Binary responses

S470/670

Last updated 2020-10-13

```
library(tidyverse)
cb_palette = c("#999999", "#E69F00", "#56B4E9", "#009E73",
    "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
```

Logistic regression, plus dealing with messy data

Recommended reading: Gelman & Hill pp. 79–86 (frequentist version); Gelman, Hill & Vehtari pp. 217–232 (Bayesian version.) For a more detailed and more traditional treatment, try Agresti, Categorical Data Analysis, ch. 5 & 6 (3rd edition.)

ANES

Download the ANES Time Series Cumulative Data File, containing data from 1948 to 2016 (registration required):

https://electionstudies.org/data-center/

The raw data is just an astonishing mess, so get the Stata file. The `import` function in `library(rio)` is my preference for reading exotic data formats into R, though the `haven` package also works if you want something tidyverse-friendly.

```
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   2.676   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    1986    1985    2002    2016
```

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   0.938   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   35118
```

Now suppose we make 1 represent Republican and 0 represent 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   35118
```

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

```
Republican = vote
```

Let's also 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  1.0453  1.0000  6.8130
```

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

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

Finally, I only want data for Presidential election years (years divisible by 4.) I'll use the `filter()` function in `dplyr`:

```
ANES.df = filter(ANES.df, year %in% seq(1948, 2016, 4))
summary(ANES.df)
```

```
##       year          income        Republican    survey.weights
##  Min.   :1948   Min.   :0.000   Min.   :0.000   Min.   :0.0212
##  1st Qu.:1972   1st Qu.:2.000   1st Qu.:0.000   1st Qu.:0.9038
##  Median :1992   Median :3.000   Median :0.000   Median :1.0000
##  Mean   :1989   Mean   :2.677   Mean   :0.481   Mean   :1.0356
```

```
##  3rd Qu.:2012   3rd Qu.:4.000   3rd Qu.:1.000   3rd Qu.:1.0000
##  Max.   :2016   Max.   :5.000   Max.   :1.000   Max.   :6.8130
##                                 NA's   :14394
```

Let's save this so we can use it later:

```
write.table(ANES.df, file = "ANES.txt", row.names = FALSE)
```

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 = filter(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. With `tidyverse`, it's easy to do the count directly:

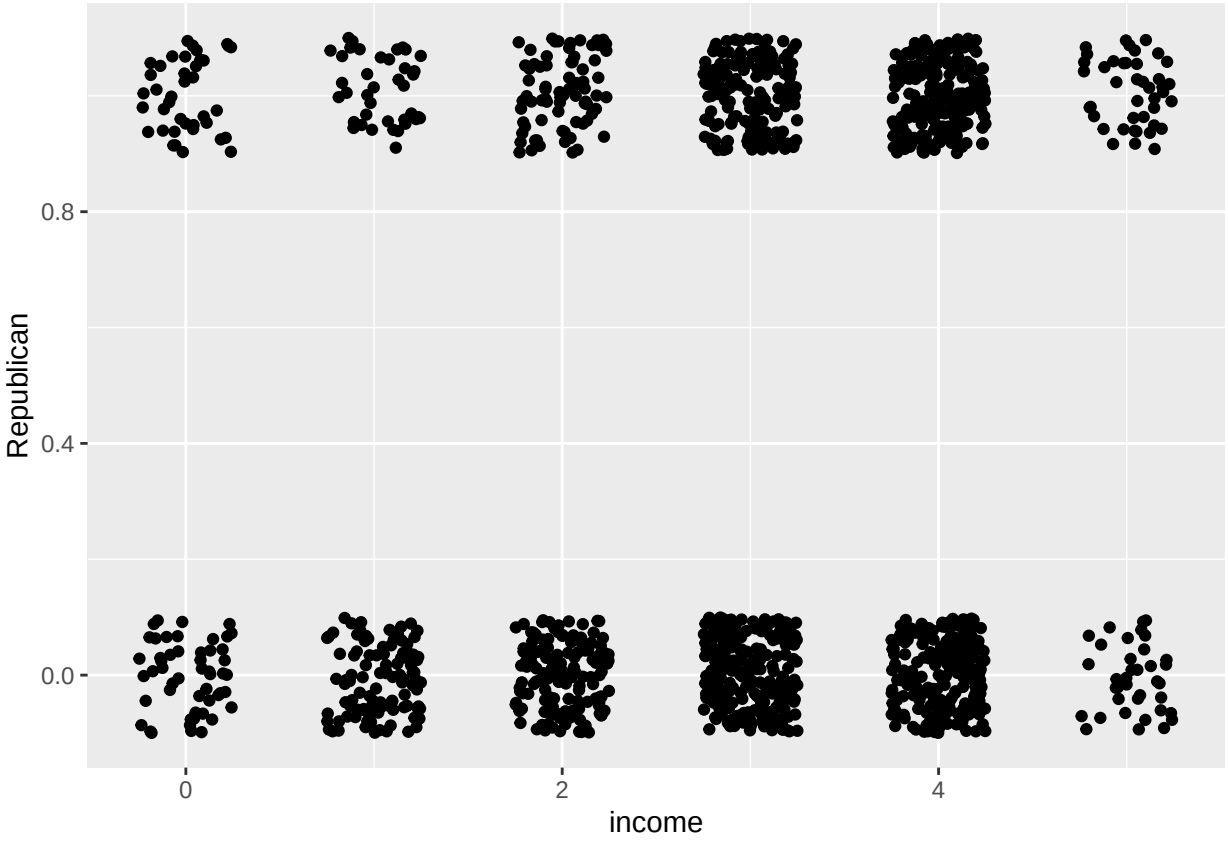
```
count(ANES1992, Republican)
```

```
## # A tibble: 3 x 2
##   Republican     n
##        <dbl> <int>
## 1          0   793
## 2          1   564
## 3         NA  1128
```

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.

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



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. How do we model this? Three options (not the only three) include:

1. Linear regression with income as a numerical predictor.
2. Logistic regression with income as a numerical predictor.
3. Regression with income as a categorical (factor) predictor. (In this case, linear and logistic give identical predictions.)

Method 1 is the easiest to interpret: we get a slope that directly tells us the change in model probability of voting Republican as income goes up one category. However, linear regression for binary responses has both technical and social limitations. The technical limitation is that it only works well when probability are far from 0 and 1. Otherwise, if x is unbounded, you can end up with negative probabilities or probabilities greater than 1. The social limitation is that about two-thirds of statistics professors will never speak to you again if they see you doing linear regression on binary data. While this may be considered a positive by some, it is not really feasible for the author, so I do not pursue it here.

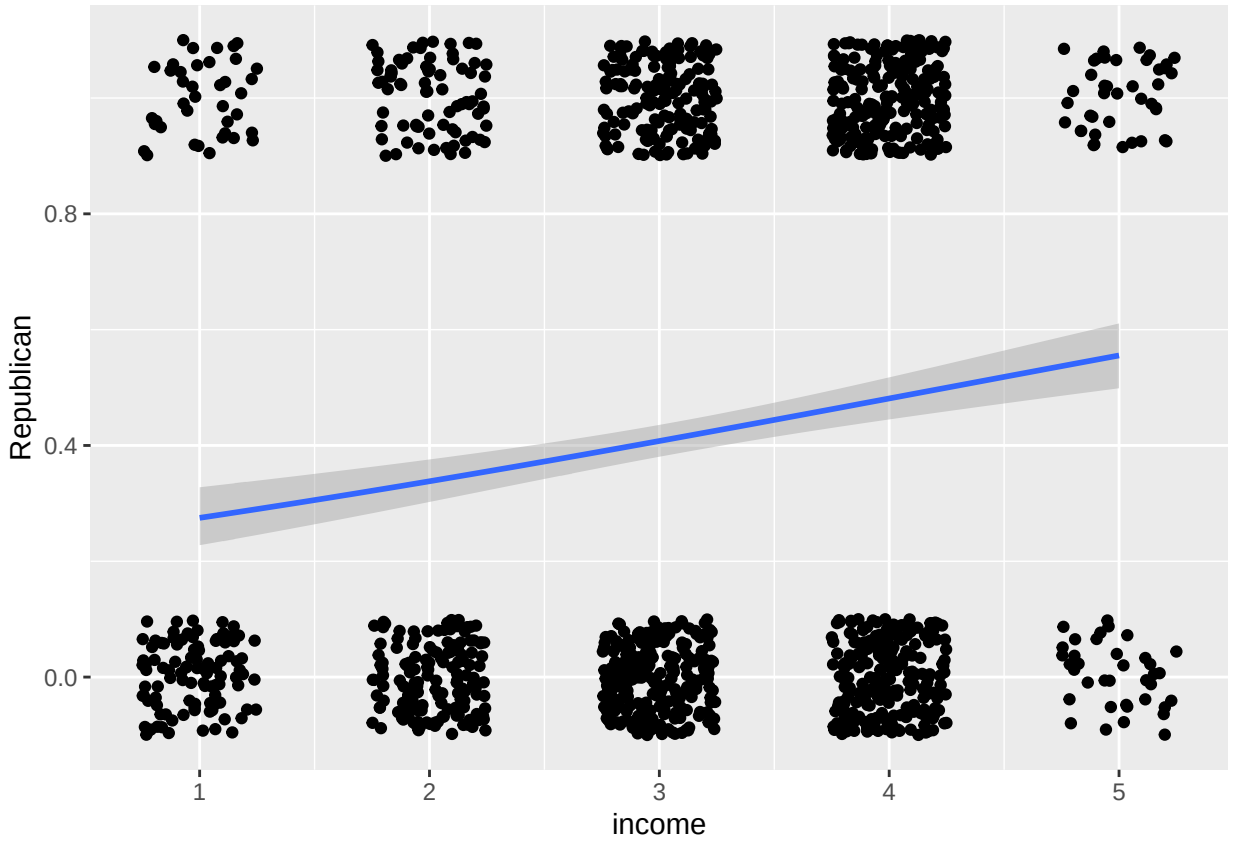
Method 3 isn't really a model at all: it just returns the proportion within each category who voted for Bush,

the same as our `aggregate()` call gave us above. There's something to be said for not fitting restrictive models when you don't have to. However, if our goal is to fit more complex models or make comparisons between different sets of data, as it eventually will be, then some degree of simplification may be useful to understand the patterns in the data. Or we might fit a simplifying model first, then go back and look at the data in more detail and see if there are any important features our model missed. That will be our basic approach here.

Method 2, logistic regression, should work well. It does require treating a predictor that isn't *inherently* a numeric variable as numeric, and requires a parametric form (effects are linear on a logit scale.) However, most of the time, doing this is reasonable as long as the pattern of the probability with x is monotonic and as long as predictive accuracy is not the sole goal. (If the pattern was non-monotonic, a nonparametric method like a generalized additive model (GAM) would be a better modeling choice. If predictive accuracy is the sole goal then it increasingly seems like you have to learn neural networks, which I haven't.)

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 = filter(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"))
```



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

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

To find P(Bush), we invert the logit:

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

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
```

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, survey 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.) Note that once you start using weighted methods, you're relying on numerical methods that occasionally can go catastrophically wrong, so you should always check your model coefficients to see if you've converged to something implausible. (For `glm()` in particular, the routine seems to work best when the average size of the weights is not too large.)

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

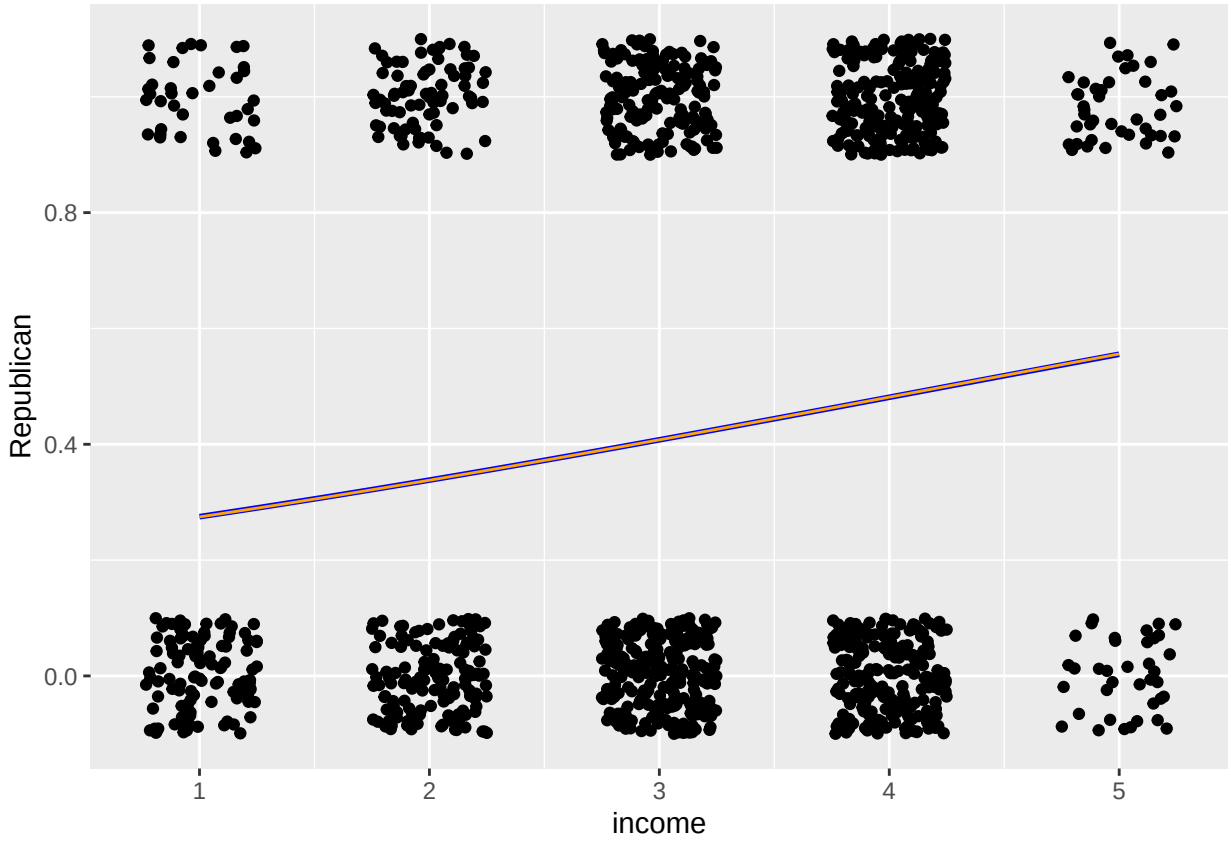
```
##
## Call:
## glm(formula = Republican ~ income, family = quasibinomial, data = ANES1992,
##     weights = survey.weights)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.3005  -1.0331  -0.8763   1.2091   1.7976
##
## Coefficients:
##             Estimate Std. Error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



The weighted and unweighted fits are nearly indistinguishable. This is quite often the case when creating regression models with high-quality data. (On the other hand, if you're just doing simple summaries of the data, or if the data is from a low-quality source such as a random digit dial poll, weights often 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.

