FES 524 Winter 2022 Lab 7

Contents

Generalized linear models for counted proportion data

In Lab 7 we make a big switch away from the linear models and linear mixed models we've been working with throughout the term and dip our toes into the world of generalized linear models.

Our response variable this week is the proportion of fish in tanks that have tumors. The research questions of interest are about comparing the proportion of fish with tumors under higher doses of aflatoxin to that of the low dose of aflatoxin (0.01

mg/L). Inference is to multiplicative differences in odds of getting tumors.

Load R packages

We'll be using **dplyr** for data manipulation, **ggplot2** for plotting, and **emmeans** for comparisons as we have done in past weeks.

```
library(dplyr)
library(ggplot2)
library(emmeans)
```

Read in the dataset

We will be working with the dataset `lab7.example.data.csv`, so make sure you have this file and have set your working directory appropriately. As usual when we read in a dataset we'll take a look at the structure and make any necessary changes.

```
fish = read.csv("lab7.example.data.csv")

head(fish) # Look at the first 6 lines of the dataset
```

```
  tank dose num.tumor num.fish
1   12 0.00         0       28
2   10 0.00         0       22
3   11 0.00         0       35
4    5 0.00         0       38
5   19 0.01         1       23
6    2 0.01         0       22
```

Look at the structure in the Environment pane.

```
'data.frame':   25 obs. of  4 variables:
 $ tank     : int  12 10 11 5 19 2 3 17 4 22 ...
 $ dose     : num  0 0 0 0 0.01 0.01 0.01 0.01 0.01 0.01 ...
 $ num.tumor: int  0 0 0 0 1 0 1 0 2 0 ...
 $ num.fish : int  28 22 35 38 23 22 39 27 20 33 ...
```

We can see right away that the `dose` variable is numeric. Different doses (mg/L) of aflatoxin are quantitative and range from no dose (the control) to a high dose of 0.5 mg/L. However, the research question today is about differences among specific doses, and so we'll want to use `dose` as a categorical variable. Let's make a new variable, `fdose`, that is a factor.

```
fish$fdose = factor(fish$dose)
```

Calculate the observed proportion of fish with tumors

Another thing to note in the dataset is that while we have a variable for the number of fish with tumors (`num.tumor`) and a variable for the total number of fish in each tank (`num.fish`), we don't have a variable for the observed proportion of fish with tumors in each tank. This variable is easy to calculate based on the information we do have, so let's make a new variable `prop.tumor` and add it to the dataset. We'll use `mutate()` for this to avoid dollar sign notation.

Note: Whenever you are working with counted proportions in R you need to have at the least the total number of trials for each observation (e.g., the number of fish in each tank) in addition to the observed proportions. Be sure you keep this sort of information in your dataset.

```
fish = mutate(fish, prop.tumor = num.tumor/num.fish)
```

As always, check the structure of the dataset again to make sure things look how you expect them to. This is easiest by looking in your RStudio Environment pane, but you can also use `str(fish)`.

Initial data exploration

Statistical summaries

We're still working with categorical explanatory variables, so let's do our standard summaries of our variable of interest, the proportion of fish with tumors, by each dose group.

Check out the control group (when the aflatoxin dose is 0); what's going on there? No fish in the control group ever had tumors. This doesn't matter a lot to us today because we have no research questions that involve the control group, but this will give us a chance to see what happens when there is no variability in the response variable when analyzing counted proportions with a logit link.

Also note that the standard deviations for each group are pretty different. We will model counted proportion data using the *binomial distribution* rather than the normal distribution that we've been working with all quarter. In the binomial distribution, the variance is directly related to the mean. When the mean changes, the variance also changes. This signifies that we no longer have the assumption of constant variance among groups, and is one reason that trying to use the normal distribution when working with counted proportions often doesn't work very well.

```
fish %>%
    group_by(dose) %>%
    summarise(n = n(),
            mprop = mean(prop.tumor),
            sdprop = sd(prop.tumor),
            minprop = min(prop.tumor),
            maxprop = max(prop.tumor),
            .groups = "drop")
```

```
# A tibble: 4 x 6
   dose     n  mprop sdprop minprop maxprop
  <dbl> <int>  <dbl>  <dbl>   <dbl>   <dbl>
1  0        4 0      0       0       0
2  0.01     7 0.0293 0.0361  0       0.1
3  0.1      7 0.562  0.0983  0.429   0.706
4  0.5      7 0.747  0.189   0.414   1
```

Graphical data exploration on the data scale

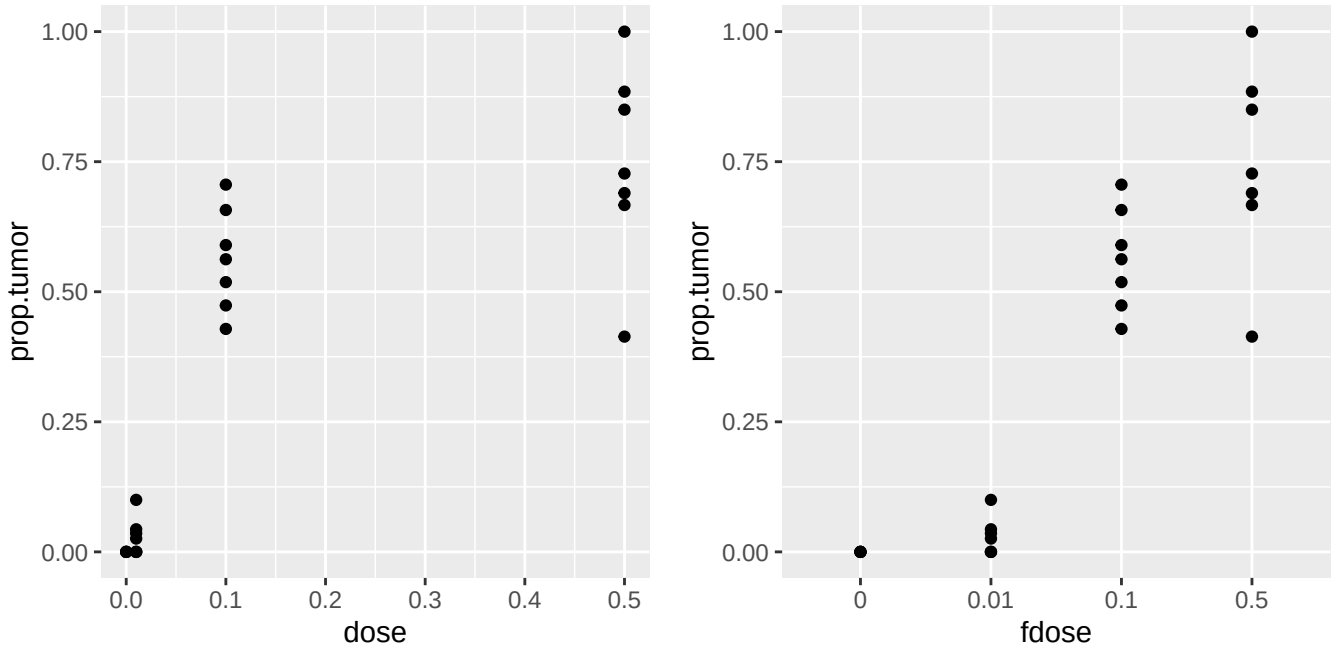
Graphical data exploration for counted proportions is a bit different than when working with continuous response variables. When working with non-Gaussian generalized linear models, the mean of the response variable is related to the additive model via some *link* function. This is the function that *links* the mean to the linear model. With counted proportions, you'll most often see what's called the *logit* link. This is why some versions of binomial generalized linear models are referred to as *logistic regression*.

