

MURRAY LOGAN'S R COURSE

Nov-Dec 2020
Notes by Sara Kophamel

INDEX

1) Introduction	1
2) Graphics	7
3) Linear modelling LM – Intro (Gaussian)	83
4) Multiple linear regression MLR – Gaussian.....	129
5) Multiple linear regression MLR – Categorical variables	165
6) Multiple linear regression MLR – Poisson	181
7) Generalised linear mixed models GLMM – Gaussian	200
8) Generalised linear mixed models GLMM – Random effects	219
9) Generalised linear mixed models GLMM – Binomial	241
10) Generalised linear mixed models GLMM – Poisson & neg Binomial	252
11) Generalised additive models GAM – Gaussian	273
12) Generalised additive models GAM – Temporal data	293
13) Generalised additive models GAM – Maps	322
14) Bayesian – Intro (Gaussian)	349
15) Bayesian – Linear Regression	390
16) Bayesian – Gaussian Log	414
17) Bayesian – Poisson Count data	447
18) Bayesian – Poisson	478
19) Bayesian – GLMM (Gaussian)	508
20) Bayesian – Repeated Measures	538
21) Regression Trees Intro	578
22) Regression Trees Boosted	600
23) Multivariate Responses	to be included

Logan_Course

Sara Kophamel

30/11/2020

Presentation 1. file:///Users/sara/Documents/PhD/HELP/STATISTICS%20and%20GRAPHS/-%20R%20and%20STATS%20COURSES/2020%20Murray%20Logan%20R%20Course/Logan_Course/docs/pres1.html#1

Presentation 2.4. file:///Users/sara/Documents/PhD/HELP/STATISTICS%20and%20GRAPHS/-%20R%20and%20STATS%20COURSES/2020%20Murray%20Logan%20R%20Course/Logan_Course/docs/pres2.4.html#1

MARKDOWN TRICKS

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

Including Plots

You can also embed plots, for example:

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

Always name your chunks

Journal styles

You can adapt whatever you have written into different journal styles:

The same can be completed in script using the `draft` function from the `rmarkdown` package.

```
rmarkdown::draft(file='./resources/plos.Rmd', template='plos_article', package='rticles', edit=FALSE)
rmarkdown::render('./resources/plos/plos.Rmd')
```

```
rmarkdown::draft(file='./resources/pnas.Rmd', template='pnas_article', package='rticles', edit=FALSE)
rmarkdown::render('./resources/pnas/pnas.Rmd')
```

```
rmarkdown::draft(file='./resources/elsevier.Rmd', template='elsevier_article', package='rticles',
edit=FALSE) rmarkdown::render('./resources/elsevier/elsevier.Rmd')
```

Tweaking with the fonts and design of the Markdown file

Murray has modified the header of the document adding the following details:

```
title: "Data wrangling" author: "Murray Logan" date: "21 December, 2020" output: html_document: code_folding: show collapse: no df_print: paged fig_caption: yes fig_height: 4 fig_width: 4 highlight: textmate theme: spacelab toc: yes toc_float: yes css: ../resources/style.css pdf_document: df_print: default fig_caption: yes fig_height: 4 fig_width: 4 highlight: tango latex_engine: xelatex number_sections: yes toc_depth: 2 word_document: fig_caption: yes fig_height: 4 fig_width: 4 highlight: tango toc: yes toc_depth: 2 output_dir: "docs" documentclass: article fontsize: 12pt mainfont: Arial mathfont: Arial monofont: DejaVu Sans Mono classoption: a4paper bibliography: ../resources/references.bib
```

For more info on R Markdown from this course:

file:///Users/sara/Documents/PhD/HELP/STATISTICS%20and%20GRAPHS/-%20R%20and%20STATS%20COURSES/2020%20Murray%20Logan%20R%20Course/R_files/R_files/docs/reproducible_research_rmarkdown.html

PACKAGES

Install (Tools -> Install packages) and load the following packages

The last package loaded will define the functions to use (will overwrite functions from other packages with same name). Murray recommends using the package name before the code, eg. dplyr::summarize

Dplyr

Look up the cheatsheet to know what manipulation functions are available to us

Link to the data transformation cheatsheet

<https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf>

Important data manipulation functions:

Task	Function	Package
Sorting	arrange()	dplyr
Re-ordering factor levels	factor()	base
Re-labelling	recode()	dplyr
Re-naming columns	rename()	dplyr
Filtering/subsetting	select()	dplyr
Transformations	mutate()	dplyr
Adding columns	mutate()	dplyr
Re-shaping data	gather()/spread()	tidyverse
Aggregating	summarize(), group_by()	dplyr
Merging/joining	*_join()	dplyr

DATA WRANGLING

Open the .Rmd called "data_wrangling.Rmd" (in folder "ws")

Open the Powerpoint: file:///Users/sara/Documents/PhD/HELP/STATISTICS%20and%20GRAPHS/-%20R%20and%20STATS%20COURSES/2020%20Murray%20Logan%20R%20Course/WORKSHOP%20MATERIALS/WORKSHOP%20MATERIALS/LINKS%20TO%20THE%20MATERIALS/SUYR_public-master/docs/pres2.4.html#24

Piping

Piping is relatively new. “Pipe”=The output of one application is piped (sent to) another application. In other words, the output of one application becomes the input of another. It prevents having to have functions nested within functions, within more functions, etc.

For example, it would be syntactically correct to have sth like this:

An analogous would be to write:

... which is easier to understand.

Another example: `data %>% # Data must be first select() %>% group_by() %>% summarise()`

It is important that the data of each piping command is in the same format (vector, matrix, dataframe):

A vector is a line of numbers

A matrix is like a table, with width and length with variables that have are the same type (eg. numeric)

A dataframe looks like a spreadsheet, and can take all sorts of variables

Load a dataset

examine the dataset

Dataframes versus tibbles

A tibble resembles a lot a dataframe, having also rows and columns and allowing to have several variables. BUT a tibble can have any other object in any cell (not just a word or number). You could eg. put a nested dataframe in the cell of a tibble. By doing this you can avoid looping. A tibble also guarantees that it will fit on the screen, truncating the rows and columns to the window you are looking at. If there were too many columns, it would just show a few of them and tell us that there are additional columns that are not being displayed. However, we are not going to use tibbles in this course

Sorting data = Arranging it in order

To sort the data in ascending order:

If we wanted to clear the arrangement step, we would have to save it:

To sort the data in descending order:

To sort by two (or more) variables:

To sort by the sum of two (or more) variables:

To sort by two (or more) variables, combining them altogether (example)

`<-` or `=` ?

We can use either `<-` or `=` (same result). However, when creating a lm we CANNOT use “`=`” eg. `summary(mod=lm(...))` # Would recognise mod as a function, not as the name of a lm

We would have to write: `summary(mod<-lm(...))` # Would recognise mod as the name of the model, and calculate the summary values for it.

Adding columns with mutate()

Mutate adds a new column, or modifies an existing column.

Transformations

Centering data

Centering data = You shift the number line so that the mean is = 0. And you do so by subtracting the mean from each value. Centering is very important for continuous variables in linear modelling.

Centering your predictors allows your y intercept to have meaning, as it will be = 0, and the graphs plotting the lm will have the intercept at the mean.

Changing vector types (character, factor, etc)

Changing factor labels

To have a look at a specific column/variable:

Renaming the variables of a specific column:

Window functions - lead and lag

lead takes your data and moves a window/shifts it back one observation = the second observation of a variable becomes the first (lead) or viceversa (lag). This is useful for time series, eg. to compare how a response compares to a year ago.

Summarise (slide 77)

Calculates specific summary values for the specified variable(s), with an output of one row (=avg value). With mutate+summarise, the output will create summary variables for each row.

Calculate a mean for a particular variable (Temp):

Summarise across different variables

For all the numeric variables calculate the mean, for all the categorical the length

As a comparison, the command mutate() would add mean and length to each row

A function to calculate SE

Grouping (aggregating)

Grouping by two categorical combinations and asking for the Mean

We can add other statistics here (Mean, Var Length, etc) and they will be added separately for those combinations of groups

(!) group_by will split your data into groups forever. You will have to specify %>% ungroup at the end of the next code chunk to make sure this does not happen. The grouping would also be maintained for new created dataframes.

- I am not sure though if this is a bit outdated, as I have tried the command below without the %>% ungroup and it did not group the variables as specified in the previous chunk.

Selecting columns (select)

Selecting specific columns (slide 33)

You could use this command to create a new dataset

Deselecting specific columns with “-”

Helping functions for select (work only for columns)

contains() Selecting anything that contains an “L” (dissapointingly, this command is NOT case sensitive)

Selecting anything that starts with an “L”

Selecting anything that ends with a “t”

Selecting anything that matches a regular expression

In this case, we are looking for anything: - whose capital letter is a T \rightarrow ^T - continued by anything between a-z \rightarrow [a-z] - that contains an m - that can be anything (chr, number...) \rightarrow .

Selecting the columns between Between and Temp

Another example

Selecting lat, long and cloud... columns

Selecting awkward names (for example with spaces) - use ' ' instead of " "

Selecting columns that don't contain a specific name

Renaming columns (vectors)

Filtering by rows

Filtering all rows of the variable Cond contain a H

Filtering all rows of the variable Cond that contain a H or an M

Alternative way of filtering all rows of the variable Cond that contain a H or an M

Filtering all rows of the variable Cond that contain a H, and all rows of the variable Temp that are <25

Filtering all rows of the variable Cond that contain a H, OR all rows of the variable Temp that are <25

Filtering only those rows with Temp less than 20 and LAT greater than 20, or LONG less than 145

From the dataset “nasa”, filter to all ozone values between 320 and 325 in the second month of the last year

Effects of filtering

Examine the levels of the Cond factor

Subset the dataset to just Cond H

Examine the levels of the Cond factor

We usually make the follwing mistakes when filtering...

To retain the same ammount of levels whenever filtering, write droplevels()

To specify “doesn’t”, we write a !

!1=2

Reshaping data frames (slide 95)

This is especially useful when transforming classically collected wide data, in a repeated measures type of form. Long data wpuld take all those repeated measures, and put them in a single variable

Wide to long (melt)

Long to wide (spread out)

This is useful to find what might be missing

Widen Resp1 and Resp2 for repeated measures (Within)

```
data %>% pivot_wider(names_from=Within, values_from=rep # sorry, missed the end)
```

Merging data frames

When merging dataframes, we will have different variables, and even row names. What does R do?

Option 1: An inner join - only keeping full matches

Option 2: Outer join - Merging bio and chem data (keeping all data)

Option 3: left join - Merge bio and chem data (only keep full BIO matches)

Option 4: Right join - Merge bio and chem data (only keep full CHEM matches)

Different ways of grouping data (I got a bit lost here, sorry)

2_Graphics

Sara Kophamel

01/12/2020

Presentation 5.2

file:///Users/sara/Documents/PhD/HELP/STATISTICS%20and%20GRAPHS/-%20R%20and%20STATS%20COURSES/2020%20Murray%20Logan%20R%20Course/Logan_Course/docs/pres5.2.html#3

PACKAGES

```
library(patchwork)
library(scales)
library(cowplot)

## 
## Attaching package: 'cowplot'

## The following object is masked from 'package:patchwork':
## 
##     align_plots

library(grid)
library(xkcd)

## Loading required package: ggplot2
## Loading required package: extrafont
## Registering fonts with R
# library(sysfonts) # Won't find the package
library(extrafont)
library(maps)
library(mapdata)
library(mapproj)
library(gmodels)
library(gridExtra)
library(tidyverse)

## -- Attaching packages -----
## v tibble   3.0.3      v dplyr    1.0.2
## v tidyr    1.1.2      v stringr  1.4.0
## v readr    1.3.1      vforcats  0.5.0
## v purrr    0.3.4

## -- Conflicts -----
## x readr::col_factor() masks scales::col_factor()
## x dplyr::combine()    masks gridExtra::combine()
```

```
## x purrr::discard()      masks scales::discard()
## x dplyr::filter()       masks stats::filter()
## x dplyr::lag()          masks stats::lag()
## x purrr::map()          masks maps::map()
```

GRAPHICS INFRASTRUCTURE

- Layers of data driven objects
- Coordinates system
- Scales
- Faceting (arranging multiple graphs together)
- Themes (overall look)

DATASET TO PLAY WITH

Examining the dataset

```
head(BOD)
```

```
##   Time demand
## 1    1   8.3
## 2    2  10.3
## 3    3  19.0
## 4    4  16.0
## 5    5  15.6
## 6    7  19.8
```

```
summary(BOD)
```

```
##        Time           demand
##  Min.   :1.000   Min.   : 8.30
##  1st Qu.:2.250   1st Qu.:11.62
##  Median :3.500   Median :15.80
##  Mean   :3.667   Mean   :14.83
##  3rd Qu.:4.750   3rd Qu.:18.25
##  Max.   :7.000   Max.   :19.80
```

Example plot - Setting all parameters

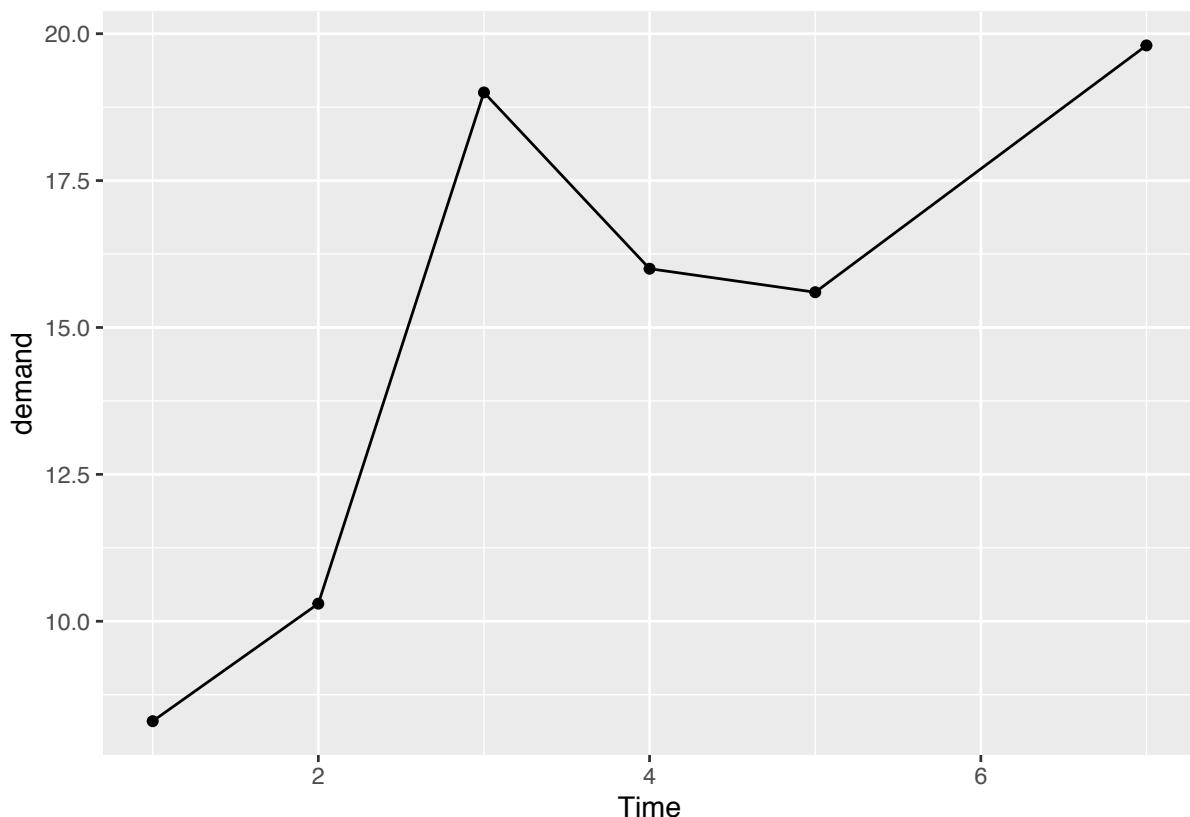
```
ggplot
```

```
## function (data = NULL, mapping = aes(), ..., environment = parent.frame())
## {
##   UseMethod("ggplot")
## }
## <bytecode: 0x7fb75dd57888>
## <environment: namespace:ggplot2>

p <- ggplot() + #single layer - points
  layer(data=BOD, #data.frame
        mapping=aes(y=demand,x=Time),
        stat="identity", #use original data
        geom="point", #plot data as points
        position="identity",
        params = list(na.rm = TRUE),
        show.legend = FALSE
```

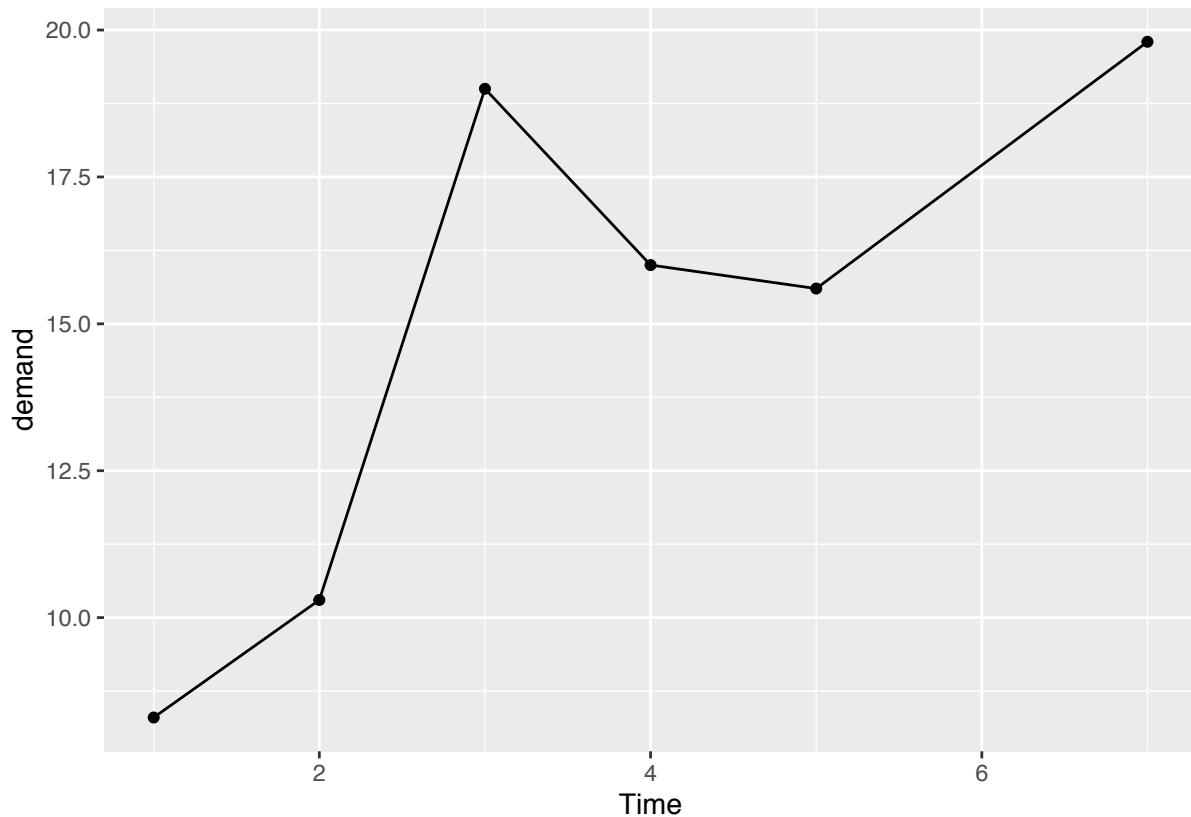
```
) + #layer of lines  
layer( data=BOD, #data.frame  
      mapping=aes(y=demand,x=Time),  
      stat="identity", #use original data  
      geom="line", #plot data as a line  
      position="identity",  
      params = list(na.rm = TRUE),  
      show.legend = FALSE  
) +  
coord_cartesian() + #cartesian coordinates  
scale_x_continuous() + #continuous x axis  
scale_y_continuous() #continuous y axis
```

p



Alternative method - Using R's defaults. Same result

```
ggplot(data=BOD, map=aes(y=demand,x=Time)) + geom_point() + geom_line()
```



Step by step plot creation:

1- Indicate the data to use

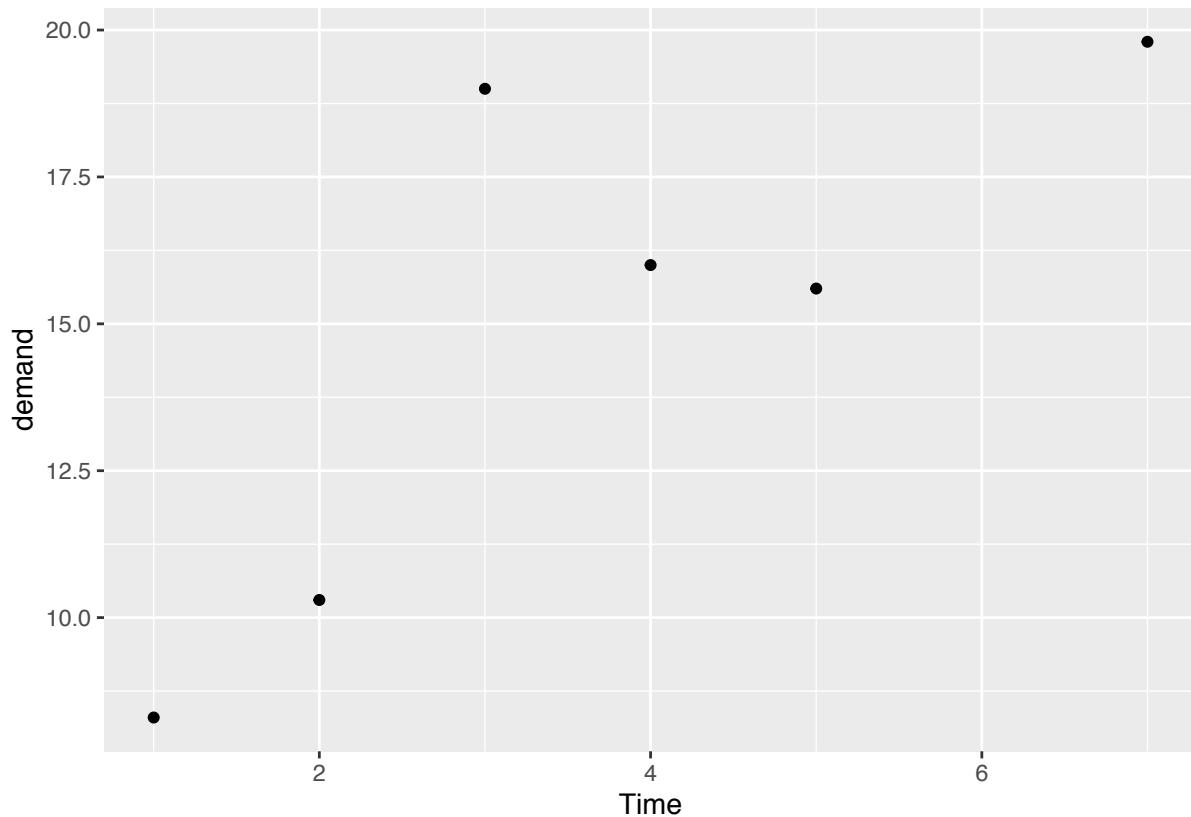
```
p<-ggplot(data=BOD)
```

```
p
```

2-

Adding layers (geoms)

