

# Chapter 7

## Binary responses

### 7.1 Logistic regression, plus dealing with messy data

**Optional reading:** Gelman & Hill pp. 79–86. For a more detailed and more traditional treatment, try Agresti, Categorical Data Analysis, ch. 5 & 6 (3rd edition.)

#### 7.1.1 ANES

Download the ANES Time Series Cumulative Data File, containing data from 1948 to 2012:

[http://www.electionstudies.org/studypages/download/datacenter\\_all\\_datasets.php](http://www.electionstudies.org/studypages/download/datacenter_all_datasets.php)

The raw data is just an astonishing mess, so get the Stata file. The `rio` library is the current best for reading exotic data formats into R:

```
# install.packages('rio')
library(rio)
ANES = import("anes_timeseries_cdf.dta")
```

What do we care about? Suppose we care about income. Looking at the codebook for the study, the relevant variable is VCF0114.

```
income = ANES$VCF0114
summary(income)
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
##	0.000	2.000	3.000	2.663	4.000	5.000	1511

Income is measured on a scale from 1 to 5:

- 1: 0 to 16 percentile
- 2: 17 to 33 percentile
- 3: 34 to 67 percentile
- 4: 68 to 95 percentile
- 5: 96 to 100 percentile

(The zeroes and NA's are missing values.) This allows comparability between years. This is really an ordinal variable but we might find some advantages in treating it as quantitative.

Next we need the year for each observation. This is VCF0004.

```
year = ANES$VCF0004
summary(year)
```

```
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 1948    1970 1984 1983 1998 2012
```

Now we need a vote variable. In order to do logistic regression, we need a binary variable. The variable VCF0704a gives the *two-party* Presidential vote: that is, those who voted for third-parties or didn't vote should be missing values.

```
vote = ANES$VCF0704a
summary(vote)
```

```
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## 0.000 0.000 1.000 0.942 2.000 2.000 20724
```

The coding is that “1” means the Democrat, “2” means the Republican, and “0” or “NA” means some other outcome. We want everything to be coded as 0, 1, or NA. First, change the zeroes to NA’s:

```
vote[vote == 0] = NA
summary(vote)
```

```
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## 1.00 1.00 1.00 1.48 2.00 2.00 33457
```

Now suppose we make 1 mean Republican and 0 mean Democrat. This just requires subtracting 1:

```
vote = vote - 1
summary(vote)
```

```
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## 0.00 0.00 0.00 0.48 1.00 1.00 33457
```

The variable really represents a two-party vote for a Republican now, so for clarity let’s just rename it as such.

```
Republican = vote
```

Finally, let’s include one other variable called `survey.weights` that we’ll end up using later on.

```
survey.weights = ANES$VCF0009z
summary(survey.weights)
```

```
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 0.0212 1.0000 1.0000 1.0488 1.0000 4.9909
```

Now let’s put our four variables into a data frame.

```
ANES.df = data.frame(year, income, Republican, survey.weights)
summary(ANES.df)
```

```
##      year      income     Republican survey.weights
##  Min. :1948  Min. :0.000  Min. :0.00  Min. :0.0212
##  1st Qu.:1970 1st Qu.:2.000  1st Qu.:0.00  1st Qu.:1.0000
##  Median :1984 Median :3.000  Median :0.00  Median :1.0000
##  Mean   :1983 Mean   :2.663  Mean   :0.48  Mean   :1.0488
##  3rd Qu.:1998 3rd Qu.:4.000  3rd Qu.:1.00  3rd Qu.:1.0000
##  Max.  :2012  Max. :5.000  Max. :1.00  Max. :4.9909
##          NA's :1511    NA's :33457
```

### 7.1.2 The 1992 election

Let's start by picking out one year and looking at the relationship between income and vote. Let's choose 1992, which pitted incumbent Republican George H.W. Bush against Democratic challenger Bill Clinton.

```
ANES1992 = subset(ANES.df, year == 1992)
summary(ANES1992)
```

```
##      year      income    Republican survey.weights
##  Min.   :1992   Min.   :0.000   Min.   :0.0000   Min.   :0.8462
##  1st Qu.:1992  1st Qu.:2.000  1st Qu.:0.0000  1st Qu.:0.9872
##  Median :1992  Median :3.000  Median :0.0000  Median :1.0000
##  Mean   :1992  Mean   :2.688  Mean   :0.4156  Mean   :1.0011
##  3rd Qu.:1992  3rd Qu.:4.000  3rd Qu.:1.0000  3rd Qu.:1.0189
##  Max.   :1992  Max.   :5.000  Max.   :1.0000  Max.   :1.4615
##           NA's   :1128
```

The summary is a bit indirect for the `Republican` variable. A trick to get a more informative summary is to (temporarily) treat it as a factor variable.

```
summary(factor(ANES1992$Republican))
```

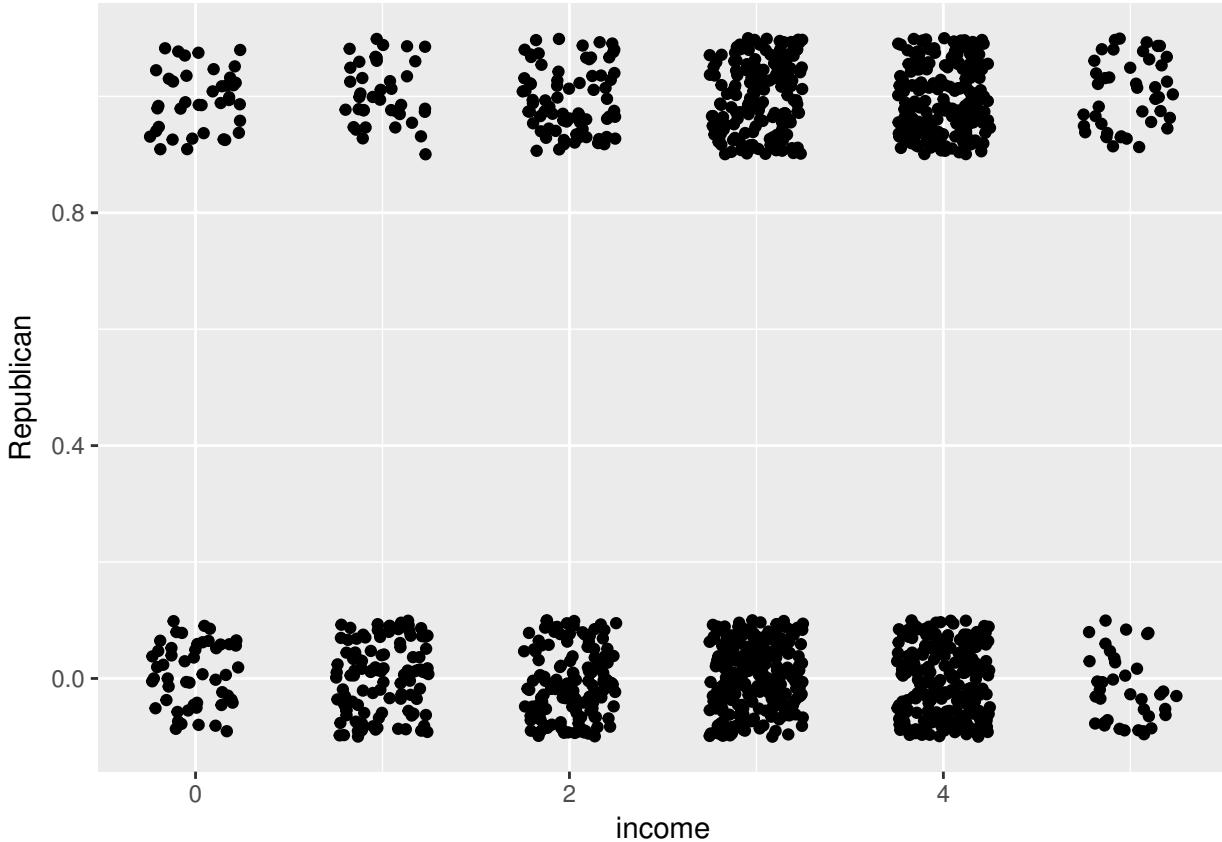
```
##     0     1 NA's
## 793 564 1128
```

### 7.1.3 Logistic regression with one predictor

Let's now look at the relationship between income and vote in 1992. If we want to look at a scatterplot, we need to jitter it, as there are only five or six  $x$ -values and two  $y$ -values.

```
library(ggplot2)
ggplot(ANES1992, aes(x = income, y = Republican)) + geom_jitter(height = 0.1,
width = 0.25)
```

```
## Warning: Removed 1128 rows containing missing values (geom_point).
```



We can also summarize the data quantitatively:

```
aggregate(Republican ~ income, mean, data = ANES1992)
```

```
##   income Republican
## 1      0  0.4333333
## 2      1  0.2671233
## 3      2  0.3454545
## 4      3  0.4035533
## 5      4  0.4883178
## 6      5  0.5316456
```

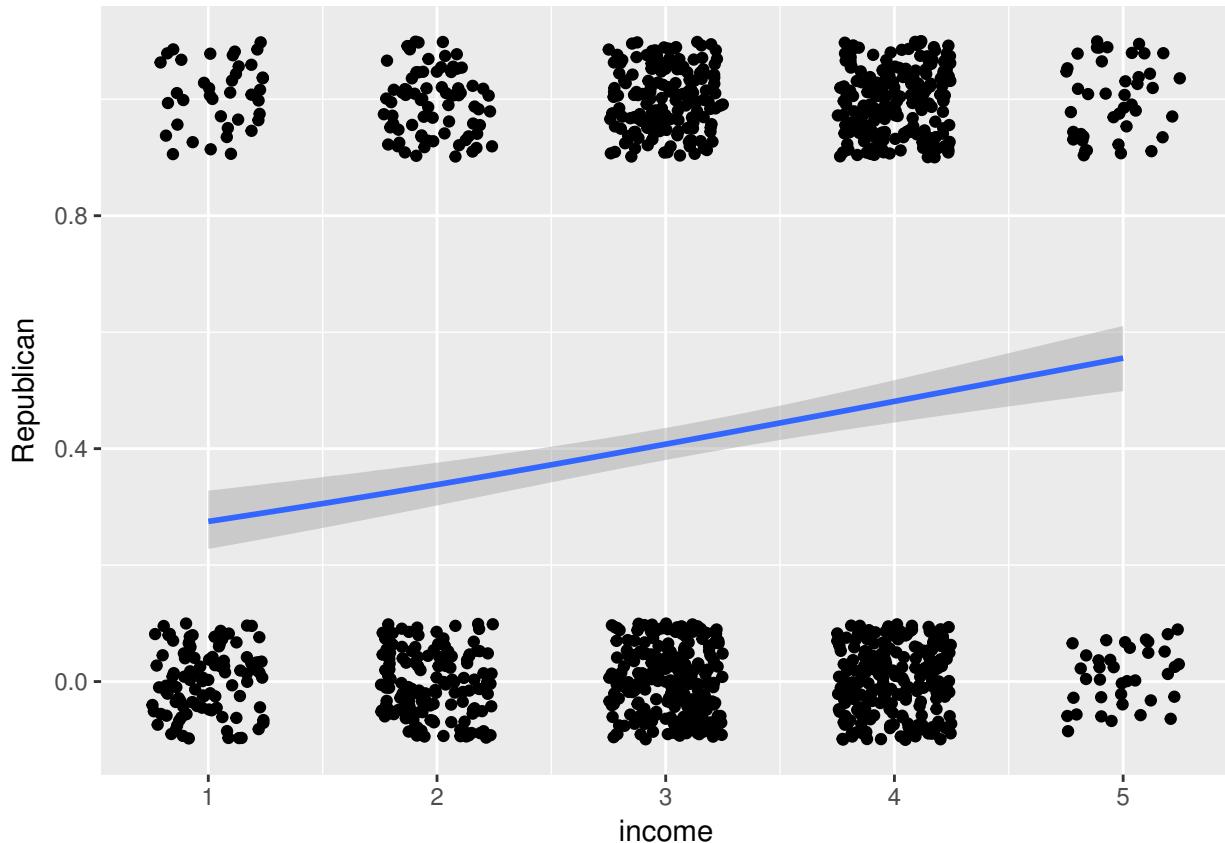
This gives the proportion (out of major party voters) who voted for Bush for each income group. Aside from group zero, which represents missing values of income, we see a strictly increasing pattern. This suggests that logistic regression might work well on the data. (If the pattern was non-monotonic, a nonparametric method like a generalized additive model (GAM) would be a better modeling choice.)

We fit such a logistic regression using `income` as a quantitative variable and omitting missing values. Logistic regression is a special case of a GLM, so we use the `glm()` function; specifying a binomial family fits a logistic regression by default. Firstly, we can just add the fitted curve to the jittered plot:

```
ANES1992 = subset(ANES1992, income > 0)
ggplot(ANES1992, aes(x = income, y = Republican)) + geom_jitter(height = 0.1,
  width = 0.25) + geom_smooth(method = "glm", method.args = list(family = "binomial"))
```

```
## Warning: Removed 1014 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 1014 rows containing missing values (geom_point).
```



We can also fit it explicitly:

```
Bush.logit = glm(Republican ~ income, family = binomial, data = ANES1992)
summary(Bush.logit)
```

```
##
## Call:
## glm(formula = Republican ~ income, family = binomial, data = ANES1992)
##
## Deviance Residuals:
##      Min        1Q     Median        3Q       Max
## -1.2733  -1.0235  -0.9086   1.2096   1.6069
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.26750   0.17789  -7.125 1.04e-12 ***
## income       0.29802   0.05379   5.541 3.01e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1719.1 on 1266 degrees of freedom
## Residual deviance: 1687.2 on 1265 degrees of freedom
## (1014 observations deleted due to missingness)
## AIC: 1691.2
##
## Number of Fisher Scoring iterations: 4
```

The summary gives a lot of information; we'll focus on the coefficients. The summary tells us that

$$\text{logit}[\text{P(Bush)}] = -1.27 + 0.298 \times \text{income}$$

where the logit function is

$$\text{logit}(x) = \log_e \frac{x}{1-x}.$$

To find  $\text{P(Bush)}$ , we invert the logit:

$$\text{P(Bush)} = \frac{e^y}{1 + e^y}$$

where

$$y = \text{logit}[\text{P(Bush)}].$$

For a quick and dirty interpretation, the “divide by 4” rule is useful: the maximum change in probability associated with a one unit change in  $x$  is the coefficient of  $x$  divided by four. So going one income group changes the model probability by up to about 7.5%. This looks like it's about the increase in the curve from income group 4 to group 5.

