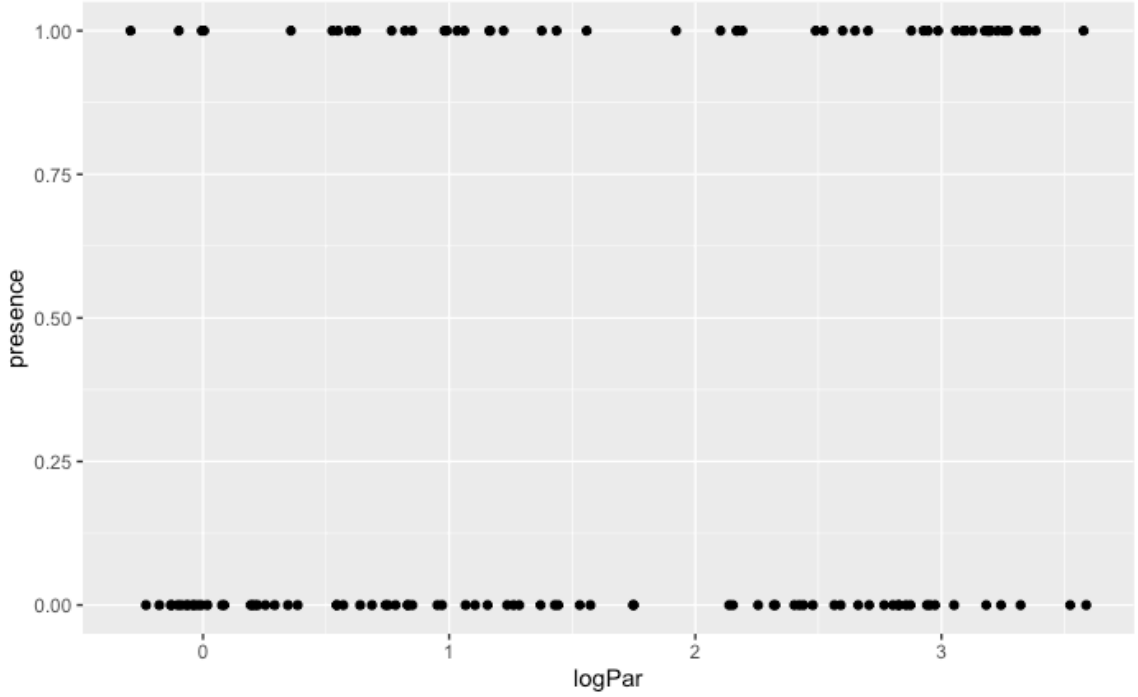
Lecture 7. Binomial GLMs

Binomial GLMs

In addition to the normal distribution and the Poisson or negative binomial, the other distribution very common in biology is the binomial. As we learned in lecture 2, the binomial distribution has two parameters, n = number of trials, and p = probability of success. When $n = 1$, the data are *binary*, i.e. can be coded as 0 or 1. This could be presence/absence of a species in sample, alive/dead, infected/uninfected, red/blue, happy/sad, etc. When $n > 1$, the binomial distribution is used for modeling proportional data of the form 'X out of a total Y', e.g. 3 survivors out of 10 initially present, 6 infected out of a total population of 25, etc. Let's start off with the binary case first.

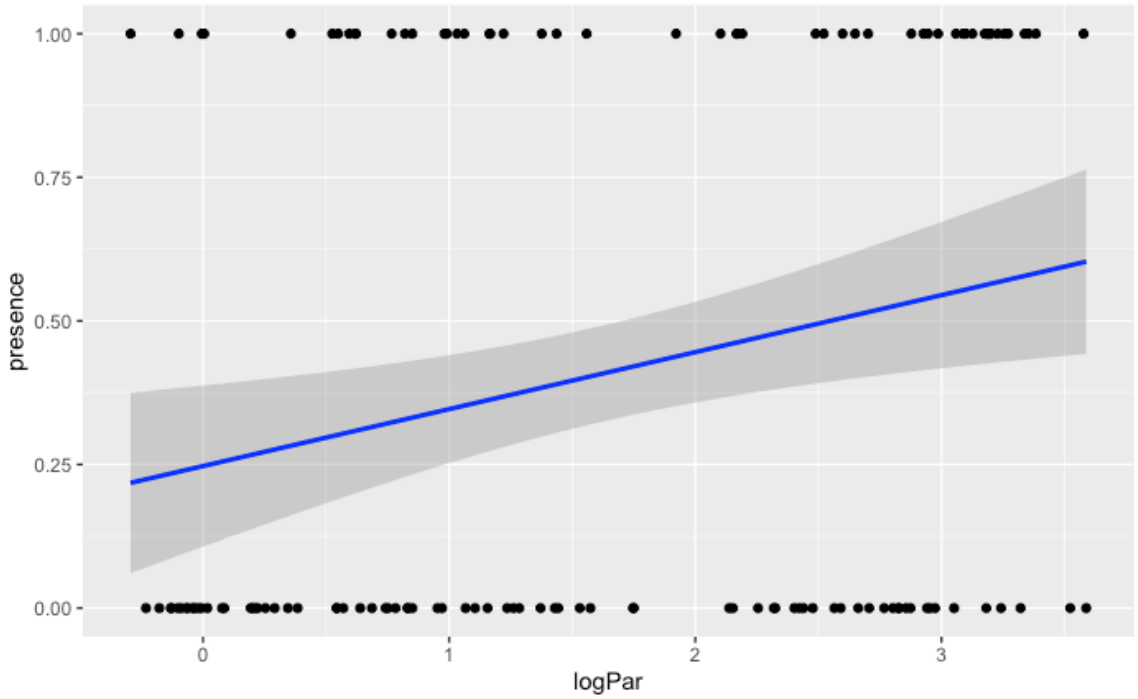
Working with binary data

Here's some data from a time series of the phytoplankton community in the western English Channel (http://www.westernchannelobservatory.org.uk/). The microplankton community is counted every 1-2 weeks. I've done some work looking at how different phytoplankton species respond to seasonal variation in light and nitrogen. Let's look at the presence/absence of a single species in the counts, compared to the irradiance entering the ocean. A scatter plot of the relationship looks like this:



The presence/absence data is coded as 1 for presence, 0 for absence. 'log PAR' is the log of photosynthetically active radiation. Looking at this data, it's immediately

clear that working with binary data is going to be more challenging than working with count data, or normally distributed data, where it's much easier to visualize the effects of continuous predictors. And beyond visualizing relationships, we want to fit some kind of regression to this data. If we fit a simple linear regression, it looks like this:

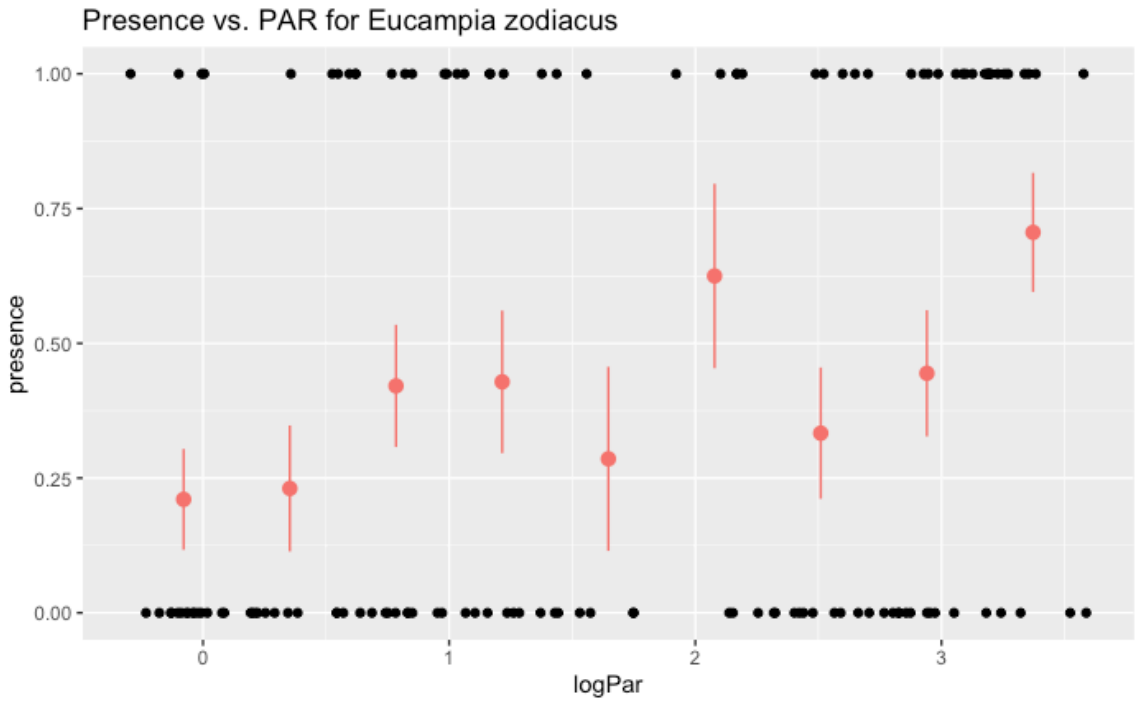


There are some immediate problems with this approach. The fitted line goes through the vacant space between the data, which can only be 0 or 1, so what does it mean that the value of the fitted relationship is ~0.4 when log PAR = 2. In addition, the fitted line will quickly go below 0 and above 1, which is even harder to interpret.