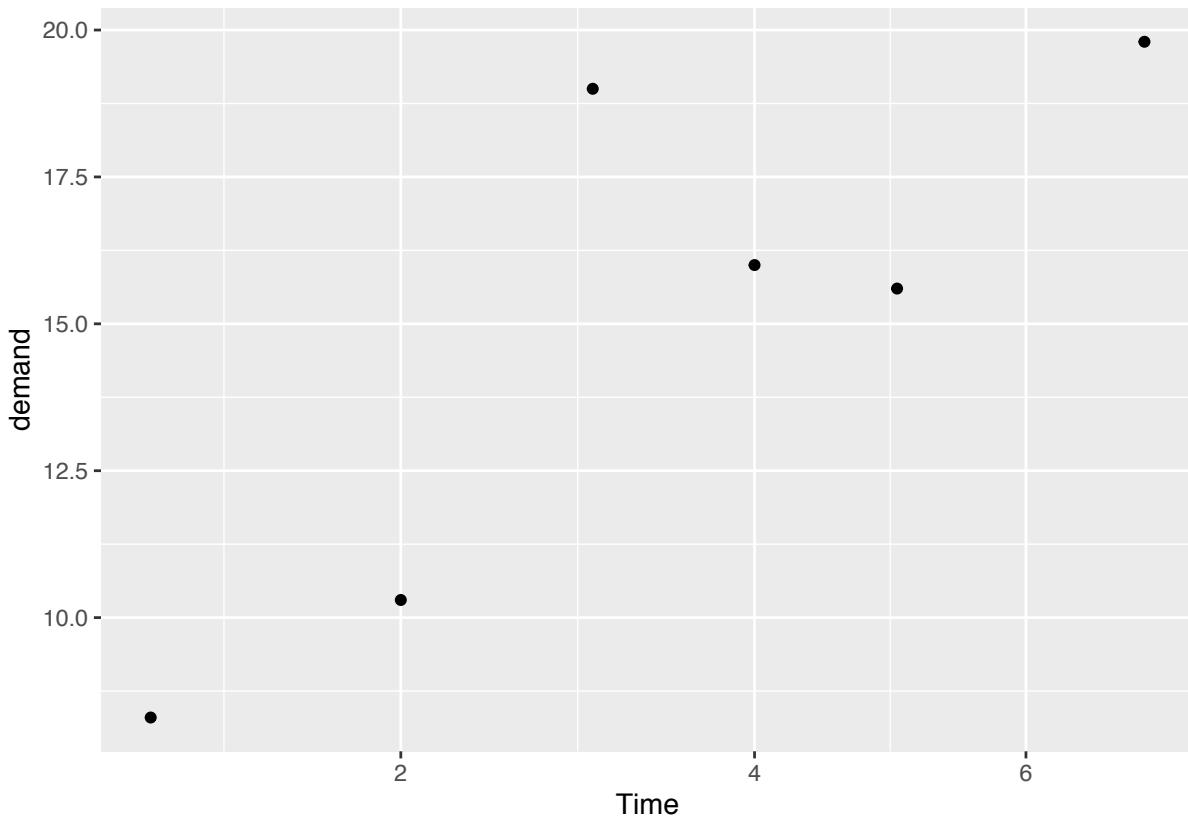
```
p<-p + geom_point(aes(y=demand, x=Time))  
p
```



3-

Change the scale name and type (square root)

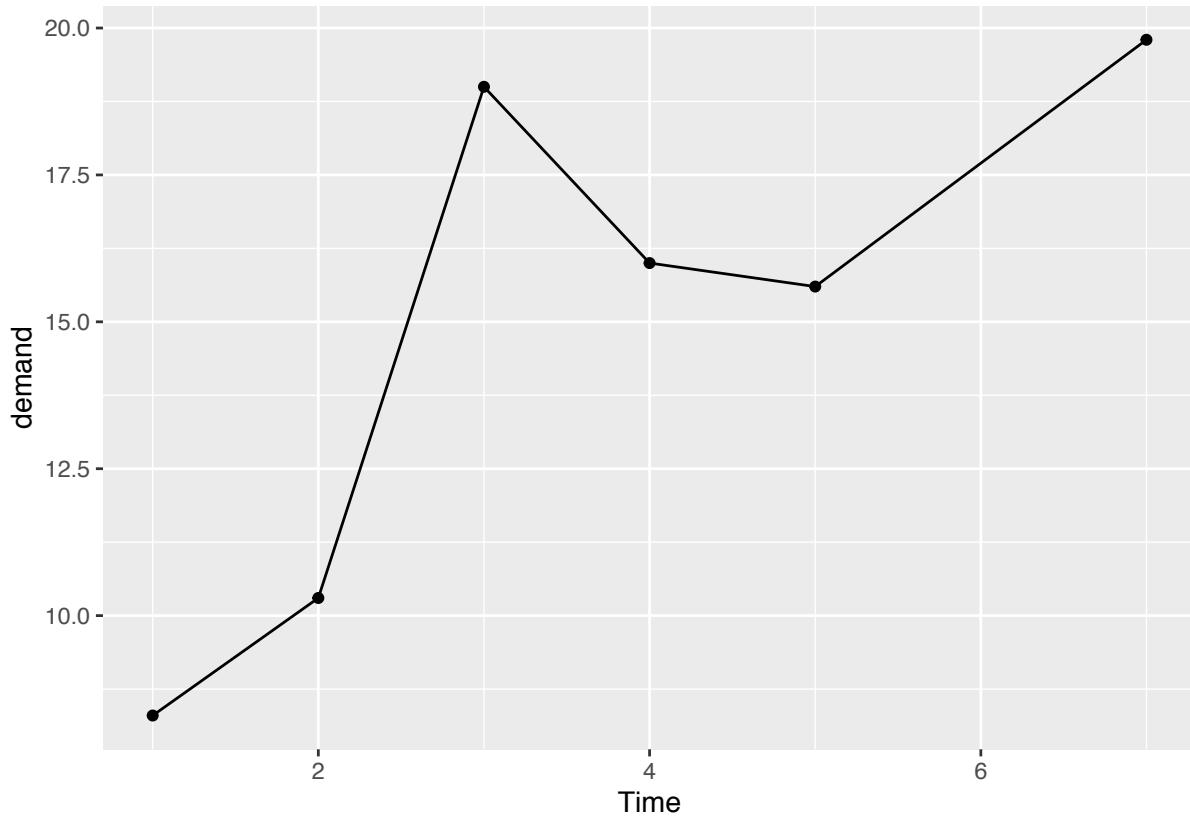
```
p <- p + scale_x_sqrt(name="Time")  
p
```



Add the datapoints

```
ggplot(data=BOD, map=aes(y=demand,x=Time)) + geom_point() + geom_line()
```

4-



Checking how R recognises our plot p

```
class(p)
```

```
## [1] "gg"      "ggplot"
```

We could also look at all the instructions to create this graph

```
str(p)
```

```
## List of 9
## $ data      :'data.frame': 6 obs. of  2 variables:
##   ..$ Time   : num [1:6] 1 2 3 4 5 7
##   ..$ demand: num [1:6] 8.3 10.3 19 16 15.6 19.8
##   ..- attr(*, "reference")= chr "A1.4, p. 270"
## $ layers    :List of 1
##   ..$ :Classes 'LayerInstance', 'Layer', 'ggproto', 'gg' <ggproto object: Class LayerInstance, Layer
##     aes_params: list
##     compute_aesthetics: function
##     compute_geom_1: function
##     compute_geom_2: function
##     compute_position: function
##     compute_statistic: function
##     data: waiver
##     draw_geom: function
##     finish_statistics: function
##     geom: <ggproto object: Class GeomPoint, Geom, gg>
##       aesthetics: function
##     default_aes: uneval
##     draw_group: function
```

```

##      draw_key: function
##      draw_layer: function
##      draw_panel: function
##      extra_params: na.rm
##      handle_na: function
##      non_missing_aes: size shape colour
##      optional_aes:
##      parameters: function
##      required_aes: x y
##      setup_data: function
##      setup_params: function
##      use_defaults: function
##      super: <ggproto object: Class Geom, gg>
##      geom_params: list
##      inherit.aes: TRUE
##      layer_data: function
##      map_statistic: function
##      mapping: uneval
##      position: <ggproto object: Class PositionIdentity, Position, gg>
##      compute_layer: function
##      compute_panel: function
##      required_aes:
##      setup_data: function
##      setup_params: function
##      super: <ggproto object: Class Position, gg>
##      print: function
##      setup_layer: function
##      show.legend: NA
##      stat: <ggproto object: Class StatIdentity, Stat, gg>
##      aesthetics: function
##      compute_group: function
##      compute_layer: function
##      compute_panel: function
##      default_aes: uneval
##      extra_params: na.rm
##      finish_layer: function
##      non_missing_aes:
##      optional_aes:
##      parameters: function
##      required_aes:
##      retransform: TRUE
##      setup_data: function
##      setup_params: function
##      super: <ggproto object: Class Stat, gg>
##      stat_params: list
##      super: <ggproto object: Class Layer, gg>
## $ scales      :Classes 'ScalesList', 'ggproto', 'gg' <ggproto object: Class ScalesList, gg>
##      add: function
##      clone: function
##      find: function
##      get_scales: function
##      has_scale: function
##      input: function
##      n: function

```

```

##      non_position_scales: function
##      scales: list
##      super: <ggproto object: Class ScalesList, gg>
##      $ mapping      : Named list()
##      ..- attr(*, "class")= chr "uneval"
##      $ theme       : list()
##      $ coordinates:Classes 'CoordCartesian', 'Coord', 'ggproto', 'gg' <ggproto object: Class CoordCartesia...
##      aspect: function
##      backtransform_range: function
##      clip: on
##      default: TRUE
##      distance: function
##      expand: TRUE
##      is_free: function
##      is_linear: function
##      labels: function
##      limits: list
##      modify_scales: function
##      range: function
##      render_axis_h: function
##      render_axis_v: function
##      render_bg: function
##      render_fg: function
##      setup_data: function
##      setup_layout: function
##      setup_panel_guides: function
##      setup_panel_params: function
##      setup_params: function
##      train_panel_guides: function
##      transform: function
##      super: <ggproto object: Class CoordCartesian, Coord, gg>
##      $ facet        :Classes 'FacetNull', 'Facet', 'ggproto', 'gg' <ggproto object: Class FacetNull, Facet...
##      compute_layout: function
##      draw_back: function
##      draw_front: function
##      draw_labels: function
##      draw_panels: function
##      finish_data: function
##      init_scales: function
##      map_data: function
##      params: list
##      setup_data: function
##      setup_params: function
##      shrink: TRUE
##      train_scales: function
##      vars: function
##      super: <ggproto object: Class FacetNull, Facet, gg>
##      $ plot_env    :<environment: R_GlobalEnv>
##      $ labels      :List of 2
##      ..$ x: chr "Time"
##      ..$ y: chr "demand"
##      - attr(*, "class")= chr [1:2] "gg" "ggplot"

```

```
# If we don't store these instructions, R will simply create the graph, and loose the set of instructions
```

LAYERS

= Layers of data driven objects.

Layers are linked to:
- geometric objects to represent data (eg. points)
- statistical methods to summarize the data:
In the case of ggplot, we rely on “useless” stats (we don’t provide ggplot a statistical analysis to create a graph). We provide an identity instead. An identity is a function that multiplies sth by 1. The statistic that you apply to ggplot is an identity = “Take my data and plot it as it is”
- mapping of aesthetics: What is on the x,y axis, what determines the color of things, etc
- positioning: What do we do with data that overlaps?

Types of identities:

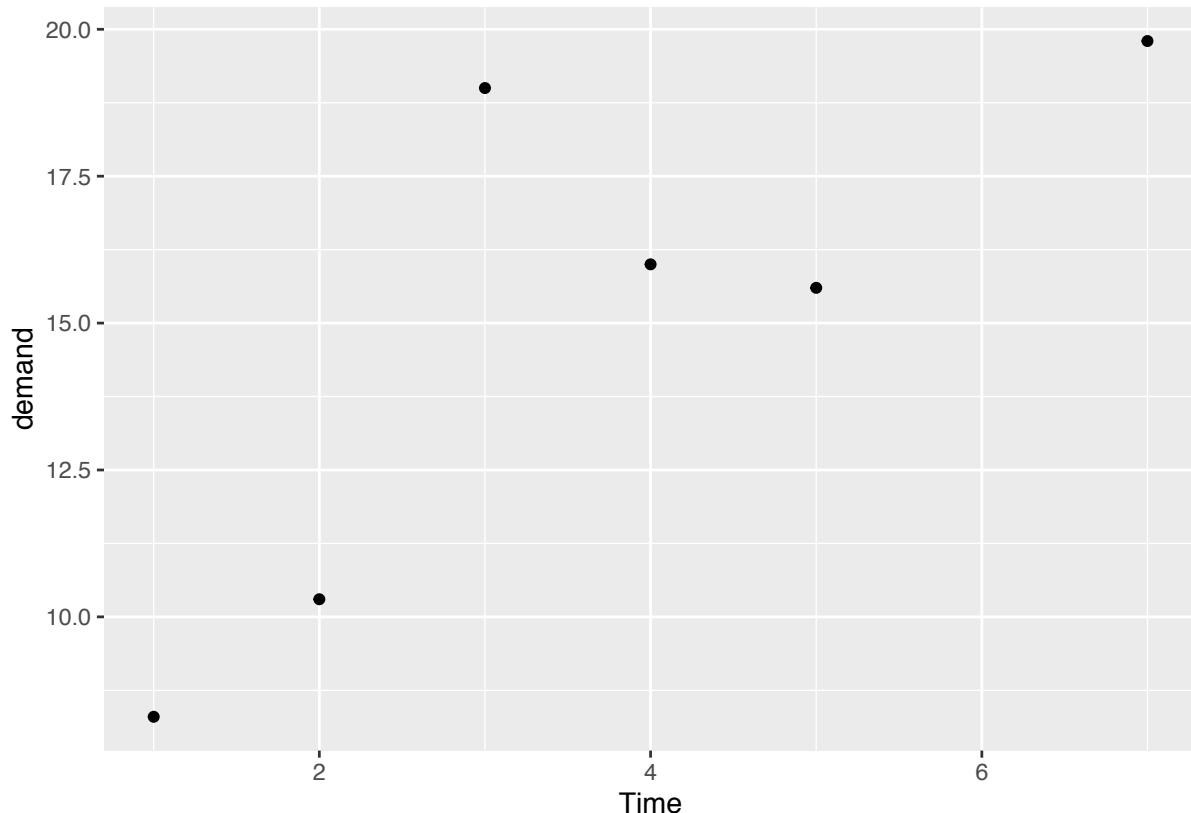
geom_ and stat_ - Both are similar. They are coupled together, and engage either.
- stat_identity

geom

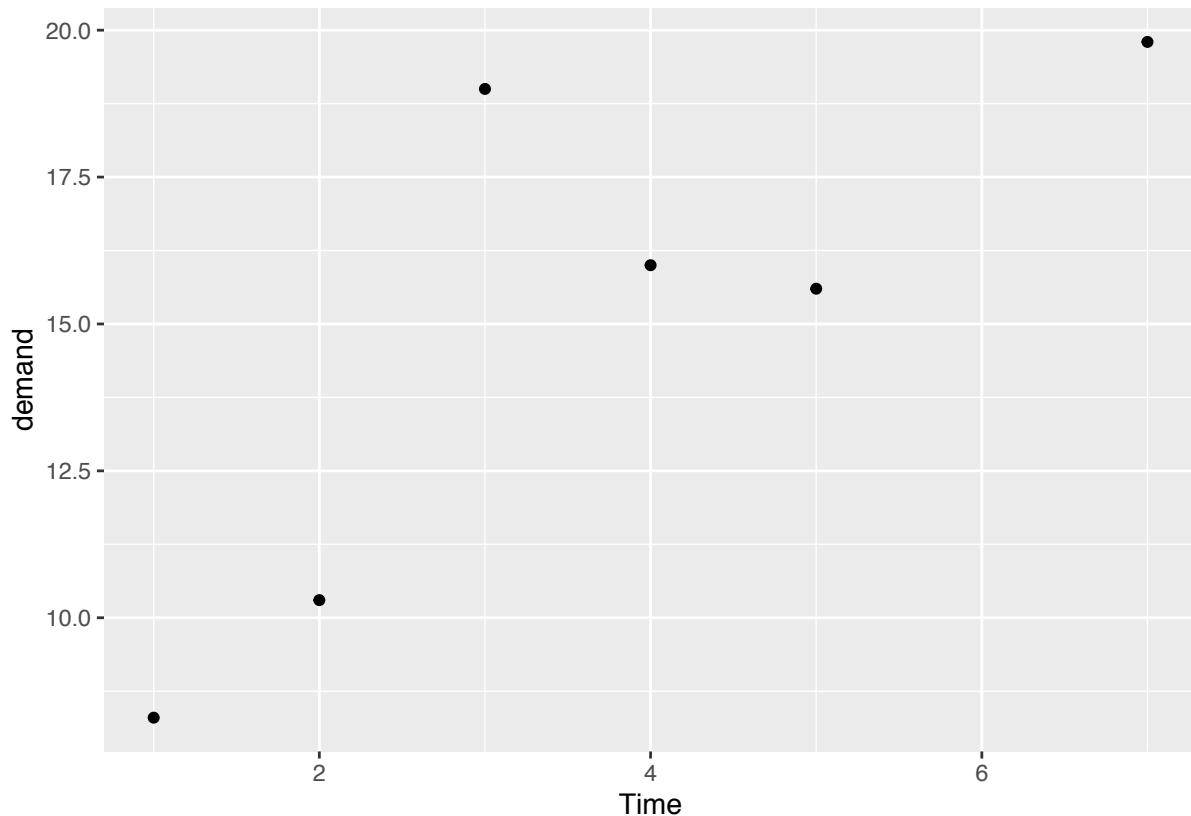
- data - obvious
- mapping - aesthetics If omitted, inherited from ggplot()
- stat - the stat_ function
- position - overlapping geoms

Example

```
ggplot(data=BOD, aes(y=demand, x=Time)) + geom_point()
```



```
#OR  
ggplot(data=BOD) + geom_point(aes(y=demand, x=Time))
```



Optional mapping

- alpha - transparency
- colour - outer colour of the geometric features
- fill - inner colour of the geometric features
- linetype - fill colour of geometric features
- size - size of geometric features such as points or text
- shape - shape of geometric features such as points
- weight - weightings of values

Dataset to play with: CO2

Examine the dataset

```
head(CO2)
```

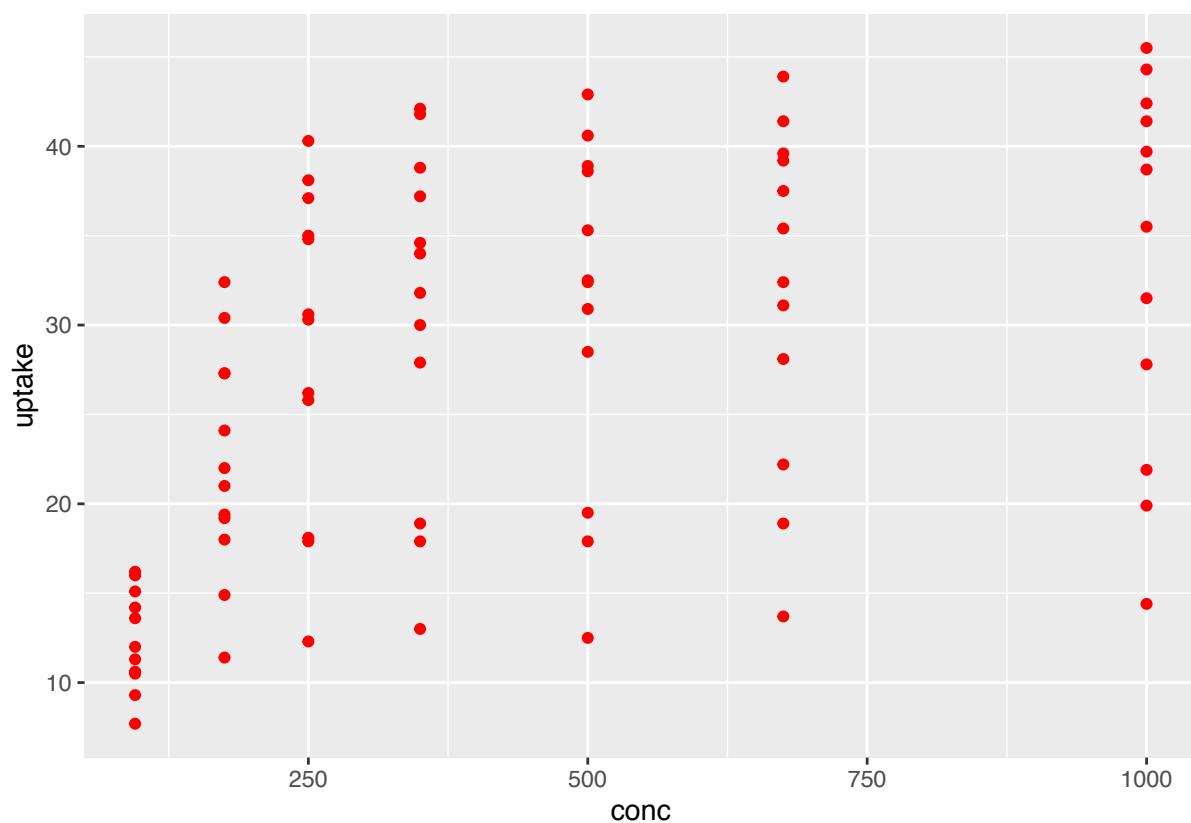
```
##   Plant  Type Treatment conc uptake
## 1   Qn1 Quebec nonchilled  95   16.0
## 2   Qn1 Quebec nonchilled 175   30.4
## 3   Qn1 Quebec nonchilled 250   34.8
## 4   Qn1 Quebec nonchilled 350   37.2
## 5   Qn1 Quebec nonchilled 500   35.3
## 6   Qn1 Quebec nonchilled 675   39.2
```

```
summary(CO2)
```

