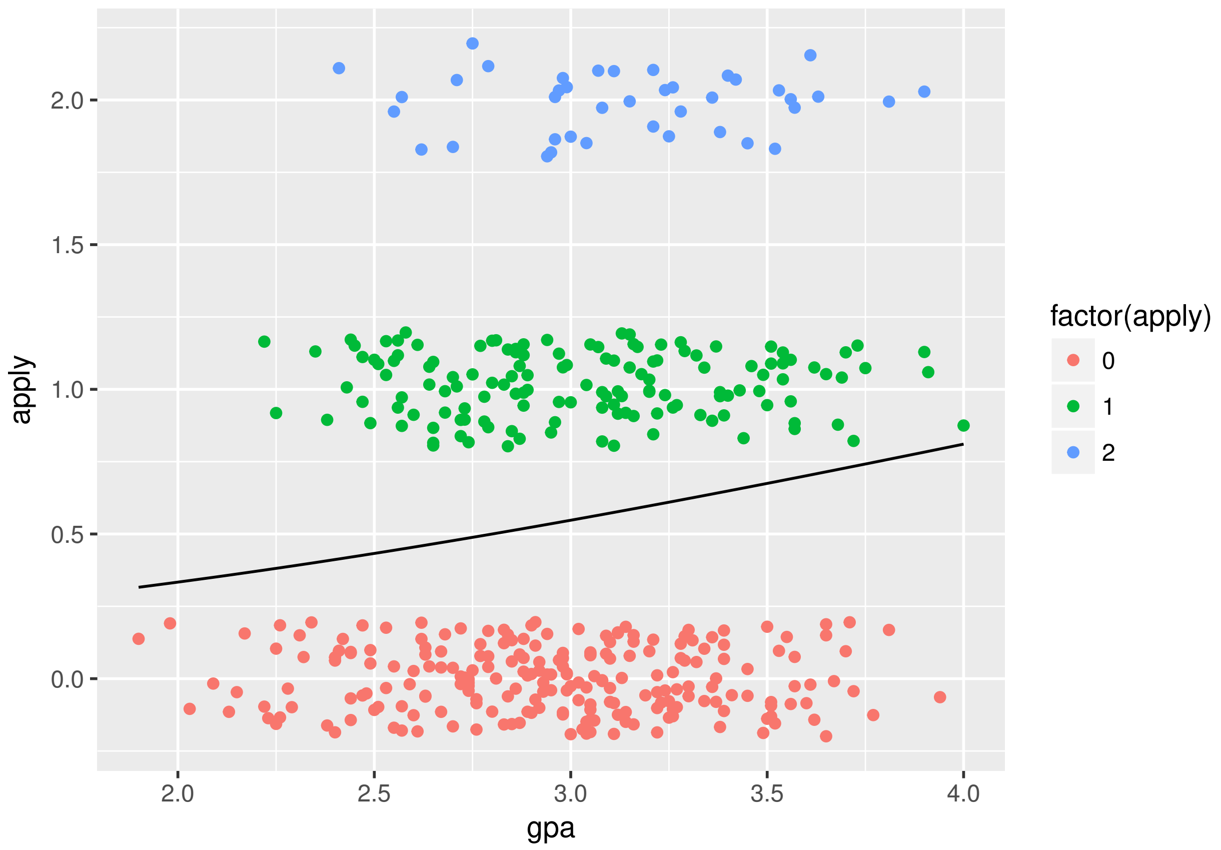
The probability of both "somewhat likely" and "very likely" increase with GPA, though "very likely" never gets very high. We can also stack the lines and use areas:

```r
ggplot(grad.probs.long, aes(x = GPA, y = Probability, group = Apply, fill = Apply)) +
    geom_area()
```



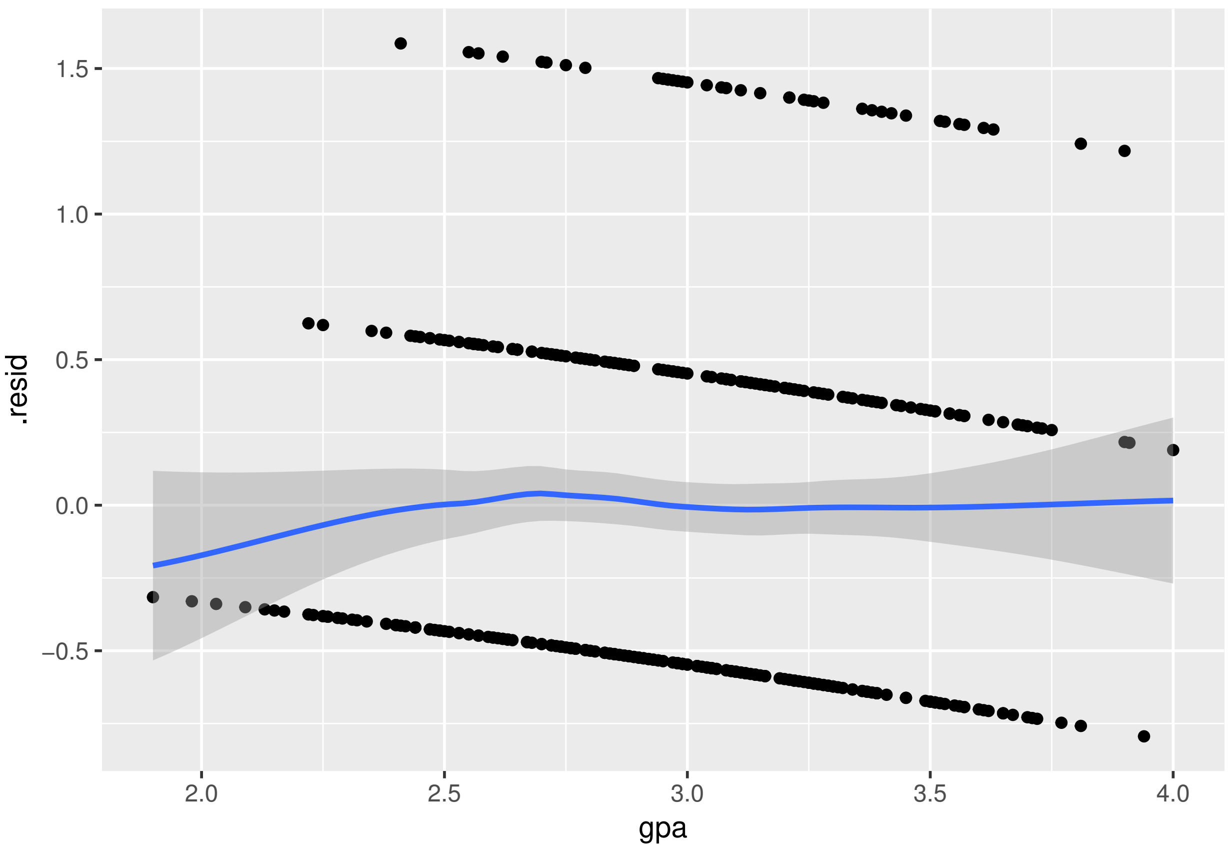
To check the fit, we could check every category separately. Instead we'll return to treating `apply` as a quantitative variable: 0 for unlikely, 1 for somewhat likely, and 2 for very likely. That lets us find means.

```r
apply.fitted = fitted.values(gpa.polr)[, 2] + 2 * fitted.values(gpa.polr)[,
    3]
apply.resid = gradschool$apply - apply.fitted
gpa.polr.df = data.frame(gradschool, .fitted = apply.fitted, .resid = apply.resid)
# Sort
gpa.polr.df = gpa.polr.df[order(gradschool$gpa), ]
ggplot(gpa.polr.df, aes(x = gpa, y = apply)) + geom_jitter(width = 0, height = 0.2,
    aes(color = factor(apply))) + geom_line(aes(x = gpa, y = .fitted))
```



Now look at the residuals:

```
ggplot(gpa.polr.df, aes(x = gpa, y = .resid)) + geom_point() + geom_smooth(method.args = list(degree =
```



This looks fine – there's a kink for GPAs below 2.25, but that could just be because there are few people with GPAs that low.