```
library(boot)
inv.logit(-1.27 + 0.298 * 4)

## [1] 0.4805099
inv.logit(-1.27 + 0.298 * 5)

## [1] 0.5547792
```

#### 7.1.4 Weighted regression

For various reasons like design and nonresponse bias, modern survey results are rarely a true simple random sample from the population. To adjust for groups being underrepresented or overrepresented in a sample, surveys results are **weighted**. In particular, the ANES variable VCF0009z contains weights to make the sample resemble the demographics of the Current Population Survey. (Note that this doesn't remove all biases, e.g. it doesn't account for people lying to you about whether they voted, so further adjustments using a voter file would improve things further.)

Using weights in logistic (and linear) regression in R is easy: just use the **weights** argument. Technically once we have weights we're no longer fitting a binomial, so use **family = quasibinomial** (this actually doesn't make a numerical difference here, but it doesn't give a warning message.)

```
Bush.weighted.logit = glm(Republican ~ income, family = quasibinomial, weights = survey.weights,
  data = ANES1992)
summary(Bush.weighted.logit)

##
## Call:
## glm(formula = Republican ~ income, family = quasibinomial, data = ANES1992,
##       weights = survey.weights)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.3005   -1.0331   -0.8763    1.2091    1.7976
```

```

## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -1.27068   0.17721 -7.170 1.27e-12 ***
## income       0.29908   0.05373  5.566 3.18e-08 ***
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for quasibinomial family taken to be 0.9998299)
## 
## Null deviance: 1715.7 on 1266 degrees of freedom
## Residual deviance: 1683.5 on 1265 degrees of freedom
##   (1014 observations deleted due to missingness)
## AIC: NA
## 
## Number of Fisher Scoring iterations: 4

```

The weights only make a small difference to the coefficients. Does this make much difference to the fit? We write a function to describe the fit:

```

our.logit = function(x) {
  coe = coef(Bush.weighted.logit)
  y = coe[1] + coe[2] * x
  return(exp(y)/(1 + exp(y)))
}

```

Now plot the unweighted and weighted fits.

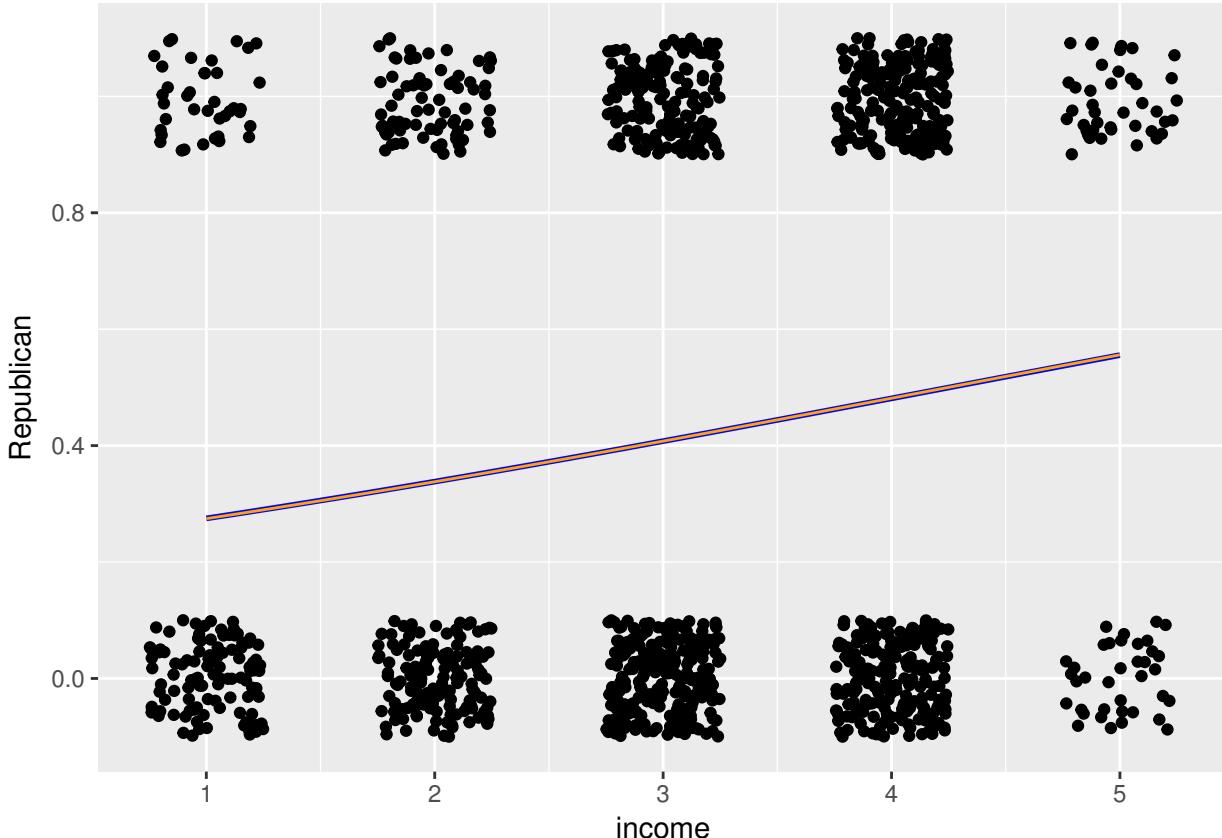
```

ggplot(ANES1992, aes(x = income, y = Republican)) + geom_jitter(height = 0.1,
  width = 0.25) + geom_smooth(method = "glm", method.args = list(family = "binomial"),
  se = FALSE, color = "blue") + stat_function(fun = our.logit, color = "orange")

```

```
## Warning: Removed 1014 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 1014 rows containing missing values (geom_point).
```



The weighted and unweighted fits are nearly indistinguishable. This is quite often the case when creating regression models. (On the other hand, if you're just doing simple summaries of the data, weights can make a big difference.) If someone has gone to the trouble of finding weights for you, they're easy to use, so you should use them.

### 7.1.5 Fitting a series of regressions

We're not just interested in 1992. We want to know the relationship between income and vote for every Presidential election we have data for – is the relationship similar for every election, or are some elections different? Has there been a consistent change over time?

In programming terms, the easiest way to fit the same model on many subsets of the data is to write a function that subsets the data and fits the model, then to write a `for` loop to fit the model for each subset. If you're literally going to run out of RAM, there are much more computationally efficient approaches, but otherwise more efficiency usually isn't worth the effort.

Here's a function to fit our weighted logistic regression of vote on income for any given year.

```
logit.ANES.subset = function(my.year, data) {
  newdata = subset(data, year == my.year)
  newdata = subset(newdata, income > 0)
  model = glm(Republican ~ income, family = quasibinomial, weights = survey.weights,
             data = newdata)
  output = c(my.year, summary(model)$coef[2, 1:2])
  return(output)
}
```

The function returns the year, the model's coefficient for income, and the standard error of that coefficient.

We shouldn't take the standard error too literally, because we haven't accounted for the complex design of the ANES surveys – if you really care about getting these right, take a sampling course.

Let's test the function out on Bush-Clinton.

```
logit.ANES.subset(my.year = 1992, data = ANES.df)
```

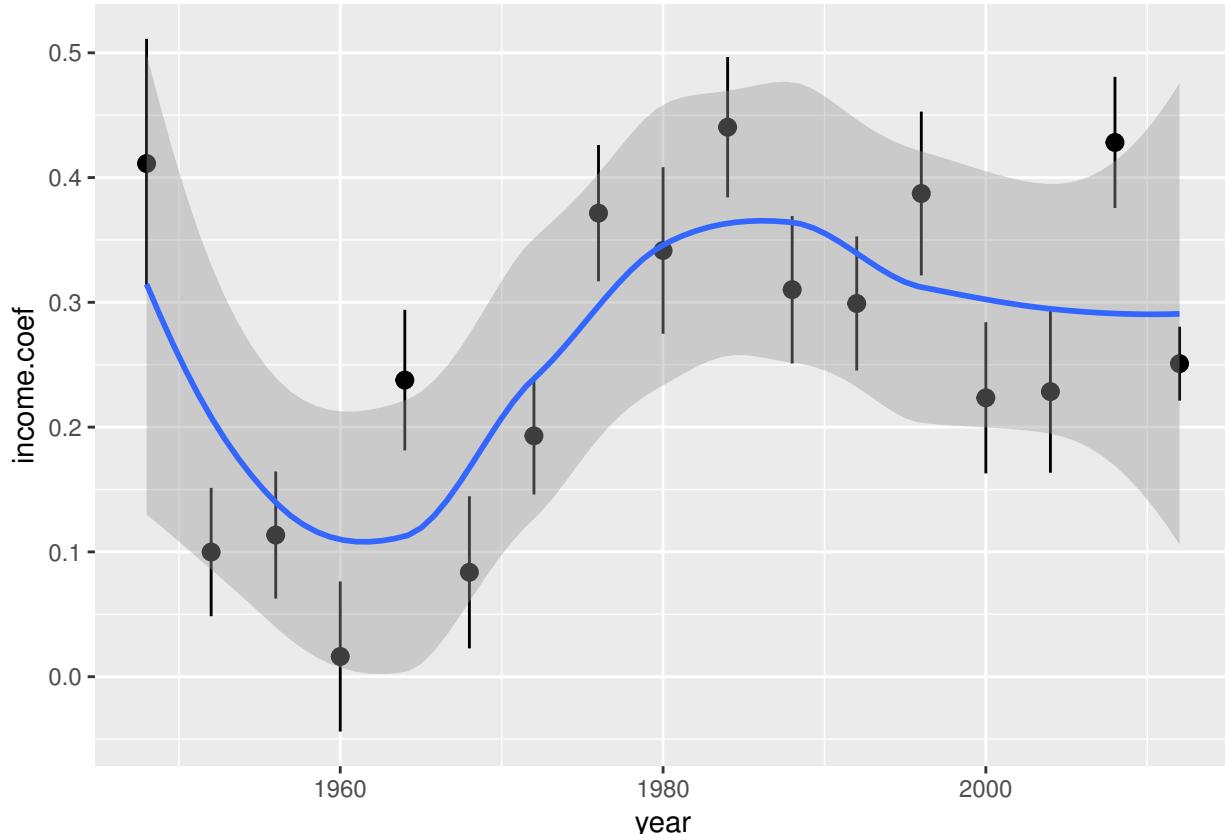
```
##           Estimate Std. Error
## 1.992000e+03 2.990845e-01 5.373261e-02
```

The “estimate” is the same as the weighted regression we fitted above, so it seems the function is working fine. Now we want to run the function for every Presidential election year from 1948 to 2012.

```
years = seq(1948, 2012, 4)
n = length(years)
income.by.year = data.frame(year = rep(NA, n), income.coef = rep(NA, n), income.se = rep(NA, n))
for (J in 1:n) {
  my.year = years[J]
  income.by.year[J, ] = logit.ANES.subset(my.year = my.year, data = ANES.df)
}
```

We'll display the results using ggplot. The nifty `geom_pointrange()` lets us add one standard error bounds. Again, we shouldn't take these too literally, just use them to get a ballpark idea of uncertainty.

```
ggplot(income.by.year, aes(x = year, y = income.coef, ymin = income.coef - income.se, ymax = income.coef + income.se)) + geom_pointrange() + geom_smooth(method.args = list(family = "sym"))
```



The income coefficient is positive for every election, meaning richer people were more likely to vote Republican every time (though 1960 was close.) The general trend was an increase in the income coefficient from 1952 to 1984, then a leveling-off. There was a huge drop from 1948 to 1952; unfortunately we don't have data

from before 1948 to know if the election was typical. Otherwise there are a couple of elections outside the confidence band: 1964 (Johnson over Goldwater) and 2008 (Obama over McCain.)

### 7.1.6 Less modeling, more detail

In our regressions, we treated `income` as a quantitative variable. A simpler approach would be to treat it as a factor, and simply track the weighted proportion of each income group that (two-party) voted Republican by year. Again, this is easiest to program using a `for` loop.

Remarkably, in R finding and using a bunch of weighted means is harder than fitting a bunch of weighted logistic regressions. I used'll use `weighted.mean()` in conjunction with `summarise()` in `dplyr` to do the computation, but if you have any better ideas let me know. Let's first work out how to do it for the 1992 data.

```
library(dplyr)
summarise(group_by(ANES1992, income), weighted.mean(Republican, w = survey.weights,
na.rm = TRUE))
```

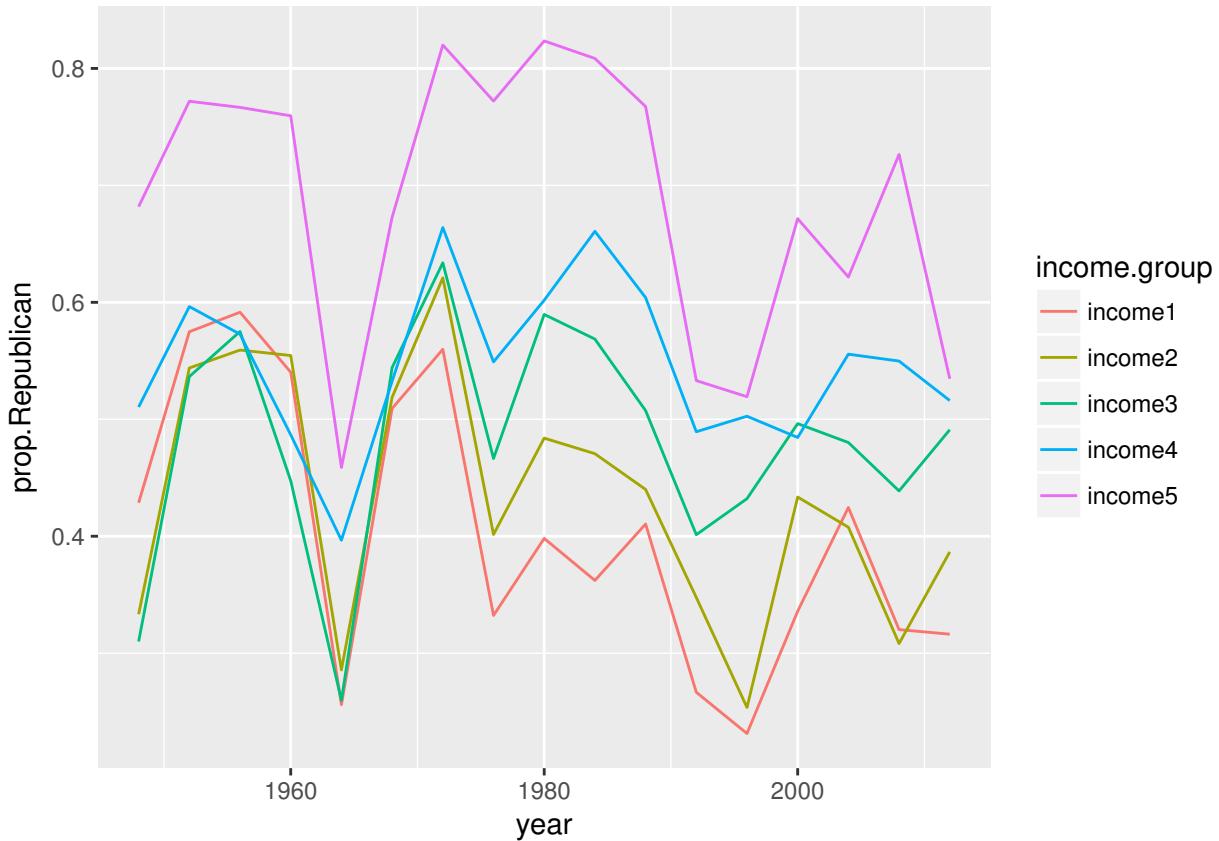
```
## # A tibble: 5 × 2
##   income `weighted.mean(Republican, w = survey....`<dbl>
##   <dbl>
## 1     1             0.2666117
## 2     2             0.3476338
## 3     3             0.4013024
## 4     4             0.4893364
## 5     5             0.5330874
```

We don't want the "0" income category so we'll remove it when we do the subsetting for each year. Now write a `for` loop to do this for every Presidential election year and write the results as a data frame.