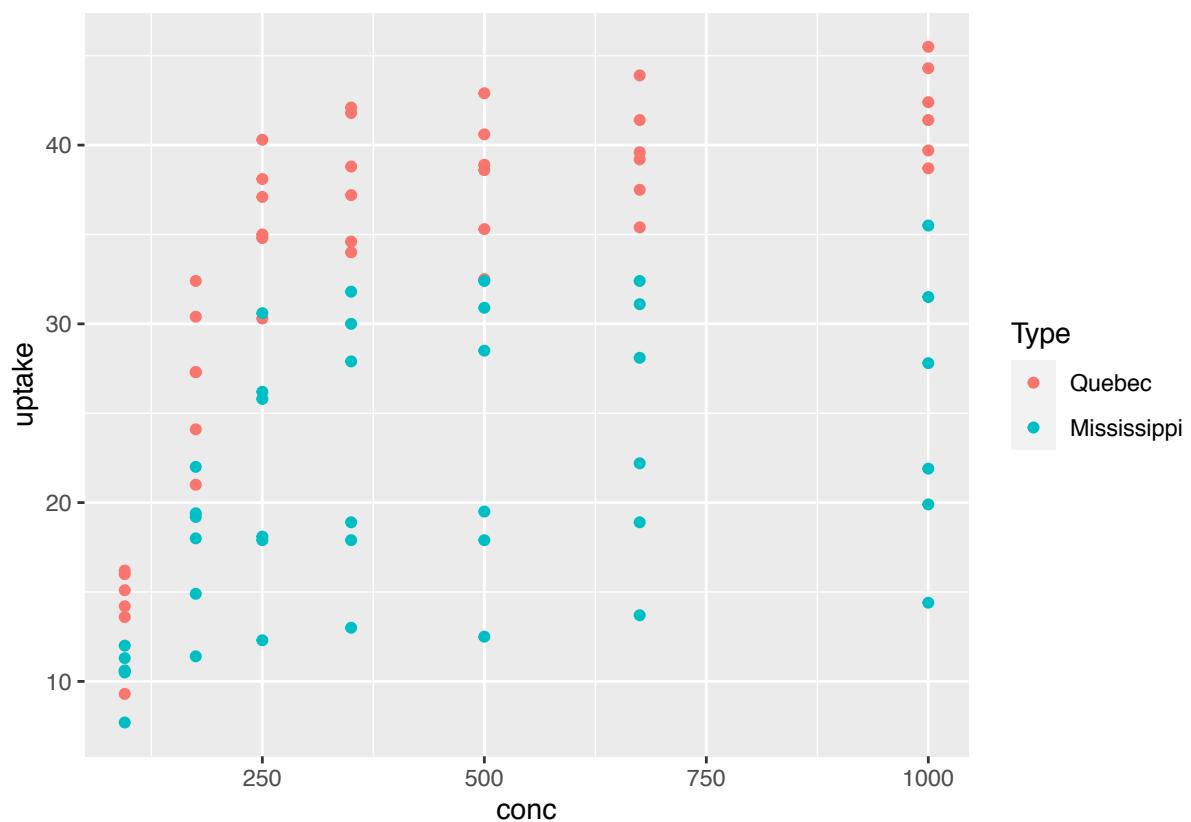
```
##      Plant          Type      Treatment      conc       uptake
##  Qn1    : 7   Quebec     :42 nonchilled:42  Min.   : 95  Min.   : 7.70
##  Qn2    : 7 Mississippi:42     chilled   :42  1st Qu.:175  1st Qu.:17.90
##  Qn3    : 7                               Median  :350  Median :28.30
##  Qc1    : 7                               Mean   :435  Mean   :27.21
##  Qc3    : 7                               3rd Qu.:675  3rd Qu.:37.12
##  Qc2    : 7                               Max.   :1000  Max.   :45.50
##  (Other):42
```

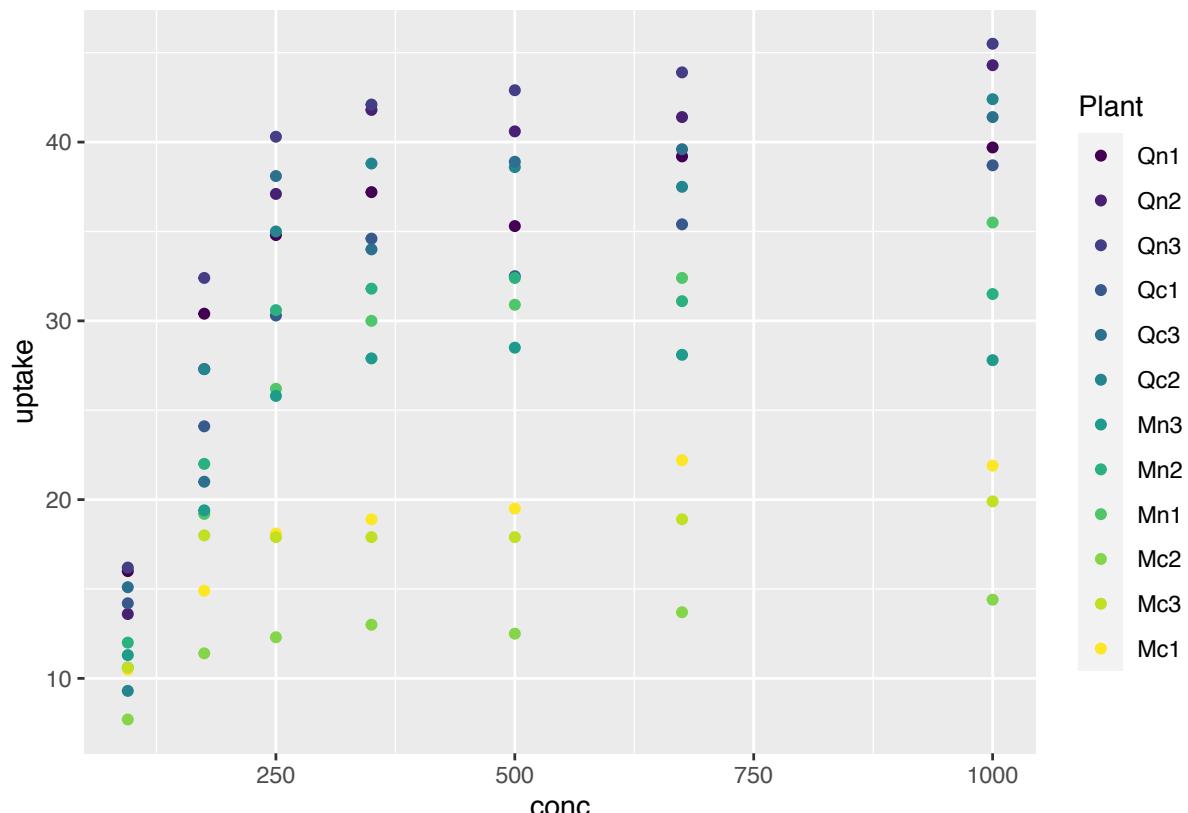
If you define CO2 by mistake as a plot (happened to me), write rm(CO2), and redo the graph.

```
ggplot(CO2)+geom_point(aes(x=conc,y=uptake), colour="red")
```



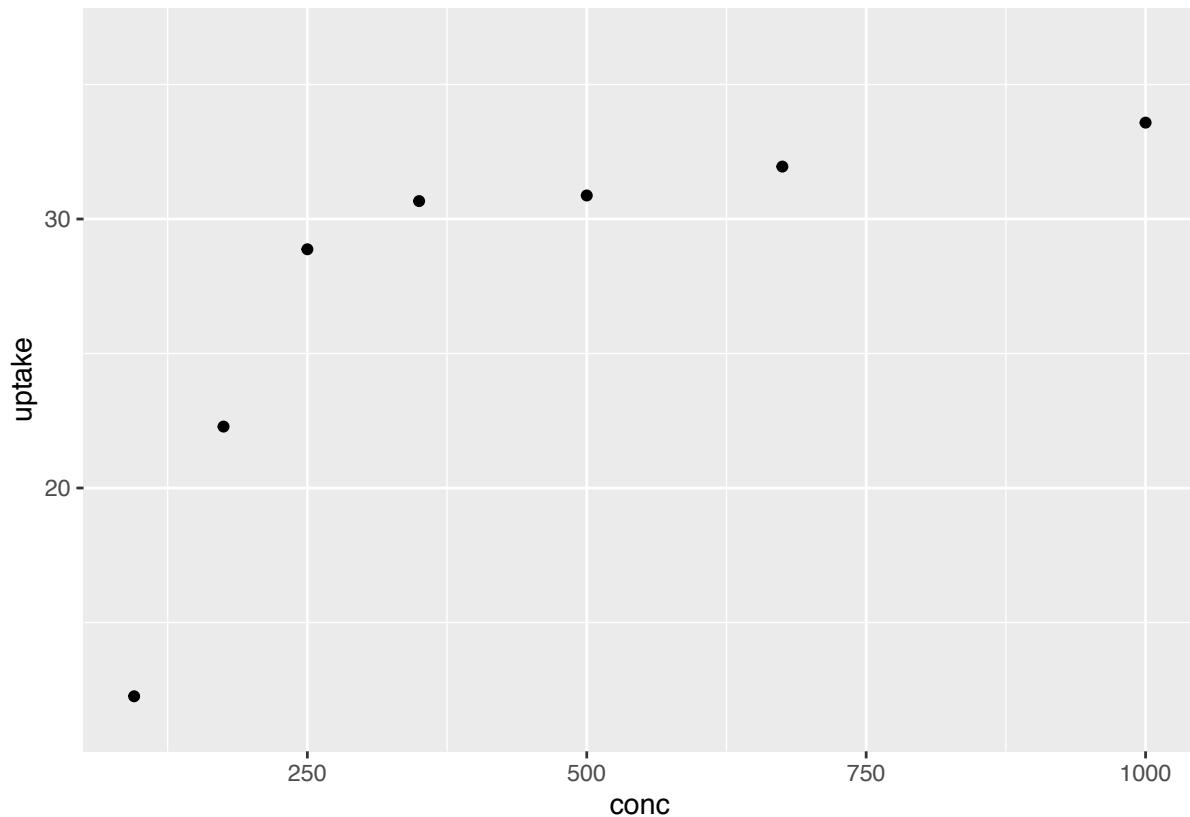
```
ggplot(CO2)+geom_point(aes(x=conc,y=uptake, colour=Type))
```





```
ggplot(CO2)+geom_point(aes(x=conc,y=uptake),
stat="summary",fun.y=mean) # fun.y = function on y axis
```

```
## Warning: Ignoring unknown parameters: fun.y
## No summary function supplied, defaulting to `mean_se()`
```



EXAMPLE PLOTS

Diamonds data set

```
head(diamonds)

## # A tibble: 6 x 10
##   carat     cut      color clarity depth table price     x     y     z
##   <dbl>    <ord>    <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23   Ideal     E     SI2     61.5   55   326  3.95  3.98  2.43
## 2 0.21   Premium   E     SI1     59.8   61   326  3.89  3.84  2.31
## 3 0.23   Good      E     VS1     56.9   65   327  4.05  4.07  2.31
## 4 0.290  Premium   I     VS2     62.4   58   334  4.2   4.23  2.63
## 5 0.31   Good      J     SI2     63.3   58   335  4.34  4.35  2.75
## 6 0.24   Very Good J     VVS2    62.8   57   336  3.94  3.96  2.48
# cut is an ordinal scale, with categories

summary(diamonds)

##      carat          cut        color       clarity       depth
##  Min.   :0.2000   Fair     : 1610   D: 6775   SI1     :13065   Min.   :43.00
##  1st Qu.:0.4000   Good    : 4906   E: 9797   VS2     :12258   1st Qu.:61.00
##  Median :0.7000   Very Good:12082  F: 9542   SI2     : 9194   Median :61.80
##  Mean   :0.7979   Premium  :13791   G:11292   VS1     : 8171   Mean   :61.75
##  3rd Qu.:1.0400   Ideal    :21551   H: 8304   VVS2    : 5066   3rd Qu.:62.50
##  Max.   :5.0100                    I: 5422   VVS1    : 3655   Max.   :79.00
##                                J: 2808   (Other) : 2531
##      table         price        x           y
##      Min.   :1000   Min.   :326000   Min.   :18
##  1st Qu.:1000   1st Qu.:326000   1st Qu.:18
##  Median :1000   Median :326000   Median :18
##  Mean   :1000   Mean   :326000   Mean   :18
##  3rd Qu.:1000   3rd Qu.:326000   3rd Qu.:18
##  Max.   :1000   Max.   :326000   Max.   :18
```

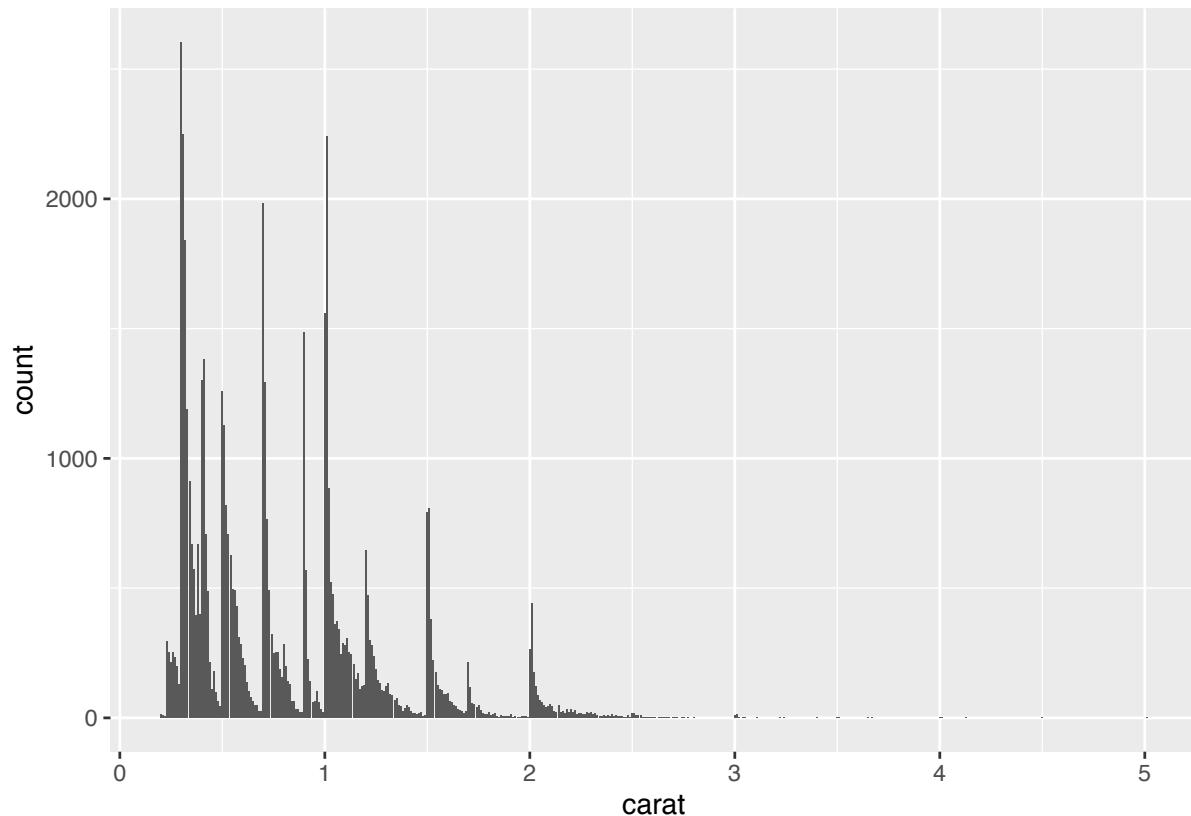
```
##  Min.   :43.00   Min.   : 326   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:56.00   1st Qu.: 950   1st Qu.: 4.710   1st Qu.: 4.720
##  Median :57.00   Median :2401    Median : 5.700   Median : 5.710
##  Mean   :57.46   Mean   :3933    Mean   : 5.731   Mean   : 5.735
##  3rd Qu.:59.00   3rd Qu.:5324    3rd Qu.: 6.540   3rd Qu.: 6.540
##  Max.   :95.00   Max.   :18823   Max.   :10.740   Max.   :58.900
##
##          z
##  Min.   : 0.000
##  1st Qu.: 2.910
##  Median : 3.530
##  Mean   : 3.539
##  3rd Qu.: 4.040
##  Max.   :31.800
##
```

geom_bar

Produces a binning statistic (counts how many data points are in a specified range of x values)

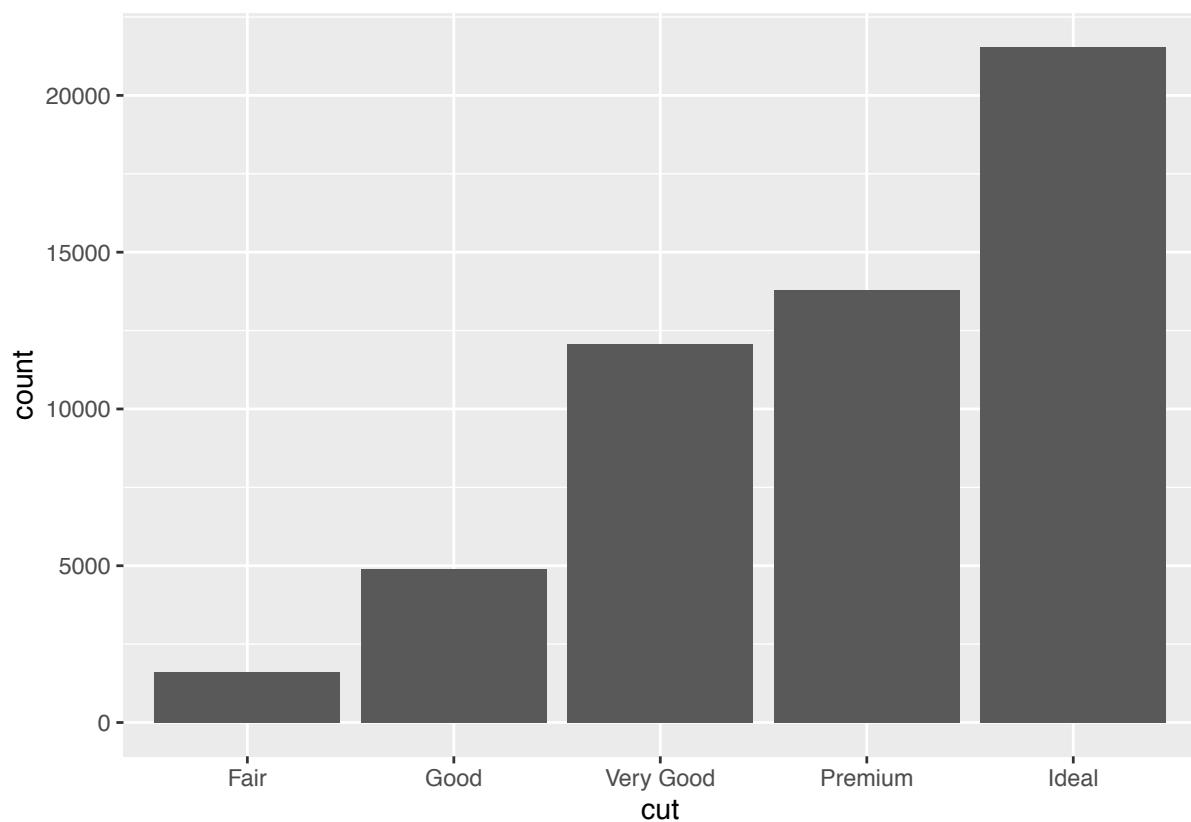
continuous variable

```
ggplot(diamonds) + geom_bar(aes(x = carat))
```



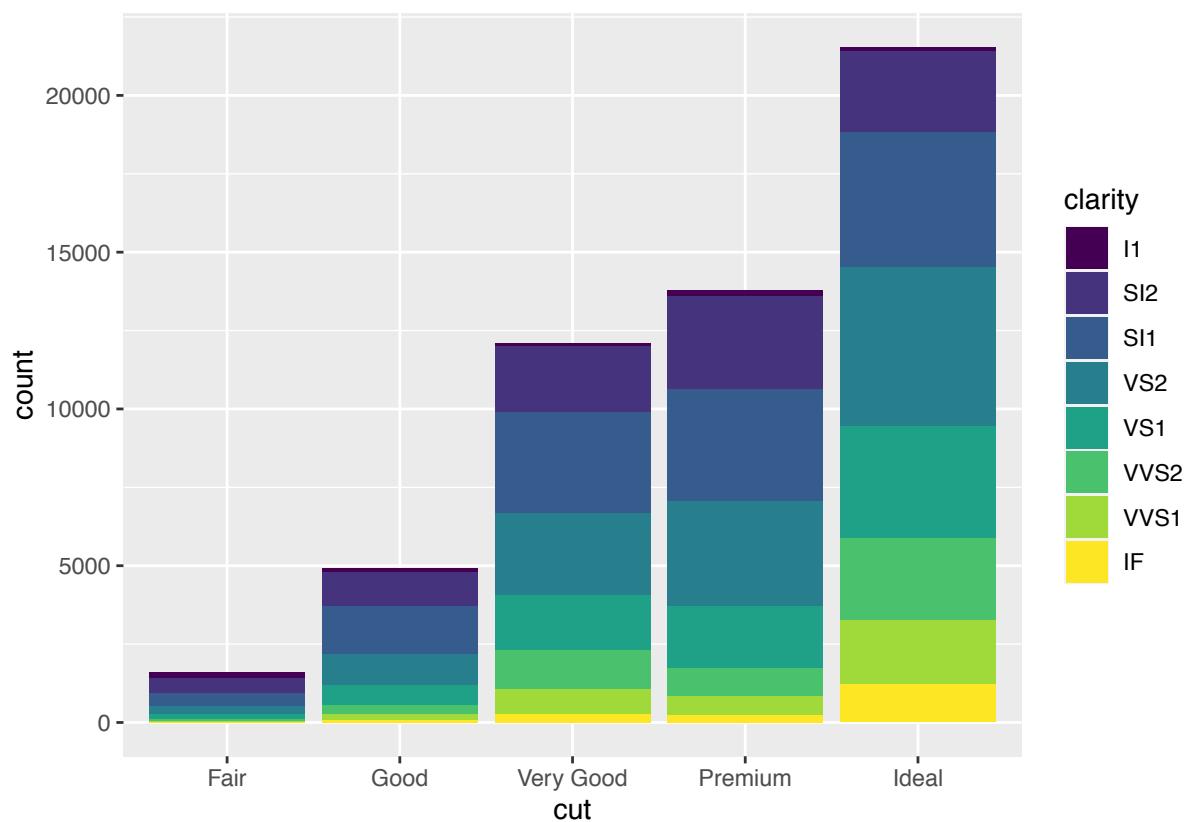
categorical variable In order

```
ggplot(diamonds) + geom_bar(aes(x = cut))
```



