Module 5: Generalized Linear Models in R

The purpose of this handout is to introduce you to some of the advanced statistical analyses using R. It is assumed you are using the RxP, RxP.clean and RxP.byTank datasets created in the earlier modules.

1 Understanding Non-Normal Data

In Modules 3 - 5, we discussed the utility of the lm() function for analyzing normally distributed data. Normal data are, as you hopefully remember, data where you have a relatively even spread of values above and below the mean. However, in the real world, data are often not normal. In these cases we can turn to Generalized Linear Models (GLM's), which extend the modeling framework of lm() to many other error structures ("error" refers to the spread of data around the mean). The basic function for running a GLM is the aptly named glm() function.

GLM's work via a "link" function, which transforms the data to a normal scale. For example, with a binomial GLM (also called a logistic regression) we are not modeling the data (usually a 0 or 1), but are instead modeling the log-odds of an event happening (getting a 1) or not (getting a 0). The built in error families available with glm() are as follows:

```
binomial(link = "logit")
gaussian(link = "identity")
Gamma(link = "inverse")
inverse.gaussian(link = "1/mu^2")
poisson(link = "log")
quasi(link = "identity", variance = "constant")
quasibinomial(link = "logit")
quasipoisson(link = "log")
```

Note that the link functions listed above are the defaults for each error family and are automatically assumed, so you do not need to actually write them in your model specification. Several families have multiple possible link functions, some of which may be preferred in certain circumstances, so it is good to know the different possible options. There are also several other specific functions that have been developed for certain error structures not encorporated in the original glm() fucntion, such as the function betabin() (in the aod package) for analyzing beta-binomial data or glm.nb() for analyzing negative binomial structured data.

2 Generalized Linear Models

Let's create a new variable which is the number of tadpoles that survived to metamorphosis, or the flip-side of that, the number that died before metamorphosis. We can once again use the dplyr package to accomplish this. However, now instead of using summarise() to calculate the mean() of each group of data, we can use the function length() to calculate how many of each group we have. Below, we've calculated the length (i.e., the number of values in a vector) of the summarized Ind variable. We could use any of the variables there, since group_by() essentially subsets the data by each tank and since we are taking the lenght fo the

```
> temp <- RxP.clean %>%
+           group_by(Tank.Unique) %>%
+           summarise(N.alive = length(Ind))
> temp
# A tibble: 78  2
   Tank.Unique N.alive
         <int>   <int>
1            1      47
2            2      42
3            3      45
4            4      26
5            5      40
6            6       8
7            7      43
8            8      39
9            9      23
10          10      44
# ... with 68 more rows
```

`Tank.Unique` is of course the number of each tank and `N.alive` is the total number of metamorphs from each tank. Now that we have these numbers calculated, we want to add this column to our existing `RxP.byTank` data frame. There are two ways to do this. First, assuming that our rows are in the 'exact' same order, we can just create a new column in `RxP.byTank`.

```
RxP.byTank$N.alive<-temp$N.alive
```

Alternatively, we could easily modify our code from the earlier module when we made the `RxP.byTank` dataframe to include the command to make the `N.alive` variable. Since you should still have this code in a script somewhere, it is super simple to modify it.

```
RxP.byTank<-RxP.clean %>%
  group_by(Tank.Unique, Pred, Res, Hatch, Block) %>%
  summarise(Age.DPO = mean(Age.DPO),
            Age.FromEmergence = mean(Age.FromEmergence),
            SVL.initial = mean(SVL.initial),
            Tail.initial = mean(Tail.initial),
            SVL.final = mean(SVL.final),
            Mass.final = mean(Mass.final),
            Resorb.days = mean(Resorb.days),
            N.alive = length(Ind)) ##Note that this line is  new!!
```