```
n = length(years)
income.prop.by.year = data.frame(year = rep(NA, n), income1 = rep(NA, n), income2 = rep(NA, n),
income3 = rep(NA, n), income4 = rep(NA, n), income5 = rep(NA, n))
for (J in 1:n) {
  my.year = years[J]
  ANES.my.year = subset(ANES.df, year == my.year & income > 0)
  income.prop.by.year[J, 1] = my.year
  income.summary = summarise(group_by(ANES.my.year, income), weighted.mean(Republican,
w = survey.weights, na.rm = TRUE))
  income.summary = as.data.frame(income.summary)
  income.prop.by.year[J, 2:6] = income.summary[, 2]
}
```

Put the results in long form and plot them:

```
library(tidyr)
income.prop.long = income.prop.by.year %>% gather(income.group, prop.Republican,
income1:income5)
ggplot(income.prop.long, aes(x = year, y = prop.Republican, group = income.group,
color = income.group)) + geom_line()
```

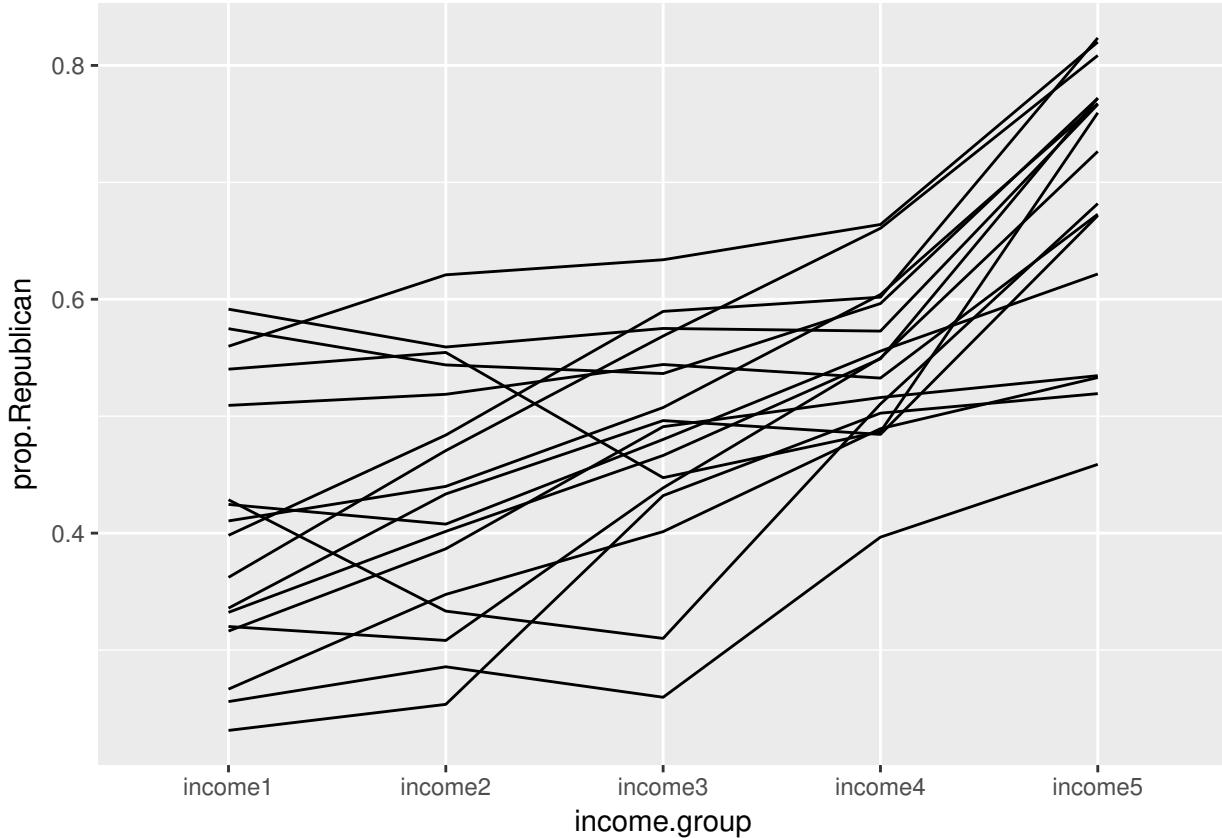


We now have a bit more detail on the trends and the aberrant results.

- The top income group is reliably the most Republican, but the bottom income group isn't always the most Democratic (although it was in the middle part of the time period.)
- In 1948 there were pretty big differences between income groups, but in the 1950s the differences between all groups except the richest were small. It's guess work whether 1948 was an aberration or whether the small income differences from 1952 to 1968 were historical unusual (though I suspect it's the latter.)
- The big coefficient for 1964 (compared to the elections before and after) might be in part an artifact of the logit scale.
- In 2008 there really was a big difference between income group, which is likely attributable to the financial crisis.

We can also draw lines to connect income groups by year. (We won't try to distinguish the years by color since there are so many of them.)

```
ggplot(income.prop.long, aes(x = income.group, y = prop.Republican, group = year)) +
  geom_line()
```



This yields less insight, but still has interesting features: notably the big magnitude of the uptick in Republicanism for the highest income group for almost every year.

### 7.1.7 Data summaries vs. models

Both data summaries (like our last plot) and models (like our logistic regressions) have their uses.

- Data summaries require fewer assumptions, and often give you a fuller picture than a model. However, they can be noisy or just too complicated to easily get a grip on.
- Models require assumptions, so in addition to being reductive, there's more potential for things to go horribly wrong. However, models can be an easy way into the data: everything gets summarized in a couple of parameters, and you can put your effort into understanding what those parameters really mean. Furthermore, complexity can easily be added to models – for example, it's easy to build a logistic regression model with multiple predictors (as we'll see in the next set of notes.)

In practice, going back and forth between models and data summaries, as we did here, is often very useful in exploratory work. Models can usefully simplify the data so you can get the big picture. Then you can look a fuller data summary and bear in mind that the big picture doesn't explain.

## 7.2 Logistic regression with multiple predictors

Optional reading: Gelman & Hill pp. 86–104.

### 7.2.1 Arsenic in wells

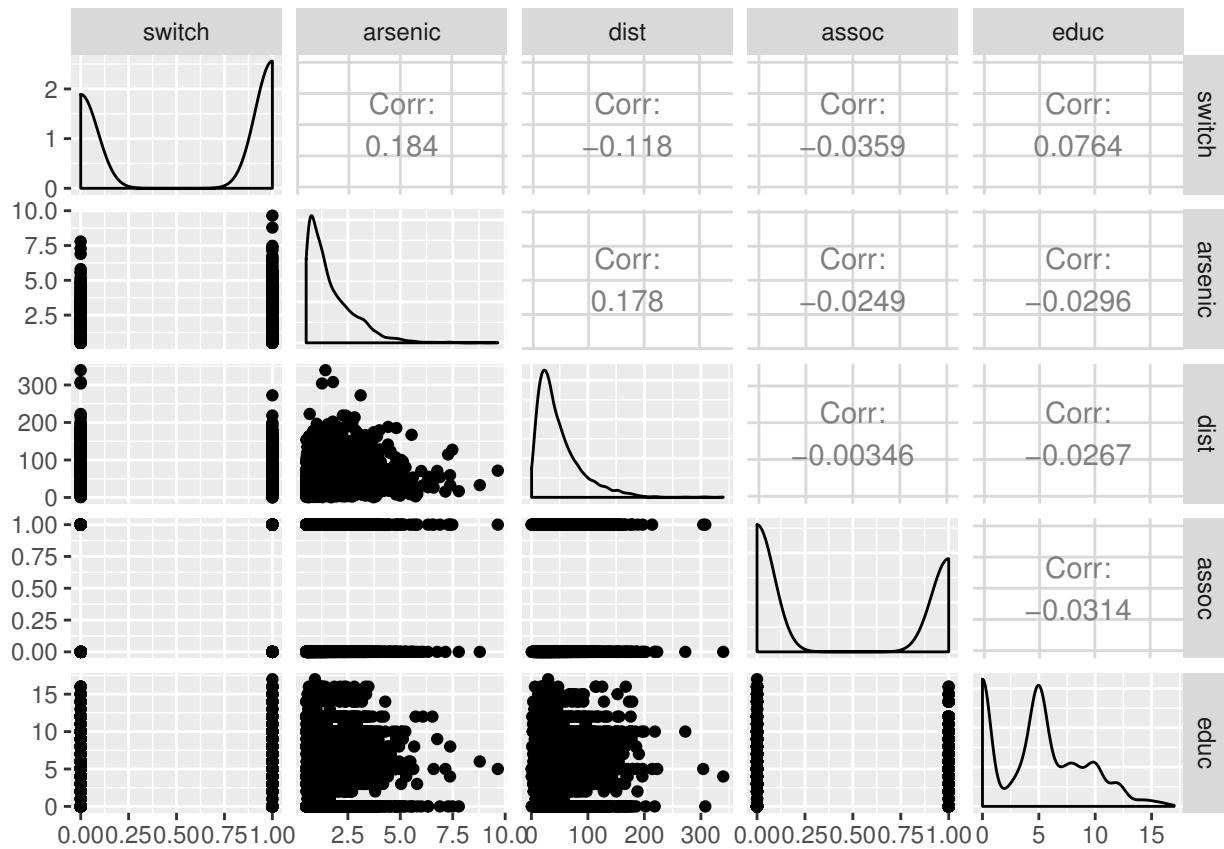
The file `wells.dat` contains data on 3,020 households in Bangladesh whose wells had arsenic levels above the national drinking water standard of 50 micrograms per liter. The variables are:

- `switch`: the response: whether or not the household switched after being encouraged to do so (after high arsenic was measured.)
- `arsenic`: the arsenic concentration in hundreds of micrograms per liter.
- `dist`: the distance to the nearest safe well in meters.
- `assoc`: whether anyone in the household is in a community association.
- `educ`: the years of education of the head of the household.

We load at the data and look at the numerical summary and a pairs plot:

```
library(GGally)
wells = read.table("wells.dat")
summary(wells)
```

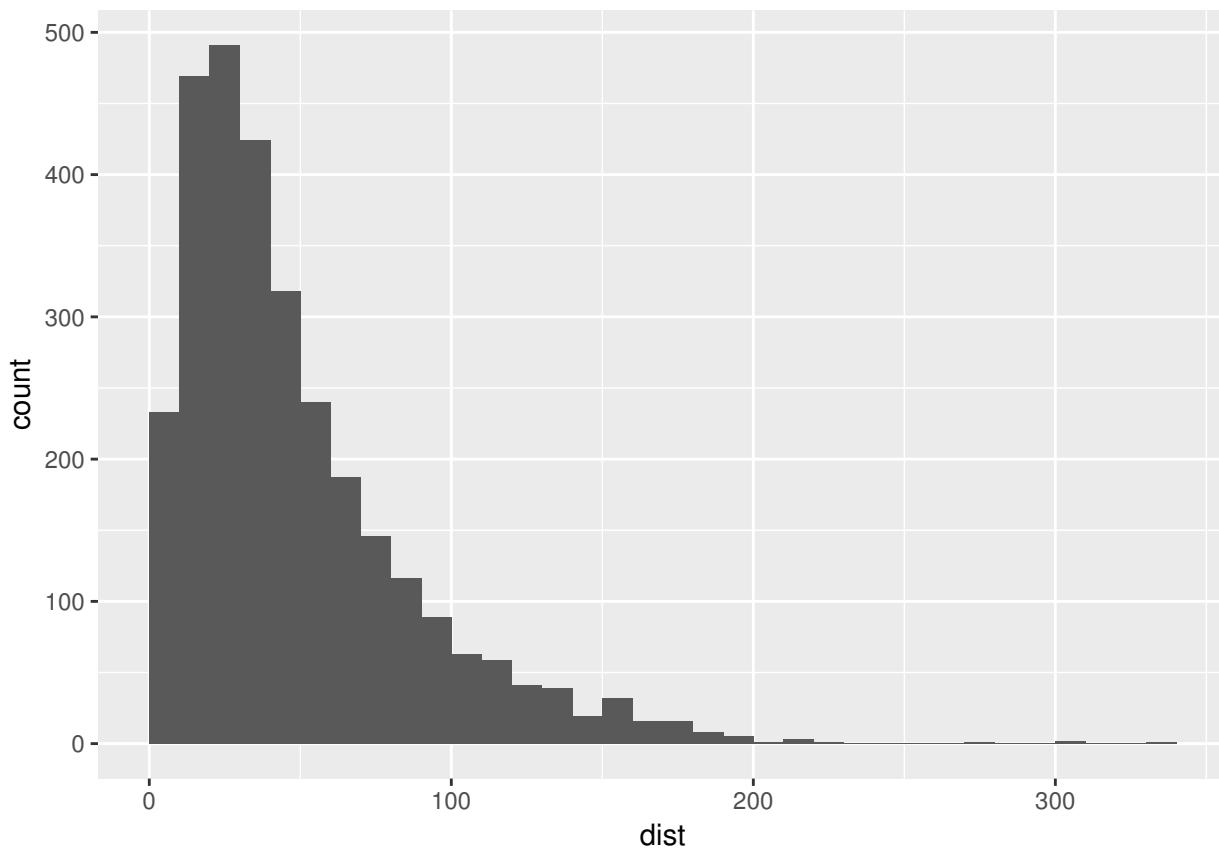
```
##      switch      arsenic       dist      assoc
##  Min.   :0.0000  Min.   :0.510  Min.   : 0.387  Min.   :0.0000
##  1st Qu.:0.0000  1st Qu.:0.820  1st Qu.: 21.117  1st Qu.:0.0000
##  Median :1.0000  Median :1.300  Median : 36.761  Median :0.0000
##  Mean   :0.5752  Mean   :1.657  Mean   : 48.332  Mean   :0.4228
##  3rd Qu.:1.0000  3rd Qu.:2.200  3rd Qu.: 64.041  3rd Qu.:1.0000
##  Max.   :1.0000  Max.   :9.650  Max.   :339.531  Max.   :1.0000
##
##      educ
##  Min.   : 0.000
##  1st Qu.: 0.000
##  Median : 5.000
##  Mean   : 4.828
##  3rd Qu.: 8.000
##  Max.   :17.000
ggpairs(wells)
```



As we might expect, distance and education level are positively associated with switching, while arsenic level is negatively associated. Surprisingly, the correlation between belonging to an association and switching is negative, though the relationship is very weak.

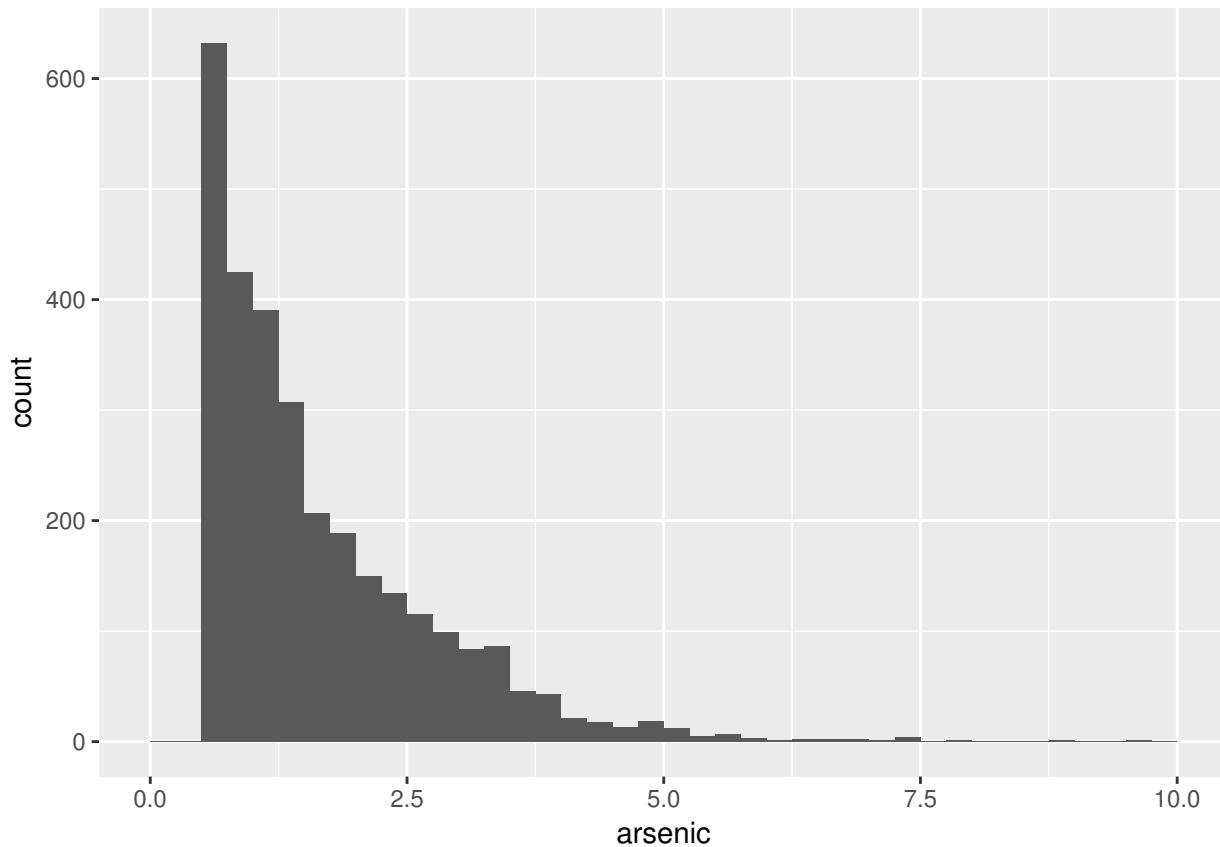
Distance and arsenic levels should be the best predictors, so let's look more closely at the distributions of those variables.

```
ggplot(wells, aes(x = dist)) + geom_histogram(breaks = seq(0, 340, 10))
```



We see the distribution of distance peaks around 40–60 meters, then is strongly right-skewed. We should keep the possibility of a transformation in mind.

```
ggplot(wells, aes(x = arsenic)) + geom_histogram(breaks = seq(0, 10, 0.25))
```

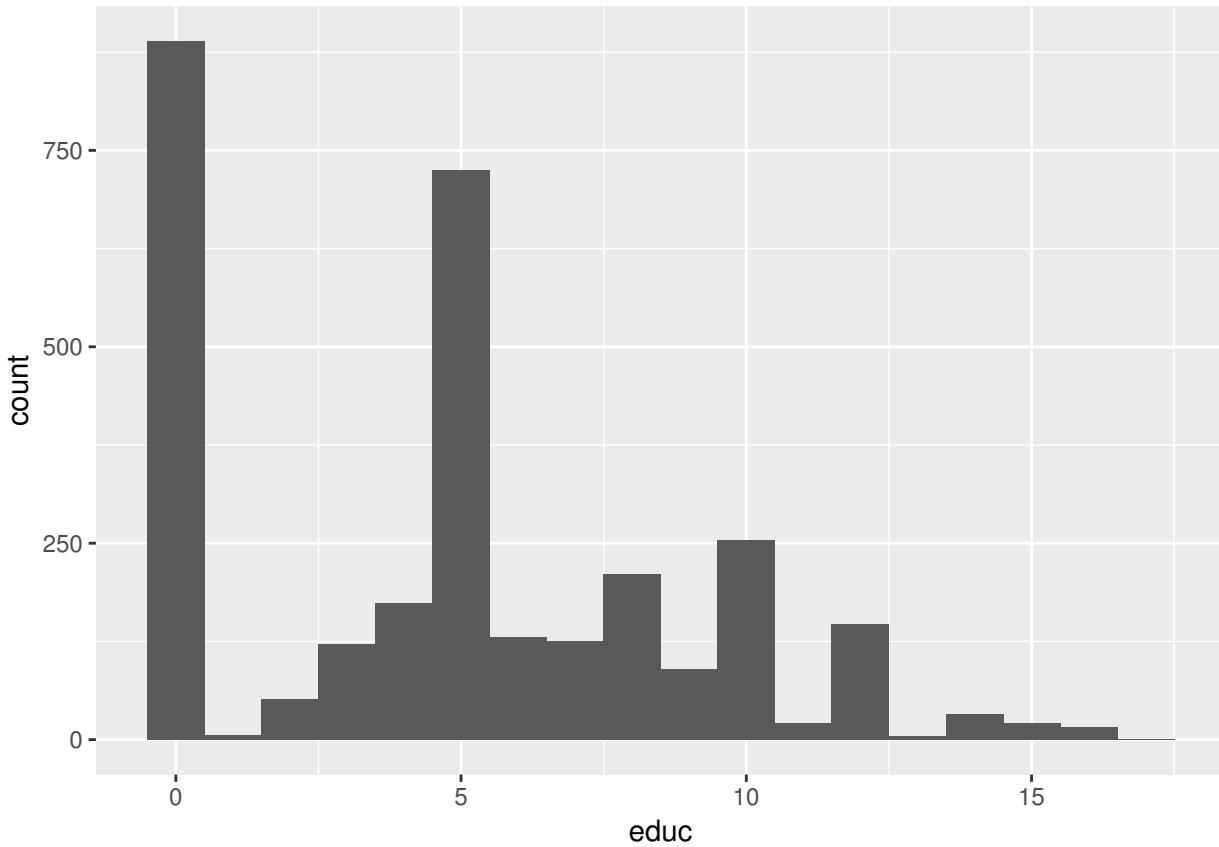


There are no observations with arsenic below 0.5, because these are considered “safe” and the households don’t get advised to switch wells. Otherwise, the distribution is strongly right-skewed, suggesting a transformation.

Note that the scales of `dist` and `arsenic` are quite different. Gelman and Hill suggest standardizing the data for interpretability, so that predictors are on approximately the same scale. That’s not a bad idea but we won’t bother.

Here’s education:

```
ggplot(wells, aes(x = educ)) + geom_histogram(breaks = -0.5:17.5)
```



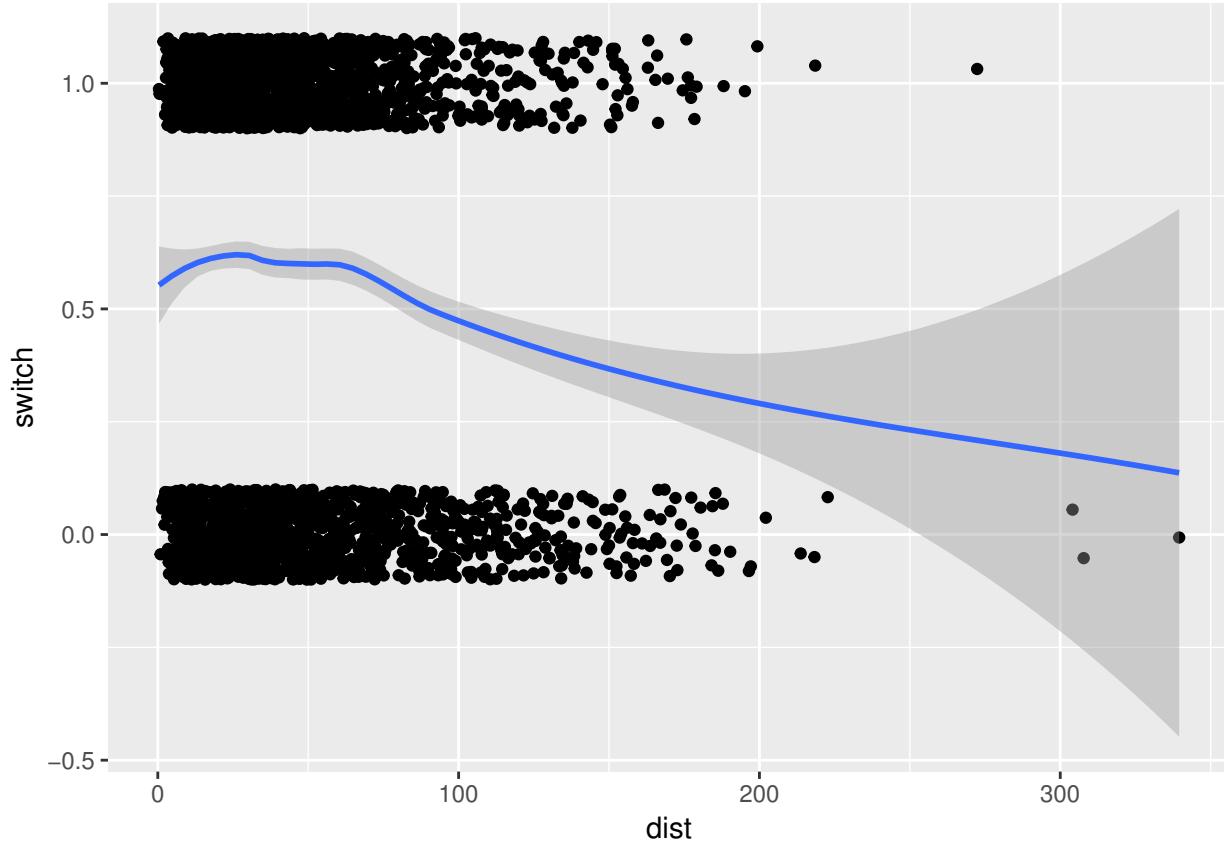
The distribution is weird. 0 (no education) and 5 (primary school only) are the magic numbers.

### 7.2.2 Starting off simple

One way of approaching EDA is to throw everything into your model and then get rid of terms you don't need. Another strategy is to start with simple models and gradually build up to complex models. Let's take this second approach for now.

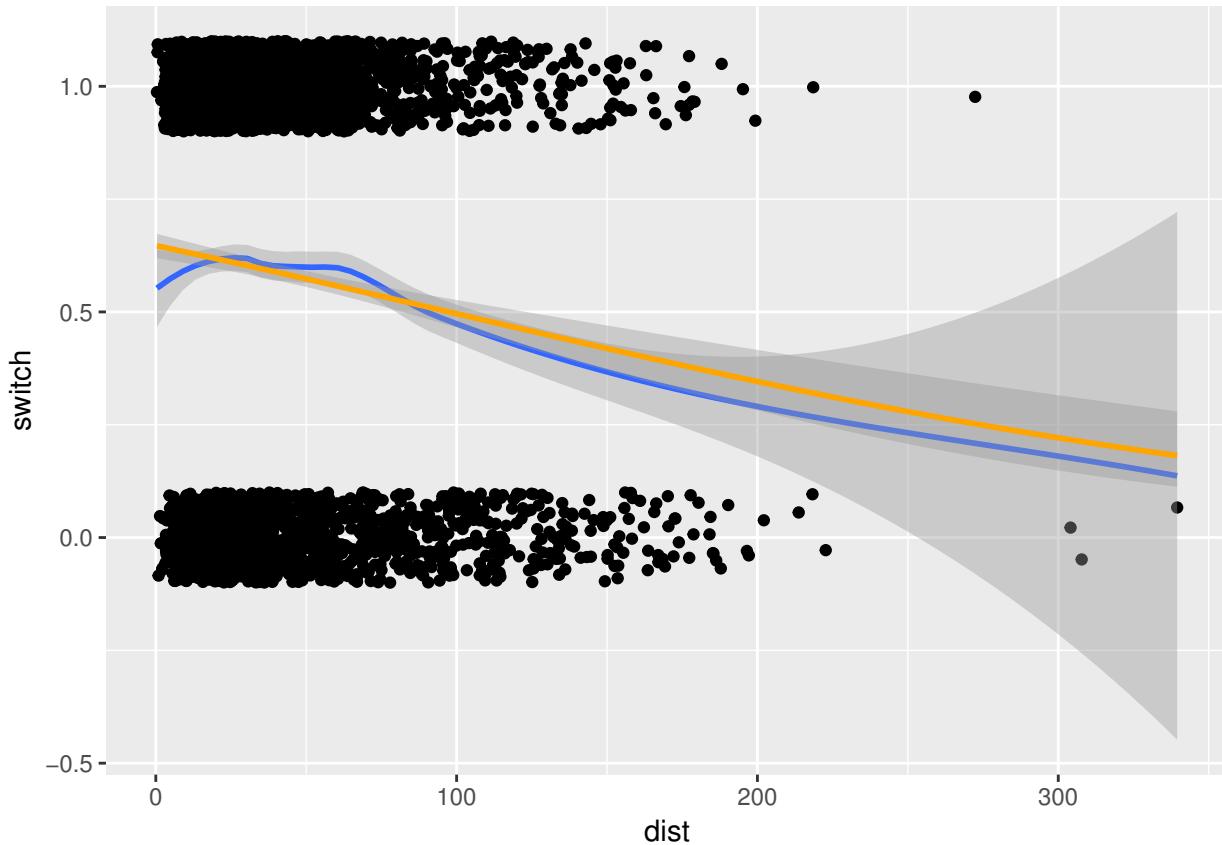
We'll start by predict switching, using distance as the only response. There's no reason why we can't try loess:

```
ggplot(wells, aes(x = dist, y = switch)) + geom_jitter(width = 0, height = 0.1) +
  geom_smooth(method = "loess")
```



We can interpret the fit as the probability of switching wells, given the distance to the nearest safe well. There's a bump for very low distances (which is probably just noise) and then a decline. We could play around with the smoothing parameters to get something more pleasant-looking. Instead, we'll fit a logistic regression to guarantee a decreasing relationship.

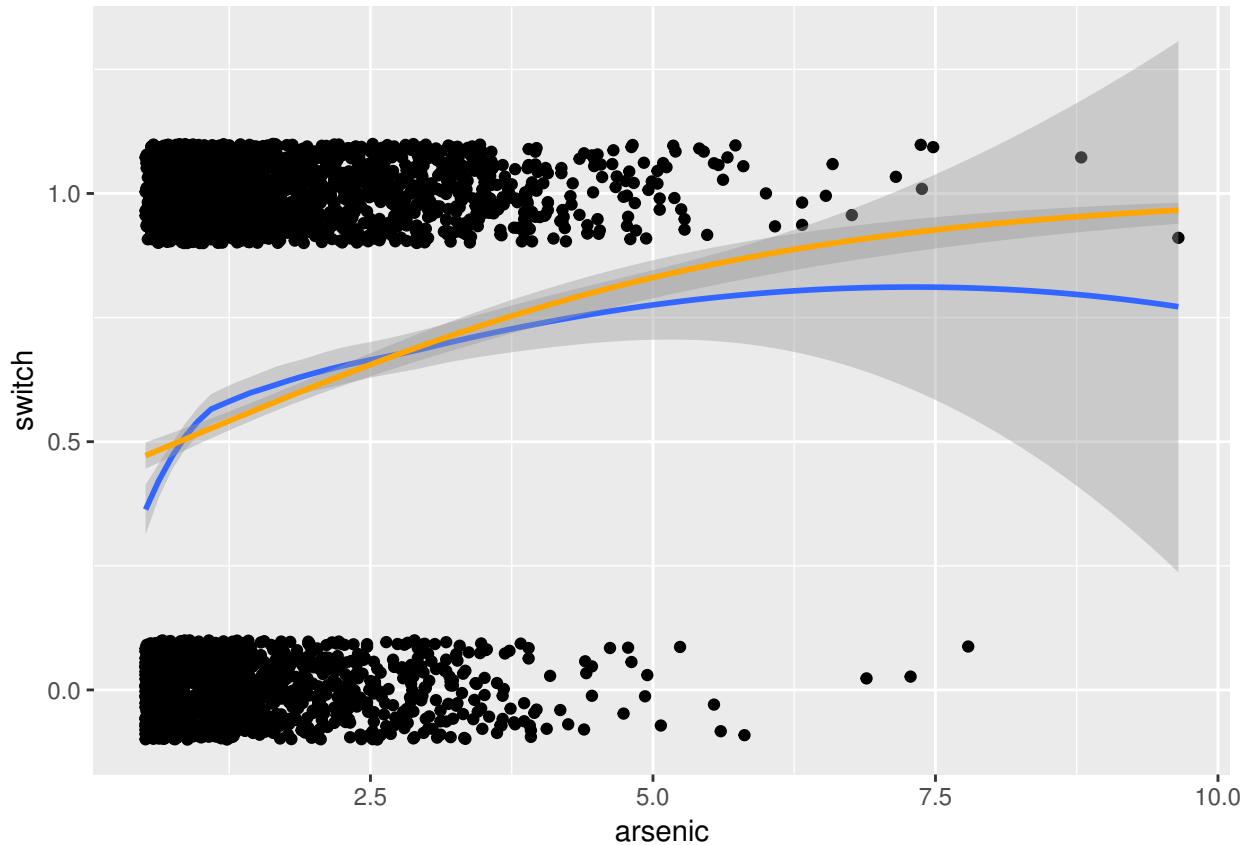
```
ggplot(wells, aes(x = dist, y = switch)) + geom_jitter(width = 0, height = 0.1) +
  geom_smooth(method = "loess") + geom_smooth(method = "glm", method.args = list(family = "binomial"),
  color = "orange")
```



The parametric method imposes a functional form on the data. This aids in interpretability: the fit can now be described in a couple of parameters, and the weird bump at the beginning is gone. The cost is some lack of fit: perhaps the probability should decrease more quickly as the distance gets beyond about 60 meters (remember that water is heavy and it's tiring to carry buckets back and forth over long distances.)

Let's try the same kind of approach using arsenic concentration as the sole predictor.

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

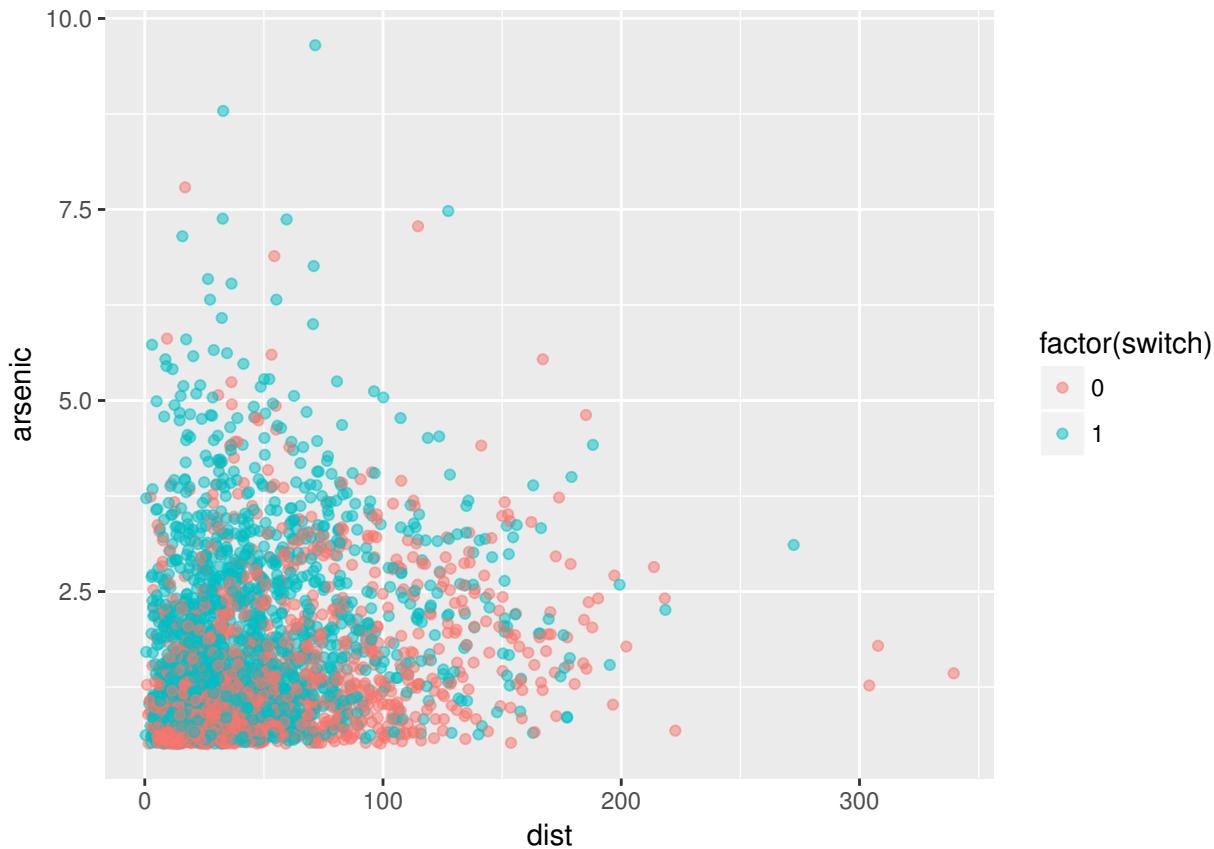


The fits diverge at arsenic levels above about 5. This seems to be because of a couple of extremely high arsenic observations which led to switching. Note however that not much of the data has arsenic above 5, so it remains to be seen how much this matters.

### 7.2.3 Two predictors

We now want to know how the chance of switching depends on distance and arsenic simultaneously. Before fitting a model, we plot the data to get a feel for it. We'll use color to distinguish between households that switch and households that don't.

```
ggplot(wells, aes(x = dist, y = arsenic, color = factor(switch))) + geom_point(alpha = 0.5)
```



The `alpha = 0.5` makes the points slightly transparent, which can help visually when you have a lot of data. Still, it's hard to work out how switching depends on the other two variables from this graph; that's why we're fitting a model. The main thing to take home from the graph is the lack of data in the top right: we have no observations at all with both distance above 200 and arsenic above 4, so we should not try to generalize to this region.

Now we'll fit a logistic regression using both distance and arsenic as predictors. We first fit an additive model, i.e. with no interaction.

```
switch.logit = glm(switch ~ dist + arsenic, family = "binomial", data = wells)
summary(switch.logit)
```

```
##
## Call:
## glm(formula = switch ~ dist + arsenic, family = "binomial", data = wells)
##
## Deviance Residuals:
##      Min        1Q     Median        3Q       Max
## -2.6351   -1.2139    0.7786    1.0702    1.7085
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 0.002749  0.079448  0.035   0.972
## dist       -0.008966  0.001043 -8.593 <2e-16 ***
## arsenic     0.460775  0.041385 11.134 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

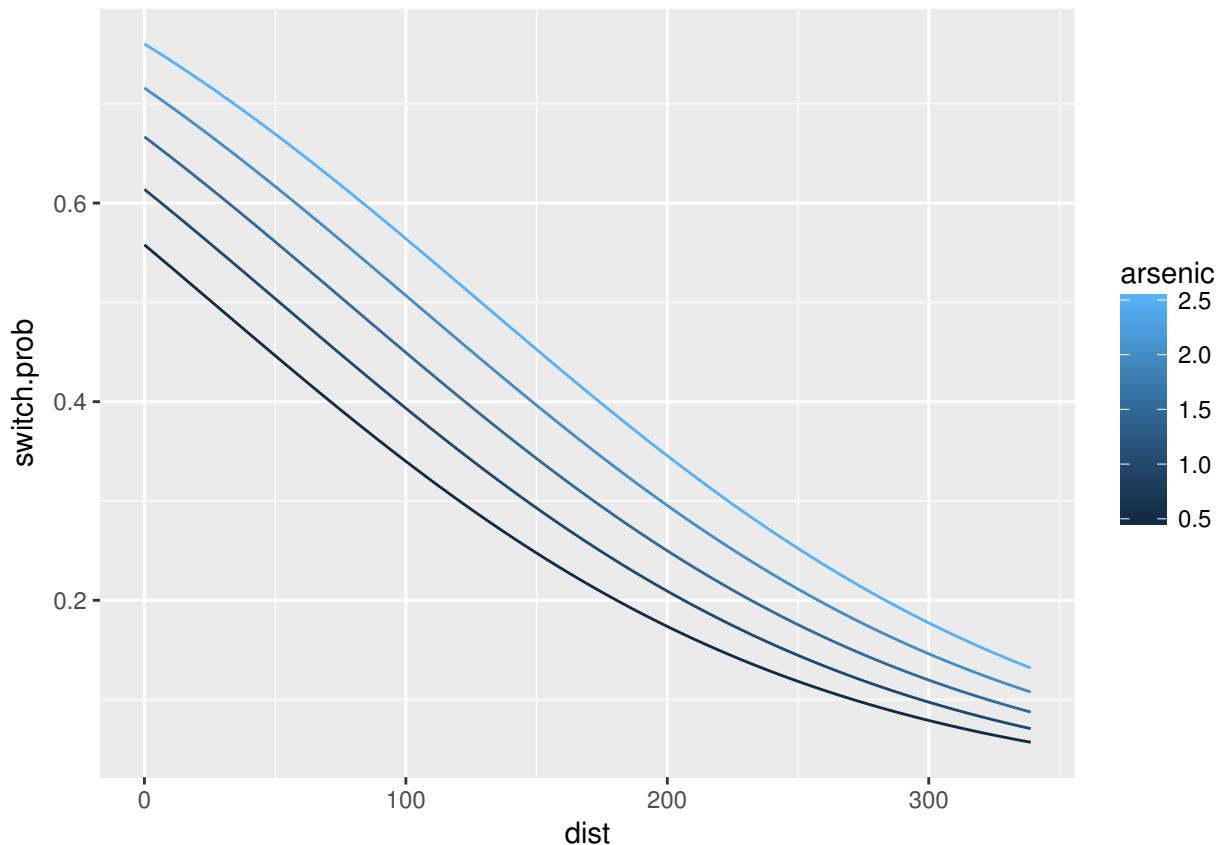
```
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 4118.1 on 3019 degrees of freedom
## Residual deviance: 3930.7 on 3017 degrees of freedom
## AIC: 3936.7
##
## Number of Fisher Scoring iterations: 4
```

As always, the coefficients are the most important things here. According to the model,

$$\text{logit}[P(\text{switch}|\text{dist}, \text{arsenic})] = 0.003 - 0.00897 \times \text{dist} + 0.4608 \times \text{arsenic}.$$

As in the continuous case, we visualize the fit by drawing multiple curves representing different values of one of the predictors. Let's display the switching probability as a function of distance for a few values of arsenic. Note that when we use the `predict()` function, we specify `type = "response"` to get the probabilities and not their logits.