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 = filter(data, year == my.year)
  newdata = filter(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)
```

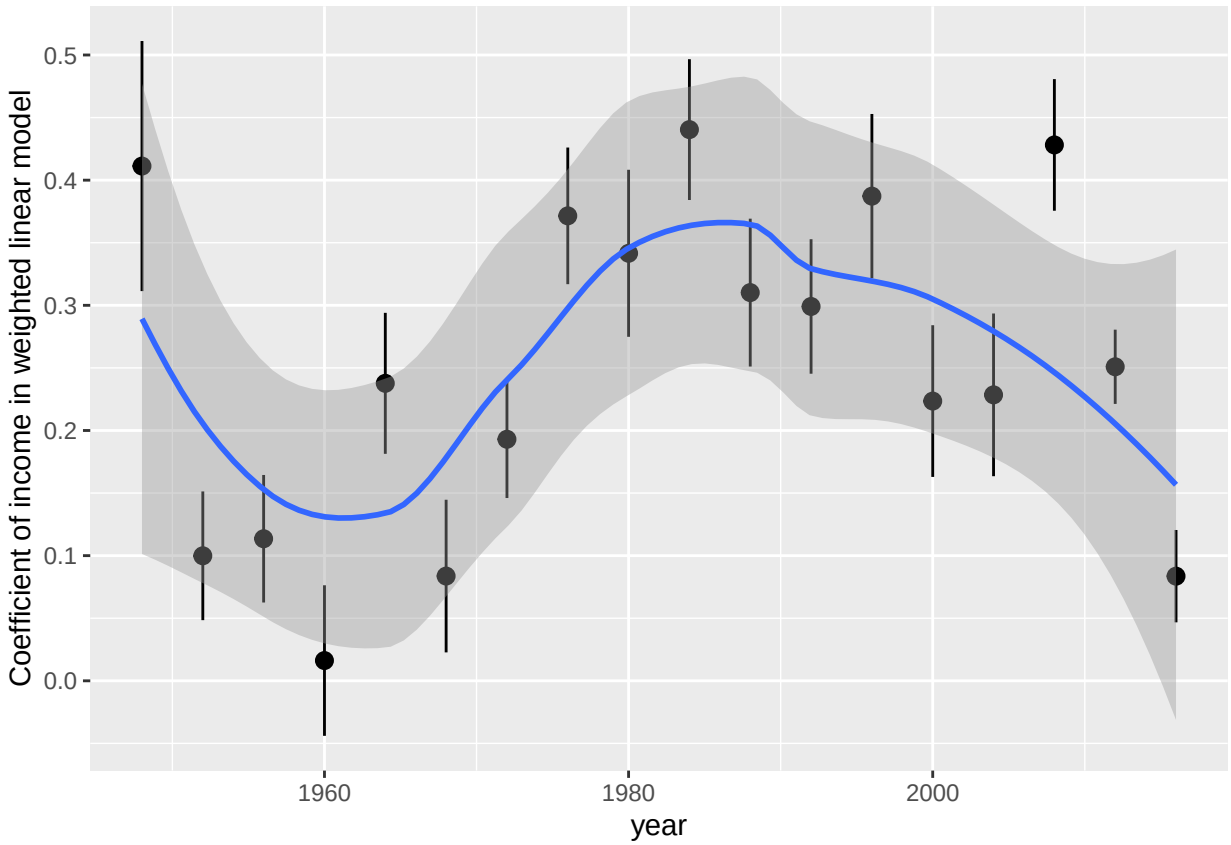
```
##                   Estimate   Std. Error
## 1.992000e+03 2.990845e-01 5.373261e-02
```

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, 2016, 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.

```
gg = ggplot(income.by.year, aes(x = year, y = income.coef,
     ymin = income.coef - income.se, ymax = income.coef + income.se))
gg + geom_pointrange() +
  geom_smooth(method = "loess",
    method.args = list(family = "symmetric")) +
  ylab("Coefficient of income in weighted linear model")
```



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.)

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 (if inefficient) using a `for` loop.

To find weighted means, I used use `weighted.mean()` in conjunction with `summarise()` in `dplyr`. Let's first work out how to do it for the 1992 data.

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

```
## # A tibble: 5 x 2
##   income `weighted.mean(Republican, w = survey.weights, na.rm = TRUE)`
##    <dbl>                                                         <dbl>
## 1      1                                                         0.267
## 2      2                                                         0.348
## 3      3                                                         0.401
## 4      4                                                         0.489
## 5      5                                                         0.533
```

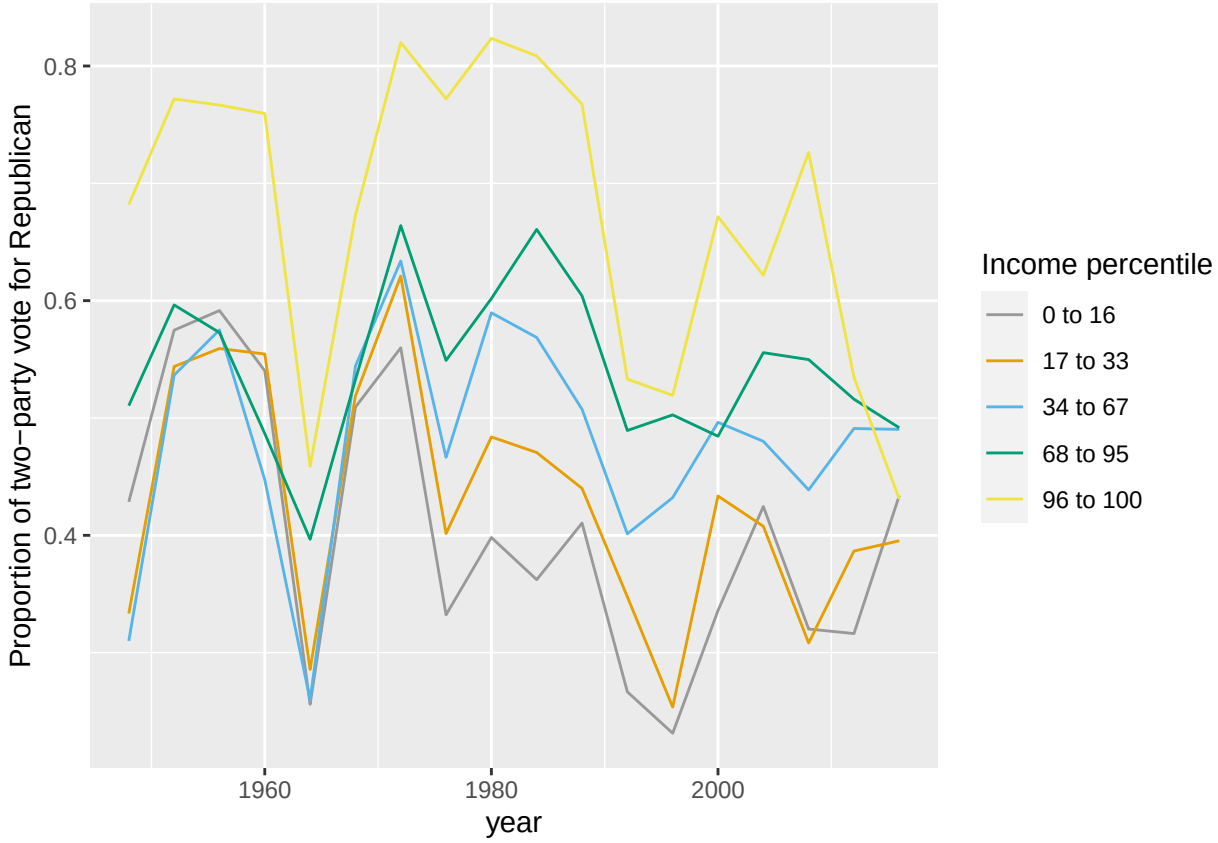
Now we do the same thing to the bigger data set, this time grouping by both income and year, then removing

the "0" income category:

```
income.prop = group_by(ANES.df, income, year) %>%
  summarise(weighted.mean(Republican,
    w = survey.weights, na.rm = TRUE))
names(income.prop) = c("income", "year", "prop.Republican")
income.prop = income.prop[income.prop$income > 0,]
```

Plot the results:

```
gg = ggplot(income.prop, aes(x = year, y = prop.Republican,
    group = income, color = factor(income))) + geom_line() +
  scale_color_manual(values = cb_palette, labels = c("0 to 16",
    "17 to 33", "34 to 67", "68 to 95", "96 to 100"))
gg + ylab("Proportion of two-party vote for Republican") +
  labs(color = "Income percentile")
```

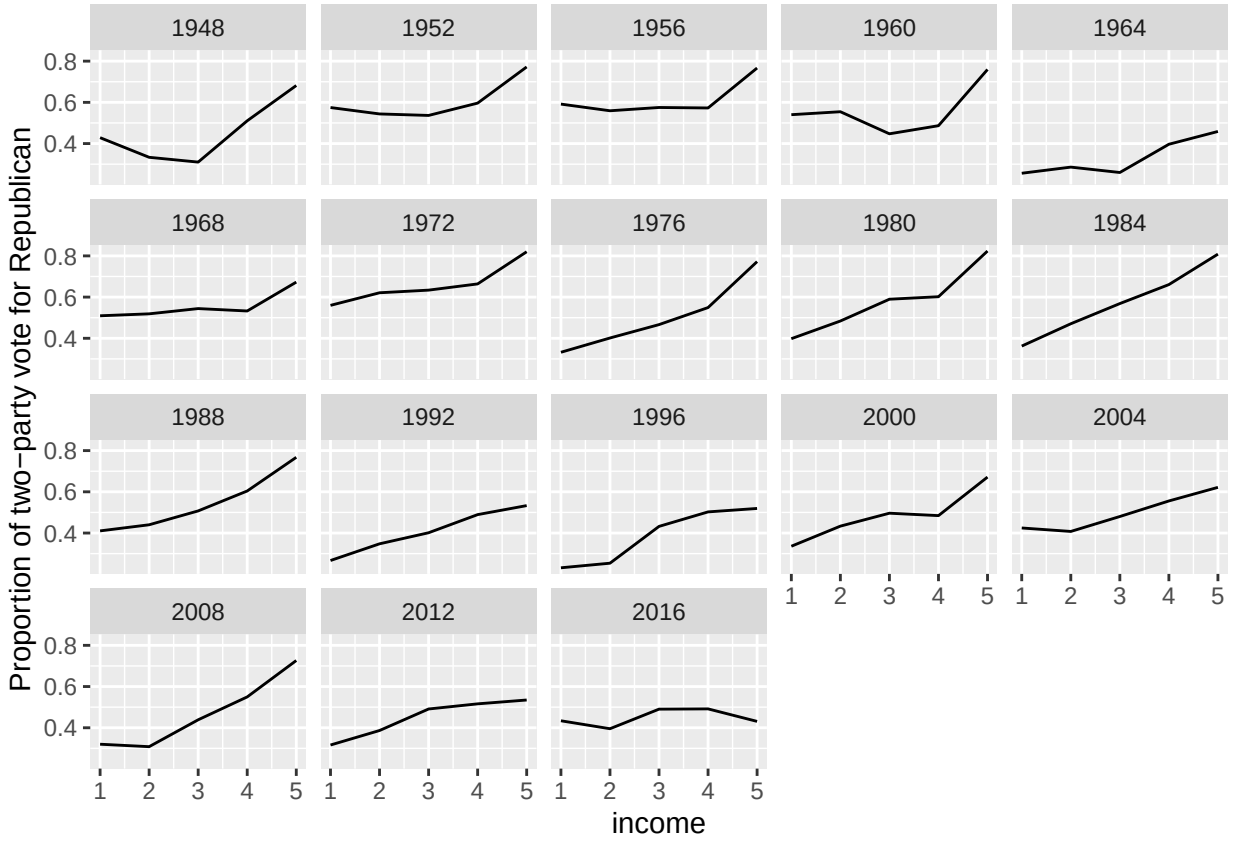


We now have a bit more detail on the trends and the aberrant results.

- 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. Because there are so many different years, we'll facet them rather than color them.