It might also be useful to calculate the number of tadpoles that died or were eaten before metamorphosis. Since we know that we started with 50 tadpoles per tank, this is very simple to calculate.

```
> RxP.byTank$N.dead<-50-RxP.byTank$N.alive
> str(RxP.byTank)
'data.frame': 78 obs. of  14 variables:
 $ Tank.Unique      : int  1 2 3 4 5 6 7 8 9 10 ...
```

```
$ Pred             : Factor w/ 3 levels "C","L","NL": 3 1 1 2 3 2 3 1 2 1 ...
$ Res              : Factor w/ 2 levels "Hi","Lo": 1 1 1 2 1 1 2 2 1 2 ...
$ Hatch            : Factor w/ 2 levels "E","L": 2 1 2 2 1 1 2 1 2 2 ...
$ Block            : int  1 1 1 1 1 1 1 1 1 1 ...
$ Age.DPO          : num  47.2 45.4 53.8 56.9 64.8 ...
$ Age.FromEmergence: num  13.2 11.4 19.8 22.9 30.8 ...
$ SVL.initial      : num  19.4 18.4 18.9 18.8 19.7 ...
$ Tail.initial     : num  4.83 5.37 4.8 4.63 5.43 ...
$ SVL.final        : num  19.7 19 19.1 19.1 20.1 ...
$ Mass.final       : num  0.418 0.382 0.412 0.382 0.486 ...
$ Resorb.days      : num  3.49 3.79 3.51 3.65 4.22 ...
$ N.alive          : int  47 42 45 26 40 8 43 39 23 44 ...
$ N.dead           : num  3 8 5 24 10 42 7 11 27 6 ...
```

Now we have two new variables which represent the number of red-eyed treefrog tadpoles in each tank that survived to metamorphosis or died before getting there. Let's look at the distribution of the mortality data.

```
> hist(RxP.byTank$N.dead)
```

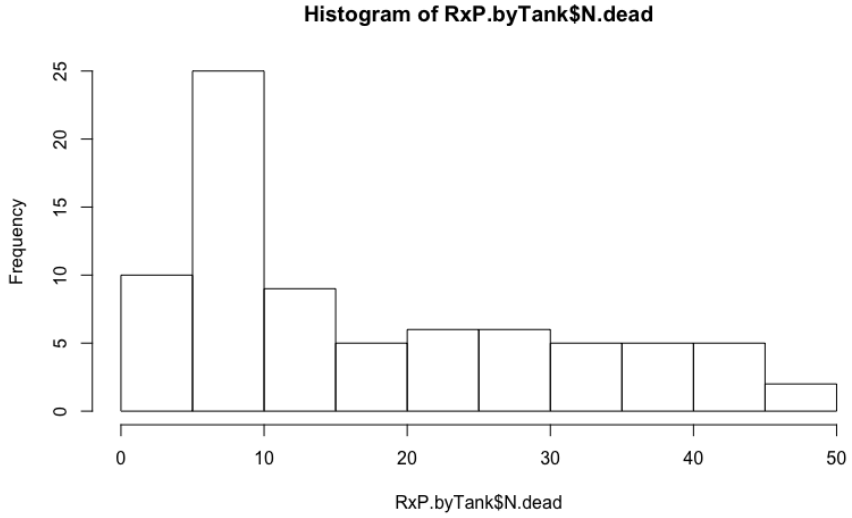

Figure 1: The distribution of the number of tadpoles that died prior to metamorphosis.

This shows us that most of the values are fairly small (i.e., low mortality) but that some tanks had very high mortality. There is a long tail of data out to the right. This shape of a data distribution is likely Poisson or negative binomial. They are count data (whole integers), cannot be negative and fit the general shape of what we expect Poisson data to look like.

So how do we decide what error distribution fits best? Recall from an earlier module that the function `fitdistr()` from the `MASS` package allows us to evaluate the fit of different error distributions

to a given set of data. We can use that here to see which error distribution works best for these data. Note that you have to have a little bit of preliminary knowledge to know which distributions you might want to choose from (don't worry, that stuff comes in time). Here, let's evaluate four distributions: 1) normal, 2) lognormal, 3) Poisson, and 4) negative binomial.

```
> fit1<-fitdistr(RxP.byTank$N.dead, "normal")
> fit2<-fitdistr(RxP.byTank$N.dead, "lognormal")
> fit3<-fitdistr(RxP.byTank$N.dead, "Poisson")
> fit4<-fitdistr(RxP.byTank$N.dead, "negative binomial")
> AIC(fit1,fit2,fit3,fit4)
fit1 2 625.6163
fit2 2 598.3991
fit3 1 1063.0072
fit4 2 598.5247
```