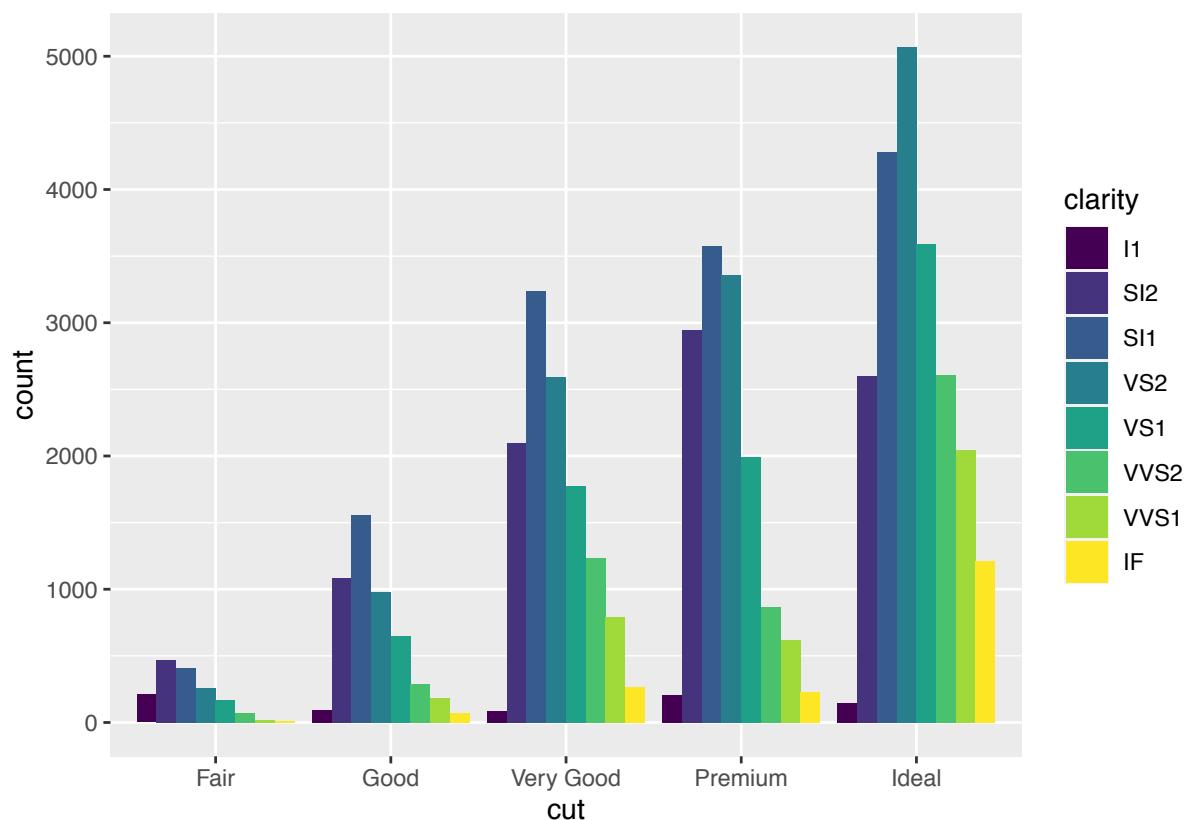
```
### multiple categorical variables - stacked bar
```

```
ggplot(diamonds) + geom_bar(aes(x = cut, fill = clarity))
```



multiple categorical variables - dodged

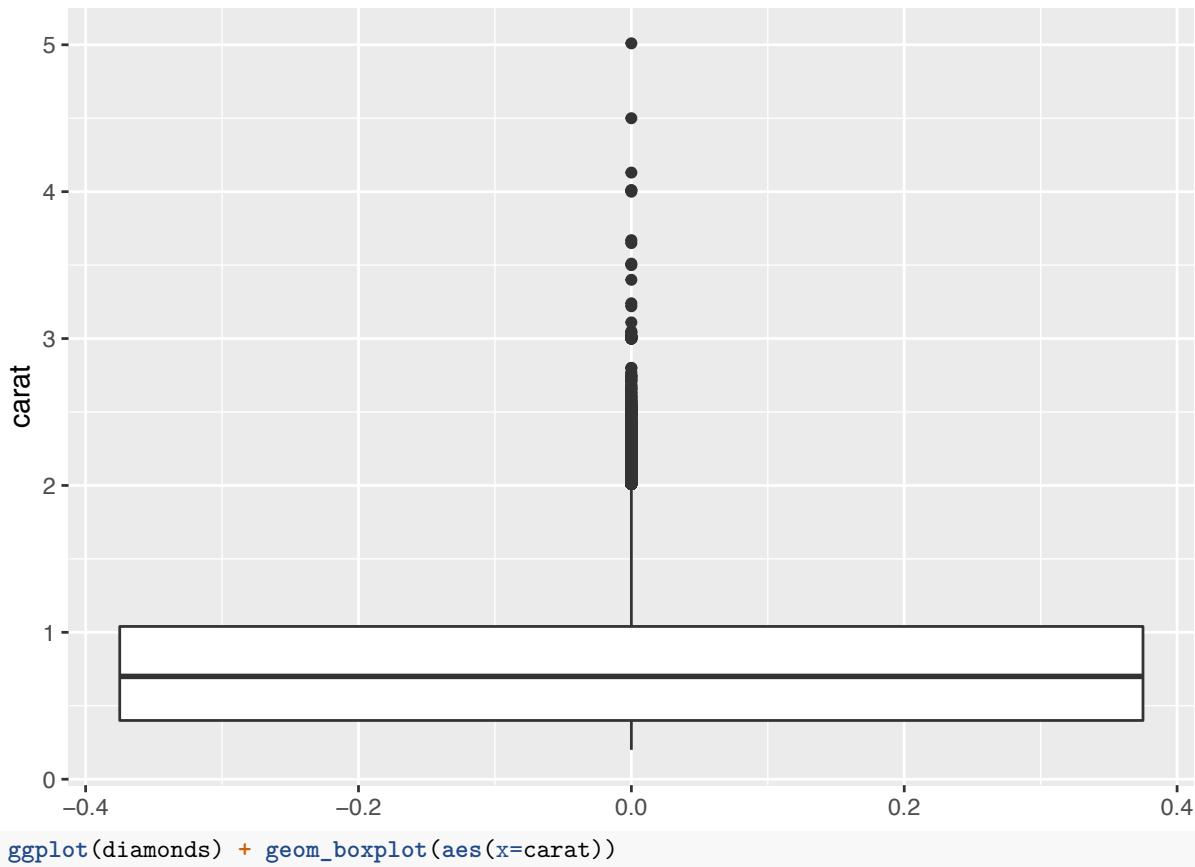
```
ggplot(diamonds) + geom_bar(aes(x = cut, fill = clarity),  
  position='dodge')
```

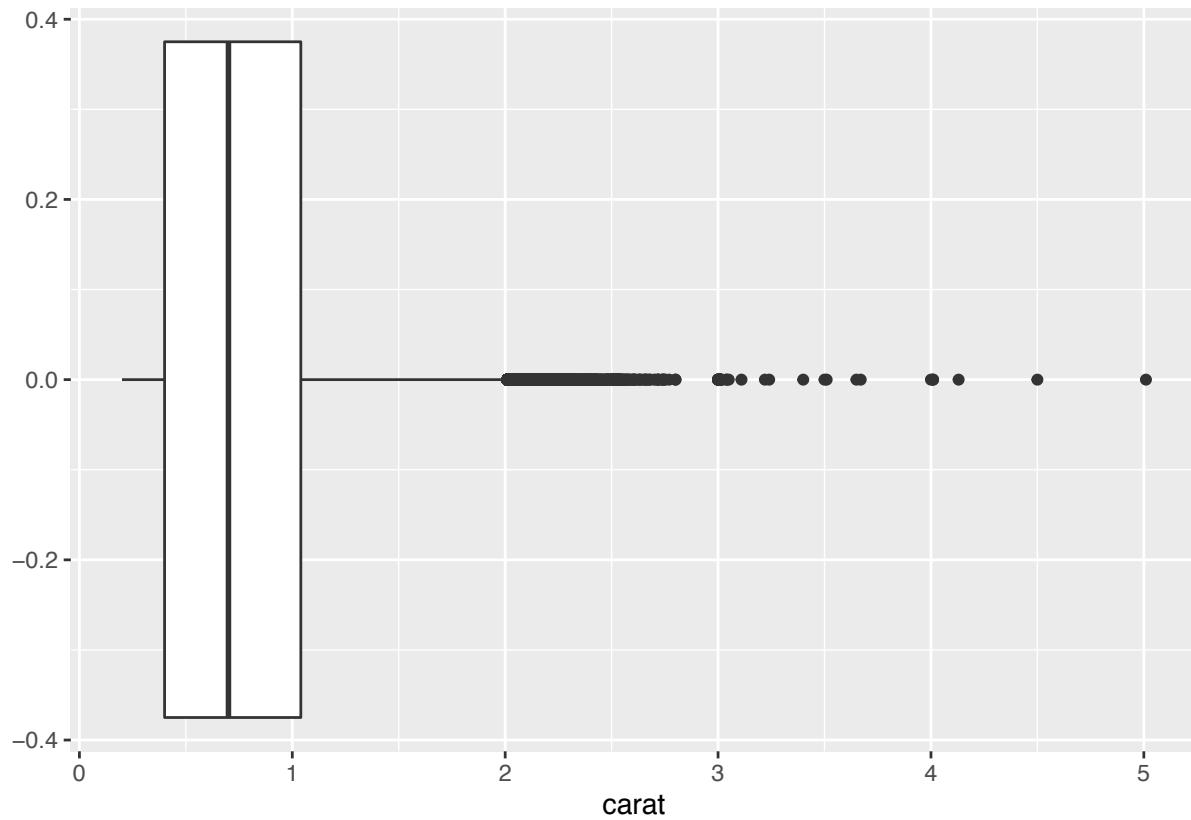


geom_boxplot

Very useful for diagnosing issues with data, and for having a quick glance at the data

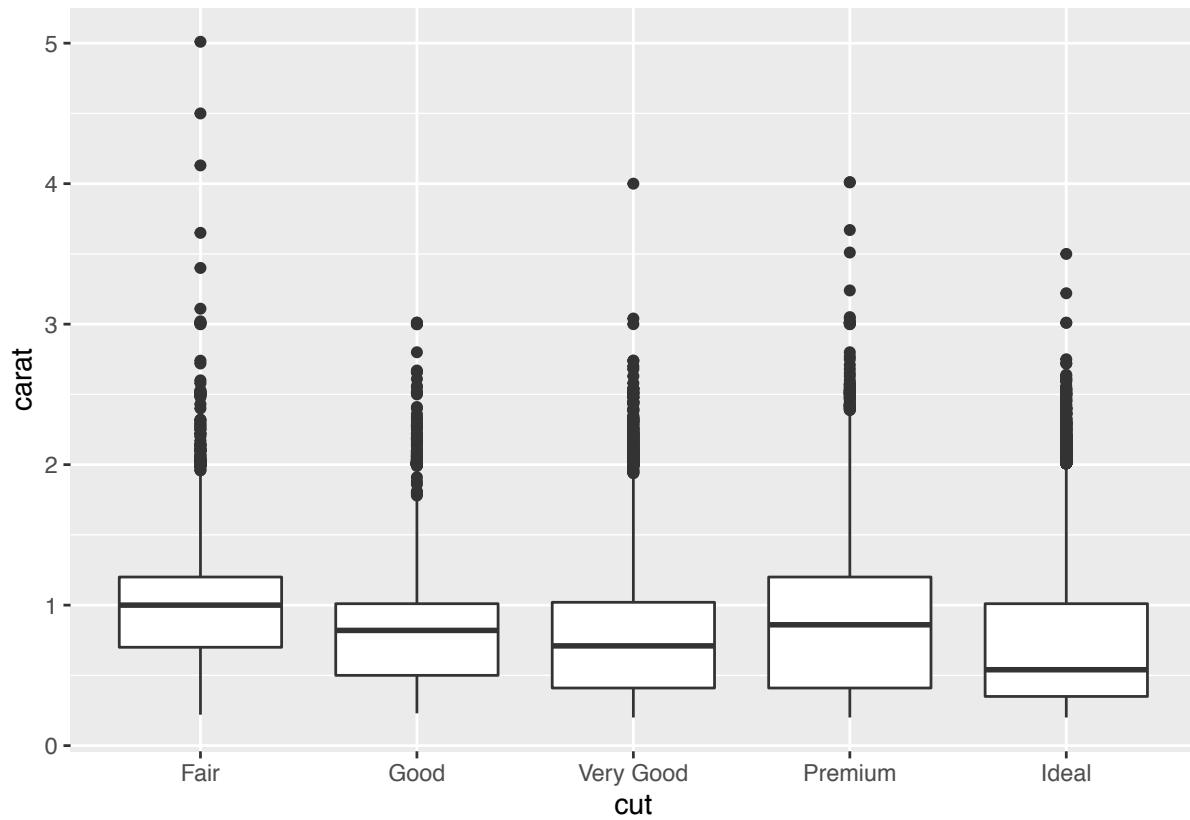
```
ggplot(diamonds) + geom_boxplot(aes(y = carat))
```





```
#### a boxplot for each level of cut
```

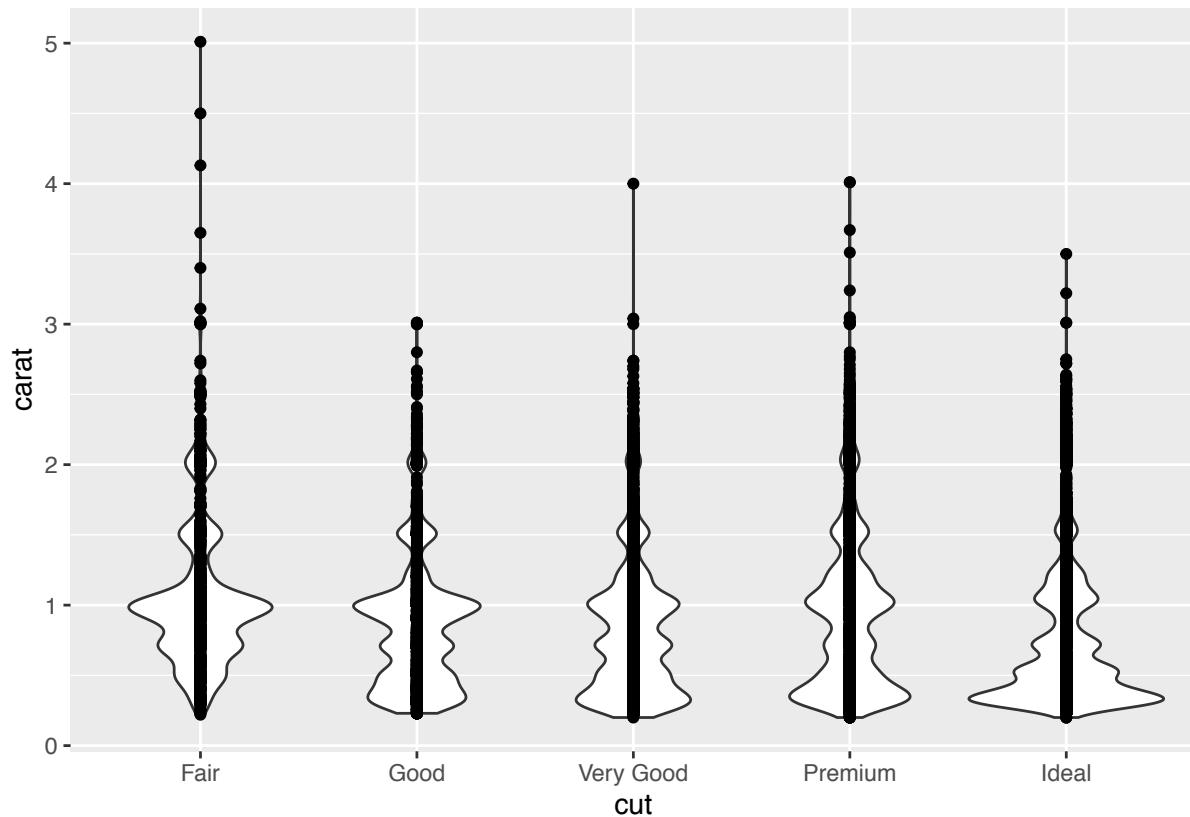
```
ggplot(diamonds) + geom_boxplot(aes(x = cut, y = carat))
```



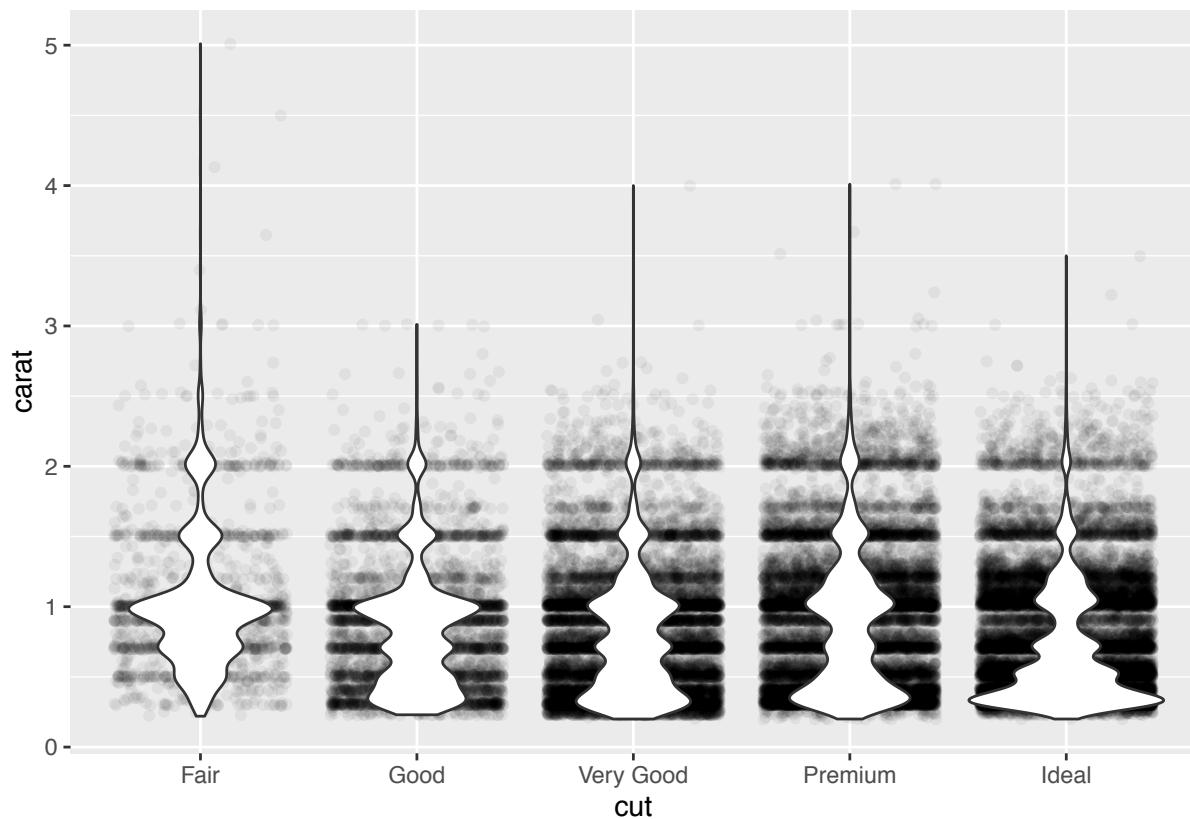
geom_violin

Same as the boxplot, but in this case, the distribution of the datapoints is displayed better

```
ggplot(diamonds) + geom_violin(aes(x = cut, y = carat)) + geom_point(aes(x = cut, y = carat))
```



```
ggplot(diamonds) + geom_point(aes(x = cut, y = carat), position="jitter", alpha=0.05)+ geom_violin(aes(
```



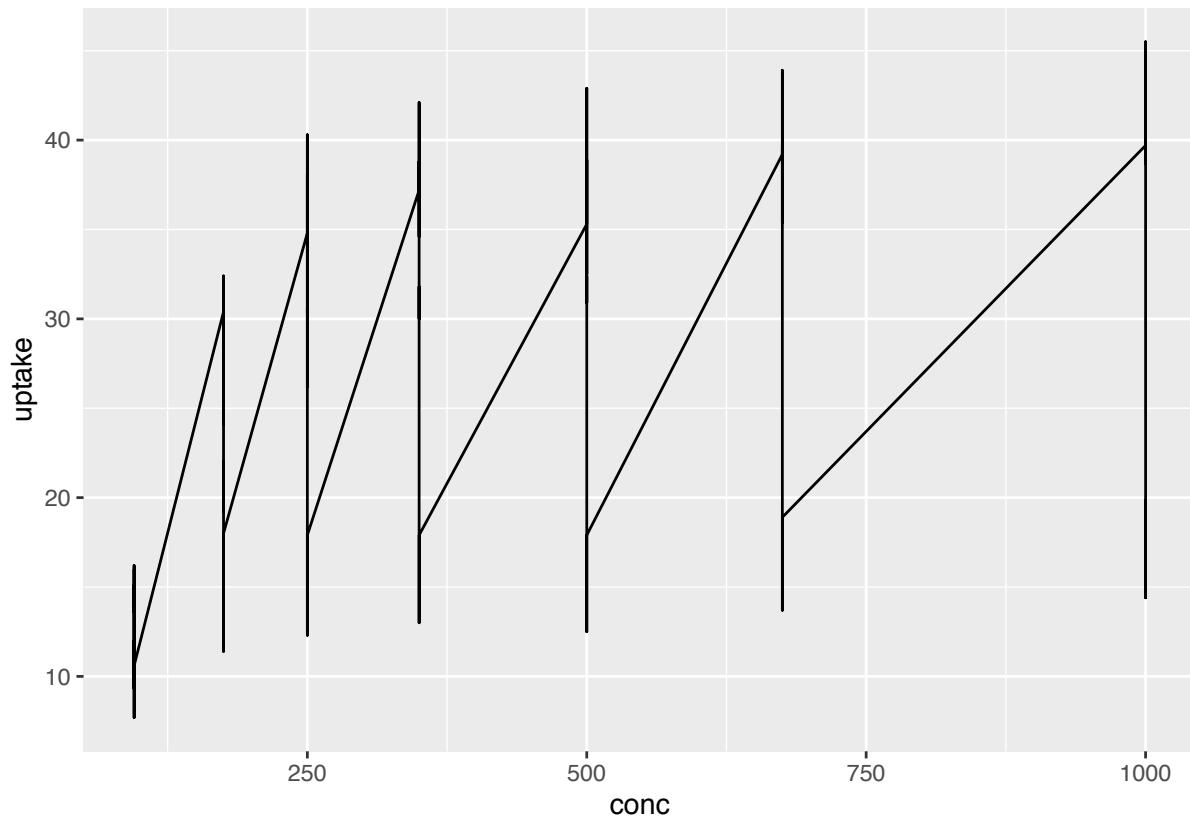
geom_line

```
head(CO2, 3)
```

```
##   Plant    Type Treatment conc uptake
## 1  Qn1 Quebec nonchilled   95   16.0
## 2  Qn1 Quebec nonchilled  175   30.4
## 3  Qn1 Quebec nonchilled  250   34.8
```