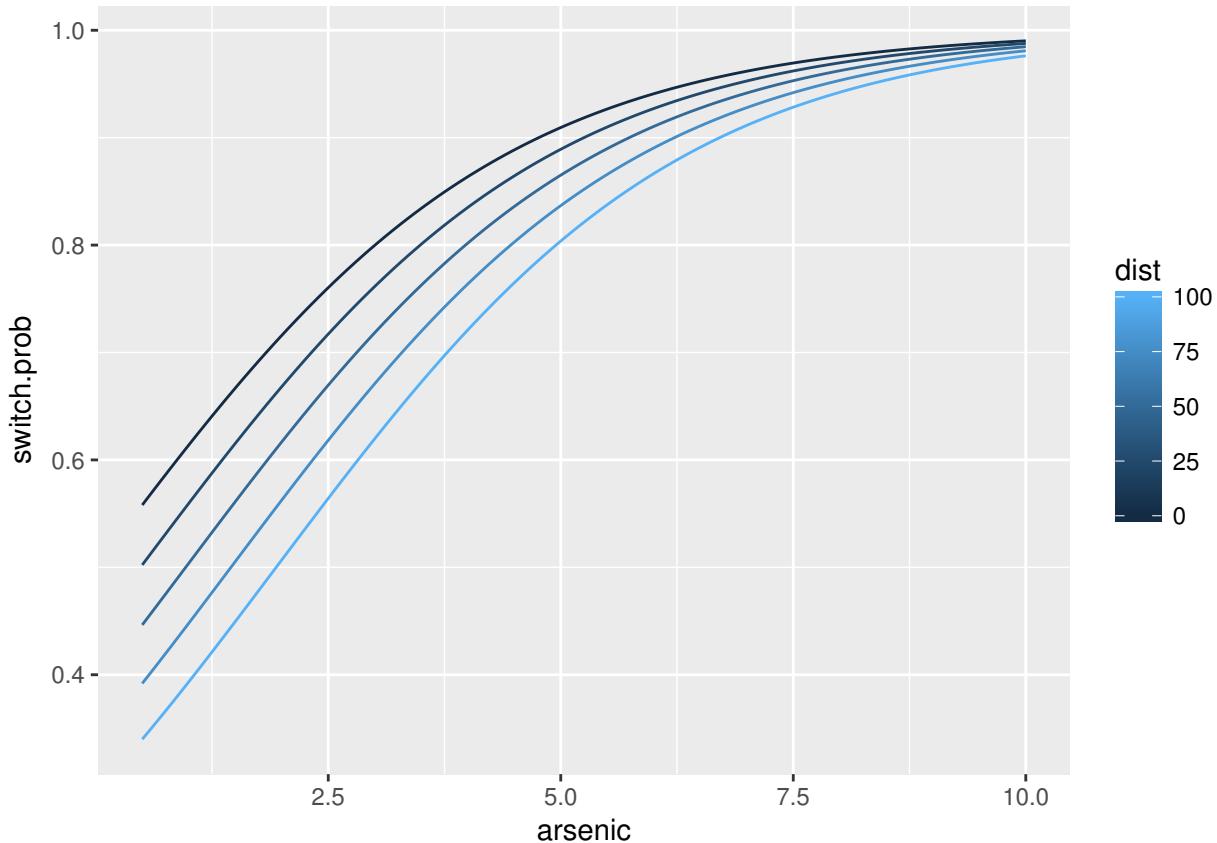
```
dist.df = expand.grid(dist = 0:339, arsenic = seq(0.5, 2.5, 0.5))
dist.pred = predict(switch.logit, type = "response", newdata = dist.df)
dist.pred.df = data.frame(dist.df, switch.prob = as.vector(dist.pred))
ggplot(dist.pred.df, aes(x = dist, y = switch.prob, group = arsenic, color = arsenic)) +
  geom_line()
```



The median arsenic level is 1.3. If arsenic is near the median, the probability of switching declines from 60-something percent if you're right by a safe well to less than 10% if the nearest safe well is hundreds of meters away.

Now find prediction curves for arsenic for a few difference distances:

```
arsenic.df = expand.grid(arsenic = seq(0.5, 10, 0.01), dist = seq(0, 100, 25))
arsenic.pred = predict(switch.logit, type = "response", newdata = arsenic.df)
arsenic.pred.df = data.frame(arsenic.df, switch.prob = as.vector(arsenic.pred))
ggplot(arsenic.pred.df, aes(x = arsenic, y = switch.prob, group = dist, color = dist)) +
  geom_line()
```



The median distance to a safe well is about 37 meters. At that distance, if the arsenic level is only just over the safety threshold, it's about 50–50 whether a household switched. On the other hand, at the highest levels of arsenic, households will almost certainly switch even if the nearest safe well is quite far.

#### 7.2.4 Adding an interaction

We now add an interaction term between distance and arsenic.

```
switch.int = glm(switch ~ dist * arsenic, family = "binomial", data = wells)
summary(switch.int)
```

```
##
## Call:
## glm(formula = switch ~ dist * arsenic, family = "binomial", data = wells)
##
## Deviance Residuals:
##      Min        1Q     Median        3Q       Max
## -2.7823   -1.2004    0.7696    1.0816    1.8476
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
```

```

## (Intercept) -0.147868  0.117538 -1.258  0.20838
## dist        -0.005772  0.002092 -2.759  0.00579 **
## arsenic      0.555977  0.069319  8.021 1.05e-15 ***
## dist:arsenic -0.001789  0.001023 -1.748  0.08040 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 4118.1 on 3019 degrees of freedom
## Residual deviance: 3927.6 on 3016 degrees of freedom
## AIC: 3935.6
##
## Number of Fisher Scoring iterations: 4

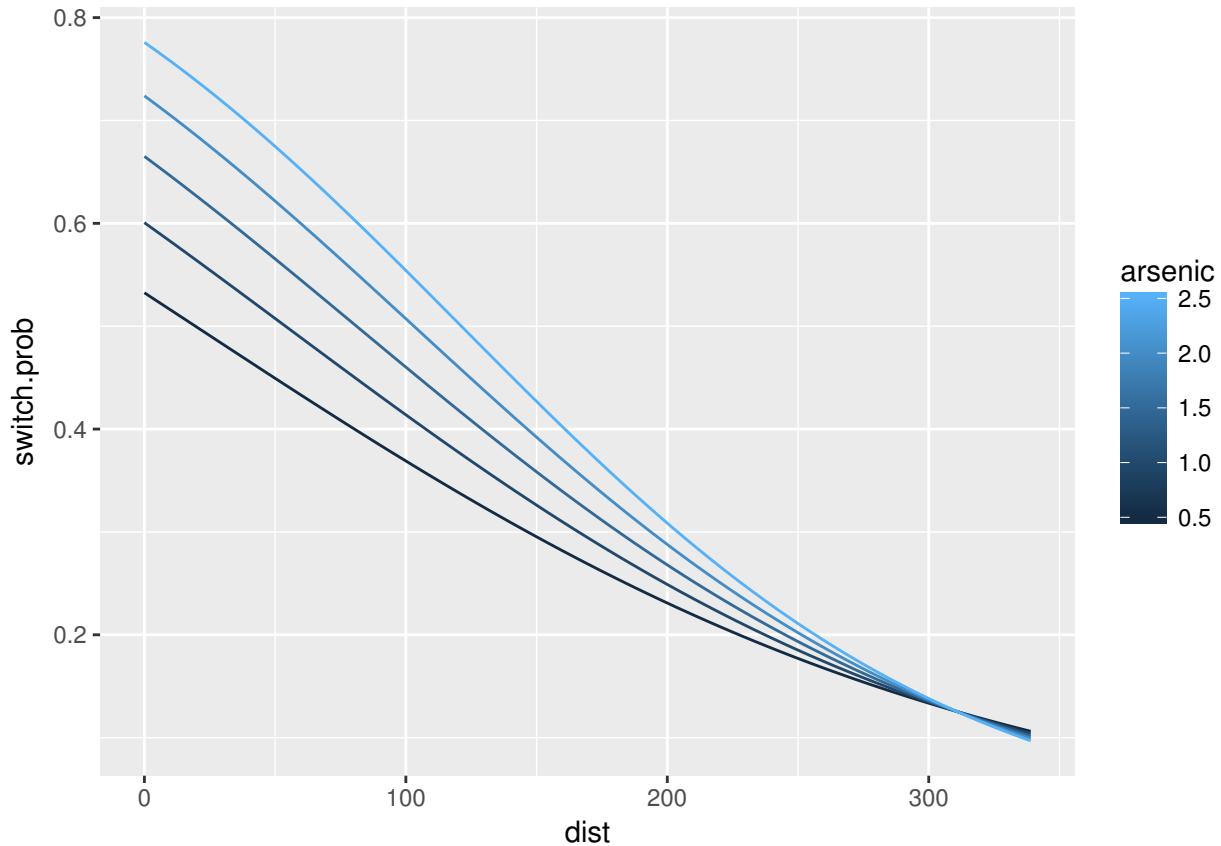
```

The numbers are a bit hard to interpret. For example, the sign of the interaction term is negative, but it's hard to know exactly what this means (especially since the signs for distance and arsenic go in different directions.) Let's just plot curves.

```

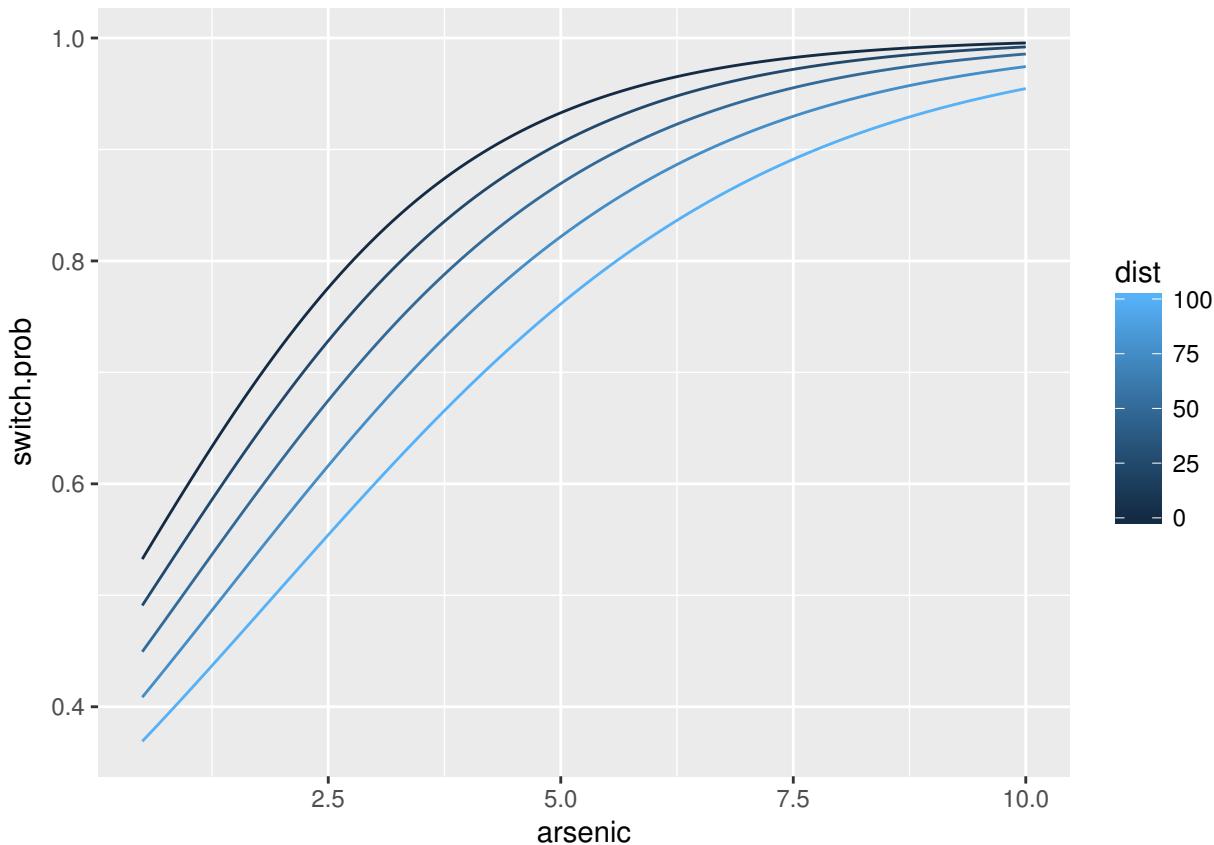
dist.int.pred = predict(switch.int, type = "response", newdata = dist.df)
dist.int.pred.df = data.frame(dist.df, switch.prob = as.vector(dist.int.pred))
ggplot(dist.int.pred.df, aes(x = dist, y = switch.prob, group = arsenic, color = arsenic)) +
  geom_line()

```



The interaction brings the curves together as distance increases. If the nearest safe well is close, it makes a big difference whether the arsenic concentration is just over the limit or much bigger. If the nearest safe well is far, it makes little difference: people are unlikely to switch no matter the concentration. The curves meet up beyond 300 meters, though we only have three observations where the distance exceeds 300 meters.

```
arsenic.int.pred = predict(switch.int, type = "response", newdata = arsenic.df)
arsenic.int.pred.df = data.frame(arsenic.df, switch.prob = as.vector(arsenic.int.pred))
ggplot(arsenic.int.pred.df, aes(x = arsenic, y = switch.prob, group = dist,
  color = dist)) + geom_line()
```



The curves actually get further apart at first as arsenic increases (up to a point.) That is, the curves for short distances rise quite quickly as arsenic increases, while the curves for long distances rise more slowly. Eventually, for exceptionally high levels of arsenic, the curves come together (because they can't go any higher than 1.)

### 7.2.5 Lots of predictors

Distance and arsenic are both good predictors and there's no reason to think there shouldn't be an interaction term. Now let's add `assoc` and `educ` to the model and see if the fit makes sense.

```
summary(glm(switch ~ dist * arsenic + assoc + educ, family = "binomial", data = wells))
```

```
##
## Call:
## glm(formula = switch ~ dist * arsenic + assoc + educ, family = "binomial",
##      data = wells)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.7303   -1.1892    0.7444    1.0675    1.6987
##
## Coefficients:
```

```

##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.291120  0.131427 -2.215  0.02675 *
## dist        -0.006081  0.002093 -2.905  0.00367 **
## arsenic      0.553238  0.069542  7.955 1.79e-15 ***
## assoc        -0.123188  0.076977 -1.600  0.10953
## educ         0.041948  0.009594  4.372 1.23e-05 ***
## dist:arsenic -0.001612  0.001022 -1.577  0.11482
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 4118.1  on 3019  degrees of freedom
## Residual deviance: 3905.4  on 3014  degrees of freedom
## AIC: 3917.4
##
## Number of Fisher Scoring iterations: 4

```

Years of education has a positive coefficient, which makes sense. We can keep that in the model.