Looking at the AIC scores above, you can see that the Poisson is by far the worst fit (recall that smaller AIC scores are better) and that the lognormal and negative binomial are pretty similar. Let's explore the differences between these different models. First, let's make four models, each using the error distributions above, and let's use Predators and Resources (and their interaction) as our predictor variables. Notice that we use several different functions below: `lm()` for the normal and lognormal distributions, `glm()` for the Poisson distribution, and a special version of the `glm()` function that is just for the negative binomial, `glm.nb()`, which is found in the `MASS` package (so make sure to load the package first). Since the function specifies that it is for a negative binomial, you do not need to specify a `family` argument, as in the `glm()` function. Also note that you could use `glm()` for the first two models if you specify `family=gaussian`, since gaussian is a fancy stats way of saying normal.

```
lm.n<-lm(N.dead~Pred*Res, data=RxP.byTank)
lm.ln<-lm(log(N.dead)~Pred*Res, data=RxP.byTank)
glm.p<-glm(N.dead~Pred*Res, family="poisson", data=RxP.byTank)
glm.negb<-glm.nb(N.dead~Age.DPO*Pred, data=RxP.byTank)
```

2.1 Understanding and Interpreting the GLM

So you just coded your first Generalized Linear Models. Hooray! What you should have noticed is that the code for a `glm` is exactly the same as `lm`, except now we have added an extra argument to specify the error family. So simple! In a moment we'll examine the `summary()` outputs and you will see that they are remarkably similar to what you have looked at before as far as linear models.

Now that we've made our models, let's examine their diagnostic plots. Remember, these are all looking at the same data - the number of tadpoles that died before metamorphosis - it's just that they make different assumptions about how the data should be modeled. It's not as simple as comparing two means or something.

```
> par(mfrow=c(2,2))##Note that this line allows you plot all 4 diagnostic plots at once!
> plot(lm.n)
> plot(lm.nl)
> plot(glm.p)
> plot(glm.negb)
```

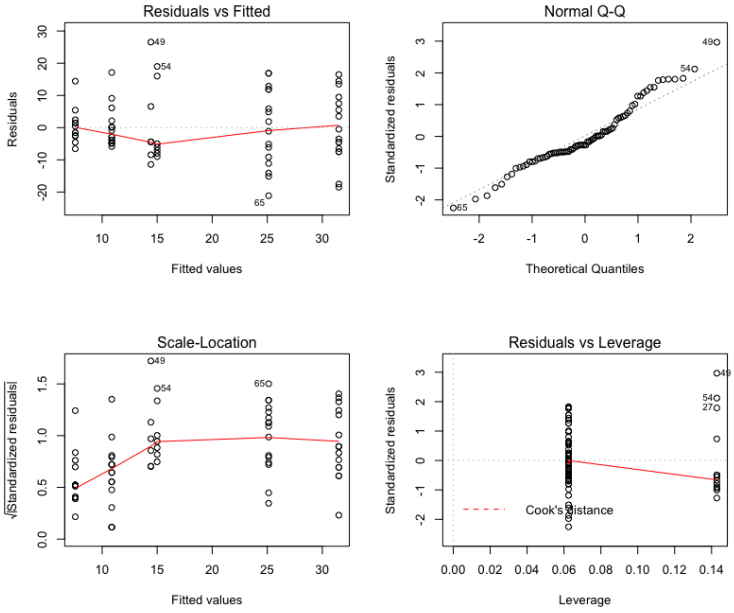

Figure 2: Diagnostic plots of `lm.n`.

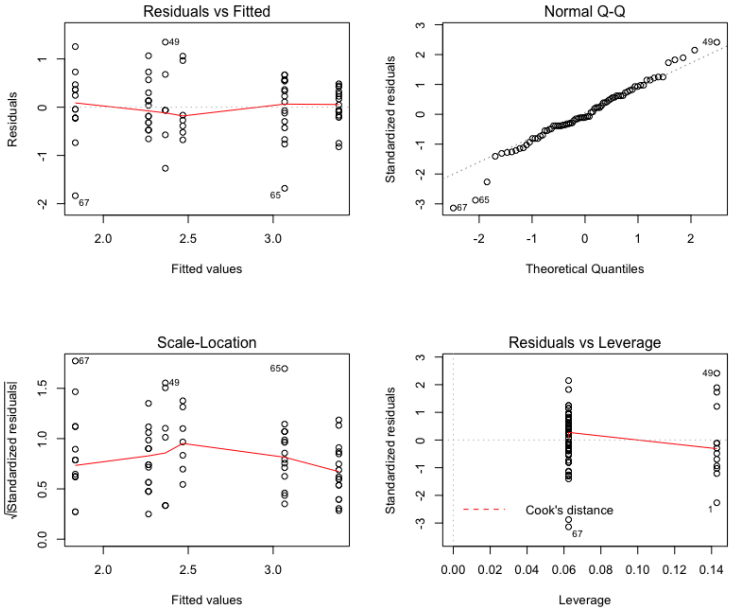

Figure 3: Diagnostic plots of `lm.nL`.