The logit is equivalent to the natural logarithm of the odds (remember that odds are calculated by dividing a proportion by 1 minus the proportion). We'll be exploring the dataset on the original scale of the data (the *data scale*) as well as on the scale of link (the *logit scale*).

On the original scale, we can see there is no overlap between the reference low dose (`0.01`) and the higher doses so the differences in proportions are stark.

```
# Using continuous dose
qplot(x = dose, y = prop.tumor, data = fish)

# Using factor dose, which makes groups equidistant in the graph
qplot(x = fdose, y = prop.tumor, data = fish)
```



Graphical data exploration on the link scale

Now we want to explore the dataset on the scale of the link. Exploring the dataset on the logit scale is most interesting when explanatory variables are continuous.

When working with generalized linear models we work directly with the original data and don't use any transformations of the response. For data exploration purposes only, though, we may need to transform the response variable. Remember, this is *only for the purposes of graphing*, so take care to always go back to the raw data when you start the generalized linear model analysis.

Creating a variable for graphing on the logit scale

We'll be making a new variable called `elprop.tumor`. The `e` stands for empirical, which I use to remind myself that this is a transformation of the observed data and not what I'll be modeling.

We'll calculate the log odds based on the proportions we've already calculated in `prop.tumor`. We can either do all the math "by hand", taking the `log()` of p/(1-p) or use the built-in `qlogis()` function directly on the proportion variable. The results are the same.

I'll use `with` here to avoid dollar sign notation while printing the results for further discussion.

```
# We could make the new variable directly using the log() function and manually
    # making the odds
with(fish, log(prop.tumor/(1 - prop.tumor) ) )
```

```
 [1]        -Inf        -Inf        -Inf        -Inf -3.09104245        -Inf -3.63758616        -Inf
 [9] -2.19722458        -Inf -3.29583687  0.07410797  0.65058757 -0.10536052  0.36290549  0.25131443
[17]  0.87546874 -0.28768207  0.69314718         Inf  1.73460106  2.03688193 -0.34830669  0.79850770
[25]  0.98082925
```

```
# R has a built in function for the logit, though, called qlogis(),
    # which does the same thing as above with less typing
with(fish, qlogis(prop.tumor) )
```

```
 [1]        -Inf        -Inf        -Inf        -Inf -3.09104245        -Inf -3.63758616        -Inf
 [9] -2.19722458        -Inf -3.29583687  0.07410797  0.65058757 -0.10536052  0.36290549  0.25131443
[17]  0.87546874 -0.28768207  0.69314718         Inf  1.73460106  2.03688193 -0.34830669  0.79850770
[25]  0.98082925
```

Trouble with 0 and 1 values

Notice that we have a problem: the log of 0 is `-Inf` and the log of 1 divided by 0 is `Inf`. Having 0 and 1 values in the dataset makes data exploration more difficult, although having those values are not necessarily problematic in the actual analysis. We'll need to get our response value strictly between (and not including) 0 and 1 before we can proceed with data exploration.

Using package car to remap proportions for graphing

The **car** package has a `logit()` function that is specifically designed for cases when we need to work on the logit scale for data that contains 0 and/or 1 values. It remaps the observed proportions so they are strictly between (not including) 0 and 1. This is a fine thing to do when trying to plot the data on the logit scale, but should **not** be something you do as a transformation for analysis without a lot of thought on your part. The logit transformation can be useful for continuous proportions in some situations, but the presence of 0 and 1 values will lead to some hard thinking.

Let's load the **car** package (if you don't have it installed, you'll need to do so).

```
library(car)
```

Take a look at the help page for `logit()`. We're using this function for the `adjust` argument.

```
?logit # Notice the "adjust" argument
```

Now we can make our new variable of the empirical logit proportions, `elprop.tumor`. Note the message from `logit()` that warns us that it remapped the variable and what it remapped it to.

```
fish = mutate(fish, elprop.tumor = logit(prop.tumor) )
```

```
Warning in logit(prop.tumor): proportions remapped to (0.025, 0.975)
```