8.2.4 Multiple predictors

Let's now include two other variables in the model: `pared` is a binary variable indicating whether a parent has a grad degree, and `public` is a binary variable indicating whether the student goes to a public college.

```
grad.polr = polr(factor(apply) ~ gpa + pared + public, data = gradschool)
display(grad.polr)
```

```
## polr(formula = factor(apply) ~ gpa + pared + public, data = gradschool)
##        coef.est coef.se
## gpa     0.62     0.26  
## pared   1.05     0.27  
## public -0.06     0.30  
## 0|1     2.20     0.78  
## 1|2     4.30     0.80  
## ---
## n = 400, k = 5 (including 2 intercepts)
## residual deviance = 717.0, null deviance is not computed by polr
```

The deviance has gone down by about 16 and the coefficients are in the direction in you'd expect – your parents going to grad school means it's more probable you'll go to grad school, while going to a public college means it's slightly less probable.

As for numerical responses, we can study the fit by using `expand.grid()` to get a data frame of explanatories and making predictions.

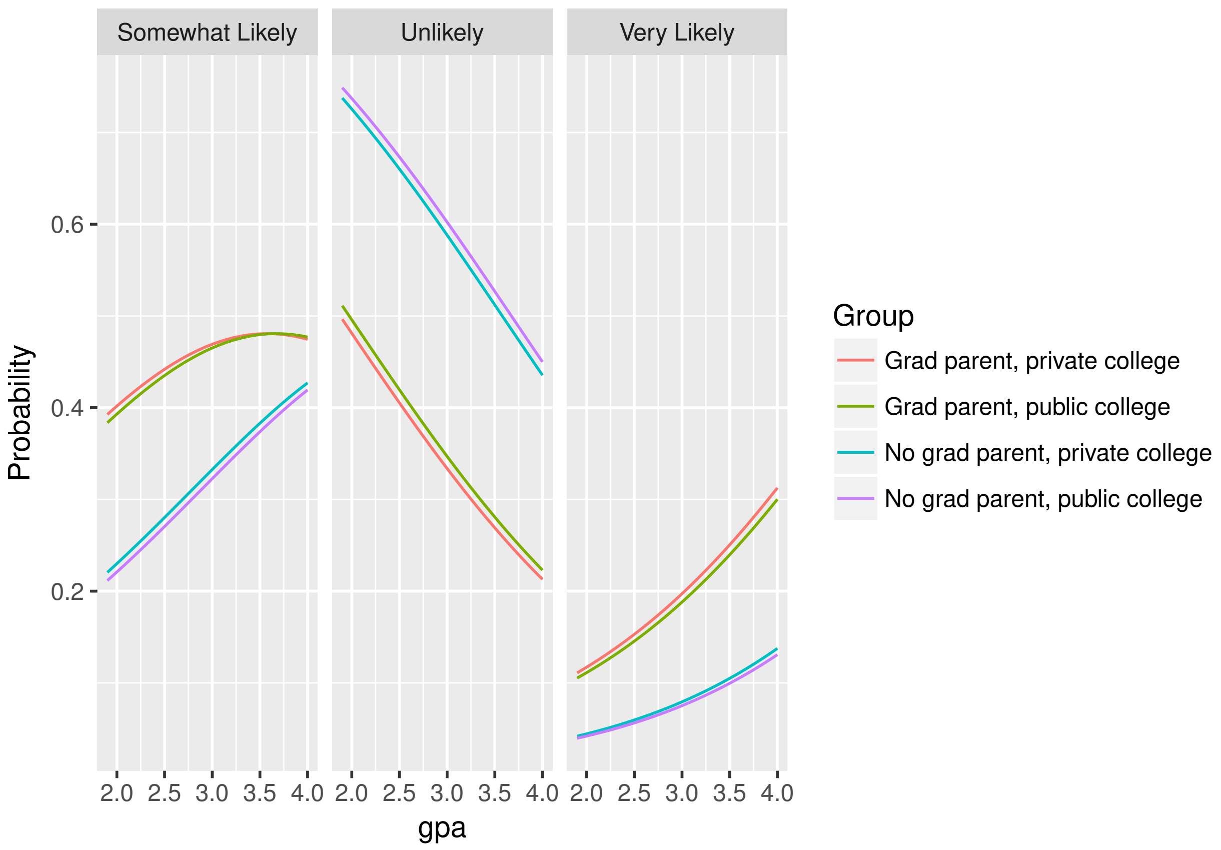
```
grad.grid = expand.grid(gpa = seq(min(gradschool$gpa), max(gradschool$gpa),
    0.01), pared = 0:1, public = 0:1)
grad.predict = as.data.frame(predict(grad.polr, newdata = grad.grid, type = "probs"))
grad.polr.df = data.frame(grad.grid, grad.predict)
names(grad.polr.df) = c("gpa", "pared", "public", "Unlikely", "Somewhat Likely",
    "Very Likely")
```

We'll append a new variable that gives the combination of `pared` and `public`.

```
group = 2 * grad.polr.df$pared + grad.polr.df$public
group[group == 3] = "Grad parent, public college"
group[group == 2] = "Grad parent, private college"
group[group == 1] = "No grad parent, public college"
group[group == 0] = "No grad parent, private college"
grad.polr.df$Group = factor(group)
```

There are a few way to view this data frame, but probably the clearest is to draw a plot for each category.

```
grad.polr.long = grad.polr.df %>% gather(Apply, Probability, 4:6)
ggplot(grad.polr.long, aes(x = gpa, y = Probability, group = Group, color = Group)) +
    geom_line() + facet_grid(~Apply)
```



We see that private or public college makes almost no difference, so we should consider dropping that from the model.

8.3 Unordered categories

Optional reading: Agresti, Categorical Data Analysis, section 8.1 (3rd edition pp. 294–297.)

With categorical regression, the main distinction is between models with **ordered** categories and models with **unordered** categories. Let's start with the ordered case.

8.3.1 Alligator food

What do alligators like to eat? Researcher captured 219 alligators in four Florida lakes, and categorized them by the primary contents of their stomach.

```
alligator = read.table("alligator.txt", header = TRUE)
head(alligator)
```

```
##      lake  sex  size    food count
## 1 Hancock male small    fish     7
## 2 Hancock male small  invert     1
## 3 Hancock male small reptile     0
## 4 Hancock male small    bird     0
## 5 Hancock male small   other     5
## 6 Hancock male large    fish     4
```

- `lake` gives the lake where the alligator was captured;
- `sex` is male or female;
- `size` is small or large;
- `food` is fish, invertebrate, reptile, bird, or other;
- `count` is how many of the 219 alligators had that combination of lake, sex, size, and food.

Check that there are $4 \times 2 \times 2 \times 5 = 80$ rows:

```
nrow(alligator)
```

```
## [1] 80
```

Check that there are 219 alligators:

```
sum(alligator$count)
```

```
## [1] 219
```

One annoying thing about categorical data is that different R function often require the data to be in different formats. To get it over with, let's put the data in wide form. This will also let us print out a table with fewer rows that gives all the data.

```
# library(tidyverse)
alligator.wide = alligator %>% spread(food, count)
alligator.wide
```

