# GLMs and Friends

Stacy DeRuiter

2023-05-25

# Table of contents

# Preface

This is a collection of notes on multiple regression models, generalized linear models, mixed-effects models, generalized estimating equations, and generalized additive models. They were originally developed for teaching the courses STAT 245 and DATA 545 at Calvin University.

They are a work in progress. If you have questions, please reach out.

# 1 Linear Regression

You probably learned something about linear regression in a previous course. Here, we briefly review the main concepts of simple linear regression and quickly expand our tool box to multiple regression (with both quantitative and categorical predictors).

## 1.1 Data

We will consider a small dataset from an article by J.S. Martin and colleagues, titled *Facial width-to-height ratio is associated with agonistic and affiliative dominance in bonobos (**Pan paniscus**)*

Notes: variable `fWHR` is the facial width-height ratio and `AssR` is the Assertiveness score of affiliative dominance. `normDS` is another dominance score.

What figures should we consider, to get a sense of the dataset and think about how we might model `fWHR`?

```
Rows: 117
Columns: 8
$ Name   <chr> "Zuani", "Zuani", "Zorba", "Zorba", "Zorba", "Zomi", "Zomi", "Z~
$ Group  <chr> "Apenheul", "Apenheul", "Wilhelma", "Wilhelma", "Wilhelma", "Fr~
$ Sex    <chr> "Female", "Female", "Male", "Male", "Male", "Female", "Female",~
$ Age    <dbl> 22, 22, 34, 34, 34, 15, 15, 14, 14, 14, 18, 18, 18, 18, 18, 12,~
$ fWHR   <dbl> 1.475052, 1.321814, 1.581446, 1.479237, 1.390086, 1.340909, 1.2~
$ AssR   <dbl> 5.36, 5.36, 2.36, 2.36, 2.36, 3.92, 3.92, 4.74, 4.74, 4.74, 2.6~
$ normDS <dbl> 1.430, 1.430, 2.341, 2.341, 2.341, 3.087, 3.087, 3.035, 3.035, ~
$ weight <dbl> 24.0, 24.0, NA, NA, NA, NA, NA, 41.6, 41.6, 41.6, 38.0, 38.0, 3~
```

## 1.2 Preliminary Considerations

Just as we imagine before we start coding to create graphics, we ought to think before we start fitting models.

Traditional ways of interpreting statistical results are premised on the idea that you made a plan, got some data, fitted the model you planned, and want to draw conclusions.

If, instead, you got data, scrutinized the data, fitted lots of different models, and now want to report results from the one that fitted best…well, generally things tend to go wrong. This is especially true if you use the data to lead you from a more complex to a simpler model. As Harrell (2015) points out in section 4.3,

- Uncertainty *underestimated* (overconfidence: standard errors and confidence intervals too small; $R^2$ too big)
- Spurious relationships look important and slope estimates are biased high
- If testing hypotheses, p-values *too small*

How can we avoid these problems? Some more insight will come when we consider model assessment and selection in future sections. For now, we need to remember:

> Fitting and interpreting **one** well-considered, sensible model is prefereable to trying many things and then trying to choose among them later.

## 1.3 Response and Predictors

A regression model is our attempt to quantify how a **response variable** of interest changes when a set of **predictor variables** change.

So, to begin, we need to identify our (one) response variable – the thing we are most interested in measuring or predicting or describing or understanding.

Then, we need to identify a set of predictor variables that we expect to be associated with changes in the response. (If we are planning an experiment, they should be variables we can collect data on; if working with data already collected, they must be in or derived from the data available.)

How do we choose which predictors to include, and how many?

First, rely on experts and previous experience. If you know the context of the problem well, *you* have a good sense of the predictors that will be of interest. If you don't, then you should consult experts (or published work on the topic).

There are also practical limits on the number of predictors you can reasonably consider, given a dataset.

### 1.3.1 Sample size and predictors

One important consideration, when planning a regression model, is: *How many predictors can I reasonably include?*

It depends on the size of the dataset: it takes several observations to get a good estimate of any statistics, so it makes sense that fitting a model with *lots* of predictors will require a bigger dataset. And if you try to fit too many, the chances of *overfitting* increase. *Overfitting* is when you model noise as well as signal, capturing in your model apparent relationships that actually exist only in the current dataset, not in reality.

For linear regression, Harrell (2015, Chapter 4.6) offers a rule of thumb: the number of parameters being estimated, $p$, should be less than $\frac{n}{10}$ or $\frac{n}{20}$. To give just one standard rule of thumb, we should aim for $p < \frac{n}{15}$. $n$ is the sample size (number of rows in the dataset).

## 1.4 Simple linear regression, Residuals & Least squares

First, let's review and consider a simple (one-predictor) linear regression model. Fit the model

```
slr <- lm(fWHR ~ AssR, data=bonobos)
```

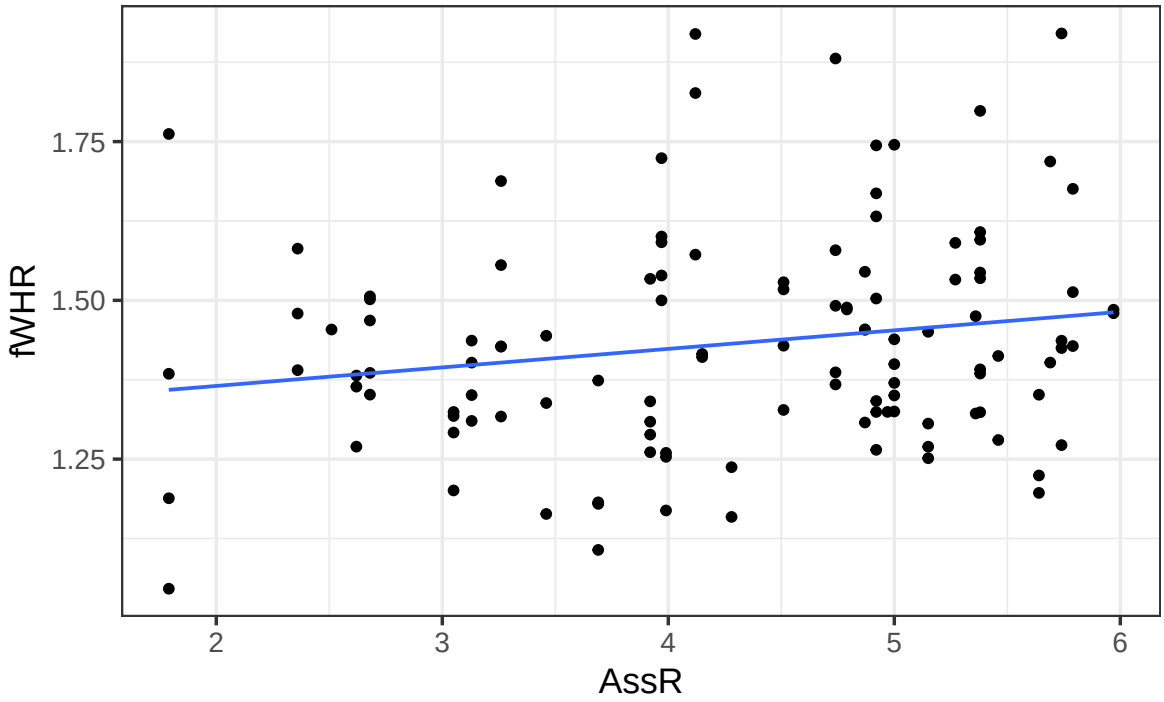
Extract the slope and intercept values:

```
coef(slr)
```

```
(Intercept)        AssR
 1.30685287  0.02918242
```

Add the regression line to the plot:

```
gf_point(fWHR ~ AssR, data=bonobos) |>
  gf_lm()
```




```
summary(slr)
```

```
Call:
lm(formula = fWHR ~ AssR, data = bonobos)

Residuals:
     Min       1Q   Median       3Q      Max
-0.31320 -0.11369 -0.01242  0.09008  0.49241

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.30685    0.06283  20.801   <2e-16 ***
AssR         0.02918    0.01420   2.055   0.0421 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1689 on 115 degrees of freedom
Multiple R-squared:  0.03542,   Adjusted R-squared:  0.02704
F-statistic: 4.223 on 1 and 115 DF,  p-value: 0.04213
```

### 1.4.1 Using `lm()` to fit a linear regression in R

We use function `lm()` with a formula of the form `y ~ x` (and an input `data = _____`).

To view the result, we ask for a `summary()`.

```
slr <- lm(fWHR ~ AssR, data = bonobos)
summary(slr)
```

```
Call:
lm(formula = fWHR ~ AssR, data = bonobos)

Residuals:
     Min       1Q   Median       3Q      Max
-0.31320 -0.11369 -0.01242  0.09008  0.49241

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.30685    0.06283  20.801   <2e-16 ***
AssR         0.02918    0.01420   2.055   0.0421 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1689 on 115 degrees of freedom
Multiple R-squared:  0.03542,   Adjusted R-squared:  0.02704
F-statistic: 4.223 on 1 and 115 DF,  p-value: 0.04213
```

From this traditional on-screen output, we can fetch the information and estimates we need.

We can also "clean up" the output into a data table using `broom::tidy()`:

```
broom::tidy(slr)
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)   1.31      0.0628     20.8  8.83e-41
2 AssR          0.0292    0.0142      2.06 4.21e- 2
```

### 1.4.2 Equation of the fitted regression line

In the process above, we chose a *response* variable $y$, a *predictor* $x$, and estimated the slope and intercept of the line $\beta_0$ and $\beta_1$.

$$y = \beta_0 + \beta_1 x$$

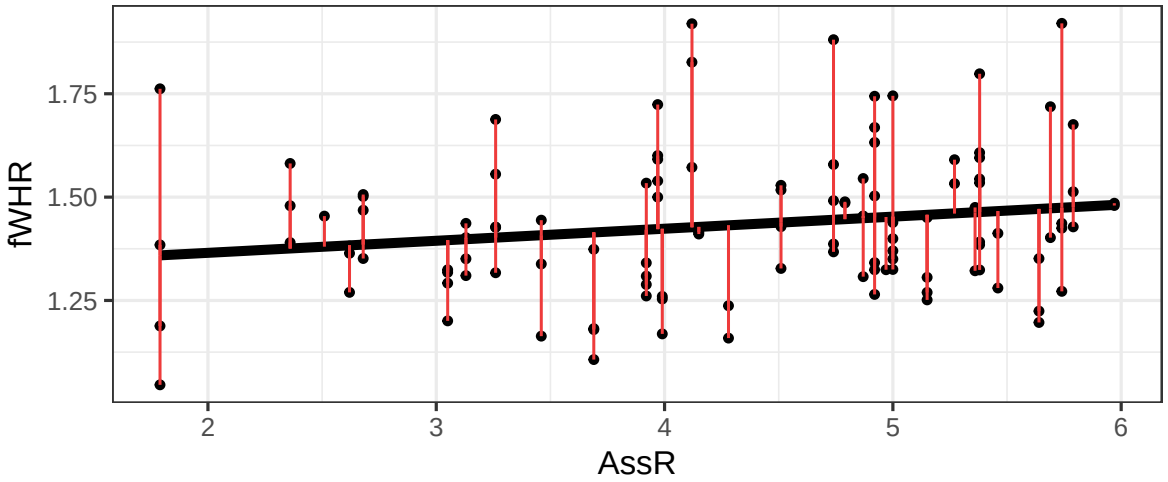
But the data don't lie perfectly on the line, and we have too include that in our model, too. There is some error $\epsilon$ in our model for the data, so we should keep track of it.

$$y = \beta_0 + \beta_1 x + \epsilon$$

But $\epsilon$ is not *one number* to be estimated – the amount of error is different for every data point! How can we express this mathematically?

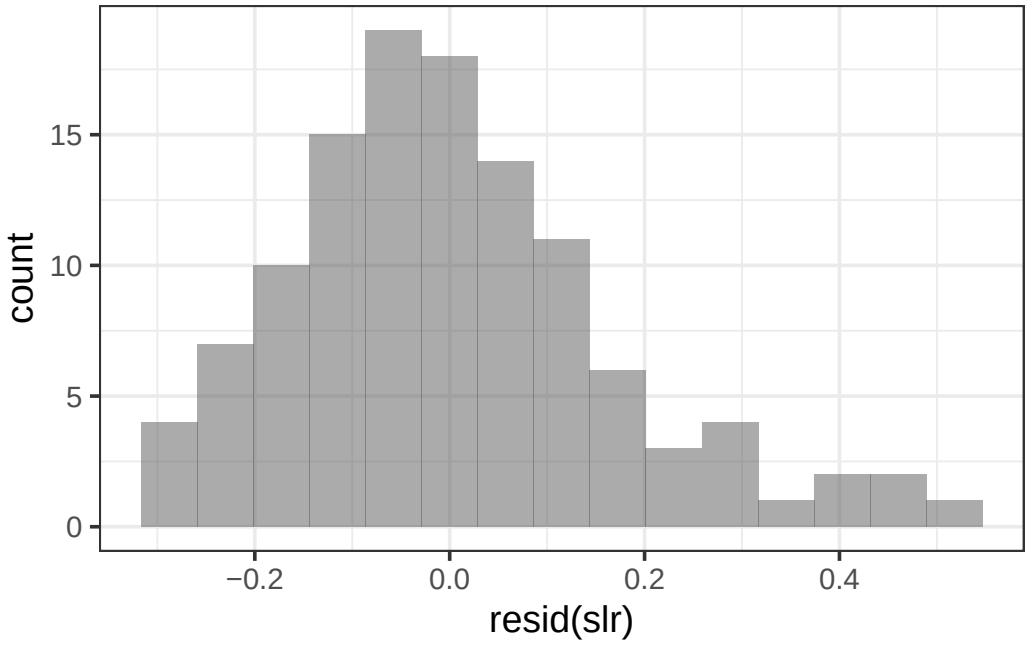
### 1.4.3 Regression residuals = "errors"

First, we need a way to measure how far the model line is from the data, for every observation in the dataset. The *residual* for the $i$th observation is the observed response variable value from the data ($y_i$) minus the *predicted response value* on the regression line ($\hat{y}_i$). This will be the response variable value for the point on the line whose predictor ($x$ value is the same as for the observed data point). These "residuals" are shown in red on the figure. (The numeric value of each residual is the length of its red line segment.)




How can we summarize this information? Let's look at the distribution of the residuals:

```
gf_histogram(~resid(slr), bins = 15)
```




Hmmm…looks pretty unimodal, and symmetric (except for a bit of right skew). It almost has that familiar bell shape…like a Normal distribution. This is a hint.

In linear regression, we model residuals with a normal distribution, with mean zero (the line should go through the "middle" of the points, and averaging over them all, our residual should be 0).

The standard deviation of the residuals depends on how far away the points are, on average, from the line. This value, the residual standard deviation $\sigma$, must be estimated in each case.

R `lm()` summary output calls it "residual standard error" and lists it near the bottom of the summary. (This is a somewhat inaccurate name but we're stuck with it.)

So now we can give a *complete* model equation. In general,

$$y = \beta_0 + \beta_1 x + \epsilon,$$

where

$$\epsilon \sim N(0, \sigma)$$

Or, in our specific case with the Bonobos,

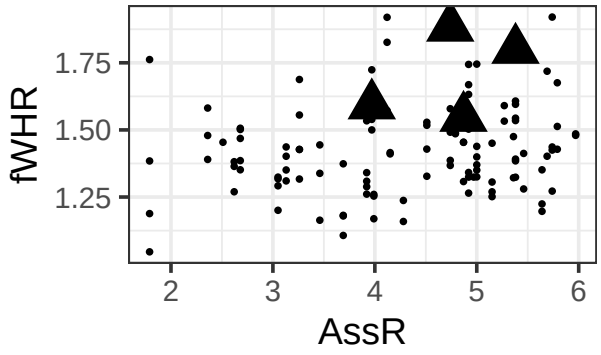
$$y = 1.31 + 0.029x + \epsilon, \text{ where } \epsilon \sim N(0, 0.169)$$

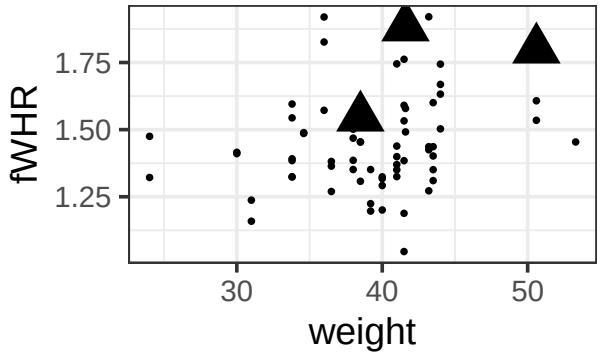
## 1.5 Multiple regression

Rarely does our response variable **really** depend on only one predictor. Can we expand our formulation to include more predictors? In R, it's super easy:

```
mlr <- lm(fWHR ~ AssR + weight, data=bonobos)
coef(mlr)
```

```
(Intercept)        AssR      weight
0.944790930 0.039888045 0.008644299
```



### 1.5.1 Choosing predictors

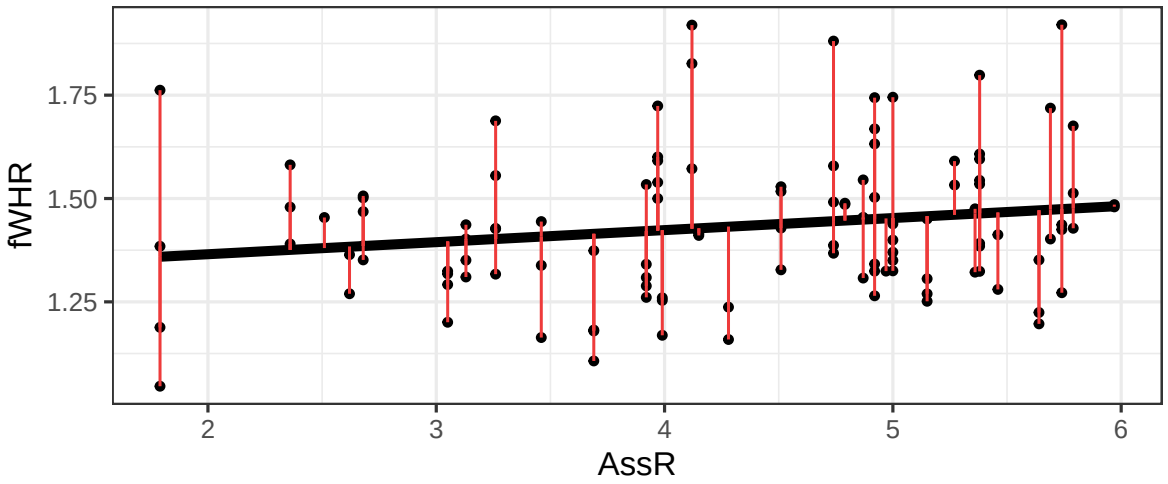
Notice that in this chapter, we are building up from a simpler model to a more complex one with more predictors. This is to reveal to you the mathematical machinery that lets us specify and fit the more complex models.

**But this is *not* how to build a model in practice**. Recall, we agreed to think carefully and choose a reasonable set of predictors to use at the outset, doing as much of the work as possible before even looking at the dataset. Our $p < \frac{n}{15}$ rule gives a rough limit to the maximum number of parameters we can estimate. If we do otherwise, we have to be cautious and know all our conclusions are on much shakier ground.

### 1.5.2 Estimation

But...how did R come up with those estimates of the slope and intercept (and residual standard deviation) parameters?

In the simple linear regression case, it's easy to visualize:




The *best* line is the one that makes the residuals the smallest (by going right through the middle of the points). OK, let's be careful: some residuals are positive and some are negative, and we want the line that minimizes their magnitude. The traditional approach is to choose the slope that minimizes the sum of the squared residuals,

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

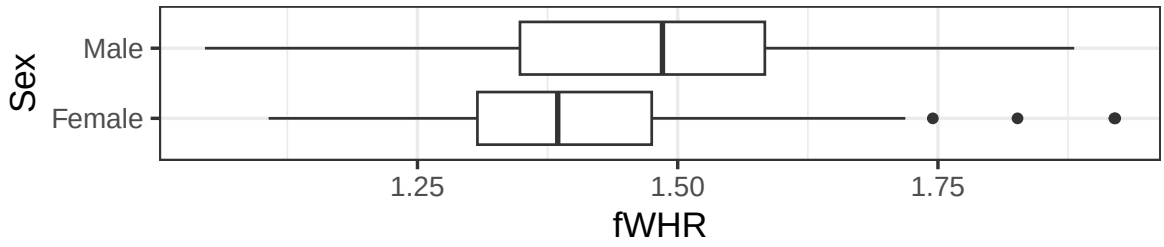
How does this work when we have *multiple* predictors? It's harder to draw, but just as easy to compute $\hat{y}$ and thus the observed residuals $e_i$.

### 1.5.3 Computing Predictions

Use the regression equation to compute **predicted values** for the three data points below:

```
# A tibble: 4 x 3
   fWHR  AssR weight
  <dbl> <dbl>  <dbl>
1  1.88  4.74   41.6
2  1.80  5.38   50.6
3  1.59  3.97   NA
4  1.55  4.87   38.5
```

## 1.6 Predictors with two categories



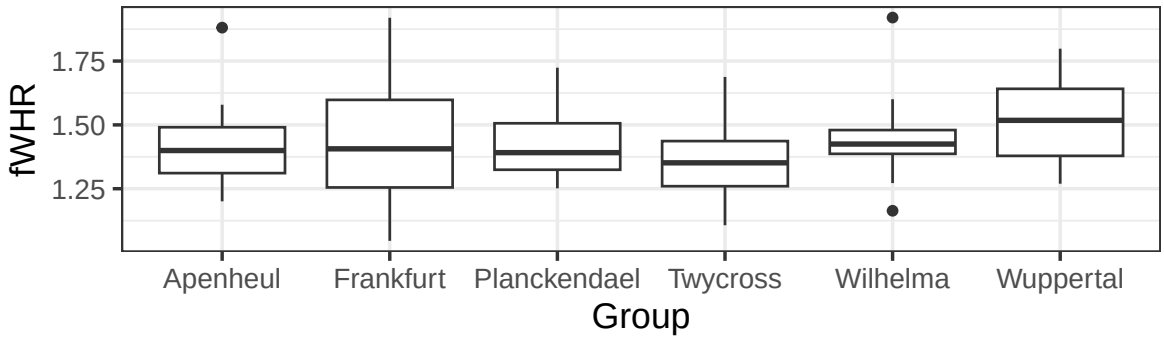

```
mlr2 <- lm(fWHR ~ AssR + weight + Sex, data = bonobos)
coef(mlr2)
```

```
(Intercept)        AssR      weight     SexMale
1.065420976 0.058435841 0.002257142 0.128484275
```

How does the model incorporate this covariate mathematically?

### 1.6.1 Predictors with more categories

```
gf_boxplot(fWHR ~ Group, data = bonobos)
```




```
mlr3 <- lm(fWHR ~ AssR + weight + Sex + Group, data = bonobos)
coef(mlr3)
```

```
    (Intercept)            AssR          weight         SexMale
    1.007734691     0.064361973     0.003458979     0.124854271
GroupFrankfurt GroupPlanckendael     GroupTwycross     GroupWilhelma
    0.037426358    -0.008464572    -0.112907589     0.011186724
GroupWuppertal
   -0.004364826
```

How does the model incorporate **this** covariate mathematically?

## 1.7 Returning to the R Model Summary

There are several bits of information you should be able to extract from the `summary()` output R produces on a fitted linear regression model:

- $\beta$s, Coefficient Estimates
- $\sigma$, labeled "residual standard error"
- $R^2$ (adjusted)

```r
mlr3 <- lm(fWHR ~ AssR + weight + Sex + Group, data = bonobos)
summary(mlr3)
```

```
Call:
lm(formula = fWHR ~ AssR + weight + Sex + Group, data = bonobos)

Residuals:
     Min       1Q   Median       3Q      Max
-0.38288 -0.09488 -0.02642  0.07196  0.48464

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)        1.007735   0.217585   4.631 2.05e-05 ***
AssR               0.064362   0.021158   3.042   0.0035 **
weight             0.003459   0.005547   0.624   0.5353
SexMale            0.124854   0.059278   2.106   0.0394 *
GroupFrankfurt     0.037426   0.074892   0.500   0.6191
GroupPlanckendael -0.008465   0.075407  -0.112   0.9110
GroupTwycross     -0.112908   0.074779  -1.510   0.1364
GroupWilhelma      0.011187   0.085538   0.131   0.8964
GroupWuppertal    -0.004365   0.071292  -0.061   0.9514
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1691 on 59 degrees of freedom
  (49 observations deleted due to missingness)
Multiple R-squared:  0.2517,    Adjusted R-squared:  0.1502
F-statistic:  2.48 on 8 and 59 DF,  p-value: 0.02167
```

## 1.8 Predictions from the model

### 1.8.1 By Hand

The equation for the fitted model above is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 I_{Male} + \beta_4 I_{Frankfurt} + \beta_5 I_{Planckendael} + \beta_6 I_{Twycross} + \beta_7 I_{Wilhelma} + \beta_7 I_{Wuppertal} + \epsilon$$

where

- $y =$
- $\beta_0 =$

- $x_1 =$
- $x_2 =$
- $\beta_1, \beta_2, \beta_3 ...$ are:
- $I_{Male} =$
- $I_{Frankfurt} =$
- $I_{Planckendael} =$ , etc.
- $\epsilon =$

#### 1.8.1.1 Comprehension check:

What is the expected fWHR (according to this model) for a 30 kg female bonobo at the Wilhelma zoo?

### 1.8.2 Prediction Plots in R

We can ask R to compute predictions for **all** the data points in the real dataset.

```
bonobos <- bonobos |>
  mutate(preds = predict(mlr3))
```

```
Error in `mutate()`:
i In argument: `preds = predict(mlr3)`.
Caused by error:
! `preds` must be size 117 or 1, not 68.
```

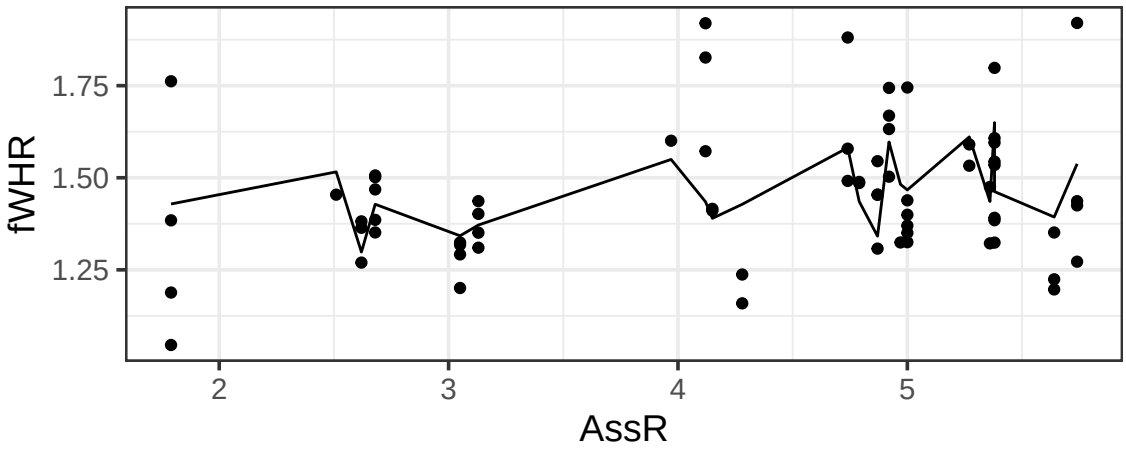
Wait, what? This error is because the `lm()` function removes rows containing missing values from the dataset, so it computes only 68 residuals (for the complete cases in the data). This doesn't match the 117 rows in the original data. We can solve the problem by omitting rows with missing values first. To be safe, we first select only the variables we need, so we don't omit rows based on missing values in unused variables.

```
b2 <- bonobos |>
  dplyr::select(fWHR, weight, AssR, Sex, Group) |>
  na.omit() |>
  mutate(preds = predict(mlr3))
```

*We have a full set of predictions!*

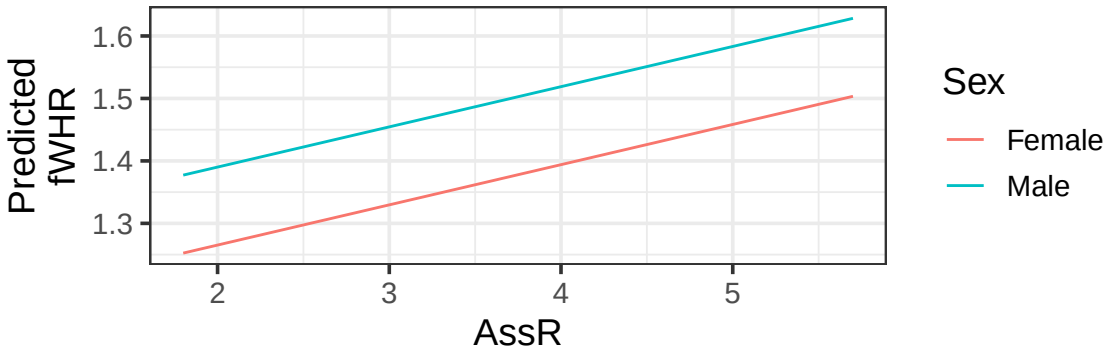
But if we plot these predictions on a scatter plot of `fWHR` as a function of `AssR`, we *do not* get a straight line, because the predictions are also impacted by varying values of `weight`, `Sex`, and `Group`:

```
gf_point(fWHR ~ AssR, data = b2) |>
  gf_line(preds ~ AssR, data=b2)
```




*But…we would really like a straight line that helps us visualize the meaning of the* $\beta$ (slope coefficient) for `AssR`. We can make predictions for a **hypothetical** dataset, in which `AssR` varies over a reasonable range, but the other predictors stay constant. This lets us see how `AssR` (and only `AssR`) affects the response, without contributions from other predictors. In choosing the values to include in hypothetical dataset, we often choose to hold variables constant at their most common or median values, but not blindly: also, avoid impossible or implausible variable combinations (for example, specifying that a person lives in the state of Michigan but the city of Chicago, or that they are a 5-year-old person with 4 children). *In this case, to match the figures in the published paper, we are also going to vary the* ***`Sex`*** *- but generally you'd only allow one predictor to vary.*

```
fake_data <- expand.grid(AssR = seq(from=1.8, to=5.7, by=0.05),
                         weight = 38.5,
                         Sex = c('Female', 'Male'),
                         Group = 'Wuppertal')

fake_data <- fake_data |>
  mutate(preds = predict(mlr3, newdata = fake_data))
gf_line(preds ~ AssR, color = ~Sex, data=fake_data) |> gf_labs(y='Predicted\nfWHR')
```

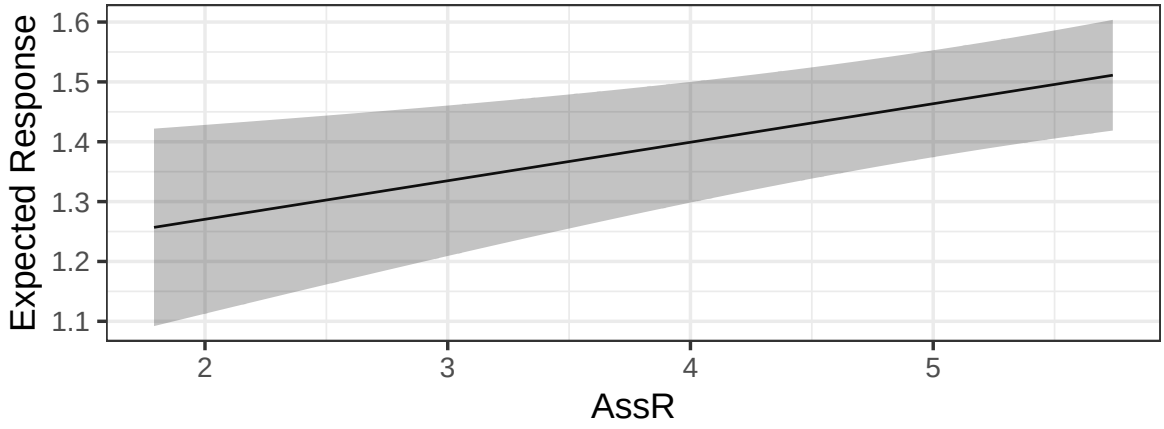



#### 1.8.2.1 Comprehension checks:

- Should we overlay prediction-plot line(s) on the data scatter plot?
- How do you think the plot would look if we changed the constant predictor values?
- What is missing from this picture?

#### 1.8.2.2 Shortcut

```
library(s245)
pred_plot(mlr3, 'AssR')
```



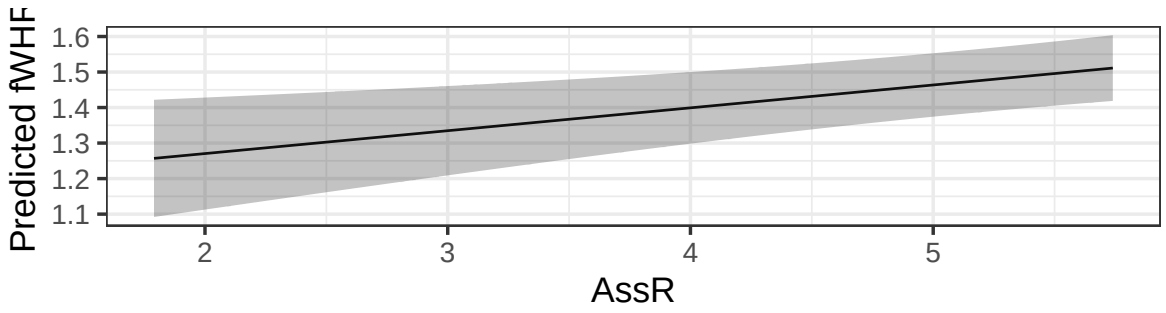

## 1.9 Why are we doing this again?

Why make prediction plots?

## 1.10 Shortcut Method - With Uncertainty

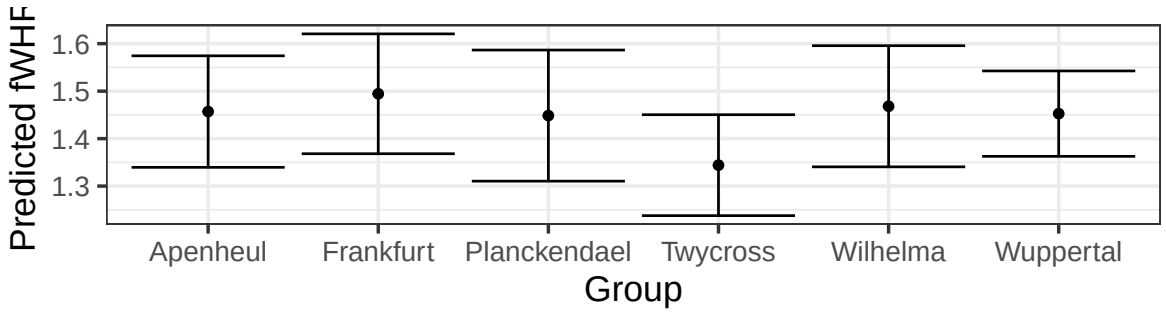
We saw before that `pred_plot()` makes it very easy for us to generate prediction plots showing what a (multiple regression) model says about the relationship between the response and *one* of the predictors:

```
library(s245)
pred_plot(mlr3, 'AssR') |>
  gf_labs(y = 'Predicted fWHR')
```




*Note the custom axis label - otherwise you get a long, unwieldy default "Predictions from fitted model"*

```
library(s245)
pred_plot(mlr3, 'Group') |>
  gf_labs(y = 'Predicted fWHR')
```




They look nice! But they should raise two questions:

- Uncertainty:

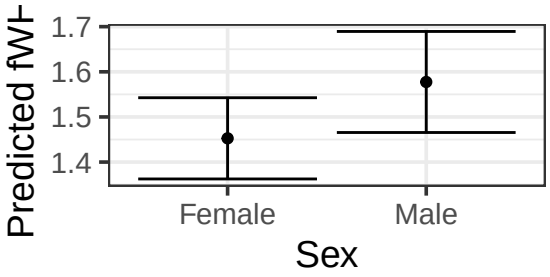
- Fixed values:

```
get_fixed(bonobos)  |>
  pander::pander()
```

Quitting from lines 594-596 [unnamed-chunk-18] (lm1.rmarkdown) Error in `if (class(data[, v]) == "character") ...`: ! the condition has length > 1 Backtrace: 1. pander::pander(get_fixed(bonobos)) 2. s245::get_fixed(bonobos)

### 1.10.1 Anatomy of a Confidence Interval

```
pred_plot(mlr3, 'Sex') |>
  gf_labs(y = 'Predicted fWHR')
```




## 1.11 DIY Method

### 1.11.1 Creating a hypothetical dataset

We would like to create a hypothetical dataset where one predictor variable varies, and all the rest stay fixed. Let's choose `AssR`. We use `expand.grid()`:

```
fake_data <- expand.grid(AssR = seq(from=1.8, to=5.7, by=0.05),
                         weight = 40,
                         Sex = 'Female',
                         Group = 'Twycross')
glimpse(fake_data)
```

```
Rows: 79
Columns: 4
```

```
$ AssR   <dbl> 1.80, 1.85, 1.90, 1.95, 2.00, 2.05, 2.10, 2.15, 2.20, 2.25, 2.3~
$ weight <dbl> 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,~
$ Sex    <fct> Female, Female, Female, Female, Female, Female, Female, Female,~
$ Group  <fct> Twycross, Twycross, Twycross, Twycross, Twycross, Twycross, Twy~
```

Now, make predictions for our fake data.

```
preds <- predict(mlr3, newdata = fake_data, se.fit = TRUE)
fake_data <- fake_data |>
  mutate(fitted = preds$fit,
         se.fit = preds$se.fit)
glimpse(fake_data)
```

```
Rows: 79
Columns: 6
$ AssR   <dbl> 1.80, 1.85, 1.90, 1.95, 2.00, 2.05, 2.10, 2.15, 2.20, 2.25, 2.3~
$ weight <dbl> 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,~
$ Sex    <fct> Female, Female, Female, Female, Female, Female, Female, Female,~
$ Group  <fct> Twycross, Twycross, Twycross, Twycross, Twycross, Twycross, Twy~
$ fitted <dbl> 1.149038, 1.152256, 1.155474, 1.158692, 1.161910, 1.165128, 1.1~
$ se.fit <dbl> 0.08347207, 0.08267088, 0.08187552, 0.08108616, 0.08030298, 0.0~
```

How do we go from *standard errors* to *confidence intervals*? We can either do this before plotting, or while plotting. To do it before and add the results to the hypothetical dataset:

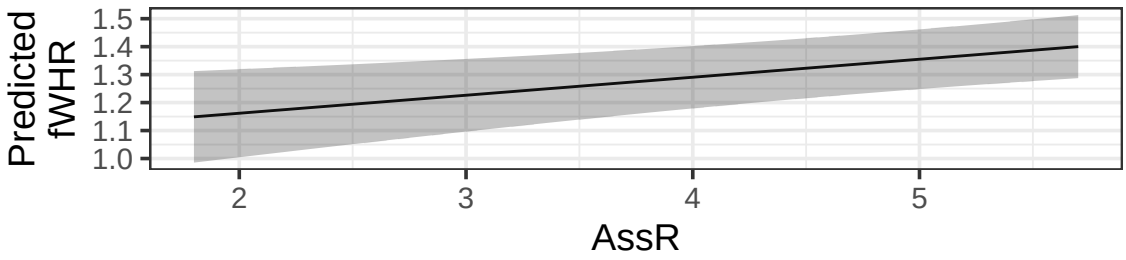
```
fake_data <- fake_data |>
  mutate(CI_lower = fitted - 1.96*se.fit,
         CI_upper = fitted + 1.96*se.fit)
glimpse(fake_data)
```

```
Rows: 79
Columns: 8
$ AssR     <dbl> 1.80, 1.85, 1.90, 1.95, 2.00, 2.05, 2.10, 2.15, 2.20, 2.25, 2~
$ weight   <dbl> 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 4~
$ Sex      <fct> Female, Female, Female, Female, Female, Female, Female, Femal~
$ Group    <fct> Twycross, Twycross, Twycross, Twycross, Twycross, Twycross, T~
$ fitted   <dbl> 1.149038, 1.152256, 1.155474, 1.158692, 1.161910, 1.165128, 1~
$ se.fit   <dbl> 0.08347207, 0.08267088, 0.08187552, 0.08108616, 0.08030298, 0~
$ CI_lower <dbl> 0.9854326, 0.9902210, 0.9949980, 0.9997632, 1.0045164, 1.0092~
$ CI_upper <dbl> 1.312643, 1.314291, 1.315950, 1.317621, 1.319304, 1.321000, 1~
```

### 1.11.2 Making the plot

Now, we just need to plot!

```
gf_line(fitted ~ AssR, data=fake_data) |>
  gf_labs(y='Predicted\nfWHR') |>
  gf_ribbon(CI_lower + CI_upper ~ AssR, data = fake_data)
```




If we wanted to figure out the CI bounds *while* plotting, we could calculate them on the fly like this:

```
gf_line(fitted ~ AssR, data=fake_data) |>
  gf_labs(y='Predicted\nfWHR') |>
  gf_ribbon((fitted - 1.96*se.fit ) + (fitted + 1.96*se.fit) ~ AssR,
            data = fake_data)
```

(which will look just the same).