Association membership has a *negative* coefficient, which doesn't make causal sense: it would be strange for association membership to make you *less* likely to switch wells, unless you were a member of the Association for Arsenic Being Good for You. If our sole goal was prediction, we could do a careful cross-validation to see if association membership did help predict more accurately. Since we're doing EDA, we can just drop the association term without thinking too hard about it.

```
summary(glm(switch ~ dist * arsenic + educ, family = "binomial", data = wells))
```

```

##
## Call:
## glm(formula = switch ~ dist * arsenic + educ, family = "binomial",
##      data = wells)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.7149 -1.1886  0.7478  1.0689  1.7223
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.349044  0.126360 -2.762  0.00574 **
## dist        -0.006047  0.002095 -2.886  0.00390 **
## arsenic      0.555367  0.069531  7.987 1.38e-15 ***
## educ         0.042306  0.009581  4.415 1.01e-05 ***
## dist:arsenic -0.001629  0.001023 -1.592  0.11145
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 4118.1  on 3019  degrees of freedom
## Residual deviance: 3907.9  on 3015  degrees of freedom
## AIC: 3917.9
##
## Number of Fisher Scoring iterations: 4

```

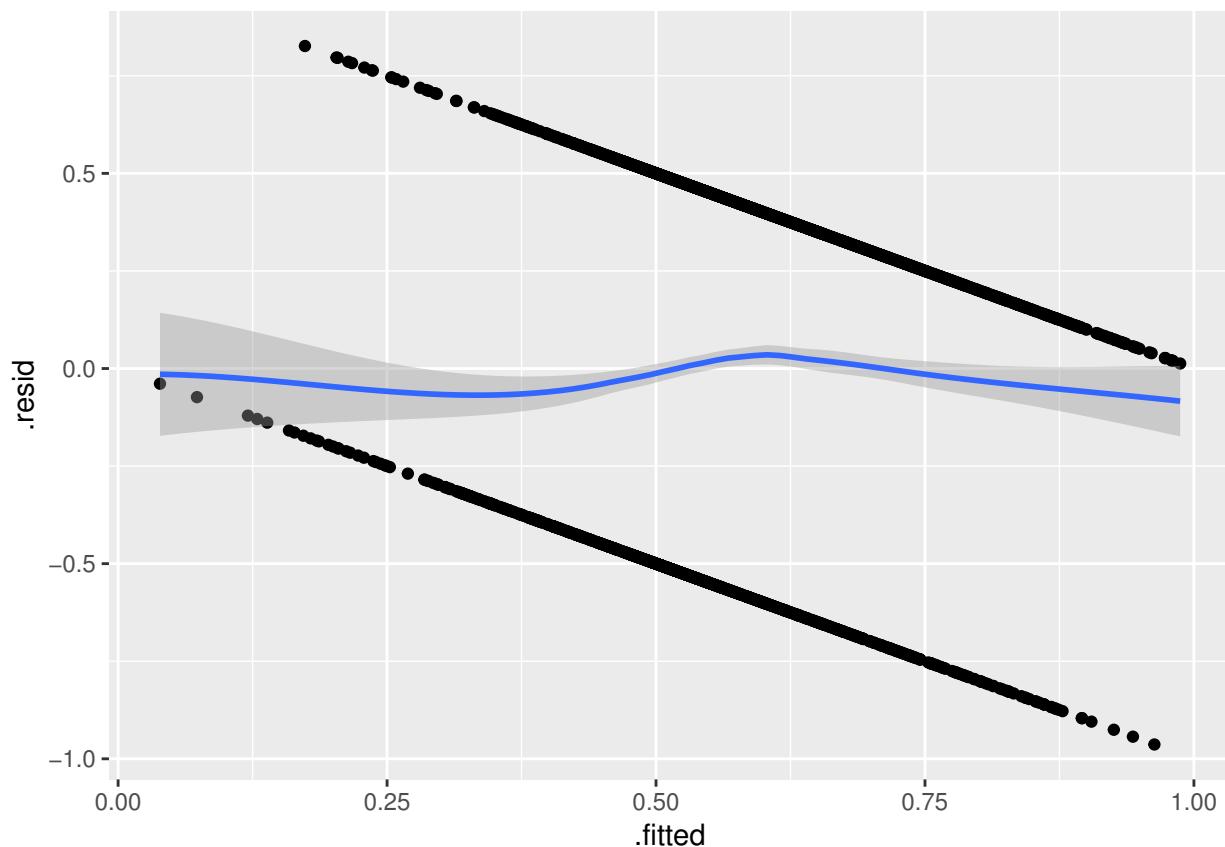
We now add the remaining two-way interactions. In general, if you only have a few terms, you might as well

include all the two-way interactions unless you have a good reason not to.

```
switch.model = glm(switch ~ dist + arsenic + educ + dist:arsenic + dist:educ +
    arsenic:educ, family = "binomial", data = wells)
```

At this point our model is too complicated for it to be worth trying to interpret the numbers. (If we had centered the variables, we might be able to do some numerical interpretation.) Let's extract the fitted values and residuals, and plot the latter against the former. Note that the default of `augment()` is to extract the deviance residuals instead of the response residuals, so we won't use `augment()`. If you paid lots of attention in S432/632 and remember what deviance residuals are, then feel free to use them and maybe remind me what they are.

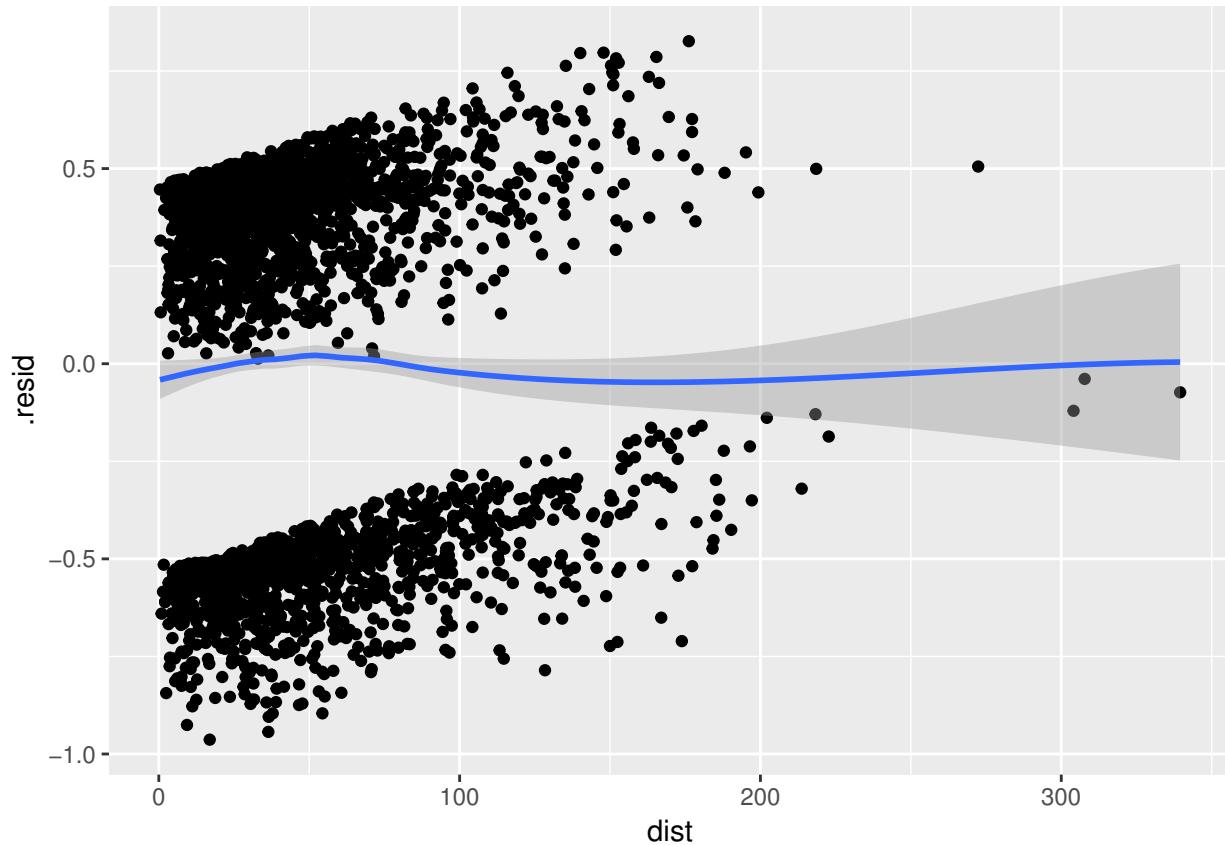
```
switch.model.df = wells
switch.model.df$.fitted = fitted.values(switch.model)
switch.model.df$.resid = residuals(switch.model, type = "response")
ggplot(switch.model.df, aes(x = .fitted, y = .resid)) + geom_point() + geom_smooth(method = "loess",
    method.args = list(degree = 1))
```



The points on the plot will always fall exactly on two lines, so look at the smooth instead. We see there's a little bit of waviness. This is fairly typical for logistic regression: there's no reason why the relationship between probability and the predictors should have that exact functional form. But the improvement in fit you get from fitting a nonparametric model is fairly small.

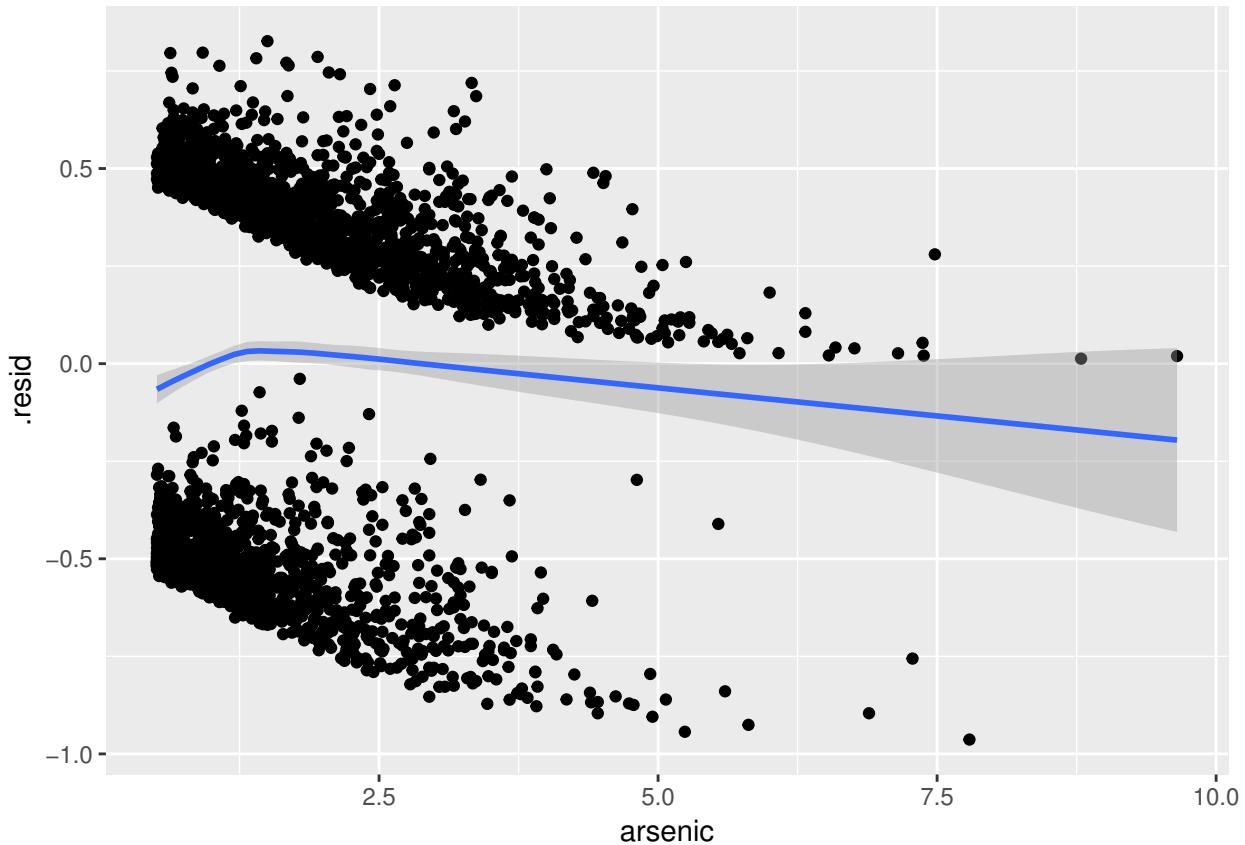
Now plot the residuals against distance.

```
ggplot(switch.model.df, aes(x = dist, y = .resid)) + geom_point() + geom_smooth(method = "loess",
    method.args = list(degree = 1))
```



There's a wiggle, but this is basically fine – the zero line is enclosed in the confidence band. Now do the same plot with arsenic on the  $x$ -axis:

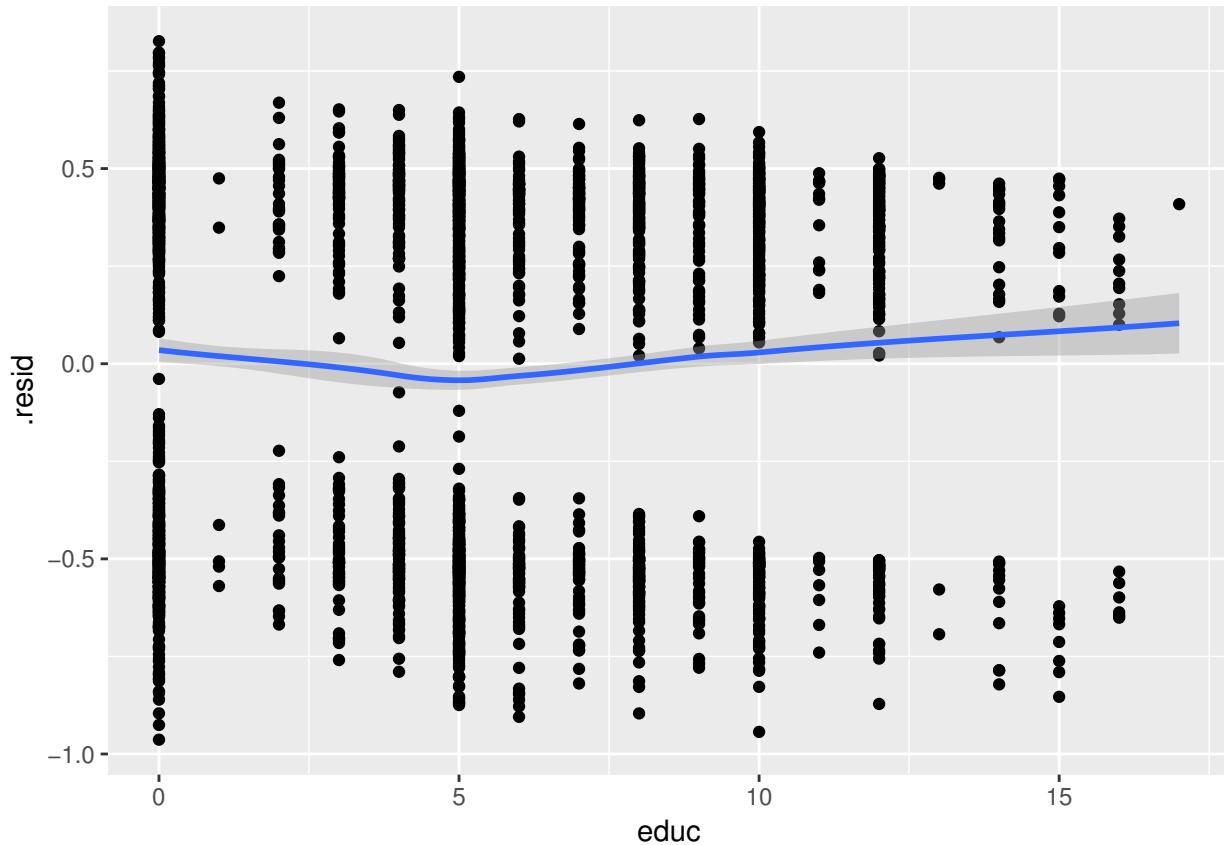
```
ggplot(switch.model.df, aes(x = arsenic, y = .resid)) + geom_point() + geom_smooth(method = "loess",
method.args = list(degree = 1))
```



This is much worse. The curve is too low, then rises too high, then declines again. In fact, the extreme left hand side is the biggest problem, since the lack of fit is real and the data is dense there. The negative average residual there means that for arsenic levels just over the threshold, the probability of switching is *overestimated* (negative average residuals occur when there are more zeroes in the response than the model expects.) When the residual curve becomes positive for arsenic between 1 and 2, that means the model *underestimates* the probability of switch for those arsenic values.