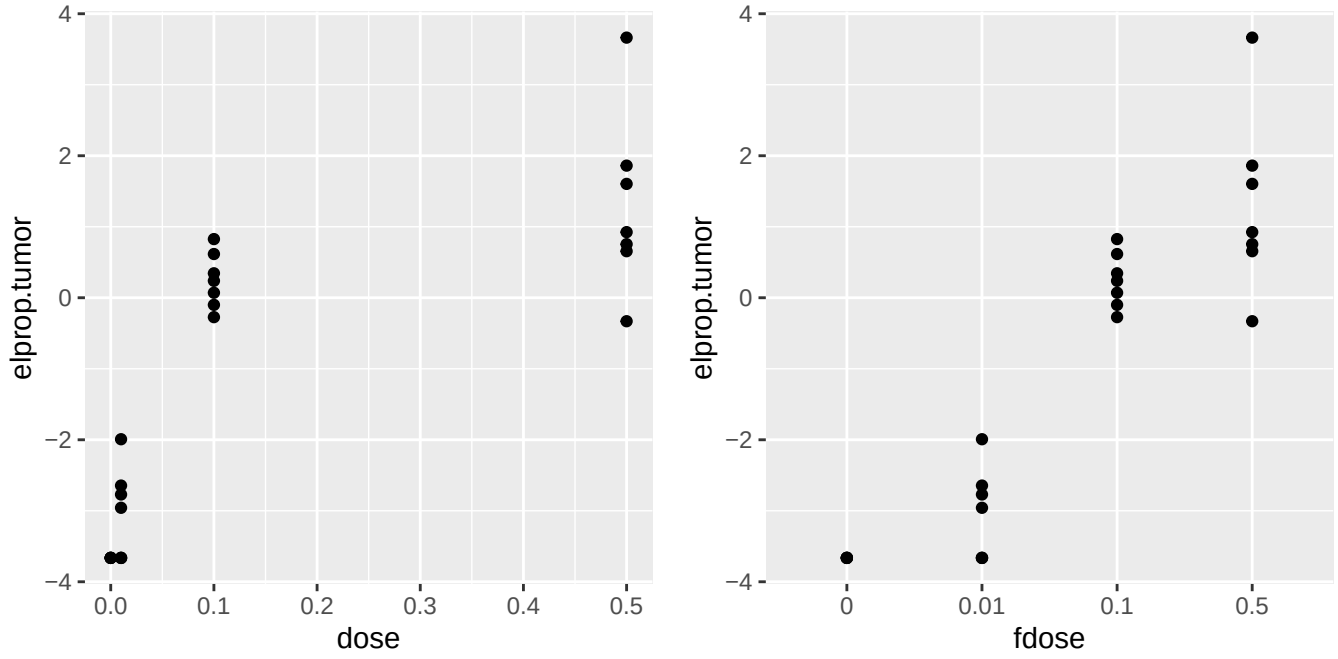
```
fish$elprop.tumor
```

```
 [1] -3.66356165 -3.66356165 -3.66356165 -3.66356165 -2.64489506 -3.66356165 -2.95801692 -3.66356165
 [9] -1.99243016 -3.66356165 -2.77069299  0.07039943  0.61590097 -0.10008346  0.34438959  0.23862592
[17]  0.82637660 -0.27311366  0.65587579  3.66356165  1.60344987  1.86075234 -0.33056380  0.75456426
[25]  0.92425890
```

With that done, we can explore the relationships among our explanatory variables and the counted proportions on the logit scale. Things look more additive on the scale to my eye than the previous plots, but other than that not much is different.

```
# Continuous dose (which we will not be using today)
qplot(x = dose, y = elprop.tumor, data = fish)

# Categorical dose
qplot(x = fdose, y = elprop.tumor, data = fish)
```



Graphical data exploration for continuous explanatory variables

For continuous explanatory variables, it can be useful to add a smoothed line to the graph. This can help us see patterns in noisy data. We have to remember that the smoothed line is just for data exploration; don't get too attached to what you see in the smoothed line! And be sure to remove the confidence ribbon in data exploration by using `se = FALSE`.

I'm bringing this up as an aside mostly because when working with data that are strictly binary (the response variable is either a 0 or a 1), data exploration gets tricky. Plotting the 0 and 1 values and adding a smoothed line can be helpful in your understanding of the dataset when the continuous variable is continuous. Otherwise, for categorical explanatory variables, the main tool you have at your disposal is jittering.

Here's how we could do this (along with jittering the points if all our values were the same), using `geom_smooth()`. The warning message you get is due to the very small dataset in this case.

This plot is not interesting in this case. I am showing this so you know how to do this if it is relevant in your own data.

```
ggplot(fish, aes(x = dose, y = prop.tumor) ) +
    geom_jitter() +
    geom_smooth(se = FALSE)
```

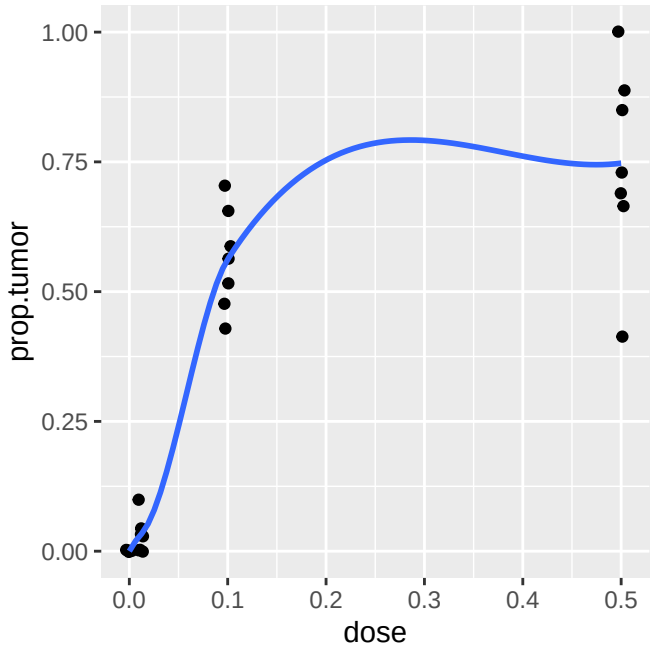
```
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric, : pseudoinverse used
at -0.0025
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric, : neighborhood
radius 0.1025
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric, : reciprocal
condition number 9.4466e-017
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric, : There are other
near singularities as well. 0.24256
```



Fitting a binomial generalized linear model

We'll fit a binomial generalized linear model with a logit link (aka logistic regression model) using the `glm()` function.

Available distributions and links in `glm()`.

Let's take a look at the help page, noting in particular the `family` argument and its defaults.

```
?glm
```

The `family` argument is new to us and is important when working with generalized linear models. On the help page for `family`, we can see the distributions we can work with along with the default link functions. The description of the `link` argument tells us alternative links for each family. Note it defaults to `gaussian` (aka a normal distribution).

While we will be using the default today for our binomial dataset and so don't need to specify it, we also have the `probit`, `cauchit`, `log`, and `cloglog` links available to us for the binomial distribution.

```
?family
```

Initial model

Let's fit our initial model. The strategy when working the generalized linear models is to fit a full model, meaning with as many explanatory variables as we are comfortable with, and start our evaluation of model fit with that full model. Today we only have a single explanatory variable - we don't have any other variables in the dataset that could be useful - so the model with just `fdose` is our fullest reasonable model.

When fitting a model using counted proportions in R, we will need to somehow indicate the total number of trials for each observation. There are two ways to do this `glm()`, and I'll show you both.