### 1.11.3 Categorical predictors

What will be different if the predictor of interest is *categorical*?

- hypothetical data:
- plot:

```
fake_sex_data <- expand.grid(AssR = 4.51,
                             weight = 40,
                             Sex = c('Male', 'Female'),
                             Group = 'Twycross')
preds <- predict(mlr3, newdata = fake_sex_data, se.fit = TRUE)
fake_sex_data <- fake_sex_data |>
```

```
  mutate(fitted = preds$fit,
         se.fit = preds$se.fit)
gf_point(fitted ~ Sex, data=fake_sex_data) |>
  gf_labs(y='Predicted fWHR') |>
  gf_errorbar((fitted - 1.96*se.fit ) + (fitted + 1.96*se.fit) ~ Sex,
              data = fake_sex_data)
```

# 2 Model Selection Using Information Criteria

So far, we have learned to fit models with multiple predictors, both quantitative and categorical, and to assess whether required conditions are met for linear regression to be an appropriate model for a dataset.

One missing piece is: If I have an appropriate model with a set of multiple predictors, how can I choose which predictors are worth retaining in a "best" model for the data (and which ones have no relationship, or a weak relationship, with the response, so should be discarded)?

## 2.1 Data and Model

Today we will recreate part of the analysis from *Vertebrate community composition and diversity declines along a defaunation gradient radiating from rural villages in Gabon*, by Sally Koerner and colleagues. They investigated the relationship between rural villages, hunting, and wildlife in Gabon. They asked how monkey abundance depends on distance from villages, village size, and vegetation characteristics. They shared their data at Dryad.org and we can read it in and fit a regression model like this:

```
defaun <- read.csv('http://sldr.netlify.com/data/koerner_gabon_defaunation.csv')
```

```
ape_mod <- lm(RA_Apes ~ Veg_DBH + Veg_Canopy + Veg_Understory +
                  Veg_Rich + Veg_Stems + Veg_liana +
                  LandUse + Distance + NumHouseholds, data = defaun)
summary(ape_mod)
```

```
Call:
lm(formula = RA_Apes ~ Veg_DBH + Veg_Canopy + Veg_Understory +
    Veg_Rich + Veg_Stems + Veg_liana + LandUse + Distance + NumHouseholds,
    data = defaun)

Residuals:
    Min      1Q  Median      3Q     Max
-3.9857 -0.9419 -0.0360  0.8239  6.3832
```

```
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     5.752517  13.372210   0.430   0.6741
Veg_DBH        -0.093171   0.073114  -1.274   0.2249
Veg_Canopy      0.670094   2.062545   0.325   0.7504
Veg_Understory -1.691235   2.071299  -0.817   0.4289
Veg_Rich        0.361960   0.480362   0.754   0.4646
Veg_Stems      -0.097211   0.169073  -0.575   0.5751
Veg_liana      -0.158505   0.253031  -0.626   0.5419
LandUseNeither  1.696755   2.058937   0.824   0.4247
LandUsePark    -2.947189   2.222710  -1.326   0.2077
Distance        0.302626   0.119865   2.525   0.0254 *
NumHouseholds  -0.002107   0.043458  -0.048   0.9621
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.725 on 13 degrees of freedom
Multiple R-squared:  0.5439,    Adjusted R-squared:  0.1931
F-statistic: 1.551 on 10 and 13 DF,  p-value: 0.2262
```

```
as.numeric(logLik(ape_mod))
```

```
[1] -50.75799
```

## 2.2 Calculations

- Information criteria allow us to **balance the conflicting goals** of having a model that *fits the data as well as possible* (which pushes us toward models with more predictors) and *parsimony* (choosing the simplest model, with the fewest predictors, that works for the data and research question). The basic idea is that we **minimize** the quantity $-(2LogLikelihood - penalty) = -2LogLikelihood + penalty$
- AIC is computed according to $-2LogLikelihood + 2k$, where $k$ is the number of coefficients being estimated (don't forget $\sigma$!) **Smaller AIC is better.**
- BIC is computed according to $-2LogLikelihood + ln(n)k$, where $n$ is the number of observations (rows) in the dataset and $k$ is the number of coefficients being estimated. **Smaller BIC is better.**
- Verify that the BIC for this model is 139.65.

## 2.3 Decisions with ICs

The following rules of thumb (**not** laws, just common rules of thumb) may help you make decisions with ICs:

- A model with lower IC *by at least 3 units* is notably better.
- If two or more models have ICs *within* 3 IC units of each other, there is not a lot of difference between them. Here, we usually choose the model with fewest predictors.
- In some cases, if the research question is to measure the influence of some particular predictor on the response, but *the IC does not strongly support including that predictor* in the best model (IC difference less than 3), you might want to keep it in anyway and then discuss the situation honestly, for example, "AIC does not provide strong support for including predictor x in the best model, but the model including predictor x indicates that as x increases the response decreases slightly. More research would be needed…"

## 2.4 All-possible-subsets Selection

The model we just fitted is our *full model*, with all predictors of potential interest included. How can we use information criteria to choose the best model from possible models with subsets of the predictors?

We can use the `dredge()` function from the `MuMIn` package to get and display ICs for all these models.

Before using dredge, we need to make sure our dataset has no missing values, and also set the "na.action" input for our model (can be done in call to `lm(..., na.action = 'na.fail')` also).

```
library(MuMIn)
ape_mod <- ape_mod |> update(na.action = 'na.fail')
ape_dredge <- dredge(ape_mod, rank='BIC')
```

```
Fixed term is "(Intercept)"
```

```
pander::pander(head(ape_dredge, 7))
```

Table 2.1: Table continues below

| | (Intercept) | Distance | LandUse | NumHouseholds | Veg_Canopy |
|---|---|---|---|---|---|
| **258** | 8.753 | 0.195 | NA | NA | NA |
| **2** | -0.6912 | 0.2303 | NA | NA | NA |
| **274** | 11.44 | 0.1848 | NA | NA | NA |
| **322** | 11.9 | 0.2033 | NA | NA | NA |
| **290** | 9.805 | 0.1884 | NA | NA | NA |
| **386** | 9.49 | 0.1976 | NA | NA | NA |
| **266** | 7.783 | 0.1896 | NA | NA | 0.2771 |

Table 2.2: Table continues below

| | Veg_DBH | Veg_liana | Veg_Rich | Veg_Stems | Veg_Understory | df |
|---|---|---|---|---|---|---|
| **258** | NA | NA | NA | NA | -2.988 | 4 |
| **2** | NA | NA | NA | NA | NA | 3 |
| **274** | -0.04551 | NA | NA | NA | -3.144 | 5 |
| **322** | NA | NA | -0.1939 | NA | -3.11 | 5 |
| **290** | NA | -0.09802 | NA | NA | -2.952 | 5 |
| **386** | NA | NA | NA | -0.03113 | -2.904 | 5 |
| **266** | NA | NA | NA | NA | -2.964 | 5 |

| | logLik | BIC | delta | weight |
|---|---|---|---|---|
| **258** | -53.9 | 120.5 | 0 | 0.3284 |
| **2** | -55.8 | 121.1 | 0.6241 | 0.2404 |
| **274** | -53.38 | 122.7 | 2.146 | 0.1123 |
| **322** | -53.55 | 123 | 2.491 | 0.09449 |
| **290** | -53.67 | 123.2 | 2.727 | 0.08399 |
| **386** | -53.82 | 123.5 | 3.03 | 0.0722 |
| **266** | -53.88 | 123.7 | 3.144 | 0.0682 |

- What is the best model according to BIC, for this dataset?

## 2.5 Which IC should I use?

AIC and BIC may give different best models, especially if the dataset is large. You may want to just choose one to use *a priori* (before making calculations). You might prefer BIC if you want to err on the "conservative" side, as it is more likely to select a "smaller" model with fewer predictors. This is because of its larger penalty.

## 2.6 Quantities derived from AIC

- $\Delta AIC$ is the AIC for a given model, minus the AIC of the best one in the dataset. (Same for $\Delta BIC$)
- *Akaike weights* are values (ranging from 0-1) that measure the weight of evidence suggesting that a model is the best one (given that there is one best one in the set)

## 2.7 Important Caution

**Very important**: IC can **ONLY** be compared for models with the same response variable, and the exact same rows of data.

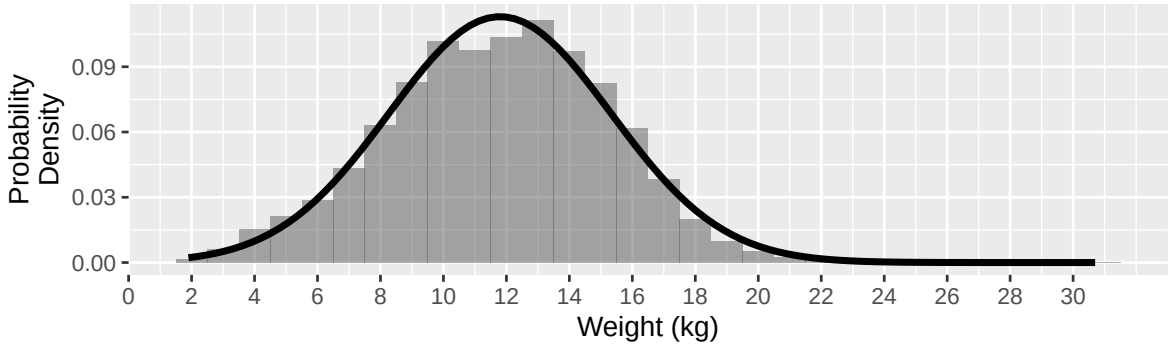
# 3 Likelihood

In the last section, we said that "likelihood" is a measure of goodness-of-fit of a model to a dataset. But what is it *exactly* and just how do we compute it?

## 3.1 Data

Today's dataset was collected in Senegal in 2015-2016 in a survey carried out by UNICEF, of 5440 households in the urban area of Dakar, Senegal. Among these households, information was collected about 4453 children under 5 years old, including their **weights in kilograms.**

```
gf_dhistogram(~AN3, data=wt, binwidth=1) |>
  gf_labs(x='Weight (kg)', y='Probability\nDensity') |>
  gf_fitdistr(dist='dnorm', size=1.3) |>
  gf_refine(scale_x_continuous(breaks=seq(from=0, to=30, by=2)))
```




## 3.2 Review - the Normal probability density function (PDF)

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

## 3.3 A simple model

The distribution of weights looks quite unimodal and symmetric, so we will model it with a normal distribution with mean 11.8 and standard deviation 3.53 (N( $\mu = 11.8$, $\sigma = 3.53$), black line).

## 3.4 Using the Model to Make Predictions

If you had to predict the weight of one child from this population, what weight would you guess?

Is it more likely for a child in Dakar to weigh 10kg, or 20kg? How much more likely?

What is the *probability* of a child in Dakar weighing 11.5 kg?

## 3.5 Likelihood to the Rescue!

Which is more likely: three children who weigh 11, 8.2, and 13kg, or three who weigh 10, 12.5 and 15 kg?

How did you:

- Find the likelihood of each observation?
- Combine the likelihoods of a set of three observations?

What did you have to assume about the set of observations?

## 3.6 How does this relate to linear regression?

What if we think of this situation as a linear regression problem (with no predictors)?

```
lm_version <- lm(AN3 ~ 1, data = wt)
summary(lm_version)
```

```
Call:
lm(formula = AN3 ~ 1, data = wt)

Residuals:
    Min      1Q  Median      3Q     Max
-9.8964 -2.3964  0.1036  2.4036 18.9036

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 11.79644    0.05435   217.1   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.529 on 4216 degrees of freedom
  (235 observations deleted due to missingness)
```

### 3.6.1 Model Equation:

## 3.7 Likelihood of a dataset, given a model

Finally, now, we can understand what we were computing when we did

```
logLik(lm_version)
```

```
'log Lik.' -11301.19 (df=2)
```

For our chosen regression model, we know that the residuals should have a normal distribution with mean 0 and standard deviation $\sigma$ (estimated Residual Standard Error from R `summary()` output).

For each data point in the dataset, for a given regression model, we can compute a model prediction.

We can subtract the prediction from the observed response-variable values to get the residuals.

We can compute the **likelihood** ($L$) of this set of residuals by finding the likelihood of each individual residual $e_i$ in a $N(0, \sigma)$ distribution.

To get the likelihood of the full dataset given the model, we use the fact that the residuals are independent (they better be, because that was one of the conditions of of linear regression

model) – we can multiply the likelihoods of all the individual residuals together to get the joint likelihood of the full set.

*That* is the "likelihood" that is used in the AIC and BIC calculations we considered earlier.

# 4 PDFs and PMFs

## 4.1 Beyond Normal

In our exploration of likelihoods, we did a little bit of work with the Normal probability density function. Here, we will state some characteristics of the normal distribution slightly more formally, and then we will get familiar with some other probability distributions (a.k.a. "the normal distribution's wierd friends").

## 4.2 Types of probability distributions

### 4.2.1 Continuous distributions

The probability distribution of a continuous variable (one that can take on continuous real values, at least within a certain range) is called a *probability density function* or **PDF** for short.

PDFs are functions of one continuous variable (we'll call it $x$) that have two properties in common:

- The total area under the curve is 1 ($\int_{-\infty}^{\infty} f(x)dx = 1$)
- The values of the function are non-negative (0, or a positive real number) for all real $x$ ($f(x) \geq 0 \forall x \in \mathbb{R}$)

For PDFs, the function values ($y$-axis values) are *probability densities*, useful for computing likelihoods; probabilities are given by finding *areas* under the curve.

### 4.2.2 Discrete Distributions

We use categorical as well as quantitative variables in our regression models, so it will prove useful to have some discrete probability distributions as well as continuous ones. Discrete distributions associate each possible value of a discrete variable with its probability of occurence.

A discrete probability distribution is characterized by a *probability mass function* or PMF for short (this is the discrete equivalent of a PDF).

A PMF is a function of one *discrete* variable (we'll call it $x$) that have two properties in common:

- The sum of all the function's values is 1 ($\sum_{x \in S} f(x) = 1$, where $S$ is the set of all possible values $x$ can have)
- All values of the function are between 0 and 1 inclusive (they are probabilities) ($f(x) \in [0, 1] \forall x \in S$)

## 4.3 Relevant Features of Distributions

For this course, the most important features to note about each probability distribuiton will be:

- The **type** of distribution: discrete or continuous?
- The **support** of the distribution: what range of possible values can the random variable $X$ take on?
- The **parameters** of the distribution. Changing the parameters tunes the center and shape of the distribution, so these are what we need to estimate to fit a particular kind of distribution to a specific dataset. (For example, the parameters of the normal distribution are the mean $\mu$ and the standard deviation $\sigma$.)
- The **shape** of the distribution: what shapes can the function take one? (For example, the normal distribution is always unimodal and symmetric.)
- The PDF or PMF of the distribution. What mathematical expression controls the shape of the distribution?
- We will also note a few examples of variables that might be well modelled by each distribution.

## 4.4 Examples of Continuous Distributions

### 4.4.1 Normal

#### 4.4.1.0.1 Type

Continuous

#### 4.4.1.0.2 Support

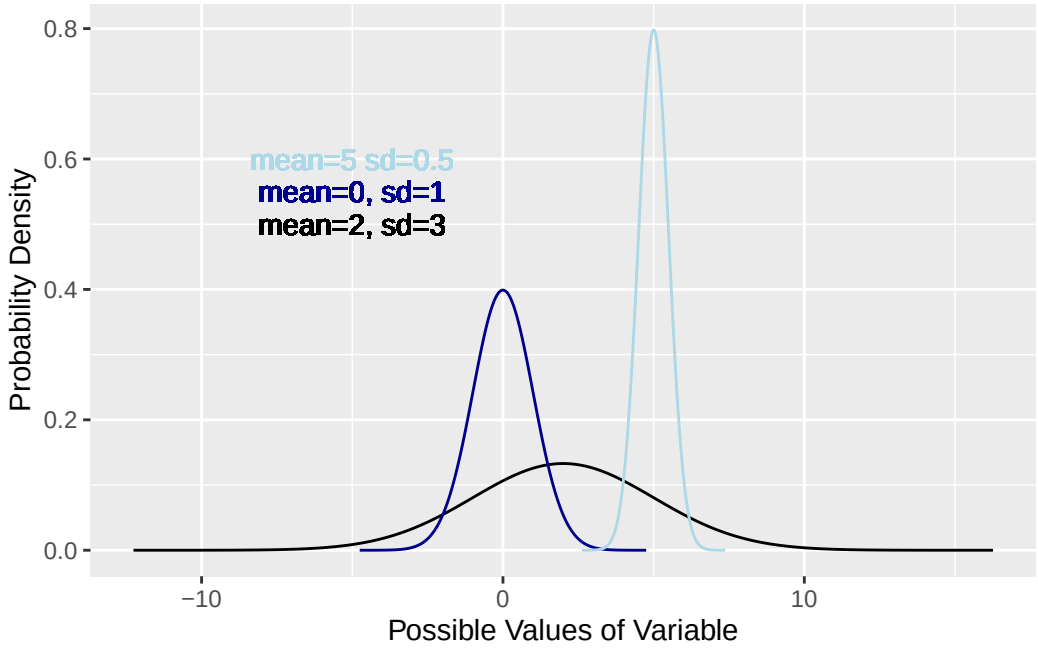
All real numbers

#### 4.4.1.0.3 Parameters

- $\mu$, the mean, which can take on any real value
- $\sigma$, the standard deviation, which can take on any positive real value

#### 4.4.1.0.4 Shapes

The shape is always unimodal and symmetric.




#### 4.4.1.0.5 PDF or PMF

The normal distribution has PDF:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

#### 4.4.1.0.6 Examples

A normal distribution might be a good fit for data on childrens' weights in kg, or for the duration of visits at a zoo, or...

### 4.4.2 Gamma

#### 4.4.2.0.1 Type

Continuous

#### 4.4.2.0.2 Support

positive real numbers

#### 4.4.2.0.3 Parameters

There are two alternate but equivalent parameterizations for the gamma distribution.

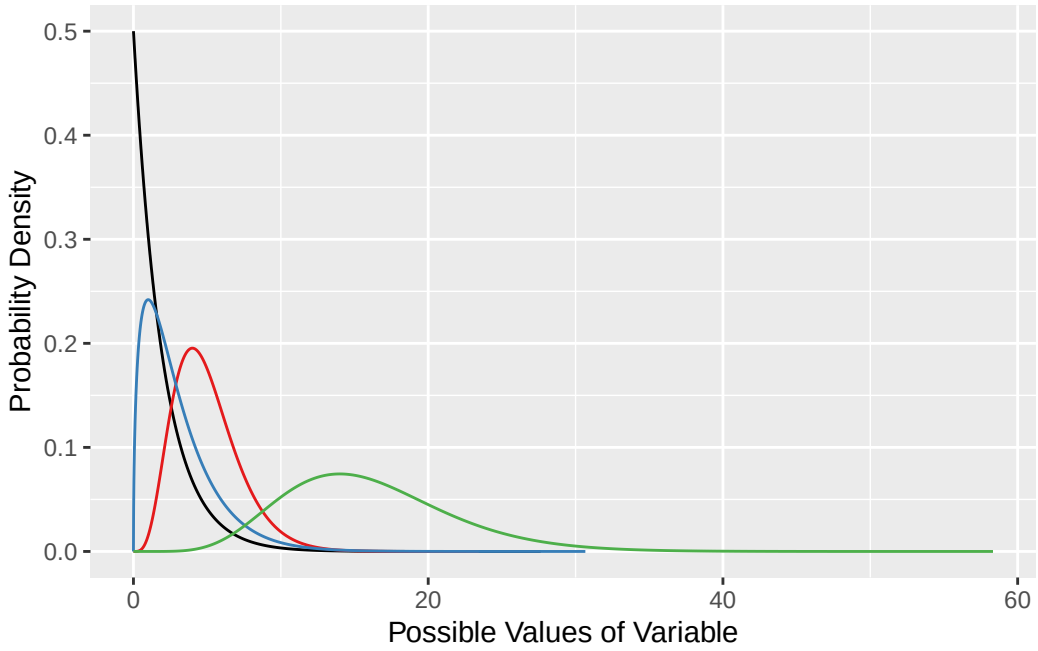
The first option: $\alpha$ (shape) and $\beta$ (rate), where $\alpha > 0$ and $\beta > 0$

The second option: $k$ (shape) and $\theta$ (scale), where $k > 0$ and $\theta > 0$.

Converting between the two parameterizations: $\alpha = k$ and $\beta = \frac{1}{\theta}$.

#### 4.4.2.0.4 Shapes

The gamma distribution can take on a unimodal, symmetric shape or a unimodal shape with any amount of right skew (up to an exponential distribution shape).



Note: you don't need to be familiar with exactly how the different values of parameters influence the shape of the gamma distribution PDF, so the curves here are not labelled with parameter values.

#### 4.4.2.0.5 PDF or PMF

The gamma distribution has PDF:

$$f(x) = \frac{1}{\Gamma(k)\theta^k} x^{(k-1)} e^{\frac{-x}{\theta}}$$

Where $\Gamma$ is the Gamma function (look up the definition if you choose), or equivalently,

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{(\alpha-1)} e^{-\beta x}$$

#### 4.4.2.0.6 Examples

Gamma distributions are often used to model things like wind speed or duration of an event (any quantity that might have right skew and is never negative).

### 4.4.3 Beta

#### 4.4.3.0.1 Type

Continuous

#### 4.4.3.0.2 Support

Real numbers between 0 and 1 ([0,1])

#### 4.4.3.0.3 Parameters

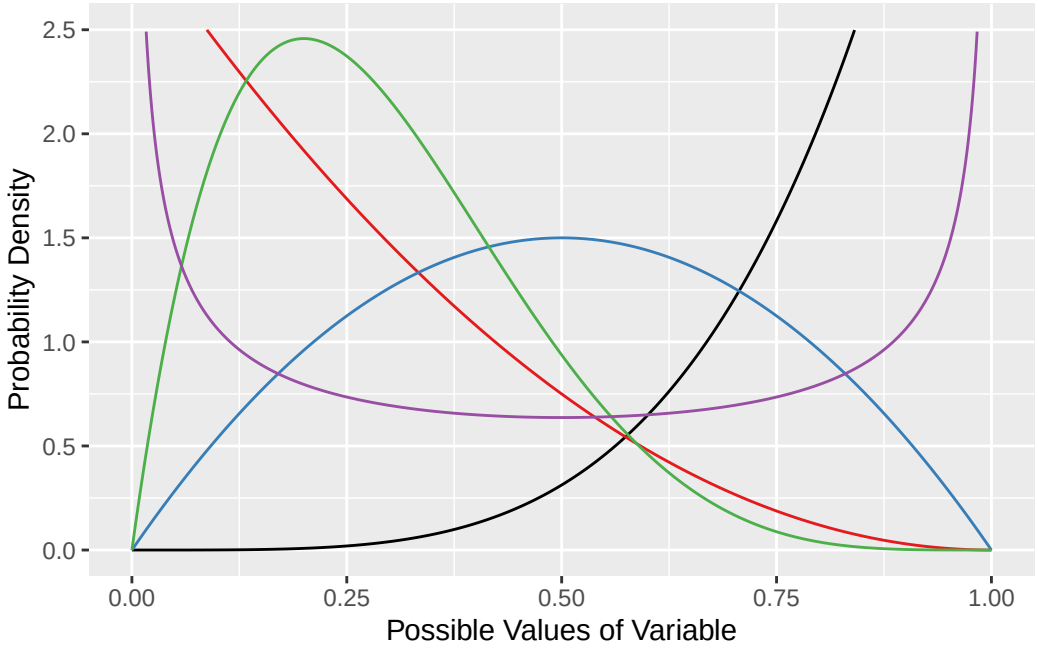
$\alpha$ (shape 1) and $\beta$ (shape 2), both of which must be $> 1$.

#### 4.4.3.0.4 Shapes

This distribution can take on almost any shape, for example:

```
gf_dist('beta', params = c(shape1=5, shape2=1)) |>
  gf_dist('beta', params = c(shape1=1, shape2=3), color=colrs[1]) |>
  gf_dist('beta', params = c(shape1=2, shape2=2), color=colrs[2]) |>
```

```
gf_dist('beta', params = c(shape1=2, shape2=5), color=colrs[3]) |>
gf_dist('beta', params = c(shape1=0.5, shape2=0.5), color=colrs[4]) |>
gf_labs(x='Possible Values of Variable', y='Probability Density') |>
gf_lims(y=c(0,2.5))
```




#### 4.4.3.0.5 PDF or PMF

The PDF is:

$$f(x) = \frac{x^{(\alpha-1)}(1-x)^{(\beta-1)}}{B(\alpha,\beta)}$$

Where $B$ is the Beta function (again, feel free to look up the definition if you are interested).

#### 4.4.3.0.6 Examples

Beta distributions could be used for any variable that takes on values between 0-1, for example, baseball players' batting averages, or test scores (as proportions).

## 4.5 Examples of Discrete Distributions

### 4.5.1 Binomial

#### 4.5.1.0.1 Type

Discrete

#### 4.5.1.0.2 Support

You can think about the support of this distribution two ways. Technically, the support is $k = 0, 1, 2, 3, ...$: 0 and positive integers, interpreted as the number of "successes" in $n$ binomial trials. Binomial trials are independent observations of a process that has two possible outcomes, "success" or "failure", with set probabilities of each occurring. (And probabilities of success and failure must sum to 1.)

So, for each individual binomial trial, the possible outcomes are 0 and 1, often interpreted as TRUE and FALSE or "success" and "failure".

For our purposes, this distribution will be useful for modelling response variables that are categorical with two possible values (and the $n$ trials will be the $n$ rows in our dataset).

#### 4.5.1.0.3 Parameters

Parameters are $n$, the number of independent trials, and $p$, the probability of success in each trial.

#### 4.5.1.0.4 Shapes

The figure below shows the shape of binomial distributions with fixed $n = 100$ and varying $p$:

```
gf_dist('binom', params = c(size=100, prob=0.1), color=colrs[1]) |>
  gf_dist('binom', params = c(size=100, prob=0.25), color=colrs[2]) |>
  gf_dist('binom', params = c(size=100, prob=0.5), color=colrs[3]) |>
  gf_dist('binom', params = c(size=100, prob=0.75), color=colrs[4]) |>
  gf_dist('binom', params = c(size=100, prob=0.99), color=colrs[5]) |>
  gf_labs(x='k, the Number of Successes in 100 Trials', y='Probability')
```


The $p$ used were 0.1, 0.25, 0.5, 0.75, and 0.99. Can you tell which is which?

#### 4.5.1.0.5 PDF or PMF

The PMF for the binomial distribution is:

$$P(X = k|n, p) = \binom{n}{k} p^k (1-p)^{n-k}$$

Where $k$ is the number of successes observed in $n$ trials (you can think of $k$ as our "x-axis variable" for this PMF).

#### 4.5.1.0.6 Examples

We might use this distribution to model any categorical variable with two possible values, like Age (if possible values are "adult" and "child") or health status ("has disease" or "does not have disease"). We'll think of each observation in the dataset as one of the $n$ indpendent trials, with one of two possible outcomes for each trial.

### 4.5.2 Poisson

#### 4.5.2.0.1 Type

Discrete

#### 4.5.2.0.2 Support

The support is 0 and positive integers (i.e., this distribution works well for count data).

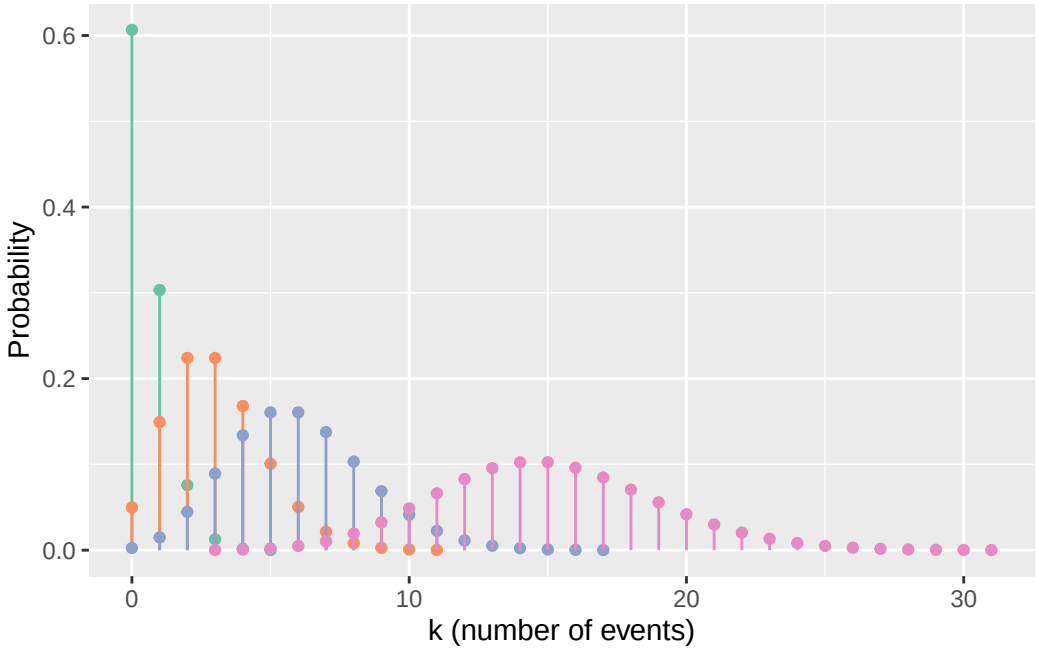
#### 4.5.2.0.3 Parameters

The Poisson distribution has one parameter, $\lambda$ (the event rate per unit time) which must be greater than 0.

#### 4.5.2.0.4 Shapes

The distribution can take on unimodal shapes with varying amounts of right skew (from none, to an exponential shape).

```
colrs <- RColorBrewer::brewer.pal(8, 'Set2')
gf_dist('pois', params=c(lambda=0.5), color=colrs[1]) |>
  gf_dist('pois', params=c(lambda=3), color=colrs[2]) |>
  gf_dist('pois', params=c(lambda=6), color=colrs[3]) |>
  gf_dist('pois', params=c(lambda=15), color=colrs[4]) |>
  gf_labs(x='k (number of events)', y='Probability')
```



#### 4.5.2.0.5 PDF or PMF

The Poisson PMF is:

$$P(X = k|\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

#### 4.5.2.0.6 Examples

The Poisson distribution might be used to model any response variable that is comprised of counts, for example, the number of birds sighted in a bird survey, or the number of people admitted to an emergency room each hour.

### 4.5.3 Negative Binomial

There are two versions or "types" of this distribution, cleverly known as NB1 (type 1) and NB2 (type 2). NB1 has "constant overdispersion" – the variance of the distribution is greater than the mean according to a constant ratio. NB2 has "variable overdispersion" – the variance is a quadratic function of the mean. The NB2 is the one that corresponds directly to conceptualization in terms of binomial trials (with the PMF giving the probability of observing $y$ failures before the $r$th success). Hardin and Hilbe 2007 describe the negative binomial this way: "Instead of counts entering uniformly, we see counts entering with a specific gamma-distributed shape."

#### 4.5.3.0.1 Type

Discrete

#### 4.5.3.0.2 Support

The support is 0 and positive integers (i.e., this distribution works well for count data). It also has a derivation in terms of binomial trials, but in our regression models, we will only use it with count data.

#### 4.5.3.0.3 Parameters

A common parameterization of the negative binomial (online and in actuarial science) has parameters $p$, the probability of success on each binomial trial, and $r$, the number of failures observed. The PMF then gives the probability of observing $k$ failures before the $r$th success in a series of Bernoulli trials.

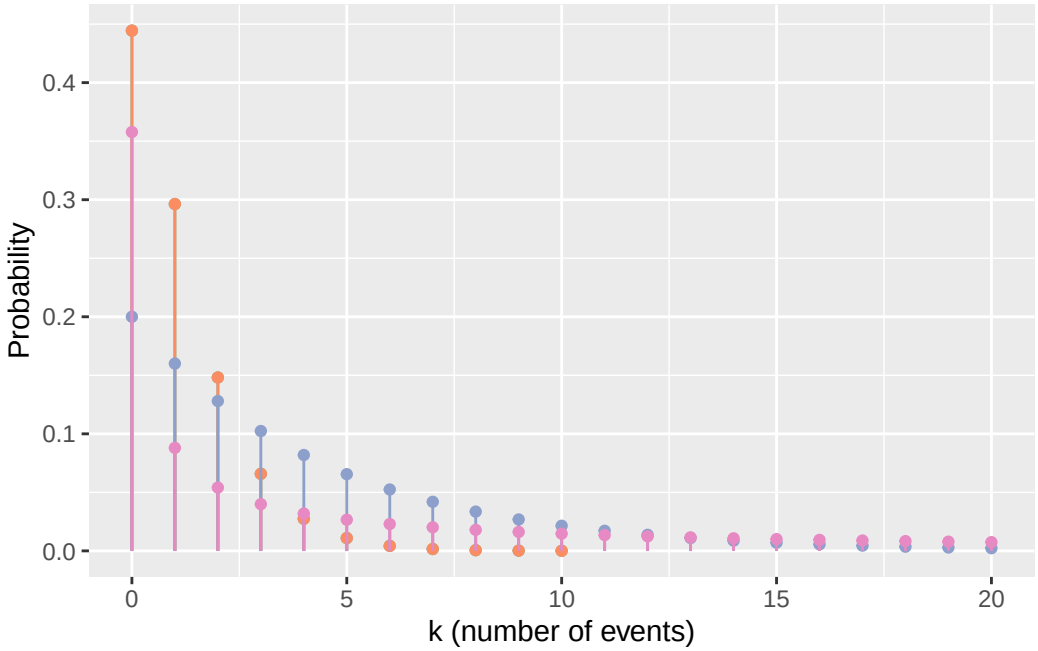
Here, we will use an alternate parameterization. The other common way to parameterize and derive the NB is as a Poisson-gamma mixture – a modified version of a Poisson distribution. In this scheme, the parameters of the distribution are $\mu$ and $\alpha$.

#### 4.5.3.0.4 Shapes

These distributions can take on unimodal shapes with varying amounts of right skew. In NB2 (type 2) distributions the variance (spread) is larger relative to the mean.

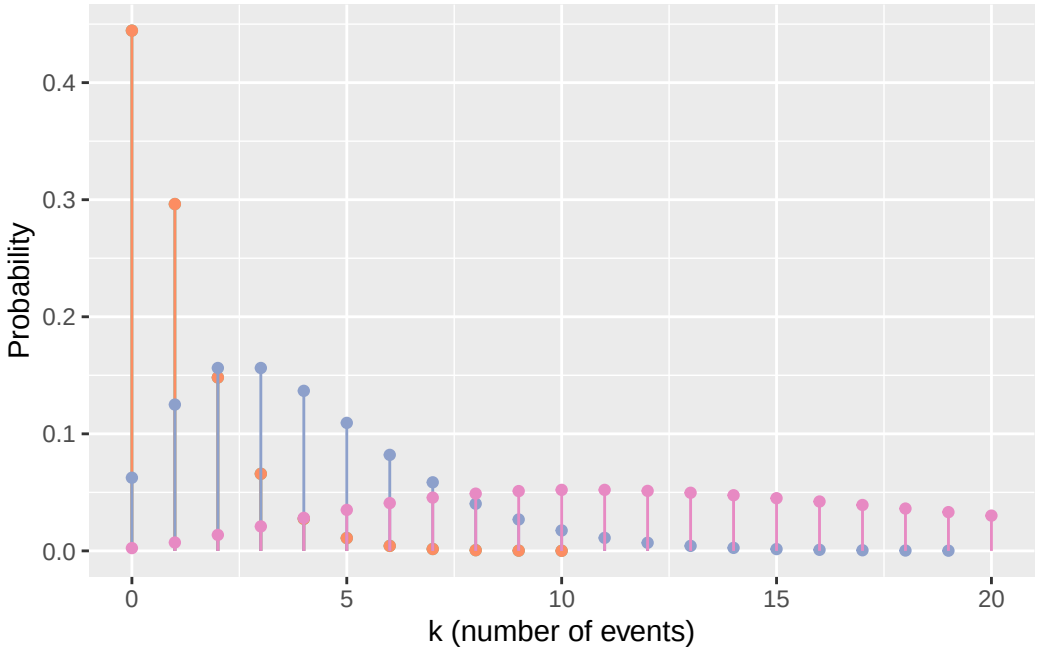
NB1:

```
library(gamlss.dist)
colrs <- RColorBrewer::brewer.pal(8, 'Set2')
gf_dist('NBI', params=c(mu=1, sigma=0.5), color=colrs[1]) |>
  gf_dist('NBI', params=c(mu=1, sigma=0.5), color=colrs[2]) |>
  gf_dist('NBI', params=c(mu=4, sigma=1), color=colrs[3]) |>
  gf_dist('NBI', params=c(mu=15, sigma=4), color=colrs[4]) |>
  gf_lims(x=c(0,20)) |>
  gf_labs(x='k (number of events)', y='Probability')
```




NB2:

```
colrs <- RColorBrewer::brewer.pal(8, 'Set2')
gf_dist('NBII', params=c(mu=1, sigma=0.5), color=colrs[1]) |>
  gf_dist('NBII', params=c(mu=1, sigma=0.5), color=colrs[2]) |>
  gf_dist('NBII', params=c(mu=4, sigma=1), color=colrs[3]) |>
  gf_dist('NBII', params=c(mu=15, sigma=4), color=colrs[4]) |>
  gf_lims(x=c(0,20)) |>
  gf_labs(x='k (number of events)', y='Probability')
```




#### 4.5.3.0.5 PDF or PMF

Details of the parameterizations and likelihood and fitting of NB1 and NB2 distributions can be found in Hardin and Hilbe 2007, if you are interested.