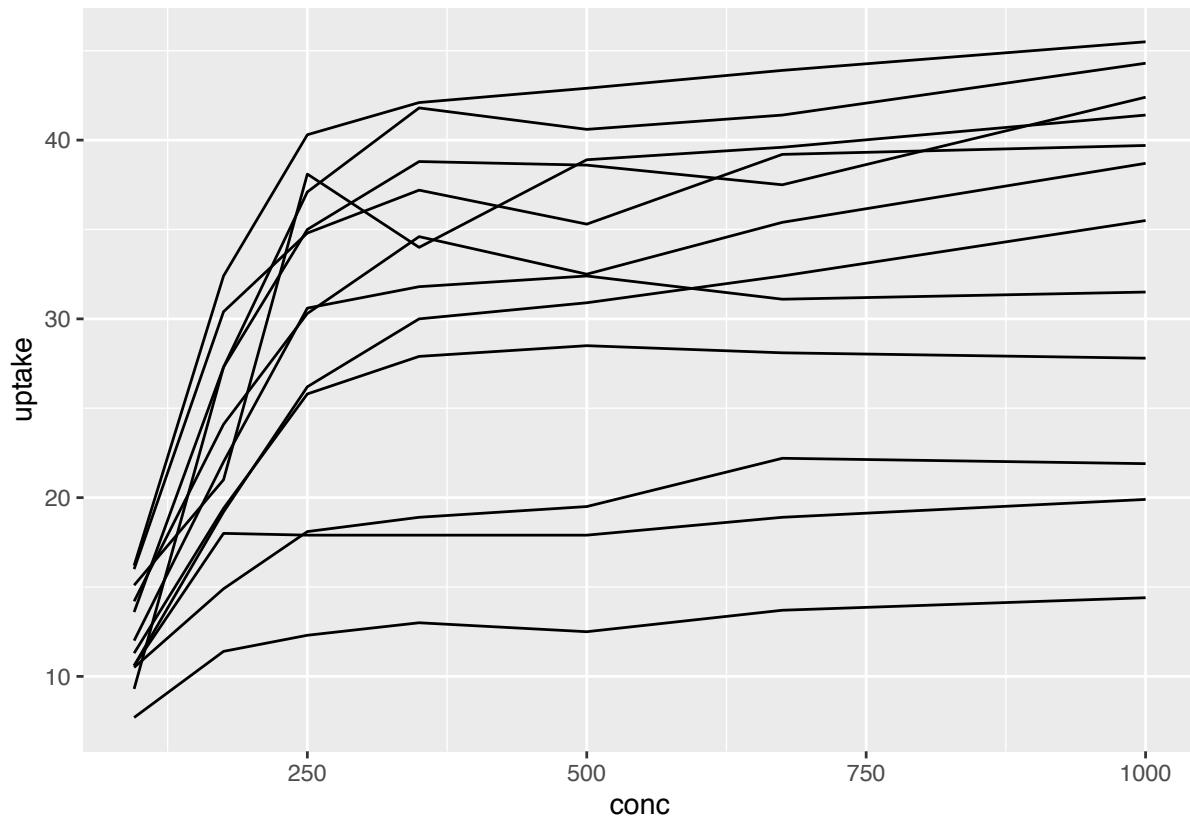
Default geom_line

```
ggplot(CO2) + geom_line(aes(x = conc, y = uptake))
```



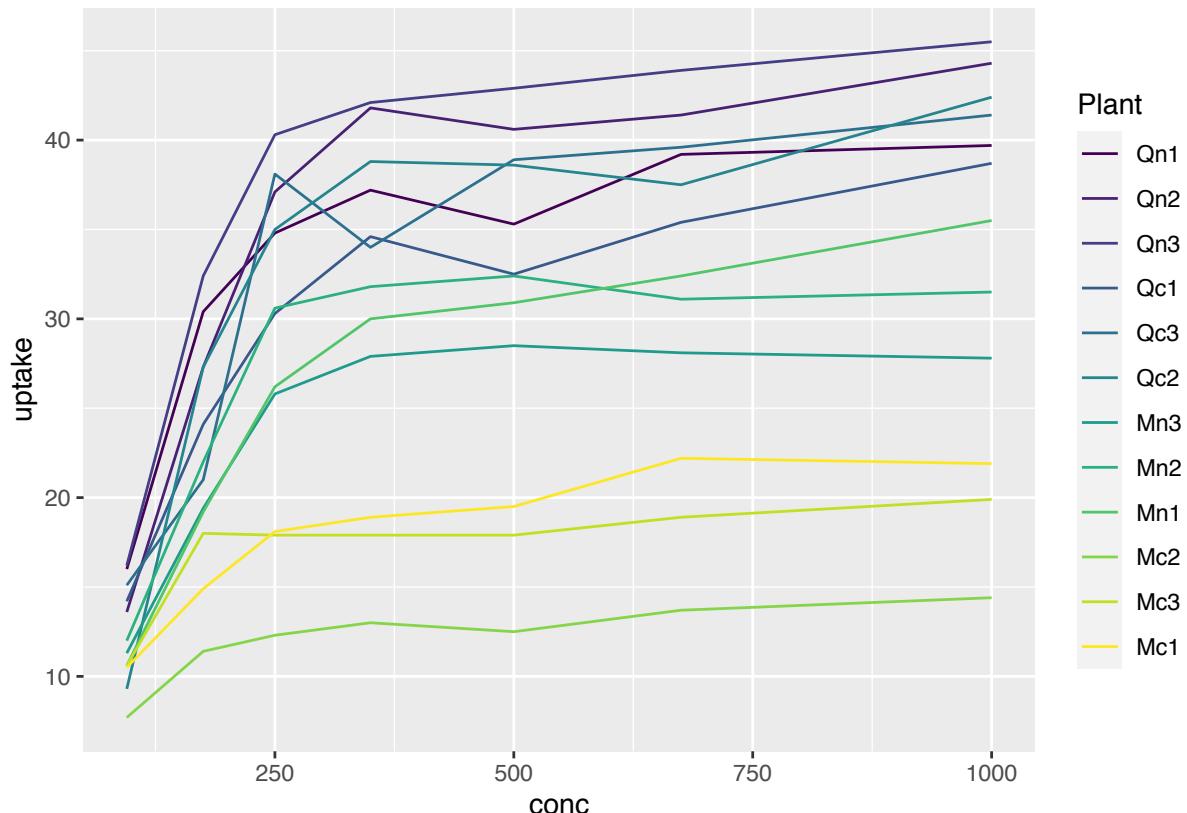
Here, R joined a line across all the rows(plants). We would have to tell R to draw a line for each plant, to GROUP according to the plant.

```
ggplot(CO2) + geom_line(aes(x = conc, y = uptake, group=Plant))
```



Now we get a line for each plant. But what if we want to know what plant is what?

```
ggplot(CO2) + geom_line(aes(x = conc, y = uptake, color=Plant)) # color=Plant, instead of group=Plant.
```



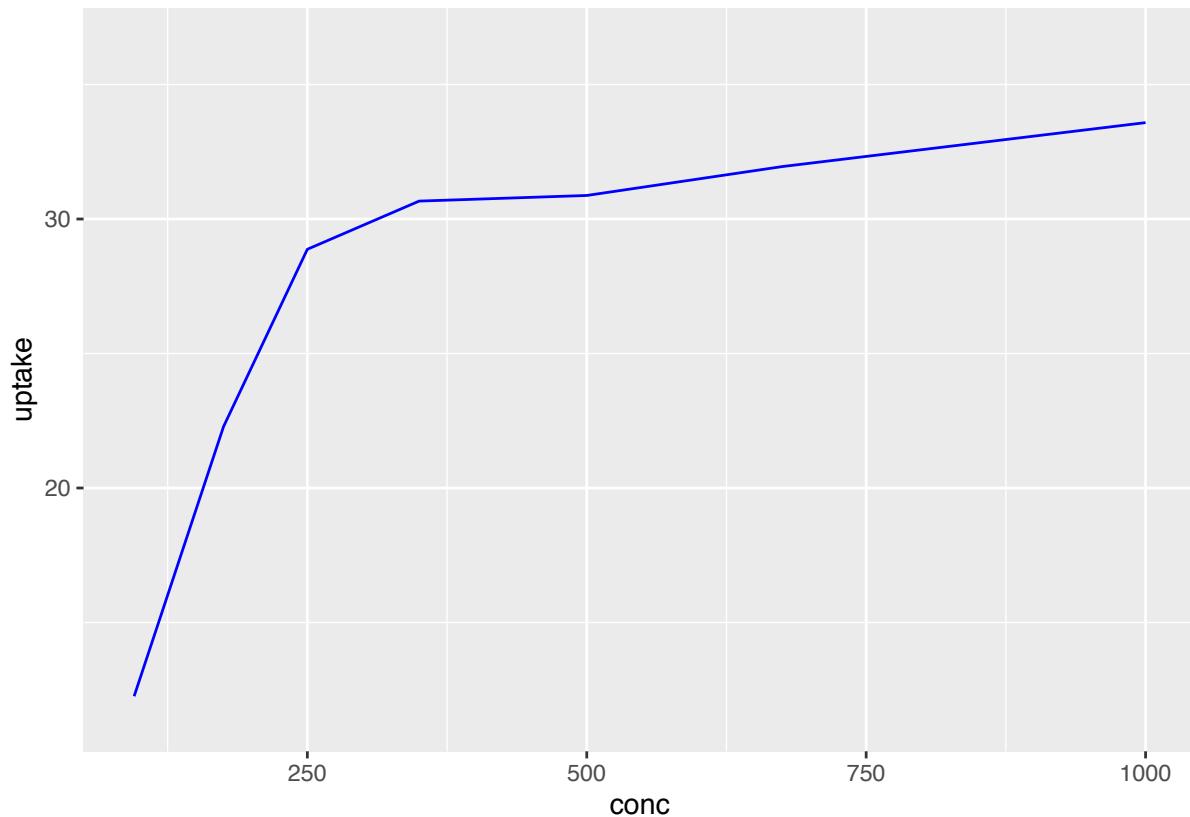
```
## geom_point vs geom_line
```

geom_line

Let's make the dots different, distinguishing btw point and line, which is often confusing

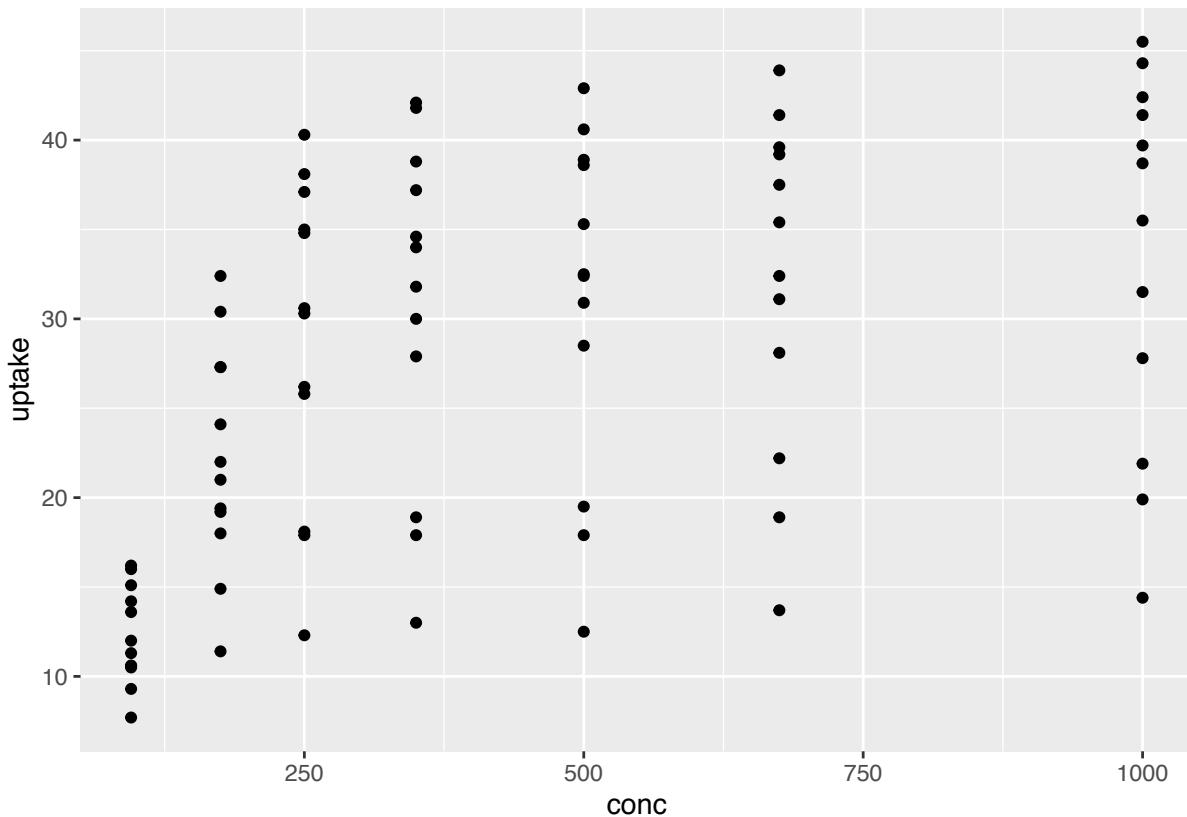
```
ggplot(CO2) + geom_line(aes(x = conc, y = uptake),
  stat = "summary", fun.y = mean, color='blue')
```

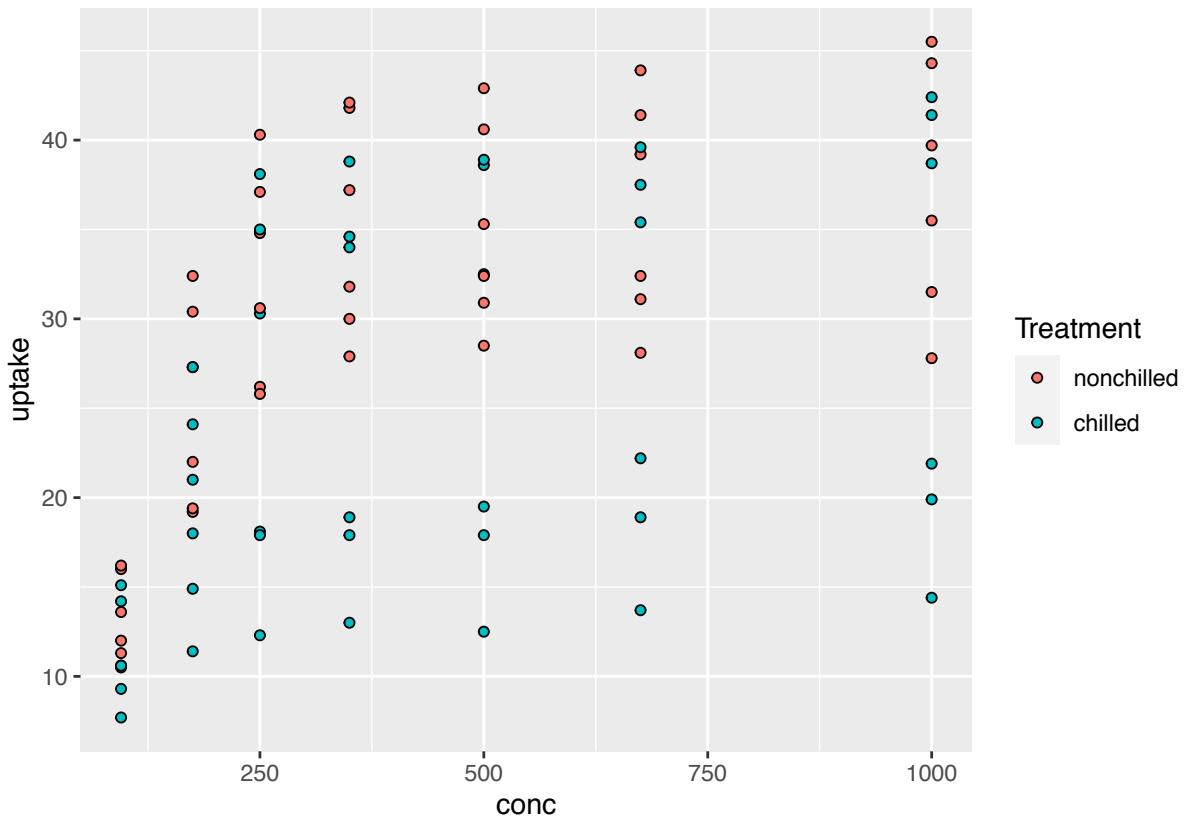
```
## Warning: Ignoring unknown parameters: fun.y
## No summary function supplied, defaulting to `mean_se()`
```



geom_point

```
ggplot(CO2) + geom_point(aes(x = conc, y = uptake))
```





```
## geom_smooth
```

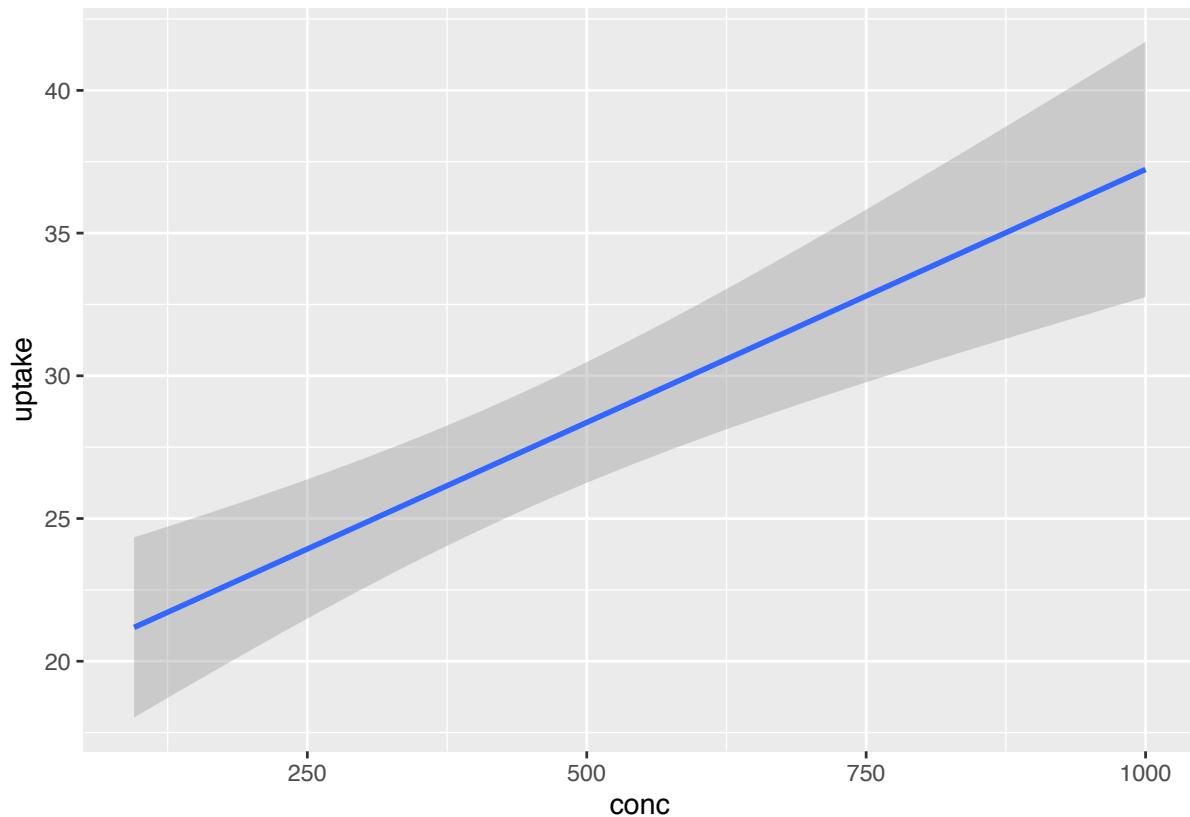
```
head(CO2,3)
```

```
##   Plant Type Treatment conc uptake
## 1 Qn1 Quebec nonchilled  95  16.0
## 2 Qn1 Quebec nonchilled 175  30.4
## 3 Qn1 Quebec nonchilled 250  34.8
```

The default smoother is the lower smoother. A lower smoother is a locally weighted regression smoother, so instead of fitting one regression line it is fitting lots of little ones. In this case though, we used the linear regression line lm.

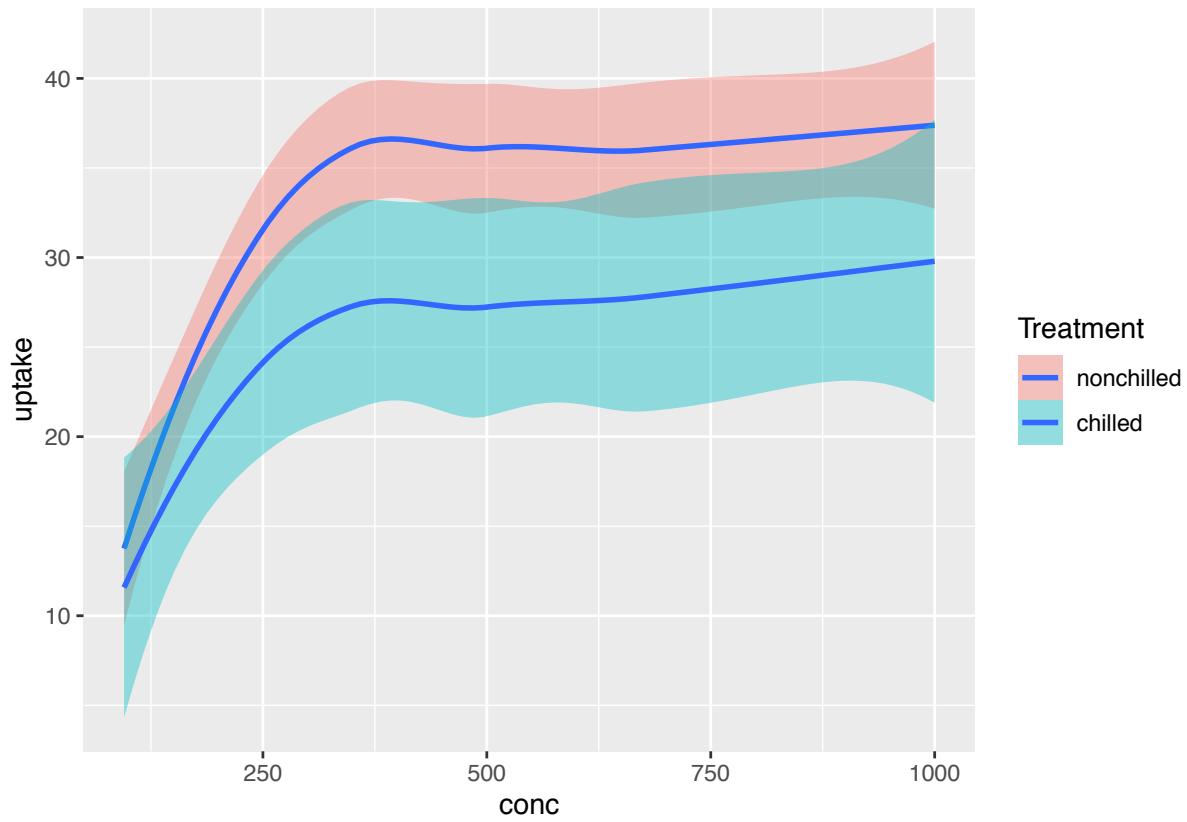
```
ggplot(CO2) + geom_smooth(aes(x = conc, y = uptake), method='lm')
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
ggplot(CO2) + geom_smooth(aes(x = conc, y = uptake, fill=Treatment)) # we created a smoother for each treatment
```