`glm()` using number of successes and number of failures

First, we can give our response as both the number of successes (in this case, the number of fish with tumors) and the number of failures using `cbind()`. The number of failures is the total number of trials minus the number of successes. Note also the use of the `family` argument, making sure we are using the binomial distribution. We use a logit link, which we just learned is the default.

```
fit1 = glm(cbind(num.tumor, num.fish - num.tumor) ~ fdose,
          data = fish,
          family = binomial)
```

`glm()` using observed proportion with total trials as weights

Our second option is to use the proportion as the response variable in the formula, and define the total number of trials (i.e., the total number of fish in each tank) in the `weights` argument. The important point here is that we don't just use the proportion as the response; we have to have the total counts, as well.

```
fit2 = glm(prop.tumor ~ fdose,
         data = fish,
         family = binomial,
         weights = num.fish)
```

The results of these models are identical, so we get the same estimates no matter which style we decide we like. We'll be using the second model, `fit2`, for the rest of today, but would get all the same results with `fit1`.

```
fit1
```

```
Call:  glm(formula = cbind(num.tumor, num.fish - num.tumor) ~ fdose,
    family = binomial, data = fish)

Coefficients:
(Intercept)    fdose0.01     fdose0.1     fdose0.5
     -21.14        17.52        21.43        22.17

Degrees of Freedom: 24 Total (i.e. Null);  21 Residual
Null Deviance:      433.8
Residual Deviance: 48.95    AIC: 112.5
```

```
fit2
```

```
Call:  glm(formula = prop.tumor ~ fdose, family = binomial, data = fish,
    weights = num.fish)

Coefficients:
(Intercept)    fdose0.01     fdose0.1     fdose0.5
     -21.14        17.52        21.43        22.17

Degrees of Freedom: 24 Total (i.e. Null);  21 Residual
Null Deviance:      433.8
Residual Deviance: 48.95    AIC: 112.5
```

```
# The coefficients are identical
identical(coef(fit1), coef(fit2) )
```

```
[1] TRUE
```

Checking the model fit

Once we've fit a model, we need to check if the model fits OK. This includes the usual check of the residuals vs fitted values to look for any patterns.

Checking for overdispersion

We also need to check that the mean-variance relationship of the binomial distribution is reasonable for the data. Remember that in the binomial distribution the mean and variance are related. The assumed mean-variance relationship can be overly restrictive for real datasets.

When the variance of the dataset is bigger than what the variance defined by the binomial distribution, it's called *overdispersion*. If we have overdispersion, standard errors used for any testing or building confidence intervals are too small. This makes any statistical inference anti-conservative. When we have substantial overdispersion, we usually need an alternative modeling strategy.

We check for overdispersion by dividing the sum of the squared Pearson residuals by the residual degrees of freedom from the model and comparing that value to 1. If it is 1 or less (aka underdispersed), we won't be concerned. If it is larger than 1, we likely need a model that addresses the overdispersion.

```
sum( residuals(fit2, type = "pearson")^2 )/fit2$df.res
```

```
[1] 2.086026
```

The estimated overdispersion is ~2.1, so there does appear to be a fairly substantial amount of overdispersion. This indicates that the variance in the response variable is bigger than it should be if our data really come from the binomial distribution. This is extremely common when working with binomial data.

Residual plots

Once we have established that we see evidence of overdispersion, we need to spend some time trying to figure out if something is causing the overdispersion that we can control. First, we'd make sure the problem isn't being caused by any outliers by looking at our residual plots. We also want to ensure that the problem isn't being caused by a missing covariate. However, overdispersion from a missing covariate can be hard to diagnose, and is the reason why we check for overdispersion on a full model that has all terms that we have that we think causes variation in the response variable.

Types of residuals

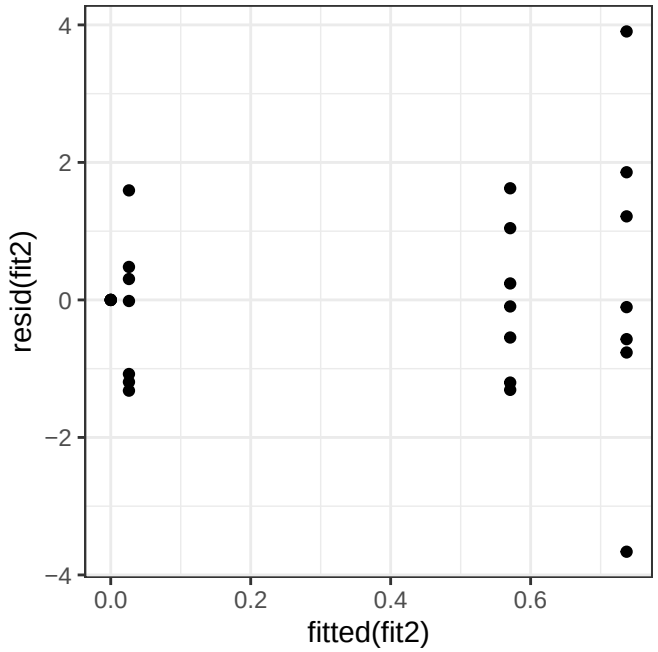
We'll start by checking the residual plot. We are now working with *glm* objects, and so functions like `residuals()` will be specific to that object type. Let's take a look at the help page for `residuals.glm` to check out the defaults.

```
?residuals.glm
```

Deviance residuals can be useful when looking for outliers because they are scaled in such a way that they should mostly fall between -2 and 2. The deviance residuals are the default residual type for `residuals()` for glm objects.

Let's look at the residual vs fitted plot.

```
qplot(x = fitted(fit2), y = resid(fit2) ) +
    theme_bw()
```



We see that 2 out of the 25 deviance residuals are substantially outside of -2 to +2 range, which is more evidence of the overdispersion we saw above. There are not any specific outliers, though, so it doesn't seem like the overdispersion is caused by a single unusually large residual that we could investigate further. We would probably want to investigate the highest dose group to see what might be going on there but, of course, in this example data don't have the ability to change anything.

The overdispersion could still be caused by a covariate missing from the model. For example, if something like the water temperature varied among tanks then our response may be noisier than expected (the variance among the response would increase), and so we could use that as a covariate in the model. However, we don't have any covariates to use today. We are fitting our fullest model and have nothing to add to the model.