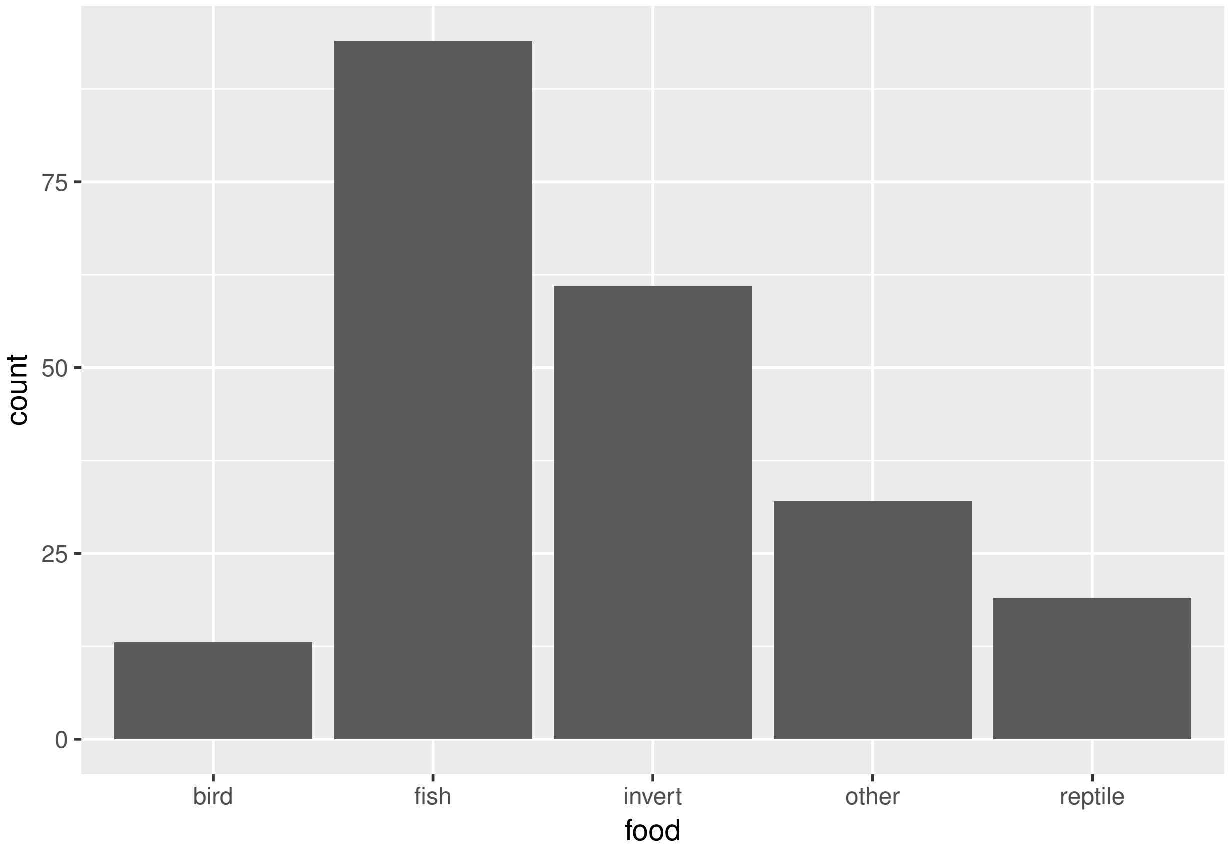
```
##        lake    sex  size bird fish invert other reptile
## 1    George female large    0    8      1     1       0
## 2    George female small    0    3      9     1       1
## 3    George   male large    1    9      0     2       0
## 4    George   male small    2   13     10     2       0
## 5   Hancock female large    2    3      0     3       1
## 6   Hancock female small    2   16      3     3       2
## 7   Hancock   male large    1    4      0     2       0
## 8   Hancock   male small    0    7      1     5       0
## 9  Oklawaha female large    1    0      1     0       0
## 10 Oklawaha female small    0    3      9     2       1
## 11 Oklawaha   male large    0   13      7     0       6
## 12 Oklawaha   male small    0    2      2     1       0
## 13 Trafford female large    0    0      1     0       0
## 14 Trafford female small    1    2      4     4       1
## 15 Trafford   male large    3    8      6     5       6
## 16 Trafford   male small    0    3      7     1       1
```

Just by looking at the numbers we see that fish are relatively popular, while birds and reptiles are unpopular. Our eventual goal will be to build a model that gives the probability an alligator prefers each type of food, based on the predictors we have.

8.3.2 Stacking vs. faceting

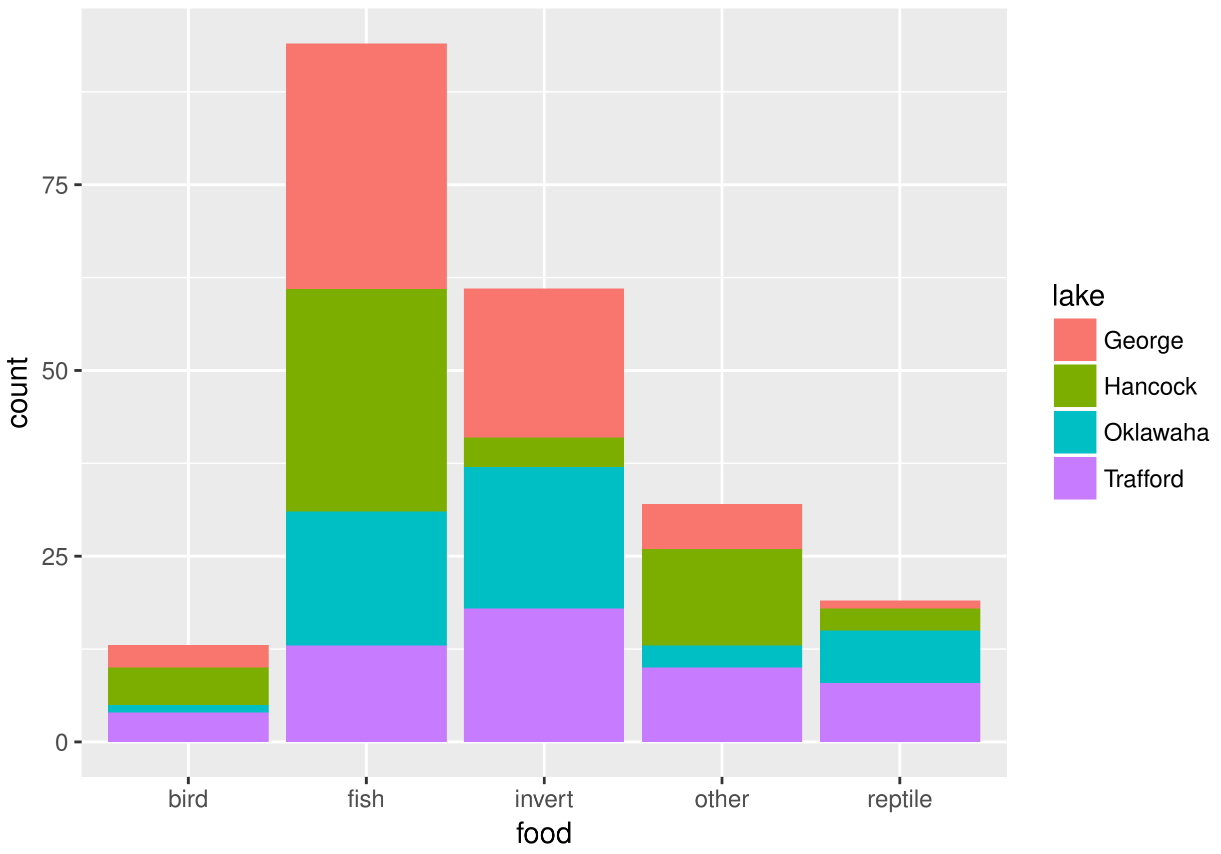
We can display the data using dots or bars. (I prefer the solidity of bars for graphs of raw count data, but this is idiosyncratic.)

```
ggplot(alligator, aes(x = food, y = count)) + geom_bar(stat = "identity")
```