Let's start with just visualizing the relationship in a better way. For this kind of data, it is useful to bin the data on the x-axis, and then take the mean of the bins:

```
datause$cut.par = cut_interval(datause$logPar, 9)
par.bins = datause %>% dplyr::select(logPar, presence, cut.par) %>% gro
up_by(cut.par) %>%
  summarise(mean.prob = mean(presence),
  se.prob = sqrt((mean(presence)*(1-mean(presence)))/length(presence)))
par.bins = separate(par.bins, cut.par, sep = ",", into = c('first', 'se
cond'), remove = FALSE) %>% mutate(bin.centers = mean(c(parse_number(fi
rst), parse_number(second))))

ggplot() + geom_point(data = datause, aes(logPar, presence)) +
  geom_point(data = par.bins, aes(bin.centers, mean.prob, col = 'red'),
 show.legend = FALSE) +
  geom_pointrange(data = par.bins, aes(x = bin.centers, y = mean.prob,
```

```
ymin = mean.prob-se.prob, ymax = mean.prob+se.prob, col = 'red'), show.
legend = FALSE) +
  labs(title = 'Presence vs. PAR for Eucampia zodiacus')
```



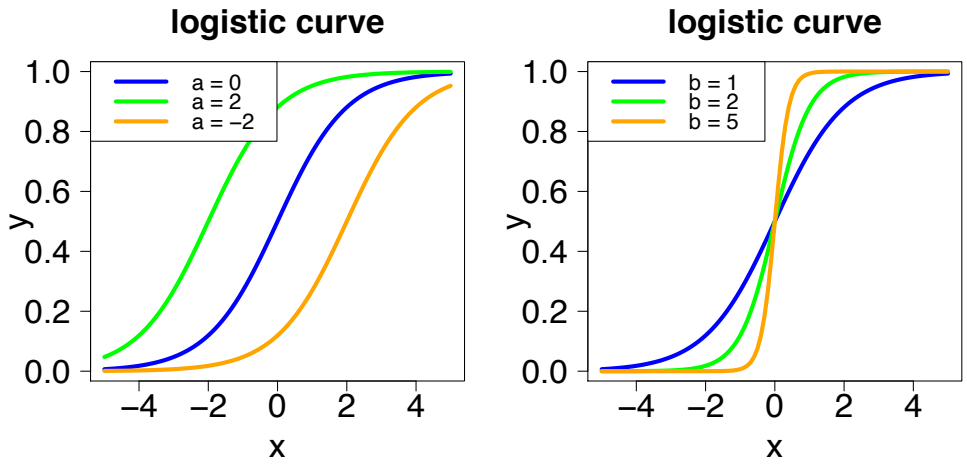
This code 1) makes 8 bins along the x-axis, 2) calculates the proportion of presence in each bin with mean(), 3) calculates the standard error of the proportion in each bin, 4) plots the binned results as well as the raw data. The formula for the standard error of a proportion is $\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}$, where $\bar{p}$ is the observed proportion and n is the number of samples used to calculate that proportion.

The logistic function, and the logit link

From the plot above we can get a better sense of the relationship, and it does look like the probability of presence increases as PAR increases. How should we model the relationship? The trick is that we are going to model the underlying *probability* that the data is 0 vs. 1. A probability ranges from 0 to 1, so we need to model the probability with a function that has an unrestricted range on the x-axis, but varies between 0 and 1 on the y-axis. We already learned such a function, the *logistic* curve:

$$y = \frac{\exp(a + bx)}{1 + \exp(a + bx)}$$

You've explored this curve in a previous homework, it has a sigmoid shape:



a acts like an intercept, moving the curve to the left or the right, and *b* acts like a slope, steepening or shallowing the rise of the curve. To use the logistic curve for a generalized linear model, we can put whatever linear model terms we want to model inside the exp() function:

$$\mu_i = \frac{\exp(\beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots)}{1 + \exp(\beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots)}$$

$$Y_i \sim \text{Binomial}(p = \mu_i, n = 1)$$

Here Y_i is the data, which is a random draw from a binomial distribution. Because we are looking at binary data right now, the binomial parameter $n = 1$, which means the data will be 0 or 1 (absence or presence in the phytoplankton example). The parameter p, the probability of 'success', is equal to μ_i, and μ_i is a logistic function of the linear model terms. It is a bit easier to look at in terms of the *link function*:

$$\text{logit}(\mu_i) = \beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots$$

$$Y_i \sim \text{Binomial}(p = \mu_i, n = 1)$$

The inverse of the logistic function is called the *logit* function. Those names are a little confusing, but make sure to distinguish them. The logit function is $\text{logit}(\mu_i) = \log\left(\frac{\mu_i}{1-\mu_i}\right)$. This model is the default for binomial GLM, and is often called logistic regression.

Let's compare with the Poisson GLM to help clarify what's going on here. For the Poisson, the deterministic part of the model is an exponential function

$$\mu_i = \exp(\beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots)$$

which is the same as having a log link function:

$$\log(\mu_i) = \beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots$$

For the binomial, the deterministic part of the model is a logistic function

$$\mu_i = \frac{\exp(\beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots)}{1 + \exp(\beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots)}$$

which is the same as having a logit link function

$$\text{logit}(\mu_i) = \beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots$$

For the Poisson GLM the exponential curve ranges from 0 to infinity, which appropriate for modeling the mean of count data. For the binomial GLM the logistic curve ranges from 0 to 1, which is appropriate for modeling the probability of success.

How can we interpret the coefficients of the logistic model? The sigmoid shape of the logistic curve makes it challenging. It's easies to think about it on the logit scale. The logit function is $\log\left(\frac{\mu_i}{1-\mu_i}\right)$. We can rewrite this as

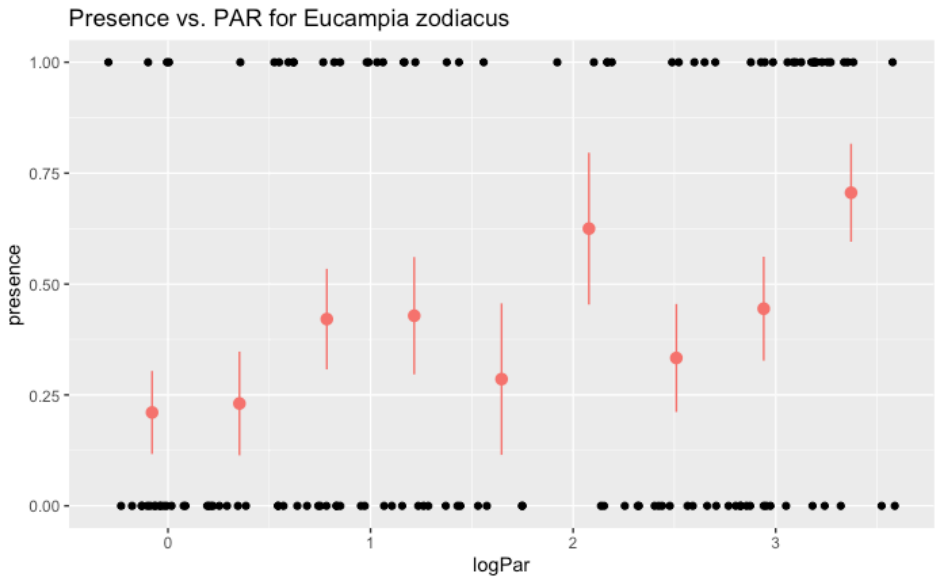
$$\text{logit}(\mu_i) = \log\left(\frac{P(Y_i = 1)}{P(Y_i = 0)}\right)$$

because μ_i is the probability that Y_i is 1, and (1-μ_i) is the probability that Y_i is 0. The quotient $\frac{P(Y_i=1)}{P(Y_i=0)}$ is the *odds* that Y_i is 1. This is just like odds in gambling; if the probability that $Y_i = 1$ is 0.75, and the probability that $Y_i = 0$ is 0.25, then the odds that $Y_i = 1$ is 3 (i.e. 3-to-1).

So we can think of the logit function as the *log-odds*. And then we can interpret the coefficients of the model accordingly. E.g., the meaning of the coefficient β_1 is "If the predictor X_1 increases by one unit, then the log-odds increases by the amount β_1).

Binomial GLM

To make this more concrete, I'll model the phytoplankton data shown earlier, which looks like this:



.