The PMF for the NB1, where the variance is a constant multiple of the mean, is:

$$f(x|\mu,\alpha) = \frac{\Gamma(x+\mu)}{\Gamma(\mu)\Gamma(x+1)}(\frac{1}{1+\alpha})^{\mu}(\frac{\alpha}{1+\alpha})^{x}$$

Where $\Gamma$ is a Gamma function. Note that if $\alpha = 0$ this becomes a Poisson distribution, so the Poisson is a special case of the NB1.

The PMF for the NB2, where the variance is a quadratic function of the mean, is:

$$f(x|\mu,\alpha) = \frac{\Gamma(x+\frac{1}{\alpha})}{\Gamma(\frac{1}{\alpha})\Gamma(x+1)}(\frac{1}{1+\alpha\mu})^{\frac{1}{\alpha}}(1-\frac{1}{1+\alpha\mu})^{x}$$

#### 4.5.3.0.6 Examples

NB distributions are good models for overdispersed count data, where (in the regression context) the residual variance is not equal to the expected (predicted) value. (Note that if you are reading this before learning about regression models for count data, you may not understand

this sentence yet…don't worry, it will make sense when you return later!) Some examples might include sightings data on numbers of animals seen on wildlife surveys, or the number of items bought per order at an online retailer.

### 4.5.4 A Mixture Distribution: Tweedie

The Tweedie family of distributions is a very large one - depending on the values of the different parameters, the PMF/PDF can be written in many different ways, and it can take on many different shapes. The description below is a simplified one, geared toward the types of Tweedie distributions we are likely to try to use in regression models in this course – mainly the "compound Poisson-gamma" type.

*Some extra resources for which you will not be held responsible in this course:*

- You can find an accessible description and example of this kind of distribution at: http://www.notenoughthoughts.net/posts/modeling-activity.html.
- The following site may also be useful in regard to using the Tweedie in regression models: http://support.sas.com/documentation/cdl/en/statug/68162/HTML/default/viewer.htm#statug_genmod_details28.htm

#### 4.5.4.0.1 Type

These distributions are *both* continuous and discrete - a kind of mix of a Poisson distribution and gamma distribution(s).

#### 4.5.4.0.2 Support

The support is non-negative real numbers (greater than or equal to 0).

#### 4.5.4.0.3 Parameters

(Note: there are multiple different ways to parameterize these distributions.) We will use:

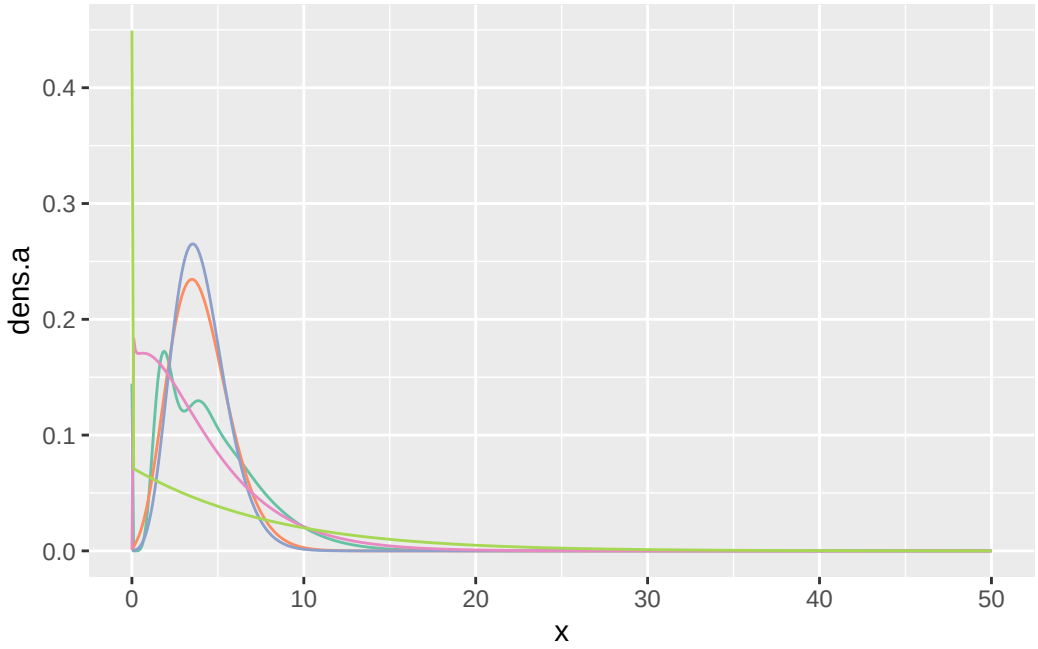
- $p$, the index or power parameter, which can be 0 (resulting in a normal distribution), 1 (resulting in a Poisson distribution), $1 < p < 2$ (a compound Poisson-gamma distribution – what we will mainly use), 2 (a gamma distribution), 3 (an inverse Gaussian distribution), $< 3$ (a positive stable distribution), or $\infty$ (an extreme positive stable distribuiton). For the case where $1 < p < 2$, and the distribution is a compound of a Poisson and a gamma, then $p = \frac{k+2}{k+1}$ where $k$ is the parameter of the gamma distribution. When $1 < p < 2$, $p$ closer to 1 means thant the Poisson distribution (the mass at 0) gets more "weight" in the compound distribution, and values of $p$ closer to 2 mean that the gamma distribution gets more "weight."

- $\mu$. For the case where $1 < p < 2$, and the distribution is a compound of a Poisson and a gamma, then $\mu = \lambda k\theta$ where $\lambda$ is the parameter of the Poisson distribution and $k$ and $\theta$ are the parameters of the gamma.
- $\phi$ For the case where $1 < p < 2$, and the distribution is a compound of a Poisson and a gamma, then $\phi = \frac{\lambda^{(1-p)}(k\theta)^{(2-p)}}{2-p}$ where $\lambda$ is the parameter of the Poisson distribution and $k$ and $\theta$ are the parameters of the gamma.

#### 4.5.4.0.4 Shapes

A compound Poisson-gamma Tweedie distribution can take on varying shapes; the main characteristic of interest for us is that it can have a mass at 0, then a unimodal or multimodal distribution with a long right tail (lots of right skew).

```
library(tweedie)
tex <- data.frame(x=seq(from=0, by=0.1, to=50)) |>
  mutate(dens.a = dtweedie(x, xi = 1.1, mu=4, phi=2)) |>
  mutate(dens.b = dtweedie(x, xi = 1.3, mu=4, phi=0.5)) |>
  mutate(dens.c = dtweedie(x, xi = 1.5, mu=4, phi=0.3)) |>
  mutate(dens.d = dtweedie(x, xi = 1.8, mu=4, phi=1)) |>
  mutate(dens.e = dtweedie(x, xi = 1.5, mu=4, phi=5))


gf_line(dens.a ~ x, data=tex, color=colrs[1]) |>
  gf_line(dens.b ~x, data=tex, color=colrs[2]) |>
  gf_line(dens.c ~x, data=tex, color=colrs[3]) |>
  gf_line(dens.d ~x, data=tex, color=colrs[4]) |>
  gf_line(dens.e ~x, data=tex, color=colrs[5])
```




#### 4.5.4.0.5 PDF or PMF

A Tweedie distribution with $p > 1$ has the form:

$$f(X|\mu, \phi, p) = a(x, \phi)e^{\frac{1}{\phi}(\frac{x\mu^{(1-p)}}{(1-p)} - \kappa(\mu,p))}$$

where $\kappa(\mu, p) = \frac{\mu^{(2-p)}}{(2-p)}$ if $p \neq 2$, and if $p = 2$, $\kappa(\mu, p) = log(\mu)$; but $a(x, \phi)$ is a function that does not have an analytical expression. This expression is from SAS documentation at https://support.sas.com/rnd/app/stat/examples/tweedie/tweedie.pdf.

Alternately (and more simply(?)), a Tweedie distribution with $1 < p < 2$ is a compound of a Poisson distribution with parameter $\lambda$ and a gamma distribution with parameters $k$ and $\theta$. For example:

"Suppose that airplanes arrive at an airport following a Poisson process, and the number of passengers in each airplane follows a certain gamma distribution. Then, the number of passengers arriving at the airport follows a compound Poisson gamma process

$$Y = \sum_{i=1}^{N} D_i$$

where $N$ is the Poisson process that the airplanes follow, and $D_i$ is the gamma distribution that the passengers follow." (Thanks to D. Mao, http://math.uchicago.edu/~may/REU2013/REUPapers/Mao.pdf for this example.)

#### 4.5.4.0.6 Examples

The Tweedie distributions may be useful for "zero-inflated" data, where there is a class of observations for which the observed value of the variable is always zero, and another class for which the variable takes on positive continuous values. For example, this might model the number of birds present per unit area (when the study area includes places of unsuitable habitat where none are ever found), or perhaps the quantity of alcohol consumed per week by different people (some of whom may drink varying amounts, and others of whom may never drink at all).

# 5 Regression for Count Data

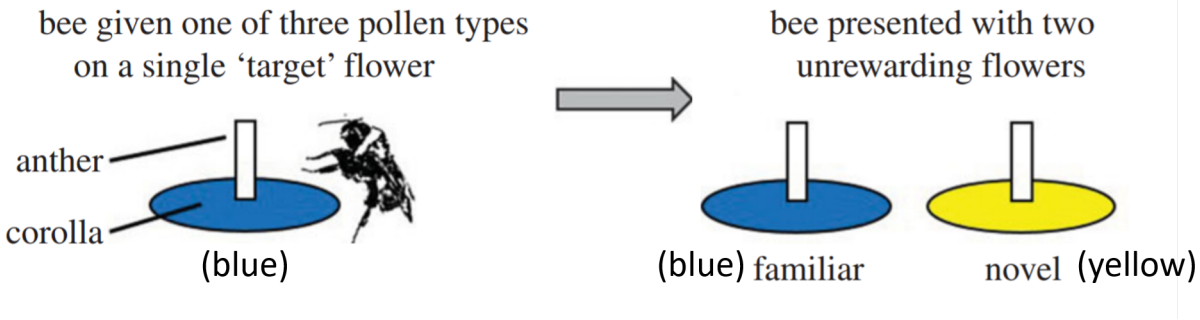
So far, we have fitted regression models for continuous-valued quantitative response variables. What if our response variable is really **count data** – discrete quantitative values limited to zero and positive integers?

## 5.1 Data Source

The dataset used here is `beevisits`, from:

Felicity Muth, Jacob S. Francis, and Anne S. Leonard. 2017. Bees use the taste of pollen to determine which flowers to visit. Biology Letters 12(7): DOI: 10.1098/rsbl.2016.0356. http://rsbl.royalsocietypublishing.org/content/roybiolett/12/7/20160356.full.pdf.

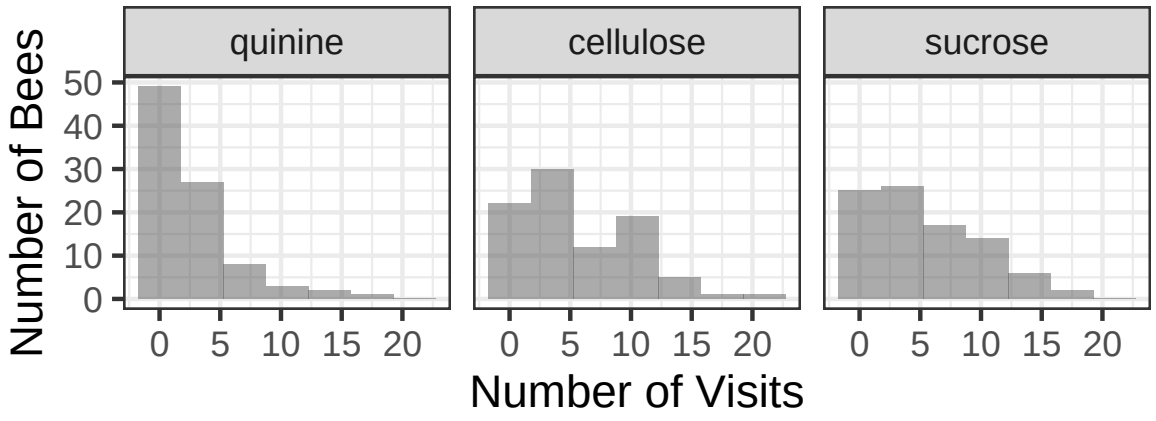
```
knitr::include_graphics('images/BeeExperiment.png')
```




## 5.2 A bad idea: multiple linear regression model

```
Rows: 270 Columns: 6
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (5): colony, beename, treatment, target colour, flower
dbl (1): novisits
```

```
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

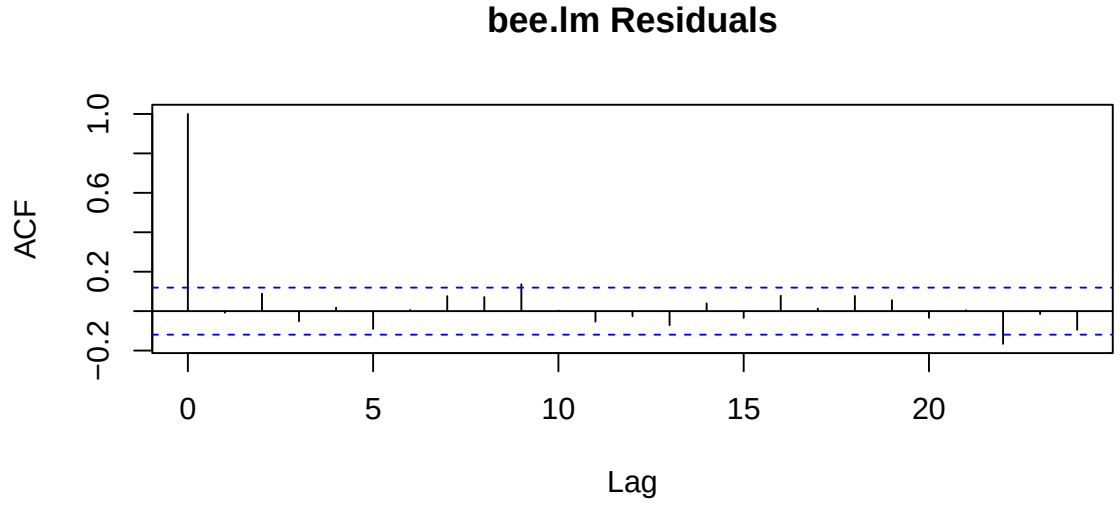


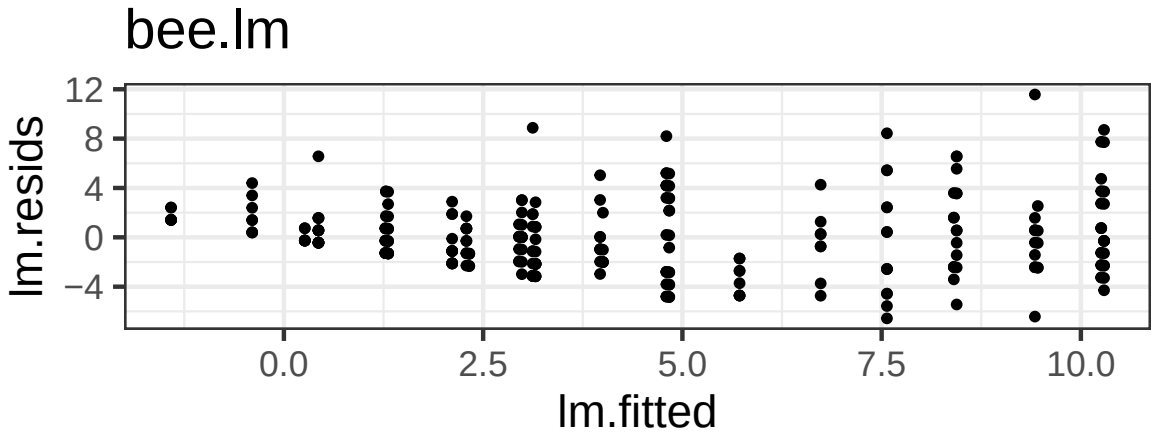

```
summary(beevisits)
```

```
   colony             beename              treatment  target colour
Length:270         Length:270         quinine  :90   Length:270
Class :character   Class :character   cellulose:90   Class :character
Mode  :character   Mode  :character   sucrose  :90   Mode  :character


   novisits         flower
Min.   : 0.000   Length:270
1st Qu.: 1.000   Class :character
Median : 3.000   Mode  :character
Mean   : 4.437
3rd Qu.: 7.000
Max.   :21.000
```

```
bee.lm <- lm(novisits ~ flower + treatment +
               colony, data=beevisits)
summary(bee.lm)
```

```
Call:
lm(formula = novisits ~ flower + treatment + colony, data = beevisits)

Residuals:
    Min      1Q  Median      3Q     Max
-6.5668 -2.1113 -0.2847  1.5665 11.5767

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)          2.1113     0.4440   4.755 3.27e-06 ***
flowernovel         -1.6778     0.4386  -3.825 0.000163 ***
flowertarget         5.4556     0.4386  12.438  < 2e-16 ***
treatmentcellulose   2.6883     0.4389   6.124 3.30e-09 ***
treatmentsucrose     2.7240     0.4415   6.170 2.56e-09 ***
colonyX             -0.8318     0.4491  -1.852 0.065110 .
colonyY             -1.8493     0.4254  -4.347 1.97e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.942 on 263 degrees of freedom
Multiple R-squared:  0.5763,    Adjusted R-squared:  0.5667
F-statistic: 59.63 on 6 and 263 DF,  p-value: < 2.2e-16
```







## 5.3 Problems with the linear model

What problems do we have with this model (its appropriateness, or goodness of fit to the data)?

- Non-constant error variance
- Non-normality of residuals
- Some predicted values are less than 0 – it's impossible that a bee could visit a flower a negative number of times!

## 5.4 Poisson Regression

*Detailed notes on the model equation were filled in here in class*

### 5.4.1 Fitting the Model

```
prm <- glm(novisits ~ flower + treatment +
               colony, data=beevisits,
           family=poisson(link='log'))
summary(prm)
```

```
Call:
glm(formula = novisits ~ flower + treatment + colony, family = poisson(link = "log"),
    data = beevisits)

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)        0.79055    0.08695   9.092  < 2e-16 ***
flowernovel       -0.75072    0.10442  -7.189 6.51e-13 ***
flowertarget       0.99945    0.06916  14.451  < 2e-16 ***
treatmentcellulose 0.69817    0.07910   8.827  < 2e-16 ***
treatmentsucrose   0.70953    0.07967   8.906  < 2e-16 ***
colonyX           -0.17521    0.07175  -2.442   0.0146 *
colonyY           -0.43724    0.07311  -5.981 2.22e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 1236.33  on 269  degrees of freedom
Residual deviance:  542.05  on 263  degrees of freedom
AIC: 1260.6

Number of Fisher Scoring iterations: 5
```

### 5.4.2 Conditions

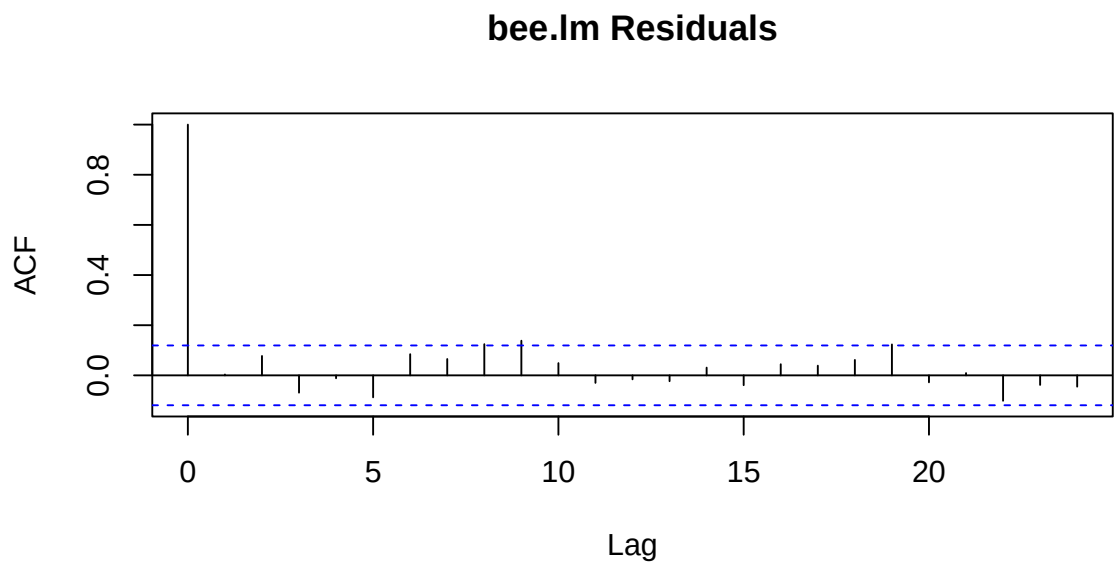
What conditions must hold for *this* model to be appropriate?

- Response variable (y) contains count data
- Linearity: $log(\lambda_i)$ is a linear function of the covariates $x_1$, $x_2$, … $x_n$. (*Later we will consider how to check this when some covariates are quantitative…in brief: plot log(rate) as a function of each covariate and look for (no non-)linear trend.*)
- **Mean = Variance**
- Independence (of residuals)
- There is *not* a condition specifying a PDF that the residuals should follow.

### 5.4.3 Model Assessment

Which conditions can we check already?

```
acf(resid(prm, type='response'), main='bee.lm Residuals')
```

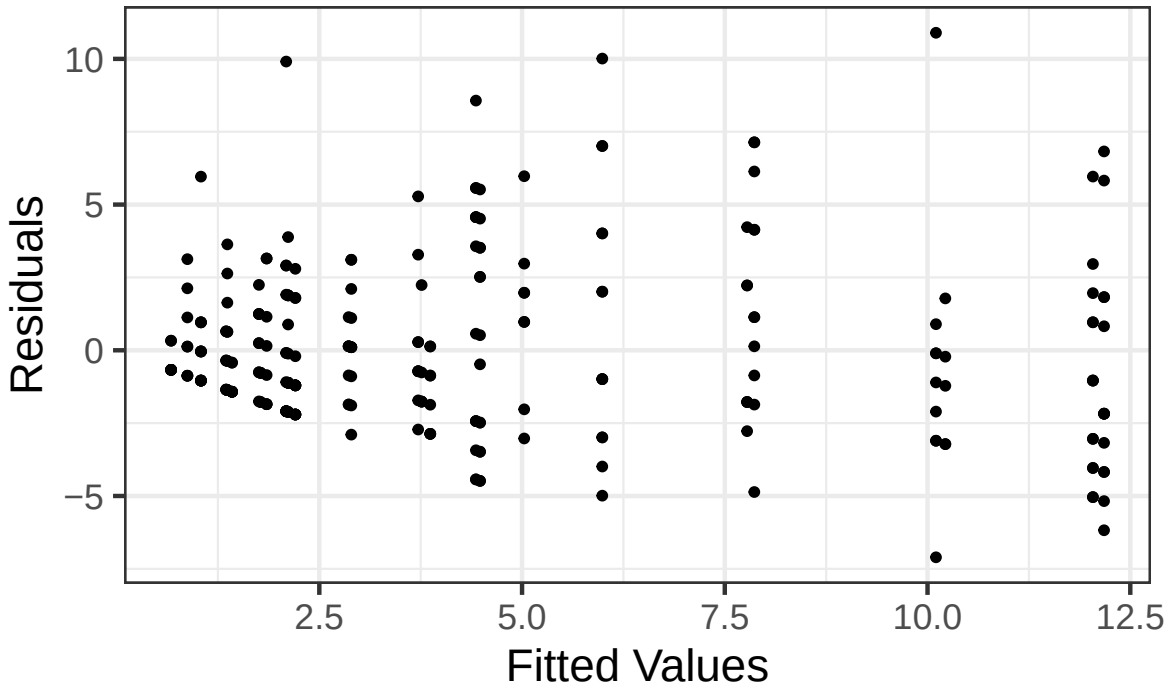



What does `type='response'` mean?

This means that the residuals are reported on the same scale as the response variable (that is, in units of counts, rather than log(counts)).

```
beevisits <- beevisits |>
  mutate(pm.resids = resid(prm, type='response'),
         pm.fitted = fitted(prm))
```

```
gf_point(pm.resids ~ pm.fitted, data=beevisits) |>
  gf_labs(x='Fitted Values', y='Residuals')
```

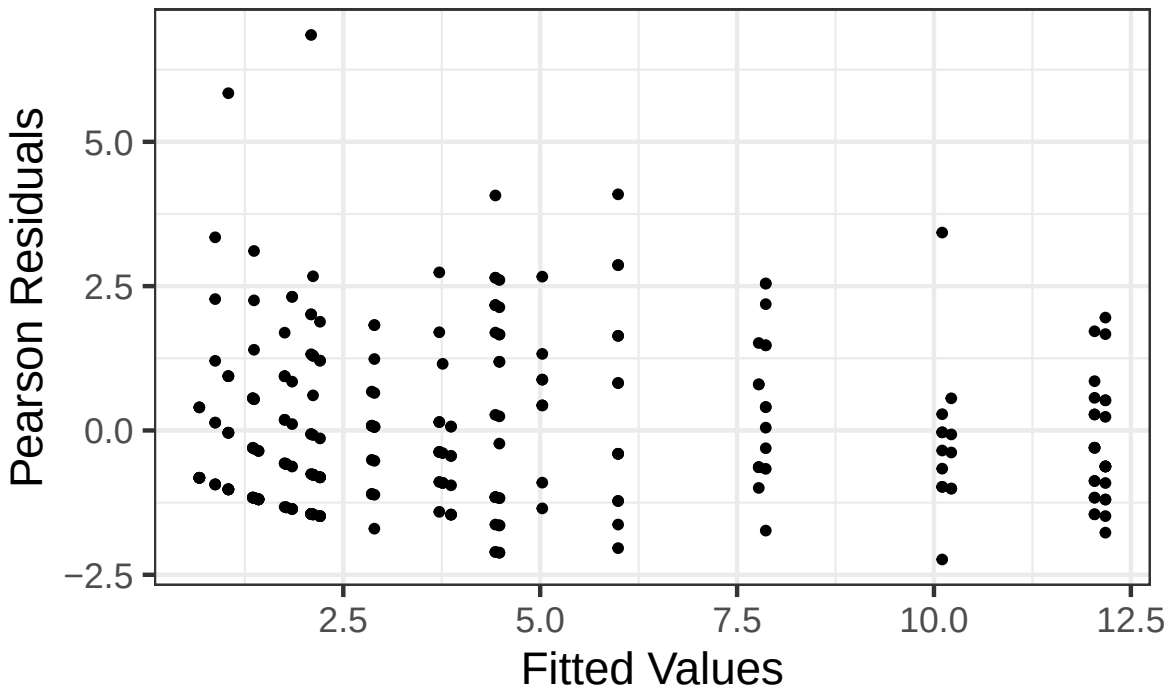



This trumpet pattern is *what we expect!* Why?

If the variance equals the mean, then as the mean (fitted value) goes up, then the variance (spread of residuals) will also be larger.

But how can we decide if it's the *right amount* of increase in variance with increase in fitted value? We can compute **Pearson** residuals, which are scaled by the *expected variance*. **The Pearson residuals should have approximately constant variance as a function of fitted values, and a standard deviation of about 1.**

```
beevisits <- beevisits |>
  mutate(pm.pearson.resids = resid(prm, type='pearson'))

gf_point( pm.pearson.resids ~ pm.fitted, data=beevisits) |>
  gf_labs(x='Fitted Values', y='Pearson Residuals')
```




A more complex solution: we could divide the fitted values into bins (choice of bin size is somewhat arbitrary; generally, you want them as small as possible, but still containing enough observations per bin to get good mean and variance estimates). In each bin, compute the mean and the variance of the residuals. Plot these means vs these variances and see if the slope is 1 (and intercept 0).

First, split the fitted values into 5 bins:

```
beevisits <- beevisits |>
  mutate(fitted.bins = cut(fitted(prm), breaks=5))

head(beevisits)
```

```
# A tibble: 6 x 12
  colony beename   treatment `target colour` novisits flower lm.resids lm.fitted
  <chr>  <chr>     <fct>     <chr>               <dbl> <chr>      <dbl>     <dbl>
1 Y      giantbee~ sucrose   yellow                 12 target     3.56       8.44
2 W      o51Wsucr~ sucrose   blue                    8 target    -2.29      10.3
3 W      o54Wquin~ quinine   blue                   13 target     5.43       7.57
4 W      o58Wcell~ cellulose yellow                  9 target    -1.26      10.3
5 W      o60Wcell~ cellulose blue                   11 target     0.745     10.3
```

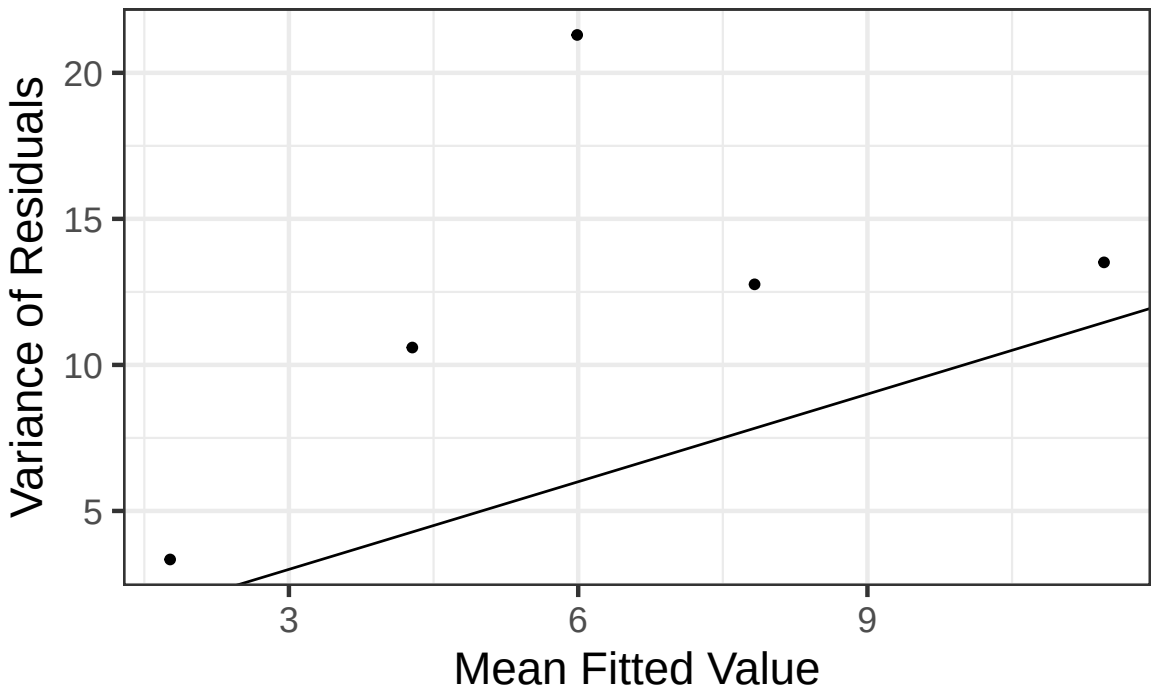
```
6 W      o63Wquin~ quinine   blue                    5 target    -2.57       7.57
# i 4 more variables: pm.resids <dbl>, pm.fitted <dbl>,
#   pm.pearson.resids <dbl>, fitted.bins <fct>
```

Next, compute the means and variances in each bin, view the result, and plot:

```
binned.bees <- beevisits |>
  group_by(fitted.bins) |>
  summarise(mean = mean(pm.fitted, na.rm=TRUE),
            var = var(pm.resids))

binned.bees
```

```
# A tibble: 5 x 3
  fitted.bins   mean   var
  <fct>        <dbl> <dbl>
1 (0.661,2.97]  1.77  3.34
2 (2.97,5.27]   4.28 10.6
3 (5.27,7.57]   5.99 21.3
4 (7.57,9.88]   7.83 12.8
5 (9.88,12.2]  11.5  13.5
```

```
gf_point(var ~ mean, data=binned.bees) |>
  gf_labs(x='Mean Fitted Value',
          y='Variance of Residuals') |>
   gf_abline(slope=1, intercept=0)
```




**That's a bit of a pain, and still a judgment call at the end.**

How *else* can we check the Mean = Variance condition?

### 5.4.4 Checking for overdispersion using overdispersion factor

We can estimate the **overdispersion factor**. This satisfies

$$\text{residual variance} = \text{overdispersion factor} * \text{mean}$$

So if it's a lot larger than 1, we have a problem. The function `overdisp_fun()` in package `s245` computes an estimate:

```
library(s245)
overdisp_fun(prm)
```

```
[1] 2.047036
```

Here, it's about 2: not too terrible, but still, residual variance is double what it should be according to our model. (Usually if the overdispersion is larger than 2 we will prefer a different model that better accounts for the fact that mean $\neq$ variance.)

## 5.5 Accounting for overdispersion: Negative Binomial Models

Finally, we could simply fit a model that allows for a more permissive mean-variance relationship (where the variance can be larger or smaller than the mean, by different amounts), and see if it is a better fit to the data according it our IC. The negative binomial distributions (type I and II) do this:

```
library(glmmTMB)
nbm1 <- glmmTMB(novisits ~ flower + treatment +
                colony, data=beevisits,
            family=nbinom1(link='log'))

nbm2 <- glmmTMB(novisits ~ flower + treatment +
                colony, data=beevisits,
            family=nbinom2(link='log'))

summary(nbm1)
```

```
 Family: nbinom1  ( log )
Formula:          novisits ~ flower + treatment + colony
Data: beevisits

     AIC      BIC   logLik deviance df.resid
  1182.6   1211.4   -583.3   1166.6      262


Dispersion parameter for nbinom1 family (): 1.07

Conditional model:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)        0.71730    0.12287   5.838 5.28e-09 ***
flowernovel       -0.74144    0.14503  -5.112 3.18e-07 ***
flowertarget       1.03838    0.09810  10.585  < 2e-16 ***
treatmentcellulose 0.71563    0.11167   6.408 1.47e-10 ***
treatmentsucrose   0.74317    0.11149   6.666 2.63e-11 ***
colonyX           -0.11496    0.09974  -1.153 0.249038
```

```
colonyY            -0.38387    0.10155  -3.780 0.000157 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(nbm2)
```

```
 Family: nbinom2  ( log )
Formula:          novisits ~ flower + treatment + colony
Data: beevisits

     AIC      BIC   logLik deviance df.resid
  1207.3   1236.1   -595.6   1191.3      262


Dispersion parameter for nbinom2 family (): 4.69

Conditional model:
                   Estimate Std. Error z value Pr(>|z|)
(Intercept)         0.73536    0.11775   6.245 4.24e-10 ***
flowernovel        -0.72548    0.12693  -5.716 1.09e-08 ***
flowertarget        1.04379    0.09983  10.456  < 2e-16 ***
treatmentcellulose  0.76192    0.11229   6.785 1.16e-11 ***
treatmentsucrose    0.76936    0.11424   6.734 1.65e-11 ***
colonyX            -0.16805    0.10838  -1.551    0.121
colonyY            -0.51527    0.10761  -4.788 1.68e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We **can** use AIC or BIC-based model selection to decide which model fits best, between Poisson, NB1, and NB2 models.

```
AIC(prm, nbm1, nbm2)
```

```
     df      AIC
prm   7 1260.621
nbm1  8 1182.575
nbm2  8 1207.286
```

We have a clear winner: NB1!

## 5.6 Accounting for overdispersion: quasi-Poisson Model

Or yet another option...in our class, we will not use quasi-Poisson models as much, as they are fitted via quasi-likelihood and this complicates model selection.

```
qprm <- glm(novisits ~ flower + treatment +
              colony, data=beevisits,
            family=quasipoisson(link='log'))
summary(qprm)
```

```
Call:
glm(formula = novisits ~ flower + treatment + colony, family = quasipoisson(link = "log"),
    data = beevisits)

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)         0.79055    0.12440   6.355 9.14e-10 ***
flowernovel        -0.75072    0.14940  -5.025 9.32e-07 ***
flowertarget        0.99945    0.09896  10.100  < 2e-16 ***
treatmentcellulose  0.69817    0.11317   6.169 2.58e-09 ***
treatmentsucrose    0.70953    0.11399   6.225 1.90e-09 ***
colonyX            -0.17521    0.10266  -1.707   0.0891 .
colonyY            -0.43724    0.10460  -4.180 3.97e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be 2.047071)

    Null deviance: 1236.33  on 269  degrees of freedom
Residual deviance:  542.05  on 263  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 5
```

We **should not** compare AIC with QAIC to compare two models, so to decide between Poisson and quasi-Poisson (if you ever use it) you must rely on the model assessment plots and the overdispersion factor to decide which is better.

## 5.7 Model selection with dredge() and (Q)AIC, BIC

Poisson and negative binomial models are fitted via maximum likelihood, so AIC or BIC may be used for model selection.

How can we use model selection criteria in a case where the likelihood can't be computed exactly?

The quasi (log) likelihood is an approximation to the likelihood. It has some properties in common with a (log) likelihood, but is not a likelihood; we resort to using it in cases where the (log) likelihood can not even be evaluated.

With some code/mathematical gymnastics, we can use principles of quasi-likelihood to estimate QAIC (quasi-AIC, or AIC based on quasi-likelihood) in R for model selection.