Education isn't that interesting since the effect is much smaller, but let's do it for completeness.

```
ggplot(switch.model.df, aes(x = educ, y = .resid)) + geom_point() + geom_smooth(method = "loess",
method.args = list(degree = 1))
```



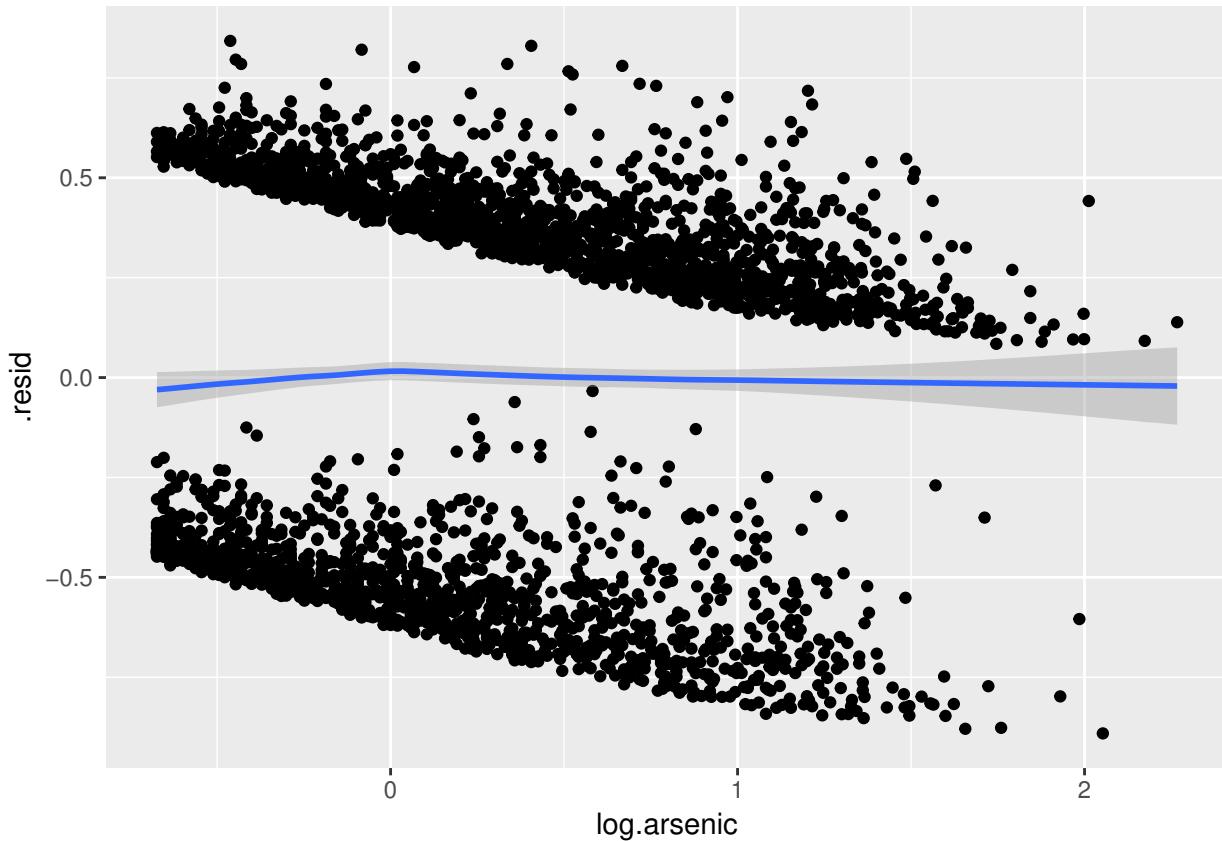
Again, the fit is imperfect.

Remember when we said we might need to transform? The last couple of graphs suggest that while a transformation of distance is unnecessary, a transformation of arsenic is. A transformation of education would probably help too, but since the variable is weird and not that important we won't bother. We'll take the log arsenic and refit the model. It's up to you what kind of log you use; it doesn't matter much in my opinion.

```
well$log.arsenic = log(well$arsenic)
log.arsenic.model = glm(switch ~ dist + log.arsenic + educ + dist:log.arsenic +
  dist:educ + log.arsenic:educ, family = "binomial", data = well)
log.arsenic.model.df = well
log.arsenic.model.df$.fitted = fitted.values(log.arsenic.model)
log.arsenic.model.df$.resid = residuals(log.arsenic.model, type = "response")
```

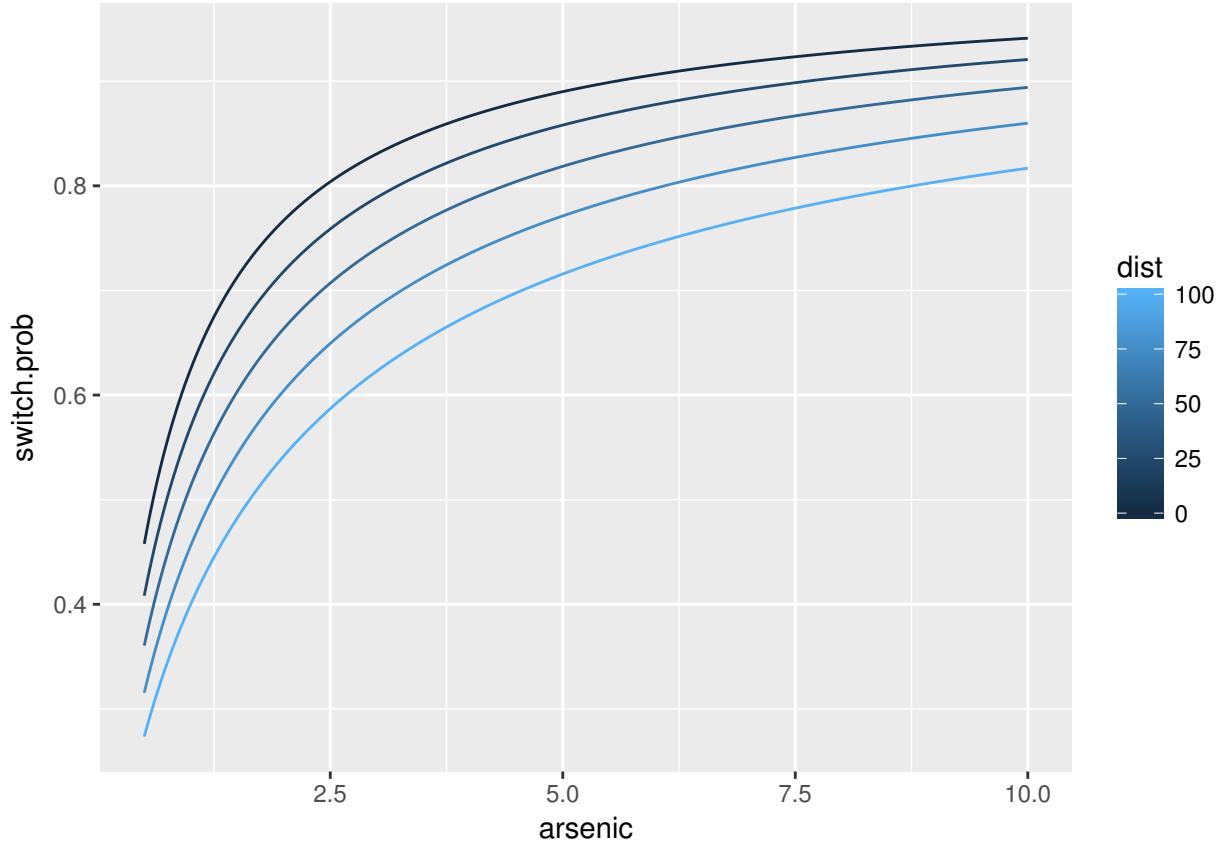
Now plot the residuals of this new model against log arsenic:

```
ggplot(log.arsenic.model.df, aes(x = log.arsenic, y = .resid)) + geom_point() +
  geom_smooth(method = "loess", method.args = list(degree = 1))
```



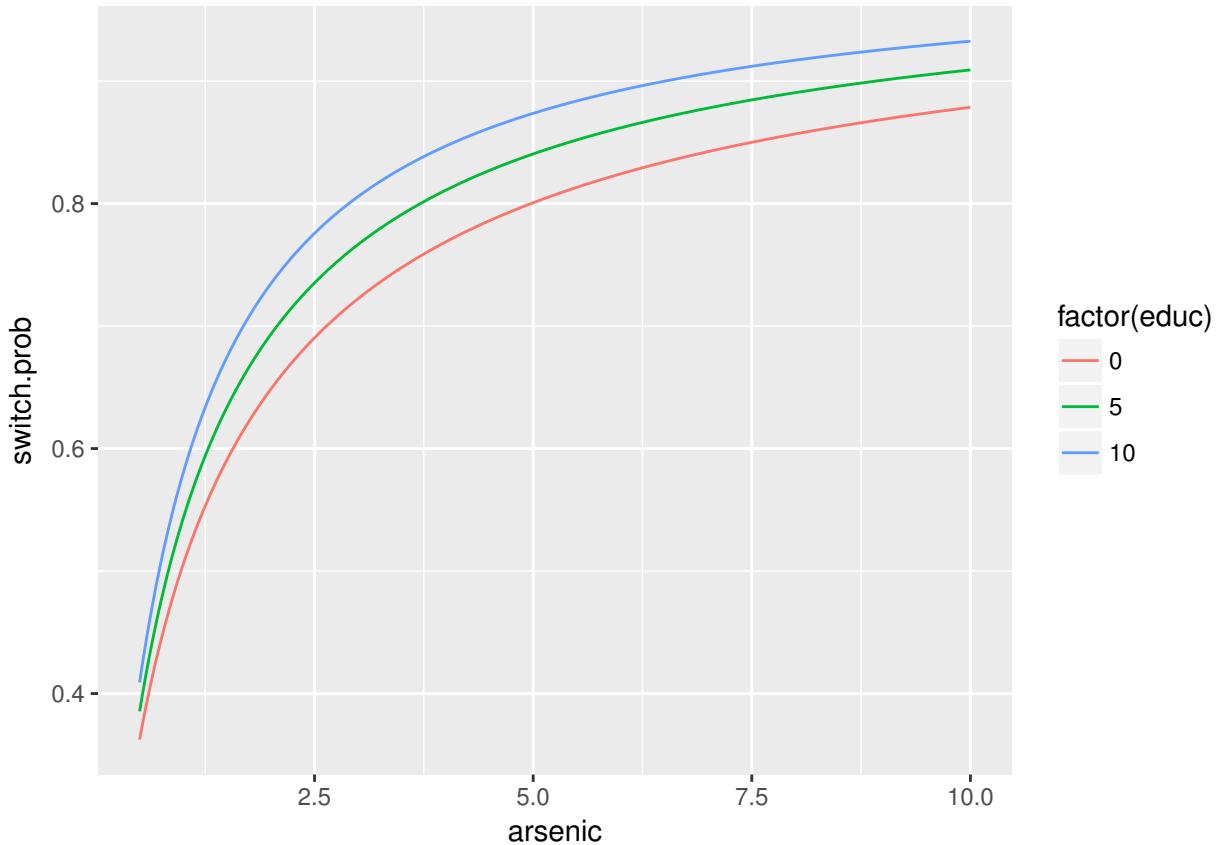
It's a lot better. I'm fairly happy with the model at this point. Let's take a look at the fit. Fix education at its median, and draw arsenic curves for several distances:

```
switch.grid = expand.grid(arsenic = seq(0.5, 10, 0.01), dist = seq(0, 100, 25),
  educ = 5)
switch.grid$log.arsenic = log(switch.grid$arsenic)
log.arsenic.pred = predict(log.arsenic.model, newdata = switch.grid, type = "response")
log.arsenic.grid = data.frame(switch.grid, switch.prob = as.vector(log.arsenic.pred))
ggplot(log.arsenic.grid, aes(x = arsenic, y = switch.prob, group = dist, color = dist)) +
  geom_line()
```



There are  $3!$  ways you can assign the three variables to  $x$ -axis, conditioning variable, and fixed. You probably get the idea at this point, though, so let's just do one more:

```
switch.grid2 = expand.grid(arsenic = seq(0.5, 10, 0.01), dist = median(wells$dist),
  educ = c(0, 5, 10))
switch.grid2$log.arsenic = log(switch.grid2$arsenic)
log.arsenic.pred2 = predict(log.arsenic.model, newdata = switch.grid2, type = "response")
log.arsenic.grid2 = data.frame(switch.grid2, switch.prob = as.vector(log.arsenic.pred2))
ggplot(log.arsenic.grid2, aes(x = arsenic, y = switch.prob, group = educ, color = factor(educ))) +
  geom_line()
```



### 7.2.6 Improving the model

If you really care about prediction but still want to be able to interpret the model, try a nonparametric model such as a generalized additive model (GAM) or loess. To overgeneralize, GAM is often better/easier when you have lots of predictors, while loess is often better/easier when you have complex interactions. You can learn more about these in S425/625.

If you really, really care about predictions then you can use machine learning techniques. The improvement in prediction will be quite small, however, and you'll lose almost all interpretability.