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

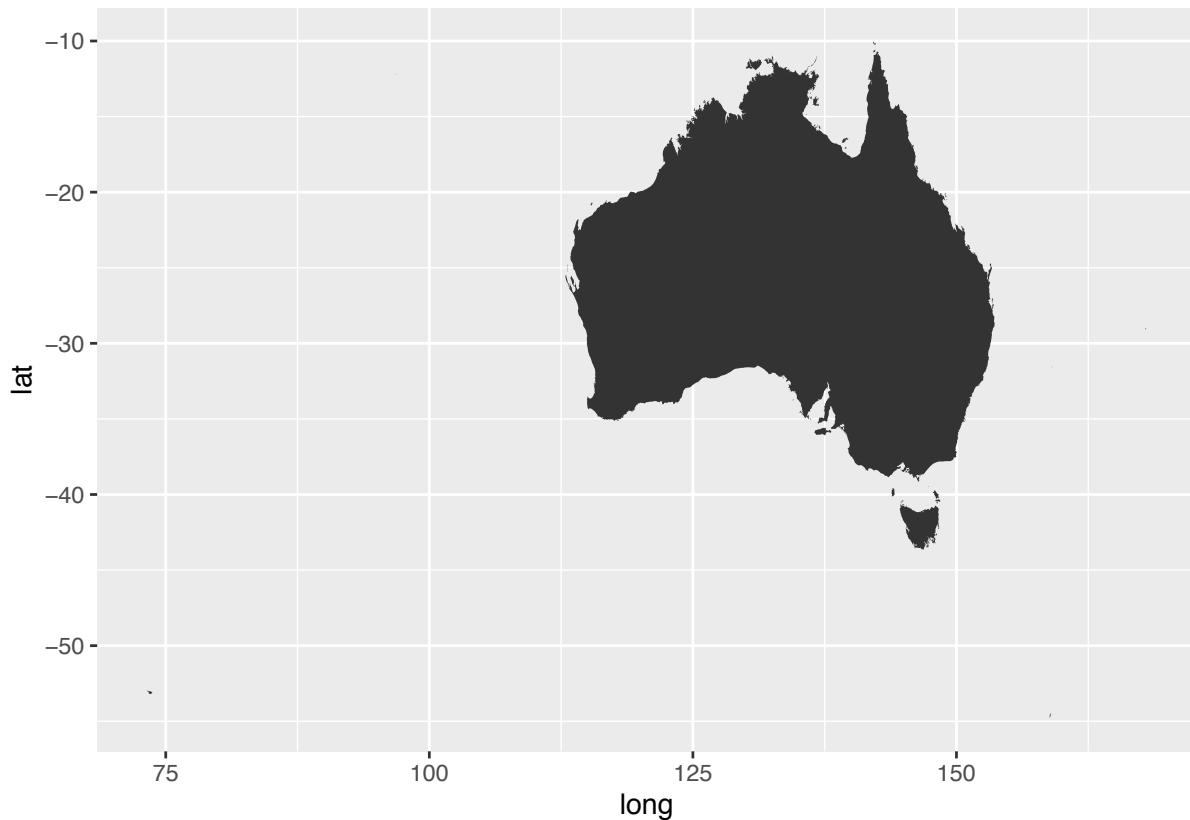


geom_polygon Don't bother about this, as it is quite outdated and GAMs provide a better way for mapping

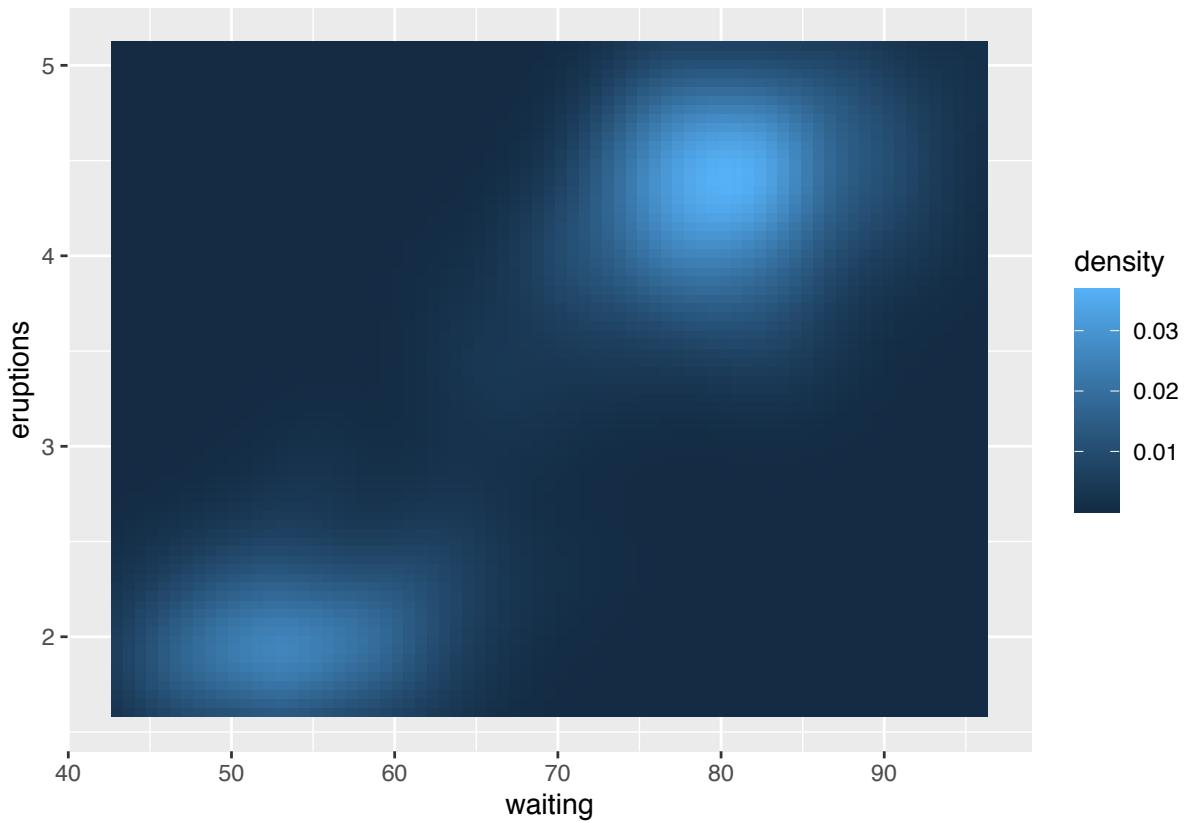
```
library(maps)
library(mapdata)
aus <- map_data("worldHires", region="Australia")
head(aus,3)

##      long      lat group order      region           subregion
## 1 142.1461 -10.74943     1     1 Australia Prince of Wales Island
## 2 142.1430 -10.74525     1     2 Australia Prince of Wales Island
## 3 142.1406 -10.74113     1     3 Australia Prince of Wales Island

ggplot(aus, aes(x=long, y=lat, group=group)) +
  geom_polygon()
```



```
## geom_tile  
head(faithfuld,3)  
  
## # A tibble: 3 x 3  
##   eruptions waiting density  
##       <dbl>    <dbl>    <dbl>  
## 1      1.6     43  0.00322  
## 2      1.65    43  0.00384  
## 3      1.69    43  0.00444  
ggplot(faithfuld, aes(waiting, eruptions)) +  
  geom_tile(aes(fill = density))
```

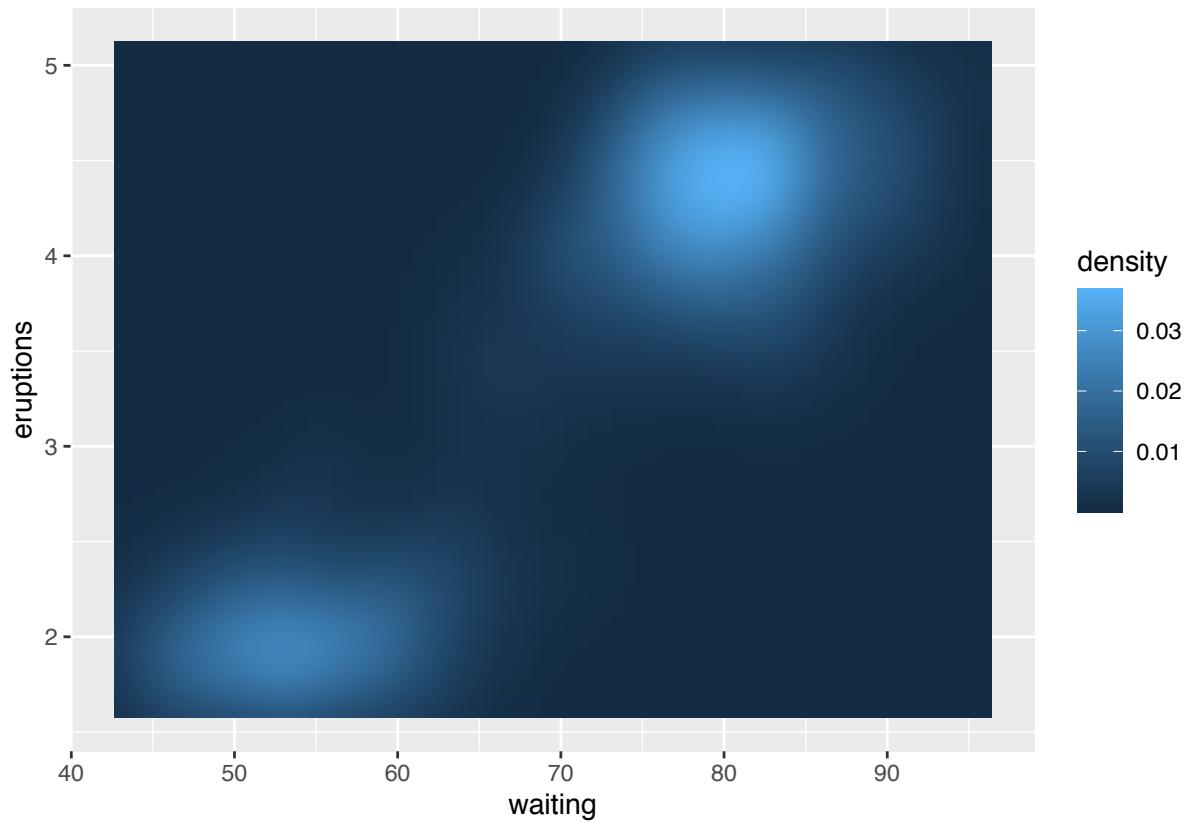


geom_raster In comparison to geom_tile, geom_raster can interpolate, filling in the gaps = will fill in the gaps if we don't have the full matrix

```
head(faithfuld,3)

## # A tibble: 3 x 3
##   eruptions waiting density
##       <dbl>     <dbl>    <dbl>
## 1      1.6      43  0.00322
## 2      1.65     43  0.00384
## 3      1.69     43  0.00444

ggplot(faithfuld, aes(waiting, eruptions)) +
  geom_raster(aes(fill = density))
```



SECONDARY GEOMETRIC OBJECTS

Example dataset

```
head(warpbreaks)
```

```
##   breaks wool tension
## 1    26    A      L
## 2    30    A      L
## 3    54    A      L
## 4    25    A      L
## 5    70    A      L
## 6    52    A      L
```

```
summary(warpbreaks)
```

```
##      breaks     wool   tension
## Min.   :10.00   A:27   L:18
## 1st Qu.:18.25   B:27   M:18
## Median :26.00          H:18
## Mean   :28.15
## 3rd Qu.:34.00
## Max.   :70.00
```

geom_errorbar

```

library(dplyr)
library(gmodels)

warpbreaks.sum <- warpbreaks %>%
  group_by(wool) %>% # We group our dataset by the wooltype
  summarise(Mean=mean(breaks), # And then we summarise
            Lower=ci(breaks)[2],
            Upper=ci(breaks)[3])

## Warning in ci.numeric(breaks): No class or unkown class. Using default
## calcuation.

## Warning in ci.numeric(breaks): No class or unkown class. Using default
## calcuation.

## Warning in ci.numeric(breaks): No class or unkown class. Using default
## calcuation.

## Warning in ci.numeric(breaks): No class or unkown class. Using default
## calcuation.

## `summarise()` ungrouping output (override with `.`groups` argument)
warpbreaks.sum

## # A tibble: 2 x 4
##   wool     Mean  Lower  Upper
##   <fct>  <dbl> <dbl> <dbl>
## 1 A        31.0  24.8  37.3
## 2 B        25.3  21.6  28.9

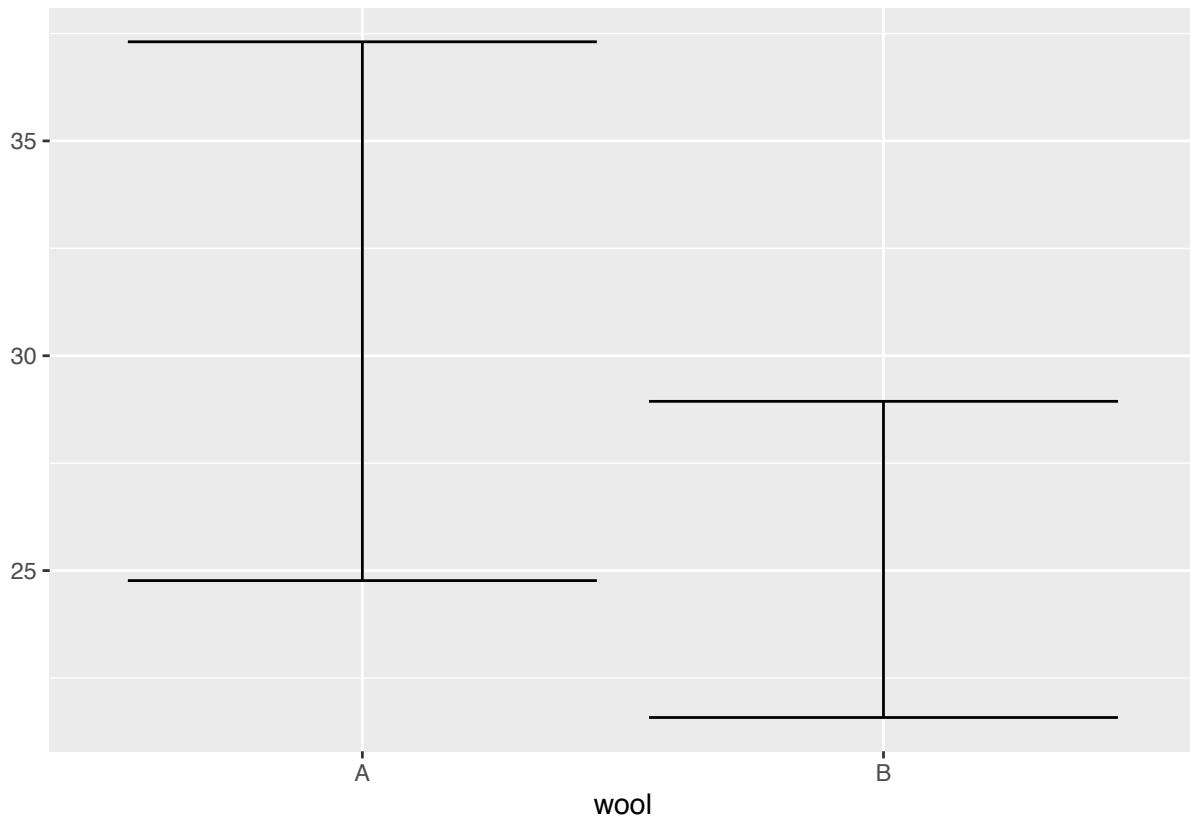
```

We can now plot the summarised statistics (mean, upper and lower CIs -> 1 CI is about 2 and a bit SDs)

```

ggplot(warpbreaks.sum) +
  geom_errorbar(aes(x = wool, ymin = Lower, ymax = Upper))

```



Summarising by SDs (1 SD)

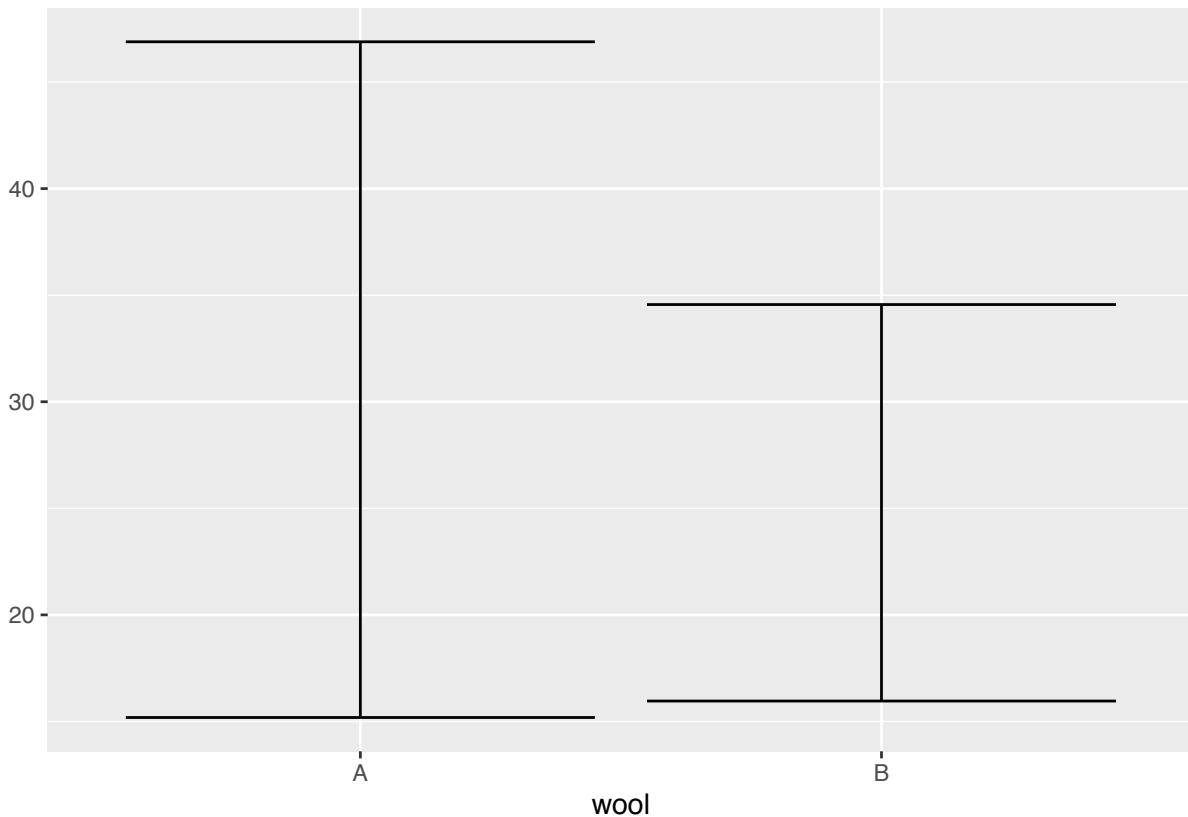
```
warpbreaks.sum.sd <- warpbreaks %>%
  group_by(wool) %>% # We group our dataset by the wooltype
  summarise(Mean=mean(breaks), # And then we summarise
            Lower=Mean-sd(breaks),
            Upper=Mean+sd(breaks))

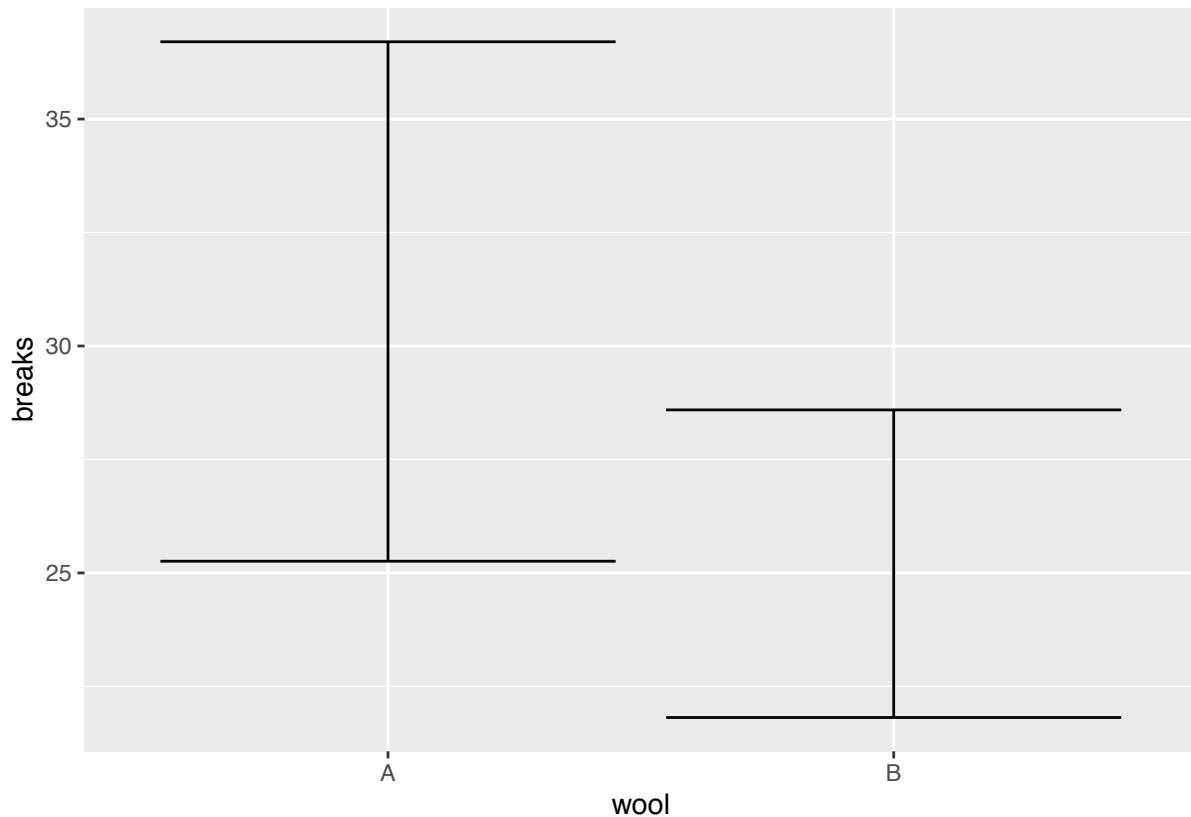
## `summarise()` ungrouping output (override with `.`groups` argument)
warpbreaks.sum.sd
```

```
## # A tibble: 2 x 4
##   wool    Mean Lower Upper
##   <fct> <dbl> <dbl> <dbl>
## 1 A      31.0  15.2  46.9
## 2 B      25.3  16.0  34.6
```

We can now plot the summarised statistics (mean, upper and lower CIs)

```
warpbreaks.sum.sd %>%
  ggplot() +
  geom_errorbar(aes(x = wool, ymin = Lower, ymax = Upper))
```