```
ggplot(income.prop, aes(x = income, y = prop.Republican)) +
  geom_line() + facet_wrap(~year, ncol = 5) +
  ylab("Proportion of two-party vote for Republican")
```



This perhaps 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. (It would be interesting to check if this continued to hold in 2016.)

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 a 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 results that the big picture doesn't explain.

Logistic regression with multiple predictors

Supplementary reading: Gelman & Hill pp. 86–104 (frequentist version); Gelman, Hill & Vehtari pp. 232–237 (Bayesian version.)

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:

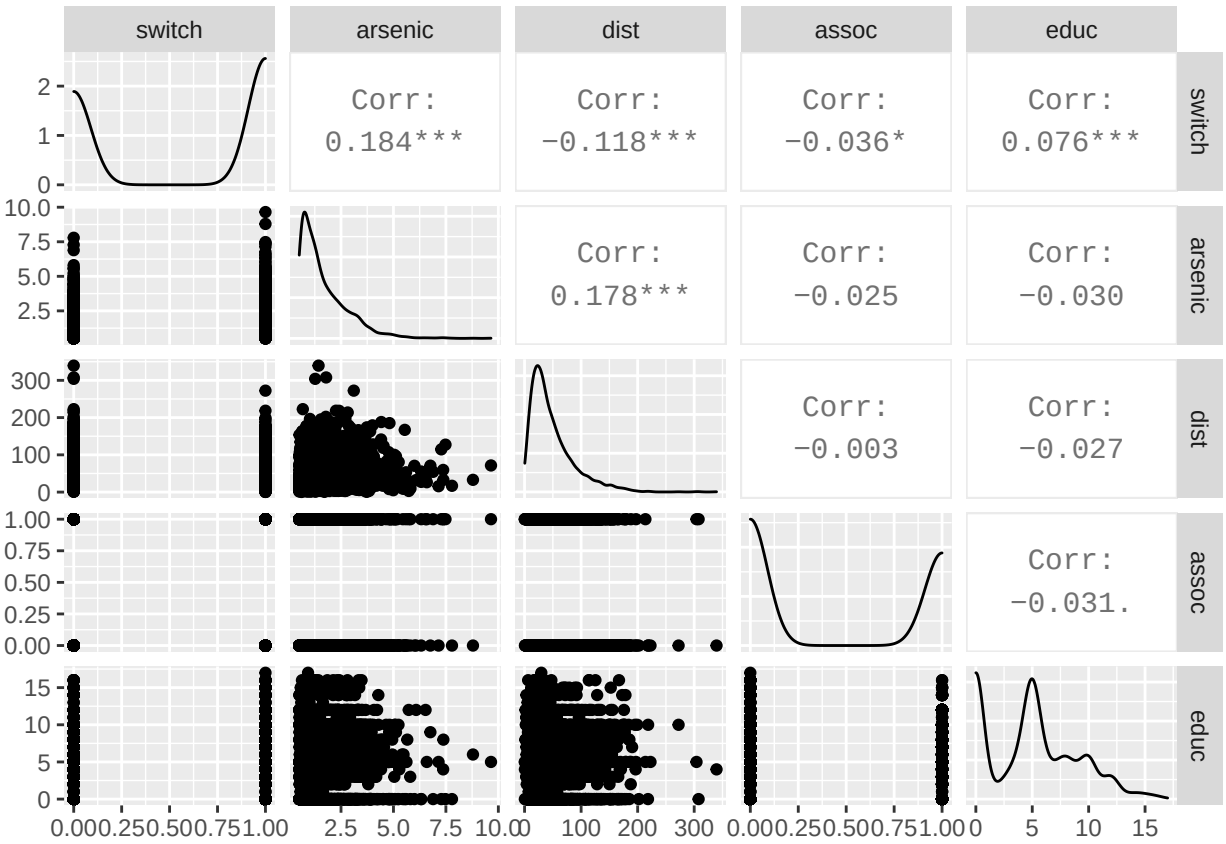
- `switch`: the response: whether or not the household switched after being encouraged to do so (after high arsenic was measured.)
- `arsenic`: the arsenic concentration in hundreds of micrograms per liter.
- `dist`: the distance to the nearest safe well in meters.
- `assoc`: whether anyone in the household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



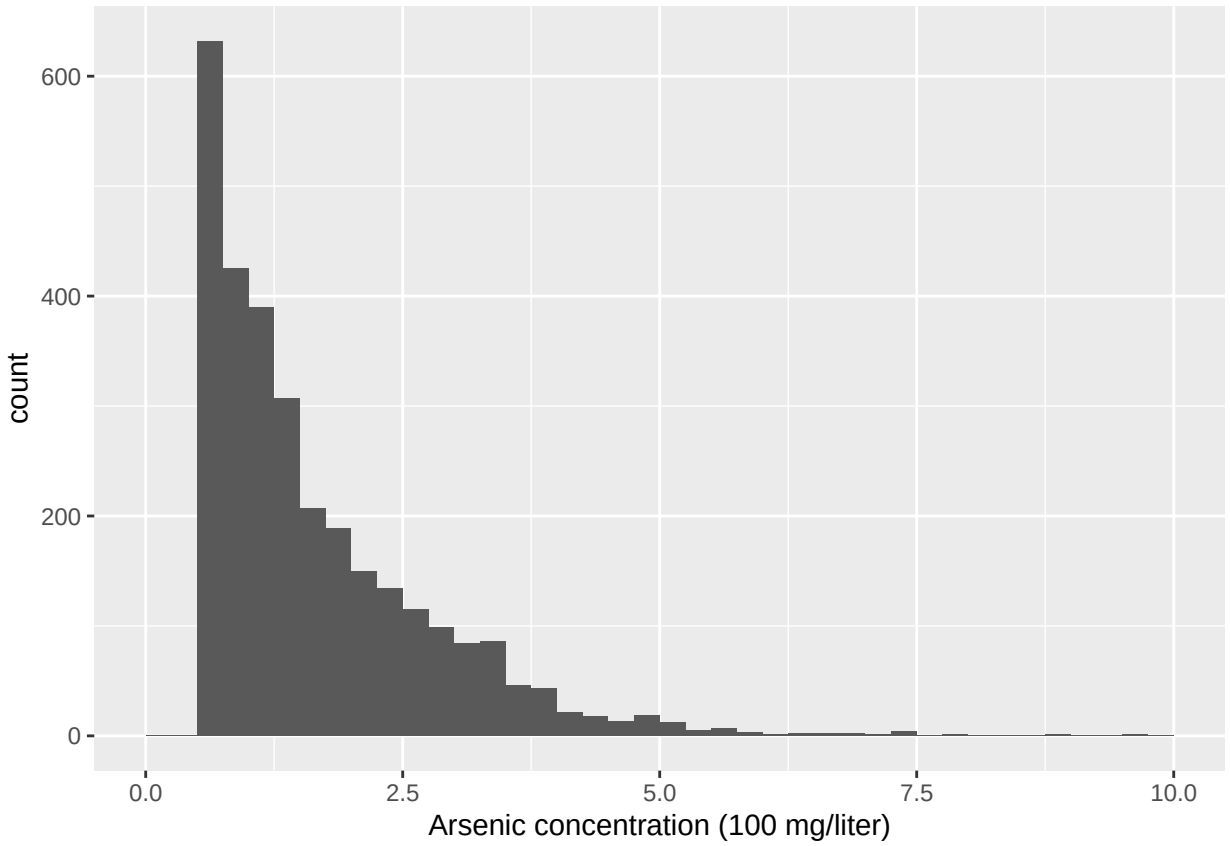
As we might expect, distance and education level are positively associated with switching, while arsenic level is negatively associated. Surprisingly, the correlation between belong to an association and switching is negative, though the relationship is very weak.

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

```
gg = ggplot(wells, aes(x = dist)) +
  geom_histogram(breaks = seq(0, 340, 10)) +
  xlab("Distance to nearest safe well (meters)")
```

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)) +
  xlab("Arsenic concentration (100 mg/liter)")
```

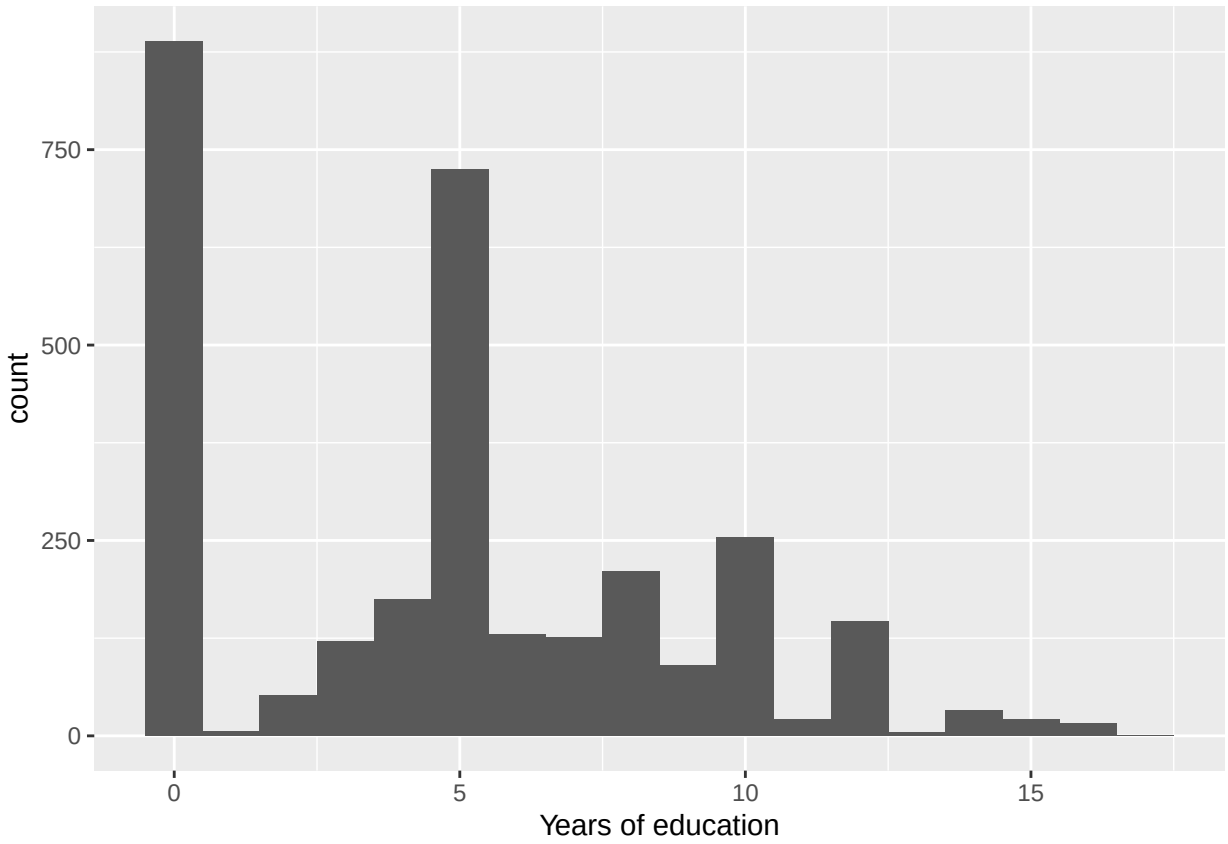


There are no observations with arsenic below 0.5, because these are considered "safe" and the households don't get advised to switch wells. Otherwise, the distribution is against strongly right-skewed, suggesting a transformation.

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) +
  xlab("Years of education")
```



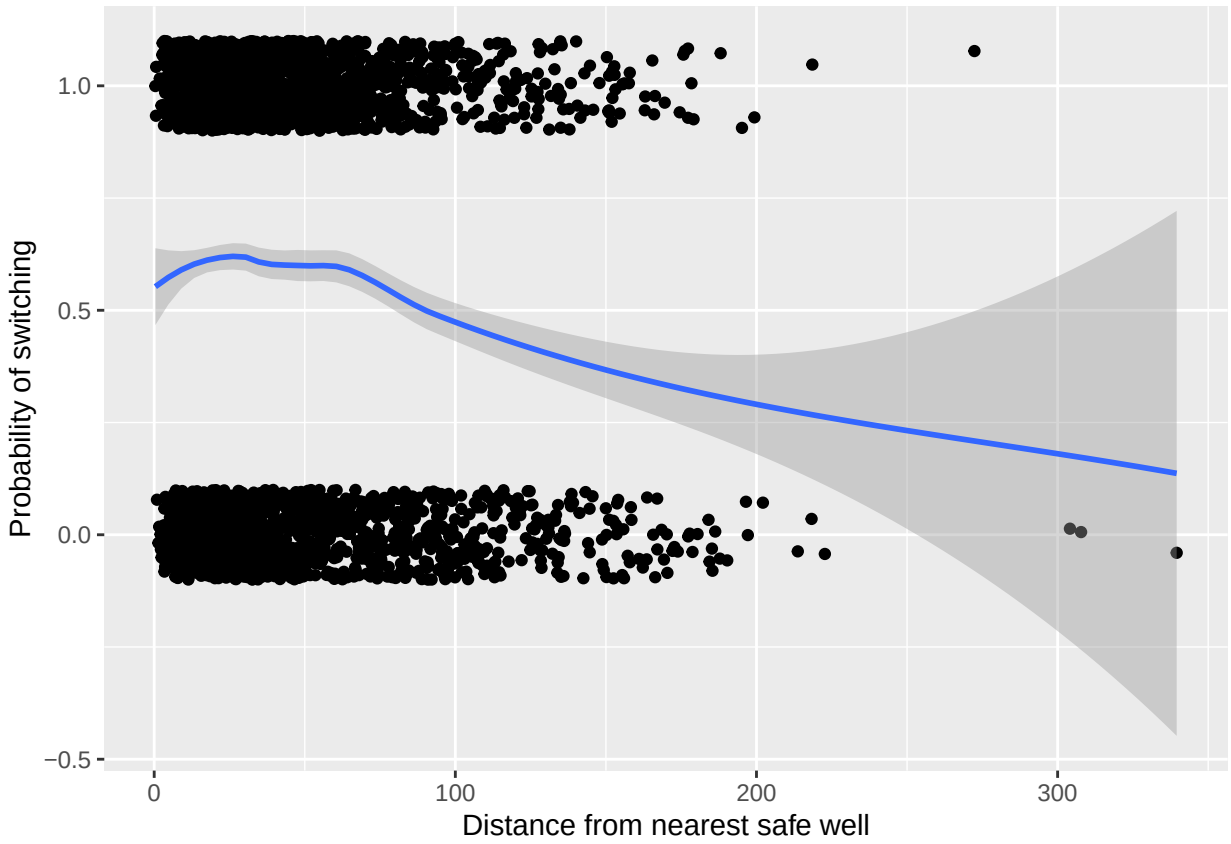
The distribution is weird. 0 (no education) and 5 (primary school only) are the magic numbers.

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.

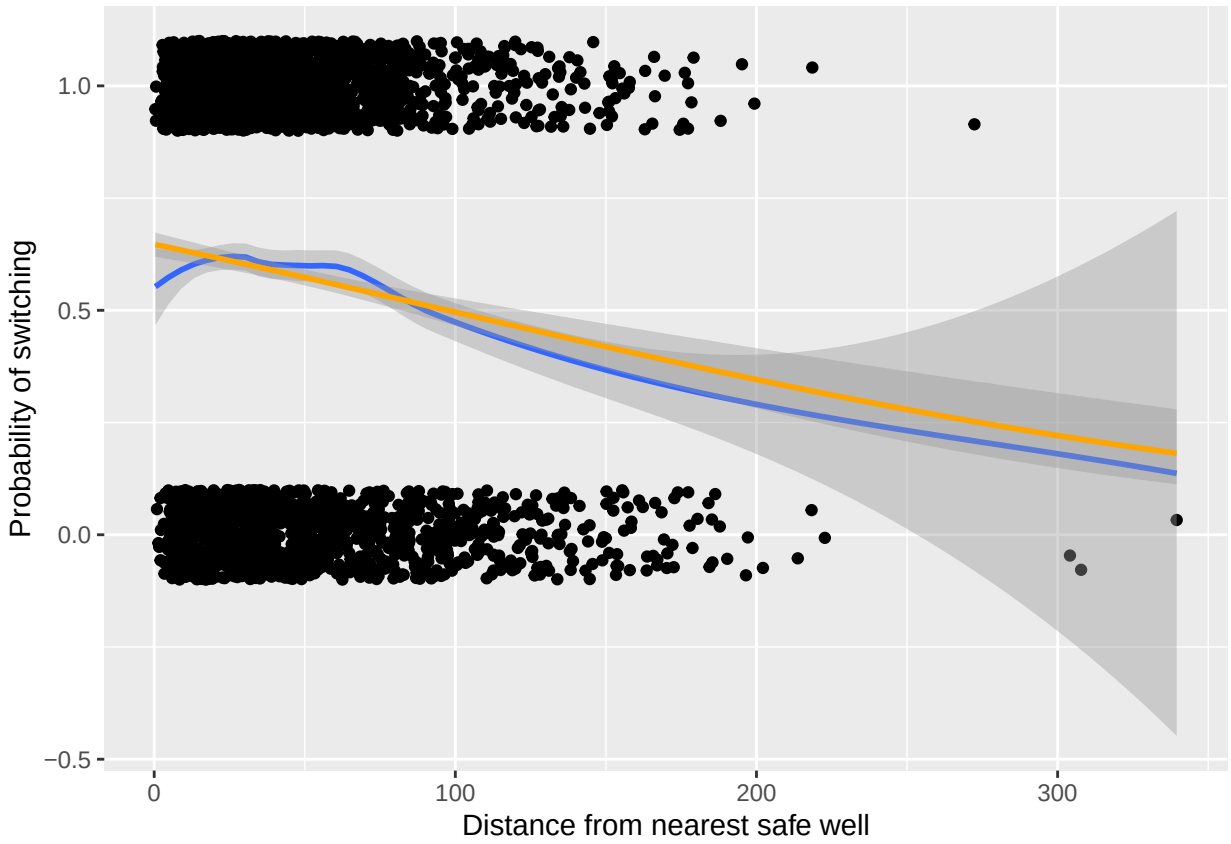
We'll start by predicting switching, using distance as the only response. There's no reason why we can't try loess:

```
ggplot(wells, aes(x = dist, y = switch)) +
  geom_jitter(width = 0, height = 0.1) +
  geom_smooth(method = "loess") +
  xlab("Distance from nearest safe well") +
  ylab("Probability of switching")
```



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.

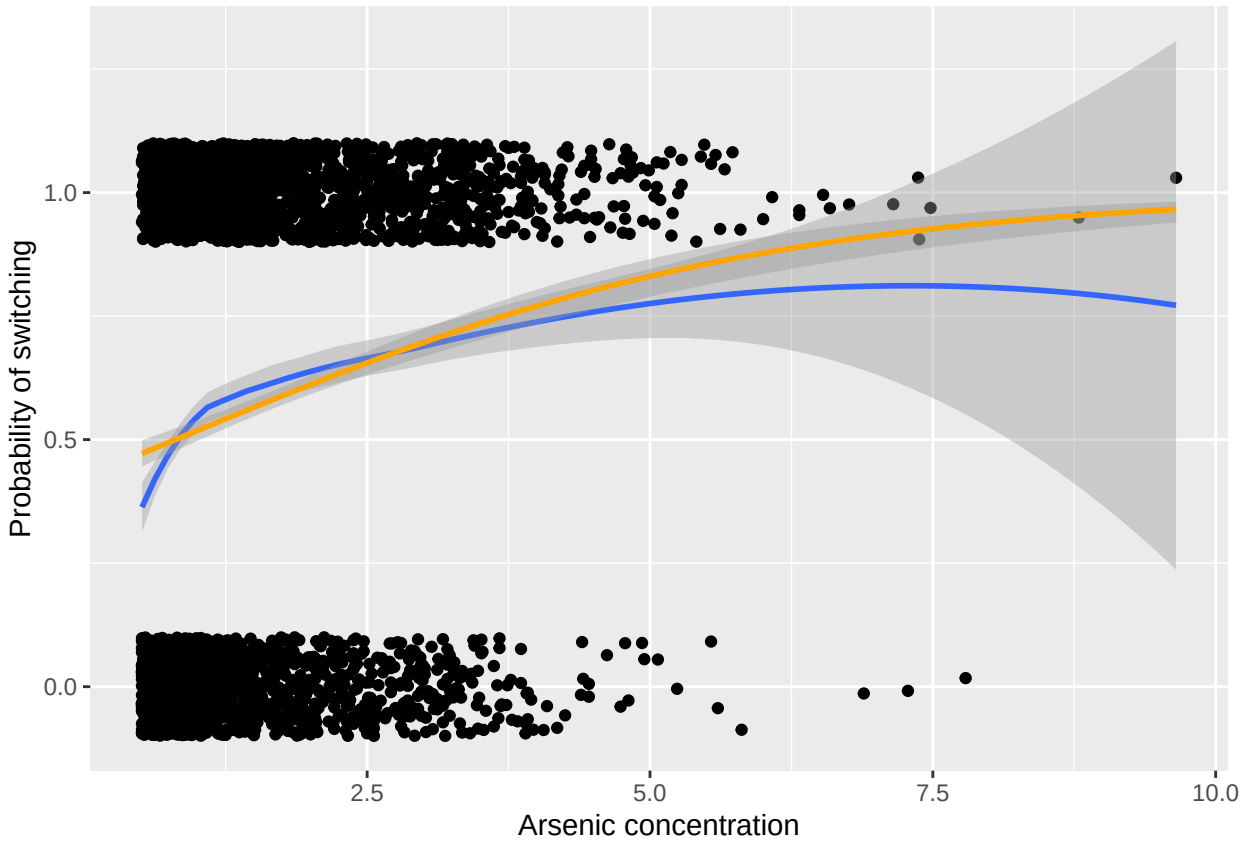
```
gg = ggplot(wells, aes(x = dist, y = switch)) +
  geom_jitter(width = 0, height = 0.1) +
  geom_smooth(method = "loess")
gg + geom_smooth(method = "glm",
    method.args = list(family = "binomial"), color = "orange") +
  xlab("Distance from nearest safe well") +
  ylab("Probability of switching")
```



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.

```
gg = ggplot(wells, aes(x = arsenic, y = switch)) +
  geom_jitter(width = 0, height = 0.1) +
  geom_smooth(method = "loess")
gg + geom_smooth(method = "glm",
    method.args = list(family = "binomial"), color = "orange") +
  xlab("Arsenic concentration") + ylab("Probability of switching")
```

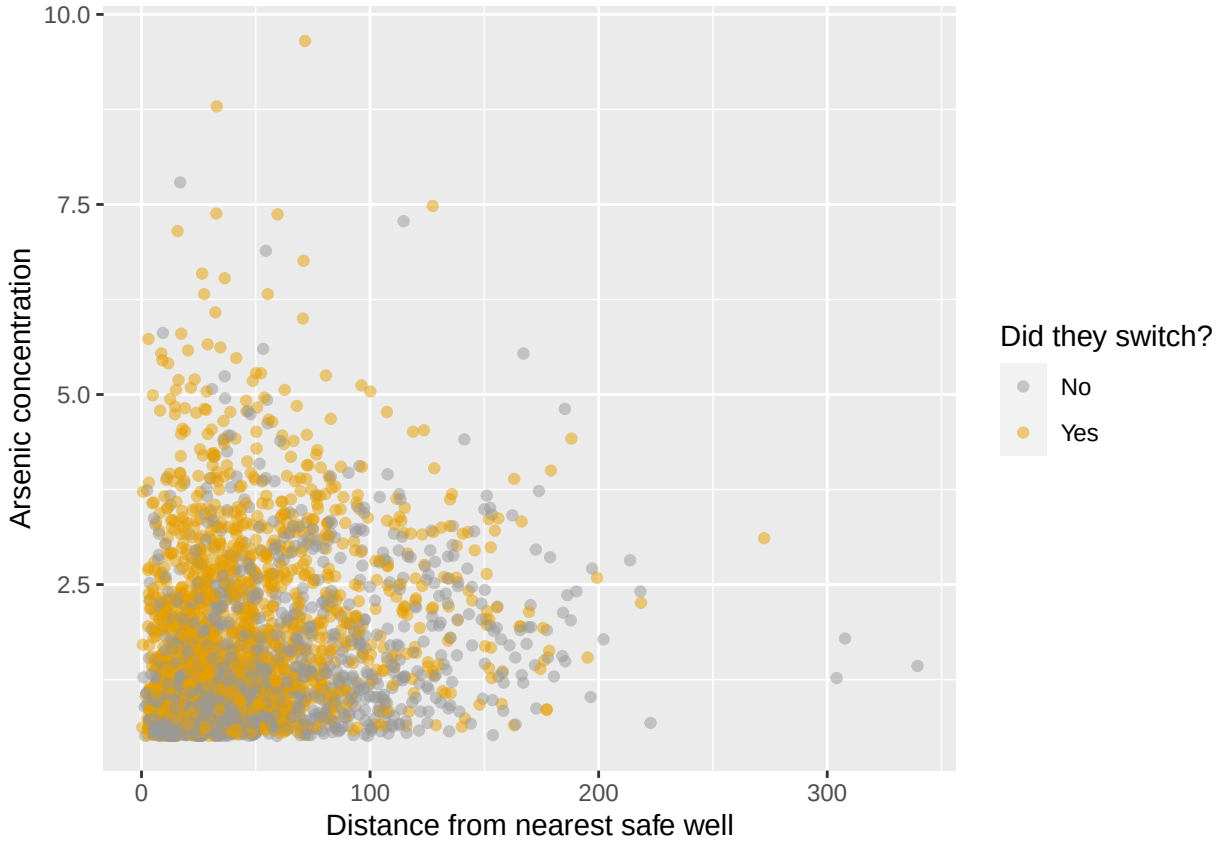


The fits diverge at arsenic levels above about 5. The seems to be because of a couple of extremely high arsenic observations which led to switching. Note however that not much of the data has arsenic above 5, so it remains to be seen how much this matters.

Two predictors

We now want to know how the chance of switching depends on distance and arsenic simulataneously. 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) +
  xlab("Distance from nearest safe well") +
  ylab("Arsenic concentration") +
  labs(color = "Did they switch?") +
  scale_color_manual(values = cb_palette, labels = c("No", "Yes"))
```



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
```

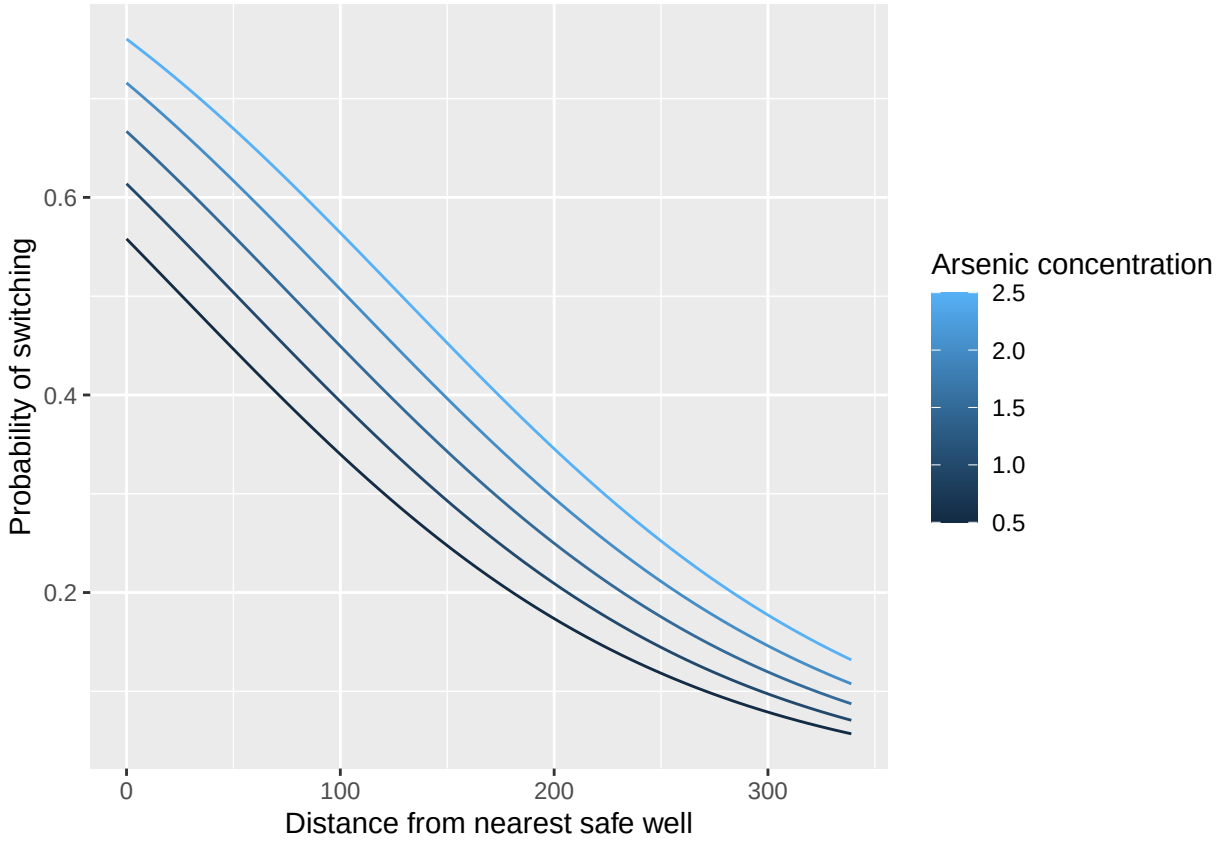
```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4118.1  on 3019  degrees of freedom
## Residual deviance: 3930.7  on 3017  degrees of freedom
## AIC: 3936.7
##
## Number of Fisher Scoring iterations: 4
```

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

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

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.

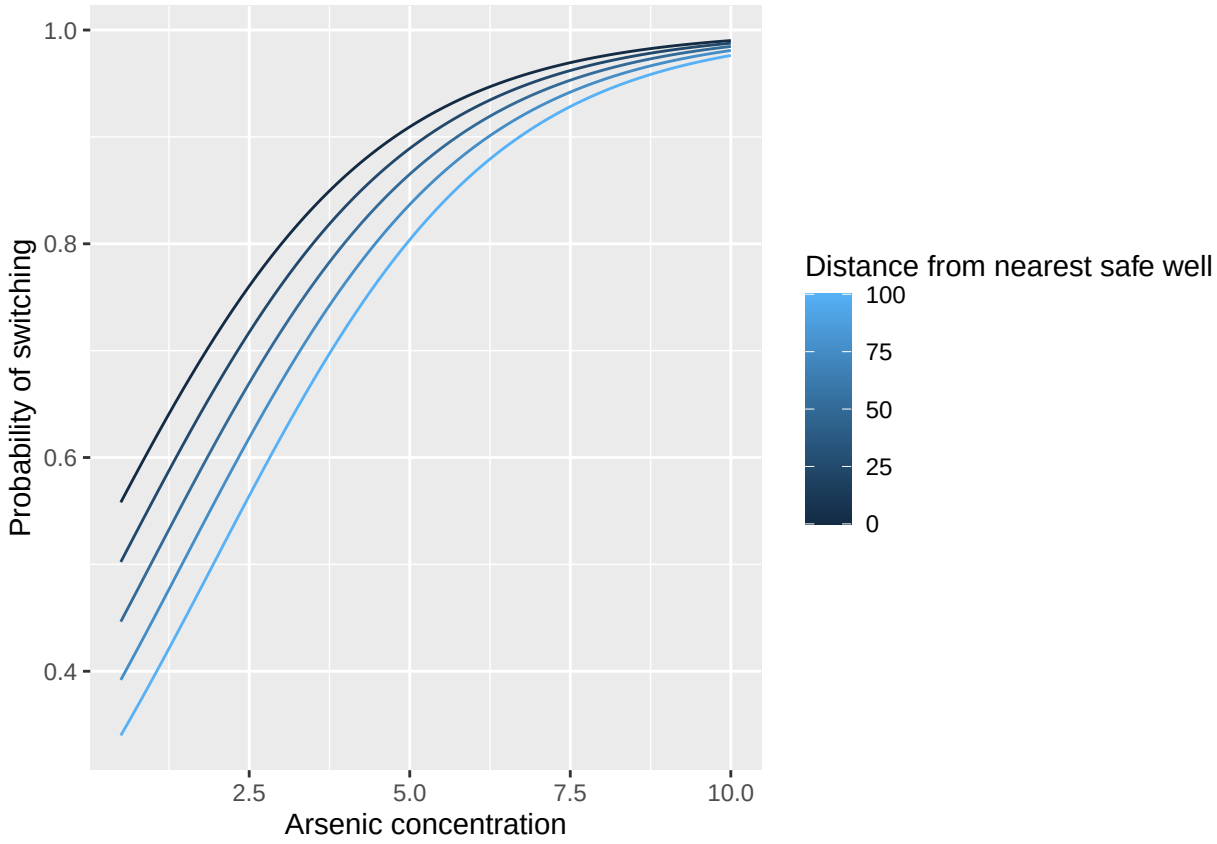
```
dist.df = expand.grid(dist = 0:339, arsenic = seq(0.5, 2.5, 0.5))
dist.pred = predict(switch.logit,
    type = "response", newdata = dist.df)
dist.pred.df = data.frame(dist.df,
    switch.prob = as.vector(dist.pred))
ggplot(dist.pred.df, aes(x = dist, y = switch.prob,
    group = arsenic, color = arsenic)) + geom_line() +
  xlab("Distance from nearest safe well") +
  ylab("Probability of switching") +
  labs(color = "Arsenic concentration")
```



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 different 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() +
  xlab("Arsenic concentration") + ylab("Probability of switching") +
  labs(color = "Distance from nearest safe well")
```