We use the glm() function just like for a Poisson model, but now we set family = binomial.

```
#binomial GLM
mod = glm(presence ~ logPar, data = datause, family = binomial)
```

We just want to model whether presence increases with logPar and we use the standard model formula syntax. The default link function for family=binomial is the logit link, so we don't need to specify the link function, but we could also write this as family=binomial(link = 'logit'). To see more info on the link functions for different probability distributions you can do ?family. There are other link functions that have a sigmoid shape like the logistic, but differ in the details of the shape. These include the probit (link = "probit") and the complementary log-log (link = "cloglog"). I won't review them here because the logistic is very common in biology. Let's look at the model output:

```
summary(mod)
```

```
##
## Call:
## glm(formula = presence ~ logPar, family = binomial, data = datause)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.369  -0.984  -0.764   1.143   1.714
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.083      0.327   -3.32  0.00091 ***
## logPar         0.424      0.156    2.71  0.00665 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 175.76  on 129  degrees of freedom
## Residual deviance: 168.03  on 128  degrees of freedom
## AIC: 172
##
## Number of Fisher Scoring iterations: 4
```

The coefficients are an Intercept and a slope parameter logPar. So this model is fitting the following logistic curve:

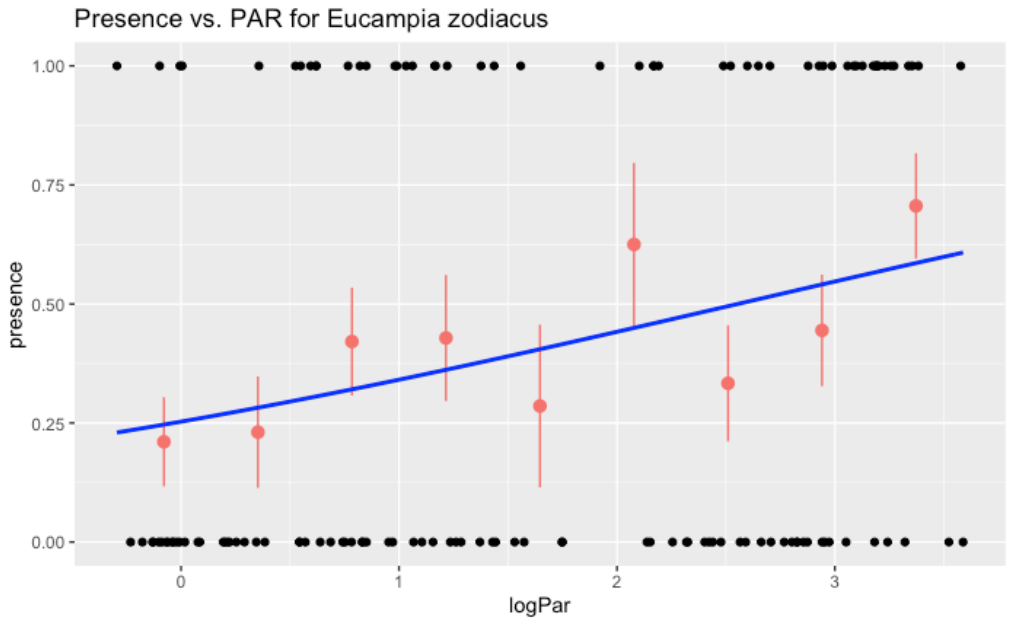
$$\mu_i = \frac{\exp(\text{Intercept} + \text{logPar} * x)}{1 + \exp(\text{Intercept} + \text{logPar} * x)}$$

$$\text{presence} \sim \text{Binomial}(p = \mu_i, n = 1)$$

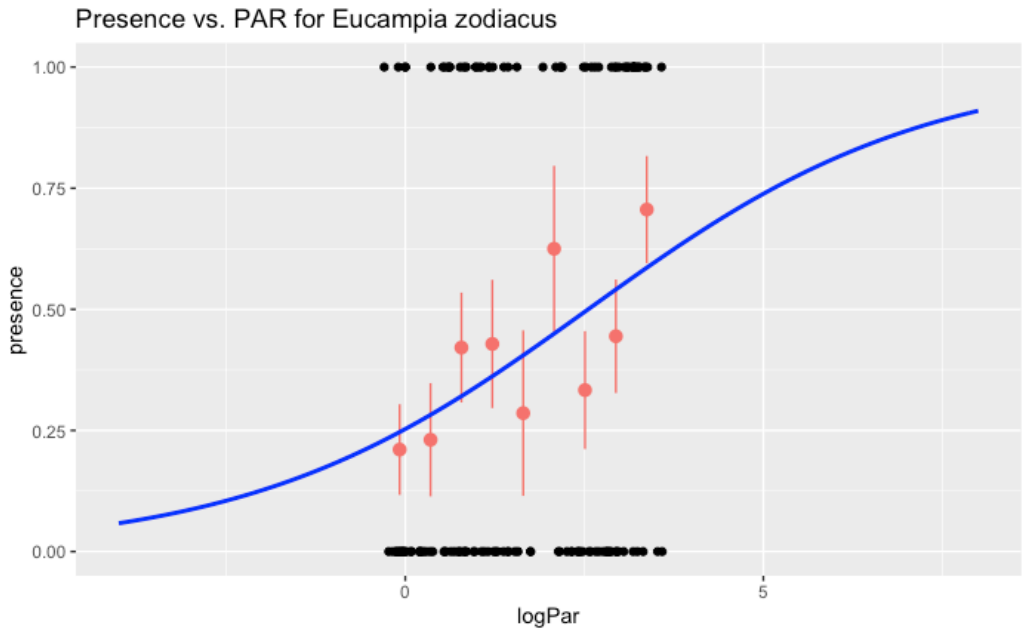
where x is the dependent variable, logPar [yes, it is confusing that the coefficient for logPar is also named logPar].

We can visualize the fitted relationship using curve(). First I'll define a logistic function, which will make the code clearer. Then I'll use it to plot a curve on top of the raw data I plotted earlier.

```
#function for logistic curve
logistic = function(x) exp(x)/(1+exp(x))
#plot the fitted curve
ggplot() + geom_point(data = datause, aes(logPar, presence)) +
  geom_point(data = par.bins, aes(bin.centers, mean.prob, col = 'red'),
 show.legend = FALSE) +
  geom_pointrange(data = par.bins, aes(x = bin.centers, y = mean.prob,
ymin = mean.prob-se.prob, ymax = mean.prob+se.prob, col = 'red'), show.
legend = FALSE) +
  labs(title = 'Presence vs. PAR for Eucampia zodiacus') +
  geom_function(fun = ~logistic(coef(mod)[1] + coef(mod)[2]*.x), col =
'blue', size = 1)
```



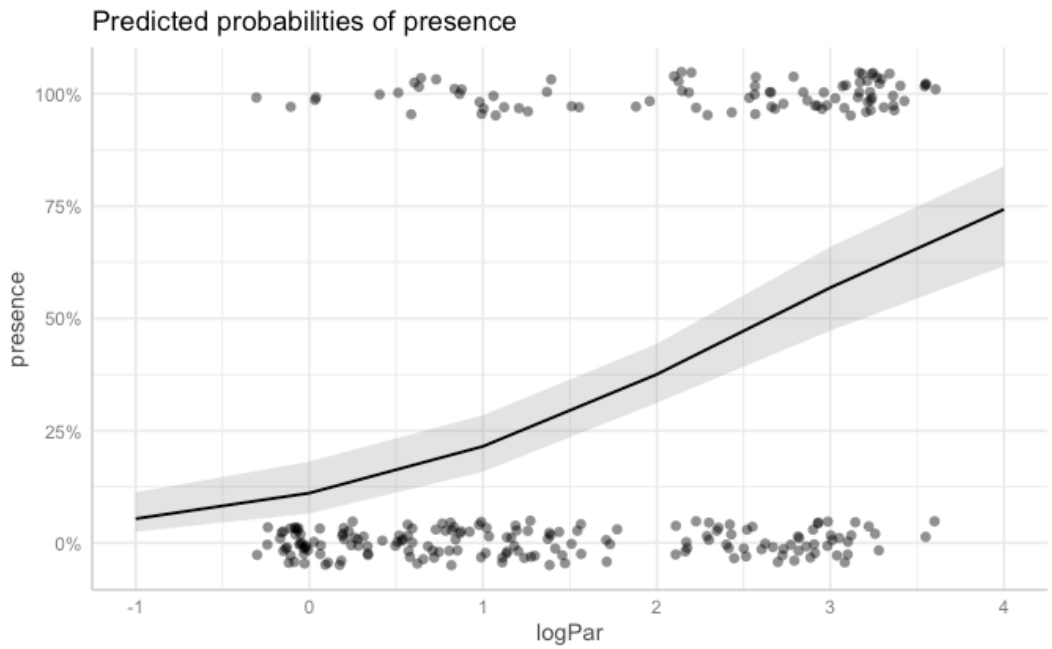
So as the binned data suggest, the model says that the probability of presence increases as logPar increases. We could also see this from the coefficient estimates from summary(), because the parameter 'logPar' is positive. In this case the fitted curve looks nearly linear. That is consistent with the binned data, where the proportion of presence does not get all the way to 0 or 1 over this range of the predictor. If we zoom out on the x-axis, we can see the sigmoid shape of the logistic curve better:



So one thing we learn from this example is that for intermediate probabilities the logistic curve is nearly linear, but it can also account for saturating effects as probabilites get close to zero or one.