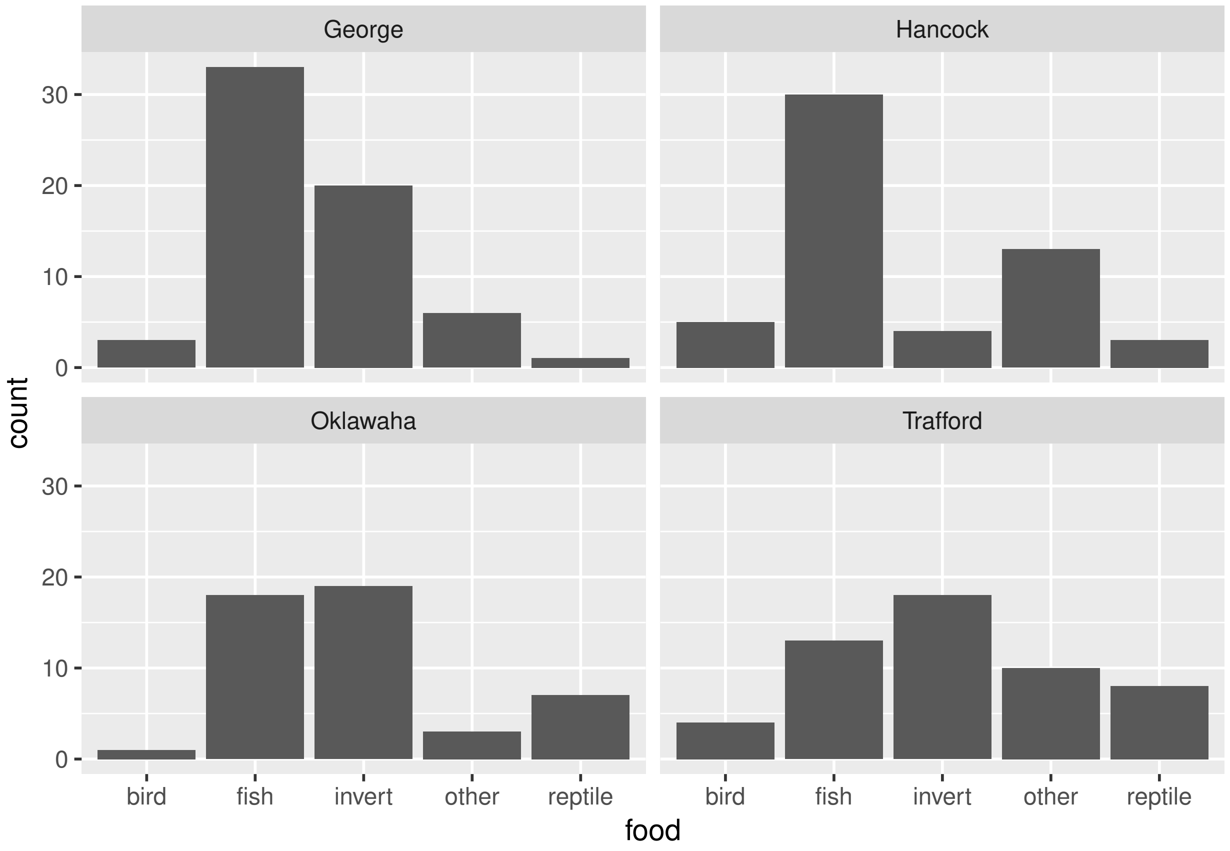
One way of including an additional categorical predictor is by stacking. This is easily achieved using `fill`:

```
ggplot(alligator, aes(x = food, y = count, fill = lake)) + geom_bar(stat = "identity")
```



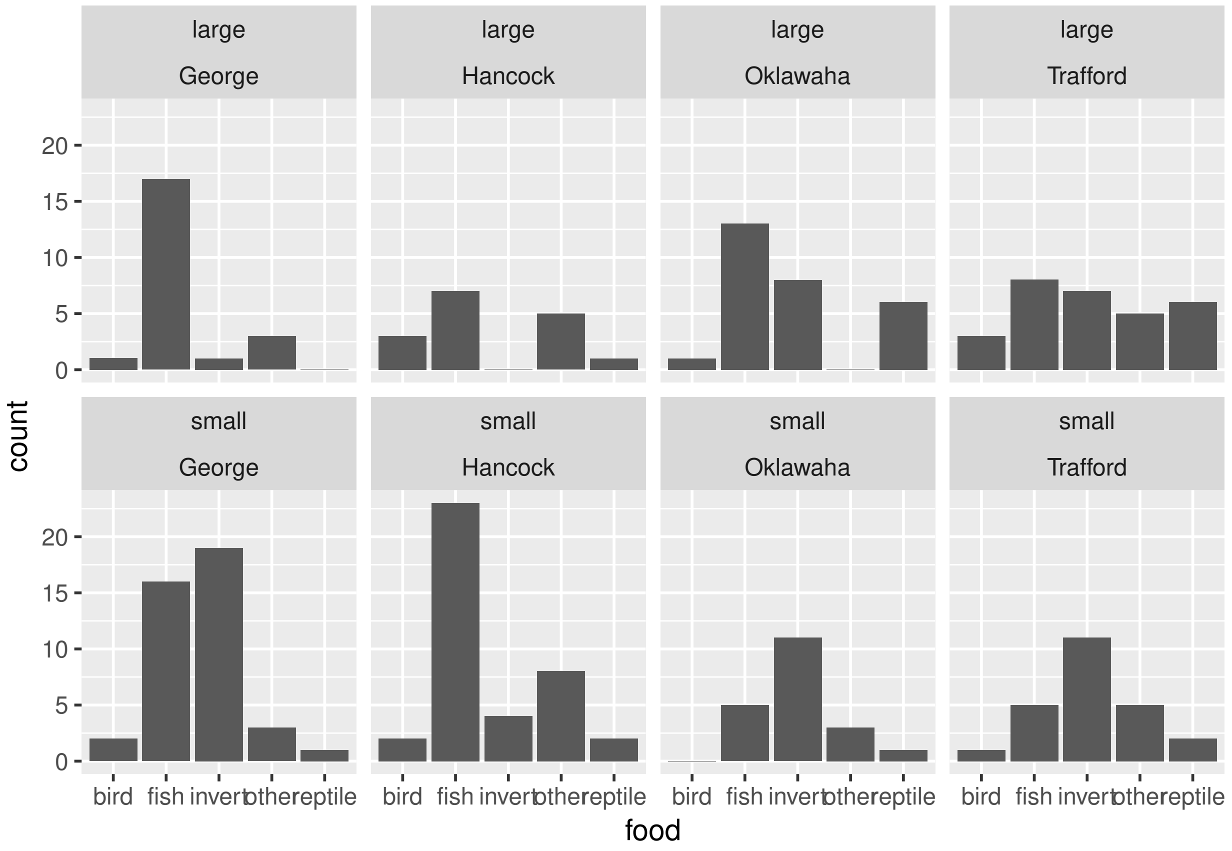
However, it's hard to clearly compare the distributions for the four lakes in the above graph. Faceting is usually better:

```
ggplot(alligator, aes(x = food, y = count)) + geom_bar(stat = "identity") +
    facet_wrap(~lake)
```



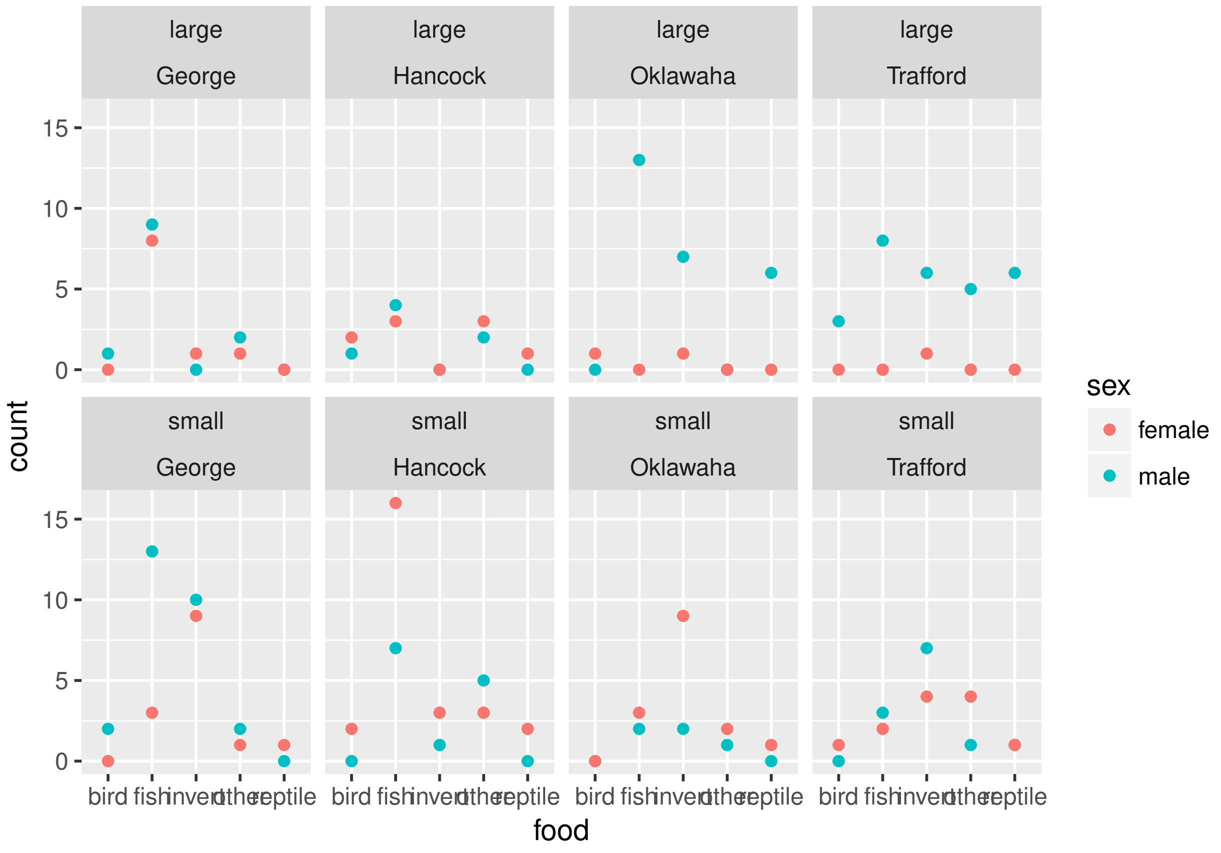
As in the continuous case, you can facet two-ways without much trouble.

```
ggplot(alligator, aes(x = food, y = count)) + geom_bar(stat = "identity") +
    facet_wrap(~size + lake, ncol = 4)
```