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.

Adding an interaction

We now add an interaction term between distance and arsenic.

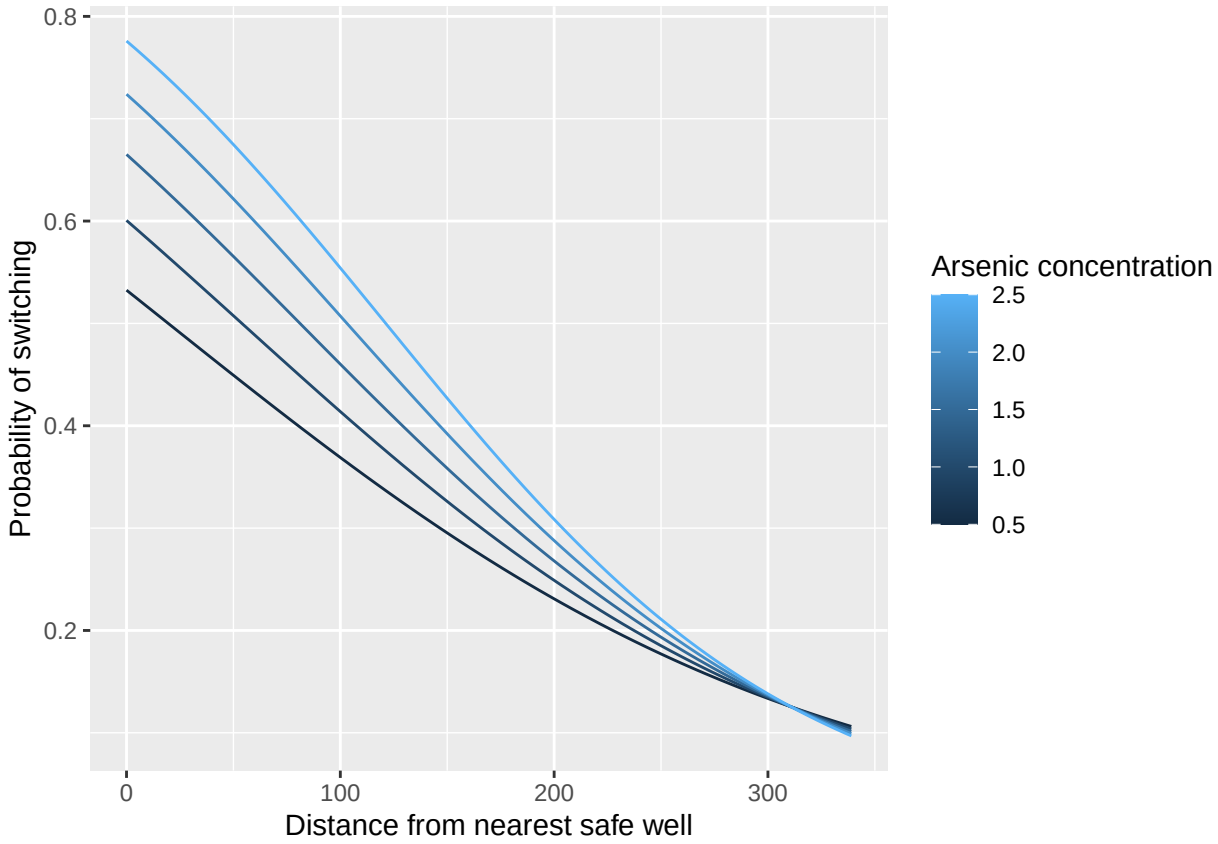
```
wells = read.table("wells.dat")
switch.int =
  glm(switch ~ dist * arsenic, family = "binomial", data = wells)
```

```
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
```

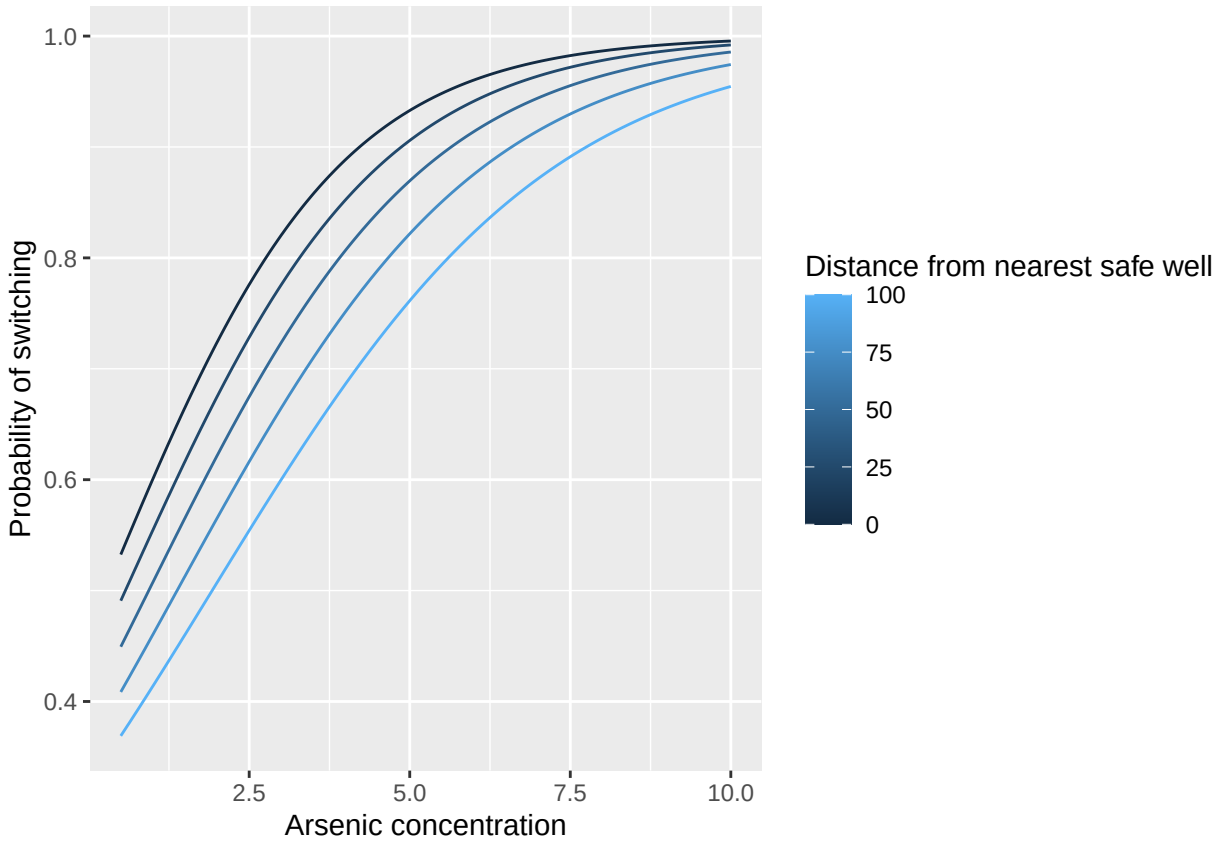
The numbers are a bit hard to interpret. For example, the sign of the interaction term is negative, but it's hard to know exactly what this means (especially since the signs for distance and arsenic go in different directions.) Let's just plot curves.

```
dist.df = expand.grid(dist = 0:339, arsenic = seq(0.5, 2.5, 0.5))
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() +
  xlab("Distance from nearest safe well") +
  ylab("Probability of switching") +
  labs(color = "Arsenic concentration")
```



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.df =
  expand.grid(arsenic = seq(0.5, 10, 0.01), dist = seq(0, 100, 25))
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() +
  xlab("Arsenic concentration") + ylab("Probability of switching") +
  labs(color = "Distance from nearest safe well")
```



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.)

Lots of predictors

Distance and arsenic are both good predictors and there's no reason to think there shouldn't be an interaction, so keep the 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))
```

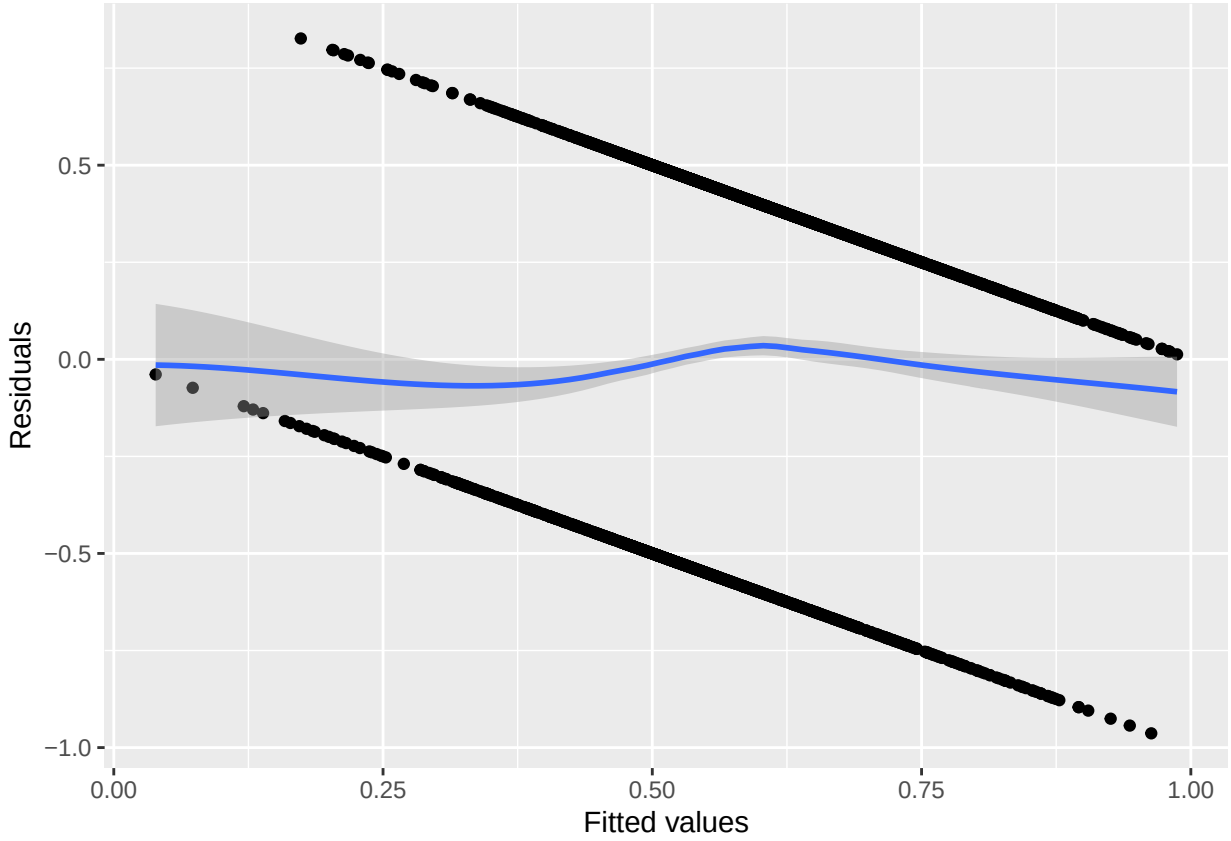
```
##
## Call:
## glm(formula = switch ~ dist * arsenic + educ, family = "binomial",
##     data = wells)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7149  -1.1886   0.7478   1.0689   1.7223
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (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()`.

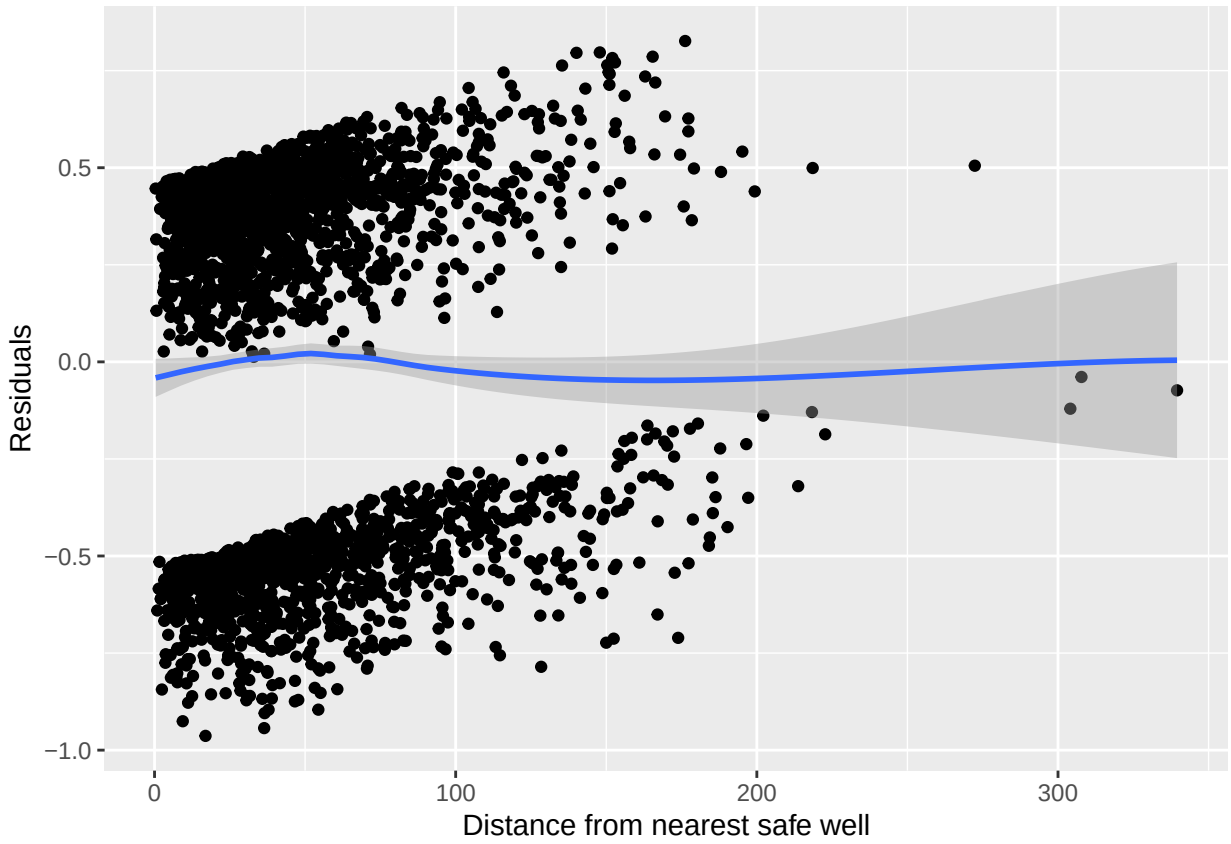
```
switch.model.df = wells
switch.model.df$.fitted = fitted.values(switch.model)
switch.model.df$.resid = residuals(switch.model, type = "response")
ggplot(switch.model.df, aes(x = .fitted, y = .resid)) +
  geom_point() +
  geom_smooth(method = "loess", method.args = list(degree = 1)) +
  xlab("Fitted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