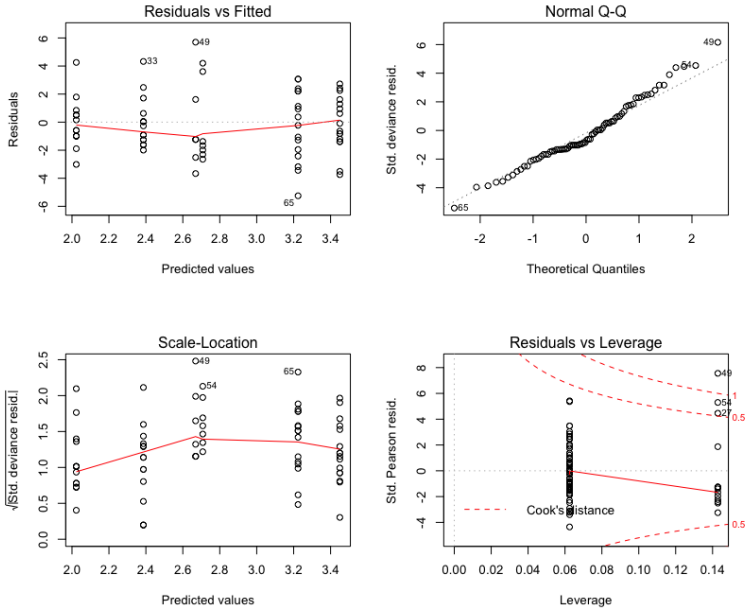

Figure 4: Diagnostic plots of `glm.p`.

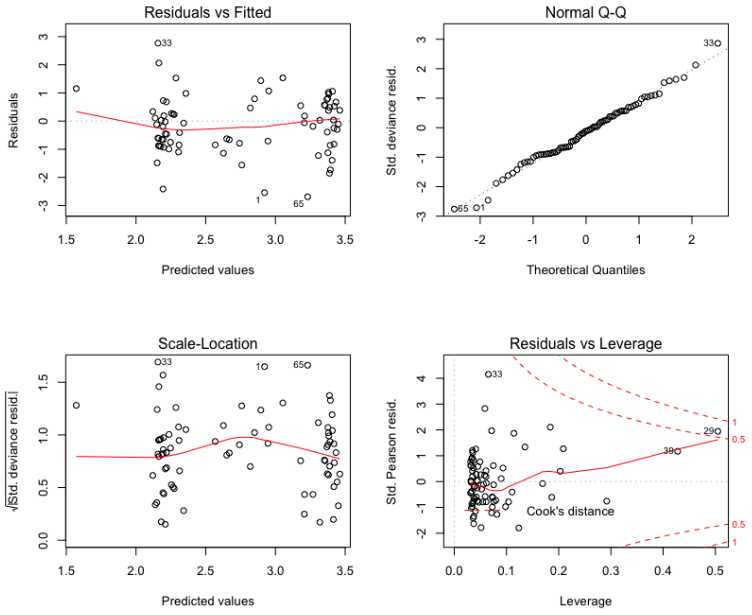

Figure 5: Diagnostic plots of `glm.negb`.

2.2 Comparing the diagnostic plots and making a decision

Okay, so what should you be looking at here? As described in an earlier module, you are looking for funky patterns and points that are labeled as nto fitting particularly well. I usually pay the most attention to the QQ-plot, which is 2nd plot of the 4 (the upper right in each set on the previous pages). Recall that in this plot you are more or less looking for the points to fall relatively close to the dashed line. The normal distribution is surprisingly not terrible, given how non-normal the data clearly are. The lognormal QQ-plot has a few points that are very far from the line, as does the Poisson. Overall, I would say that the negative biomial fits the best but the Poisson is pretty okay. However, if you look at the Residuals vs Leverage plot (the 4th plot in each set) you can see that there are a number of problematic points in the Poisson model and fewer in the negative binomial. Neither are terrible, but in general the negative binomial looks the best of these four models.