All of this taken together means the model doesn't fit that data well and we didn't find any way to modify the model to make it fit better; we now need to find an alternative model.

Alternatives to a binomial generalized linear model

There are three alternatives to the binomial generalized linear model when dealing with overdispersion in counted proportions (given in no particular order).

Observation-level random effect

We could add an observation-level random effect to account for observation-level noise that is accounted for by the binomial model. We could never do this with Gaussian linear models because the observation-level random effect *is* the residual error term. When working with a one parameter distribution, where the variance is solely a function of the mean, we can add observation-level random effects to model the overdispersion. This would put us into the world of generalized linear *mixed* models.

Beta-binomial

We could fit a beta-binomial model, where we allow the observed proportions to come from the beta distribution, which we then model via a binomial generalized linear model. This can be done in the **VGAM** package in R.

Quasibinomial

We could fit a quasilikelihood model by using `family = quasibinomial`. This multiplies all standard errors from the binomial by the square root of the estimated dispersion. This can be simple and useful, although we no longer have the standard likelihood tools available to us. Traditionally, we do t and F tests when using quasilikelihood distributions instead of the standard z and χ^2 tests we use in generalized linear models.

Today we will fit a quasibinomial generalized linear model.

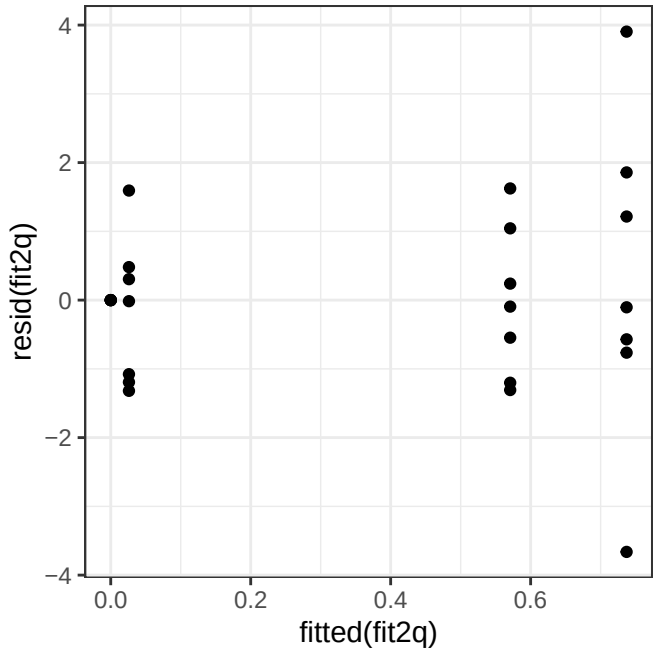
Fitting a quasibinomial generalized linear model

We can fit a quasibinomial generalized linear model in R by using `family = quasibinomial` in `glm()`. We will still use the `logit` link, which is the default for the quasibinomial family.

```
fit2q = glm(prop.tumor ~ fdose,
          data = fish,
          family = quasibinomial,
          weights = num.fish)
```

The residuals from this model haven't changed, so while we can plot the residual vs fitted values again we won't see anything new.

```
qplot(x = fitted(fit2q), y = resid(fit2q) ) +
    theme_bw()
```



Normality and non-Gaussian generalized linear models

Please note that we haven't (and aren't going to) check for the normality of the residuals. *There is no assumption of normality of errors in non-Gaussian generalized linear models.* The deviance residuals *could* be normally distributed in certain situations and you may see folks making quantile-quantile normal plots of the deviance residuals. However, this is not an actual assumption we are making with this model.

Model results

We now have a model to make inference with. Let's take a look at the `fit2q` model, starting with the summary output.

```
summary(fit2q)
```

```
Call:
glm(formula = prop.tumor ~ fdose, family = quasibinomial, data = fish,
    weights = num.fish)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.6623  -0.7647  -0.0002   0.4799   3.9050

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -21.14    3079.95  -0.007    0.995
fdose0.01      17.52    3079.95   0.006    0.996
fdose0.1       21.43    3079.95   0.007    0.995
fdose0.5       22.17    3079.95   0.007    0.994

(Dispersion parameter for quasibinomial family taken to be 2.086026)

    Null deviance: 433.790  on 24  degrees of freedom
Residual deviance:  48.945  on 21  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 17
```

Hmm, the coefficients table looks weird here. The estimates and SE are all gigantic. What is going on?

Complete separation

Remember that our reference group right now in the model is the control group (i.e., dose of 0 mg/L). No fish in any of the tanks got tumors in the control group. What we are seeing is what's called *complete separation* (aka *perfect* separation), when the proportions in one group are all 0 or all 1.

Without getting into a lot of details, complete separation isn't generally considered a problem aside from the Wald tests in the summary output don't work right and confidence intervals need to be based on profiling the likelihood. As Wald tests for generalized linear models are considered questionable, anyway, this isn't usually a big deal. It mostly reminds us that while groups of all 0 or 1 may be very interesting practically, groups without variation are generally not interesting statistically. See http://stats.stackexchange.com/a/68917/29350 for a discussion of complete separation and some options when it is causing you problems.

When working with complicated binomial generalized linear models, you can diagnose what's called *quasi*-complete separation when the estimates and standard errors in the model summary output become very large (like estimates > 8 in absolute value). This also may or may not be a problem, but could indicate that you've fit an overly complicated model to relatively few data values.

Let's look at the summary of a model that doesn't have the control group as the reference group.

```
summary( glm(prop.tumor ~ relevel(fdose, ref = "0.01"),
             data = fish,
             family = quasibinomial,
             weights = num.fish) )
```

```
Call:
glm(formula = prop.tumor ~ relevel(fdose, ref = "0.01"), family = quasibinomial,
    data = fish, weights = num.fish)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.6623  -0.7647  -0.0002   0.4799   3.9050

Coefficients:
                                Estimate Std. Error t value Pr(>|t|)
(Intercept)                      -3.6217     0.6545  -5.534 1.72e-05 ***
relevel(fdose, ref = "0.01")0   -17.5206  3079.9499  -0.006    0.996
relevel(fdose, ref = "0.01")0.1   3.9068     0.6827   5.723 1.11e-05 ***
relevel(fdose, ref = "0.01")0.5   4.6528     0.6999   6.648 1.40e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasibinomial family taken to be 2.086026)

    Null deviance: 433.790  on 24  degrees of freedom
Residual deviance:  48.945  on 21  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 17
```

We can see we have more reasonable looking estimates now; the complete separation of the control group was muddling things. As the control group isn't part of our research question (it was to establish that the baseline tumor rate really was very low), we can ignore the complete separation in that group, remove it from the analysis, and do any comparisons of interest to answer our research questions.