If you really want to get another variable in, you can use switch back to dots and use color:

```r
ggplot(alligator, aes(x = food, y = count, color = sex)) + geom_point() + facet_wrap(~size +
    lake, ncol = 4)
```



8.3.3 Mosaic plots

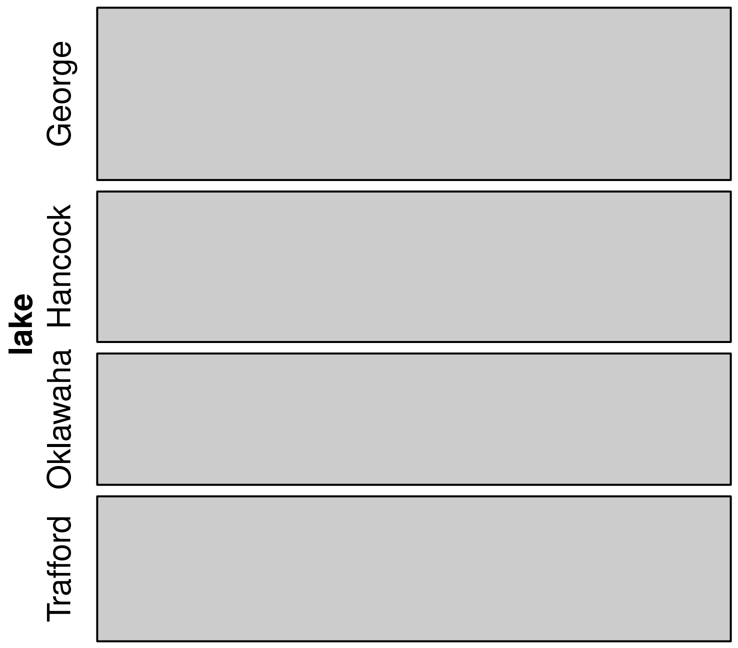
A (slightly controversial) alternative to the aobve is **mosaic plots**. I'll use the `mosaic()` function in package `vcd` (there's a new `ggmosiac` package but I haven't tried it out yet.) `mosiac` requires data to be in cross-tabular form. This is achieved using `xtabs()`:

```
xtabs(count ~ food + size, data = alligator)
```

```
##          size
## food      large small
##   bird        8     5
##   fish       45    49
##   invert     16    45
##   other      13    19
##   reptile    13     6
```

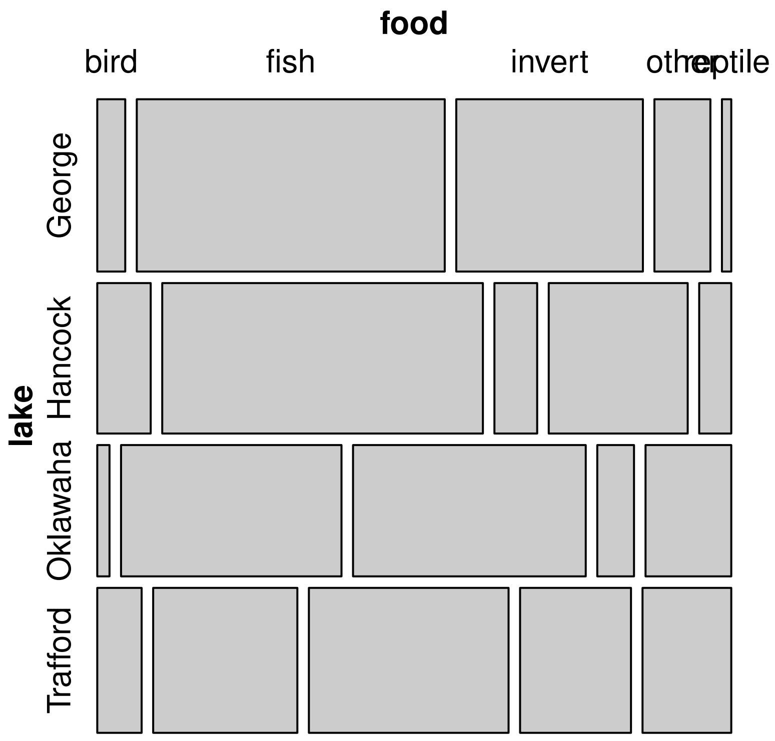
Let's create a table with all four categorical variables, then start drawing.

```
alligator.table = xtabs(count ~ food + size + sex + lake, data = alligator)
# install.packages('vcd')
library(vcd)
mosaic(~lake, alligator.table)
```



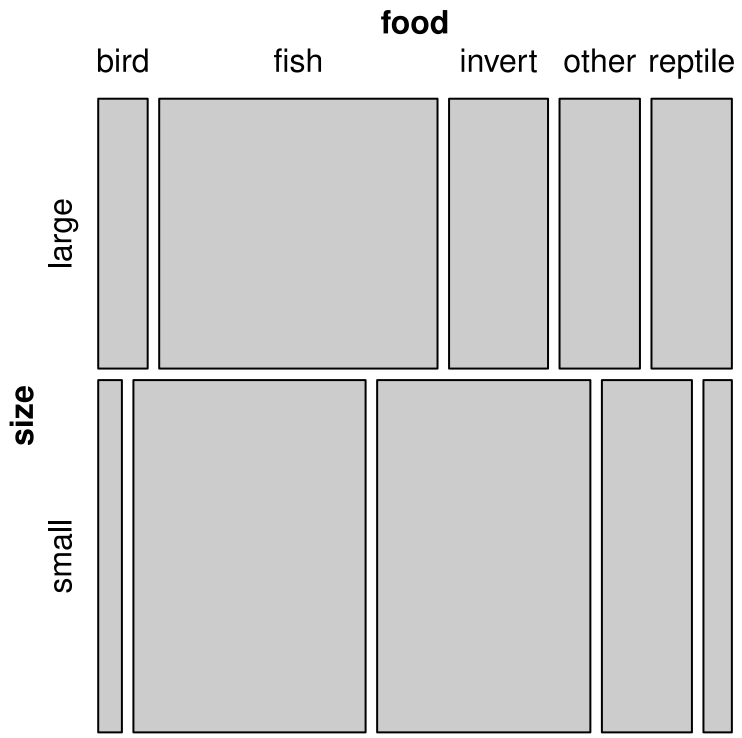
Here, the height of each block indicate the relative frequency of the variable. This isn't very interesting, but we can also divide up the area two ways:

```
mosaic(~lake + food, alligator.table)
```

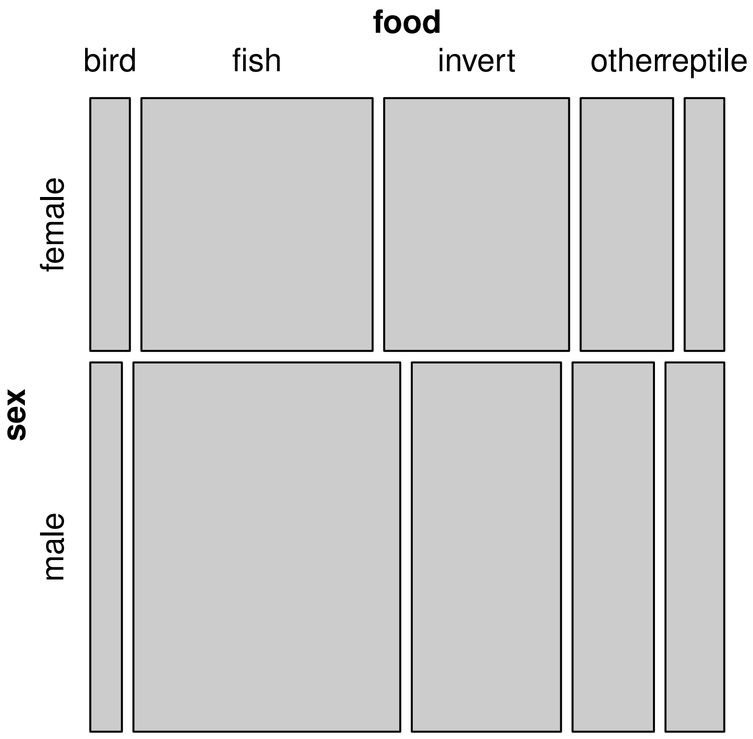


The above plot shows us the *conditional* distribution of each type of food, given the lake, as well as the unconditional relative frequency of each type of lake – they're all roughly comparable. The preferred types of food do seem to differ a lot by lake.

```
mosaic(~size + food, alligator.table)
```