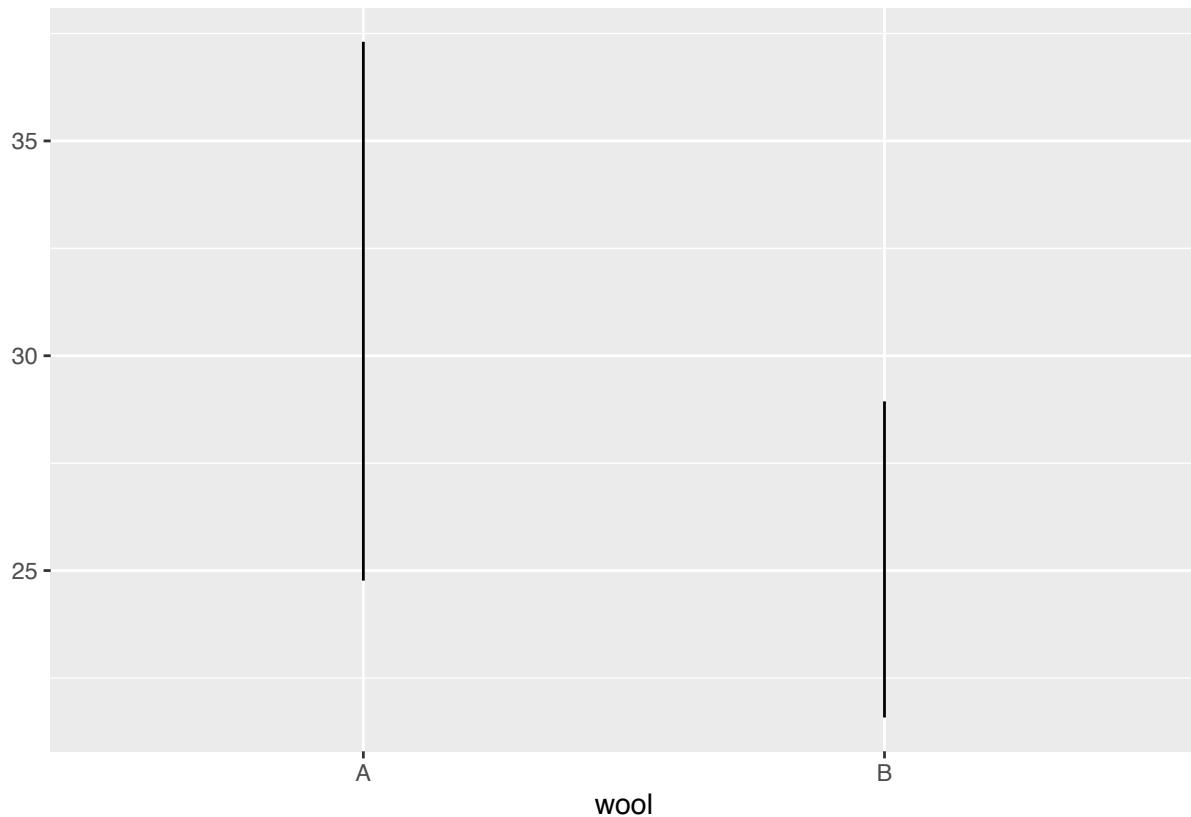




geom_linerange

To get rid of the bits at the top and bottom

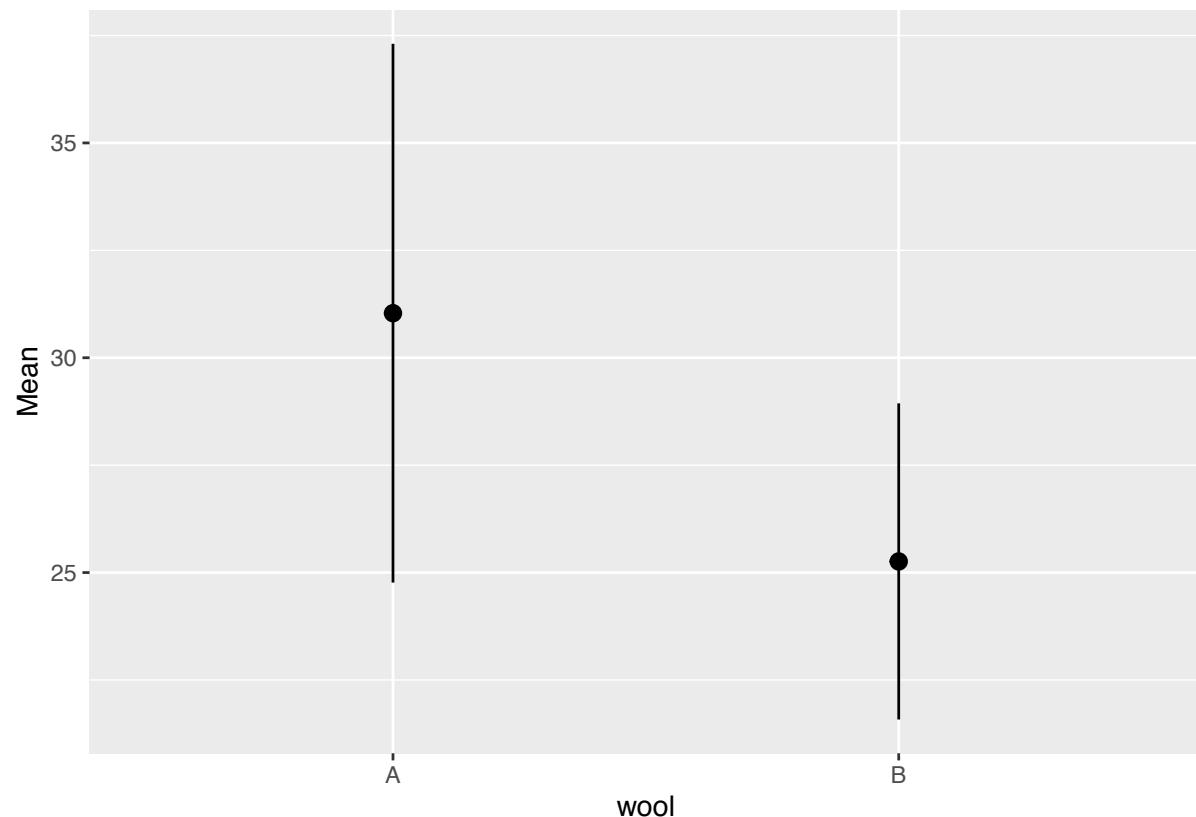
```
ggplot(warpbreaks.sum) +  
  geom_linerange(aes(x = wool, ymin = Lower, ymax = Upper))
```



geom_pointrange

This will display the mean (or median)

```
ggplot(warpbreaks.sum) +  
  geom_pointrange(aes(x = wool, y=Mean, ymin = Lower, ymax = Upper))
```

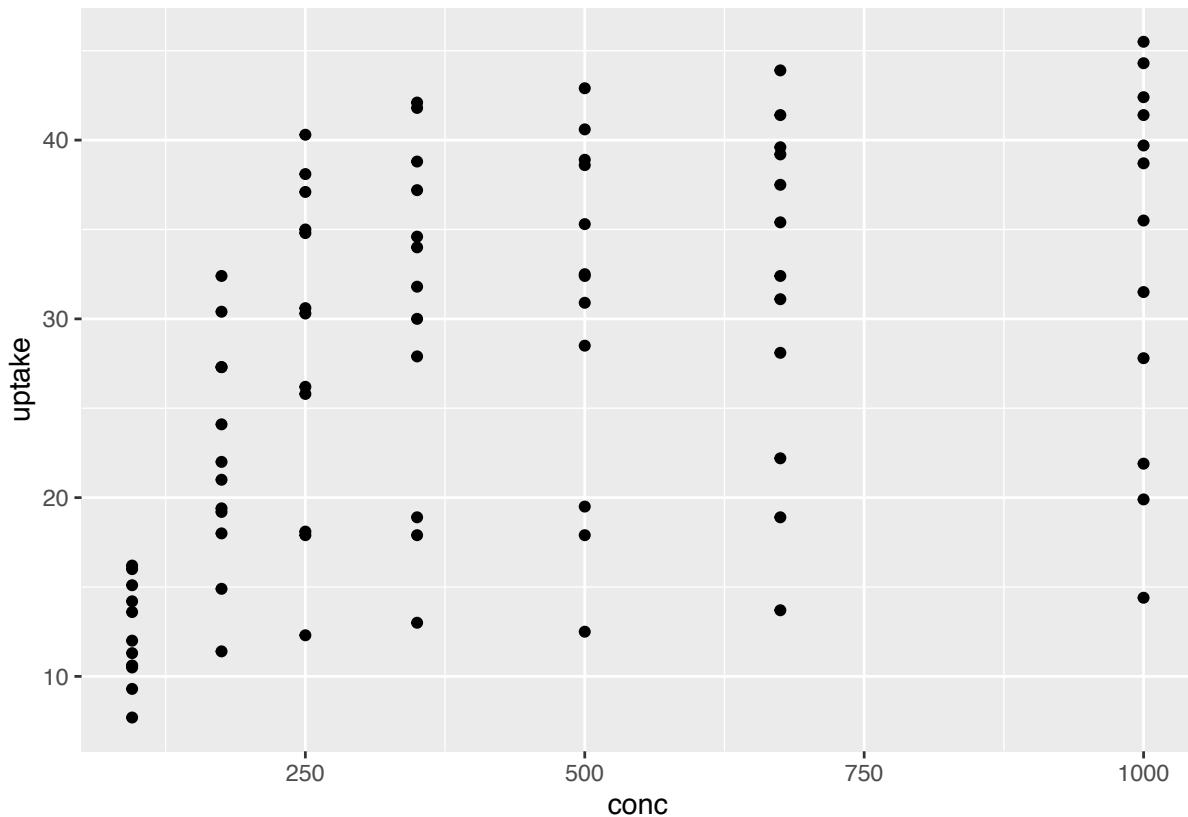


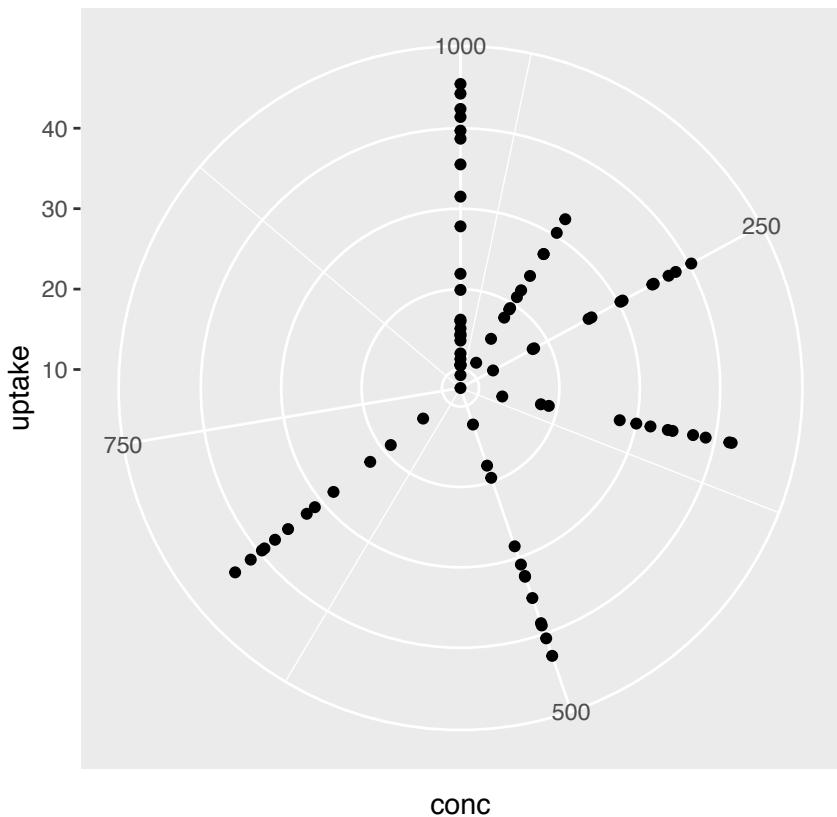
COORDINATE SYSTEMS

```
head(CO2, 3)

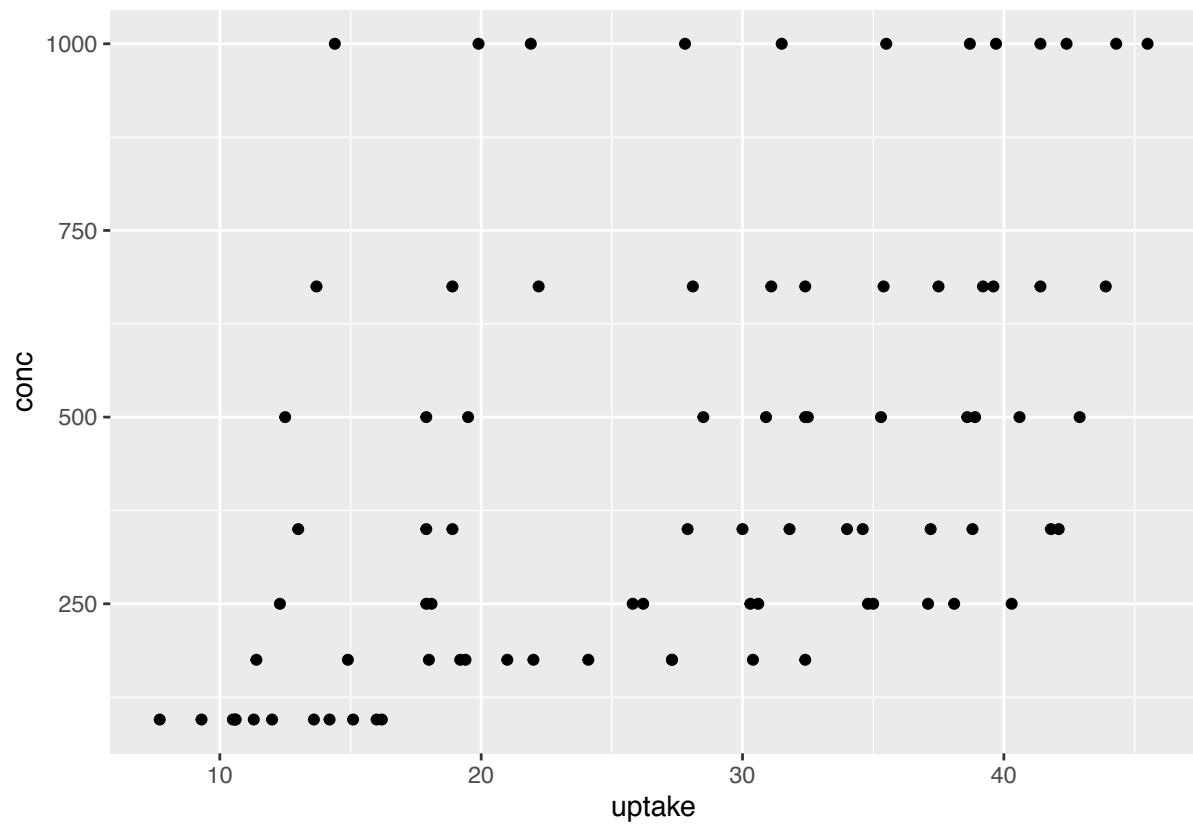
## Grouped Data: uptake ~ conc | Plant
##   Plant Type Treatment conc uptake
## 1  Qn1 Quebec nonchilled  95    16.0
## 2  Qn1 Quebec nonchilled 175    30.4
## 3  Qn1 Quebec nonchilled 250    34.8

ggplot(CO2)+geom_point(aes(x=conc,y=uptake))+
  coord_cartesian() #default
```



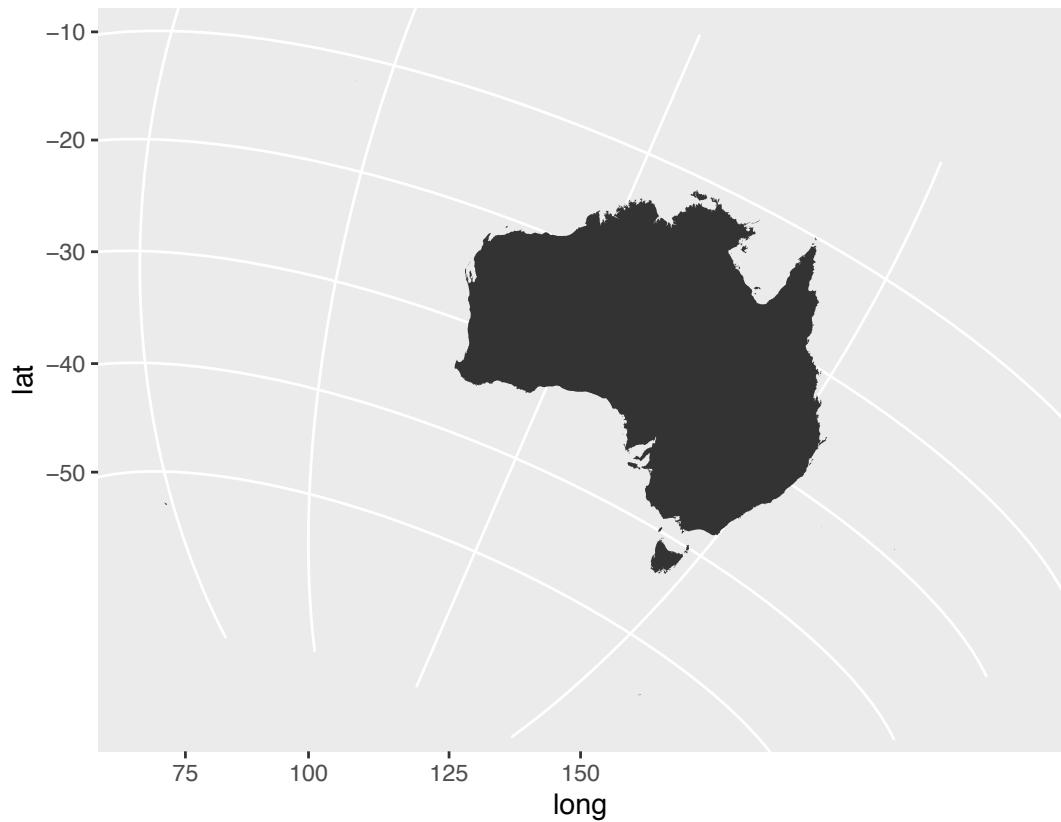


```
ggplot(CO2)+geom_point(aes(x=conc,y=uptake))+  
  coord_flip() # Useful in caterpillar plots, for Bayesian analyses (see image online)
```



```
#Orthographic coordinates
library(maps)
library(mapdata)
library(mapproj)

aus <- map_data("worldHires", region="Australia")
ggplot(aus, aes(x=long, y=lat, group=group)) +
  coord_map("ortho", orientation=c(-20,125,23.5))+
```



SCALES

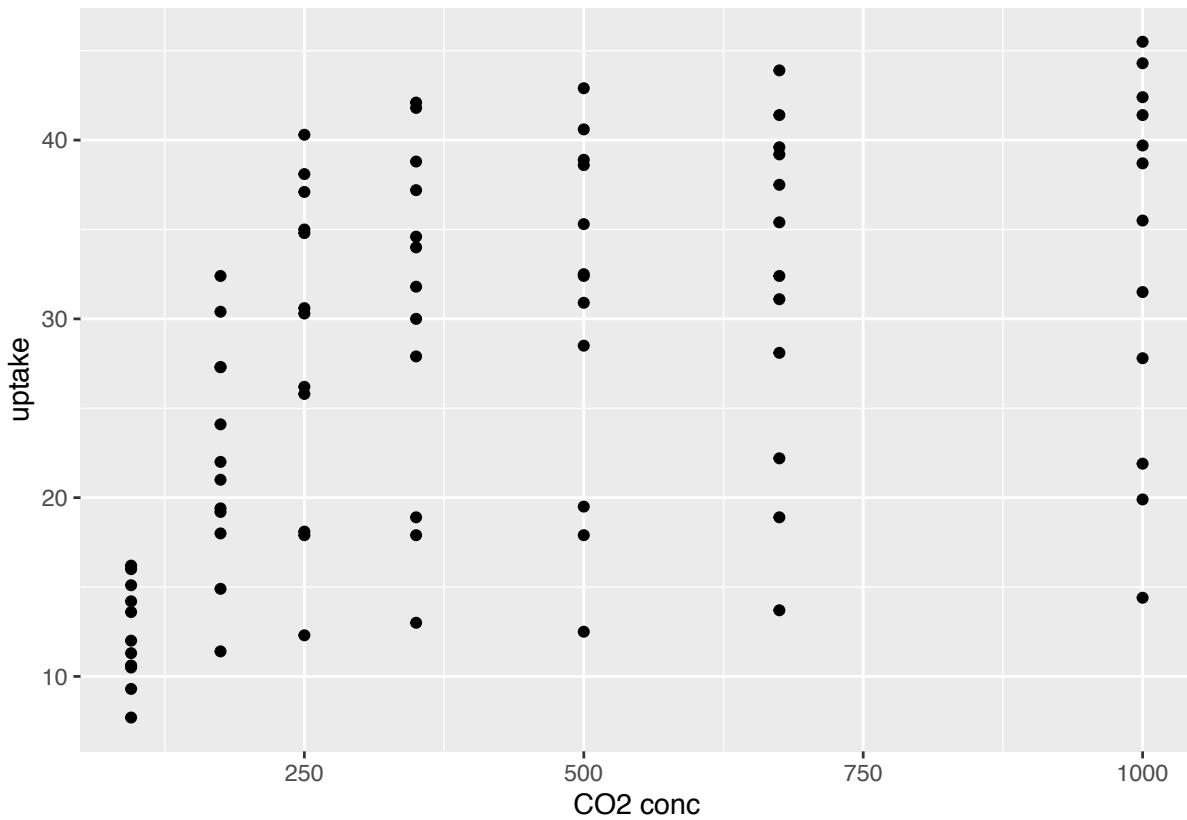
`scale_x_` and `scale_y_`

Axis titles

```
head(CO2, 2)

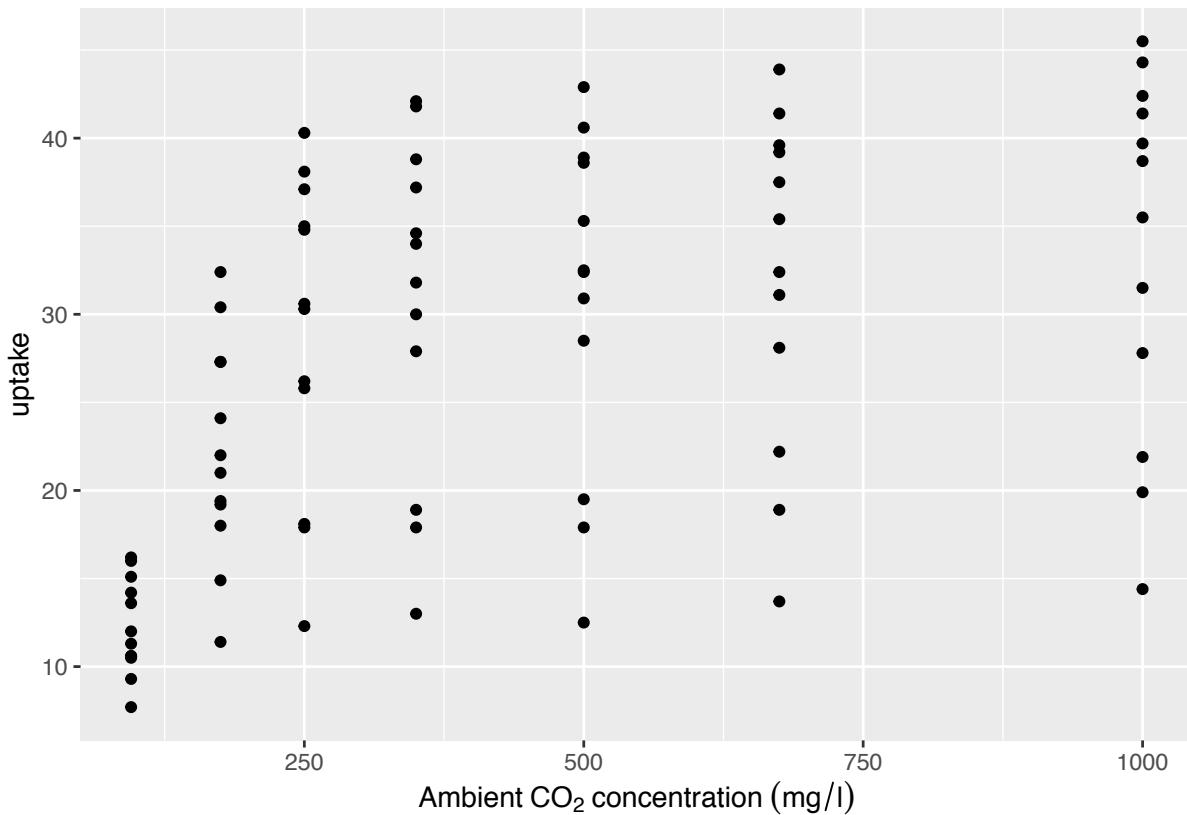
## Grouped Data: uptake ~ conc | Plant
##   Plant Type Treatment conc uptake
## 1 Qn1 Quebec nonchilled  95   16.0
## 2 Qn1 Quebec nonchilled 175   30.4

ggplot(CO2, aes(y=uptake, x=conc)) + geom_point() +
  scale_x_continuous(name="CO2 conc")
```