Now, let's compare the summery outputs for the Poisson and negative binomial models. First, let's walk through the Poisson output.

```
> summary(glm.p)

Call:
glm(formula = N.dead ~ Pred * Res, family = "poisson", data = RxP.byTank)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-5.2488  -1.3954  -0.7289   1.0861   5.7004

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   2.02320    0.09091  22.255  < 2e-16 ***
PredNL        0.64601    0.13478   4.793 1.64e-06 ***
PredL         1.20066    0.10369  11.579  < 2e-16 ***
ResLo         0.36326    0.11837   3.069  0.00215 **
PredNL:ResLo -0.32442    0.18286  -1.774  0.07603 .
PredL:ResLo  -0.13714    0.13595  -1.009  0.31310
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 710.99  on 77  degrees of freedom
Residual deviance: 352.95  on 72  degrees of freedom
AIC: 712.4

Number of Fisher Scoring iterations: 5
```

First, notice that your output looks almost exactly the same as for the linear models we have looked at before. However, in the world of GLMs we no longer talk about the **variance** of the data in our model, but instead the **deviance** of the data. They mean the same thing. Here, the `Null deviance` gives us a unitless measure of how much variation there is in the data, and `Residual deviance` gives

us an estimate of how much of the variation is explained by our model. As a general rule of thumb, you should hope that your `Residual deviance` is not more than twice your degrees of freedom. Above, our `Residual deviance` is nearly 5X our df's, which is not good. This is a phenomenon called *overdispersion*, which is when your data has more variation than is expected by model. As mentioned above, there are no great metrics or tests for "what is too much overdispersion?" but a good rule of thumb is that if the Residual deviance is more than double the Residual degrees of freedom, you are in trouble.

Luckily for us, an overdispersed Poisson model *is* a negative binomial error distribution.

```
 > summary(glm.negb)

Call:
glm.nb(formula = N.dead ~ Pred * Res, data = RxP.byTank, init.theta = 4.618042907,
    link = log)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5788  -0.7213  -0.3361   0.4428   2.4394

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)    2.0232     0.1476  13.703  < 2e-16 ***
PredNL         0.6460     0.2503   2.581  0.00984 **
PredL          1.2007     0.1945   6.174 6.66e-10 ***
ResLo          0.3633     0.2027   1.792  0.07308 .
PredNL:ResLo  -0.3244     0.3498  -0.927  0.35371
PredL:ResLo   -0.1371     0.2695  -0.509  0.61081
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for Negative Binomial(4.618) family taken to be 1)

    Null deviance: 156.263  on 77  degrees of freedom
Residual deviance:  80.091  on 72  degrees of freedom
AIC: 554.85

Number of Fisher Scoring iterations: 1


              Theta:  4.618
          Std. Err.:  0.962

 2 x log-likelihood:  -540.854
```

You can see that the Residual deviance is much lower, meaning that the model has done a better job of accounting for the variation in the model. We will proceed with this model for the examples here.

The output from a GLM is on the scale of the link function. The link for a negative binomial is the same as for Poisson, a log-link. Thus, you would use the `exp()` to transform the parameters back onto the original scale of the data (numbers of tadpoles that died). Note that calculating the various effects sizes is exactly the same as what we did for `lm()` in Module 3. Recall that to calculate the estimates from the summary output, interaction effects are the sum of all the lower level effects. For example:

```
> exp(2.0232)#Control high
[1] 7.562486
> exp(2.0232 + 0.6460)#Nonlethal high
[1] 14.42842
> exp(2.0232 + 1.2007)#Lethal high
[1] 25.12592
> exp(2.0232 + 0.3633)#Control low
[1] 10.87536
> exp(2.0232 + 0.6460 + 0.3633 -0.3244)#Nonlethal low
[1] 15.00075
> exp(2.0232 + 0.6460 + 0.3633 -0.1371)#Lethal low
[1] 18.09074
```

2.3 Calculating statistical significance

Just as with linear models, we can use the `Anova()` function in the `car` package to calculate the significant predictors in our model. You can use the `anova()` but I highly recommend the version with the capital 'A' as it is just more conservative and better.

```
> Anova(glm.negb)
Analysis of Deviance Table (Type II tests)

Response: N.dead
         LR Chisq Df Pr(>Chisq)
Pred       71.617  2  2.809e-16 ***
Res         3.979  1    0.04607 *
Pred:Res    0.873  2    0.64615
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

As you can see, both predators and resources (although to a lesser extent) affected the number of tadpoles that survived to metamorphosis. We can see this if we plot the data. It's easy to see the effects of both predictors if we use the `qplot()` function from the `ggplot2` package.