I'll fit a new model without the control group.

```
fit2q = glm(prop.tumor ~ fdose,
          data = filter(fish, dose > 0),
          family = quasibinomial,
          weights = num.fish)
```

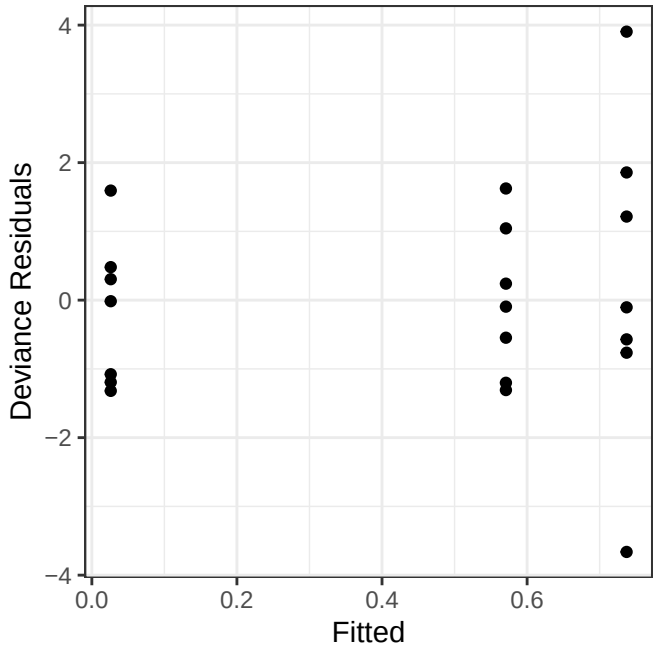
Since this model drops the control group, I need to get the estimated overdispersion to report in my model write-up.

```
summary(fit2q)$dispersion
```

```
[1] 2.433697
```

And a residual plot.

```
qplot(x = fitted(fit2q), y = resid(fit2q) ) +
    theme_bw() +
    labs(x = "Fitted",
         y = "Deviance Residuals")
```



Drop-in-deviance test

Let's do the overall test for `fdose` to see the evidence against the null hypothesis that all groups have the same estimated proportion. Based on our initial data exploration we really don't need this test, as higher doses are clearly different in proportion tumors from the low dose since the ranges don't overlap, but I wanted you to see how to calculate this.

Because we are working with a quasibinomial distribution, we will do this using an F test (for non "quasi" generalized linear models we use χ^2 tests). See the help page for `anova.glm` to see the description of what the output is and what the `test` argument does. The output is definitely not "an ANOVA".

```
?anova.glm
```

And here is the F test. This test is a test of the deviance, and should be referred to as either a drop-in-deviance test or a likelihood ratio test.

```
anova(fit2q, test = "F")
```

```
Analysis of Deviance Table

Model: quasibinomial, link: logit

Response: prop.tumor

Terms added sequentially (first to last)

      Df Deviance Resid. Df Resid. Dev      F    Pr(>F)
NULL                     20    306.718
fdose  2   257.77        18     48.945 52.959 2.879e-08 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Comparisons among groups

This research was set up to compare the 0.1 mg/L and 0.5 mg/L dose groups to the low dose group of 0.01 mg/L.

Profile likelihood confidence intervals from model

Because we have such simple results, we could pull coefficients and confidence intervals out of the summary output. These are profile likelihood confidence intervals, the gold standard.

Note the results are on the model scale, so are estimates involving log odds. You can exponentiate to odds via `exp()`.

```
coef(fit2q)
```

```
(Intercept)    fdose0.1    fdose0.5
  -3.621671    3.906772    4.652842
```

```
confint(fit2q)
```

```
Waiting for profiling to be done...
```

```
                2.5 %    97.5 %
(Intercept) -5.396108 -2.479803
fdose0.1     2.681097  5.718882
fdose0.5     3.381791  6.489438
```

Using emmeans for the comparisons

We can also use `emmeans()` from package **emmeans** for generalized linear models. This returns asymptotic (Wald) confidence intervals instead of the gold standard profile likelihood confidence intervals. I want you to see how **emmeans** works with *glm* objects so will proceed.

Below is the code for `emmeans()`, using the `trt.vs.ctrl` option to compare each group vs the reference level.

Since I fit a quasibinomial model, I want to base all tests and confidence intervals on the t distribution instead of the z distribution, which is standard for other types of generalized linear models. To do this I have to provide `emmeans()` the degrees of freedom of the model via the `df` argument.

```
emmeans(fit2q, trt.vs.ctrl ~ fdose,
        df = fit2q$df.residual)
```

```
$emmeans
 fdose emmean    SE df lower.CL upper.CL
 0.01  -3.622 0.707 18   -5.107   -2.136
 0.1    0.285 0.210 18   -0.155    0.726
 0.5    1.031 0.268 18    0.468    1.594

Degrees-of-freedom method: user-specified
Results are given on the logit (not the response) scale.
Confidence level used: 0.95

$contrasts
 contrast   estimate    SE df t.ratio p.value
 0.1 - 0.01     3.91 0.737 18   5.298  0.0001
 0.5 - 0.01     4.65 0.756 18   6.155  <.0001

Degrees-of-freedom method: user-specified
Results are given on the log odds ratio (not the response) scale.
P value adjustment: dunnettx method for 2 tests
```

Exponentiating back to odds ratios for interpretation

These results I've made so far are on the *logit scale* (the scale of the model), so are expressed as log odds. To get the estimated values back to proportions and the estimated differences as odds ratios we can use the `type` argument to choose `"response"` instead of the default `"link"`. Note you cannot express results as differences in proportions when working with binomial generalized linear models with a logit link, only as odds ratios.

We could do this by hand with `plogis()` (inverse of the logit) and `exp()` (inverse of the log) to get to estimated proportions and estimated odds, respectively. It is simply convenient to use the built-in `emmeans()` option.

Remember that standard errors cannot be transformed, so we can't use an inverse link directly on the standard errors. You'll see that `emmeans()` reports standard errors for the original values, which are calculated via the delta method. Since standard errors are really only a good description of a symmetric distribution, it doesn't make sense to report these for either odds ratios or proportions. Stick with confidence interval limits, which are made on the model scale and then transformed to the scale of interest.