### 5.7.1 Review: all subsets selection with dredge()

What if, instead of comparing two (or a few) specific models, we want to compare all possible models that contain some combination of a set of candidate covariates? The package `MuMIn` contains a function, `dredge()`, that takes as input a "full" model (one containing all the candidate covariates). It then computes AIC (or other model selection criteria) for all possible models containing subsets of the covariate and reports the results in a ranked table. For example, for our Poisson regression model, we could do:

```
library(MuMIn)
#have to make sure na.action is 'na.fail' for input model
prm <- update(prm, na.action='na.fail')
dredge(prm, rank='AIC')
```

```
Fixed term is "(Intercept)"
```

```
Global model call: glm(formula = novisits ~ flower + treatment + colony, family = poisson(li
    data = beevisits, na.action = "na.fail")
---
Model selection table
  (Intrc) colny flowr trtmn df   logLik    AIC  delta weight
8  0.7905     +     +     +  7 -623.310 1260.6   0.00      1
7  0.6428           +     +  5 -642.317 1294.6  34.01      0
4  1.3100     +     +        5 -677.205 1364.4 103.79      0
3  1.1560           +        3 -695.120 1396.2 135.62      0
6  1.1240     +           +  5 -898.637 1807.3 546.65      0
5  0.9767                 +  3 -917.644 1841.3 580.67      0
```

```
2  1.6440     +              3 -952.532 1911.1 650.44      0
1  1.4900                    1 -970.446 1942.9 682.27      0
Models ranked by AIC(x)
```

```
dredge(prm, rank='BIC')
```

```
Fixed term is "(Intercept)"
```

```
Global model call: glm(formula = novisits ~ flower + treatment + colony, family = poisson(li
    data = beevisits, na.action = "na.fail")
---
Model selection table
  (Intrc) colny flowr trtmn df   logLik    BIC  delta weight
8  0.7905     +     +     +  7 -623.310 1285.8   0.00      1
7  0.6428           +     +  5 -642.317 1312.6  26.82      0
4  1.3100     +     +        5 -677.205 1382.4  96.59      0
3  1.1560           +        3 -695.120 1407.0 121.23      0
6  1.1240     +           +  5 -898.637 1825.3 539.46      0
5  0.9767                 +  3 -917.644 1852.1 566.27      0
2  1.6440     +              3 -952.532 1921.9 636.05      0
1  1.4900                    1 -970.446 1946.5 660.68      0
Models ranked by BIC(x)
```

### 5.7.2 Review: IC "weights"

Note the last two columns of the `dredge` output: the "delta" (or $\delta$) AIC or BIC values and the "weights". The $\delta$s are obtained by simply subtracting the best model's IC value from that of each other model.

We already mentioned a rule of thumb: that the $\delta IC$ should be at least 3 or to provide reasonable evidence that one model is really better than the other. Another way of measuring the differences between models is to use model *weights*. Theoretically, these measure the relative likelihoods of different models; you can think of them as giving the probability that a given model is the best-fitting one in the set of models examined, according to the IC being used.

Model weights are computed simply according to:

$$e^{\frac{-\delta IC}{2}}$$

And they sum to one for all models in a `dredge()`.

### 5.7.3 Extending dredge() to quasi-Likelihood

With (rather a lot of) effort to define some custom functions, we can do the same thing for a quasi-Poisson model using quasi-AIC. *Note: This idea is based on notes by Ben Bolker provided with the R package* `bbmle`.

```
library(MuMIn)

# modify a glm() output object so that
# it contains a quasipoisson fit but the
# AIC (likelihood) from the equivalent regular Poisson model
x.quasipoisson <- function(...) {
res <- quasipoisson(...)
res$aic <- poisson(...)$aic
res
}

# function to extract the overdispersion parameter
# from a quasi model
dfun <- function(object) {
with(object,sum((weights * residuals^2)[weights > 0])/df.residual)
}

# function that modifies MuMIn::dredge()
# for use with quasi GLM
qdredge <- function(model, family='x.quasipoisson', na.action=na.fail, chat = dfun(model),
  model2 <- update(model, family=family, na.action=na.action)
  (dt <- dredge(model2, rank=rank, chat=chat, ...))
}

#do "dredge" for model selection
qdredge(qprm)
```

```
Fixed term is "(Intercept)"

Global model call: glm(formula = novisits ~ flower + treatment + colony, family = family,
    data = beevisits, na.action = na.action)
---
Model selection table
  (Intrc) colny flowr trtmn df   logLik  rank  delta weight
8  0.7905     +     +     +  8 -622.310 626.0   0.00  0.999
7  0.6428           +     +  6 -641.317 640.6  14.57  0.001
```

```
4  1.3100     +     +        6 -676.205 674.7  48.66  0.000
3  1.1560           +        4 -694.120 688.2  62.16  0.000
6  1.1240     +           +  6 -897.637 891.0 265.00  0.000
5  0.9767                 +  4 -916.644 905.6 279.57  0.000
2  1.6440     +              4 -951.532 939.7 313.65  0.000
1  1.4900                    2 -969.446 953.2 327.15  0.000
Models ranked by rank(x, chat = 2.04707063774377)
```

Note: the `qdredge()` function is also provided for you in the package `s245`. So if you do want to use this, all you need to do is use `s245::qdredge()` instead of `dredge()`.

## 5.8 Offsets

In our model, the response variable was the number of flowers visited by a bee, and each bee was in the same experimental setting for the same amount of time, so there was no need to account for effort or time spent in each case.

This is not always true: consider, for example:

- Dolphin surveys
- Bird counts
- Consider the schools and crime example from homework as well

In this case, it would be natural to adjust for effort. The intuitive way to do it would be to use counts per unit effort as the response variable:

$$log(\frac{\lambda_i}{effort}) = \beta_0 + ...$$

But notice that this is equivalent to including $log(effort)$ as an "offset" on the right hand side of the regression equation:

$$log(\lambda_i) = \beta_0 + \cdots + log(effort)$$

This is how we specify models with *offsets* in R:

```
offset.mod <- glm(counts ~ predictor1 + predictor2 +
                    offset(log(effort)), family=poisson)
```

**Note: if you use dredge() with a model with an offset, be sure to specify the offset as a "fixed" term, i.e. a term that must be included in *all* models:**

```
dredge(offset.mod, fixed = 'offset(log(effort))')
```

## 5.9 Prediction Plots

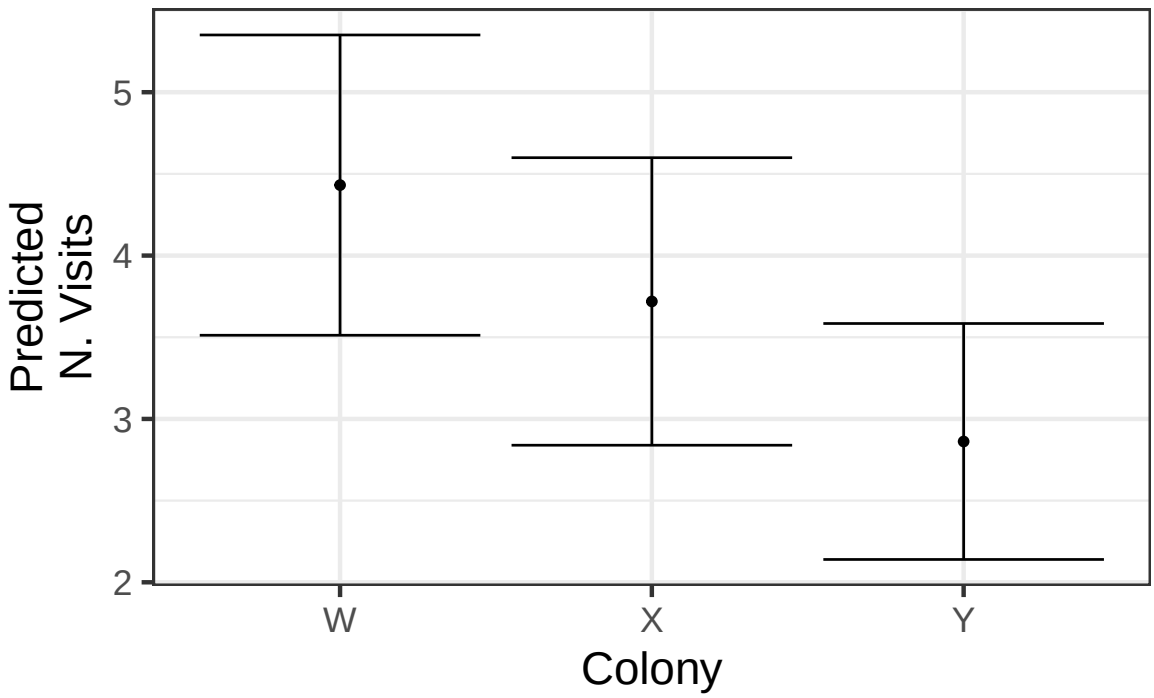
Once you have a "best" model, how do you interpret it?

When we went to great lengths to make prediction plots for a linear regression model so we could "see" the slope of the predicted relationship, you may have wondered: *why bother?* I can just look at the slope estimate and get the same insight!

Now, with a more complicated model equation with a link function, it's not so easy. Now, we will really appreciate those prediction plots!

With the link function and the Poisson distribution, it is more challenging to interpret the coefficients of the model directly. The easiest way to understand the effects of different predictors is to look at plots of model predictions. However, as always we don't want to plot `fitted(model)` as a function of each predictor in a model with more than one predictor; predictors other than the one we are interested in will influence the predictions, introducing extra variation. Instead, we will construct a new (fake) dataset to make predictions for, in which all predictors but the one we are interested in are held constant. For example, using our quasi-Poisson model and looking at the effect of colony (predicted values with 95% CIs):

```
newdata <- data.frame(colony=c('W', 'X', 'Y'),
                      flower='familiar',
                      target.colour='blue',
                      treatment='cellulose')

pred = predict(qprm, newdata=newdata,
                 type='response',
                 se.fit=TRUE)

newdata <- newdata |>
  mutate(preds = pred$fit,
         CIlow = pred$fit - 1.96*pred$se.fit,
         CIup = pred$fit + 1.96*pred$se.fit)

gf_point(preds ~ colony, data=newdata) |>
  gf_labs(x='Colony', y='Predicted\nN. Visits') |>
  gf_errorbar(CIlow + CIup ~ colony, data=newdata)
```




If we had a quantitative predictor instead, the process would be similar, except when we define `newdata`, we would have to choose a range and granularity for which to make predictions. For example, imagine if bee length in mm was one of our predictors, and we wanted predictions for lengths between 0.5 and 3 mm (with a value every 0.05mm). We might begin:

```
newdata <- expand.grid(bee.length = seq(from=0.05, by=0.05, to=3),
                       colony=c('W'),
                       flower='familiar',
                      target.colour='blue',
                      treatment='cellulose')
head(newdata)
```

```
  bee.length colony   flower target.colour treatment
1       0.05      W familiar          blue cellulose
2       0.10      W familiar          blue cellulose
3       0.15      W familiar          blue cellulose
4       0.20      W familiar          blue cellulose
5       0.25      W familiar          blue cellulose
6       0.30      W familiar          blue cellulose
```

Then, when we make the plot, we would need to use `gf\_ribbon()` instead of `gf\_errorbar()` to show the CI.

We can also use `pred_plot(fitted_model, 'variable_name')` to make these plots with a lot less code (and if you need to know the values at which other predictors are fixed, use `get_fixed(dataset)`).

# 6 Model Averaging

So far, we have used AIC and/or BIC for model selection, to decide which variables to keep in a "best" model and which to exclude. But we have already seen a number of cases where there is not **one** model that is clearly superior to all the others. In those cases, we have decided on the smallest model (with the fewest predictors), but there are other options when many competing models have similar scores.

One option is *not to choose* – instead, keep them all, and make predictions via a weighted average of all the models. The models with the best IC scores get more weight. This can be a good option if the main goal is accurate prediction (rather than deciding definitively which predictors are "good" ones and which are not).

How can we do it? Let's explore an example.

## 6.1 Data: School Survey on Crime and Safety

The data for this example are from a survey of U.S. schools, the 2000 School Survey on Crime and Safety. There is information about the study at

http://catalog.data.gov/dataset/2000-school-survey-on-crime-and-safety,

which says the study "is a cross-sectional survey of the nation's public schools designed to provide estimates of school crime, discipline, disorder, programs and policies. SSOCS is administered to public primary, middle, high, and combined school principals in the spring of even-numbered school years…Public schools were sampled in the spring of 2000 to participate in the study."

The dataset you will use is available online at:

http://sldr.netlify.com/data/sscrime.csv

It contains a number of variables:

- `VisitorCheckIn`: Whether visitors to the school must check in to gain entry to the school.
- `LockedGates`: Whether there are locked gates at the entry to the school.
- `MetalDetectors`: Whether there is a metal detector at the entrance to the school.

- `DrugSniffDog`: Whether a drug-sniffing dog is randomly brought into the school to carry out inspections.
- `DrugTesting`: Whether any drug testing of students occurs.
- `UniformsRequired`:Whether students are required to wear uniforms.
- `DressCode`: Whether a strict dress code is enforced.
- `Lockers`: Whether students have lockers.
- `StudentIDBadges`: Whether students are required to wear ID badges.
- `StaffIDBadges`: Whether teachers and other staff are required to wear ID badges.
- `SecurityCameras`: Whether there are security cameras on the premises.
- `OfficialRiotPlan`: Whether the school has a written plan in place for how to deal with a riot or large-scale fight.
- `ViolenceReductionProgram`: Whether the school has a Violence Reduction Program in place.
- `Security`: Whether security officers are present on the premises.
- `TrainingHours`: Average amount of time (in hours) that teachers and staff have devoted to training related to violence reduction.
- `AttacksWithoutWeapon`: Number of attacks that have occurred at the school, not involving a weapon.
- `Thefts`: Number of thefts.
- `Vandalism`: Number of incidents of vandalism.
- `ViolentIncidentsTotal`: Number of violent incidents of all types that have occurred at the school.
- `Enrollment`: Number of students enrolled in the school (categorical)
- `NEnrollment`: Number of students enrolled in the school (numeric)
- `SurveyRespondent`: The identity of the person who filled out the survey.
- `Location`: Whether the location of the school is Urban, Rural, etc.

```
ssc <- read.csv('http://sldr.netlify.com/data/sscrime.csv')
```

## 6.2 Modelling number of violent incidents per school

We will fit a model for the number of violent incidents total as a function of a number of predictors. This is count data and we will fit a negative binomial regression model:

```
library(glmmTMB)
school.nb2 <- glmmTMB(ViolentIncidentsTotal ~ TrainingHours + Location +
                        SecurityCameras + DressCode + UniformsRequired +
                        NEnrollment, data=ssc,
                family=nbinom2(link='log'),
                na.action = 'na.fail')
```

I will use AIC and the `dredge()` function to compare all possible subsets of my saturated model and figure out which variables should be included in the best model. I chose AIC in this case because it is perhaps more widely used than BIC (that's not a good reason unless you really have no better one, but there you have it) and because with the relatively small sample size here, I don't feel a particular need to use BIC for its larger penalty term.

```
library(MuMIn)
#do "dredge" for model selection
mod.sel <- dredge(school.nb2, rank='AIC')
head(mod.sel, 8)
```

```
Global model call: glmmTMB(formula = ViolentIncidentsTotal ~ TrainingHours + Location +
    SecurityCameras + DressCode + UniformsRequired + NEnrollment,
    data = ssc, family = nbinom2(link = "log"), na.action = "na.fail",
    ziformula = ~0, dispformula = ~1)
---
Model selection table
   cnd((Int)) dsp((Int)) cnd(DrC) cnd(Lct)  cnd(NEn) cnd(ScC)  cnd(TrH)
4       3.610          +        +        +
36      3.585          +        +        +
12      3.635          +        +        +                  +
8       3.570          +        +        + 2.537e-05
20      3.608          +        +        +                    0.0008916
44      3.610          +        +        +                  +
40      3.529          +        +        + 3.410e-05
16      3.588          +        +        + 3.140e-05        +
   cnd(UnR) df    logLik    AIC delta weight
4            6 -1745.380 3502.8  0.00  0.287
36         + 7 -1745.017 3504.0  1.27  0.152
12           7 -1745.103 3504.2  1.45  0.139
8            7 -1745.280 3504.6  1.80  0.117
20           7 -1745.380 3504.8  2.00  0.106
44         + 8 -1744.734 3505.5  2.71  0.074
40         + 8 -1744.840 3505.7  2.92  0.067
16           8 -1744.953 3505.9  3.15  0.060
Models ranked by AIC(x)
```

Because the first 7 or so models all have AIC scores within 3 units of each other, it is hard to choose one best model here. In this situation, one way to choose is to pick the model that includes the smallest number of predictors, and still acheives an AIC that is among the best. Another option would be to use **model averaging**.

## 6.3 Model Averaging

What if we wanted to use model averaging to find the best model, instead? We might choose this route because there are several models that all have AIC that are close to each other and thus fit the data approximately equally well. So we might choose to make predictions (and compute coefficients) that are the *average* of all the models (weighted by IC weights).

Notes of caution:

- If the model is not a linear regression (if there is a link function for instance) then it's important to get **predictions** by *averaging the predictions from the different models*, **not** by making predictions using the model-averaged coefficients. The code below is careful to do this.
- Model averaging is used pretty widely but is also controversial (like most model selection methods, in fact!) For example, see: [https://esajournals.onlinelibrary.wiley.com/doi/full/10.1890/14-1639.1] and [https://drewtyre.rbind.io/post/rebutting_cade/].

To do model averaging, we use package `MuMIn` (function `model.avg`).

### 6.3.1 Getting the Averaged Model

The following code gets the average model. If we did the default (`fit=FALSE`), it would be a bit faster, but we would then not be able to get predictions from the model.

```
mod.sel2 <- dredge(school.nb2)
ave.model <- MuMIn::model.avg(mod.sel2, fit=TRUE)
summary(ave.model)
```

```
Call:
model.avg(object = get.models(object = mod.sel2, subset = NA))

Component model call:
glmmTMB(formula = ViolentIncidentsTotal ~ <64 unique rhs>, data = ssc,
     family = nbinom2(link = "log"), ziformula = ~0, dispformula = ~1,
     na.action = na.fail)

Component models:
       df   logLik    AICc delta weight
12      6 -1745.38 3502.98  0.00   0.24
126     7 -1745.02 3504.33  1.35   0.12
124     7 -1745.10 3504.51  1.52   0.11
```

```
123     7 -1745.28 3504.86  1.88   0.09
125     7 -1745.38 3505.06  2.08   0.08
1246    8 -1744.73 3505.86  2.87   0.06
1236    8 -1744.84 3506.07  3.08   0.05
1234    8 -1744.95 3506.29  3.31   0.05
1256    8 -1745.02 3506.42  3.44   0.04
1245    8 -1745.10 3506.59  3.60   0.04
1235    8 -1745.28 3506.95  3.96   0.03
12346   9 -1744.49 3507.46  4.48   0.03
12456   9 -1744.73 3507.95  4.96   0.02
12356   9 -1744.84 3508.17  5.18   0.02
12345   9 -1744.95 3508.38  5.40   0.02
123456 10 -1744.48 3509.56  6.58   0.01
2       5 -1753.09 3516.33 13.35   0.00
26      6 -1752.29 3516.80 13.82   0.00
24      6 -1752.60 3517.42 14.44   0.00
246     7 -1751.79 3517.87 14.89   0.00
25      6 -1753.09 3518.40 15.41   0.00
23      6 -1753.09 3518.40 15.41   0.00
236     7 -1752.26 3518.82 15.84   0.00
256     7 -1752.29 3518.87 15.89   0.00
234     7 -1752.59 3519.48 16.49   0.00
245     7 -1752.59 3519.49 16.50   0.00
2346    8 -1751.72 3519.82 16.83   0.00
2456    8 -1751.78 3519.96 16.97   0.00
235     7 -1753.09 3520.47 17.49   0.00
2356    8 -1752.26 3520.90 17.92   0.00
2345    8 -1752.59 3521.56 18.57   0.00
23456   9 -1751.71 3521.91 18.92   0.00
136     5 -1756.63 3523.42 20.43   0.00
1346    6 -1756.08 3524.39 21.41   0.00
13      4 -1758.25 3524.60 21.62   0.00
16      4 -1758.59 3525.28 22.29   0.00
1356    6 -1756.61 3525.44 22.45   0.00
134     5 -1757.74 3525.65 22.66   0.00
1       3 -1759.81 3525.68 22.69   0.00
13456   7 -1756.08 3526.46 23.48   0.00
135     5 -1758.21 3526.59 23.60   0.00
146     5 -1758.30 3526.76 23.78   0.00
14      4 -1759.53 3527.16 24.17   0.00
156     5 -1758.54 3527.24 24.26   0.00
15      4 -1759.75 3527.61 24.62   0.00
1345    6 -1757.73 3527.69 24.71   0.00
```

```
1456    6 -1758.28 3528.79 25.80   0.00
145     5 -1759.50 3529.15 26.17   0.00
36      4 -1765.82 3539.76 36.77   0.00
346     5 -1764.96 3540.08 37.10   0.00
6       3 -1767.12 3540.31 37.33   0.00
46      4 -1766.54 3541.19 38.20   0.00
356     5 -1765.77 3541.70 38.71   0.00
3456    6 -1764.95 3542.12 39.14   0.00
56      4 -1767.04 3542.18 39.20   0.00
(Null)  2 -1769.34 3542.72 39.74   0.00
3       3 -1768.50 3543.05 40.07   0.00
456     5 -1766.50 3543.17 40.18   0.00
34      4 -1767.71 3543.52 40.54   0.00
4       3 -1768.77 3543.61 40.63   0.00
5       3 -1769.24 3544.54 41.55   0.00
35      4 -1768.41 3544.93 41.95   0.00
345     5 -1767.68 3545.51 42.53   0.00
45      4 -1768.72 3545.54 42.55   0.00

Term codes:
       cond(DressCode)          cond(Location)       cond(NEnrollment)
                     1                       2                       3
 cond(SecurityCameras)    cond(TrainingHours) cond(UniformsRequired)
                     4                       5                       6

Model-averaged coefficients:
(full average)
                            Estimate Std. Error Adjusted SE z value Pr(>|z|)
cond((Int))                 3.596e+00  1.316e-01   1.320e-01  27.252  < 2e-16
cond(DressCodeyes)          3.765e-01  9.608e-02   9.639e-02   3.906 9.39e-05
cond(LocationRural)        -5.597e-01  1.328e-01   1.332e-01   4.201 2.65e-05
cond(LocationTown)         -5.874e-01  1.543e-01   1.548e-01   3.795 0.000148
cond(LocationUrban Fringe) -1.024e-01  1.179e-01   1.183e-01   0.866 0.386523
cond(UniformsRequiredyes)   5.685e-02  1.373e-01   1.376e-01   0.413 0.679432
cond(SecurityCamerasyes)   -2.384e-02  6.451e-02   6.466e-02   0.369 0.712383
cond(NEnrollment)           8.826e-06  2.812e-05   2.818e-05   0.313 0.754163
cond(TrainingHours)        -2.156e-04  2.509e-02   2.517e-02   0.009 0.993166

cond((Int))                ***
cond(DressCodeyes)         ***
cond(LocationRural)        ***
cond(LocationTown)         ***
cond(LocationUrban Fringe)
```

```
cond(UniformsRequiredyes)
cond(SecurityCamerasyes)
cond(NEnrollment)
cond(TrainingHours)

(conditional average)
                            Estimate Std. Error Adjusted SE z value Pr(>|z|)
cond((Int))                 3.596e+00  1.316e-01   1.320e-01  27.252  < 2e-16
cond(DressCodeyes)          3.771e-01  9.499e-02   9.530e-02   3.956 7.61e-05
cond(LocationRural)        -5.598e-01  1.328e-01   1.332e-01   4.203 2.63e-05
cond(LocationTown)         -5.874e-01  1.542e-01   1.547e-01   3.796 0.000147
cond(LocationUrban Fringe) -1.024e-01  1.179e-01   1.183e-01   0.866 0.386505
cond(UniformsRequiredyes)   1.661e-01  1.921e-01   1.927e-01   0.862 0.388948
cond(SecurityCamerasyes)   -7.409e-02  9.597e-02   9.628e-02   0.770 0.441588
cond(NEnrollment)           3.049e-05  4.550e-05   4.565e-05   0.668 0.504120
cond(TrainingHours)        -8.265e-04  4.912e-02   4.928e-02   0.017 0.986617

cond((Int))                ***
cond(DressCodeyes)         ***
cond(LocationRural)        ***
cond(LocationTown)         ***
cond(LocationUrban Fringe)
cond(UniformsRequiredyes)
cond(SecurityCamerasyes)
cond(NEnrollment)
cond(TrainingHours)
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

**If you are trying to get model-averaged coefficients from the summary output above, be sure to look for the "full average" ones and not the "conditional average" (which only includes models where the predictor was included, i.e., where the coefficient was not 0).**

### 6.3.2 Getting Predictions from the Averaged Model

```r
ma.preds <- predict(ave.model, se.fit=TRUE,
                    type = 'response',
                    backtransform = FALSE)
```

The resulting predictions are a list with entries `fit` and `se.fit` just like we are used to. (So you could make predictions with a `newdata` data set and use them for prediction plots, for example. Be careful – your "new" dataset now has to include values for all candidate predictors in the full model.)

Comparing with the predictions from our previous "best" model:

```
best.school.nb2 <- glmmTMB(ViolentIncidentsTotal ~ DressCode + Location,
                           data=ssc,family=nbinom2(link='log'))


# pred_plot(ave.model, 'DressCode', ylab = 'N. Incidents',
#             data = ssc, color = 'red')
#
# pred_plot(best.school.nb2, 'DressCode', ylab = 'N. Incidents',)
```

So they are pretty comparable, but a little different (the differences may be bigger the more there are different models with similar IC results contributing to the average model – when one model carries almost all the weight, then the "single best" model and the model-averaging model will give almost the same results). It also makes sense that there will be a bit more uncertainty in the average model.

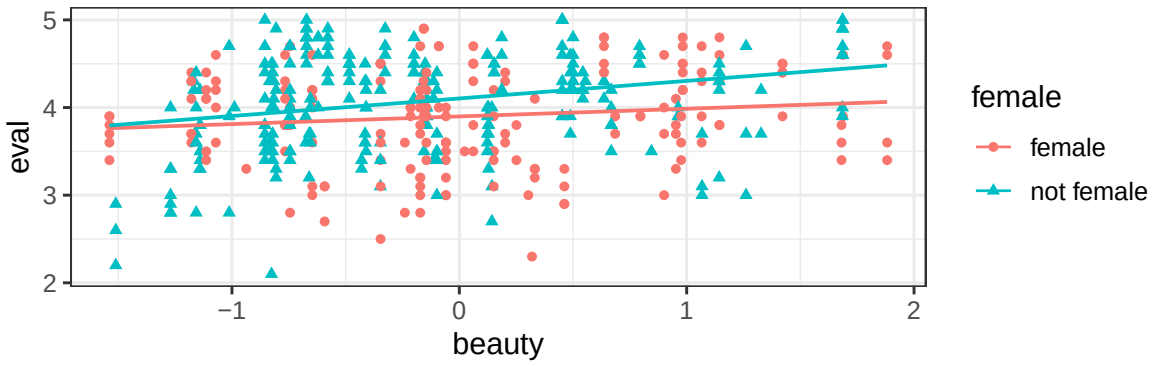
# 7 Interactions

Two predictors **interact** when you need to know values of *both* in order to make an accurate prediction of the response variable value.

Predictors can interact in *any* type of regression model (so this chapter could really be placed almost anywhere).

## 7.1 Example: Quantitative-Categorical Interaction

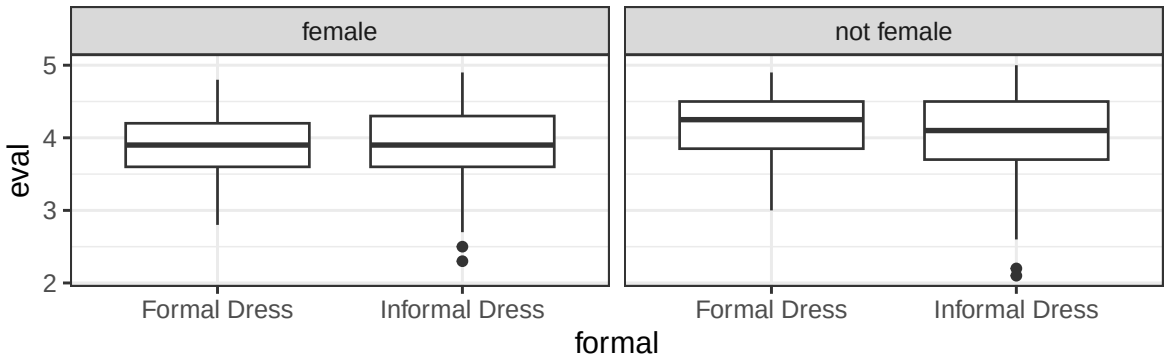
```
gf_point(eval ~ beauty, color = ~female, shape = ~female, data = teach_beauty) |>
  gf_lm()
```




`Eval` may go up as `beauty` increases, but the *slope* of the relationship is *different* for females and non-females. This is an **interaction** between `beauty` and `female`.

## 7.2 Categorical-Categorical Interaction Example

```
gf_boxplot(eval ~ formal | female, data = teach_beauty)
```




Perhaps *Informal Dress* affects `eval` scores, but really only for non-females – for females, `formal` dress doesn't make a difference either way.

The effect of `formal` dress is *different* depending on the value of `female`. This is an interaction between `formal` and `female`.

## 7.3 Quant-Quant interactions?

Yes, these are possible, but very hard to visualize and conceptualize. Basically, it would mean that the slope of the line for one predictor changes gradually as the value of a second variable changes.

## 7.4 R code

If you want to include an interaction term in a model in R, use a `*` rather than a `+` between the predictors that (may) interact. For example, based on our exploration above, we might try:

```
beauty_mod <- lm(eval ~ beauty*female +
                   formal*female,
                 data = teach_beauty,
                 na.action = 'na.fail')
summary(beauty_mod)
```

```
Call:
lm(formula = eval ~ beauty * female + formal * female, data = teach_beauty,
```

```
    na.action = "na.fail")

Residuals:
     Min       1Q   Median       3Q      Max
-1.83418 -0.36763  0.04966  0.39789  1.07161

Coefficients:
                                    Estimate Std. Error t value Pr(>|t|)
(Intercept)                          3.84839    0.10813  35.591   <2e-16 ***
beauty                               0.09021    0.04744   1.902   0.0578 .
femalenot female                     0.27615    0.13130   2.103   0.0360 *
formalInformal Dress                 0.05751    0.11574   0.497   0.6195
beauty:femalenot female              0.10841    0.06452   1.680   0.0936 .
femalenot female:formalInformal Dress -0.08378    0.14276  -0.587   0.5576
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5371 on 457 degrees of freedom
Multiple R-squared:  0.07326,   Adjusted R-squared:  0.06312
F-statistic: 7.226 on 5 and 457 DF,  p-value: 1.587e-06
```

Notice the additional indicator variables in the coefficient table/model equation. Now we need to *adjust* the effects of the `beauty` predictor depending on the values of `formal` and `female`, which interact with it.

We can use IC-based model selection to determine whether including these interactions in a model is important or not.

```
library(MuMIn)
dredge(beauty_mod, rank = 'AIC')
```

```
Global model call: lm(formula = eval ~ beauty * female + formal * female, data = teach_beauty
    na.action = "na.fail")
---
Model selection table
   (Int)     bty fml frm bty:fml fml:frm df   logLik   AIC delta weight
12 3.899 0.08762   +           +          5 -366.309 742.6  0.00  0.414
4  3.897 0.14860   +                      4 -367.868 743.7  1.12  0.236
16 3.896 0.08773   +   +       +          6 -366.308 744.6  2.00  0.152
8  3.897 0.14860   +   +                  5 -367.868 745.7  3.12  0.087
32 3.848 0.09021   +   +       +       +  7 -366.134 746.3  3.65  0.067
24 3.833 0.14880   +   +               +  6 -367.560 747.1  4.50  0.044
```

```
2  4.010 0.13300                          3 -375.323 756.6 14.03  0.000
6  4.032 0.13170       +                  4 -375.249 758.5 15.88  0.000
3  3.901           +                      3 -378.503 763.0 20.39  0.000
7  3.932           +   +                  4 -378.367 764.7 22.12  0.000
23 3.872           +   +                +  5 -378.103 766.2 23.59  0.000
1  3.998                                  2 -383.747 771.5 28.88  0.000
5  4.044               +                  3 -383.431 772.9 30.24  0.000
Models ranked by AIC(x)
```

In the case of the particular model we fitted, the "best" model starting from this full model is actually one *without* interactions. If you want to explore the dataset further, you will find that actually a model where `age`, `beauty` AND `female` interact fits much better…

## 7.5 Cautionary note

If you include an interaction in a regression model, you **must** also include the corresponding "fixed effects" – this means if you have an indicator variable/slope term for an interaction in your model, you must also have the indicator variables/slopes corresponding to the individual predictors. Our fitting functions (`lm()`, `glm()`, `glmmTMB()`, etc.) are smart enough to ensure this for you. So is `dredge()`. (It would take effort to mess this up in R.)

# 8 Binary Regression

Our next goal: establish a framework for doing regression modelling when the response variable is a categorical one, with two categories.

## 8.1 Data Source

The dataset used here is on Alaskan wood frogs, detailing some physical characteristics, habitat characteristics, and the number of developmental and other abnormalities found in the frogs. It was originally obtained from:[http://datadryad.org/resource/doi:10.5061/dryad.sq72d].

The data file can be accessed online at: [http://sldr.netlify.com/data/FrogAbnormalities.csv]

```
  FrogID TotalLength TailLength    Stage Year RoadDistance RoadType Abnormal
1     25    19.19701 31.8285559 Stage 43 2011           22   Gravel       No
2     29    20.99035  0.1151022 Stage 46 2012          387   Gravel       No
3     29    20.50364  0.1055217 Stage 46 2011          781    Paved       No
  SkeletalAbnormality EyeAbnormality orig.id
1                  No             No    3450
2                  No             No    2276
3                  No             No     158
```

## 8.2 Logistic Regression

Recall, for linear regression we fitted a model for continuous (numeric) response variable $y$ according to:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ...\beta_k x_k + \epsilon$$

where $x$s are the $k$ predictor variables, $\beta$s are the parameters to be estimated by the model, and $\epsilon \sim N(0, \sigma)$ are the model residuals.

When our response variable was a *count* variable, we modified our equation to:

$$log(\lambda_i) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ...\beta_k x_k + \epsilon_{link}$$

positing that $y_i \sim Pois(\lambda_i)$ for Poisson regression; similarly for quasiPoisson or negative binomial regression, we just replaced that Poisson distribution with a quasiPoisson or a negative binomial distribution.

What if our response variable is *logical* – a categorical variable with just two possible values? We will designate one of the two values a "success," and then we want to predict the probability of success as a function of some set of predictors. What will our model equation look like in this case?

$$logit(p_i) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ...\beta_k x_k + \epsilon_{link}$$

where the `logit` function is $logit(x) = log(\frac{x}{1-x})$. This function maps probabilities to positive and negative real numbers, effectively "spreading them out" from the range 0-1 to the full range of real numbers. How does this equation relate back to our desired response variable? Well, $i$th observation of the response variable is assumed to follow a binomial distribution with probability $p_i$ ($y_i \sim Binom(n_i, p_i)$). ($n_i$ depends on the setup of the data – often n=1 for each row of the dataset, as here where each row is one frog. We can think of each frog as one binomial trial, with success/failure meaning abnormality/normality of the frog.)

## 8.3 Checking the data setup

We would like to model the proportion frogs with abnormalities as a function of a set of covariates. The variable `Abnormal` has values "Yes" and "No". In R, if we use this (factor) variable as our response, how will R determine which level (value of the variable) is a "success"?

R uses the FIRST variable value as "failure" and the second as "success" – this makes sense if you imagine coding 0 for failure and 1 for success (and then sorting in numeric/alphabetical order). If you have a categorical variable with informative values, you will need to make sure that the "base" (first) level is the one you want to equate with "failure".

```
levels(frogs$Abnormal)
```

```
NULL
```

If you do need to rearrange the levels, one way to do it is to use the `forcats::fct_relevel()` function. Example:

```
#ref will be the FIRST level after releveling
frogs <- frogs |>
  mutate(Abnormal = forcats::fct_relevel(Abnormal, 'No'))
frogs |> pull(Abnormal) |> levels()
```

```
[1] "No"  "Yes"
```

## 8.4 Fitting a saturated model

Let's try fitting a model for Abnormalities as a function of Stage, Year, RoadType, and RoadDistance. Why do you think these variables and not others were chosen?

Perhaps it makes sense that type and distance to road are proxies for urbanization, and frogs may do better in more pristine habitats. It also seems likely that there would be differences over time. There may also be differences by Stage, if frogs with severe abnormalities have trouble even surviving to the later/older stages.