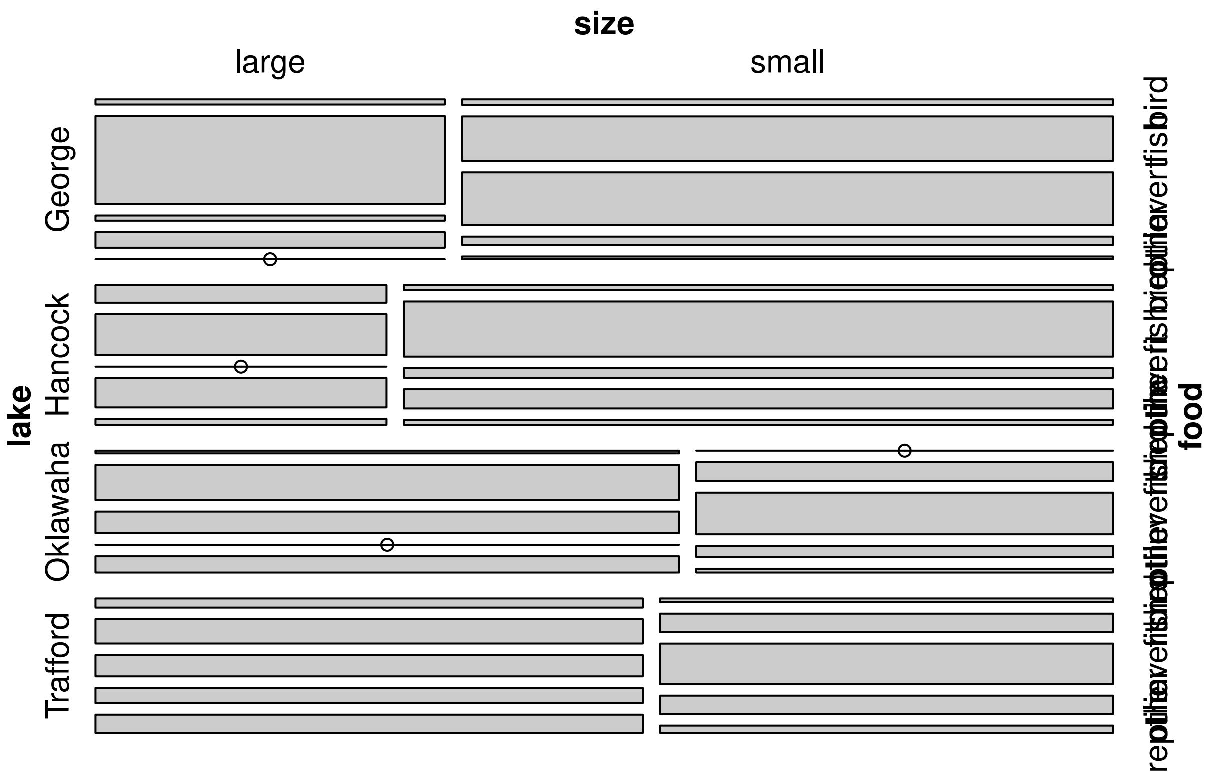
```r
mosaic(~sex + food, alligator.table)
```



We see that in the sample, there are more small gators than large ones, and more males than females. More importantly, the conditional distribution of food looks quite different between big and small gators, but quite similar comparing males and females.

We could keep on subdividing the bars in an attempt to look for interactions, but this starts to get messy and the data gets sparse.

```
mosaic(~lake + size + food, alligator.table)
```



8.3.4 Multinomial regression

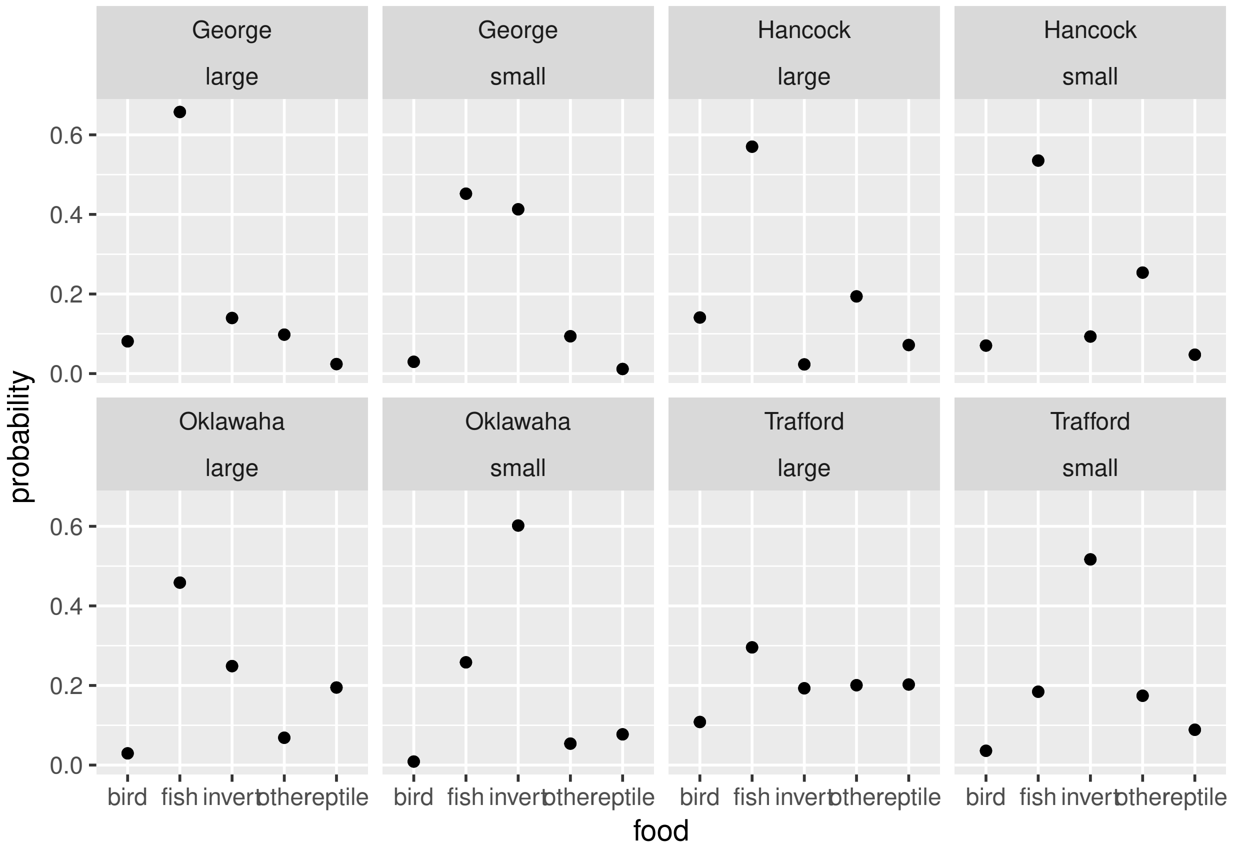
Let's fit a model using `lake` and `size` as predictors. For categorical responses, we want the conditional distribution given the predictors to be `multinomial`. I use the `vglm()` function (vector GLM) in package `VGAM` to fit multinomial regressions. The syntax is similar to that of `glm()` with family `multinomial`, except you need to specify a matrix of responses (one column for each category.) This can be done using `cbind()` with the data in wide format.

```
# install.packages('VGAM')
library(VGAM)
alligator.mlogit = vglm(cbind(bird, fish, invert, other, reptile) ~ lake + size,
    family = multinomial, data = alligator.wide)
alligator.mlogit
```

```
##
## Call:
## vglm(formula = cbind(bird, fish, invert, other, reptile) ~ lake +
##     size, family = multinomial, data = alligator.wide)
##
##
## Coefficients:
##  (Intercept):1  (Intercept):2  (Intercept):3  (Intercept):4  lakeHancock:1
##      1.2214559      3.3145327      1.7655141      1.4102610     -0.5476591
##  lakeHancock:2  lakeHancock:3  lakeHancock:4 lakeOklawaha:1 lakeOklawaha:2
##     -1.2427766     -2.9011352     -0.4165804     -3.1120797     -2.4588720
## lakeOklawaha:3 lakeOklawaha:4 lakeTrafford:1 lakeTrafford:2 lakeTrafford:3
##     -1.5216526     -2.4532189     -1.8474865     -2.9352533     -1.8132685
## lakeTrafford:4    sizesmall:1    sizesmall:2    sizesmall:3    sizesmall:4
##     -1.4188846     -0.2793969      0.3512628      1.8094675      0.6828131
##
## Degrees of Freedom: 64 Total; 44 Residual
## Residual deviance: 52.47849
## Log-likelihood: -74.42948
##
## This is a multinomial logit model with 5 levels
```