```
> qplot(data=RxP.byTank, x=Pred, y=N.alive, geom="boxplot", facets=~.Res)
```

2.4 Coding the data as a binomial GLM

This is a bit of a side note, but we could also think about analyzing these data with a binomial GLM (also called a Logistic Regression) since they are proportional data. Traditionally, binomial data are 0's

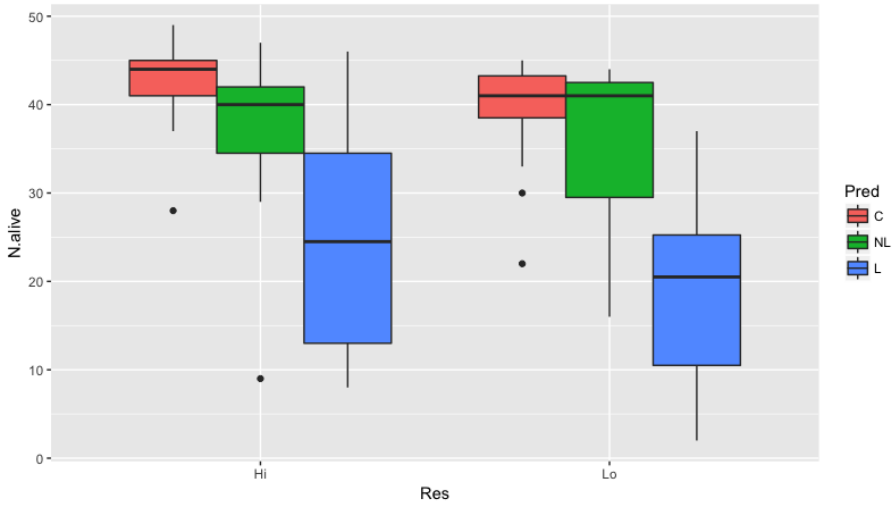

Figure 6: Predators, and to a lesser extent resources, greatly affect how many tadpoles survive to metamorphosis. Tadpoles with higher levels of resources had greater survival, while lethal predators consumed LOTS of tadpoles.

and 1's. However, R allows you to model proportional data using a binomial error family. In this case, the data are coded as a two-column table featuring wins and losses, or animals that lived and died. Thus, the two columns add up to the total starting number of individuals. This is important, you cannot code the data as individuals that died and total starting number (well, you could code it that way but your results would be incorrect). To make the two column table, you 'bind' together two 'columns' with the `cbind()` function.

```
> glm.b<-glm(cbind(N.alive,N.dead)~Pred*Res, family="binomial", data=RxP.byTank)
> summary(glm.b)

Call:
glm(formula = cbind(N.alive, N.dead) ~ Pred * Res, family = "binomial",
    data = RxP.byTank)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-7.804  -1.666   1.018   1.788   6.470

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   1.72483    0.09868  17.480  < 2e-16 ***
PredNL       -0.82250    0.15380  -5.348  8.9e-08 ***
PredL        -1.73483    0.12140 -14.291  < 2e-16 ***
ResLo        -0.44454    0.13070  -3.401 0.000671 ***
PredNL:ResLo  0.38950    0.21120   1.844 0.065146 .
PredL:ResLo  -0.07768    0.16566  -0.469 0.639134
```

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

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1212.65  on 77  degrees of freedom
Residual deviance:  641.86  on 72  degrees of freedom
AIC: 955.54

Number of Fisher Scoring iterations: 4
```

Wow, that is really overdispersed! If this was the approach you wanted to take, you can model an overdispersed binomial dataset with a type of model called a *beta-binomial*, using the function `betabin()` in the aod package ('aod' stands for 'analysis of overdispersed data').

2.5 Mixing GLMs and ANCOVA's together

Let's make our model more interesting, at least for the point of illustrating the techniques we want to explore. Perhaps we want to know how survival to metamorphosis relates to the length of the larval period and resource levels?

```
> glm.negb2<-glm.nb(N.dead~Age.DPO*Pred, RxP.byTank)
> summary(glm.negb2)

Call:
glm.nb(formula = N.dead ~ Age.DPO * Pred, data = RxP.byTank,
    init.theta = 5.128645402, link = log)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6883  -0.8392  -0.1132   0.5432   2.7670

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)     2.043076   0.308071   6.632 3.32e-11 ***
Age.DPO         0.002329   0.003834   0.607 0.543607
PredNL          1.559002   0.506024   3.081 0.002064 **
PredL           1.953070   0.535289   3.649 0.000264 ***
Age.DPO:PredNL -0.016738   0.006883  -2.432 0.015024 *
Age.DPO:PredL  -0.015103   0.009171  -1.647 0.099593 .
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for Negative Binomial(5.1286) family taken to be 1)

    Null deviance: 168.971  on 77  degrees of freedom
Residual deviance:  80.845  on 72  degrees of freedom
```

```
AIC: 549.55

Number of Fisher Scoring iterations: 1


              Theta:  5.13
          Std. Err.:  1.11

 2 x log-likelihood:  -535.552
```