```
frog.logr <- glm(Abnormal ~ Stage + factor(Year) + RoadType + RoadDistance,
                 data=frogs, family=binomial(link='logit'))
summary(frog.logr)
```

```
Call:
glm(formula = Abnormal ~ Stage + factor(Year) + RoadType + RoadDistance,
    family = binomial(link = "logit"), data = frogs)

Coefficients:
                   Estimate Std. Error z value Pr(>|z|)
(Intercept)      -0.3509350  0.4198798  -0.836 0.403268
StageStage 43     0.1129785  0.1136966   0.994 0.320376
StageStage 44    -0.0987133  0.1290558  -0.765 0.444338
StageStage 45    -0.4616044  0.1301337  -3.547 0.000389 ***
StageStage 46    -0.5954533  0.1451580  -4.102 4.09e-05 ***
factor(Year)2001 -1.4231818  0.8393630  -1.696 0.089971 .
factor(Year)2002 -1.3820162  1.1210725  -1.233 0.217664
factor(Year)2003 -0.9744573  0.5216890  -1.868 0.061778 .
factor(Year)2004 -1.4345710  0.4272057  -3.358 0.000785 ***
factor(Year)2005 -1.1185067  0.4214799  -2.654 0.007960 **
factor(Year)2006 -1.1232649  0.4229028  -2.656 0.007905 **
factor(Year)2010 -0.4985424  0.4299638  -1.159 0.246253
```

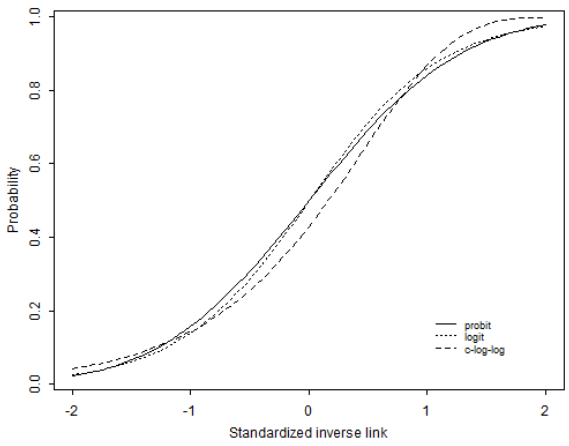
```
factor(Year)2011 -1.1674055  0.4163277  -2.804 0.005046 **
factor(Year)2012 -1.0488208  0.4157733  -2.523 0.011650 *
RoadTypePaved    -0.1689634  0.0842935  -2.004 0.045020 *
RoadDistance      0.0005653  0.0002471   2.287 0.022176 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 4787.5  on 5370  degrees of freedom
Residual deviance: 4701.4  on 5355  degrees of freedom
AIC: 4733.4

Number of Fisher Scoring iterations: 4
```

## 8.5 Link Functions

Here, we have used the logit link function, which is the most common. However, there are other functions that translate proportions to real numbers, and are sometimes used in regression for binary data. Two common options are:

- Probit regression: link='probit'
- Complementary log-log regression: link='cloglog'

There are not closed-form expressions for the the probit and complementary log-log functions that are easy to write down, so that is why the exact functions are not given here. As shown below, the shapes of these three functions are very similar. So it may come as no big surprise that frequently they provide similar goodness of fit to data (according to IC). If that is the case, choose logit (which makes some of the interpretation of results easier).



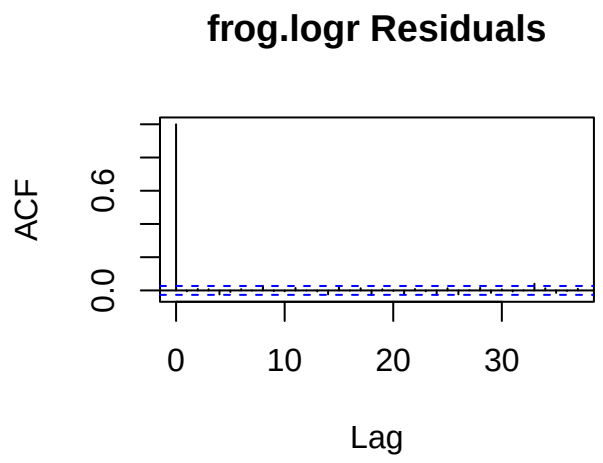

Note: figure is from [http://data.princeton.edu/wws509/notes].

## 8.6 Conditions

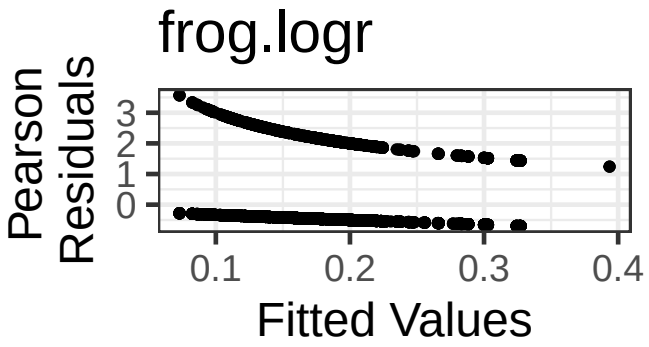
Under what conditions is a logistic regression model appropriate?

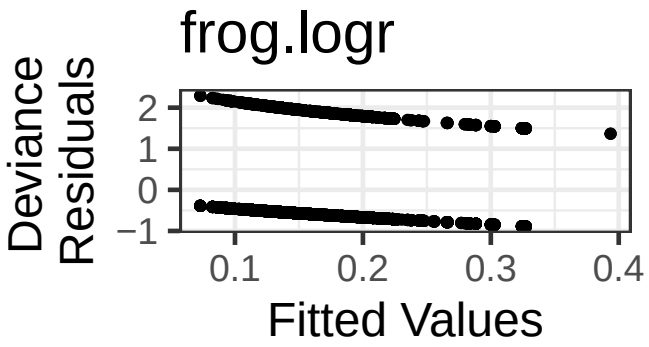
- Response variable is logical – you can characterize it as the outcome of a binomial trial (or a set of independent binomial trials). Some response variables can be expressed as proportions, but can *not* be well modelled with binomial regression. For example, you might take one-minute recordings in the woods and measure the proportion of each minute during which bird song was audible. The data will look like proportions, but you can't think of them as binomial trials and should not model them with binomial regression (what is a "trial" here, and what is a "success"? Make sure you can answer those questions before using binomial regression.)
- Linearity: logit(p) should have a linear relationship with each predictor variable. (A bit hard to check - see solutions to HW8 for an example of how it can be done.)
- Independence: Same as usual.
- Mean-variance relationship: The Pearson or Deviance residuals will decrease as a function of fitted value, and should have approximately constant variance as a function of fitted value. But a residuals vs fitted plot is of almost no use to us – the examples later on show how you can expect it to look, and if it deviates from the expected appearance, try to figure out why and what is going on; but if it looks as expected, you can say "there is no evidence in this figure of a violation of the conditions of binary regression."
- NO distributional assumptions about residuals.

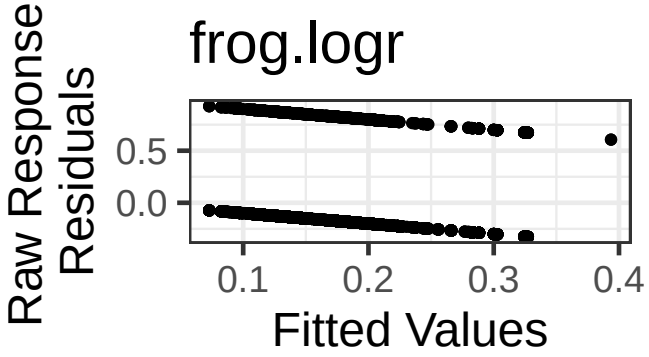
## 8.7 Model Assessment Plots

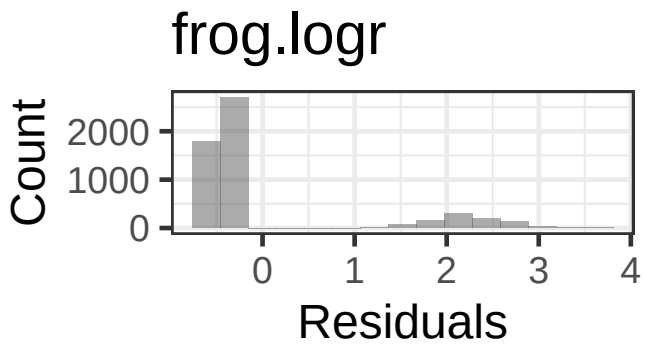



```
gf_point(resid(frog.logr, type='pearson') ~ fitted(frog.logr)) |>
  gf_labs(title='frog.logr',
          y=' Pearson\nResiduals',x='Fitted Values')

gf_point(resid(frog.logr, type='deviance') ~ fitted(frog.logr)) |>
  gf_labs(title='frog.logr',
          y=' Deviance\nResiduals',x='Fitted Values')

gf_point(resid(frog.logr, type='response') ~ fitted(frog.logr)) |>
  gf_labs(title='frog.logr',
          y=' Raw Response\nResiduals',x='Fitted Values')

gf_histogram(~resid(frog.logr, type='pearson'), bins=15) |>
  gf_labs(title='frog.logr',
          x='Residuals', y='Count')
```


frog.logr
Pearson Residuals
3
2
1
0
0.1
0.2
0.3
0.4
Fitted Values


frog.logr
Deviance Residuals
2
1
0
−1
0.1
0.2
0.3
0.4
Fitted Values


frog.logr
Raw Response Residuals
0.5
0.0
0.1
0.2
0.3
0.4
Fitted Values




The two "lines" in the residuals vs fitted plots correspond with the two possible values of the response variable in the data.

And remember - there is not a strict distributional assumption about the residuals (in other words, they don't have to follow, say, a normal distribution), so we don't really have to make a histogram of them. The one here is shown just to help you remember that you don't *have* to check it, and if you do, it will look "strange" (bimodal like this) yet it is nothing to worry about.

## 8.8 Odds Ratios

The **odds** (or *odds ratio*) is $\frac{p}{1-p}$ – the ratio of success to failure. So if P(success) = 0.75, then the odds will be $\frac{0.75}{0.25} = 3$ or "three to one" – you will usually succeed three times for every failure.

Remember, the logit function was $logit(x) = log(\frac{p}{1-p})$? In other words, the logit is the log of the odds ratio. This means that the coefficients of a binary regression model with logit link function have special interpretations in terms of odds ratios.

Let's consider a simplified version of our model (just to make it easier to write out the model equation):

```
simple <- glm(Abnormal ~ Stage, data = frogs, family=binomial(link='logit'))
coef(simple)
```

```
(Intercept) StageStage 43 StageStage 44 StageStage 45 StageStage 46
 -1.4618893     0.1001146    -0.1094112    -0.4527238    -0.5119712
```

So our model equation is:

$$logit(p_i) = -1.46 + 0.10I_{s43} - 0.11I_{s44} - 0.45I_{s45} - 0.51I_{s46}$$

According to this model, the log-odds (logit(p)) for a Stage 42 frog is -1.46, so the odds of being Abnormal for a Stage 42 frog are $e^{-1.46} = 0.23$.

The log-odds for a Stage 46 frog are -1.46 - 0.51 = -1.97, so the odds of it being Abnormal are $e^{-1.97} = 0.14$.

The *change* in odds going from Stage 42 to 46 is then $\frac{0.14}{0.23} = 0.61$ – the odds of a Stage 42 frog being abnormal are nearly double those of a Stage 46 frog.

Notice – we didn't actually have to compute all that to find the 0.6 value!

We know that for Stage 46

$$log(\frac{p}{1-p}) = -1.46 - 0.51$$

so

$$\frac{p}{1-p} = e^{-1.46-0.51} = e^{-1.46}e^{-0.51}$$

And $e^{-1.46}$ is the odds for Stage 42…aha! So, $e^{-0.51} = 0.60$ is the *multiplier* on the odds ratio to go from stage 42 to 46. And in general, $e^{\beta}$ is the multiplier on the odds ratio for a one-unit change in the predictor variable for which $\beta$ is the model coefficient.

## 8.9 Model Selection

As usual:

```
library(MuMIn)
frog.logr <- update(frog.logr, na.action='na.fail')
mod.sel <- dredge(frog.logr, rank='AIC')
```

```
Fixed term is "(Intercept)"
```

```
head(mod.sel,5)
```

```
Global model call: glm(formula = Abnormal ~ Stage + factor(Year) + RoadType + RoadDistance,
    family = binomial(link = "logit"), data = frogs, na.action = "na.fail")
---
Model selection table
     (Int) fct(Yer)       RdD RdT Stg df    logLik    AIC delta weight
16 -0.3509        + 0.0005653   +   + 16 -2350.719 4733.4  0.00  0.590
12 -0.3426        + 0.0005161       + 15 -2352.740 4735.5  2.04  0.213
14 -0.3749        +             +   + 15 -2353.232 4736.5  3.03  0.130
10 -0.3653        +                 + 14 -2354.889 4737.8  4.34  0.067
15 -1.4510          0.0004694   +   +  7 -2368.278 4750.6 17.12  0.000
Models ranked by AIC(x)
```

```
afm <- model.avg(mod.sel, fit=TRUE)
coef(afm)
```

```
     (Intercept) factor(Year)2001 factor(Year)2002 factor(Year)2003
   -0.3535582013    -1.4237089763    -1.3808976121    -0.9748675931
factor(Year)2004 factor(Year)2005 factor(Year)2006 factor(Year)2010
   -1.4331851168    -1.1210426509    -1.1258664109    -0.5125316516
factor(Year)2011 factor(Year)2012     RoadDistance    RoadTypePaved
   -1.1735417137    -1.0548360421     0.0005522245    -0.1658639112
   StageStage 43    StageStage 44    StageStage 45    StageStage 46
    0.1093083878    -0.1003833764    -0.4577677670    -0.5886749698
```

## 8.10 Prediction Plots

Shown here are example prediction plots for Stage and RoadDistance. First, check out a summary table for the variables in the model to help determine fixed values.

```
frogs$Year <- factor(frogs$Year)
summary(frogs[,c('RoadDistance', 'RoadType', 'Stage',
                 'Year')])
```

```
  RoadDistance      RoadType            Stage                Year
 Min.   :  3.00   Length:5371        Length:5371        2012   :1561
 1st Qu.: 15.00   Class :character   Class :character   2011   :1433
 Median : 31.00   Mode  :character   Mode  :character   2005   : 683
 Mean   : 99.38                                        2006   : 647
 3rd Qu.: 83.00                                        2004   : 605
```

```
 Max.   :781.00                                          2010   : 300
                                                         (Other): 142
```

```
library(s245)
# pred_plot(afm, 'Stage', data = frogs)
```

```
# pred_plot(afm, 'RoadDistance', data = frogs)
```

How does this compare to the raw data?

```
tally(~Abnormal|Stage, data=frogs, format='prop')
```

```
        Stage
Abnormal  Stage 42  Stage 43  Stage 44  Stage 45  Stage 46
     No  0.8118215 0.7960480 0.8279689 0.8715365 0.8780252
     Yes 0.1881785 0.2039520 0.1720311 0.1284635 0.1219748
```

```
bins <- cut(frogs$RoadDistance,breaks=c(0, 25, 50, 100, 250,800))
prop(~Abnormal=='Yes'|bins, data=frogs)
```

```
   prop_TRUE.(0,25]   prop_TRUE.(25,50]  prop_TRUE.(50,100] prop_TRUE.(100,250]
          0.1516080           0.1925424           0.1379747           0.1694215
prop_TRUE.(250,800]
          0.1613876
```

But…remember, in the raw data, other predictors may also be influencing the patterns that you see in the data. In addition, we can look at the width of the confidence bands on the model estimates, and look at the model selection results to get an idea of whether this predictor is really important in the model or not. This is just an example to get you thinking about what prediction plots are showing you and what you can do with them!

# 9 Binary regression: Data with more than one trial per row

So far, the dataset we used for binary regression had one "trial" per row: there was a categorical variable in the dataset with two categories, "success" and "failure" (for the frogs: Abnormal and Normal). We wanted to estimate the probability of "success" as a function of several predictors.

If there are multiple trials that all have the same predictor-variable values, we can group them together into one row of data, and just record the number of "trials" and the number of "successes". (From this, we can also get the number of "failures" = "trials" - "successes", if needed.) If we have a dataset that is stored in this format, we can still use `glm()` to fit a binary regression model. The R code to fit the model changes just a bit, and we are able to do better model assessment a bit more easily.

An example follows.

## 9.1 Data

The dataset used here is a reality-based simulated EIA dataset on duck sightings before and after windfarm installation (`impact`). Hourly, observers did up to 200 scans of the study area, and for each scan recorded whether duck(s) were seen (a success) or not (a failure).

The data file can be accessed online at:\ http://sldr.netlify.com/data/EIApropdata.csv

```
  successes trials day month impact
1       172    200   7     1      0
2       190    200  13     1      0
3       191    200  27     1      0
```

## 9.2 Checking the data setup

We would like to model the proportion scans with ducks sighted as a function of a set of covariates. Each row of our dataset gives us the number of successes in some number of trials

(and also gives the corresponding values of the covariates for ALL those trials). We can also use this kind of summary data with a logistic regression; we will just need to add a column for the number of failures:

```
pd <- pd |> mutate(failures = trials-successes)
#or same thing in base R
pd$failures = pd$trials - pd$successes
```

## 9.3 Fitting a saturated model

Let's try fitting a model for proportion sightings as a function of day, month, and impact.

We need a response "variable" that is really 2 variables bound together: a column with the "successes" and a column with the "failures". These don't have to be literally called successes and failures – you can use whatever variable names you like – but the first one of the two should be successes (the thing you want to compute the proportion for) and the second failures.

```
duck.logr <- glm( cbind(successes, failures) ~ day + month + impact,
                  family=binomial(link='logit'),
                  data=pd)
summary(duck.logr)
```

```
Call:
glm(formula = cbind(successes, failures) ~ day + month + impact,
    family = binomial(link = "logit"), data = pd)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  2.770286   0.077096  35.933  < 2e-16 ***
day         -0.011727   0.003564  -3.290    0.001 **
month       -0.091546   0.007819 -11.708  < 2e-16 ***
impact      -0.222642   0.048407  -4.599 4.24e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 488.19  on 71  degrees of freedom
Residual deviance: 244.96  on 68  degrees of freedom
```

```
AIC: 607.85

Number of Fisher Scoring iterations: 4
```

```r
#or maybe...
duck.logr2 <- glm( cbind(successes, failures) ~ day + factor(month) + impact,
                   family=binomial(link='logit'),
                   data=pd)
summary(duck.logr2)
```

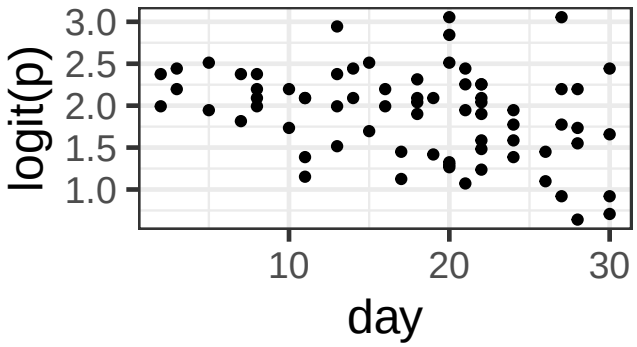
```
Call:
glm(formula = cbind(successes, failures) ~ day + factor(month) +
    impact, family = binomial(link = "logit"), data = pd)

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)      2.618402   0.123703  21.167  < 2e-16 ***
day             -0.011735   0.004058  -2.892 0.003829 **
factor(month)2  -0.297341   0.135091  -2.201 0.027734 *
factor(month)3  -0.014290   0.141766  -0.101 0.919712
factor(month)4  -0.287869   0.135761  -2.120 0.033971 *
factor(month)5  -0.258362   0.141490  -1.826 0.067850 .
factor(month)6  -0.404561   0.132914  -3.044 0.002336 **
factor(month)7  -0.141972   0.139208  -1.020 0.307799
factor(month)8  -0.452317   0.132294  -3.419 0.000628 ***
factor(month)9  -0.632855   0.128257  -4.934 8.05e-07 ***
factor(month)10 -0.665992   0.129788  -5.131 2.88e-07 ***
factor(month)11 -1.136006   0.123841  -9.173  < 2e-16 ***
factor(month)12 -0.949485   0.127997  -7.418 1.19e-13 ***
impact          -0.223631   0.048515  -4.609 4.04e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 488.19  on 71  degrees of freedom
Residual deviance: 197.84  on 58  degrees of freedom
AIC: 580.73

Number of Fisher Scoring iterations: 4
```

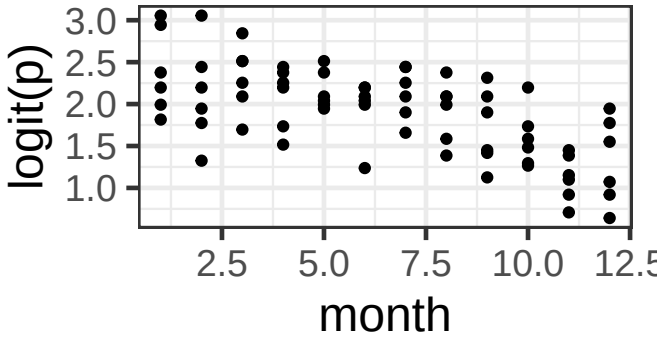
## 9.4 Checking linearity

What should be linear here? Well, `logit(p)` (where `p` is the probability of success, for a given set of predictor-variable values) should be a linear function of the predictors. *We can actually check this graphically now that we have multiple trials per row of data!* (But remember that the effects of other, unplotted predictors may also be influencing the plot that you see…)

Here, we need to decide: Do we see a linear pattern (or no pattern)? For the month and day data here, we might also consider whetehr it would make more sense to fit either of them as a categorical covariate rather than numeric.

```
gf_point(logit(successes/trials) ~ day, data=pd) |>
  gf_labs(y='logit(p)')
gf_point(logit(successes/trials) ~ month, data=pd)|>
  gf_labs(y='logit(p)')
```



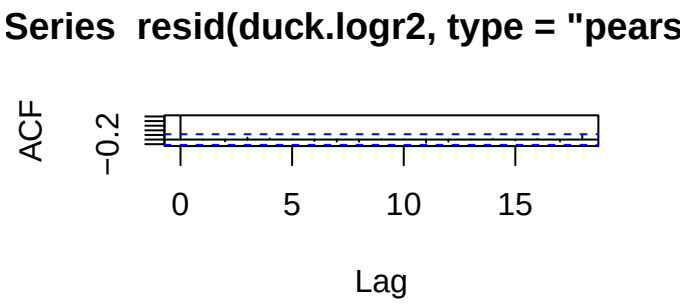



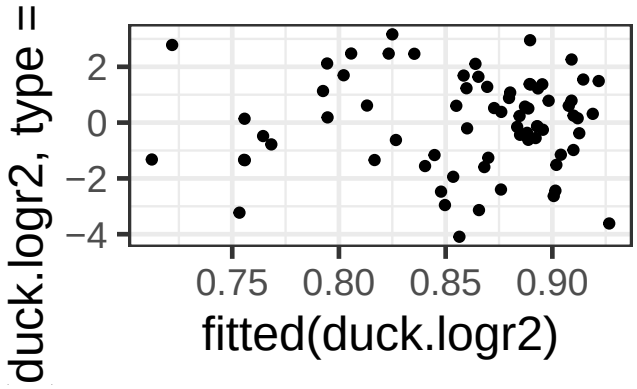
## 9.5 Model Assessment

With data set up as proportions (many trials with the number of successes and failures in each row, rather than one row per trial), model assessment plots are a bit more useful. Specifically, we can check the Pearson residuals vs. fitted plot for constant variance as a function of fitted value, to confirm that the mean-variance relationship matches what we expect.

Since the Pearson residuals are already adjusted for the expected variance, we should see approximately constant spread, with values ranging from about -2 to 2 (and not more than a few larger than $\pm$ 3).

```
acf(resid(duck.logr2, type='pearson'))
gf_point(resid(duck.logr2, type='pearson') ~ fitted(duck.logr2))
```



## 9.6 Model Selection

We can do model selection as usual. Here, it looks like the best model is the saturated (full) model.

```
library(MuMIn)
duck.logr2 <- update(duck.logr2, na.action='na.fail')
dredge(duck.logr2)
```

```
Fixed term is "(Intercept)"
```

```
Global model call: glm(formula = cbind(successes, failures) ~ day + factor(month) +
    impact, family = binomial(link = "logit"), data = pd, na.action = "na.fail")
---
Model selection table
  (Int)      day fct(mnt)     imp df   logLik  AICc  delta weight
8 2.618 -0.01173        + -0.2236 14 -276.365 588.1   0.00  0.934
7 2.430                 + -0.2235 13 -280.560 593.4   5.30  0.066
4 2.501 -0.01172        +         13 -287.029 606.3  18.24  0.000
3 2.314                 +         12 -291.218 611.7  23.63  0.000
6 2.444 -0.02848          -0.2204  3 -371.770 749.9 161.80  0.000
2 2.328 -0.02844                   2 -382.278 768.7 180.63  0.000
5 1.918                   -0.2192  2 -411.086 826.3 238.25  0.000
1 1.804                            1 -421.538 845.1 257.04  0.000
Models ranked by AICc(x)
```

We might also try using model selection to help us decide whether to use quantitative or categorical month and/or day…

```
duck.logr2 <- update(duck.logr2, formula= . ~ . + month + factor(day), na.action='na.fail'
dredge(duck.logr2, rank='BIC')
```

```
Fixed term is "(Intercept)"
```

```
Global model call: glm(formula = cbind(successes, failures) ~ day + factor(month) +
    impact + month + factor(day), family = binomial(link = "logit"),
    data = pd, na.action = "na.fail")
---
Model selection table
   (Int)      day fct(day) fct(mnt)     imp      mnt df   logLik   BIC  delta
```

```
14 2.618 -0.01173               + -0.2236          14 -276.365 612.6   0.00
30 2.618 -0.01173               + -0.2236          14 -276.365 612.6   0.00
13 2.430                        + -0.2235          13 -280.560 616.7   4.11
29 2.430                        + -0.2235          13 -280.560 616.7   4.11
26 2.770 -0.01173                 -0.2226 -0.09155  4 -299.926 617.0   4.36
15 2.313                +       + -0.2250          33 -238.023 617.2   4.57
16 2.342 -0.01464       +       + -0.2250          33 -238.023 617.2   4.57
31 2.313                +       + -0.2250          33 -238.023 617.2   4.57
32 2.342 -0.01464       +       + -0.2250          33 -238.023 617.2   4.57
25 2.631                          -0.2224 -0.10250  3 -305.371 623.6  10.97
6  2.501 -0.01172               +                  13 -287.029 629.7  17.05
22 2.501 -0.01172               +                  13 -287.029 629.7  17.05
5  2.314                        +                  12 -291.218 633.8  21.15
21 2.314                        +                  12 -291.218 633.8  21.15
18 2.653 -0.01171                         -0.09141  3 -310.543 633.9  21.31
7  2.195                +       +                  32 -248.752 634.4  21.75
23 2.195                +       +                  32 -248.752 634.4  21.75
8  2.224 -0.01459       +       +                  32 -248.752 634.4  21.75
24 2.224 -0.01459       +       +                  32 -248.752 634.4  21.75
27 2.884                +         -0.2237 -0.11950 24 -267.701 638.0  25.44
28 2.911 -0.01337       +         -0.2237 -0.11950 24 -267.701 638.0  25.44
17 2.514                                  -0.10230  2 -315.978 640.5  27.91
20 2.793 -0.01335       +                 -0.11930 23 -278.371 655.1  42.50
19 2.766                +                 -0.11930 23 -278.371 655.1  42.50
10 2.444 -0.02848                 -0.2204           3 -371.770 756.4 143.77
11 2.285                +         -0.2216          23 -335.678 769.7 157.12
12 2.347 -0.03102       +         -0.2216          23 -335.678 769.7 157.12
2  2.328 -0.02844                                   2 -382.278 773.1 160.51
3  2.170                +                         22 -346.244 786.6 173.97
4  2.232 -0.03097       +                         22 -346.244 786.6 173.97
9  1.918                          -0.2192           2 -411.086 830.7 218.12
1  1.804                                            1 -421.538 847.4 234.75
   weight
14  0.360
30  0.360
13  0.046
29  0.046
26  0.041
15  0.037
16  0.037
31  0.037
32  0.037
25  0.001
```

```
6   0.000
22  0.000
5   0.000
21  0.000
18  0.000
7   0.000
23  0.000
8   0.000
24  0.000
27  0.000
28  0.000
17  0.000
20  0.000
19  0.000
10  0.000
11  0.000
12  0.000
2   0.000
3   0.000
4   0.000
9   0.000
1   0.000
Models ranked by BIC(x)
```

Here it looks like day as quantitative and month as categorical works best.

# 10 Collinearity and Multicollinearity

Problematic collinearity and multicollinearity happen when two (collinearity) or more than two (multicollinearity) predictor variables are *highly* correlated with each other.

This can result in variance inflation: our uncertainty estimates (standard errors of coefficients, and confidence intervals on predictions) get bigger. Basically, if several predictors bear a lot of the same information, it can be hard for the model-fitting process to "allocate" the relationship between that predictor information and the response variable - it is "not sure" which predictor has which effect.

## 10.1 Graphical Checks

There are two functions in the package `GGally` that we can use to visualize the level of correlation between predictor variables.

As a simple example, let's consider a small dataset on blood pressure and other variables for 20 adults with high blood pressure. It contains variables:

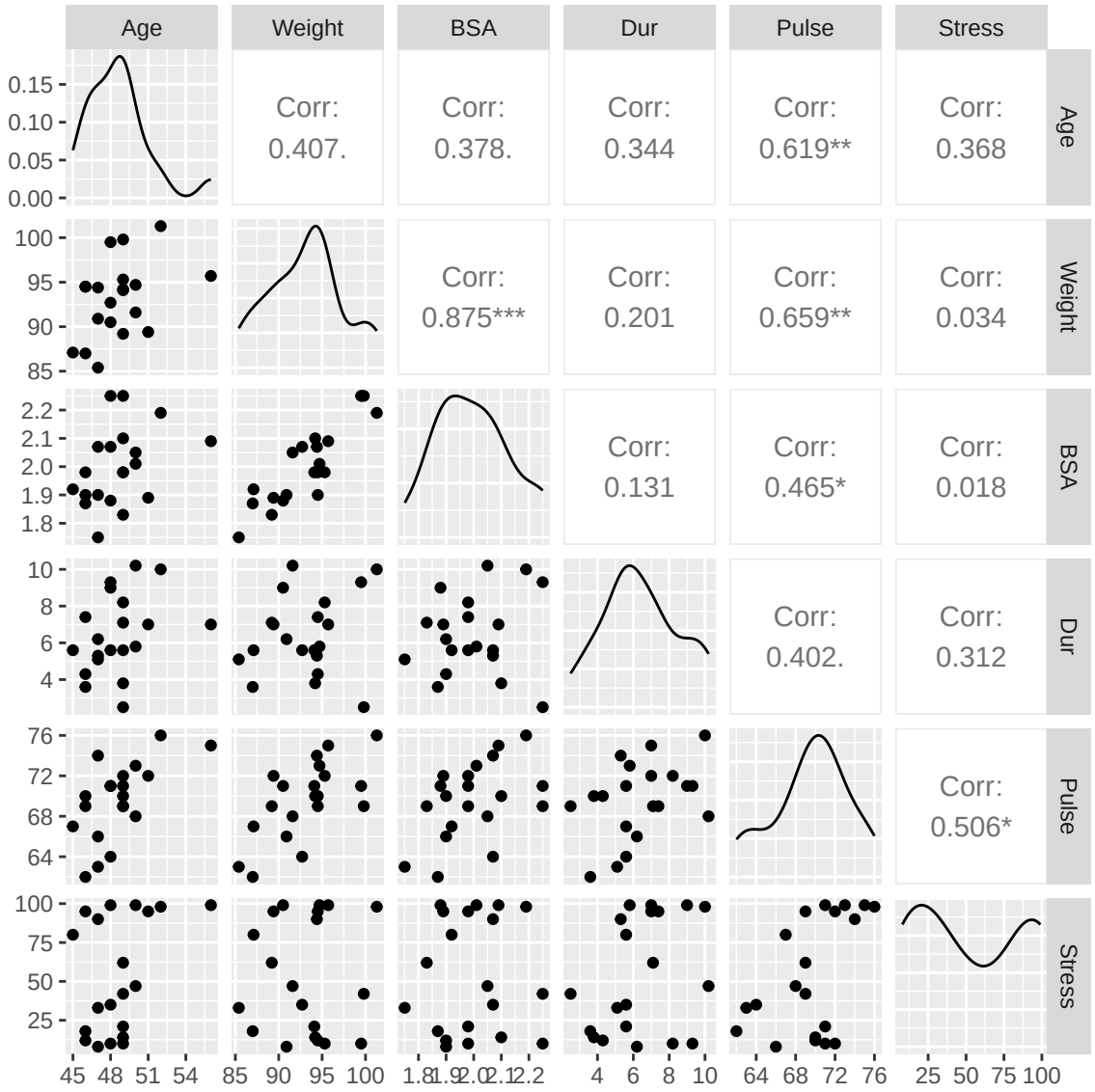
- patient number (`Pt`)
- blood pressure (`BP` in mm Hg)
- `Age` in years
- Weight, in kg
- body surface area (`BSA`, in $m^2$)
- duration of hypertension (`Dur`, in years)
- basal pulse (`Pulse`, in beats per minute)
- stress index (`Stress`)

```
bp <- read.table('http://sldr.netlify.com/data/bloodpress.txt',
                 header=TRUE)
```

Imagine we want to predict BP as a function of the other variables in the dataset. How can we use a plot to check whether any of the predictors at (too) correlated with each other?

### 10.1.1 Preferred option: Correlation Scatter Plot

```
library(GGally)
bp2 <- bp |> dplyr::select(Age, Weight, BSA, Dur, Pulse, Stress)
ggpairs(bp2)
```



### 10.1.2 Another option: Heat map of correlation coefficients

```
# label=TRUE input adds numeric corr coef labels
ggcorr(bp2, label=TRUE)
```


### 10.1.3 How to use this information

But how do we know how much correlation is *too much*? Some people suggest excluding predictors from a model if their pairwise correlation is above about 0.8, but that is a very arbitrary rule; sometime model estimate works just fine in the face of that much correlation. In other cases correlation of each pair of variables is low, but there is multicollinearity (a group of variables that predicts one other variable very well). Having a look at the data is always useful, but what else can we do to decide which predictors to include or exclude?

So, one solution is to decide if there is "too much" collinearity or multicollinearity, and if there *is* "too much" for a given predictor, remove that one from the model.

*There are also other options - model fitting techinques that get around the problem in other ways. (Key words in this arena are things like "lasso" and "neural net regression"). We will likely not have time to consider these in our class.*

## 10.2 Variance Inflation Factors

A more precise way to check for collinearity *and* multicollinearity is to use VIFs. To get them, we will use the function `vif()` from package `car` (Companion to Applied Regression).

But…what are the actually measuring?

$$VIF = \frac{1}{1 - R_i^2}$$

Where

$$R_i^2$$

is the $R^2$ value for a regression model with the $i$th predictor as the *response* variable, and all the *other* predictors as a predictor. So $R_i^2$ measures how well values of *predictor* $i$ can be estimated based on the *other* predictors. High $R_i^2$ indicates a predictor that is redundant, providing the same information as other predictors.

Since $R_i^2$ is in the *denominator* of the $VIF$ expression, **larger** VIFs indicate **more** problematic correlations between predictors.

*Note: an alternate interpretation of VIFs is that they measure the factor by which variance of model coefficient estimates is "inflated" due to collinearity between predictors. Big is still bad, because big VIF means inflated (high) uncertainty.*

### 10.2.1 Quantitative predictors (VIFs)

For our blood pressure example:

```
library(car)
m1 <- lm(BP ~ Age + Weight + BSA + Dur + Pulse + Stress, data=bp)
vif(m1)
```

```
     Age   Weight      BSA      Dur    Pulse   Stress 
1.762807 8.417035 5.328751 1.237309 4.413575 1.834845 
```

```
m2 <- lm(BP ~ Age + BSA + Dur + Pulse + Stress, data=bp)
vif(m2)
```

```
     Age      BSA      Dur    Pulse   Stress 
1.703115 1.428349 1.237151 2.360939 1.502936 
```

### 10.2.2 (Some) Categorical Predictors (GVIFs)

What if some predictors are categorical? In this case, `vif()` will report generalized VIFs (GVIFs) and scaled GVIFs ($GVIF^{\frac{1}{2DF}}$).

### 10.2.3 Rules of Thumb

If a VIF (or squared scaled GVIF) is 1, that means there is no added uncertainty in model estimates because of collinearity. If VIF (or squared scaled GVIF) is greater than 4, then there's a problem and you should probably try to fix it; if VIF (or squared scaled GVIF) is more than 10, then something *definitely* must be done to correct the problem.

*As suggested above…To use these rules of thumb with scaled GVIFs ($GVIF^{\frac{1}{2DF}}$),* **square** the scaled GVIF value before applying the rule.

Let's look at an example from the built-in R dataset Duncan, which has data from 1950 on characteristics of different professions (type ?Duncan in R if you want more info on the data).

If we only include quantiative predictors, we get VIFs as before:

```
m2 <- lm(prestige ~ income + education, data=Duncan)
vif(m2)
```

```
income education
2.1049    2.1049
```

If we include a categorical predictor, then we get GVIFs:

```
m3 <- lm(prestige ~ income + education + type, data=Duncan)
vif(m3)
```

```
             GVIF Df GVIF^(1/(2*Df))
income    2.209178  1        1.486330
education 5.297584  1        2.301648
type      5.098592  2        1.502666
```

In this case, it seems that the only apparent problem is with the `education` variable in the three-predictor model. We might try to correct this by excluding the education predictor. If we do, we see that the $R^2$ value of the model will not decrease by very much (since most of the information encoded in the education variable was already present in the others):

```
summary(m3)$r.squared
```

```
[1] 0.9130657
```

```
m4 <- update(m3, . ~ . -education)
summary(m4)$r.squared
```

```
[1] 0.8929864
```

## 10.3 Was it worth it?

(How would we have done at sorting this out using graphical methods only? Check by creating a pairwise scatterplot using the Duncan data on your own.)

# 11 Zero-Inflation

Zero-inflated data are count data that:

- Have a lot of zeros; **and**
- It may not be appropriate to explain the extra zeros via a model for count data with overdispersion (NB or quasi-Poisson) because:
  - There is too much overdispersion to account for this way
  - The zeros in the data are of two classes: *true* zeros (definite absence) and *other* zeros (might have been a positive count but just happened to be a 0 this time).

## 11.1 Reference material

On the topic of zero-inflated data (as on many other topics), your book is …terse! I have several copies of Mixed models and extensions for ecology in R if you want to borrow – You may want to check out Chapter 11 in that book.

## 11.2 Data for Example

The state wildlife biologists want to model how many fish are being caught by fishermen at a state park. Visitors are asked how long they stayed, how many people were in the group, were there children in the group and how many fish were caught. Some visitors do not fish, but there is no data on whether a person fished or not. Some visitors who did fish did not catch any fish so there are excess zeros in the data because of the people that did not fish.

We have data on 250 groups that went to a park. Each group was questioned about how many fish they caught (count), how many children were in the group (child), how many people were in the group (persons), and whether or not they brought a camper to the park (camper).

In addition to predicting the number of fish caught, there is interest in predicting the existence of excess zeros, i.e., the probability that a group caught zero fish. We will use the variables child, persons, and camper in our model. Let's look at the data.

```
zinb <- read.csv("https://stats.idre.ucla.edu/stat/data/fish.csv")
head(zinb)
```

```
  nofish livebait camper persons child         xb         zg count
1      1        0      0       1     0 -0.8963146  3.0504048     0
2      0        1      1       1     0 -0.5583450  1.7461489     0
3      0        1      0       1     0 -0.4017310  0.2799389     0
4      0        1      1       2     1 -0.9562981 -0.6015257     0
5      0        1      0       1     0  0.4368910  0.5277091     1
6      0        1      1       4     2  1.3944855 -0.7075348     0
```

```
zinb <- zinb |> mutate(nofish=factor(nofish),
                       livebait=factor(livebait),
                       livebait = ifelse(livebait==0, 'No', 'Yes'),
                       camper=factor(camper),
                       camper = ifelse(camper ==0, 'No', 'Yes'),
                       child_cat = factor(child))
head(zinb)
```

```
  nofish livebait camper persons child         xb         zg count child_cat
1      1       No     No       1     0 -0.8963146  3.0504048     0         0
2      0      Yes    Yes       1     0 -0.5583450  1.7461489     0         0
3      0      Yes     No       1     0 -0.4017310  0.2799389     0         0
4      0      Yes    Yes       2     1 -0.9562981 -0.6015257     0         1
5      0      Yes     No       1     0  0.4368910  0.5277091     1         0
6      0      Yes    Yes       4     2  1.3944855 -0.7075348     0         2
```

## 11.3 Visualization?

In class, we created several figures for data exploration. They are not printed here.

One idea was to make a pairs plot to look at possible collinearity. Good idea! But all our predictors are either yes/no or have very small ranges of integers as their values, so it doesn't prove very informative with our data - feel free to try for yourself to illustrate:

```
library(GGally)
ggpairs(zinb[, c('livebait', 'camper', 'persons', 'child')])
```

Another idea would be to plot response as a function of possible predictors, to check the linearity conditions of the parts of the model. This could be a good idea, but on the other

hand we don't have any predictors that are unequivocally numerical. (If we had one, we could plot log(count) as a function of the predictor.)

## 11.4 Collinearity/Multicollinearity?

We might want to check VIFs, but we can't compute them directly for a ZIP model (fitted later) using `vif()` from the `car` package.

However, note: VIFs really depend mainly on the covariates, not the model fitted. We can check them effectively by fitting the Poisson model (and the logistic regression, if we had more than one predictor in that part of the model) and checking **its** VIFs:

```
P <- glm(count ~ camper + child + persons, data=zinb, family=poisson)
library(car)
vif(P)
```

```
  camper    child  persons
1.008616 1.011875 1.008656
```

No evidence of any problem there!

## 11.5 Fitting models

### 11.5.1 Zero-inflated Poisson

```
library(pscl)
ZIP <- zeroinfl(count ~ child + camper + persons | 1, data = zinb,
                dist = 'poisson')
summary(ZIP)
```

```
Call:
zeroinfl(formula = count ~ child + camper + persons | 1, data = zinb,
    dist = "poisson")

Pearson residuals:
     Min       1Q   Median       3Q      Max
-1.08778 -0.77677 -0.57757 -0.09894 27.97095
```

```
Count model coefficients (poisson with log link):
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.99805    0.17557  -5.685 1.31e-08 ***
child       -1.36091    0.09410 -14.462  < 2e-16 ***
camperYes    0.79561    0.09411   8.454  < 2e-16 ***
persons      0.87214    0.04496  19.398  < 2e-16 ***

Zero-inflation model coefficients (binomial with logit link):
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.3659     0.1868  -1.958   0.0502 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Number of iterations in BFGS optimization: 10
Log-likelihood: -776.5 on 5 Df
```

### 11.5.2 Zero-inflated NB: one way

We can also use this function to fit a NB model. *BUT BEWARE:* this is a different parameterization of the NB model than the NB1 and NB2 models we use in `glmmTMB()` and may be closer to type 1 or type 2 depending on the dataset. You could compare all three NB versions via IC.