Just like for the Poisson GLM, we can also use an effects plot with the fitted model to visualize the relationship:

```
ggpredict(mod), add.data = TRUE, jitter = 0.05)
```



The ggeffects plotting function has a built in jittering function that you can adjust – that's why the 0 and 100% values do not lie on a line.

We can also test the slope coefficient with a likelihood ratio test, e.g. using Anova():

```
Anova(mod)

## Analysis of Deviance Table (Type II tests)
##
## Response: presence
##        LR Chisq Df Pr(>Chisq)
## logPar     7.73  1     0.0054 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The effect is highly significant.

Binomial diagnostics

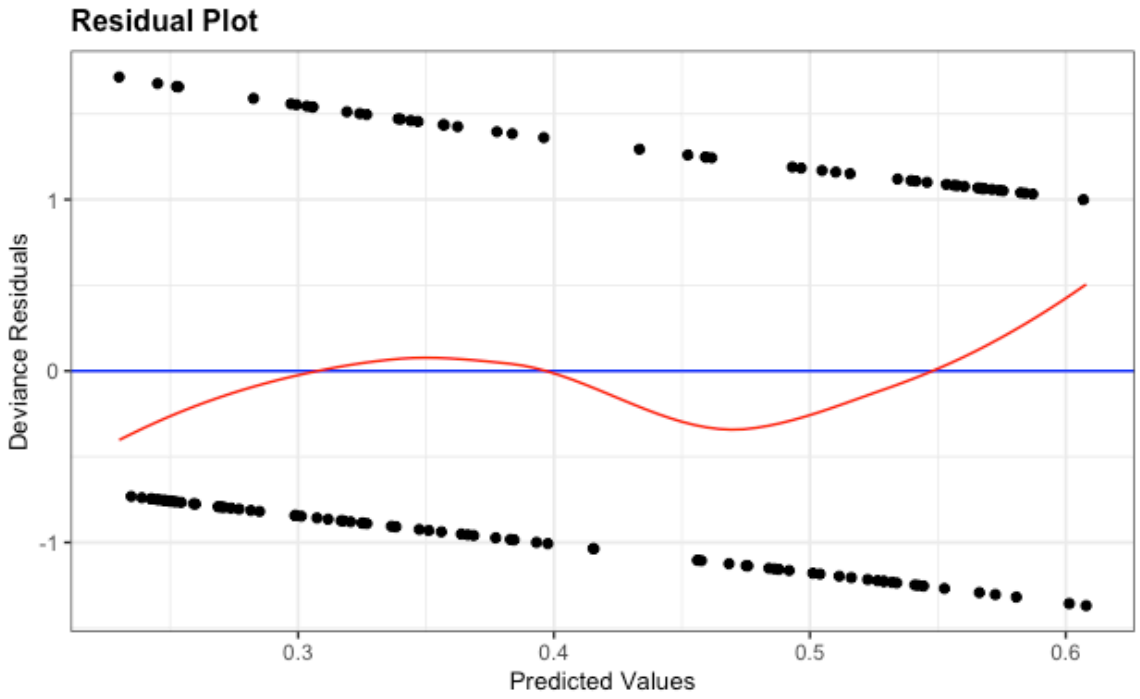
As we saw with Poisson GLMs, residual diagnostics are more challenging to interpret when the data is not normally distributed. This is especially true for binary data. For normally distributed data, we can add the fitted values to the residuals to get the raw data, i.e.

$$Y_i = \beta_0 + \beta_1 * X_i + \varepsilon_i = \mu_i + \varepsilon_i$$

For binary data, the fitted values μ_i are the *probability* that $Y_i = 1$. The data can only be 0 or 1, while the fitted values range continuously from 0 to 1. Does it even make sense to define a 'residual' in this case, where the data can never be equal to the fitted values? This issue actually applies to the Poisson distribution as well, because the fitted values are the mean of the Poisson, and the mean of the Poisson ranges from 0 to infinity, while the actual draws from the Poisson distribution are discrete values (counts).

We can still define deviance residuals and Pearson residuals in the same way as for a Poisson or negative binomial distribution. Let's see what they look like:

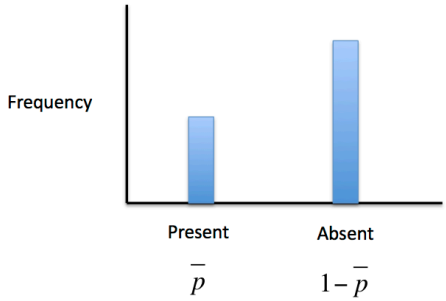
```
resid_panel(mod, plots = 'resid', smoother = TRUE)
```



The plot of deviance residuals vs. predicted values (i.e. fitted values) shows strong patterning, and this is because the data only has two possible values: 0 or 1. Nonetheless, this kind of plot is still somewhat useful, because the red line is a smoother showing the mean of the residuals as a function of the predicted values. If this mean shows strong pattern then that indicates a problem with the model, e.g. nonlinearity that is not accounted for, or an inappropriate link function. In this case

there is no strong pattern so the model assumptions are OK. In general, for binary models (and binomial models in general) it is useful to plot residuals (deviance or Pearson) vs. the predicted values and vs. the individual predictors, to make sure there is no strong patterning.

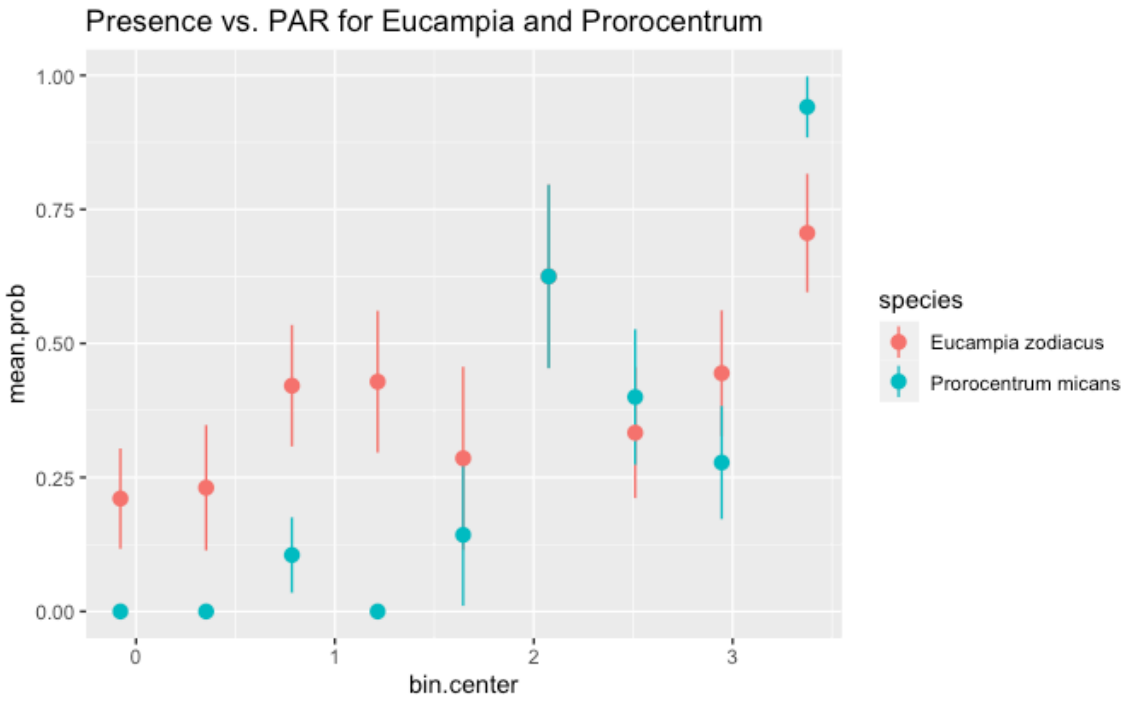
The good news is that for binary data, *overdispersion is not possible*. This is because variance of the data is not independent of the mean. For example, let's say we were modeling how the presence of *Eucampia zodiacus* varied between months. In March the mean of the data, i.e. the proportion of presences, is $\bar{p} = \frac{\sum Y}{N}$, where *Y* is the data (0 or 1) and *N* is the number of samples taken in March. It can be shown with some algebra that the variance of the presence data in March is then equal to $\bar{p}(1-\bar{p})$. So the variance is entirely defined by the mean of the data, and is always equal to this number. Another way of thinking about this is that if the data has some mean presence $\bar{p}$, then we know the mean absence is $1-\bar{p}$:



For binary data the distribution only has these two categories, and so the variability in the data is fully determined by the mean $\bar{p}$. This will not be the case for binomial data with $n > 1$, which we will get to shortly.

Let's do another binary example, using the same English channel time series but now comparing two phytoplankton species, the diatom *Eucampia zodiacus* and the dinoflagellate *Prorocentrum micans*. I'll plot the binned data for each species:

```
datause$cut.par = cut_interval(datause$logPar, 9)
par.bins = datause %>% dplyr::select(logPar, presence, cut.par, specie
s) %>% group_by(cut.par, species) %>%
  summarise(mean.prob = mean(presence),
  se.prob = binomial.SE(presence))

par.bins = separate(par.bins, cut.par, sep = ",", into = c('first', 'se
cond'), remove = FALSE) %>% mutate(bin.center = mean(c(parse_number(fir
st), parse_number(second))))

ggplot(par.bins, aes(bin.center, mean.prob, col = species)) + geom_poin
t() + geom_pointrange(aes(x = bin.center, y = mean.prob, ymin = mean.pr
ob-se.prob, ymax = mean.prob+se.prob)) + labs(title = 'Presence vs. PAR
 for Eucampia and Prorocentrum')
```



It looks like *P. micans* is less likely to be present at low irradiance, but at high irradiance it catches up and is even a little more common at the highest irradiance.

I'm interested in how species respond differentially to environmental variation. To quantify this, we can fit a GLM where the slope vs. logPar differs between the two species:

```
mod = glm(presence ~ logPar*species, data = datause, family = binomial)
summary(mod)

##
## Call:
## glm(formula = presence ~ logPar * species, family = binomial,
##     data = datause)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.514  -0.906  -0.294   1.043   2.498
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                        -1.083      0.327   -3.32  0.00091 ***
## logPar                              0.424      0.156    2.71  0.00665 **
## speciesProrocentrum micans         -3.351      0.872   -3.84  0.00012 ***
## logPar:speciesProrocentrum micans   1.212      0.347    3.49  0.00048 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 332.81  on 259  degrees of freedom
## Residual deviance: 266.68  on 256  degrees of freedom
## AIC: 274.7
##
## Number of Fisher Scoring iterations: 6
```

The same model syntax as the last example, now with an interaction between logPar and species. It looks like the interaction is significant, but the best way to test this is with a likelihood ratio test:

```
Anova(mod)

## Analysis of Deviance Table (Type II tests)
##
## Response: presence
##                LR Chisq Df Pr(>Chisq)
## logPar             44.9  1    2.1e-11 ***
## species             6.7  1     0.0096 **
## logPar:species     15.7  1    7.6e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Two ways to do likelihood ratio tests

Let's review what the Anova() function does when you give it a model with an interaction. For the interaction, it compares the full model to a restricted model that has the interaction removed, and computes the likelihood ratio test. For the main effects (e.g. 'logPar'), it first removes the interaction, and then compares the 'full' model with logPar to a restricted model that removes logPar. In other words, Anova() uses the model you give it to test the interaction, and it test the main effects assuming the interaction does not exist, because it doesn't really make sense to try and remove 'logPar' while leaving 'logPar:species' in the model.

The Anova() function is convenient because it automatically creates appropriate full and restricted models, and then compares them with a likelihood ratio test. But to make sure you understand exactly what is being done, you could also create the full and restricted models yourself, and then compare them using the lowercase anova() function:

```
model.all = glm(presence ~ logPar*species, data = datause, family =
binomial)
```

```
model.no.interaction = glm(presence ~ logPar + species, data = datause,
family = binomial)
```

```
model.no.logPar = glm(presence ~ species, data = datause, family =
binomial)
```

```
model.no.species = glm(presence ~ logPar, data = datause, family =
binomial)
```

I've made four models: model.all is the model with the interaction, and model.no.interaction removes the interaction. We can then compare these models by giving both to anova(), and telling it to do a Chi-square test:

```
anova(model.all, model.no.interaction, test = 'Chisq')
```

```
## Analysis of Deviance Table
##
## Model 1: presence ~ logPar * species
## Model 2: presence ~ logPar + species
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       256        267
## 2       257        282 -1    -15.7  7.6e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This has the same chi-square probability as the test for logPar:species reported by Anova(). When a model has a significant interaction, that is typically the result you want to focus on, because it means the effect of one predictor depends on the value of the other predictor. But you still may be interested in saying 'Is there a significant mean effect of logPar, averaging over the two species?'. You can test this by comparing the model with no interaction (because in that model logPar does not depend on species) to a model with no logPar term:

```
anova(model.no.interaction, model.no.logPar, test = 'Chisq')
```

```
## Analysis of Deviance Table
##
## Model 1: presence ~ logPar + species
## Model 2: presence ~ species
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       257        282
## 2       258        327 -1    -44.9  2.1e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This has the same chi-square probability as the Anova() result for 'logPar', because this is the same comparison that Anova() is making. Finally, you can ask 'Is there a significant overall difference between the species, averaging over values of logPar?'. You can test this by comparing the model with no interaction (because in that model the species effect does not depend on logPar) to a model with no species term:

```
anova(model.no.interaction, model.no.species, test = 'Chisq')
```

```
## Analysis of Deviance Table
##
```

```
## Model 1: presence ~ logPar + species
## Model 2: presence ~ logPar
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       257        282
## 2       258        289 -1     -6.7   0.0096 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This has the same chi-square probability as the Anova() result for 'species'. So hopefully it is clear that the Anova() function is making all these comparisons by creating the appropriate full and restricted models behind the scenes. One more note: if you wanted to compare a model with a single term, such as logPar, to a model with no terms, the model syntax for the restricted model is glm(presence ~ 1, family = binomial…). Putting '1' on the right hand side of a formula means 'just fit one parameter, the mean of the data'. The single parameter will be called Intercept.

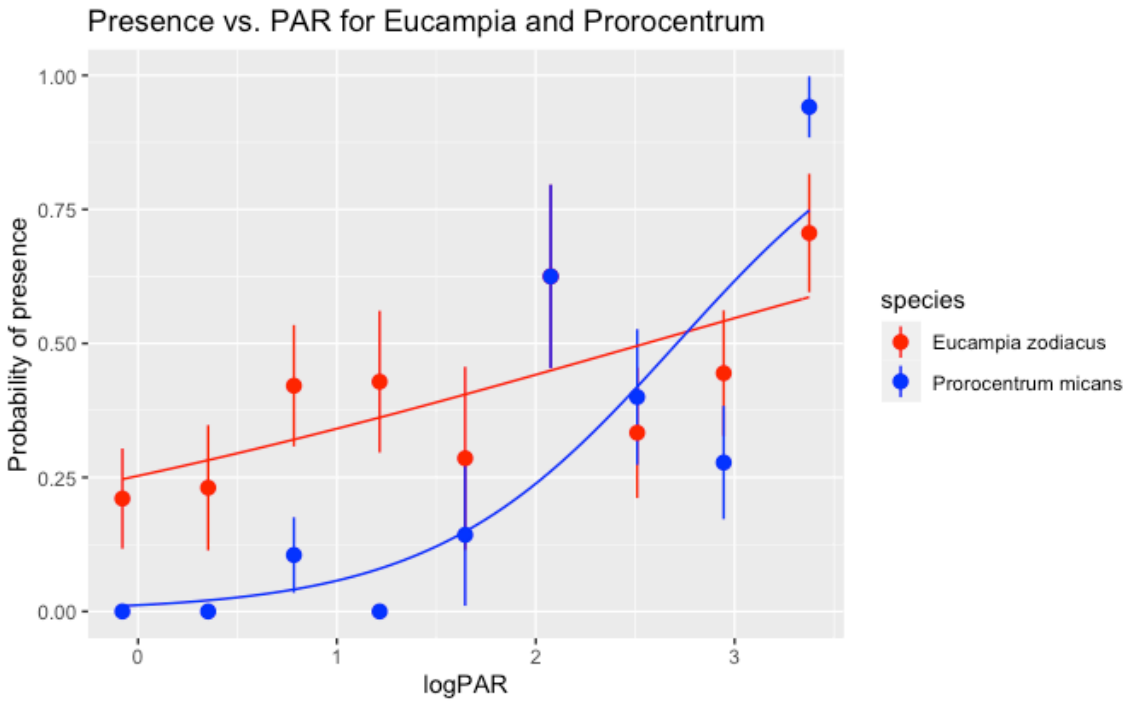
It is a bit confusing to keep track of the different kinds of output that you can get from anova() and Anova(). These are both multi-use functions that use different methods depending on the class of model you give it (lm or glm), depending on whether you give it a single model or two models to compare (for the lowercase anova() function), and depending on which options you specify (e.g. test = 'Chisq' to get a likelihood ratio test). Nonetheless these are the most useful functions for comparing linear models and generalized linear models, so you will need to get familiar with how they work.

The final note I'll make is that all of the above examples work the same for quasipoisson (or quasibinomial) models, but for these it is best to say test="F", because the dispersion parameter from the quasi model can be used to correct the log-likelihood and get an approximate F statistic.

Interpreting and visualizing the fitted model

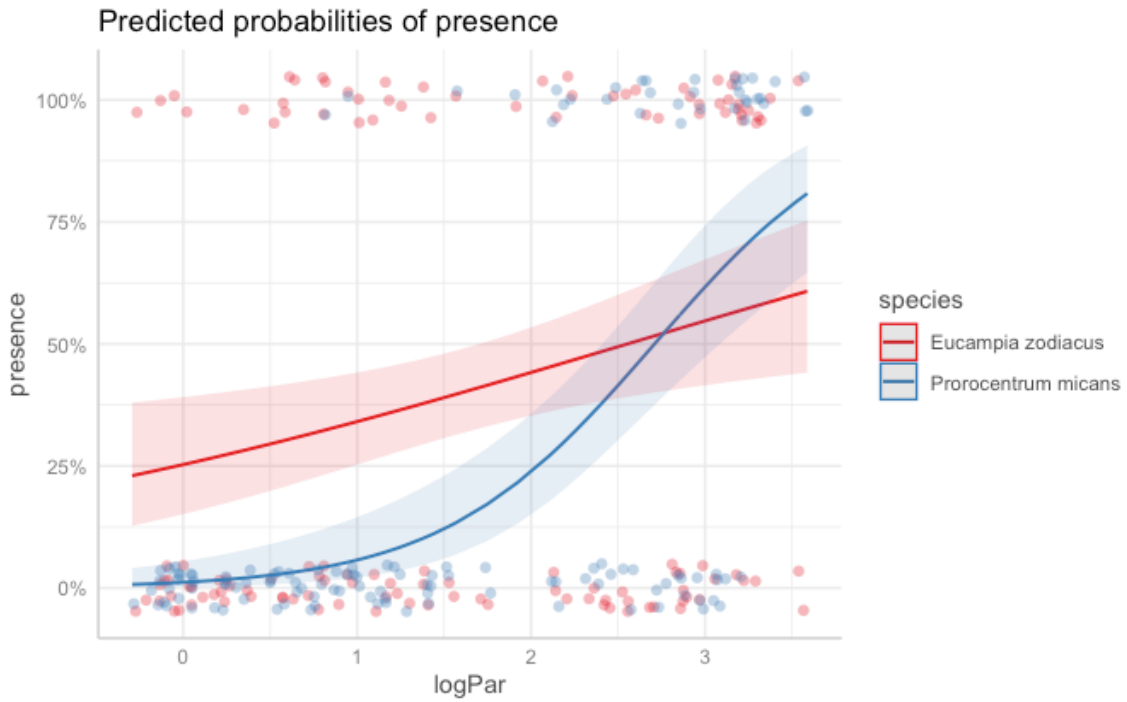
Let's think about what the model coefficients mean. We are modeling the probability of presence, and the coefficients are for a linear model on the scale of the link function, which is the logit by default. So '(Intercept)' is the probability of presence, on the logit scale, for *E. zodiacus*, when logPar = 0. 'logPar' is the increase in the probability of presence as logPar increases, for *E. zodiacus*. 'speciesProrocentrum micans' is the difference in probability between *P. micans* and *E. zodiacus*, when logPar = 0. And 'logPar:speciesProrocentrum micans' is the difference between *P. micans* and *E. zodiacus* in the slope that quantifies how the probability of presence increases as logPar increases. This is all getting a bit ugly, but it's still manageable to plot the fitted curves for the two species without too much effort:

```
logistic = function(x) exp(x)/(1+exp(x))
#plot the fitted curves

ggplot(par.bins, aes(bin.center, mean.prob, col = species)) + geom_poin
t() +
  geom_pointrange(aes(x = bin.center, y = mean.prob, ymin = mean.prob-s
e.prob, ymax = mean.prob+se.prob)) +
  scale_color_manual(values = c('red', 'blue')) +
  labs(title = 'Presence vs. PAR for Eucampia and Prorocentrum', x = "l
ogPAR", y = "Probability of presence") +
  geom_function(fun = ~logistic(coef(mod)[1]+coef(mod)[2]*.x), col = 'r
ed') +
  geom_function(fun = ~logistic(coef(mod)[1]+coef(mod)[3]+(coef(mod)[2]
+coef(mod)[4])*.x), col = 'blue')
```



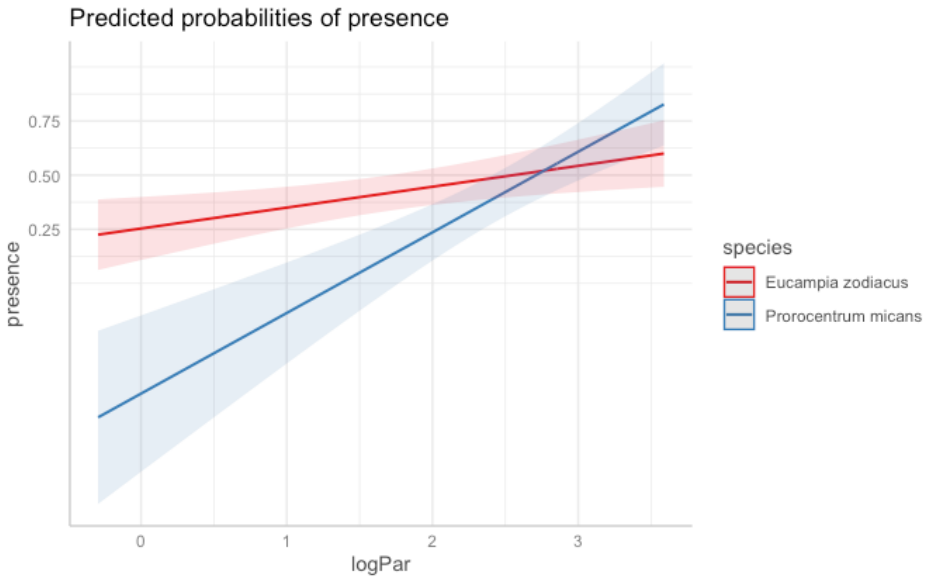
The fitted curve is pretty similar to what we would imagine from looking at the binned data, and in this case the curvature of the logistic curve is readily apparent for *P. micans*. As I've noted before, it is easier though maybe less revealing to plot the fitted curves using ggeffect:

```
plot(ggeffect(mod, c("logPar [all]", "species")))
```



So here we get the fitted effects of logPar, plus confidence intervals, for the two species. Note that we are now plotting the raw data instead of the binned summaries, which is less work but less informative. When comparing the slopes for the two species it can be easier to see the differences at a glance if we plot the curves on the link (logit) scale, because a GLM assumes linear relationships on the link scale:

```
plot(ggeffect(mod, c("logPar [all]", "species"))) + scale_y_continuous
(trans = logit_trans())
```



Note that I had to transform the scale manually (for some reason the back.transform in ggpredict does not work here), and that the raw data cannot be shown on this plot (because logit(0) and logit(1) are undefined).

Binomial with $n > 1$: proportional data

Binary data is the special case of the binomial when $n = 1$. The more general case of the binomial is useful for modeling proportions, specifically proportions where you can say 'X out of Y'. For example, the honey badger gave birth to 3 females out of a total litter of 4; 7 out of 10 cod sampled had at least one parasite. There are other kinds of proportional data that are not appropriate for the binomial distribution. For example, sessile benthic organisms are often measured in terms of percent cover, and plant herbivory can be quantified in terms of percent leaf damage. Neither of these involves an underlying binary variable that has a certain number of 'trials', so we can't use the binomial, and we will return to these more challenging cases later.

Binomial data is always fundamentally binary, because each 'trial' has two possible outcomes, and that's why the distribution is called bi-nomial. When $n > 1$ the important additional fact is that the data comes in sets of trials. For example, if we were recording the sexes of litters of honey badgers, the litters might looks like:

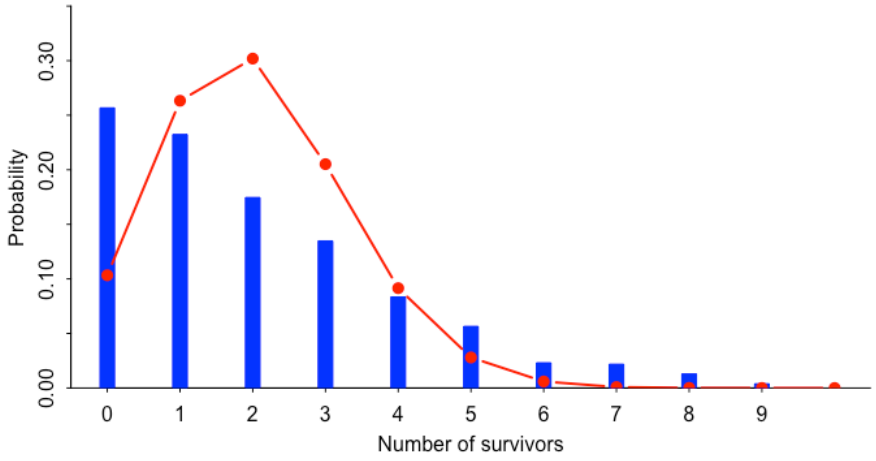
(M, M, F, F)
(M, M, F)
(F, M, F, F)
(F, M)
…

where M=male and F=female. In this case the parameter n would vary between litters, e.g. being 4, 3, 4, and 2 in the above examples.

When it comes to modeling this kind of data, the quantity we want to model is p, the probability of 'success', just like in the binary case we've already discussed. For honey badgers this would be the probability of giving birth to a female, if we coded F as 1 and M as 0. The binomial parameter n is not going to be an unknown quantity we're trying to model. We already know what n is, e.g we already know the total number of offspring, and we'll input that into the model.

So how will the general binomial case differ from the simple binary case? The most important difference is that when the data comes in groups of trials, there is the potential for # of successes to vary among groups in a way that is *overdispersed*. This is easiest to visualize if we have sets of trials where n is the same for all sets. For example, earlier in the course we looked at an experiment where the silverleaf whitefly was subjected to heat shocks, and the number of surviving flies was

recorded. Each experimental replicate had 10 flies, so this data is appropriate to model with a binomial distribution where $n = 10$ and p is the parameter we want to estimate. Here is a histogram of what a subset of the data looks like:



This plots the results for 55 replicates of the experiment. The histogram shows the number of survivors, out of 10 total, in each experiment. The red line shows the expected probabilities under a binomial distribution with $n = 10$ and p = the mean proportion of survivors across the 55 replicates (0.175). In this case the data appear to be somewhat overdispersed. There are too many very low and very high values, and not enough intermediate values. The variance of #survivors is 2.9, while the binomial distribution has a variance of $n*p*(1-p) = 10*0.175*(1-0.175) = 1.44$.

So the binomial distribution has a particular expectation for how much #survivors should vary, and if there is excess variability across the sets of trials then the data are overdispersed. This is important for the same reason it was important for the Poisson distribution: the variability in the data determines our confidence in the estimated parameters, and so if the real variability is greater than what the model expects then the confidence intervals and p-values will be inaccurate (too small). Here I am just showing a histogram of raw data to visualize what overdispersion looks like; when we actually fit a GLM it is not easy to visualize overdispersion, but it can be quantified just like we did with a Poisson model.

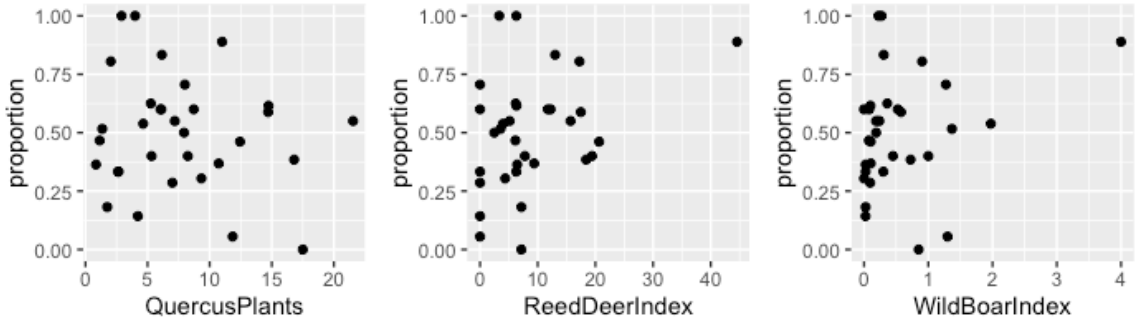
Example binomial GLM with n > 1.

Here's an example of modeling binomial data with n > 1. Data was collected on the incidence of tuberculosis (TB) in wild boar in Spain. Each sample consists of a certain number of boar that were trapped and inspected for TB. Samples were taken at many different locations, and the locations were coded for a number of variables that might predict incidence of TB. Here I'll look at the effects of three variables: the density of wild boar in the area (WildBoarIndex), which might affect

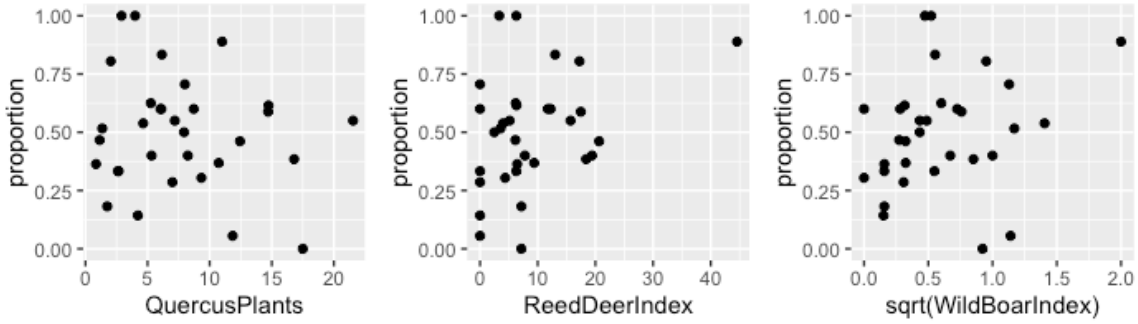
transmission rate; the density of deer in the area (ReedDeerIndex), which might affect transmission rate because deer also carry TB; and the degree to which the area is forested (QuercusPlants = oaks), which might affect transmission rate by affecting foraging activity etc. The dataset includes variables for how many boars had TB in a sample (BoarPosTB), and the total number of boars inspected in that sample (BoarSampledTB). We need both of these pieces of information to fit the binomial model. First I'll make a new variable that records the *proportion* of boars with TB in each sample:

```
boar$proportion = with(boar, BoarPosTB/BoarSampledTB)
```

We will use this variable in the model, but first let's do some exploratory plots:

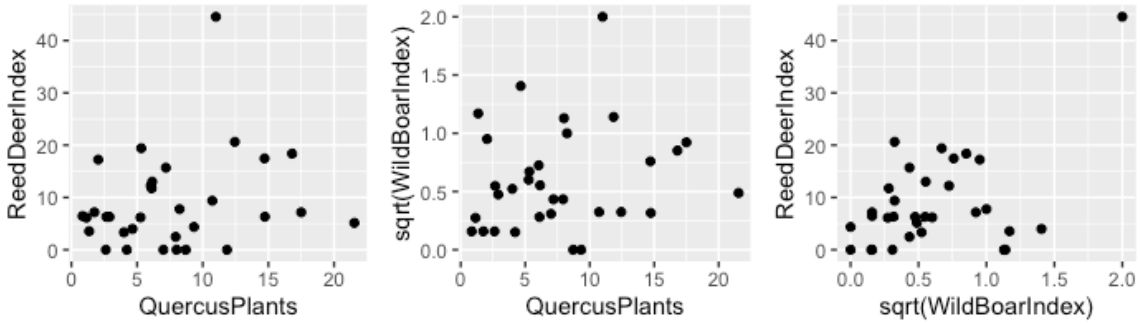


None of the scatter plots show super strong relationships, but there are some potential trends, and in a multivariate context the effects of individual predictors can become much clearer when a model with multiple terms is fit. I'm not that happy about the variation in the variable WildBoarIndex. It has a lot of small values and a few large values. This means that the few large values will have a large effect on the fitted relationship, which will tend to make the results less robust. We'll talk more about this in a future lecture, but one way to deal with this is to transform the predictor, e.g. a log transform. In this case I'll use a square root transform because the predictor contains zeros.



Now that third plot looks more sensible as something a (generalized) linear model can fit. Note that this does change the meaning of the coefficient for that variable, but since we're looking for general trends that's OK.

Before we put all these predictors in one model we should see if they are strongly collinear, which we can do with more scatterplots:



This is all the pairwise combinations of the three predictors. There might be some weak trends, but no strong collinearity, so it won't be problematic to put them all in one model.

Now let's make the model. There are two ways to specify a binomial model where $n > 10$. Here's the first way. As the response variable we use the proportion of 'successes', which in this case is the vector 'proportion', and we also need to use the argument weights=BoarSampledTB, which tells glm() how many boars were sampled for each row of the dataset.