There are lots of coefficients here! These can be interpreted in terms of log odds, but instead we'll examine the model fit graphically.

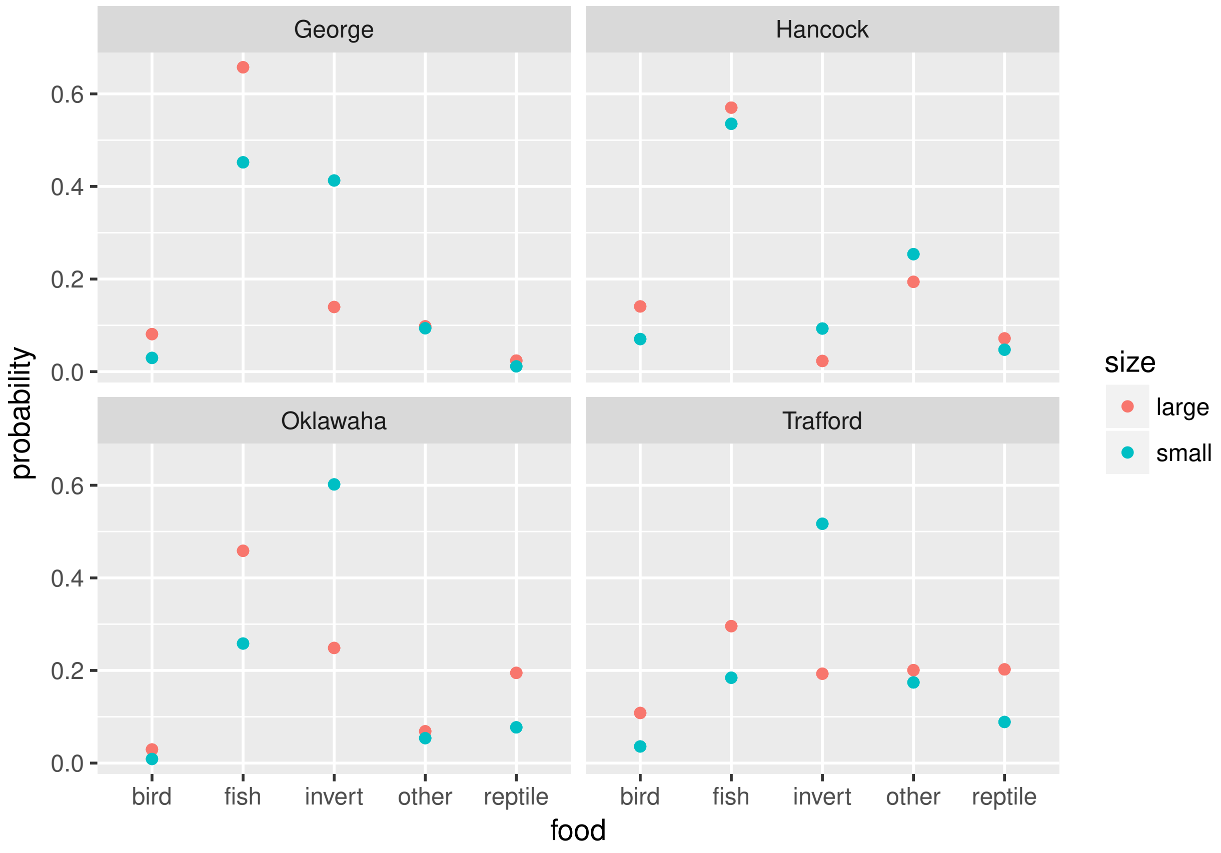
```
alligator.mlogit.df = data.frame(model.frame(alligator.mlogit), fitted.values(alligator.mlogit))
alligator.mlogit.long = alligator.mlogit.df %>% gather(food, probability, bird:reptile)
ggplot(alligator.mlogit.long, aes(x = food, y = probability)) + geom_point() +
    facet_wrap(~lake + size, ncol = 4)
```



This is basically a standardized and smoothed version of our faceted bar graphs above. The smoothing might have some advantages over just taking raw data, e.g. there are no zero probabilities.

We could also collapse the large and small rows and color-code:

```
ggplot(alligator.mlogit.long, aes(x = food, y = probability, col = size)) +
    geom_point() + facet_wrap(~lake)
```



Let's check the deviance of a couple of alternatives:

```
deviance(vglm(cbind(bird, fish, invert, other, reptile) ~ lake + size + sex,
    family = multinomial, data = alligator.wide))
```

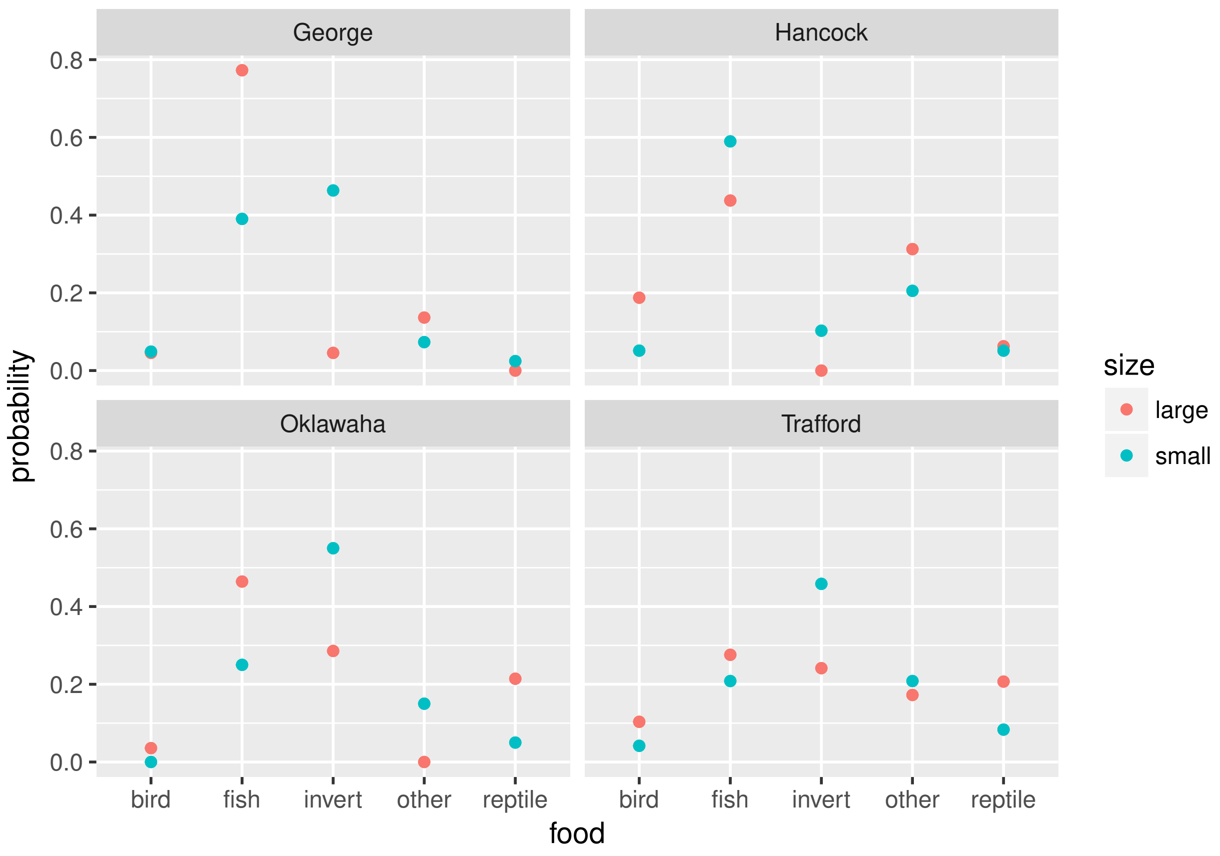
```
## [1] 50.26369
```

```
deviance(vglm(cbind(bird, fish, invert, other, reptile) ~ lake * size, family = multinomial,
    data = alligator.wide))
```

```
## [1] 35.39866
```