```
ZINBx <- zeroinfl(count ~ child + camper + persons | 1, data = zinb,
                  dist='negbin')
summary(ZINBx)
```

```
Call:
zeroinfl(formula = count ~ child + camper + persons | 1, data = zinb,
    dist = "negbin")

Pearson residuals:
     Min       1Q   Median       3Q      Max
-0.66364 -0.54174 -0.42090 -0.03142 20.44992

Count model coefficients (negbin with log link):
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.6250     0.3294  -4.933 8.09e-07 ***
child        -1.7805     0.1920  -9.272  < 2e-16 ***
```

```
camperYes     0.6211     0.2358   2.634  0.00844 **
persons       1.0608     0.1175   9.030  < 2e-16 ***
Log(theta)   -0.7689     0.1539  -4.998 5.80e-07 ***

Zero-inflation model coefficients (binomial with logit link):
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -13.63     217.63  -0.063     0.95
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Theta = 0.4635
Number of iterations in BFGS optimization: 40
Log-likelihood: -405.2 on 6 Df
```

#### 11.5.2.1 What do the `summary()` outputs mean?

The second part of the output (labelled "Zero Inflation model coefficients") is from a logistic regression with one predictor (persons) to predict the probability of an observation being a "true zero" - a group that was not in the park to fish. (We can't really tell from thsi output whether being there to fish, or not, is considered a true zero by the model...I figured it out by reading the help file for zeroinfl and looking at prediction plots below). The first part of the out put (labelled "Count model coefficients") is a Poisson regression fitted to the count data for the proportion of the groups in the data set who were *not* true zeros (and were thus actually there to fish). Both parts of the model can have predictors included; to have no predictors in the logistic regression part, you would use a formula like **y ~ x1 + x2 ... + xn | 1**, and to include the *same* predictors for both the Poisson and logistic parts, you exclude the **|...** part, for example ** y ~ x1+x2** is the same as **y ~ x1 + x2 | x1 + x2**.

### 11.5.3 Zero-inflated negative binomial (other way)

```r
library(glmmTMB)
ZINB1 <- glmmTMB(count ~ child + camper + persons, ziformula = ~1, data=zinb, family=nbino
ZINB2 <- glmmTMB(count ~ child + camper + persons, ziformula = ~1, data=zinb, family=nbino
summary(ZINB1)
```

```
 Family: nbinom1  ( log )
Formula:          count ~ child + camper + persons
Zero inflation:         ~1
Data: zinb
```

```
     AIC      BIC   logLik deviance df.resid
   823.7    844.8   -405.8    811.7      244


Dispersion parameter for nbinom1 family (): 9.21

Conditional model:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.82828    0.32013  -2.587 0.009673 **
child       -1.42884    0.16569  -8.624  < 2e-16 ***
camperYes    0.72240    0.19303   3.742 0.000182 ***
persons      0.77145    0.08394   9.190  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Zero-inflation model:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)    -20.1     4603.4  -0.004    0.997
```

Note the difference from the `zeroinfl()` model formula and function call. There is an separate input, `ziformula`, to add zero inflation to a `glmmTMB()` model. (The default is no zero inflation, if you do not include a `ziformula`). The zero-inflation model always uses a logit link.

### 11.5.4 Tweedie Model

Another option for this type of data might be to try a Tweedie model, because it also has a probability mass at 0 (in other words, allows for zero inflation). The Tweedie distributions we are considering are not exactly perfect for count data since the positive part of the distribution is like a gamma distribution (continuous, not counts), but the shape and support of the distribution are pretty close to our data. Let's try:

```
ZIT <- glmmTMB(count ~ child + camper + persons, data=zinb, family=tweedie(link = 'log'))
summary(ZIT)
```

```
 Family: tweedie  ( log )
Formula:          count ~ child + camper + persons
Data: zinb

     AIC      BIC   logLik deviance df.resid
   857.6    878.8   -422.8    845.6      244
```

```
Dispersion parameter for tweedie family (): 3.95

Conditional model:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.75575    0.32878  -5.340 9.29e-08 ***
child       -1.75659    0.18369  -9.563  < 2e-16 ***
camperYes    0.76704    0.22200   3.455  0.00055 ***
persons      1.06495    0.09683  10.998  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## 11.6 Model Assessment

We can do model assessment "like" we do for other count data models…more or less. (It is much harder to get something like Pearson residuals for these models, though…) *Please stay tuned!*

## 11.7 Model Selection?

One thing to understand is: is the ZIP model a better fit to the data than just a Poisson model? We can think of ZI as a different way of modelling overdispersion, an alternative to NB or quasi-Poisson models when we know there are true zeros in the data set.

```
P <- glm(count ~ camper + child + persons, data=zinb, family=poisson)
NB1 <- glmmTMB(count ~ camper + child + persons, data=zinb,
               family=nbinom1(link='log'))
NB2 <- glmmTMB(count ~ camper + child + persons, data=zinb,
               family=nbinom2(link='log'))
AIC(ZIP,P, NB1, NB2)
```

```
    df       AIC
ZIP  5 1562.9324
P    4 1682.1450
NB1  5  821.6926
NB2  5  820.4440
```

Well, yes, the ZIP model is much better than the Poisson! But, the NB1 and NB2 models are *much* better than both.

Among the different families and types of models that we have tried, which seems to be best for this data set?

```
BIC(P, ZIP, ZINB1, ZINB2, ZINBx, ZIT)
```

```
      df       BIC
P      4 1696.2308
ZIP    5 1580.5397
ZINB1  6  844.8214
ZINB2  6  843.5728
ZINBx  6  843.5728
ZIT    6  878.7701
```

For these data, the zero-inflated models are very similar to the other NB models, in terms of the AIC. If we used BIC instead, the non-zero inflated models would look quite a bit better! The improvement in goodness of fit with the zero-inflation is not really enough to make us prefer these ZI models to simpler models that also allow overdispersion, like the NB models.

The NB models and the ZI models (with whatever "family" you choose) are two *different* ways of dealing with overdispersion caused by an abundance of zeros in the data. I'd recommend choosing ZI models if:

- There are *true zeros* and *other zeros* in your data, and you want to model this explicitly; or if
- The ZI models fit the data much better than a simpler non-ZI NB (or perhaps Tweedie) model.

The zero-inflated negative binomial models seem to be best, among the ZI models. However, there is still a lot more work that could be done here! For example…

### 11.7.1 Zero inflation covariates

what if the Logistic regression part of the model *also* depends on covariates? That seems reasonable – maybe the number of people present, whether they are camping, and the number of kids present would help you to predict whether a group was not going to go fishing at all. To check that, we would need to change the `ziformula` (in NB models) or the part of the formula after the | in ZIP models. It is possible this change **might** make the ZI models competitive with, or better than, the non-ZI NB models in IC terms.

### 11.7.2 Interaction terms

For this dataset, it may also make sense to consider some more predictors, and interaction terms. For example, maybe live bait use is a good predictor, and maybe the relationship between live bait use and number of fish caught changes depending on the number of children present – for example, we might try something like:

```
ZINB1b <- glmmTMB(count ~ camper + child*livebait + persons, ziformula= ~1,
                  data=zinb, family=nbinom1(link='log'))
summary(ZINB1b)
```

```
 Family: nbinom1  ( log )
Formula:          count ~ camper + child * livebait + persons
Zero inflation:         ~1
Data: zinb

     AIC      BIC   logLik deviance df.resid
   818.0    846.2   -401.0    802.0      242


Dispersion parameter for nbinom1 family (): 8.47

Conditional model:
                   Estimate Std. Error z value Pr(>|z|)
(Intercept)       -1.648964   0.457499  -3.604 0.000313 ***
camperYes          0.649637   0.191185   3.398 0.000679 ***
child             -1.469266   0.643409  -2.284 0.022397 *
livebaitYes        0.900232   0.352927   2.551 0.010749 *
persons            0.790689   0.082140   9.626  < 2e-16 ***
child:livebaitYes -0.008494   0.663504  -0.013 0.989786
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Zero-inflation model:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -19.73    4306.92  -0.005    0.996
```

```
AIC(ZINB1, ZINB1b)
```

```
      df      AIC
```

```
ZINB1   6 823.6926
ZINB1b  8 817.9976
```

Hmmm…it looks like we made an improvement. But do you think the improvement is because of the interaction term?

Looking at the p-values, we might suspect that while `livebait` helps predict fish caught, there isn't a clear interaction between effects of live bait use and the number of kids present.

In this case, it is hard to fathom all the model selection options because there are predictors in *both parts* of the model.

If you continue by hand, one thing to remember is: You can **never** include an interaction term in a model without also including the corresponding "main effect" term. For example, you *can not* have an interaction between `child` and `livebait` without also having both `child` and `livebait` as individual predictors in the model. (In fact, I have intentionally not taught you how to fit the "wrong" model…as long as you use the * to specify an interaction in your model formula, you should be fine!)

### 11.7.3 Dredge

We *can* also use dredge(), but:

- It will only put covariates in/out of the model, not compare zero-inflated with non-zero-inflated models.
- It *will* put covariates in and out of both the zero-inflation and regular parts of the model.

So to do a comprehensive comparison, you would need to start with several full models – one for each "family" you want to try, with and without ZI, and with all possible predictors included for both the count and ZI parts of the models – and compare dredge results for all of them.

`dredge()` is also nice and smart about interaction terms – it will never include an interaction unless the main effect terms are also present.

```
library(MuMIn)
ZINB1c <- glmmTMB(count ~ child_cat*livebait + camper + persons,
                  ziformula = ~ child_cat + camper + persons,
                  data=zinb, family=nbinom1(link='log'), na.action = 'na.fail')
ms <- dredge(ZINB1c, rank='AIC')
head(ms)
```

```
Global model call: glmmTMB(formula = count ~ child_cat * livebait + camper + persons,
    data = zinb, family = nbinom1(link = "log"), ziformula = ~child_cat +
        camper + persons, na.action = "na.fail", dispformula = ~1)
---
Model selection table
    cnd((Int)) zi((Int)) dsp((Int)) cnd(cmp) cnd(chl_cat) cnd(lvb) cnd(prs)
16      -1.655    -20.13          +        +            +        +   0.7894
144     -1.655    -14.99          +        +            +        +   0.7894
48      -1.655    -19.44          +        +            +        +   0.7894
176     -1.655    -25.77          +        +            +        +   0.7894
32      -1.685    -24.09          +        +            +        +   0.7890
80      -1.655    -23.88          +        +            +        +   0.7894
    cnd(chl_cat:lvb) zi(cmp) zi(chl_cat) zi(prs) df   logLik   AIC delta weight
16                                                9 -399.843 817.7  0.00  0.504
144                                         -3.018 10 -399.843 819.7  2.00  0.186
48                          +                     10 -399.843 819.7  2.00  0.186
176                         +               -4.369 11 -399.843 821.7  4.00  0.068
32                  +                             12 -399.634 823.3  5.58  0.031
80                                      +         12 -399.843 823.7  6.00  0.025
Models ranked by AIC(x)
```

In this case, which model do you think is best? What else might you try?

## 11.8 Prediction Plots

We can also use prediction plots as usual. Here, they will help us clarify what "success" is in the logistic regression part of the model – according to the help file for the `zeroinfl()` function, we should be able to see that a "success" is to be a "true zero".

Let's try an example - making prediction plots for our ZIP model for different numbers of persons in the group.

First let's check the range of the data for different variables:

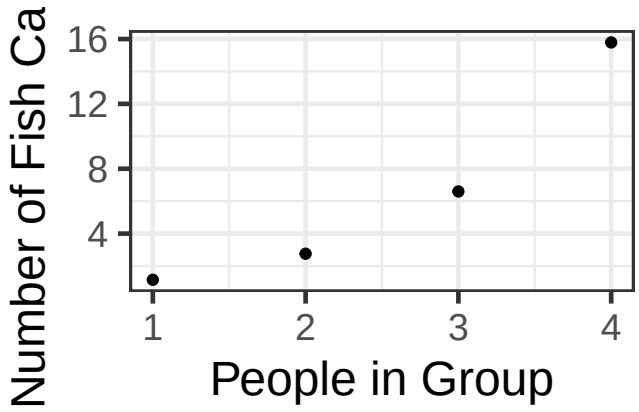
```
summary(zinb[,c('camper', 'child', 'persons')])
```

```
   camper              child          persons
 Length:250         Min.   :0.000   Min.   :1.000
 Class :character   1st Qu.:0.000   1st Qu.:2.000
 Mode  :character   Median :0.000   Median :2.000
                    Mean   :0.684   Mean   :2.528
```

```
 3rd Qu.:1.000   3rd Qu.:4.000
 Max.   :3.000   Max.   :4.000
```

Now, the prediction plots:

```
new_data <- data.frame(persons=c(1,2,3,4),
                       camper='Yes',
                       child=0)
pred <- predict(ZIP, newdata=new_data)
gf_point(pred~persons, data=new_data) |>
  gf_labs(x='People in Group', y='Number of Fish Caught')
```




The number of fish caught goes *up* as the number of people in the group goes up. But persons was a predictor for the logistic regression (zero-inflation part) of the model! And its coefficient was estimated to have a negative value. So according to our model, the more persons in the group, the less likely you are to be a "true zero" (and the more likely you are to be in the park to fish). Therefore, the number of fish your group catches goes up.

However, notice that unfortunately, the `predict()` function for `zerofinl()` ZIP models does not return standard errors on the fitted values. Shucks! We would have to compute these ourselves to show uncertainty on the prediction plot. We could use something called a parametric or nonparametric bootstrap to do it. We may learn how to do this later on in the course, but for now we don't have a tool to do it.

If you use one of the `glmmTMB()` models, though, you should be able to make prediction plots with uncertainty as usual.

## 11.9 Acknowledgements

Many thanks to: https://stats.idre.ucla.edu/r/dae/zip/, a source of the data set and information used in preparing these notes.

# 12 Non-constant variance (and other unresolved problems)

So far, we have worked to assemble a tool-kit that allows us to fit appropriate regression models for a variety of types of response and predictor variables. If we fit an appropriate model (conditions are met) to an appropriate dataset (representative sample of the population of interest), we should be able to draw valid conclusions – but as we have seen, if conditions are *not* met, then our conclusions (model predictions, judgements about which predictors are important, etc.) may all be unreliable.

So what can we do if we've fit the best model we know how to fit, and it's still not quite right? We have seen a number of examples so far where, even after we fit a model with what seem to be the right family and sensible predictors, conditions are not met. The most common problems are non-constant variance of residuals (often seen with non-normal residuals), and non-independent residuals. Solutions for non-independent residuals are coming soon, in future sections. Here we'll review options (some already in our tool-kit, and some new) to try to improve the situation when non-constant variance (NCV) is present.

What can we try if a model seems to fit data well, except for the fact that the constant variance condition doesn't hold (for linear regression) or the mean-variance relationship is not as expected (GLMs etc.)? (Note: this problem is also often accompanied by right skew in the distribution of the residuals.) Possible solutions are presented approximately in order of desirability…

### 12.0.1 Already in Our Tool Box: Make sure the model is "right"

Improving the model specification sometimes corrects a problem with non-constant variance. Make sure you have considered:

- Is the right "family" being used for the data type being modelled?
- Are **all** the variables that you think are important predictors (and are available to you in the data) included in the model?

### 12.0.2 Already in Our Tool Box: Models that estimate dispersion parameters

Negative binomial (and quasi-Poisson) models include estimation of a dispersion parameter that helps to account for over- or under-disperion (that is - the variance is larger, or smaller, than the mean value). If one of these models is being used (or is appropriate for the data) - NCV should be accounted for! (We can also verify this: If all is working well, the Pearson residuals will have constant variance.)

### 12.0.3 Gamma GLMs

**If** you are fitting a linear regression and:

- The response variable of interest is non-negative, **and**
- There is right skew in residuals **and/or**
- There is non-constant variance of residuals…

You may want to try fitting a model using the **Gamma** family (`link = 'log'` or `link = 'inverse'` link functions are the most common). If needed you can use model assessment plots and/or model selection criteria (AIC, BIC…) to decide between link functions. For example:

```
gamma.mod <- glm(resp ~ pred1 + pred2, data=dataset,
                 family=Gamma(link='log')) # or can use link='inverse'
```

### 12.0.4 Beta GLMs

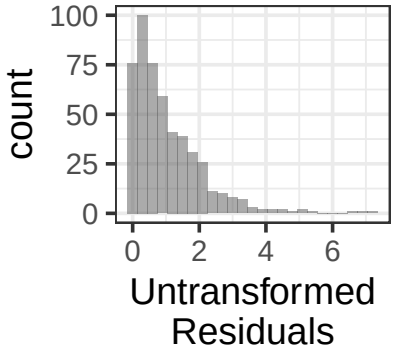
If your response variable happens to be bounded between 0-1 (but NOT a probability, so that a binary data regression would not be appropriate), you can also try a Beta regression (using the beta distribution). Since the shape of the distribution is extremely flexible, it will help with residuals whose distribution is not as expected. The variance of the beta distribution also *does* depend on its mean, so it is likely to be better than a linear model at addressing NCV. A code example is below. You must use `glmmTMB()` to fit this model.

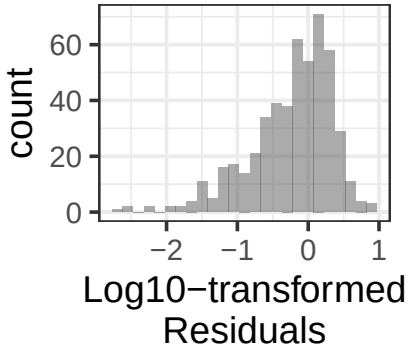
```
beta_model <- glmmTMB(response ~ predictor1 + predictor2,
                      data=my_data,
                      family=beta_family(link='logit'))
```

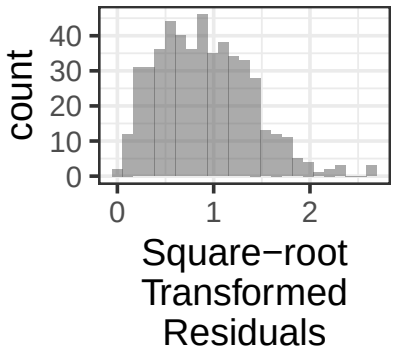
### 12.0.5 Transformations

In some cases, a logarithmic or square-root transformation of the **response variable** can correct a problem with the variance of the residuals. This is most sensible as a solution if the `log(variable)` or `sqrt(variable)` makes some "sense" to a human...for example, if the response variable is `wages` in dollars, then `log10(wages)` is kind of the order of magnitude of the salary, which makes some sense as a measure of income. Another example: sound pressure is measured in $\mu Pa$, but we perceive sound logarithmically, so that a sound that has 10 times greater pressure "sounds" about twice as loud. So `log10(sound_pressure)` could be a sensible metric.

Why does this work? These transformations don't affect the magnitude of *small* residual values very much, but they make *large* residuals get *a lot* smaller. The result is that the histogram of residuals has less right skew and the large-magnitude residuals (the ones causing the "flare" of the "trumpet" in the residuals vs. fitted plot) get much smaller.






### 12.0.6 Modelling non-constant variance

If your model is a linear regression, then there are a few specialized fitting functions that can allow you to fit models with non-constant variance of the residuals.

For multiple linear regression (original fit with **lm()**), you can use **gls()** from package **nlme** and add input *weights=varPower()* – inputs are otherwise the same as for **lm()**.

```
library(nlme)
nls(response ~ pred1 + pred2, data=dataset,
    weights=varPower())
```

*If you're interested in learning more about this method (and other options for specification of the form of the non-constant variance), see the references below in R (you will not be held responsible for the information contained in these help files for STAT 245 - just provided for further reference.)*

```
?nlme::nls
?nlme::varClasses
```

# 13 Random Effects

We have seen a number of cases where model residuals were not independent, violation regression model conditions. What kind of model can address this kind of *dependent* data? Hierarchical models - here, models including *random effects* - are one way to approach this problem. These kinds of models go by many names, including hierarchical models, multi-level models, random effects models,or mixed effects models.

## 13.1 Dataset

From: Falcone et al. 2017, http://rsos.royalsocietypublishing.org/content/royopensci/4/8/170629.full.pdf

Satellite tags were used to record dive data and movements of 16 Cuvier's beaked whales for up to 88 days each. The whales were incidentally exposed to different types of naval sonar exercises during the study period. How did characteristics of their dives change during sonar exposure? We will look specifically at shallow dive duration as a response variable.

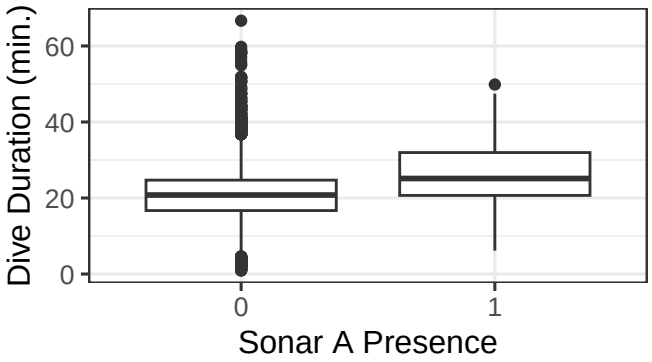
```
d <- read.csv('http://sldr.netlify.com/data/zshal.csv')
d$SonarA <- factor(d$SonarA)
d$SonarB <- factor(d$SonarB)
```

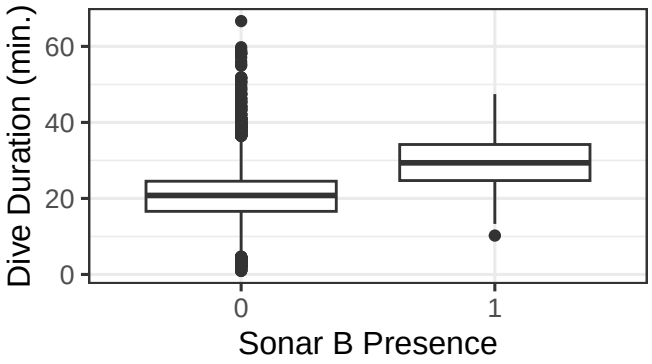
## 13.2 Data Exploration

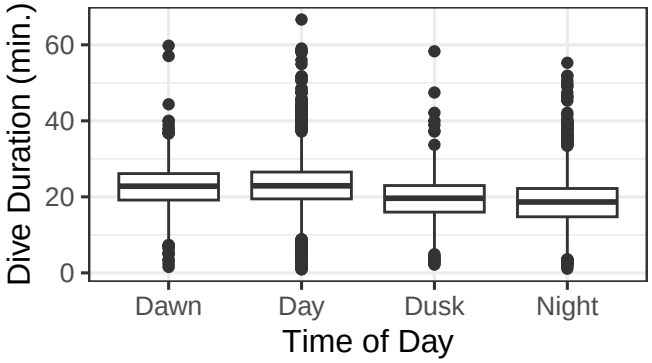
For these data, we are especially interested in how dive duration depends on sonar exposure. We also need to control for effects of other variables like depth and time of day.

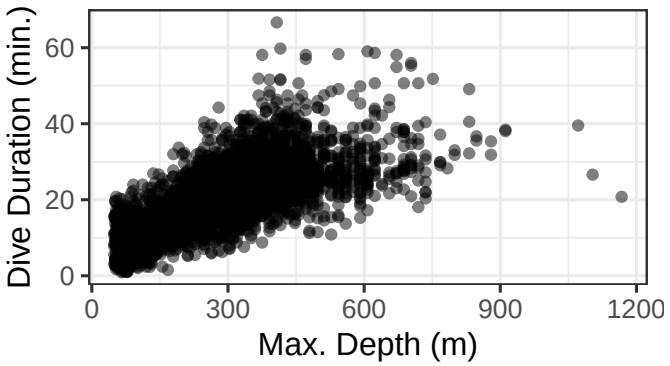
```
gf_boxplot(DurAvg ~ factor(SonarA), data=d) |>
  gf_labs(x='Sonar A Presence', y='Dive Duration (min.)')
gf_boxplot(DurAvg ~ factor(SonarB), data=d) |>
  gf_labs(x='Sonar B Presence', y='Dive Duration (min.)')
gf_boxplot(DurAvg ~ TransClass, data=d) |>
  gf_labs(x='Time of Day', y='Dive Duration (min.)')
gf_point(DurAvg ~ DepthAvg, data=d, alpha=0.5) |>
```

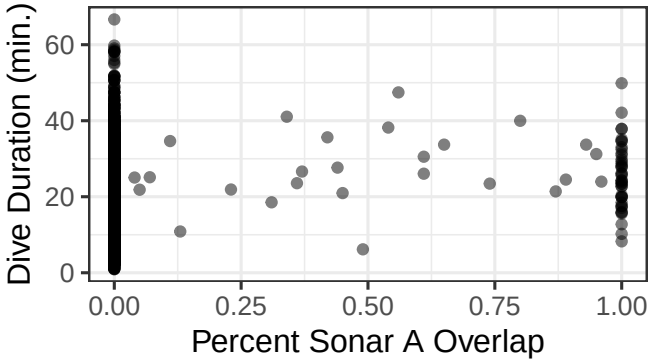
```
  gf_labs(x='Max. Depth (m)', y='Dive Duration (min.)')
gf_point(DurAvg ~ SonarAPercOL.fill, data=d, alpha=0.5) |>
  gf_labs(x='Percent Sonar A Overlap', y='Dive Duration (min.)')
gf_point(DurAvg ~ SonarBPercOL.fill, data=d, alpha=0.5) |>
  gf_labs(x='Percent Sonar B Overlap', y='Dive Duration (min.)')
```

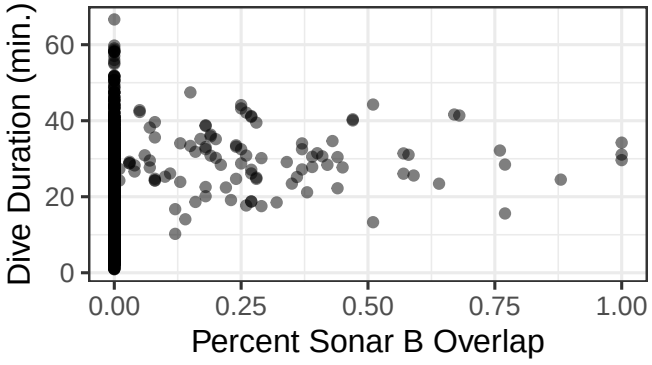





## 13.3 A Base Linear Model

A starting point for these data would be a basic linear regression, because the response variable is continuous, and we don't have strong indication of nonlinear predictor-response relationships.

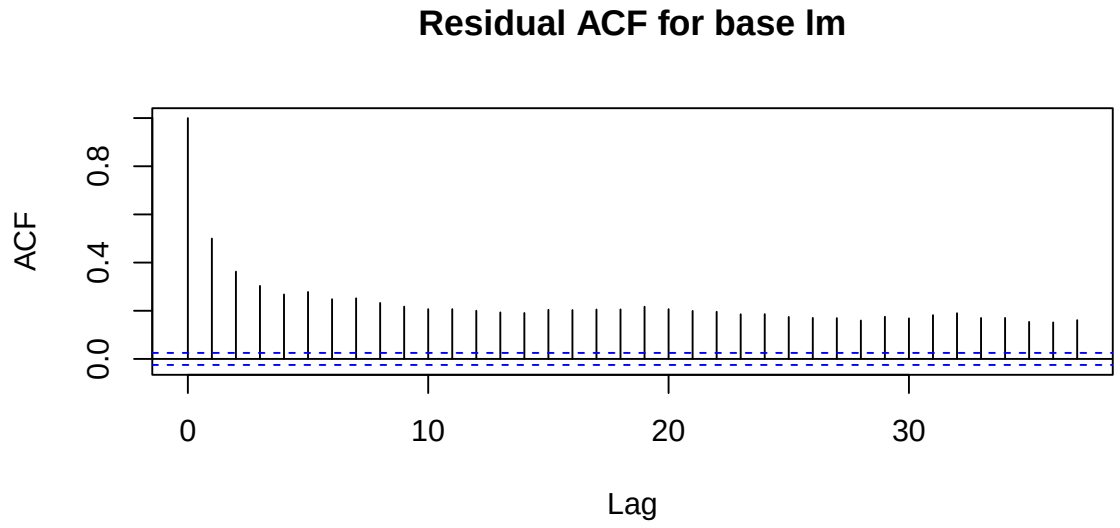
```
base.model <- lm(DurAvg ~ DepthAvg + TransClass +   SonarA +
                   SonarB +SonarAPercOL.fill +
                   SonarBPercOL.fill, data=d)
summary(base.model)
```

```
Call:
lm(formula = DurAvg ~ DepthAvg + TransClass + SonarA + SonarB +
    SonarAPercOL.fill + SonarBPercOL.fill, data = d)

Residuals:
    Min      1Q  Median      3Q     Max
-34.344  -3.174  -0.148   2.891  40.732

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)       11.0774181  0.3056179  36.246  < 2e-16 ***
DepthAvg           0.0384292  0.0005743  66.920  < 2e-16 ***
TransClassDay     -0.8195877  0.2718055  -3.015  0.00258 **
TransClassDusk    -2.1080411  0.3536873  -5.960 2.66e-09 ***
TransClassNight   -2.4488002  0.2708857  -9.040  < 2e-16 ***
SonarA1            2.6646627  1.8160219   1.467  0.14234
SonarB1            5.1562595  0.8556973   6.026 1.78e-09 ***
SonarAPercOL.fill  0.7555749  2.1077356   0.358  0.72000
SonarBPercOL.fill  0.8554161  2.2865382   0.374  0.70834
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.242 on 6174 degrees of freedom
Multiple R-squared:  0.4844,    Adjusted R-squared:  0.4837
F-statistic: 725.1 on 8 and 6174 DF,  p-value: < 2.2e-16
```

### 13.3.1 Model assessment

Let's take a look right away at the model assessment plot that we suspect will be problematic for time-series data like ours. As we fear…

```
acf(resid(base.model), main='Residual ACF for base lm')
```




## 13.4 A Random Effects model

This time we will try to account for the correlation over time within individuals using something called a *random effect* model (also known as a *mixed effects model*, *multilevel level*, among others). How does this model change our regression equation?

Recall that the form of a base linear model (with just 2 predictors) would be:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

Where $\epsilon \sim N(0, \sigma)$ are the normally distributed residuals with mean 0.

Now…

### 13.4.1 The Formula

The function to fit a linear random effect model is `lmer()`. For a Poisson or Logistic regression with random effects, it's `glmer()`. Both are from the package `lme4`. We add random effects to the model formula with:

$$+(1|variable)$$

or nested:

$$+(1|variable1/variable2)$$

Let's try a random effect of individual whale first. We have:

```
rem1 <- lmer(DurAvg ~ DepthAvg + TransClass +
  SonarA + SonarB +SonarAPercOL.fill+
    SonarBPercOL.fill + (1|TagID),
  data=d)
```

```
Warning: Some predictor variables are on very different scales: consider
rescaling
```

Why yes - we *should* consider rescaling...what/why/how?

```
d$SonarAPercScale <- scale(d$SonarAPercOL.fill)
d$SonarBPercScale <- scale(d$SonarBPercOL.fill)
rem2 <- lmer(DurAvg ~ DepthAvg + TransClass +
  SonarA + SonarB + SonarAPercScale +
    SonarBPercScale + (1|TagID),
  data=d)
```

### 13.4.2 The Results

```
summary(rem2)
```

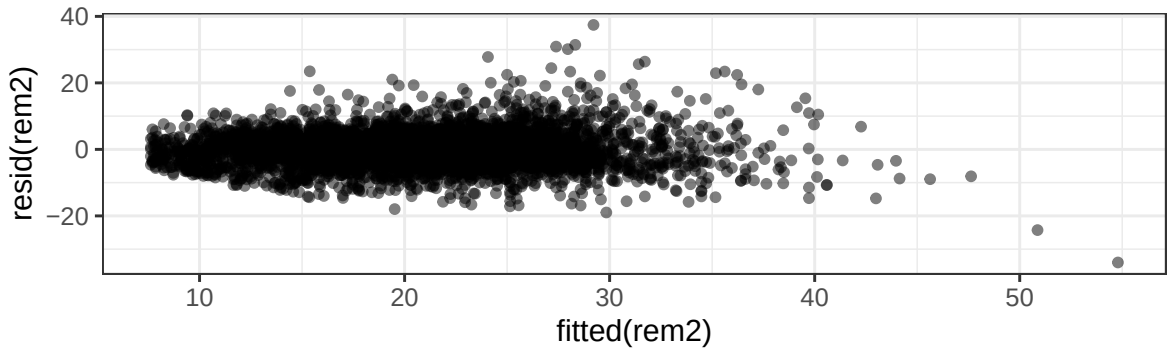
```
Linear mixed model fit by REML ['lmerMod']
Formula: DurAvg ~ DepthAvg + TransClass + SonarA + SonarB + SonarAPercScale +
    SonarBPercScale + (1 | TagID)
   Data: d

REML criterion at convergence: 36982.5

Scaled residuals:
    Min      1Q  Median      3Q     Max
-7.1028 -0.5632 -0.0081  0.5297  7.8183

Random effects:
 Groups   Name        Variance Std.Dev.
 TagID    (Intercept)  3.401   1.844
 Residual             22.928   4.788
Number of obs: 6183, groups:  TagID, 15

Fixed effects:
                  Estimate Std. Error t value
(Intercept)     11.0158506  0.5547014  19.859
DepthAvg         0.0391448  0.0005327  73.490
TransClassDay   -0.6416909  0.2490852  -2.576
TransClassDusk  -1.9171752  0.3235698  -5.925
TransClassNight -2.3456311  0.2479312  -9.461
SonarA1          3.4520183  1.6608961   2.078
SonarB1          4.5271312  0.7842742   5.772
SonarAPercScale  0.0362979  0.1716890   0.211
SonarBPercScale  0.1119528  0.0991365   1.129

Correlation of Fixed Effects:
            (Intr) DpthAv TrnsClssDy TrnsClssDs TrnsCN SonrA1 SonrB1 SnrAPS
DepthAvg    -0.282
TransClssDy -0.365 -0.055
TrnsClssDsk -0.308  0.051  0.650
TrnsClssNgh -0.410  0.097  0.844      0.658
SonarA1     -0.029 -0.011 -0.011      0.004     -0.001
SonarB1     -0.006 -0.026 -0.037     -0.026     -0.006 -0.032
SonrAPrcScl  0.029  0.009  0.001      0.001      0.001 -0.933 -0.002
SonrBPrcScl  0.017  0.005 -0.005      0.011      0.003  0.020 -0.785  0.000
```

How does this model compare to the original linear regression model? (Coefficient estimates? SEs? Additional stuff in the summary output?)

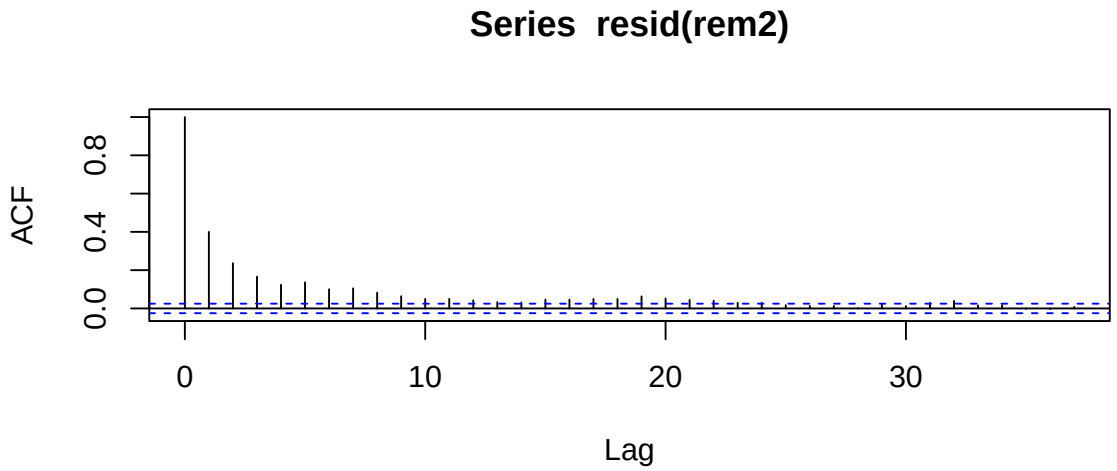
### 13.4.3 Model Assessment

How have the model assessment plots changed? Here we'll focus mainly on the problem ACF.

```
gf_point(resid(rem2)~fitted(rem2), alpha=0.5)
```




```
acf(resid(rem2))
```




### 13.4.4 Refinement

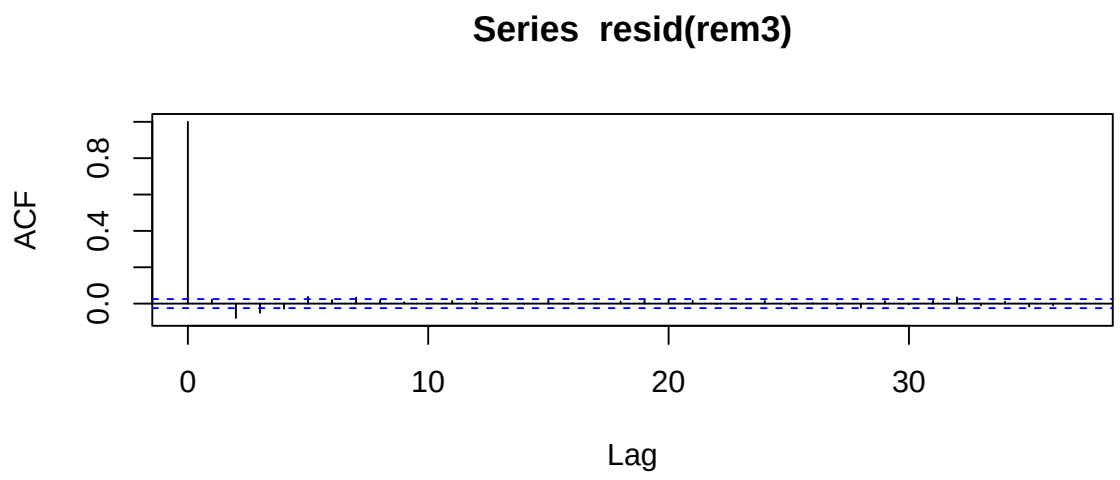
What can we try next?

```
head(d, 3)
```

```
  TagID DurAvg           StartTime DepthAvg TransClass SonarA SonarB
1    14  17.58 2011-01-06 20:45:30    335.5        Day      0      0
2    14  19.71 2011-01-06 22:13:23    351.5        Day      0      0
3    14  18.11 2011-01-06 22:34:48    287.5        Day      0      0
  SonarAMinKm.fill SonarBMinKm.fill SonarAPercOL.fill SonarBPercOL.fill
1              500              500                 0                 0
2              500              500                 0                 0
3              500              500                 0                 0
      TagDay  Period       TagDayPeriod SonarAPercScale SonarBPercScale
1 2011-01-06 (18,20] 2011-01-06.(18,20]     -0.09784411      -0.1020368
2 2011-01-06 (20,22] 2011-01-06.(20,22]     -0.09784411      -0.1020368
3 2011-01-06 (20,22] 2011-01-06.(20,22]     -0.09784411      -0.1020368
```

```
rem3 <- lmer(DurAvg ~ DepthAvg + TransClass + SonarA +
               SonarB +SonarAPercScale + SonarBPercScale +
               (1|TagID/TagDayPeriod), data=d)
```

```
acf(resid(rem3))
```



## 13.5 Model Selection for Mixed Models

Can we use our standard likelihood-based model selection criteria with random effects models?