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.

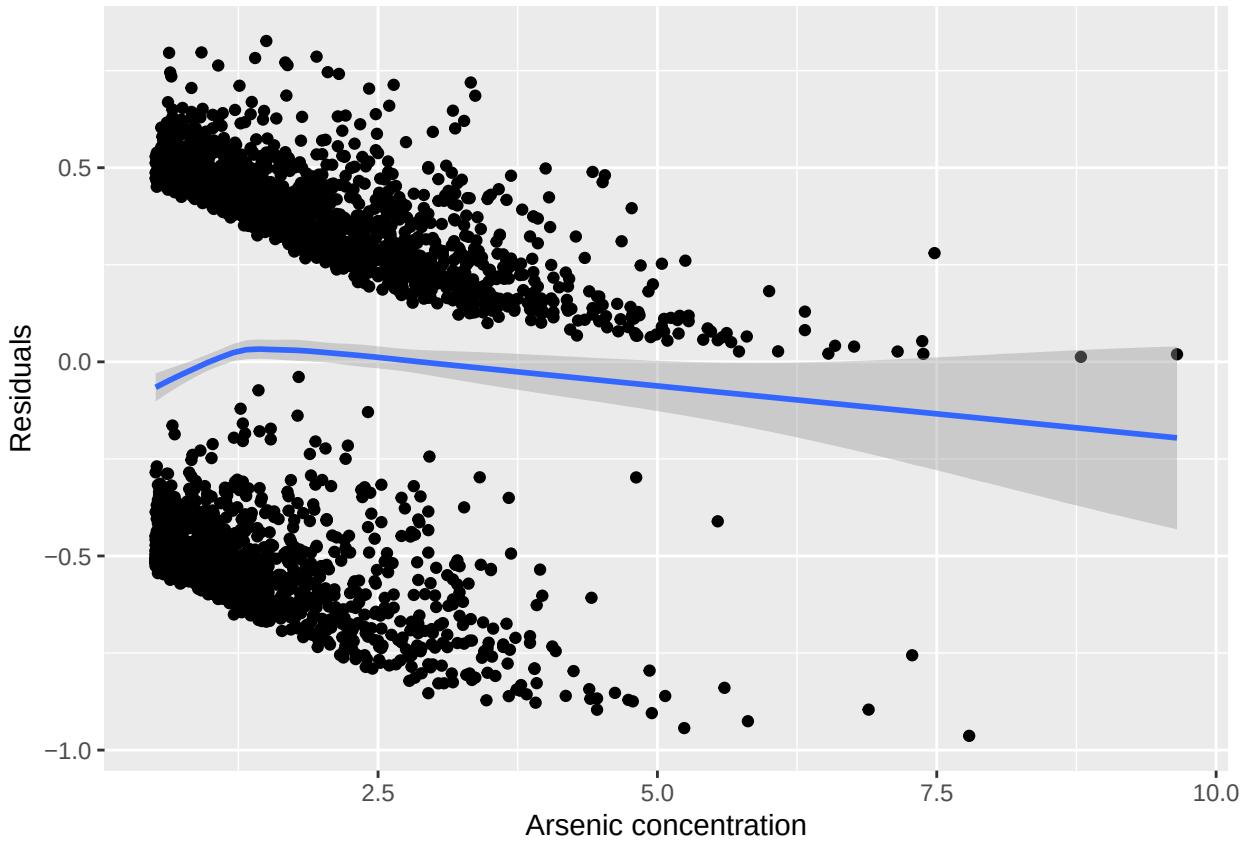
Now plot the residuals against distance.

```
ggplot(switch.model.df, aes(x = dist, y = .resid)) + geom_point() +
  geom_smooth(method = "loess", method.args = list(degree = 1)) +
  xlab("Distance from nearest safe well") + ylab("Residuals")
```



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:

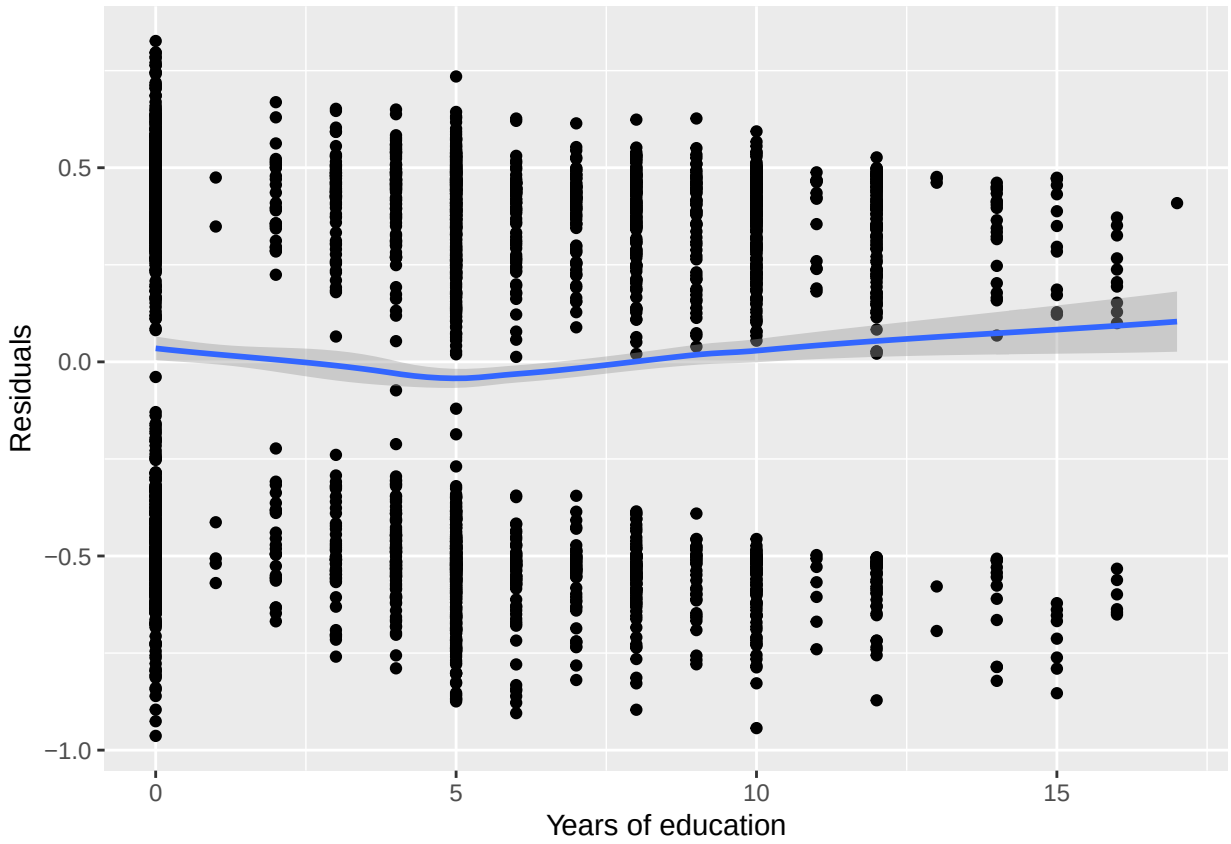
```
ggplot(switch.model.df, aes(x = arsenic, y = .resid)) +
  geom_point() +
  geom_smooth(method = "loess", method.args = list(degree = 1)) +
  xlab("Arsenic concentration") + ylab("Residuals")
```