```
mod = glm(proportion ~ ReedDeerIndex + WildBoarIndex + QuercusPlants,
weights = BoarSampledTB, data = boar, family = binomial)
```

So this is the same syntax as the earlier binomial example with presence-absence data (the diatoms), but now the response variable is the proportion of 'successes', which in this case is infected animals, and the 'weights' argument specifies how many 'trials' there are for each sample.

There is a second way to fit the same model. Instead of using the vector of proportions as the response variable, we can give glm() a *matrix* where the first column of the matrix is the number of 'successes' and the second column is the number of 'failures'. First we need to make a new variable in the dataframe that stores the number of failures. We can calculate that just as (total inviduals sampled - # infected):

```
boar$BoarNegTB = boar$BoarSampledTB - boar$BoarPosTB
```

Then we can give glm() the appropriate matrix, using cbind() to turn the appropriate vectors into matrix columns:

```
mod = glm(cbind(BoarPosTB, BoarNegTB) ~ ReedDeerIndex + WildBoarIndex +
QuercusPlants, data = boar, family = binomial)
```

In this case we don't use the 'weights' argument. This will fit the same model as the previous syntax.

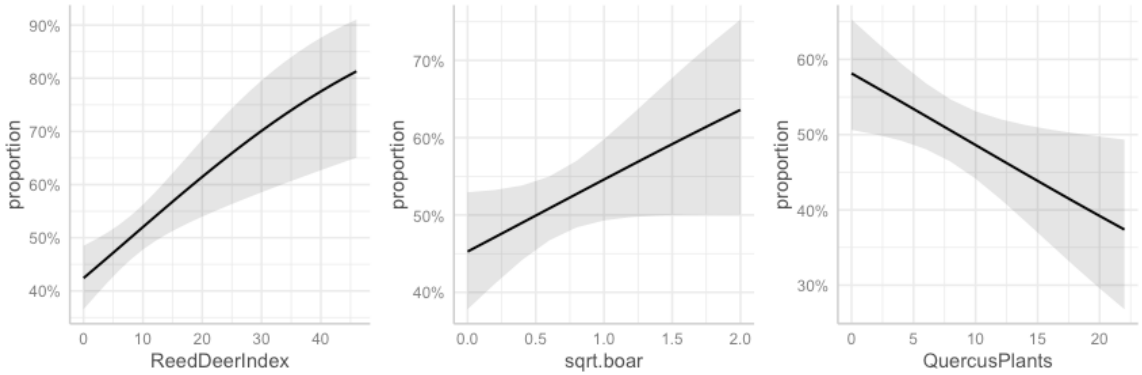
OK now let's see what the model fit looks like.

```
summary(mod)
```

```
##
## Call:
## glm(formula = proportion ~ ReedDeerIndex + WildBoarIndex + QuercusPlants,
##     family = binomial, data = boar, weights = BoarSampledTB)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -3.580  -1.065  -0.316   0.908   4.548
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -0.1224     0.1791   -0.68  0.49424
## ReedDeerIndex   0.0386     0.0115    3.36  0.00079 ***
## WildBoarIndex   0.2104     0.1317    1.60  0.10994
## QuercusPlants  -0.0382     0.0166   -2.29  0.02179 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 122.473  on 31  degrees of freedom
## Residual deviance:  97.155  on 28  degrees of freedom
## AIC: 194.6
##
## Number of Fisher Scoring iterations: 4
```

We get a coefficient estimate for each of the three predictors in the model, as expected. Let's visualize the fitted relationships using the ggeffects package:

```
grid.arrange(grobs = plot(ggeffect(mod), show.title = FALSE), nrow = 1,
 respect = TRUE)
```



It looks like the probability of a boar having TB increases as deer abundance increases, and also increases as boar abundance increases. Both of these are consistent with *a priori* expectation, because having more boar and deer around should increase the opportunity for transmission. There is also a negative trend with the amount of oaks. Like I said before, this function plots effects on the scale of the link function (the logit), and on that scale they look linear. It would also be nice to plot these effects against the raw data, to better visualize exactly what the model is doing, but with 3 predictors that can be challenging, and for now we'll just be satisfied with the scatterplots of the raw data, and the fitted effects.

Are the estimated effects significant? Here's a likelihood ratio test for each of the terms:

```
Anova(mod)
```

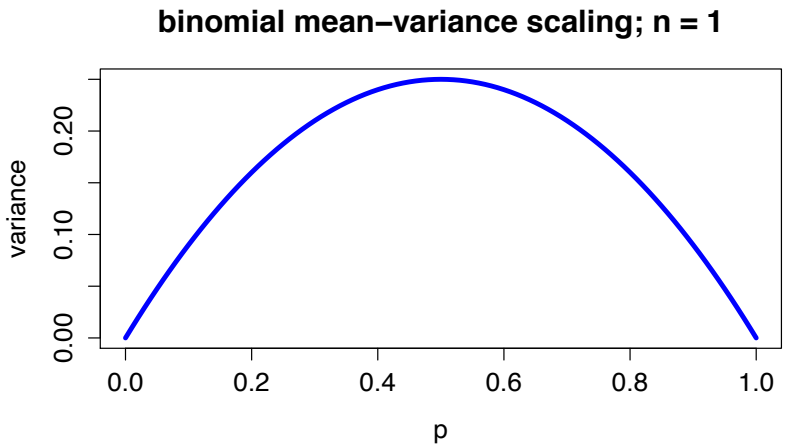
```
## Analysis of Deviance Table (Type II tests)
##
## Response: proportion
##                LR Chisq Df Pr(>Chisq)
## ReedDeerIndex    12.58  1    0.00039 ***
## sqrt.boar         3.46  1    0.06289 .
## QuercusPlants     5.45  1    0.01955 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The effect of deer and oaks is significant, the effect of boar is marginal.

Quasibinomial

So far I've been acting like the binomial distribution is appropriate for this data, but what about potential overdispersion? We could calculate the dispersion factor using the pearson residuals, as shown in lecture 7, or we could just fit a *quasibinomial* model. The quasibinomial approach is directly analogous to the quasipoisson approach. The logic works like this: we have data that we think is binomial-ish, in the sense that the mean-variance scaling should have the same overall shape. For the binomial the variance is $n*p*(1 - p)$, which looks like this if

you vary p and hold n constant:



The quasi-binomial approach just multiplies the expected binomial variance by a dispersion parameter, often abbreviated as φ. This allows for the variability in the data to be too large for the binomial, while still having the same shape as the binomial.

Let's fit the same model as before with the quasibinomial:

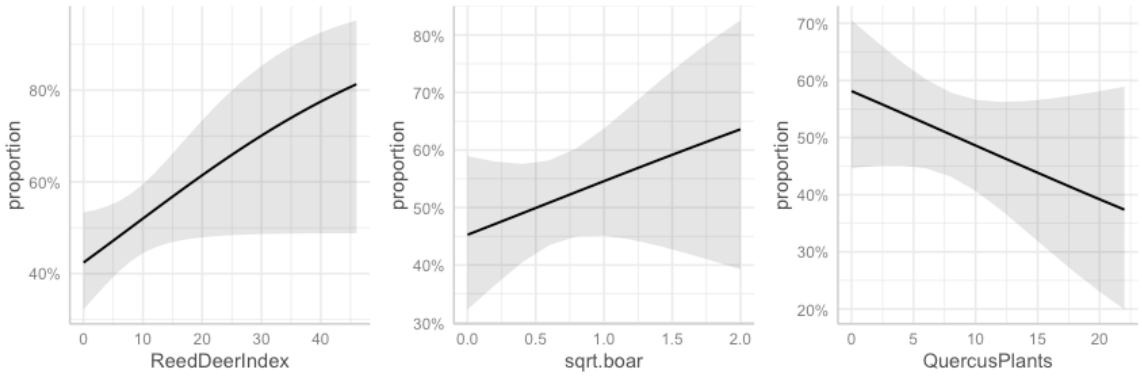
```
modq = glm(proportion ~ ReedDeerIndex + sqrt.boar + QuercusPlants, weights = B
oarSampledTB, data = boar, family = quasibinomial)

summary(modq)

##
## Call:
## glm(formula = proportion ~ ReedDeerIndex + sqrt.boar + QuercusPlants,
##     family = quasibinomial, data = boar, weights = BoarSampledTB)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -3.647  -1.078  -0.284   0.865   4.512
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -0.2372     0.3518   -0.67    0.506
## ReedDeerIndex   0.0386     0.0193    2.01    0.055 .
## sqrt.boar       0.3741     0.3457    1.08    0.288
## QuercusPlants  -0.0384     0.0284   -1.35    0.187
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasibinomial family taken to be 2.934)
##
##     Null deviance: 122.473  on 31  degrees of freedom
## Residual deviance:  96.331  on 28  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 4
```

The dispersion factor is 2.934, which is fairly large. As for the Poisson, a value of 1 means no overdispersion, and a value of >1.5 is often considered problematic. As for the Poisson, we can get corrected hypothesis tests with an F-test that is derived from the likelihood ratio corrected with the dispersion factor:

```
Anova(modq, test = 'F')
```

```
## Analysis of Deviance Table (Type II tests)
##
## Response: proportion
##                  SS Df    F Pr(>F)
## ReedDeerIndex 12.6  1 4.29  0.048 *
## sqrt.boar      3.5  1 1.18  0.287
## QuercusPlants  5.5  1 1.86  0.184
## Residuals     82.1 28
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Now only the effect of deer is significant, and even its p-value is much larger than before. The effects look the same, but with bigger confidence intervals:



So maybe there is an effect of boar density and oak density on TB, but we don't have much evidence from the data for those effects once we properly account for uncertainty due to overdispersion.

For count data we could use the quasipoisson approach for overdispersion, or we could use the negative binomial distribution, which has the advantage of being fit with normal likelihood methods. Is there an analogous distribution to the negative binomial for overdispersed proportion data? There is, and it is called the beta-binomial distribution. But for some reason it is not as commonly used as the negative binomial, and requires some less commonly used packages to fit, so we'll skip it here. When we get to mixed models we will learn another way to deal with overdispersed binomial data.