Well…yes, and no.

### 13.5.1 REML or ML?

There are two different ways to fit these models to data:

- by maximizing the likelihood (ML, as we learned about earlier in the course). Unfortunately, it turns out that in this case, the ML estimates of the variance components (the random effects) is biased, toward underestimating variance, when sample size is small.
- by maximizing the restricted maximum likelihood (REML), which separates the likelihood into two parts (one with the fixed effects and one with the variance components). Maximizing parameters with respect to the second part only yields the REML estimators, which are unbiased and so preferred for smaller sample sizes. BUT there's a catch…REML values can be used to compare models with different error and random effects structures, but *not* to determine which predictor variables should remain in a best model.

Here, we do have a large sample size, so if we ensure our model is fitted by ML we can try using AIC or BIC for model selection. The default of `lmer()` and `glmer()` is to use REML, so if we want ML we have to add the input REML=FALSE to our call.

```
rem4 <- lmer(DurAvg ~ DepthAvg + TransClass + SonarA + SonarB +
               SonarAPercScale +  SonarBPercScale +
               (1|TagID/TagDayPeriod), data=d,
             na.action='na.fail', REML=FALSE)
```

```
Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.00309987 (tol = 0.002, component 1)
```

In doing model selection for random effects models, `dredge()` knows to keep the random effects terms present in all models, so we don't have to specify them as *fixed* terms.

```
library(MuMIn)
rem4_sel <- dredge(rem4, rank='BIC')
```

```
Fixed term is "(Intercept)"
```

```
Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
Model failed to converge with max|grad| = 0.00309987 (tol = 0.002, component 1)
```

```
head(rem4_sel)
```

```
Global model call: lmer(formula = DurAvg ~ DepthAvg + TransClass + SonarA + SonarB +
    SonarAPercScale + SonarBPercScale + (1 | TagID/TagDayPeriod),
    data = d, REML = FALSE, na.action = "na.fail")
---
Model selection table
   (Intrc)   DpthA SonrA  SnAPS SonrB   SnBPS TrnsC df    logLik     BIC delta
44   10.66 0.03985     +            +             + 10 -18081.06 36249.4  0.00
46   10.69 0.03986       0.2679     +             + 10 -18081.52 36250.3  0.92
42   10.68 0.03987                  +             +  9 -18088.50 36255.6  6.15
48   10.67 0.03985     + 0.1051     +             + 11 -18080.83 36257.7  8.28
60   10.66 0.03985     +            + 0.02154     + 11 -18081.03 36258.1  8.67
62   10.69 0.03986       0.2688     + 0.02007     + 11 -18081.50 36259.0  9.60
   weight
44  0.583
46  0.368
42  0.027
48  0.009
60  0.008
62  0.005
Models ranked by BIC(x)
Random terms (all models):
  1 | TagID/TagDayPeriod
```

### 13.5.2 Best model so far:

```
rem5 <- lmer(DurAvg ~ DepthAvg + TransClass + SonarA + SonarB +
               (1|TagID/TagDayPeriod), data=d,
             na.action='na.fail', REML=FALSE)
```

## 13.6 Random Slopes?

What we just practiced and called a "random effect" is sometimes also called a "random intercept" model because, although we allowed for an offset between the overall average predicted

response value and that of an individual, we did not allow the *slope* of the relationship with any of the predictor variables to vary randomly with individual. It is possible to do this, although in my experience it often makes interpretation difficult.

Before you do it, think to yourself: do you really think that there is random variation in the relationship of the predictor with the response? One case where random slopes will work well is where there is a strong, clear overall effect and small variations in its magnitude between individuals. Another might be where the relationship with a certain predictor has very strong and very different slopes for different individuals, and you want to account for the added variability this adds to the model.

In the `(g)lmer()` formula, a model with a random slope *and* intercept in relation to a particular predictor is specified with the form:

$$... + (PredictorVariable|GroupingVariable)$$

or equivalently

$$... + (1 + PredictorVariable|GroupingVariable)$$

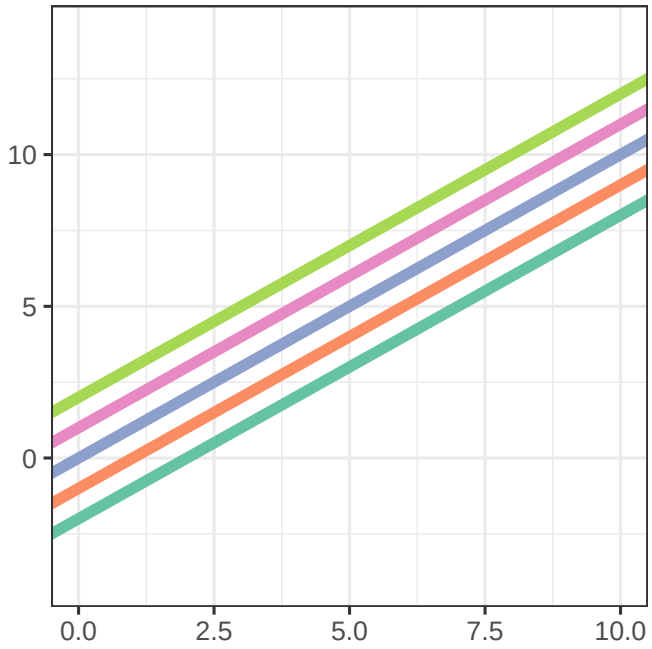
If you want to have a random slope for a certain predictor *without* the corresponding random intercept ( I can't think of an example where this would be a good idea but you can do it), then use:

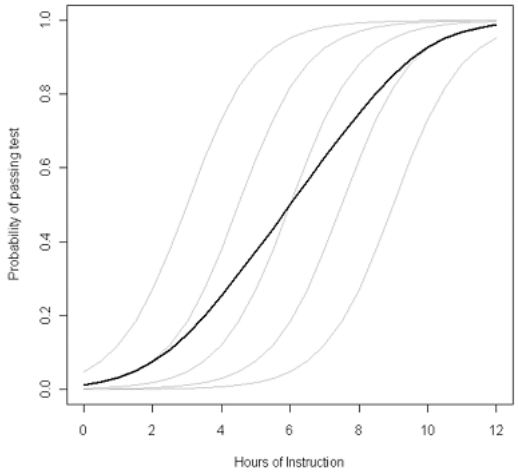
$$... + (0 + PredictorVariable|GroupingVariable)$$

## 13.7 Prediction Plots

There is a bit of added work involved in making prediction plots for some random effects models.

Unlike GEEs, which provide *marginal* predictions (predictions of the population average value for any combination of predictor variable values), random effects models provide predictions for an *average individual*. **For a linear regression model (or any model with the identity link function, that is, no link function), the predicted values for the population average and average individual are the same**. But with a link function in the mix, **it's different**. Consider a (hypothetical) example of a logistic regression modelling probability of passing a test as a function of hours of instruction spent before the test.




### 13.7.1 Parametric bootstrap to the rescue!

How can we get around this problem? We can make predictions from our model for many, many (simulated) individuals to get a "population" of predictions. Then, we can take a pointwise average over all those individuals (and also use them to find a CI), to get population average predictions and confidence intervals.

We can do this with help from the function `bootMer()` from the `lme4` package.

To make this work, we first need a **function** that makes **predictions from our model**.

```
# function to make predictions from a fitted model
library(s245)
# predict_rem4 <- function(model){
#   orig_dat <- model@frame
#   fixed_vals <- get_fixed(orig_dat[,c(2:ncol(orig_dat))])
#   new_dat <- get_new_data(orig_dat, predictor='SonarA', fixed_vals)
#   return(predict(model, newdata = new_dat,
#                  type = "response", allow.new.levels=TRUE))
# }
```

`bootMer()` does parametric bootstrap simulations and each time, computes some function of the fitted model (here, predictions.) We can then examine the quantiles of these bootstrap predictions (the median or mean is our *estimate* or best-guess predicted value, and the *2.5 and 97.5 percentiles* are the *bounds of a 95 percent CI*).

```
# boot_rem4 <- bootMer(rem4, FUN = predict_rem4, nsim = 1000,
#                      type = "parametric", use.u = FALSE)
```

```
# glimpse(boot_rem4$t )
```

```
# orig_dat <- rem4@frame
# fixed_vals <- get_fixed(orig_dat[,c(2:ncol(orig_dat))])
# new_dat <- get_new_data(orig_dat, predictor='SonarA',
#                           fixed_vals)
# new_dat <- new_dat |>
#   mutate(pred = apply(boot_rem4$t, 2, mean),
#          CIlow = apply(boot_rem4$t, 2, quantile, probs=0.025),
#          CIhigh = apply(boot_rem4$t, 2, quantile, probs=0.975)
#          )
#
# gf_point(pred ~ SonarA, data=new_dat) |>
#   gf_labs(x='Sonar A Presence', y='Dive Duration (min.)') |>
#   gf_errorbar(CIlow + CIhigh ~ SonarA, data=new_dat, width=0.3)
```

(Because...)

```
# pred_plot(rem4, 'SonarA')
```

# 14 Random effects with glmmTMB and standardized residuals

In the previous chapter we considered fitting mixed-effects models with `lme4`. What about fitting the same models with package `glmmTMB`? And how can we assess expectations about residual variance for models like these?

## 14.1 Model for whale dive duration

We start with our previous model for whale dive duration.

```
rem5 <- lmer(DurAvg ~ DepthAvg + TransClass + SonarA + SonarB +
               (1|TagID/TagDayPeriod), data=d,
             na.action='na.fail', REML=FALSE)
```

## 14.2 glmmTMB

We fitted these models using `lmer()` and `glmer()` from package `lme4`, which is probably the most commonly used R package to fit these models. But we've also used `glmmTMB` in this course, and *it* can also fit random effects models. The syntax to add random effects to a `glmmTMB()` model is *exactly the same* as for the `lme4` functions.

One difference is that for `glmmTMB()` **REML = FALSE is the default**.

`glmmTMB` may be faster in some cases.

```
library(glmmTMB)
rem6 <- glmmTMB(DurAvg ~ DepthAvg + TransClass + SonarA + SonarB +
               (1|TagID/TagDayPeriod), data=d,
             na.action='na.fail', REML=FALSE)
```

If you view the summaries, you will see that `rem5` and `rem6` are basically identical (in terms of model coefficients and variance estimates).

## 14.3 Model assessment with scaled residuals

For models where Pearson residuals are not helpful/can not be computed, one option is **scaled residuals** via simulation.

- Simulate many replicates from the fitted model corresponding to each data observation
- Examine the distribution of the simulated data, and use it to scale the observed value to get a "scaled residual" such that a residual of 0 means that *all simulated values are larger than the observed value*, and a *residual of 0.5 means half of the simulated values are larger than the observed value*, and a *residual of 1 means that the observed value is larger than all the simulated values.*

**These scaled residuals should be uniformly distributed between 0 and 1, if the model is correct.**

There is a much more detailed explanation in the DHARMa package documentation online.

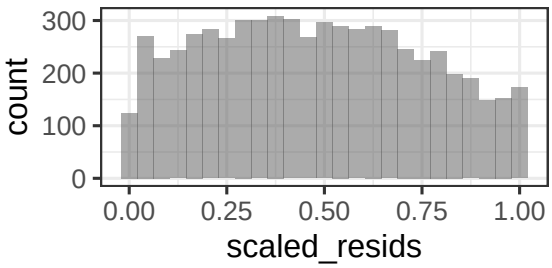
Example: computing scaled residuals, creating 1000 simulated datasets (don't lower it much below this to get good estimates). Note: use R chunk setting "message = FALSE" to print messages during simulation to your screen, rather than your knitted file.

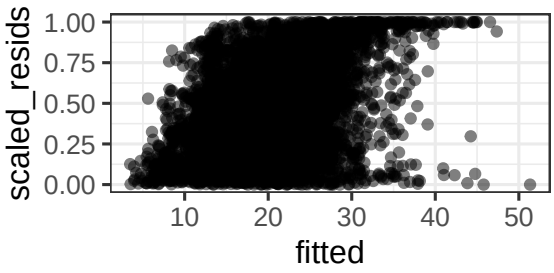
```
library(DHARMa)
sims <- simulateResiduals(fittedModel = rem5, n = 1000)
d <- d |>
  mutate(scaled_resids = sims$scaledResiduals,
         fitted = fitted(rem5))
```

If our model is perfectly correct, we would expect:

- Histogram of scaled residuals looks uniform between 0 and 1
- Plot of scaled residual vs. fitted, or vs. any predictor, look uniform (evenly spread, no trends, constant in variance).

```
gf_histogram(~scaled_resids, data =d)
gf_point(scaled_resids ~ fitted, data = d, alpha = 0.5)
```

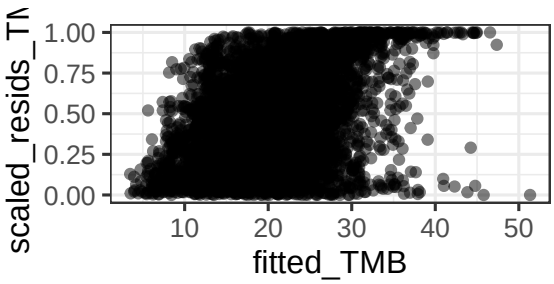





### 14.3.1 glmmTMB version

```
sims2 <- simulateResiduals(fittedModel = rem6, n = 1000)
d <- d |>
  mutate(scaled_resids_TMB = sims2$scaledResiduals,
         fitted_TMB = fitted(rem6))
```

```
gf_point(scaled_resids_TMB ~ fitted_TMB, data = d, alpha = 0.5)
```




### 14.3.2 You now have the power!

You may use these scaled residuals for any of the models considered so far this semester. It is basically an alternative method of scaling residuals so that we know "how they should look" in residual plots.

# 15 GEEs

So far, we looked at random effects as one way to account for non-independence of residuals due to a variable that we don't want to include as a regular predictor. Another option for this kind of situation is to use Generalized Estimating Equations (GEEs).

## 15.1 Data Source

The dataset used here is industry data from a skin care company. It contains data from experiments with 20 subjects. Each person tested 6 different skin moisturizers, and the hydration level of their skin was measured every 2 hours for 24 hours following application of each product. The variables are:

- `Subjects` Numeric code identifying the person
- `CorneoDiff` The hydration CorneoDiff
- `Time` Time in hours since product application
- `Product` Which product was used

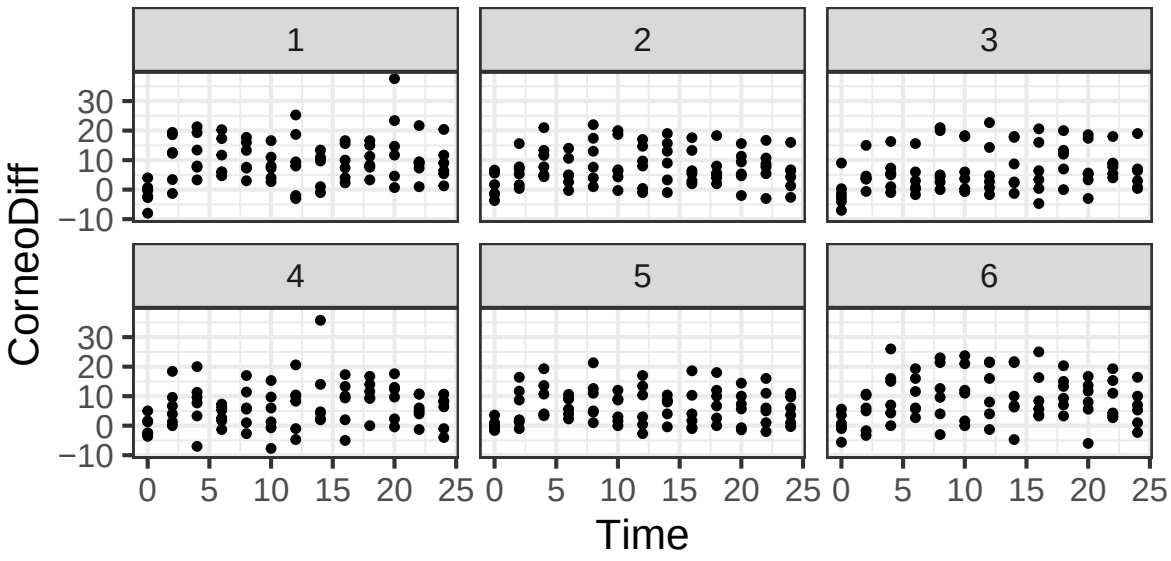
The data file can be accessed online at:

http://sldr.netlify.com/data/hydrationData.csv

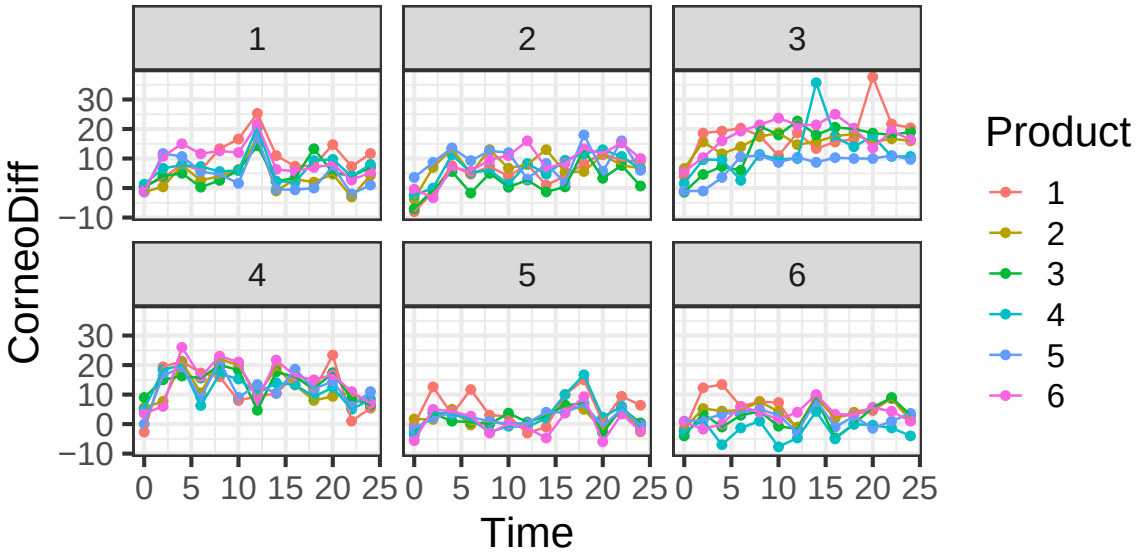
## 15.2 Data Exploration

We would like to model the hydration, `CorneoDiff`, over time and as a function of product.

```
gf_point(CorneoDiff ~ Time | Product, data=hyd.sm) |>
  gf_lims(x=c(0,24))
```




```
gf_point(CorneoDiff ~ Time | Subjects, color=~Product, data=hyd.sm) |>
  gf_line(CorneoDiff ~ Time | Subjects, color=~Product, data=hyd.sm) |>
  gf_lims(x=c(0,24))
```



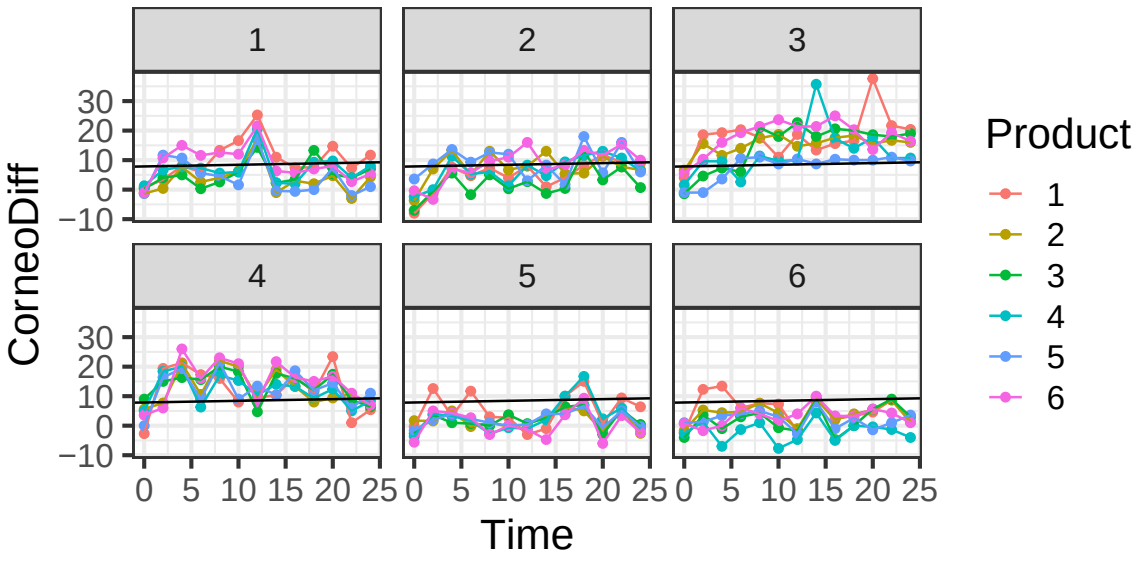
## 15.3 Linear Regression

We could try just fitting a linear regression. What do you expect?

```
lm1 <- lm(CorneoDiff ~ Time + Product, data=hyd.sm)
summary(lm1)
```

```
Call:
lm(formula = CorneoDiff ~ Time + Product, data = hyd.sm)

Residuals:
    Min      1Q  Median      3Q     Max
-17.791  -5.349  -1.164   4.415  29.691

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  9.79072    0.87716  11.162  < 2e-16 ***
Time        -0.05726    0.02720  -2.105  0.03577 *
Product2    -1.90238    1.10665  -1.719  0.08623 .
Product3    -2.80079    1.10665  -2.531  0.01169 *
Product4    -2.98016    1.10665  -2.693  0.00732 **
Product5    -3.02659    1.10665  -2.735  0.00646 **
Product6    -0.43929    1.10665  -0.397  0.69157
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.172 on 497 degrees of freedom
Multiple R-squared:  0.03701,   Adjusted R-squared:  0.02538
F-statistic: 3.183 on 6 and 497 DF,  p-value: 0.004487
```

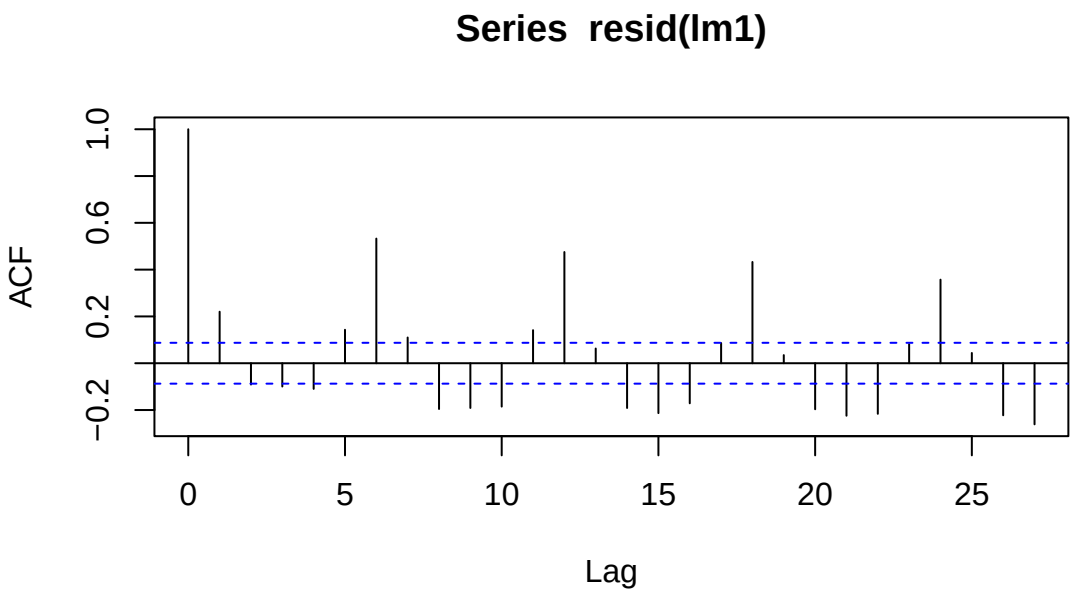
```
gf_point(CorneoDiff ~ Time | Subjects, color=~Product, data=hyd.sm) |>
  gf_line(CorneoDiff ~ Time | Subjects, color=~Product, data=hyd.sm) |>
  gf_lims(x=c(0,24))  |>
  gf_abline(intercept=7.86, slope=0.05608)
```



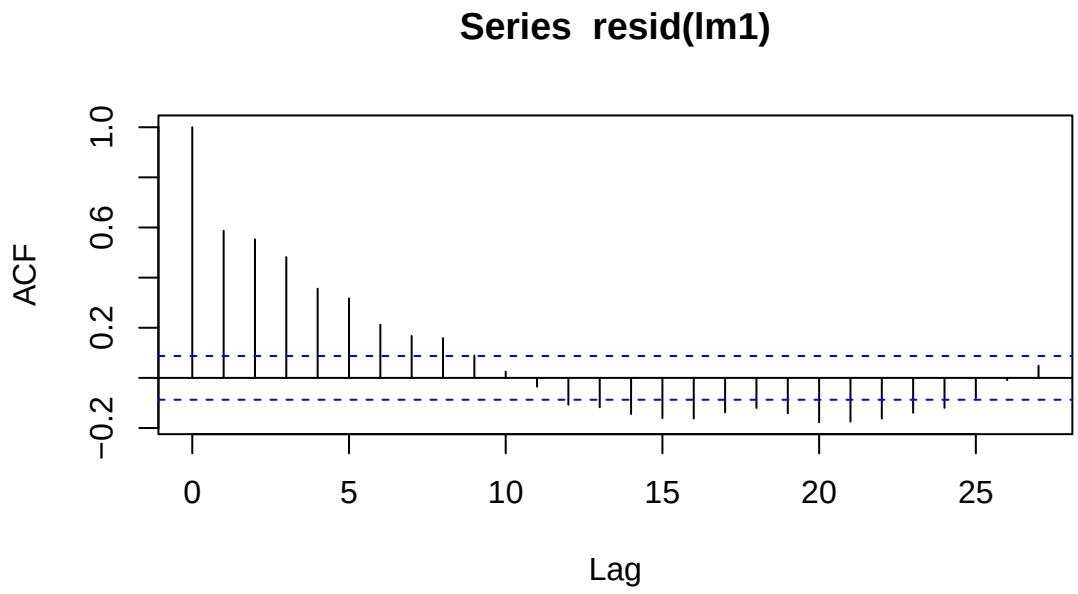

## 15.4 Model Assessment

For the linear regression:

```
acf(resid(lm1))
```




```
hyd.sm <- arrange(hyd.sm,Time, Subjects)
lm1 <- lm(CorneoDiff ~ Time + Product, data=hyd.sm)
acf(resid(lm1))
```




*As we expected, things do not look good…*

## 15.5 Linear Regression

We tried a linear regression and encountered two problems:

- The residuals are not independent. There seems to be correlation over time within subjects.
- We can't account for inter-person differences unless we include person as a predictor, but we don't want to do that, because if we do we can not make predictions from the fitted model without specifying which of these exact people we want to predict for. That's not ideal - we want predictions for all people, or at least averaged over all people.

## 15.6 Generalized Estimating Equations (GEEs)

A potential solution we will invesitgate today is to use a generalized estimating equation (GEE) instead of a GLM. GEEs:

- Are a "PA" model:
- Work by changing…

What residual correlation structures can be accomodated in this framework?

- Independence (`corstr=`independence'')
- Exchangeable = Block Diagonal (`corstr=`exchangeable'')
- AR1 (first-order auto-regressive) (`corstr=`ar1'')
- Unstructured (CAUTION!) (`corstr=`unstructured'')

### 15.6.1 Fitting GEEs with different correlation structures

```
library(geepack)
```

```
Attaching package: 'geepack'

The following object is masked from 'package:MuMIn':

    QIC
```

```
hyd <- arrange(hyd, Subjects, Time)
lm1 <- lm(CorneoDiff ~ Time + Product, data=hyd)
summary(lm1)
```

```
Call:
lm(formula = CorneoDiff ~ Time + Product, data = hyd)

Residuals:
     Min       1Q   Median       3Q      Max
-22.5776  -5.3165  -0.0275   5.1061  28.0872

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 11.42313    0.48032  23.782  < 2e-16 ***
Time        -0.09552    0.01489  -6.413 1.85e-10 ***
Product2    -2.08655    0.60598  -3.443 0.000589 ***
Product3    -2.49940    0.60598  -4.125 3.90e-05 ***
Product4    -1.96107    0.60598  -3.236 0.001235 **
Product5    -2.37238    0.60598  -3.915 9.41e-05 ***
Product6    -0.50833    0.60598  -0.839 0.401669
```

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

Residual standard error: 7.17 on 1673 degrees of freedom
Multiple R-squared:  0.04082,   Adjusted R-squared:  0.03738
F-statistic: 11.87 on 6 and 1673 DF,  p-value: 4.489e-13
```

```
gee.ind <- geeglm(CorneoDiff ~ Time + Product, data=hyd,
                  id = Subjects, corstr='independence')
summary(gee.ind)
```

```
Call:
geeglm(formula = CorneoDiff ~ Time + Product, data = hyd, id = Subjects,
    corstr = "independence")

 Coefficients:
            Estimate  Std.err   Wald Pr(>|W|)
(Intercept) 11.42313  1.20102 90.462  < 2e-16 ***
Time        -0.09552  0.02342 16.638 4.52e-05 ***
Product2    -2.08655  0.98810  4.459   0.0347 *
Product3    -2.49940  1.03782  5.800   0.0160 *
Product4    -1.96107  1.02207  3.681   0.0550 .
Product5    -2.37238  1.32981  3.183   0.0744 .
Product6    -0.50833  0.95744  0.282   0.5955
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation structure = independence
Estimated Scale Parameters:

            Estimate Std.err
(Intercept)     51.2   4.017
Number of clusters:   20  Maximum cluster size: 84
```

```
gee.ar1 <- geeglm(CorneoDiff ~ Time + Product, data=hyd,
                  id = Subjects, corstr='ar1')
summary(gee.ar1)
```

```
Call:
geeglm(formula = CorneoDiff ~ Time + Product, data = hyd, id = Subjects,
    corstr = "ar1")

 Coefficients:
            Estimate Std.err  Wald Pr(>|W|)
(Intercept)  11.4005  1.3968 66.61  3.3e-16 ***
Time         -0.2045  0.0322 40.27  2.2e-10 ***
Product2     -2.2329  0.9941  5.04   0.0247 *
Product3     -2.7965  1.0486  7.11   0.0077 **
Product4     -2.4137  1.0402  5.38   0.0203 *
Product5     -2.9857  1.3513  4.88   0.0271 *
Product6     -1.2880  0.9735  1.75   0.1858
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation structure = ar1
Estimated Scale Parameters:

            Estimate Std.err
(Intercept)     56.9    4.48
  Link = identity

Estimated Correlation Parameters:
      Estimate Std.err
alpha    0.951 0.00905
Number of clusters:   20  Maximum cluster size: 84
```

```r
gee.exch <- geeglm(CorneoDiff ~ Time + Product, data=hyd,
                   id = Subjects, corstr='exchangeable')
```

What is **the same** (or similar) and what is very **different** between the models?

### 15.6.2 Comparing different correlation structures

We can use a specific variance of QIC, $QIC_R$, to compare models with different correlation structures:

```r
library(MuMIn)
# QIC(gee.ind, gee.exch, gee.ar1, typeR=TRUE)
```

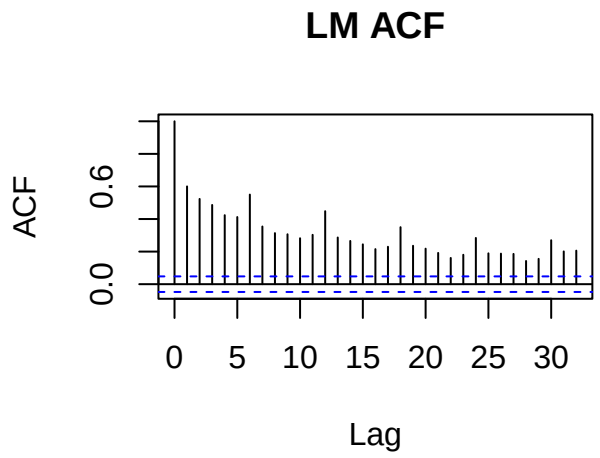
How can we interpret this result?

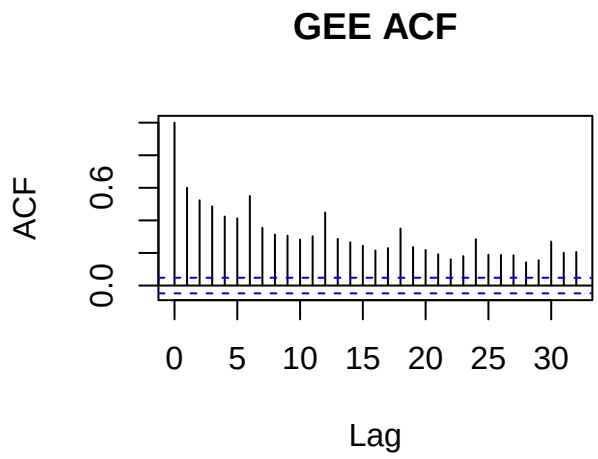
## 15.7 GEE model assessment

Model assessment for a GEE is mostly the same as for the corresponding linear regression or GLM (Poisson, Logistic, etc.)

We were using GEEs to try to correct for issues with non-independent residuals. How does the residual plot change for a GEE relative to the corresponding (g)lm? *Does* it change? *Should* it?

```
acf(resid(lm1), main='LM ACF')
acf(resid(gee.ind), main='GEE ACF')
```






**What is going on here?**

## 15.8 Model Selection - Which variables?

We can use another variant of the QIC to do model selection to determine which variables are important to retain in a GEE model.