We can also use the `Anova()` function from the `car` package to calculate our summary statistics for the model.

```
> Anova(glm.negb2)
Analysis of Deviance Table (Type II tests)

Response: N.dead
             LR Chisq Df Pr(>Chisq)
Age.DPO         2.482  1    0.11519
Pred           45.980  2  1.036e-10 ***
Age.DPO:Pred    8.306  2    0.01571 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

2.6 Using the `predict()` function with a GLM

This model is analogous to an ANCOVA, as we have a mix of categorical and continuous predictors. We could try to figure out the nonlinear regression from the summary output, but instead we can use the very handy `predict()` function to calculate predicted values from the model. Remember that to use the `predict()` function, you need to provide a model and an empty data frame that has starter values each of the predictors in your model. We will provide a sequence from 40-140 (the length of our x-axis), and our three Predator treatments. When you start to have multiple predictors that need to be accounted for when, it can be very useful to use the function `expand.grid()` to build your blank data frame, which we have discussed before. `expand.grid()` will create a data frame for every combination of values you provide it. This can be very useful, but be careful, because you can easily create an *enormous* dateframe.

Let's make an object called `predicted.data` where the output from `predict()` will be stored. Note that using `expand.grid()` in this manner does not work if you are trying to get confidence intervals (because that output is a list, which does not work with a data frame). By looking at the `head` and `tail` of the object we can see what it has created. I will show you a very easy way to calculate confidence intervals shortly. Below, I've also made a blank column where we will put our predicted values.

```
> predicted.data<-expand.grid(Age.DPO=40:140, Pred=c("C","NL","L"), N.dead=NA)
> head(predicted.data)
  Age.DPO Pred N.dead
1      40    C     NA
2      41    C     NA
```

```
3      42    C     NA
4      43    C     NA
5      44    C     NA
6      45    C     NA
> tail(predicted.data)
    Age.DPO Pred N.dead
298     135    L     NA
299     136    L     NA
300     137    L     NA
301     138    L     NA
302     139    L     NA
303     140    L     NA
```

Now, we can use this new data frame to easily get our predicted values.

```
> predicted.data$N.dead<-predict(glm.negb2, newdata=data.frame(predict.data), type="response")
> head(predicted.data)
  Age.DPO Pred   N.dead
1      40    C 8.467445
2      41    C 8.487187
3      42    C 8.506975
4      43    C 8.526810
5      44    C 8.546690
6      45    C 8.566617
```

Now, let's plot how age at metamorphosis interacts with predator treatment to affect survival. To add the predicted nonlinear regression lines, we can use the function `lines()` which will add a line to an existing plot. We just have to provide `lines()` with a series of x-values and y-values. We can use indexing to just plot the data we want.

```
> plot(N.dead~Age.DPO, RxP.byTank, col=RxP.byTank$Pred, pch=16, xlab="Age at
  metamorphosis (days post-oviposition)", ylab="Number of tadpoles that died before metamorphosis")
> lines(x=40:140, y=predicted.data$N.dead[predicted.data$Pred=="C"], col=1,
  lwd=2)
> lines(x=40:140, y=predicted.data$N.dead[predicted.data$Pred=="NL"], col=2,
  lwd=2)
> lines(x=40:140, y=predicted.data$N.dead[predicted.data$Pred=="L"], col=3,
  lwd=2)
> legend(x="topright", legend=c("Control","Nonlethal","Lethal"), cex=2, col=1:3,
  pch=16, bty="n")
```

2.7 Making a much easier ANCOVA plot using `ggplot2`

Now that you are a pro at using the `predict()` function, let me teach you a much easier way to make nice looking scatterplots with confidence intervals. We have previously seen that we can use `qplot()` to make quick and easy scatterplots. In our model above, we are looking at the interaction between predators and age at metamorphosis, which can be plotted with one line, as such(notice I've tacked on a 'theme' to the end, which gets rid of the grid lines and does some other things to make a nice plot):