```
( emm_fit2q = emmeans(fit2q, trt.vs.ctrl ~ fdose,
                      df = fit2q$df.residual,
                      type = "response") )
```

```
$emmeans
 fdose  prob     SE df lower.CL upper.CL
 0.01  0.026 0.0179 18  0.00602    0.106
 0.1   0.571 0.0514 18  0.46124    0.674
 0.5   0.737 0.0519 18  0.61499    0.831

Degrees-of-freedom method: user-specified
Confidence level used: 0.95
Intervals are back-transformed from the logit scale

$contrasts
 contrast   odds.ratio   SE df null t.ratio p.value
 0.1 / 0.01       49.7 36.7 18    1   5.298  0.0001
 0.5 / 0.01      104.9 79.3 18    1   6.155  <.0001

Degrees-of-freedom method: user-specified
P value adjustment: dunnettx method for 2 tests
Tests are performed on the log odds ratio scale
```

As we are only doing two comparisons today and have two degrees of freedom for testing, we can justify not adjusting for multiple comparisons. I will set `adjust = "none"` at the same time I get confidence intervals via `summary()`.

These are the results I will plot, so I put these into a data.frame.

```
( comp_fit2q = summary(emm_fit2q$contrasts, infer = TRUE, adjust = "none") )
```

```
 contrast   odds.ratio   SE df lower.CL upper.CL null t.ratio p.value
 0.1 / 0.01       49.7 36.7 18     10.6      234    1   5.298  <.0001
 0.5 / 0.01      104.9 79.3 18     21.4      513    1   6.155  <.0001

Degrees-of-freedom method: user-specified
Confidence level used: 0.95
Intervals are back-transformed from the log odds ratio scale
Tests are performed on the log odds ratio scale
```

Consecutive comparisons

One option that **emmeans** offers that we haven't talked about yet this quarter is *consecutive* comparisons via `consec`. This is for situations when we want to compare only consecutive groups. The order of the factor becomes very important if doing consecutive comparisons.

Here is code to demonstrate, even though this is not of interest for this problem. Note that we could always write out the comparisons with custom contrasts, as well, but `consec` can be convenient.

```
emmeans(fit2q, consec ~ fdose, type = "response")
```

```
$emmeans
 fdose  prob     SE  df asymp.LCL asymp.UCL
 0.01  0.026 0.0179 Inf   0.00664    0.0966
 0.1   0.571 0.0514 Inf   0.46859    0.6673
 0.5   0.737 0.0519 Inf   0.62389    0.8258

Confidence level used: 0.95
Intervals are back-transformed from the logit scale

$contrasts
 contrast   odds.ratio     SE  df null z.ratio p.value
 0.1 / 0.01      49.74 36.675 Inf    1   5.298  <.0001
 0.5 / 0.1        2.11  0.717 Inf    1   2.193  0.0554

P value adjustment: mvt method for 2 tests
Tests are performed on the log odds ratio scale
```

Reporting estimated proportions for each group

When working with proportions, which means our statistical results are about odds ratios, it can be useful to also report the estimated proportions for each group on the data scale. This can help your audience understand what the ratios of odds mean practically. I could see adding such information onto a results plot, for example.

This information has already been calculated by `emmeans()`, so we can pull the info out of `emm_fit2q`. Again make note that these standard errors don't make sense to report for proportions, so report confidence intervals.

```
( prop_fit2q = summary(emm_fit2q$emmeans) )
```

```
 fdose  prob     SE df lower.CL upper.CL
 0.01  0.026 0.0179 18  0.00602    0.106
 0.1   0.571 0.0514 18  0.46124    0.674
 0.5   0.737 0.0519 18  0.61499    0.831

Degrees-of-freedom method: user-specified
Confidence level used: 0.95
Intervals are back-transformed from the logit scale
```

Wrapping up the analysis

Plotting the results

Now we can plot the results as we normally would. This looks essentially like what we did all the way back in lab 2. The results are multiplicative, however, and the biologically important increase in odds that we wanted to detect was a ten-fold increase. The statistical value of no difference for odds ratios is 1, which we can include for reference if we want. This depends a bit on the scale of the plot.

Let's make sure we have a variable for the x axis. These multiplicative differences are odds ratios, so I try to indicate the order of division in the x axis labels using the word "over".

```
comp_fit2q$contrast = c("0.1 over 0.01 mg/L dose",
                        "0.5 over 0.01 mg/L dose")
```

And now we can make the standard plot the results.

```
( g1 = ggplot(comp_fit2q, aes(x = contrast, y = odds.ratio) ) + # Set axes for graph
       geom_errorbar(width = .1, lwd = .75, aes(ymin = lower.CL, ymax = upper.CL) ) + # Add error bar
       geom_point(size = 3) + # Add points for means
       labs(x = NULL,
            y = "Odds ratio") + # Change x and y labels
```