```
gee.ind <- update(gee.ind, na.action='na.fail')
dredge(gee.ind, rank='QIC', typeR=FALSE)
```

```
Fixed term is "(Intercept)"

Global model call: geeglm(formula = CorneoDiff ~ Time + Product, data = hyd, na.action = "na
    id = Subjects, corstr = "independence")
---
Model selection table
  (Intrc) Prdct    Time qLik  QIC delta weight
1    8.46               -840 1729  0.00  0.954
3    9.85       -0.0955 -840 1735  6.06  0.046
2   10.03     +         -840 1756 26.88  0.000
4   11.42     + -0.0955 -840 1763 33.71  0.000
Models ranked by QIC(x, typeR = FALSE)
```

How would you interpret these results and present them to the cosmetics company that collected the data?

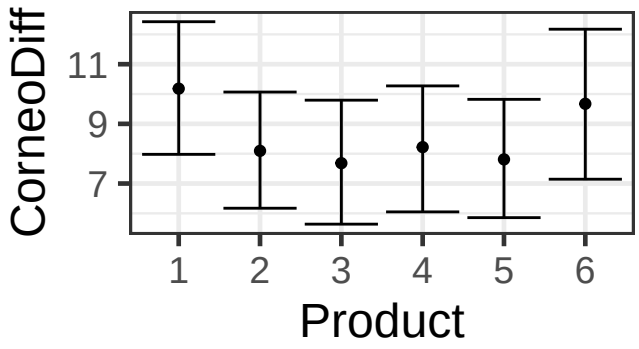
## 15.9 Prediction Plots

As for models we studied previously, we can make prediction plots to visualize the relationships a model specifies between the predictor and response variables.

However, we can not use `predict()` to get model predictions **with standard errors** from a GEE.

`pred_plot()` works, though; for example:

```
s245::pred_plot(gee.ind, 'Product') |>
  gf_labs(y = 'CorneoDiff')
```




Once again we're grateful for the parametric bootstrap! (This time, `pred_plot()` is silently doing the work for us.)

# 16 Correlation Structures

This section is an appendix to the GEE chapter, to illustrate and discuss the different correlation structures that we can specify when fitting a GEE.

### 16.0.1 Variance/Covariance or Correlation?

Statisticians often talk about a model's **variance-covariance** matrix (which gives residual variance on the diagonal and covariance between residuals in its off-diagonal elements). Here we will consider **correlation** matrices instead. This will give us a simplified way of looking at similar ideas. In a correlation matrix, the diagonal entries will be all 1s (the correlation of something with itself is 1), and off-diagonal elements will show correlation between residuals.

### 16.0.2 Example Case

Let's consider an example for a dataset with 9 observations; three observations for each of three individuals. *(Side note: The groupings causing non-independence in the residuals don't have to be "individuals" – it could be some other relationship, like being from the same school or town or country or ethnic group – any single categorical variable that defines the groups of interest and induces non-independence withing the groups, could work. Here we are just calling them "individuals" for convenience, and because when time-series data are collected on multiple individuals, this kind of model often works well.)*

Each of the correlation matrices below has one row and one column for each observation.

The *diagonal* entries will all, always, be 1.

The *off-diagonal* entries indicate how the different residuals depend on each other – in other words, they describe mathematically exactly *how* each residual is correlated with the others.

We will use the letter $\rho$ for correlations.

### 16.0.3 Independence

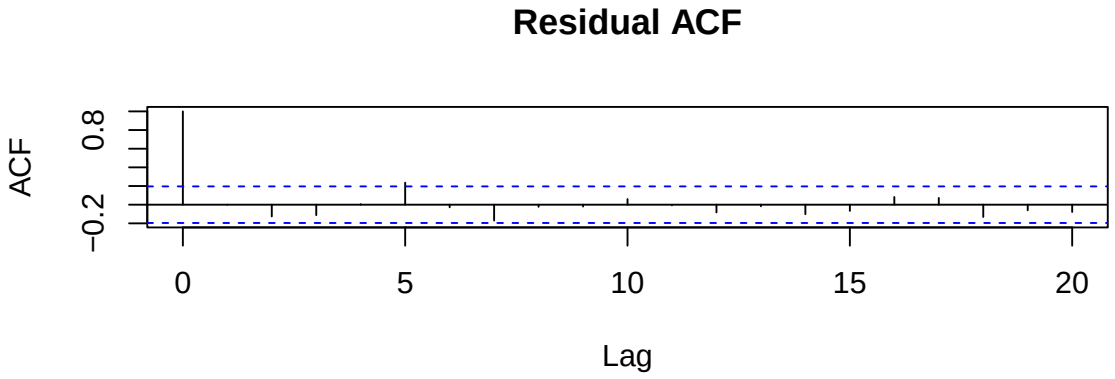
If the residuals are all independent of each other, then all the correlations between them will be zero:

$$\begin{vmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{vmatrix}$$

This corresponds to all of the models we have considered previously in class (which all have a condition that residuals should be independent).

### 16.0.4 ACF Example

The residual ACF from a model fitted to data that perfectly embody this structure with 5 observations per individual might look like:




### 16.0.5 Exchangeable = Block Diagonal

In an exchangeable or block diagonal structure, all residuals for one individual are equally correlated (correlation = $\rho$). All residuals from different individuals are independent (correlation = 1).

$$\begin{vmatrix} 1 & \rho & \rho & 0 & 0 & 0 & 0 & 0 & 0 \\ \rho & 1 & \rho & 0 & 0 & 0 & 0 & 0 & 0 \\ \rho & \rho & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \rho & \rho & 0 & 0 & 0 \\ 0 & 0 & 0 & \rho & 1 & \rho & 0 & 0 & 0 \\ 0 & 0 & 0 & \rho & \rho & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \rho & \rho \\ 0 & 0 & 0 & 0 & 0 & 0 & \rho & 1 & \rho \\ 0 & 0 & 0 & 0 & 0 & 0 & \rho & \rho & 1 \end{vmatrix}$$

#### 16.0.5.1 ACF Example

The residual ACF from a model fitted to data that perfectly embody this structure would show non-independence in the residual ACF, out to about the number of lags that there are observations per individual, but the autocorrelation coefficients would not decline at exactly the rate predicted by an AR(1) process (see below).

### 16.0.6 AR1 (first-order autoregressive process)

In this structure, we again assume that residuals are uncorrelated (independent) between individuals.

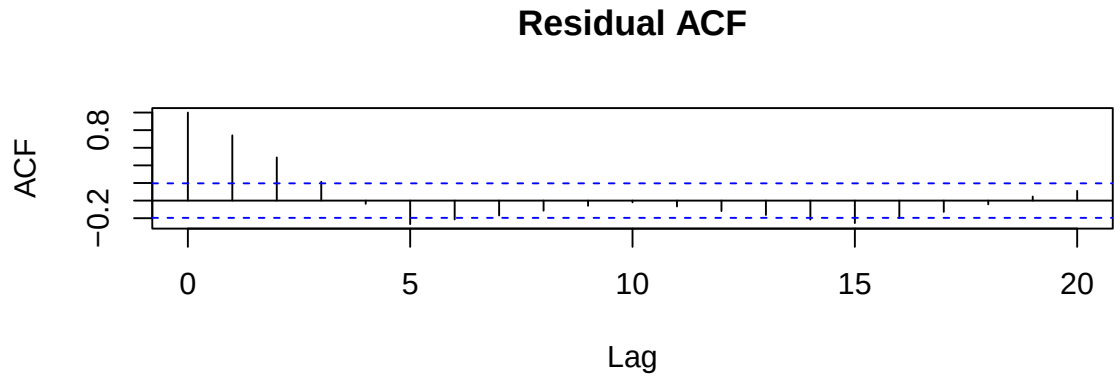
Within an individual, we assume measurements that are closer to one another are more correlated. More precisely, we say that observations 1 lag apart have correlation $\rho$; observations 2 lags apart have correlation $\rho^2$; three lags apart $\rho^3$, and so on.

To illustrate this, I show below a correlation structure for a model with 10 observations: 5 observations from each of 2 individuals.

$$\begin{vmatrix} 1 & \rho & \rho^2 & \rho^3 & \rho^4 & 0 & 0 & 0 & 0 & 0 \\ \rho & 1 & \rho & \rho^2 & \rho^3 & 0 & 0 & 0 & 0 & 0 \\ \rho^2 & \rho & 1 & \rho & \rho^2 & 0 & 0 & 0 & 0 & 0 \\ \rho^3 & \rho^2 & \rho & 1 & \rho & 0 & 0 & 0 & 0 & 0 \\ \rho^4 & \rho^3 & \rho^2 & \rho & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & \rho & \rho^2 & \rho^3 & \rho^4 \\ 0 & 0 & 0 & 0 & 0 & \rho & 1 & \rho & \rho^2 & \rho^3 \\ 0 & 0 & 0 & 0 & 0 & \rho^2 & \rho & 1 & \rho & \rho^2 \\ 0 & 0 & 0 & 0 & 0 & \rho^3 & \rho^2 & \rho & 1 & \rho \\ 0 & 0 & 0 & 0 & 0 & \rho^4 & \rho^3 & \rho^2 & \rho & 1 \end{vmatrix}$$

#### 16.0.6.1 ACF Example

The residual ACF from a model fitted to data that perfectly embody this structure with $\rho = 0.99$ and 5 observations per individual, might look like:




### 16.0.7 Unstructured

This one is not of practical use because it is so hard to estimate.

$$\begin{vmatrix} 1 & \rho_{1,2} & \rho_{1,3} & 0 & 0 & 0 & 0 & 0 & 0 \\ \rho_{1,2} & 1 & \rho_{2,3} & 0 & 0 & 0 & 0 & 0 & 0 \\ \rho_{1,3} & \rho_{2,3} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \rho_{1,2} & \rho_{1,3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \rho_{1,2} & 1 & \rho_{2,3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \rho_{1,3} & \rho_{2,3} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \rho_{1,2} & \rho_{1,3} \\ 0 & 0 & 0 & 0 & 0 & 0 & \rho_{1,2} & 1 & \rho_{2,3} \\ 0 & 0 & 0 & 0 & 0 & 0 & \rho_{1,3} & \rho_{2,3} & 1 \end{vmatrix}$$

# 17 Other Model Selection Approaches

The way we have been doing model selection thus far is *definitely* not the only way. What other options are out there? Many - let's consider a few.

This section also includes some miscellaneous R notes (making summary tables, and sources of inspiration for cool figures).

### 17.0.1 Rationale

Until now, we have focused on using information criteria for model selection, in order to get very familiar with one coherent framework for choosing variables across model types. But:

- In some fields, using hypothesis tests for variable selection is preferred
- For datasets that are large and/or models that are complex, `dredge()` can be a challenge (taking a very long time to run and perhaps timing out on the server)
- Using hypothesis tests for selection is quite common, so we should know how it's done!

### 17.0.2 Hypotheses

Basically, for each (fixed effect) variable in a model, we'd like to test:

$$H_0: \text{all } \beta\text{s for this variable are 0; it's not a good predictor}$$

$$H_1: \text{ at least one } \beta \text{ is non-zero; it's a good predictor}$$

We want to test these hypotheses *given that all the other predictors in the current full model are included*. Because of this condition, and because there are *multiple* $\beta$s for categorical predictors with more than 2 categories, we can **not** generally just use the p-values from the model `summary()` output.

Instead, we use `Anova()` from the package `car`. `lm()` example:

```
iris_mod <- lm(Petal.Length ~ Petal.Width + Species + Sepal.Length, data = iris)
summary(iris_mod)
```

```
Call:
lm(formula = Petal.Length ~ Petal.Width + Species + Sepal.Length,
    data = iris)

Residuals:
     Min       1Q   Median       3Q      Max
-0.76508 -0.15779  0.01102  0.13378  0.66548

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       -1.45957    0.22387  -6.520 1.09e-09 ***
Petal.Width        0.50641    0.11528   4.393 2.15e-05 ***
Speciesversicolor  1.73146    0.12762  13.567  < 2e-16 ***
Speciesvirginica   2.30468    0.19839  11.617  < 2e-16 ***
Sepal.Length       0.55873    0.04583  12.191  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2664 on 145 degrees of freedom
Multiple R-squared:  0.9778,    Adjusted R-squared:  0.9772
F-statistic:  1600 on 4 and 145 DF,  p-value: < 2.2e-16
```

```r
library(car)
Anova(iris_mod)
```

```
Anova Table (Type II tests)

Response: Petal.Length
              Sum Sq  Df F value    Pr(>F)
Petal.Width   1.3691   1  19.296 2.147e-05 ***
Species      13.6137   2  95.936 < 2.2e-16 ***
Sepal.Length 10.5454   1 148.627 < 2.2e-16 ***
Residuals    10.2880 145
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notice that `Anova()` reports *one* p-value for each predictor (excellent!). If the p-value is small, that gives evidence against $H_0$, and we'd conclude we should keep the predictor in the model. Many people use $\alpha = 0.05$ as the "dividing line" between "small" and "large" p-values and

thus "statistically significant" and "non-significant" test results, but remember the p-value is a probability - there's no magical difference between 0.049 and 0.051.

*Warning: be careful with your capitalization! The R function* `anova()` *does someting kind of similar to* `Anova()` *but* **NOT** *the same and should be avoided – it does sequential rather than marginal tests.*

## 17.1 Backward selection

How do we use p-value-based selection to arrive at a best model? There are many options and much controversy about different approaches; here I'll suggest one. None of these methods are guaranteed to arrive at a model that is theoretically "best" in some specific way, but they do give a framework to guide decision-making and are computationally quick. The premise is that we'd like a simple algorithm to implement, and we will begin with a full model including all the predictors that we think *should* or *could* reasonably be important (not just throwing in everything possible).

### 17.1.1 Algorithm

- Obtain p-values for all predictors in full model
- Remove the predictor with the largest p-value that you judge to be "not small" or "not significant"
- Re-compute p-values for the new, smaller model
- Repeat until all p-values are "significant"

### 17.1.2 Example

Let's consider a logistic regression to predict whether a person in substance-abuse treatment is homeless.

```
home_mod0 <- glm(homeless ~ sex + substance + i1 + cesd +
                   racegrp + age,
                 data = HELPrct, family = binomial(link = 'logit'))
Anova(home_mod0)
```

```
Analysis of Deviance Table (Type II tests)

Response: homeless
          LR Chisq Df Pr(>Chisq)
sex         3.3837  1   0.065846 .
```

```
substance   2.2326  2   0.327483
i1         10.6261  1   0.001115 **
cesd        1.1751  1   0.278351
racegrp     3.1811  3   0.364541
age         0.4392  1   0.507491
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Removing `age`:

```
home_mod <- update(home_mod0, .~. - age)
Anova(home_mod)
```

```
Analysis of Deviance Table (Type II tests)

Response: homeless
          LR Chisq Df Pr(>Chisq)
sex         3.2184  1   0.072817 .
substance   2.6874  2   0.260877
i1         11.0836  1   0.000871 ***
cesd        1.1300  1   0.287773
racegrp     3.1561  3   0.368180
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Removing `racegrp`

```
home_mod <- update(home_mod, .~. - racegrp)
Anova(home_mod)
```

```
Analysis of Deviance Table (Type II tests)

Response: homeless
          LR Chisq Df Pr(>Chisq)
sex         3.5883  1  0.0581886 .
substance   3.6342  2  0.1624971
i1         11.0174  1  0.0009026 ***
cesd        1.6055  1  0.2051242
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Remove `cesd` (a score indicating depression level)

```
home_mod <- update(home_mod, .~. - cesd)
Anova(home_mod)
```

```
Analysis of Deviance Table (Type II tests)

Response: homeless
          LR Chisq Df Pr(>Chisq)
sex         2.7855  1   0.095118 .
substance   3.6743  2   0.159272
i1         13.2405  1   0.000274 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Remove `substance`

```
home_mod <- update(home_mod, .~. - substance)
Anova(home_mod)
```

```
Analysis of Deviance Table (Type II tests)

Response: homeless
    LR Chisq Df Pr(>Chisq)
sex   2.9647  1     0.0851 .
i1   25.7861  1  3.814e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Remove `sex`

```
home_mod <- update(home_mod, .~. - sex)
Anova(home_mod)
```

```
Analysis of Deviance Table (Type II tests)

Response: homeless
   LR Chisq Df Pr(>Chisq)
i1   27.187  1  1.847e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 17.1.3 Can't this be automated?

Strangely...functions are not widely available.

### 17.1.4 Stepwise IC-based selection

Another option may be to use **backward stepwise selection** (same algorithm as above), but using AIC or BIC as the criterion at each stage instead of p-values. If the IC value is better (by *any* amount) without a variable, it gets dropped. Variables are dropped one by one until no further IC improvement is possible.

This evaluates many fewer models than `dredge` so should be much faster, but may not find the best of all possible models.

For example, for our model using AIC (*note: this may or may not work for all model types.*):

```
library(MASS)
stepAIC(home_mod0)
```

```
Start:  AIC=606.26
homeless ~ sex + substance + i1 + cesd + racegrp + age

            Df Deviance    AIC
- racegrp    3   589.44 603.44
- substance  2   588.49 604.49
- age        1   586.70 604.70
- cesd       1   587.43 605.43
<none>           586.26 606.26
- sex        1   589.64 607.64
- i1         1   596.88 614.88

Step:  AIC=603.44
homeless ~ sex + substance + i1 + cesd + age

            Df Deviance    AIC
- age        1   589.85 601.85
- substance  2   592.45 602.45
- cesd       1   591.10 603.10
<none>           589.44 603.44
- sex        1   593.20 605.20
- i1         1   600.00 612.00
```

```
Step:  AIC=601.85
homeless ~ sex + substance + i1 + cesd

            Df Deviance    AIC
- cesd       1   591.46 601.46
- substance  2   593.49 601.49
<none>           589.85 601.85
- sex        1   593.44 603.44
- i1         1   600.87 610.87

Step:  AIC=601.46
homeless ~ sex + substance + i1

            Df Deviance    AIC
- substance  2   595.13 601.13
<none>           591.46 601.46
- sex        1   594.24 602.24
- i1         1   604.70 612.70

Step:  AIC=601.13
homeless ~ sex + i1

       Df Deviance    AIC
<none>      595.13 601.13
- sex   1   598.10 602.10
- i1    1   620.92 624.92



Call:  glm(formula = homeless ~ sex + i1, family = binomial(link = "logit"),
    data = HELPrct)

Coefficients:
(Intercept)      sexmale           i1
    0.92969     -0.39976     -0.02657

Degrees of Freedom: 452 Total (i.e. Null);  450 Residual
Null Deviance:      625.3
Residual Deviance: 595.1    AIC: 601.1
```

Note that we might want to still remove *one more* variable than `stepAIC()` does! Above, you see that if you were to remove `age`, the AIC would only go up by about 1 unit. So according to our $\Delta AIC \sim 3$ threshold, we would take `age` out too.

Using BIC instead, we need to specify the input `k = log(nrow(data))` (the BIC penalty multiplier):

```
stepAIC(home_mod0, k = log10(nrow(HELPrct)))
```

```
Start:  AIC=612.82
homeless ~ sex + substance + i1 + cesd + racegrp + age

            Df Deviance    AIC
- racegrp    3   589.44 608.03
- substance  2   588.49 609.74
- age        1   586.70 610.60
- cesd       1   587.43 611.34
<none>           586.26 612.82
- sex        1   589.64 613.55
- i1         1   596.88 620.79

Step:  AIC=608.03
homeless ~ sex + substance + i1 + cesd + age

            Df Deviance    AIC
- substance  2   592.45 605.73
- age        1   589.85 605.79
- cesd       1   591.10 607.04
<none>           589.44 608.03
- sex        1   593.20 609.14
- i1         1   600.00 615.94

Step:  AIC=605.73
homeless ~ sex + i1 + cesd + age

       Df Deviance    AIC
- age   1   593.49 604.11
- cesd  1   594.20 604.82
<none>      592.45 605.73
- sex   1   596.48 607.11
- i1    1   611.94 622.57

Step:  AIC=604.11
homeless ~ sex + i1 + cesd

       Df Deviance    AIC
```

```
- cesd  1   595.13 603.10
<none>      593.49 604.11
- sex   1   597.25 605.22
- i1    1   615.70 623.66

Step:  AIC=603.1
homeless ~ sex + i1

       Df Deviance    AIC
<none>      595.13 603.10
- sex   1   598.10 603.41
- i1    1   620.92 626.23


Call:  glm(formula = homeless ~ sex + i1, family = binomial(link = "logit"),
    data = HELPrct)

Coefficients:
(Intercept)      sexmale           i1
    0.92969     -0.39976     -0.02657

Degrees of Freedom: 452 Total (i.e. Null);  450 Residual
Null Deviance:      625.3
Residual Deviance: 595.1    AIC: 601.1
```

To get less verbose output, set `trace = 0` – but then you won't know whether it would make sense to perhaps remove additional variables...

```
stepAIC(home_mod0, k = log10(nrow(HELPrct)), trace = 0)
```

```
Call:  glm(formula = homeless ~ sex + i1, family = binomial(link = "logit"),
    data = HELPrct)

Coefficients:
(Intercept)      sexmale           i1
    0.92969     -0.39976     -0.02657

Degrees of Freedom: 452 Total (i.e. Null);  450 Residual
Null Deviance:      625.3
Residual Deviance: 595.1    AIC: 601.1
```

## 17.2 Summary tables

You may want to compute and display summary tables for your projects. Here are a few examples of how to do it.

### 17.2.1 Mean (or sd, median, IQR, etc.) by groups

Compute the mean and sd (could use any other summary stats you want, though) for several quantitative variables, by groups.

Example: find mean and sd of iris flower `Petal.Length` and `Petal.Width` by `Species` and display results in a pretty table. The dataset is called `iris`.

Make a little one-row table for each variable being summarized, then stick them together.

```
library(knitr)

length_stats <- iris |>
  df_stats(Petal.Length ~ Species, mean, sd, long_names = FALSE) |>
  mutate(variable = 'Petal Length')

width_stats <- iris |>
  df_stats(Petal.Width ~ Species, mean, sd, long_names = FALSE) |>
  mutate(variable = 'Petal Width')

my_table <- bind_rows(length_stats, width_stats)

kable(my_table)
```

| response | Species | mean | sd | variable |
|---|---|---|---|---|
| Petal.Length | setosa | 1.462 | 0.1736640 | Petal Length |
| Petal.Length | versicolor | 4.260 | 0.4699110 | Petal Length |
| Petal.Length | virginica | 5.552 | 0.5518947 | Petal Length |
| Petal.Width | setosa | 0.246 | 0.1053856 | Petal Width |
| Petal.Width | versicolor | 1.326 | 0.1977527 | Petal Width |
| Petal.Width | virginica | 2.026 | 0.2746501 | Petal Width |

What if we want to round all table entries to 2 digits after the decimal?

```
kable(my_table, digits = 2)
```

| response | Species | mean | sd | variable |
|---|---|---|---|---|
| Petal.Length | setosa | 1.46 | 0.17 | Petal Length |
| Petal.Length | versicolor | 4.26 | 0.47 | Petal Length |
| Petal.Length | virginica | 5.55 | 0.55 | Petal Length |
| Petal.Width | setosa | 0.25 | 0.11 | Petal Width |
| Petal.Width | versicolor | 1.33 | 0.20 | Petal Width |
| Petal.Width | virginica | 2.03 | 0.27 | Petal Width |

What if we want the column order to be Variable, Species, mean, sd, and sort by Species and then Variable?

```
my_table <- my_table |>
  dplyr::select(variable, Species, mean, sd) |>
  arrange(Species, variable)
kable(my_table, digits = 2)
```

| variable | Species | mean | sd |
|---|---|---|---|
| Petal Length | setosa | 1.46 | 0.17 |
| Petal Width | setosa | 0.25 | 0.11 |
| Petal Length | versicolor | 4.26 | 0.47 |
| Petal Width | versicolor | 1.33 | 0.20 |
| Petal Length | virginica | 5.55 | 0.55 |
| Petal Width | virginica | 2.03 | 0.27 |

What if we actually want a column for mean length, sd length, etc. and one row per species?

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v forcats   1.0.0     v stringr   1.5.0
v lubridate 1.9.2     v tibble    3.2.1
v purrr     1.0.1     v tidyr     1.3.0
v readr     2.1.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x mosaic::count() masks dplyr::count()
x purrr::cross()  masks mosaic::cross()
x mosaic::do()    masks dplyr::do()
x tidyr::expand() masks Matrix::expand()
x dplyr::filter() masks stats::filter()
```

```
x dplyr::lag()    masks stats::lag()
x tidyr::pack()   masks Matrix::pack()
x car::recode()   masks dplyr::recode()
x MASS::select()  masks dplyr::select()
x purrr::some()   masks car::some()
x mosaic::stat()  masks ggplot2::stat()
x mosaic::tally() masks dplyr::tally()
x tidyr::unpack() masks Matrix::unpack()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```
my_table2 <- my_table |>
  pivot_wider(names_from = variable,
              values_from = c("mean", "sd"),
              names_sep = ' ')
kable(my_table2, digits = 2, align = 'c')
```

| Species | mean Petal Length | mean Petal Width | sd Petal Length | sd Petal Width |
|---|---|---|---|---|
| setosa | 1.46 | 0.25 | 0.17 | 0.11 |
| versicolor | 4.26 | 1.33 | 0.47 | 0.20 |
| virginica | 5.55 | 2.03 | 0.55 | 0.27 |

### 17.2.2 Proportions in categories by groups

You may also want to make a table of proportion observations in each category by groups, potentially for many variables.

For just one variable, we can use tally:

```
tally(~substance | sex, data = HELPrct, format = 'prop') |>
  kable(caption = 'Proportion using each substance', digits = 2)
```

Table 17.5: Proportion using each substance

| | female | male |
|---|---|---|
| alcohol | 0.34 | 0.41 |
| cocaine | 0.38 | 0.32 |
| heroin | 0.28 | 0.27 |

For many variables we can use a loop. For example, we might want to know the proportion homeless and housed **and** proportion using each substance, both by sex, from the `HELPrct` dataset. Above we were using the function `knitr::kable()` to make tables, but we can use `pander::pander()` too:

```
# select only variables needed for the table
# make the first variable the groups one
cat_data <- HELPrct |> dplyr::select(sex, substance, homeless)

for (i in c(2:ncol(cat_data))){
tally(~cat_data[,i] | cat_data[,1], format = 'prop') |>
    pander::pander(caption = paste('Proportion in each ',
                                   names(cat_data)[i]))
  # can rename variables in cat_data if you want better captions
  }
```

## 17.3 Figures

We've made a lot of figures in this class, and almost all have been kind of mediocre. To aim for awesome, here are a couple of great references for inspiration, ideas, and best practices:

- *Fundamentals of Data Visualization* by Claus Wilke. https://serialmentor.com/dataviz/
- https://infogram.com/blog/20-best-data-visualizations-of-2018/
- visualizingdata.com blog
  - https://www.visualisingdata.com/2019/08/10-significant-visualisation-developments-january-to-june-2019/
  - https://www.visualisingdata.com/2016/03/little-visualisation-design/

# 18 GAMs: Generalized Additive Models

So far, we have learned ways to model continuous, logical, and count response variables as functions of quantitative and categorical predictors. We started with linear models - where both the response and predictor variables are quantitative and the relationship between them is *linear*. What about nonlinear relationships?

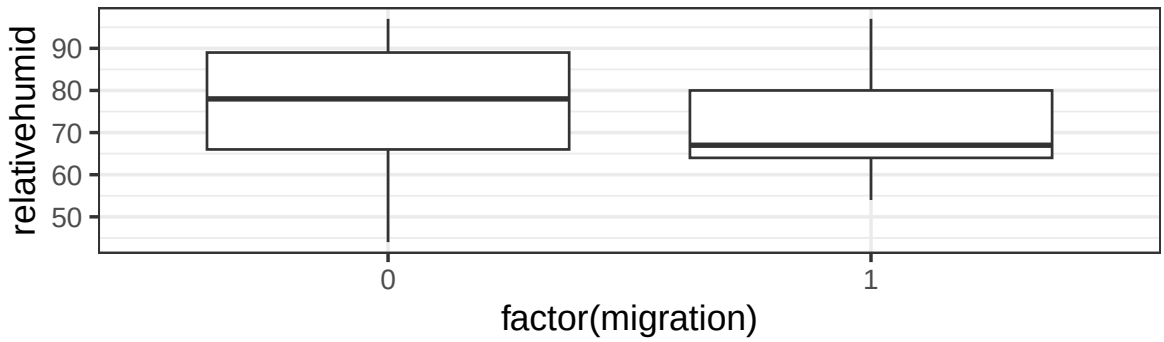
So far, we have considered…

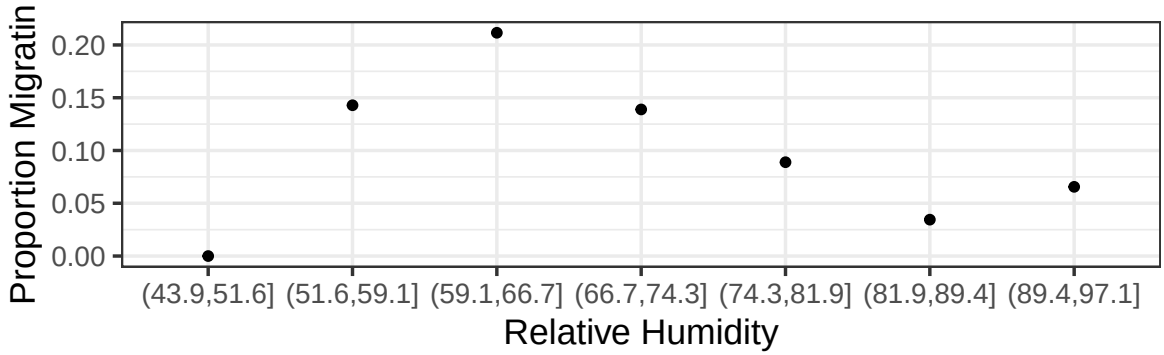
- **Categorical predictor variables**. Making use of indicator variables for (all but one of the) categories, we can model a situation where each value of the predictor variable has a different effect on the response. But…
  - How many categories?
  - What about periodicity?
- **GLMs**. In logistic, Poisson, etc. regression, the action of the link function results in a relationship between the predictors and the response variable that is linear on the scale of the link function ( = scale of the RHS of the equation), but non-linear on the scale of the response variables (LHS). But…
  - Nonlinear, but **monotonic**

## 18.1 Non-linear, non-monotonic relationships

It's not true that all interesting predictor-response relationships are linear or monotonic. One example is in some data on bat migration: how does the probability of bats leaving on their migratory journey depend on air humidity?

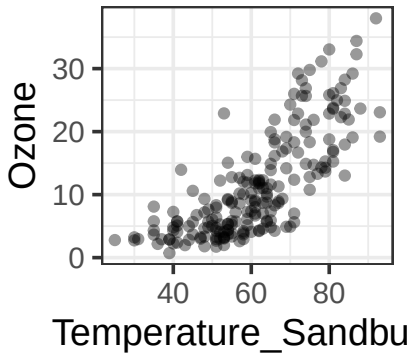
```
bats <- read.csv('https://ndownloader.figshare.com/files/9348010')
```

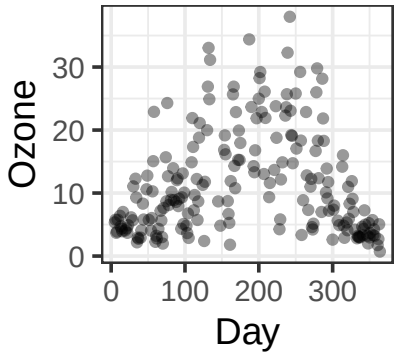


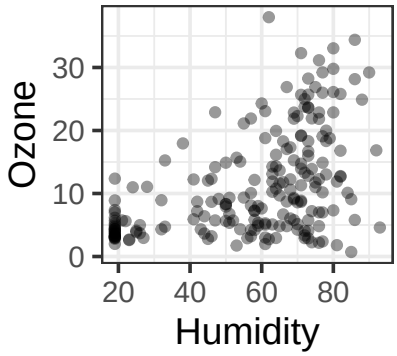




Another dataset (our example for the day) – ozone levels as a function of temperature, day, and humidity:

```
ozone <- read_csv('https://raw.githubusercontent.com/selva86/datasets/master/ozone.csv') |
  mutate(Date = lubridate::mdy(paste(Month, Day_of_month, '2018')),
         Day = lubridate::yday(Date))
gf_point(ozone_reading ~ Temperature_Sandburg,
         data=ozone, alpha=0.4, ylab = 'Ozone')
gf_point(ozone_reading ~ Day,
         data=ozone, alpha=0.4, ylab = 'Ozone')
gf_point(ozone_reading ~ Humidity,
         data=ozone, alpha=0.4, ylab = 'Ozone')
```




## 18.2 Smooth terms

We can fit a model where the relationship between the response and the predictor is a "smooth" – no linearity or monotonicity requirement.

### 18.2.1 Basis functions

- A smooth term is constructed as the sum of several parts, or *basis functions*. Each of these functions has a relatively simple shape, but scaled and added together, they can produce nearly any "wiggly" shape.
- Increasing the dimension of the basis (more functions added together) can allow more wiggliness.
- Goal: allow enough wiggliness to fit the data well, without *overfitting* (smooth goes through every point in the data, or follows "trends" that are spurious)

We will fit smooth models to data using the function `gam()` from the package `mgcv`. It includes many options for basis functions (types of smooths) - see `?mgcv::gam` or [https://rsconnect.calvin.edu:3939/content/28/] for details.

## 18.3 Fitting GAMs

An excellent resource: https://converged.yt/mgcv-workshop/.

### 18.3.1 Choosing model formulation

Which terms should be modelled as smooth terms? Explore the data!

- Pros:
- Cons:

### 18.3.2 Model formula

Let's fit a simple GAM for the ozone data as a function of radiation, temperature and wind. Note the `s()` function for specifying a smooth, which takes as input:

- a variable name (or more than one, for advanced users)
- `k`
- `bs`

How do we choose? For some exploration, see: [https://rsconnect.calvin.edu:3939/content/28/].

We can also fit the model and smooths by different methods and with options:

- `method = 'GCV.Cp'`
- `method = 'REML'`
- `method = 'ML'`
- `select = TRUE` (or `FALSE`)

```r
library(mgcv)
oz.gam <- gam(ozone_reading ~ s(Day, k = 7, bs = 'cc') +
                s(Wind_speed, k = 5, bs = 'tp') +
                s(Temperature_Sandburg,
                  k = 5, bs = 'tp'),
              data = ozone,
              method = 'ML',
              select = TRUE)
summary(oz.gam)
```

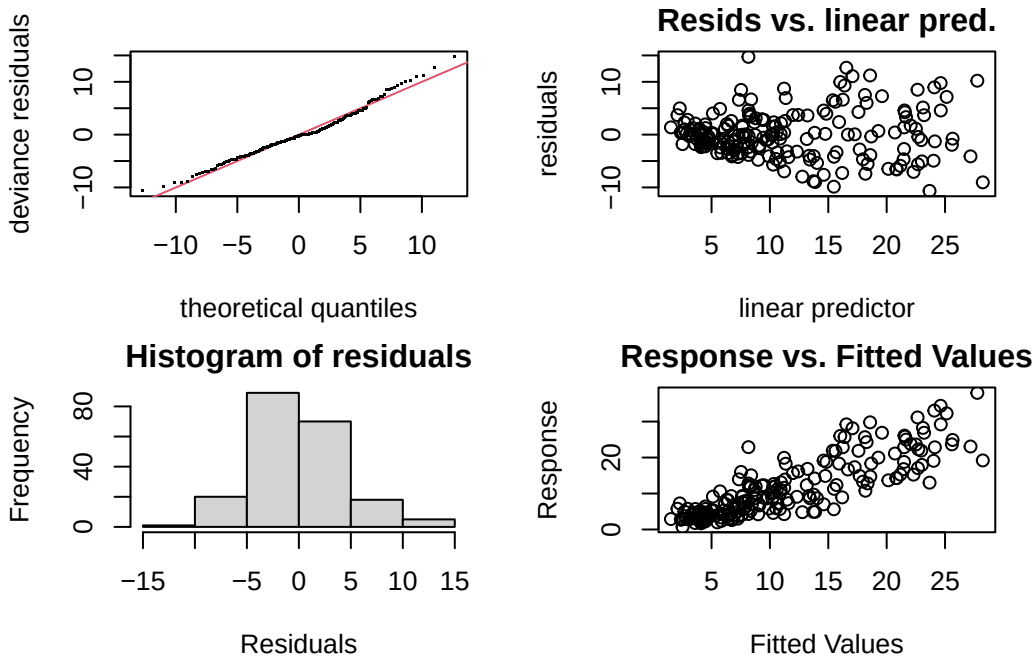
```
Family: gaussian
Link function: identity

Formula:
ozone_reading ~ s(Day, k = 7, bs = "cc") + s(Wind_speed, k = 5,
    bs = "tp") + s(Temperature_Sandburg, k = 5, bs = "tp")

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  11.3740     0.3163   35.96   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
                          edf Ref.df     F p-value
s(Day)                  3.641      5  4.93 2.1e-05 ***
s(Wind_speed)           1.180      4  0.77  0.0696 .
s(Temperature_Sandburg) 3.283      4 45.87 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.697   Deviance explained =   71%
-ML =  604.6  Scale est. = 20.309    n = 203
```

## 18.4 Model Assessment

In addition to what you already know (…which all still holds, except linearity expectation!) `mgcv` has some nice model checking functions built in.

```r
par(mar=c(4,4,2,2))
gam.check(oz.gam)
```

```
Method: ML   Optimizer: outer newton
full convergence after 10 iterations.
Gradient range [-0.000846257,0.001371855]
(score 604.6047 & scale 20.3091).
Hessian positive definite, eigenvalue range [0.0003470333,101.5511].
Model rank =  14 / 14

Basis dimension (k) checking results. Low p-value (k-index<1) may
indicate that k is too low, especially if edf is close to k'.

                          k'  edf k-index p-value
s(Day)                 5.00 3.64    0.86    0.02 *
s(Wind_speed)          4.00 1.18    1.14    0.99
s(Temperature_Sandburg) 4.00 3.28    1.10    0.91
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



### 18.4.1 Concurvity

Like collinearity and multicollinearity, but for smooths...values of 0 indicate no problem, and 1 a huge problem (total lack of identifiability – same information in multiple predictors).

Overall, does the model have problems with concurvity?

```
concurvity(oz.gam, full=TRUE)
```

```
                 para    s(Day) s(Wind_speed) s(Temperature_Sandburg)
worst    2.191674e-20 0.6717847     0.2733692               0.6505342
observed 2.191674e-20 0.1609172     0.1601073               0.6453322
estimate 2.191674e-20 0.2421204     0.2119838               0.5270197
```

Or alternatively, which specific pairs of terms cause problems?

```
concurvity(oz.gam, full=FALSE)
```

```
$worst
                             para       s(Day) s(Wind_speed)
para                 1.000000e+00 6.612020e-32  1.595828e-29
s(Day)               1.243525e-31 1.000000e+00  2.476773e-01
s(Wind_speed)        2.467911e-29 2.476773e-01  1.000000e+00
s(Temperature_Sandburg) 2.081406e-20 6.398944e-01  2.054144e-01
                        s(Temperature_Sandburg)
para                              2.081412e-20
s(Day)                            6.398944e-01
s(Wind_speed)                     2.054144e-01
s(Temperature_Sandburg)           1.000000e+00

$observed
                             para       s(Day) s(Wind_speed)
para                 1.000000e+00 1.710467e-32  1.312129e-30
s(Day)               1.243525e-31 1.000000e+00  8.351370e-02
s(Wind_speed)        2.467911e-29 9.567723e-02  1.000000e+00
s(Temperature_Sandburg) 2.081406e-20 6.230468e-02  7.925646e-02
                        s(Temperature_Sandburg)
para                              7.645727e-23
s(Day)                            6.353482e-01
s(Wind_speed)                     8.030772e-02
s(Temperature_Sandburg)           1.000000e+00
```

```
$estimate
                                para       s(Day) s(Wind_speed)
para                    1.000000e+00 1.411658e-32  9.541351e-31
s(Day)                  1.243525e-31 1.000000e+00  1.685925e-01
s(Wind_speed)           2.467911e-29 7.801012e-02  1.000000e+00
s(Temperature_Sandburg) 2.081406e-20 1.970558e-01  8.331840e-02
                        s(Temperature_Sandburg)
para                              1.703578e-22
s(Day)                            4.902313e-01
s(Wind_speed)                     9.916508e-02
s(Temperature_Sandburg)           1.000000e+00
```

## 18.5 Model Selection

### 18.5.1 Shrinkage and Penalties

With GAMs, in a sense, some model selection is (or can be) done during model fitting - what smooth is best? Or is the relationship a line? A flat line? Using *shrinkage* basis or including `select=TRUE` allows for this.

### 18.5.2 P-value selection

Cautions: **p-values are approximate!** Successfulness of the procedure best when fitting method is: ML (1st choice), REML (2nd choice).

```
anova(oz.gam)
```

Interpretation as usual. Note that `anova()` (not `Anova()`) works here - especially for GAMs, it does *not* do sequential tests; and `Anova()` doesn't handle GAMs.

### 18.5.3 Information criteria

- Conditional/Approximate - bias
- Fitting method:
  - REML-based IC scores can be used to compare models with different *random effects* but **not** different predictors. (IF `select = TRUE` and using a shrinkage basis.)
  - ML-based IC scores can be used to compare models with different fixed effects (regular predictors) and different `family`, but **not** different random effects

```r
library(MuMIn)
oz.ml <- update(oz.gam, method='ML', na.action='na.fail')
head(dredge(oz.ml, rank='AIC'),2)
```

```
Global model call: gam(formula = ozone_reading ~ s(Day, k = 7, bs = "cc") + s(Wind_speed,
    k = 5, bs = "tp") + s(Temperature_Sandburg, k = 5, bs = "tp"),
    data = ozone, na.action = "na.fail", method = "ML", select = TRUE)
---
Model selection table
  (Int) s(Day,7,"cc") s(Tmp_Snd,5,"tp") s(Wnd_spd,5,"tp") df   logLik    AIC
8 11.37             +                 +                 + 11 -589.011 1201.6
4 11.37             +                 +                   10 -591.448 1203.0
  delta weight
8  0.00  0.666
4  1.38  0.334
Models ranked by AIC(x)
```

# References

```
r if (knitr:::is_html_output()) ' '
```