Adding sex only reduces deviance by a trivial amount (less than the 4 extra degrees of freedom), and so is unlikely to be worth it. Adding an interaction between lake and size reduces deviance by a lot, but also makes the model much more complicated, so that's a judgment call. It's probably worth it if you want to do prediction.

```
alligator.int = vglm(cbind(bird, fish, invert, other, reptile) ~ lake * size,
    family = multinomial, data = alligator.wide)
alligator.int.df = data.frame(model.frame(alligator.int), fitted.values(alligator.int))
alligator.int.long = alligator.int.df %>% gather(food, probability, bird:reptile)
ggplot(alligator.int.long, aes(x = food, y = probability, col = size)) + geom_point() +
    facet_wrap(~lake)
```



But when we add an interaction between categorical predictors, this is just equivalent to taking the raw proportions for each two-way combination of predictors. So we're back to the two-way faceted bar graph, only standardized so that each panel adds up to probability 1.

8.3.5 Quantitative predictors

We can also fit multinomial models with quantitative predictors. In the file `gator2.txt`, the numerical predictor is the length of the alligator in meters.

```
gator2 = read.table("gator2.txt", header = TRUE)
summary(gator2)
```

```
##      length                  food
##  Min.   :1.240   Fish         :31
##  1st Qu.:1.575   Invertebrates:20
##  Median :1.850   Other        : 8
##  Mean   :2.130
##  3rd Qu.:2.450
##  Max.   :3.890
```

```
gator2.mlogit = vglm(food ~ length, family = multinomial, data = gator2)
gator2.mlogit
```

```
##
## Call:
## vglm(formula = food ~ length, family = multinomial, data = gator2)
##
##
```

```
## Coefficients:
## (Intercept):1 (Intercept):2      length:1      length:2
##      1.617731      5.697444     -0.110109     -2.465446
##
## Degrees of Freedom: 118 Total; 114 Residual
## Residual deviance: 98.34124
## Log-likelihood: -49.17062
##
## This is a multinomial logit model with 3 levels
```

As we did in the ordered categories case, let's start making predictions to understand the fit. First, on the linear predictor (i.e. transformed) scale:

```
log.ratios = predict(gator2.mlogit, newdata = data.frame(length = 2))
log.ratios
```

```
##   log(mu[,1]/mu[,3]) log(mu[,2]/mu[,3])
## 1           1.397513          0.7665519
```

This gives us the log probability ratios for one type of food to another. The log of the probability ratio for fish to other is $1.62 - 0.11 \times 2 \approx 1.4$ and for invertebrates to other is $5.7 - 2.47 \times 2 \approx 0.77$. (Note that `vglm()` take the *last* level of the factor as the baseline, which is weird but is what it is.)

Now, on the probability scale:

```
twometerprobs = predict(gator2.mlogit, newdata = data.frame(length = 2), type = "response")
twometerprobs
```

```
##        Fish Invertebrates     Other
## 1 0.5620216     0.2990405 0.1389379
```

To go from the probability scale to the linear predictor scale:

```
log(twometerprobs[1:2]/twometerprobs[3])
```

```
## [1] 1.3975134 0.7665519
```

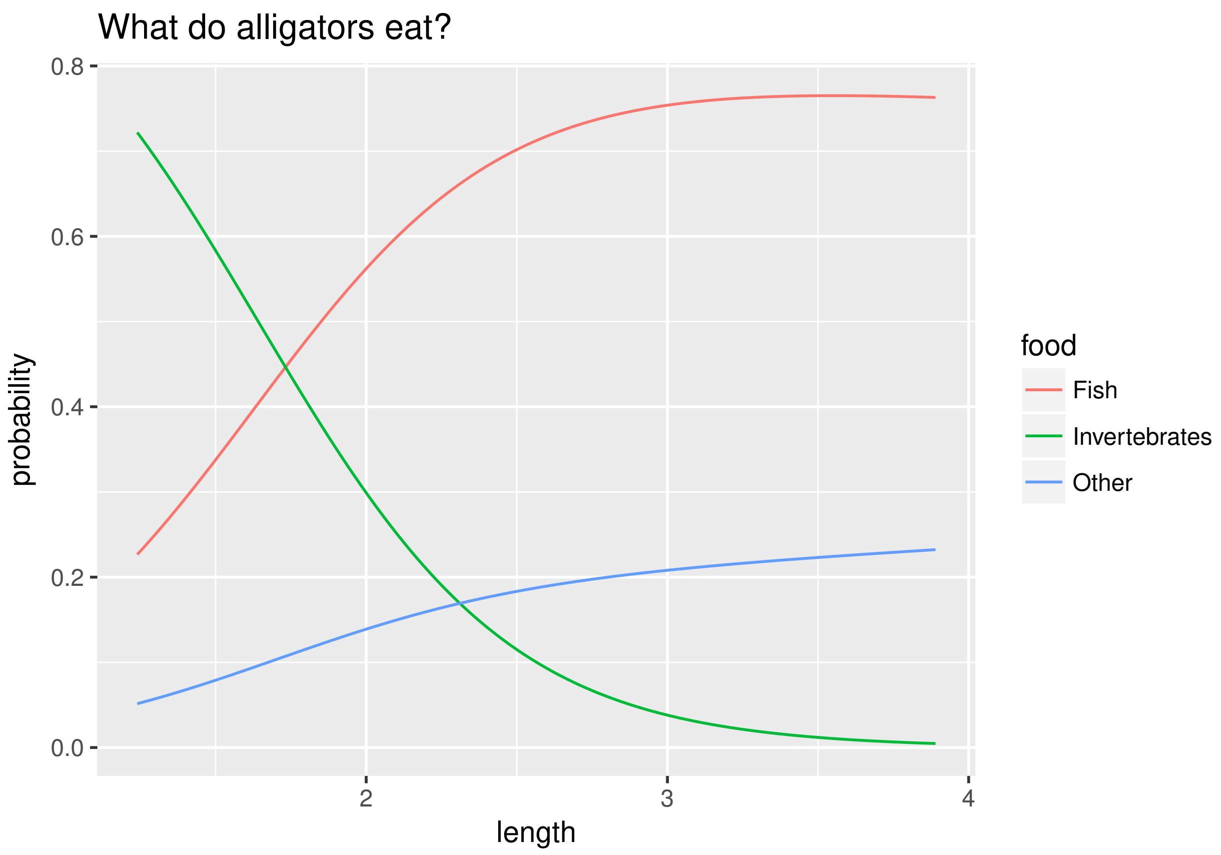
To go from the linear predictor scale to the probability scale:

```
exp(c(log.ratios, 0))/sum(exp(c(log.ratios, 0)))
```

```
## [1] 0.5620216 0.2990405 0.1389379
```

Now let's look at how these probabilities vary with length:

```
length = data.frame(length = seq(1.24, 3.89, 0.01))
gator2.pred = predict(gator2.mlogit, newdata = length, type = "response")
gator2.pred.df = data.frame(length, gator2.pred)
gator2.pred.long = gator2.pred.df %>% gather(food, probability, Fish:Other)
ggplot(gator2.pred.long, aes(x = length, y = probability, group = food, color = food)) +
    geom_line() + ggtitle("What do alligators eat?")
```



Bigger alligators prefer fish and, to a lesser extend, “other.” Smaller alligators prefer invertebrates.

We finally note that just as with the Poisson, multinomial data is often overdispersed, so be careful of taking standard errors literally.