```
      theme_bw(base_size = 18) + # b&w and increase text size
      theme(panel.grid.major.x = element_blank(),
            panel.grid.minor = element_blank() ) + # remove gridlines
      geom_hline(yintercept = 1, colour = "grey34", lty = 2) + # horizontal line at 1 for reference
      geom_rect(xmin = -Inf, xmax = Inf, ymin = -Inf, ymax  = 10, alpha = .1) +
      # difference at 10 indicates no practical difference
      scale_y_continuous(limits = c(1, NA),
                         breaks = c(10, 100, 200, 300, 400, 500),
                         # Set breaks to show bio important value
                         # Start y axis at exactly 1 via expansion()
                         expand = expansion(mult = c(0, .05) ) ) )
```

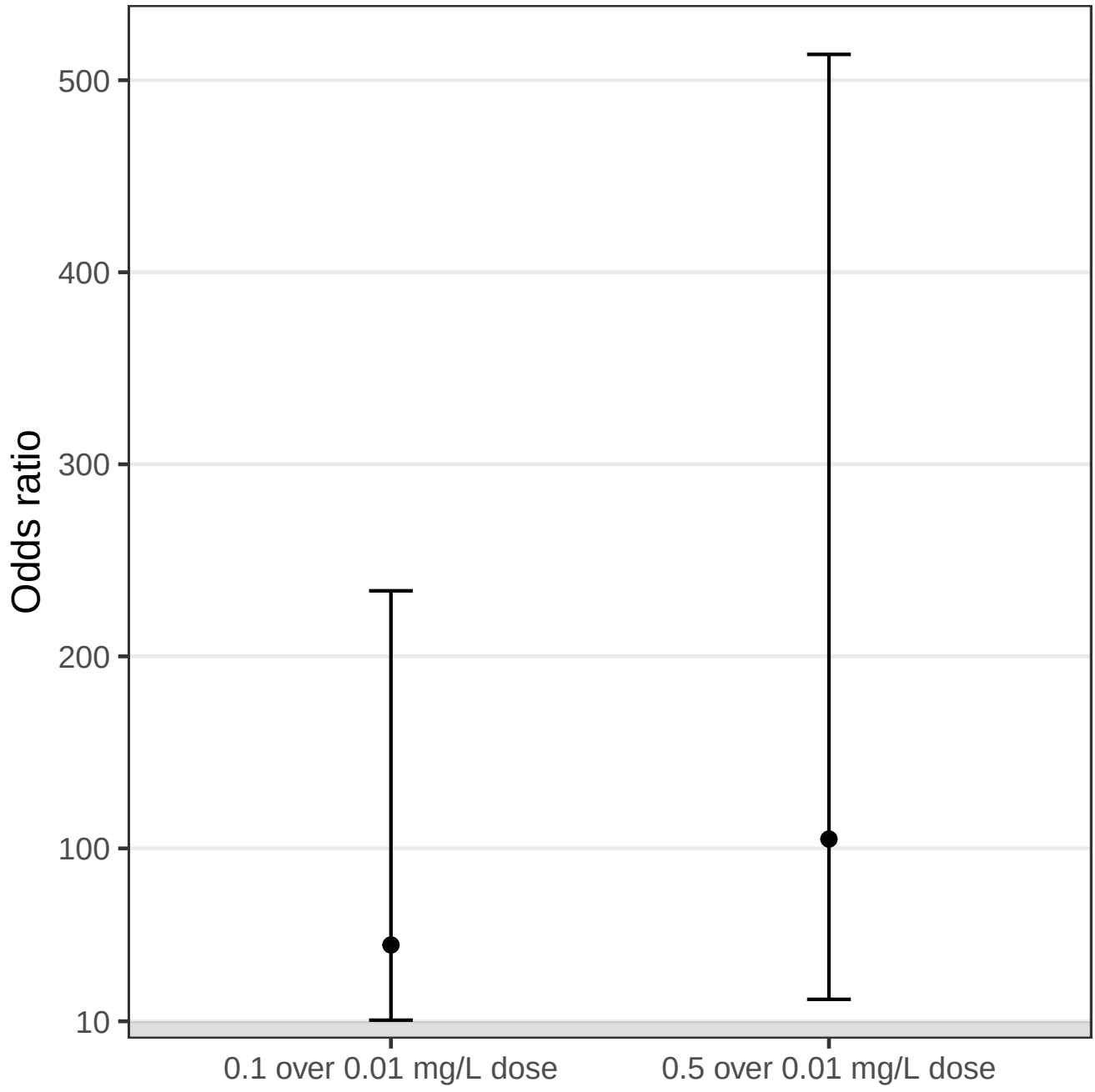


Table of estimated proportions or percentages

Including the table of estimated proportions can be useful for the reader, as it helps them interpret the odds ratios. Remember to only include it in your write-up if it's an integral part of the report and you refer to it in the text.

```
prop_fit2q %>%
    select(-SE, -df) %>%
    mutate_if(is.numeric, list(~round(., 2)) ) %>%
    setNames(c("Dose", "Proportion", "Lower", "Upper") )
```

Dose	Proportion	Lower	Upper
0.01	0.03	0.01	0.11
0.1	0.57	0.46	0.67
0.5	0.74	0.61	0.83

Showing these as percentages is another options

```
prop_fit2q %>%
    select(-SE, -df) %>%
    mutate_if(is.numeric, list(~round(., 3)*100) ) %>%
    setNames(c("Dose", "Percent", "Lower", "Upper") )
```

Dose	Percent	Lower	Upper
0.01	2.6	0.6	10.6
0.1	57.1	46.1	67.4
0.5	73.7	61.5	83.1

If we want percentage symbols, we can add them via `paste0`.

```
prop_fit2q %>%
    select(-SE, -df) %>%
    mutate_if(is.numeric, list(~paste0(round(., 3)*100, "%")) ) %>%
    setNames(c("Dose", "Percent", "Lower", "Upper") )
```

Dose	Percent	Lower	Upper
0.01	2.6%	0.6%	10.6%
0.1	57.1%	46.1%	67.4%
0.5	73.7%	61.5%	83.1%

R tidbit

The forcats package for working with factors

We've learned the basics of working with factors over the last few weeks. Package **forcats** is a package that is filled with convenience functions to make working with factors easier. There are functions to, e.g., help reorder, relabel, relevel, and lump the levels of a factor.

One of the functions I use a lot from this package is `fct_inorder()`, for setting the order of the levels in the order they appear rather than alphanumerically.

To show you how this works, let's say we had a vector of month-day dates, in order by time.

```
vec_dates = c("Jan 2", "Jan 2", "Feb 2", "Feb 2", "Mar 12", "Apr 10")
```

As you know, if we factor this via `factor()`, by default the levels are going to be ordered alphabetically. So "April" will come first, followed by "February".

```
factor(vec_dates)
```

```
[1] Jan 2  Jan 2  Feb 2  Feb 2  Mar 12 Apr 10
Levels: Apr 10 Feb 2 Jan 2 Mar 12
```

We can use `fct_inorder()` from **forcats**, instead, to conveniently get the levels in the order they are given in the vector. This is convenient when the categories of a variable are already in the order we want them to be when we read our data in.

I use the `forcats::` coding instead of loading the package, but I could have loaded the package first instead.

```
forcats::fct_inorder(vec_dates)
```

```
[1] Jan 2  Jan 2  Feb 2  Feb 2  Mar 12 Apr 10
Levels: Jan 2 Feb 2 Mar 12 Apr 10
```

Of course this work can be done in base R, as well. It involves passing the `unique()` values of the vector to `levels`. The **forcats** package contains functions to make such tasks easier.

```
factor( vec_dates,
       levels = unique(vec_dates) )
```

```
[1] Jan 2  Jan 2  Feb 2  Feb 2  Mar 12 Apr 10
Levels: Jan 2 Feb 2 Mar 12 Apr 10
```