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 is 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)) +
  xlab("Years of education") + ylab("Residuals")
```



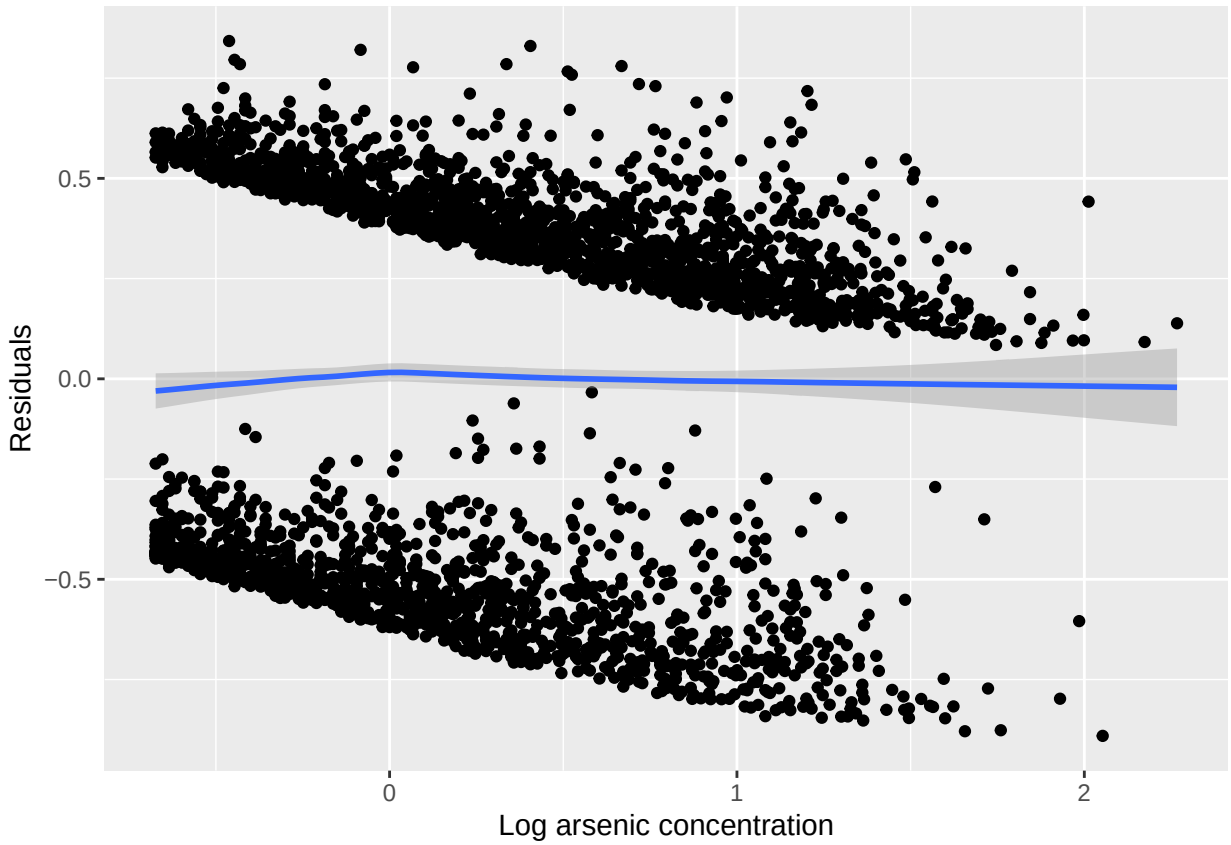
Again, the fit is imperfect.

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.

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

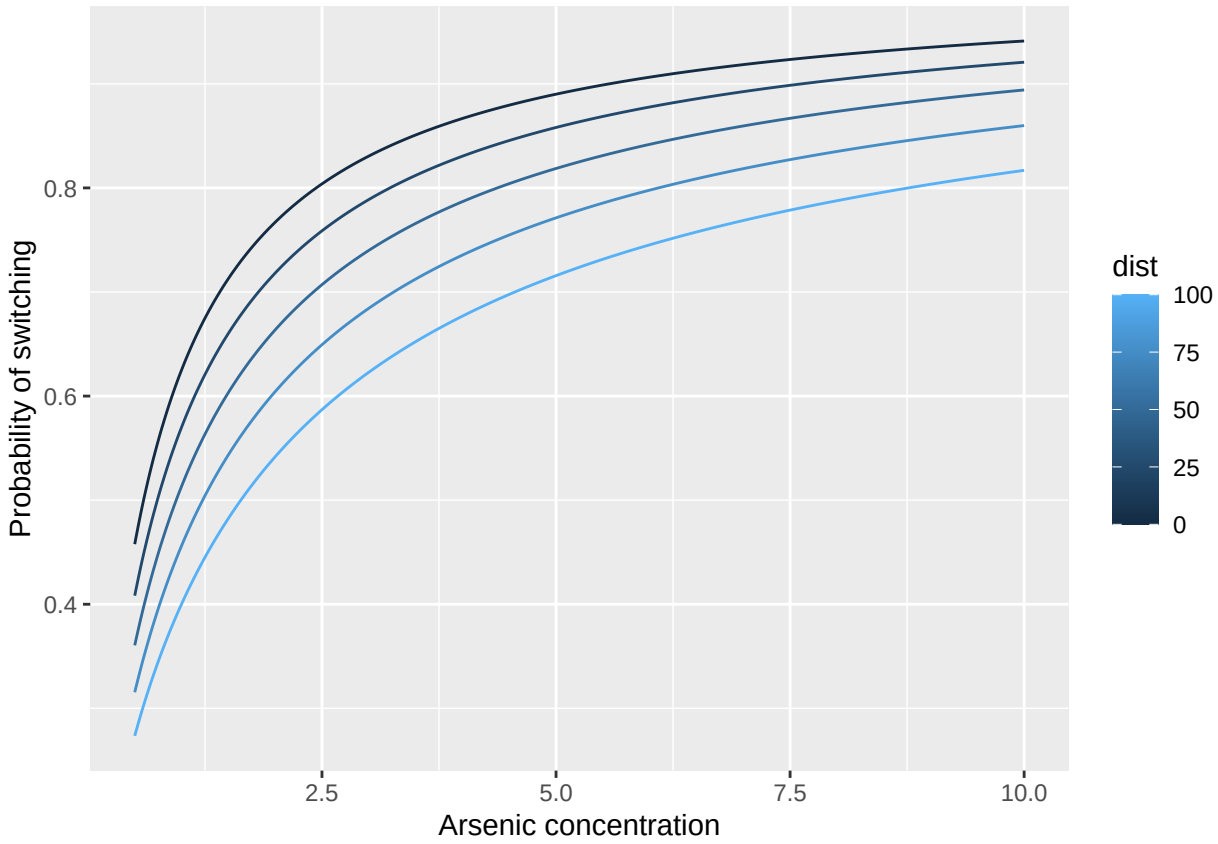
Now plot the residuals of this new model against log arsenic:

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



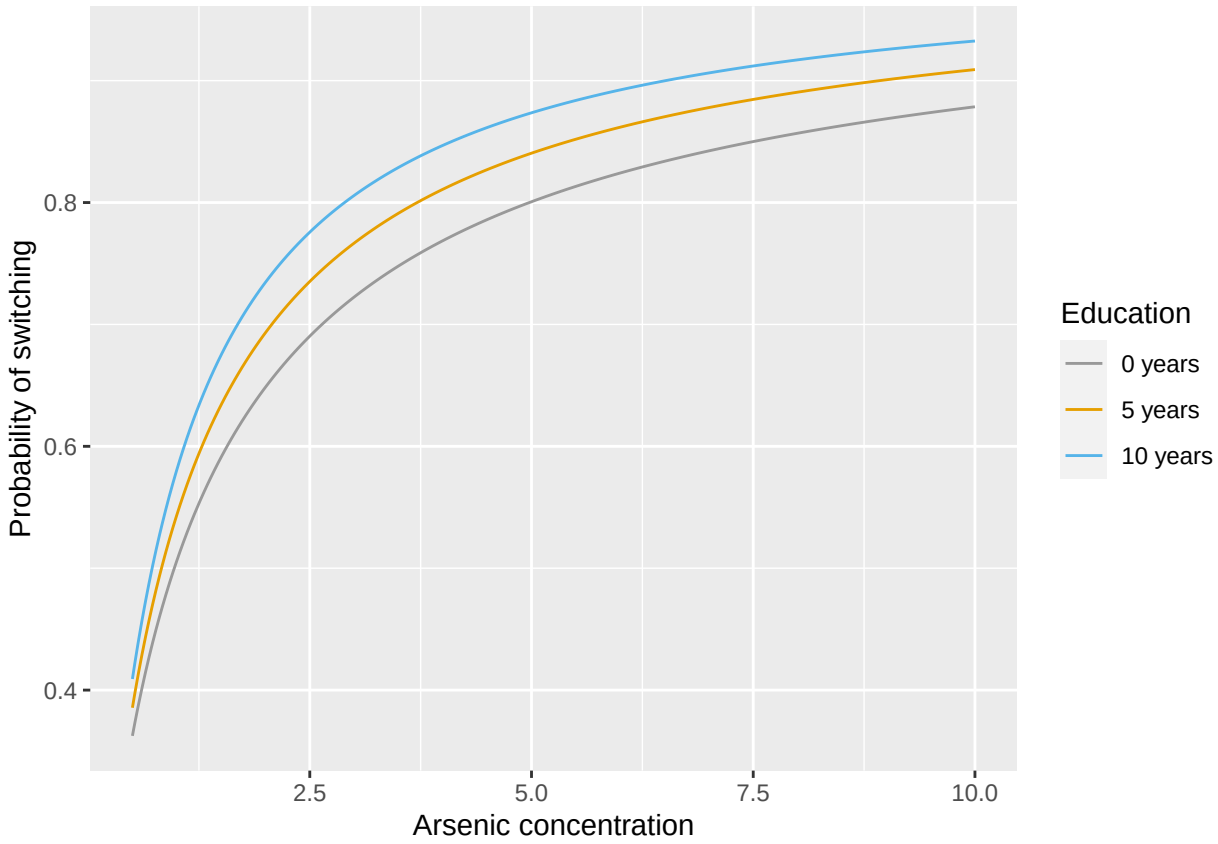
It's a lot better. I'm fairly happy with the model at this point. Let's take a look at the fit. Fix education at its median, and draw arsenic curves for several distances:

```
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() +
  xlab("Arsenic concentration") + ylab("Probability of switching")
```



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() +
  xlab("Arsenic concentration") + ylab("Probability of switching") +
  labs(color = "Education") + scale_color_manual(values = cb_palette,
    labels = c("0 years", "5 years", "10 years"))
```



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 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.

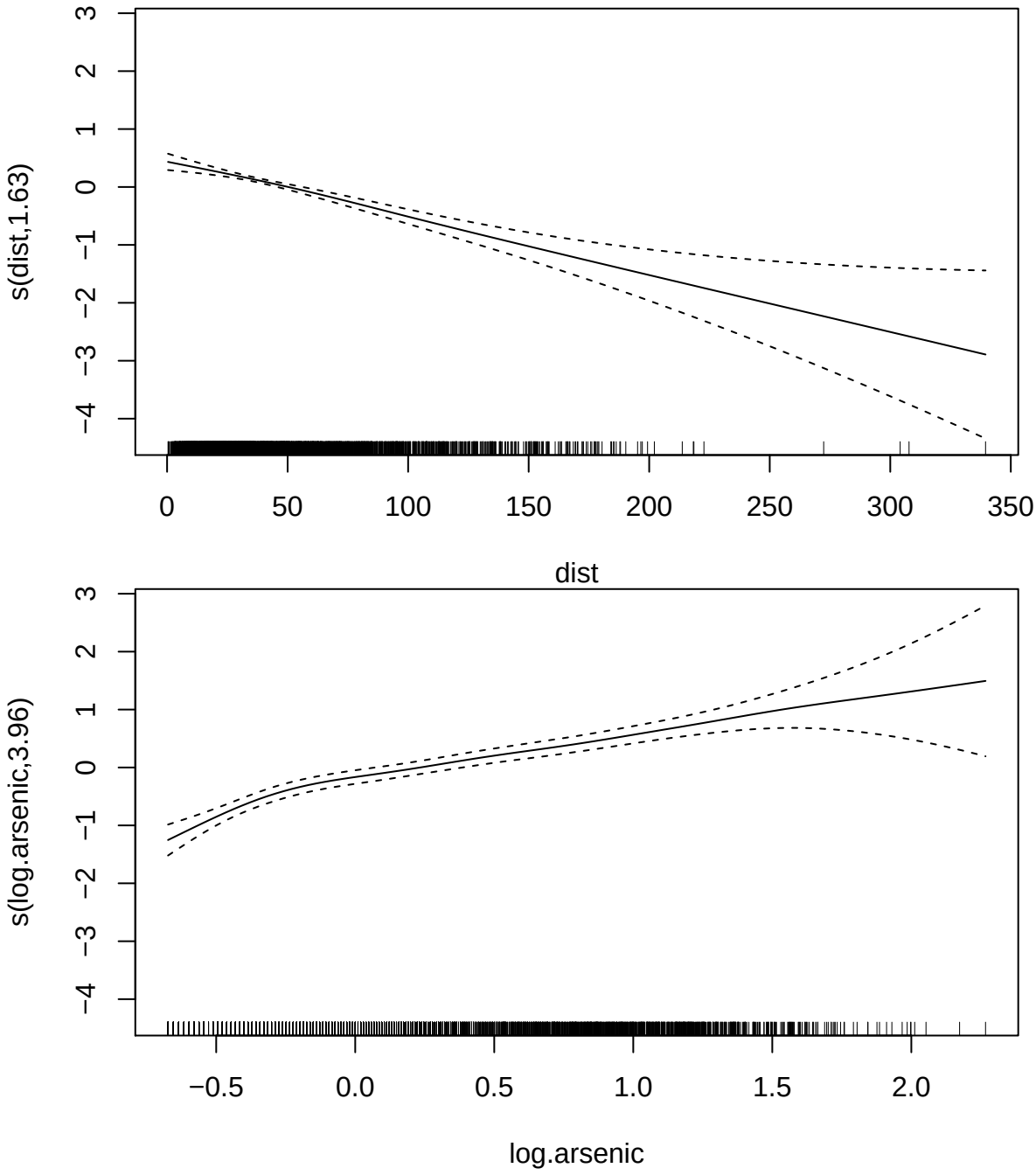
If you really, really care about predictions then you can use machine learning techniques. The improvement in prediction is usually quite small, however, and you'll lose a lot or all the interpretability.

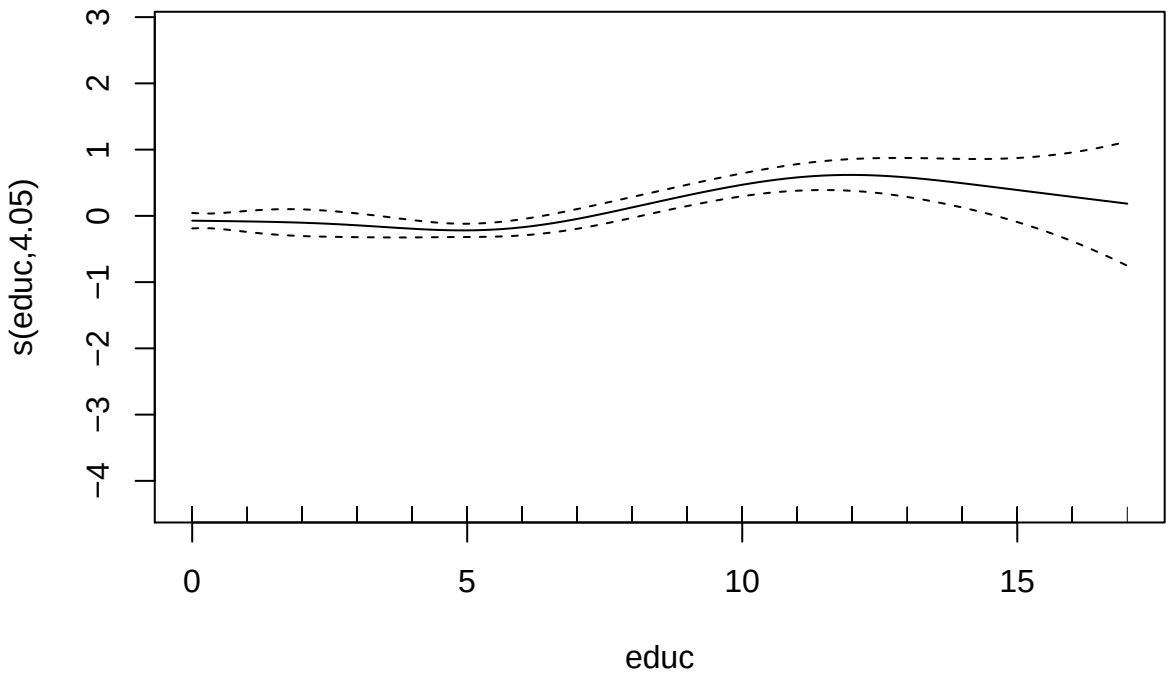
Try fitting a GAM. Note that for models that are not fundamentally least-squares based (e.g. almost anything non-Gaussian), using REML (restricted maximum likelihood) reduces the risk of overfitting.

```
library(mgcv)
switch.gam = gam(switch ~ s(dist) + s(log.arsenic) + s(educ),
    family = "binomial", data = wells, method = "REML")
```

Plot the partial response function:

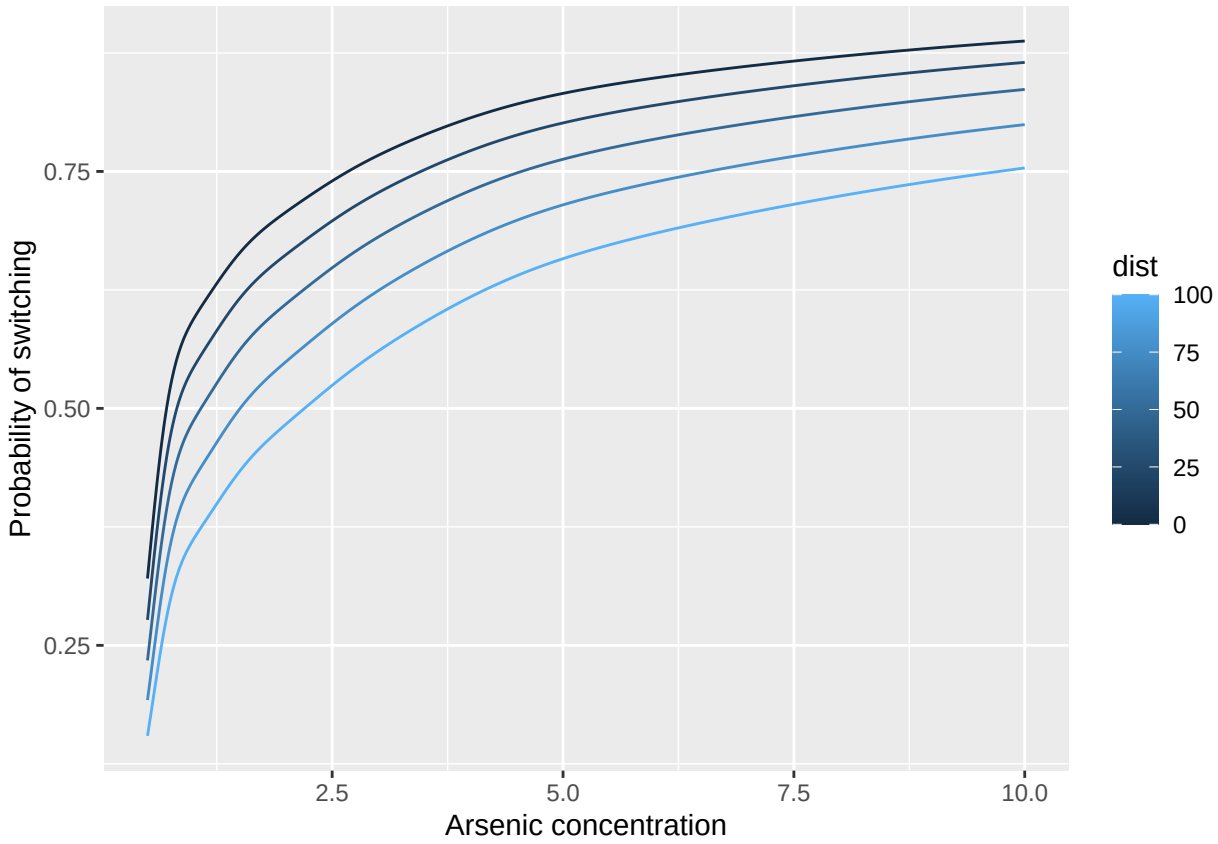
```
plot(switch.gam)
```


s(dist,1.63)
3
2
1
0
−1
−2
−3
−4
0
50
100
150
200
250
300
350
dist
s(log.arsenic,3.96)
3
2
1
0
−1
−2
−3
−4
−0.5
0.0
0.5
1.0
1.5
2.0
log.arsenic



Plot the results:

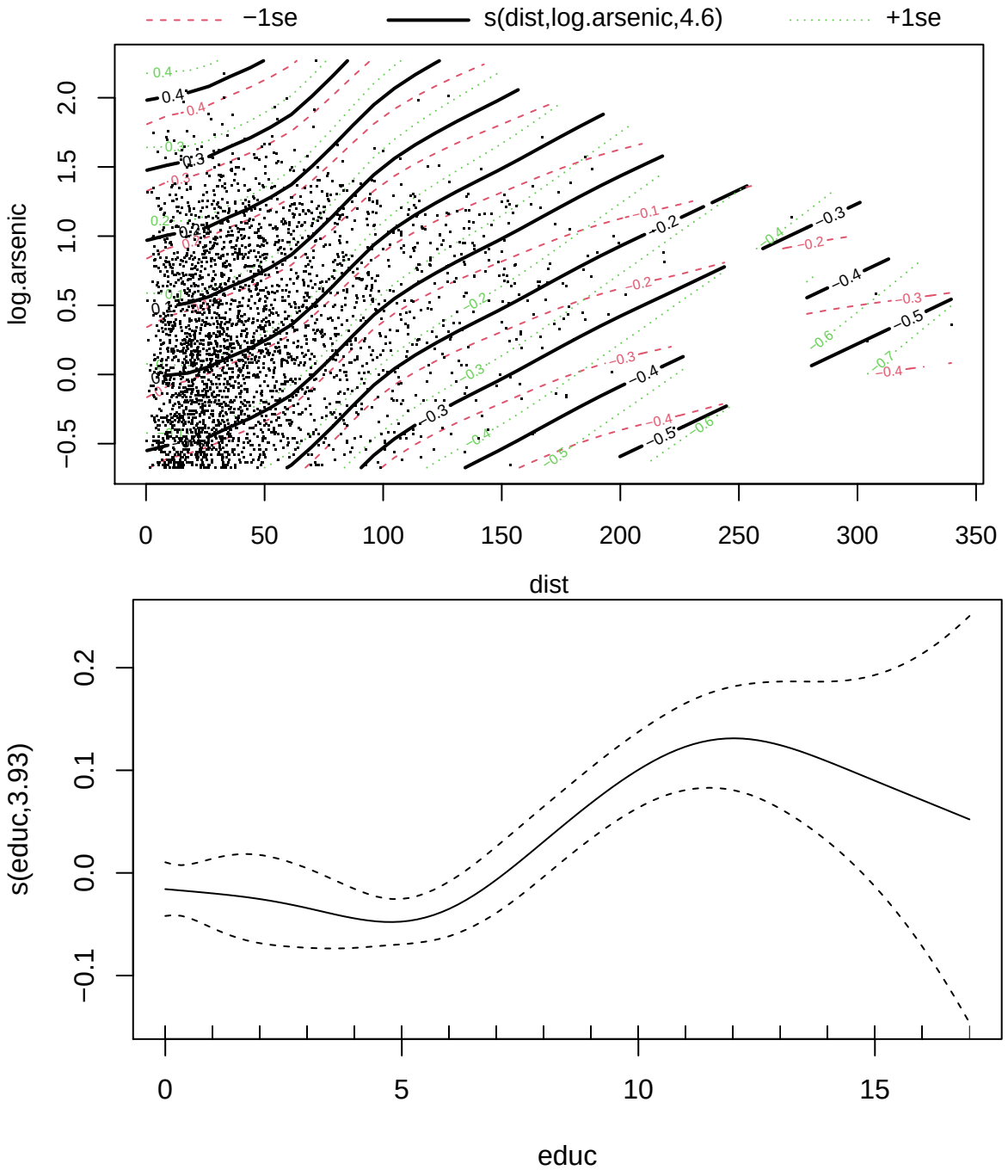
```
gam.arsenic.pred =
  predict(switch.gam, newdata = switch.grid, type = "response")
gam.arsenic.grid =
  data.frame(switch.grid, switch.prob = as.vector(gam.arsenic.pred))
ggplot(gam.arsenic.grid, aes(x = arsenic, y = switch.prob,
    group = dist, color = dist)) + geom_line() +
  xlab("Arsenic concentration") + ylab("Probability of switching")
```



Does an interaction work?

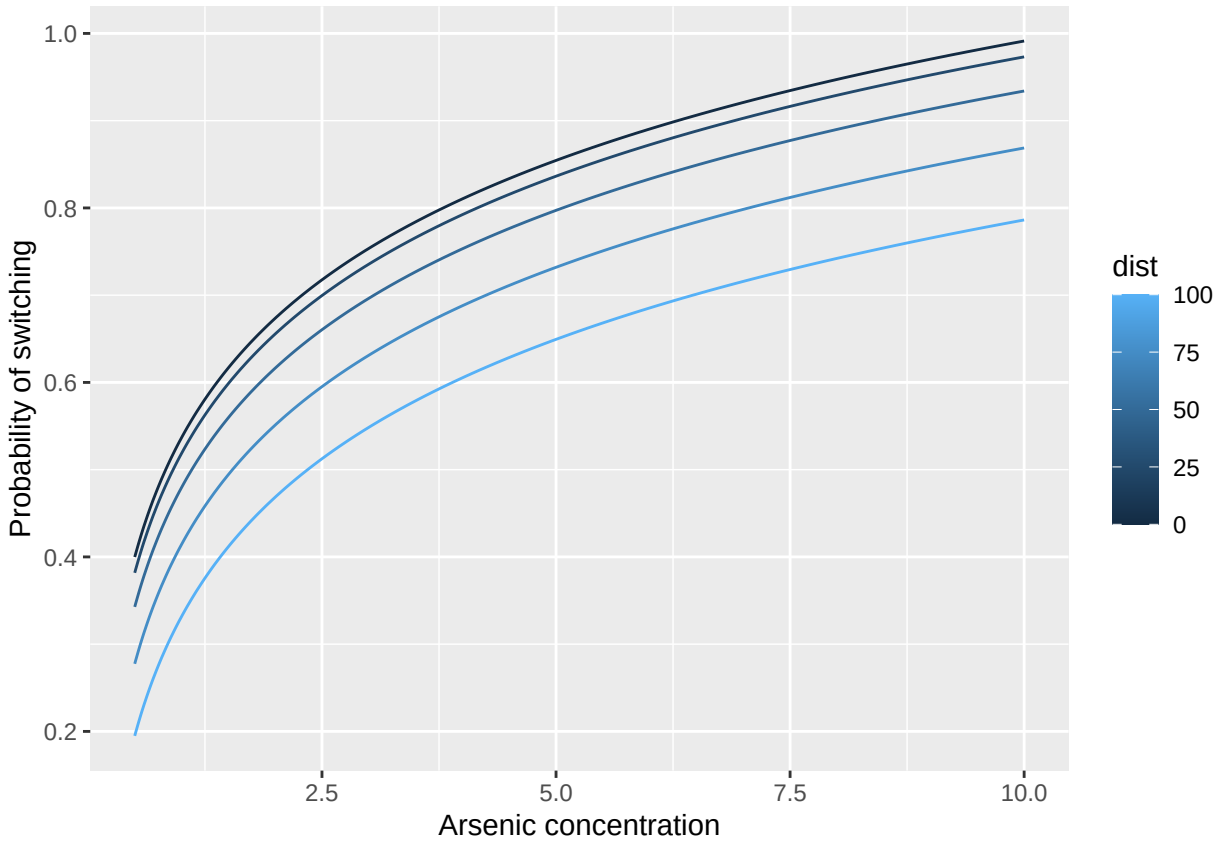
```
switch.gam2 = gam(switch ~ s(dist, log.arsenic) + s(educ),
    amily = "binomial", data = wells, method = "REML")
```

```
plot(switch.gam2)
```

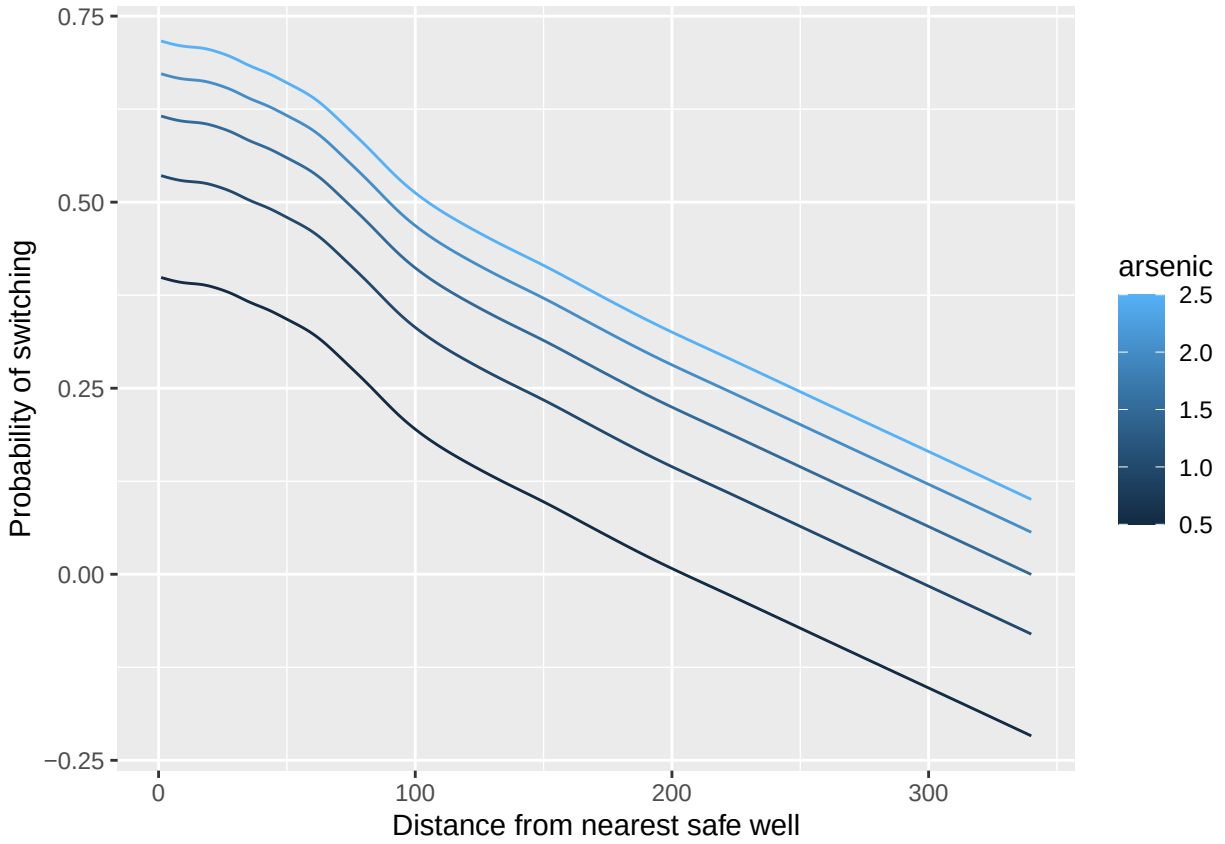


Plot on a grid:

```
gam.arsenic.pred2 =
  predict(switch.gam2, newdata = switch.grid, type = "response")
gam.arsenic.grid2 =
  data.frame(switch.grid, switch.prob = as.vector(gam.arsenic.pred2))
ggplot(gam.arsenic.grid2, aes(x = arsenic, y = switch.prob,
    group = dist, color = dist)) + geom_line() +
  xlab("Arsenic concentration") + ylab("Probability of switching")
```




```
switch.grid3 = expand.grid(arsenic = seq(0.5, 2.5, 0.5),
    dist = seq(0:339), educ = 5)
switch.grid3$log.arsenic = log(switch.grid3$arsenic)
gam.dist.pred =
  predict(switch.gam2, newdata = switch.grid3, type = "response")
gam.dist.grid =
  data.frame(switch.grid3, switch.prob = as.vector(gam.dist.pred))
ggplot(gam.dist.grid, aes(x = dist, y = switch.prob,
    group = arsenic, color = arsenic)) + geom_line() +
  xlab("Distance from nearest safe well") +
  ylab("Probability of switching")
```



What do you think?