```
> qplot(x=Age.DPO, y=N.dead, data=RxP.byTank, col=Pred, ylab="Number of
  dead tadpoles")+theme_classic()
```

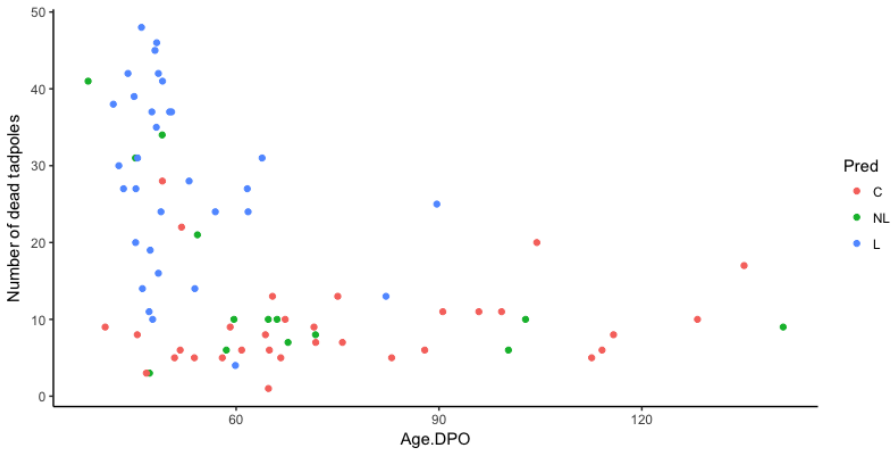

Figure 7: A scatterplot made with `qplot`.

"How do we add lines to that scatterplot?", you might be asking. It is amazingly easy. All you do is add an extra function at the end of your `qplot()`. This style of coding is different than other plotting we've done and it can take some getting used to, but it works very well. Just like how we've defined "geoms" in other plots, we can add a "geom" to this plot. Here we will add a `geom_smooth` to the figure. Now, you have to tell `geom_smooth` what sort of line you want to draw, and to do that it is important to think for a moment how the line gets there in the first place. A regression line on a scatterplot is the result of a formula, right? And that formula (e.g., y=mx+b) comes from a statistical model. In our case, we want to add a regression that is based on a negative binomial model, so we add an argument about the "method" to use, which is `method=glm.nb`. Thus, inside the plotting function `R` is going to run a negative binomial model on our data, calculate the curves and their confidence intervals for us, and plot it all nicely.

```
> qplot(x=Age.DPO, y=N.dead, data=RxP.byTank, col=Pred, ylab="Number of
  dead tadpoles", xlab="Age at metamorphosis (days post-oviposition") +
  geom_smooth(method=glm.nb)+theme_classic()
```

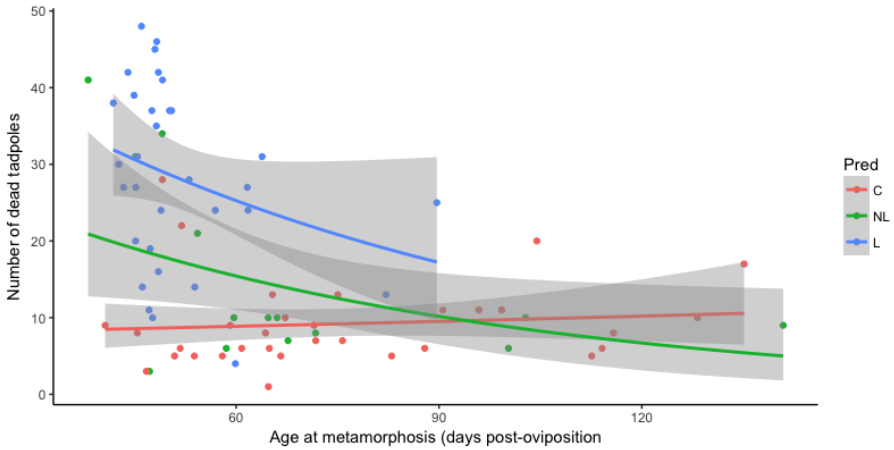

Figure 8: A scatterplot made with `qplot`, which added nonlinear regression curves and their confidence intervals from a negative binomial regression. Wasn't that a piece of cake?

3 Assignment!

Here is an assignment to work on the skills you have just developed.

1. Explore how all three predictors from the experiment (Hatch, Pred and Res) effect (or don't effect) the number of animals that *died* before metamorphosis. Start with the full model including all interactions and reduce it to the minimal adequate model, aka the model with just significant predictors.

2. Plot the data in an appropriate manner to visualize what significant effects you find.

3. Run one other model of your own design that utilizes an ANCOVA approach (mix of categorical and continuous predictors) and plot it using `qplot()`, as above.

Turn in to me (via Moodle) a word doc or pdf with both models and figures. The assignment is due the evening of Wednesday 3/28/2018. Happy coding.