Special characters (eg. CO₂ with a small 2)

```
ggplot(CO2, aes(y=uptake,x=conc)) + geom_point() +  
  scale_x_continuous(name=expression(  
    Ambient[CO[2]]concentration~(mg/l)))
```

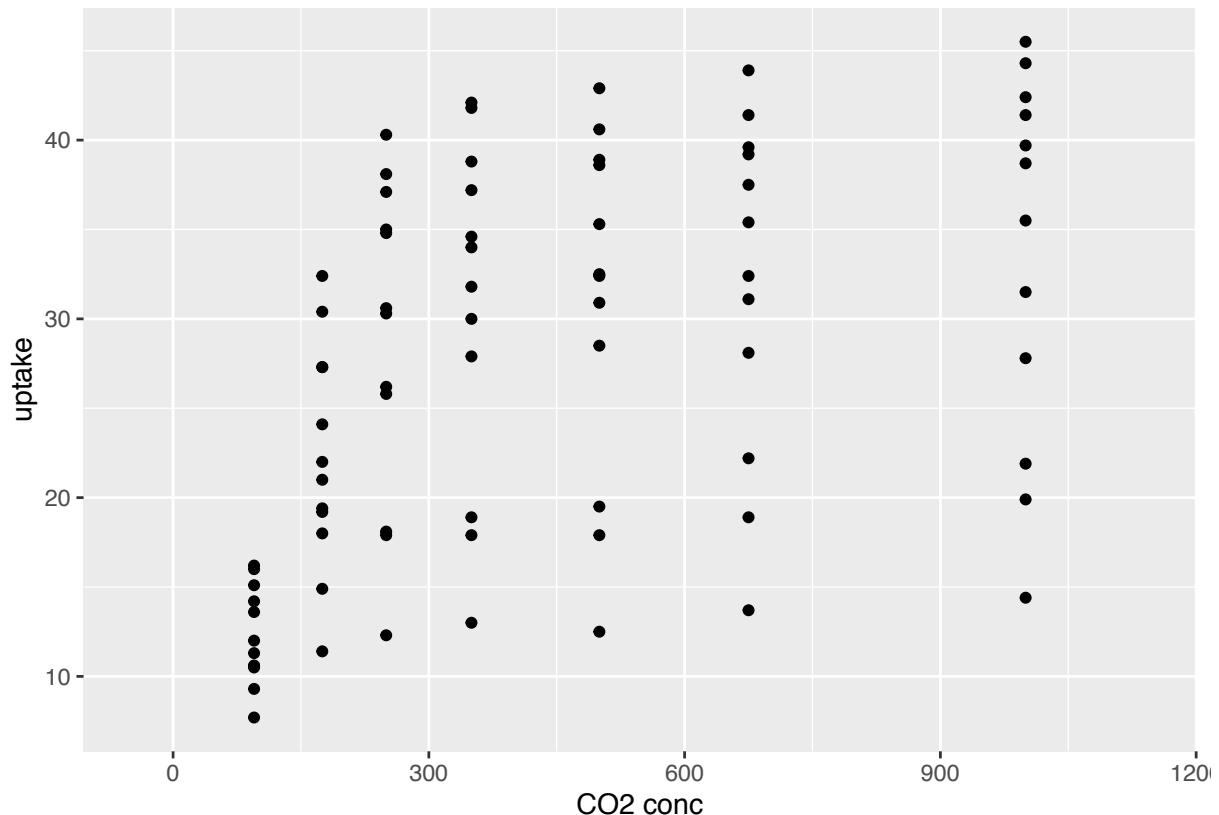


```
# A ~ indicates a space, as special characters cannot have spaces
```

demo(plotmatch) takes you through a set of screens that show you what mathematical operations and special characters you can insert, and how. (!) If you use it, do it in the console. Otherwise, if you KNIT the document, it will block it from running and will prompt you to click RETURN all the time.

Axis more padding

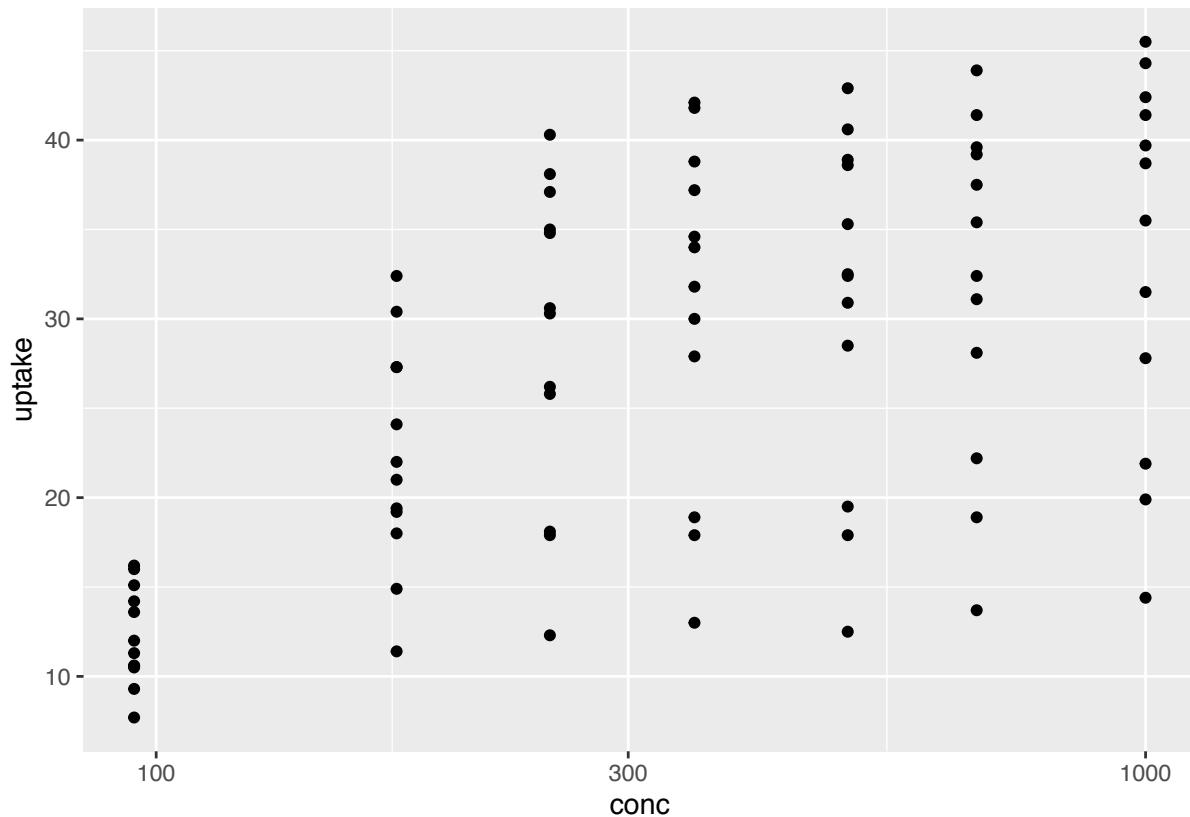
```
ggplot(CO2, aes(y=uptake,x=conc)) + geom_point()+
  scale_x_continuous(name="CO2 conc", expand=c(0,200))
```



We ask R to add 200 units to the x axis at the start and at the end. By default, R will put 4% at the

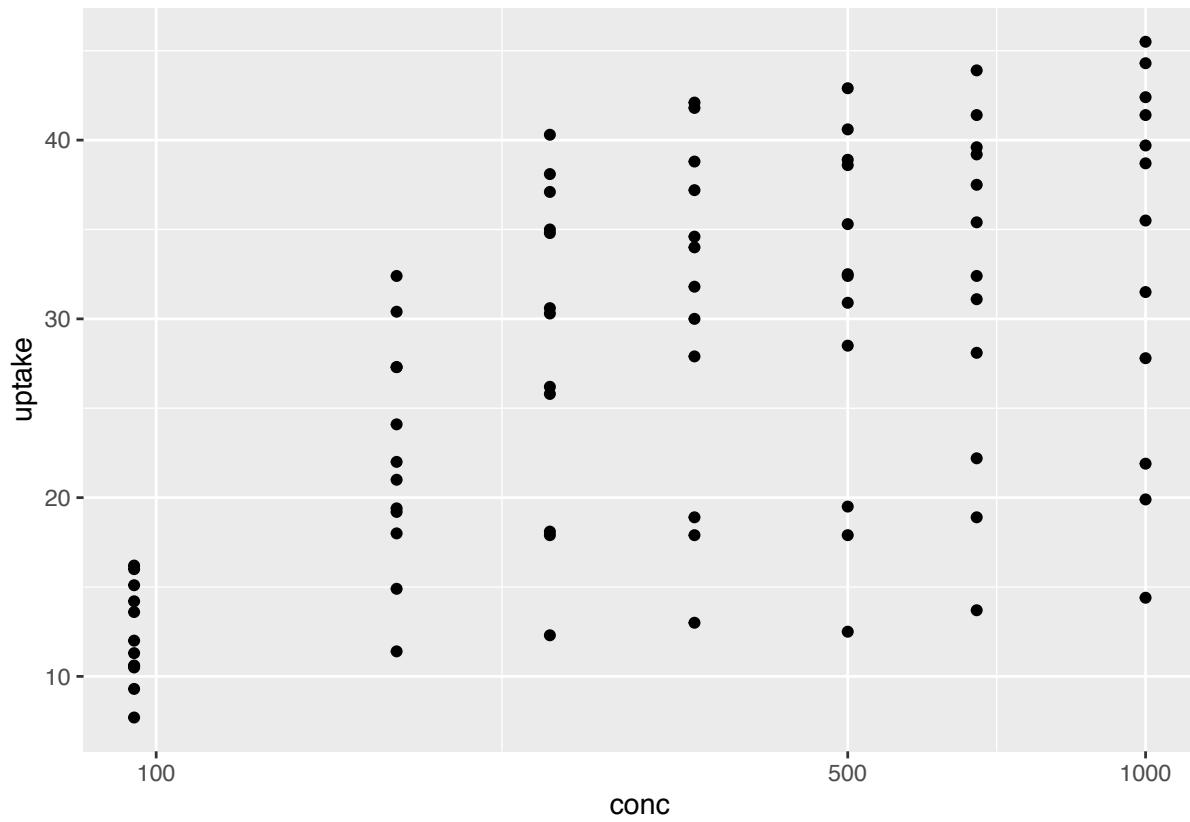
Axis on a log scale

```
ggplot(CO2, aes(y=uptake,x=conc)) + geom_point() +  
  scale_x_log10()
```



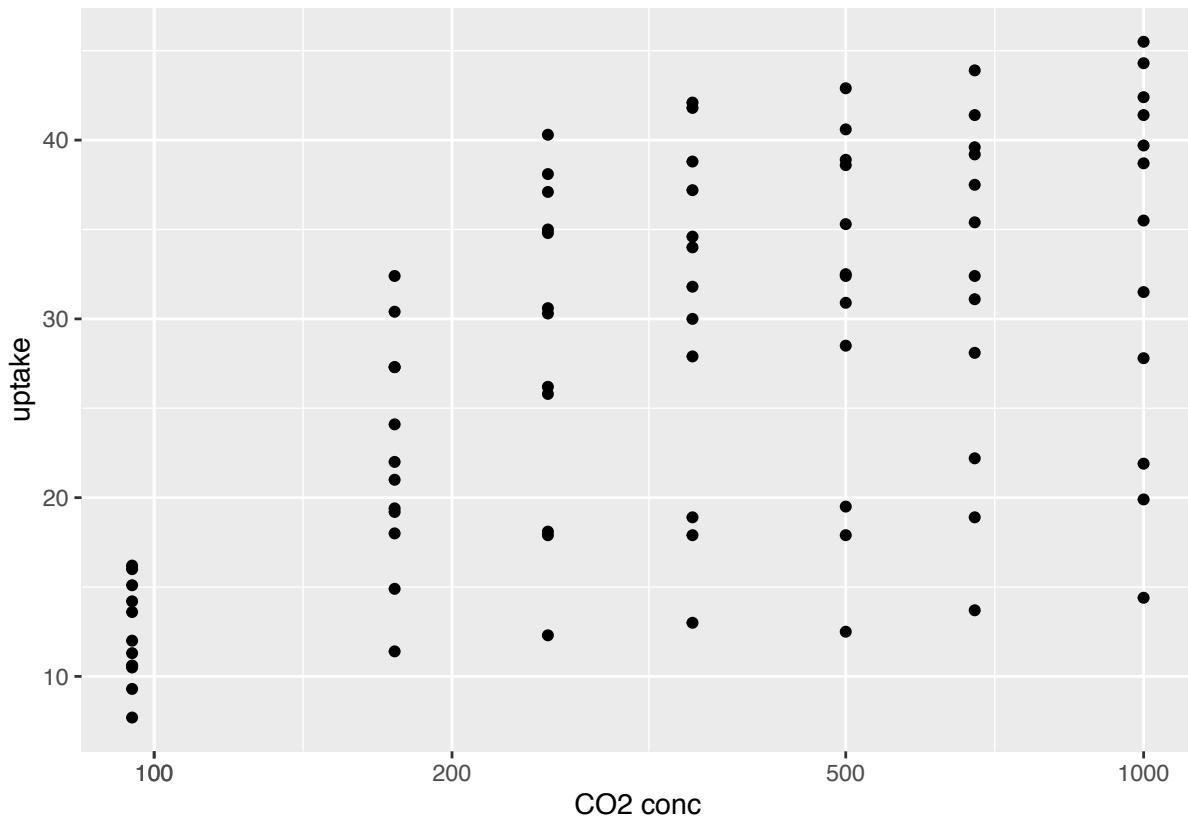
Alters the x scale to a log scale. However, there are not that many tick marks, so it is not very readable. We could tell R where to put tick marks. For example, we could say:

```
ggplot(CO2, aes(y=uptake,x=conc)) + geom_point()+
  scale_x_log10(breaks=c(1,100,500,1000))
```



Alternatively, we can write the following code:

```
ggplot(CO2, aes(y=uptake,x=conc)) + geom_point()+
  scale_x_log10(name="CO2 conc",
  breaks=as.vector(c(1,2,5,10) %o% 10^{(-1:2)}))
```



In this case, we multiply the points by the power of 10 of each. We create a matrix to adapt the x axis to our data. If we display the scale, we would end up with a whole set of numbers that will get tickmarks:

```
c(1,2,5,10) %o% 10^(-1:2)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  0.1   1    10   100
## [2,]  0.2   2    20   200
## [3,]  0.5   5    50   500
## [4,]  1.0  10   100  1000
```

Some points are duplicated, which does not matter. This is useful for axes that start very small and end really large.

We can tweak these numbers (-1:2),(-1:3),(-1:4), etc, until the data points fit well on the scale

```
c(1,2,5,10) %o% 10^(-1:3)
```

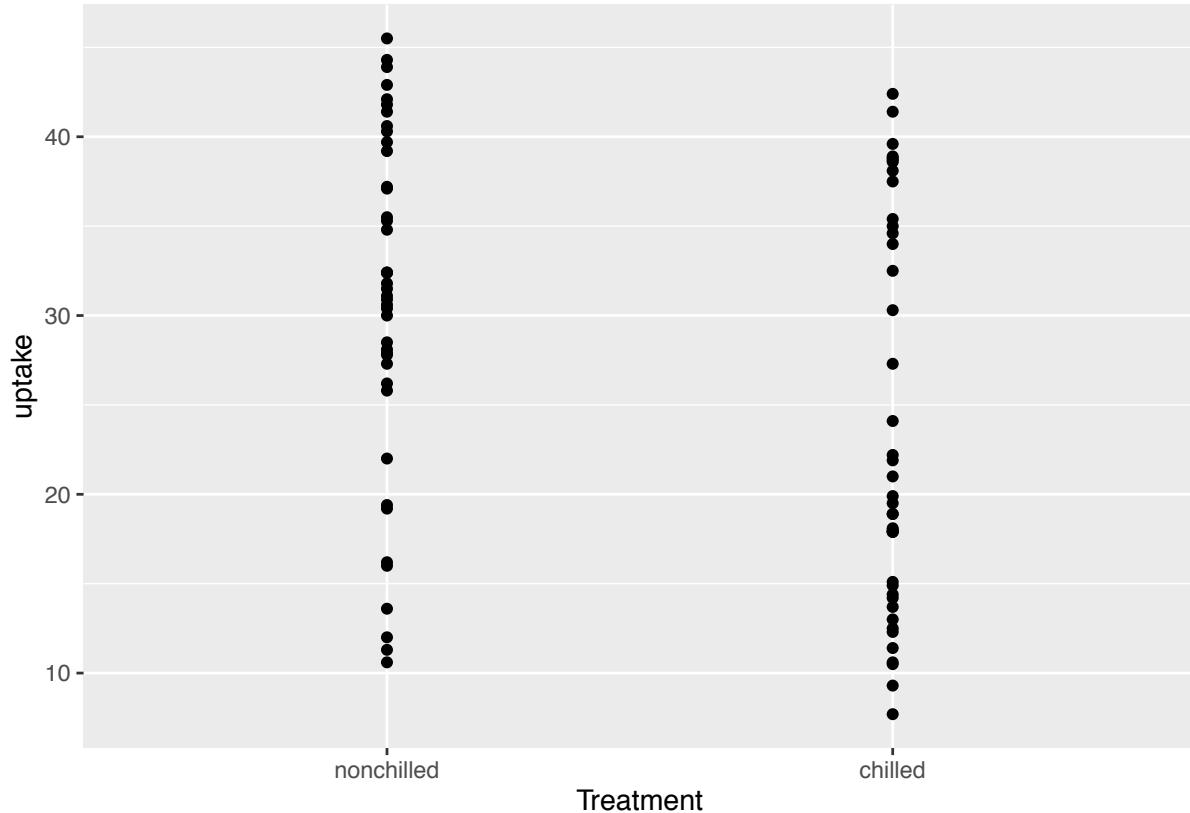
```
##      [,1] [,2] [,3] [,4]   [,5]
## [1,]  0.1   1    10   100  1000
## [2,]  0.2   2    20   200  2000
## [3,]  0.5   5    50   500  5000
## [4,]  1.0  10   100  1000 10000
```

```
c(1,2,5,10) %o% 10^(-1:4)
```

```
##      [,1] [,2] [,3] [,4]   [,5]   [,6]
## [1,]  0.1   1    10   100  1000  1e+04
## [2,]  0.2   2    20   200  2000  2e+04
## [3,]  0.5   5    50   500  5000  5e+04
## [4,]  1.0  10   100  1000 10000 1e+05
```

Axis representing categorical data

```
ggplot(CO2, aes(y=uptake,x=Treatment)) + geom_point()+
  scale_x_discrete(name="Treatment")
```



Other scales

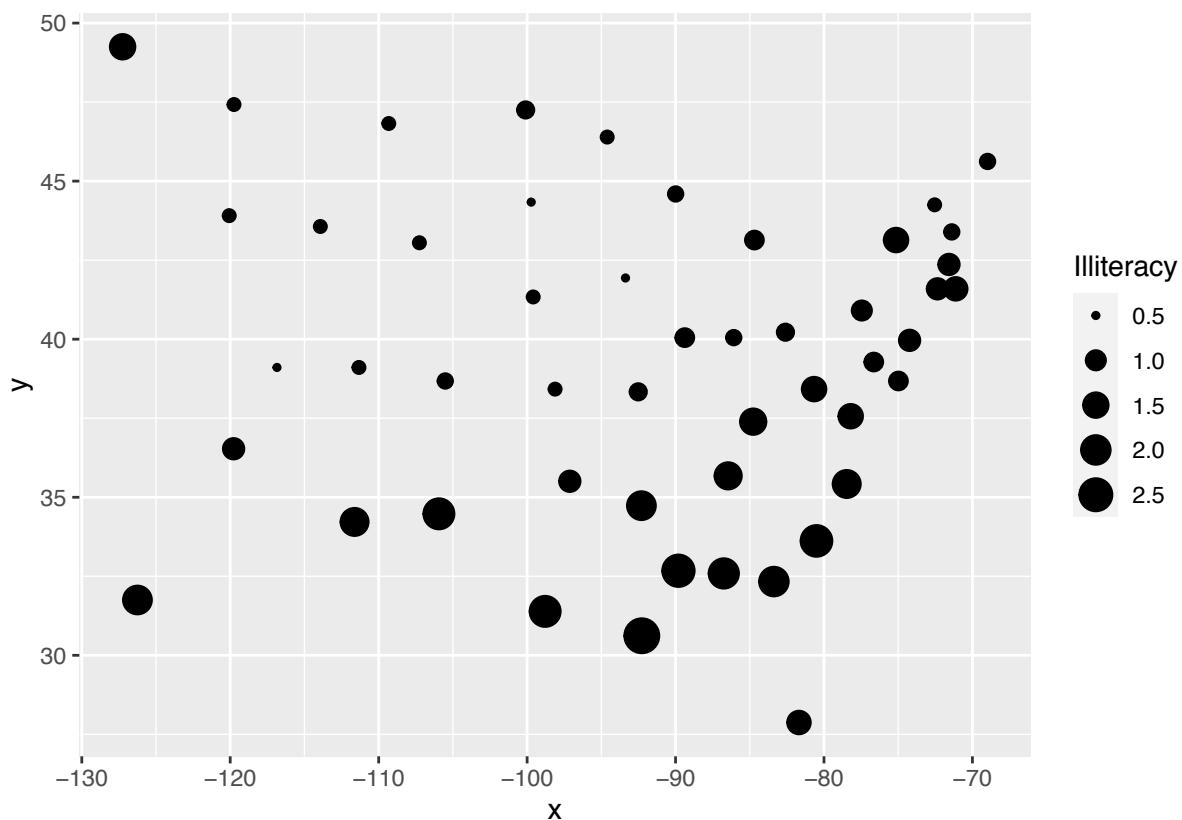
- size of points (thickness of lines)
- shape of points
- linetype of lines
- color of lines or points
- fill of shapes

scale_size

Determines the size according to a continuous variable

```
state=data.frame(state.x77,state.region, state.division,state.center) %>%
  select(Illiteracy,state.region,x,y)
head(state,2)

##           Illiteracy state.region      x      y
## Alabama        2.1       South -86.7509 32.5901
## Alaska         1.5       West -127.2500 49.2500
ggplot(state, aes(y=y,x=x)) + geom_point(aes(size=Illiteracy))
```



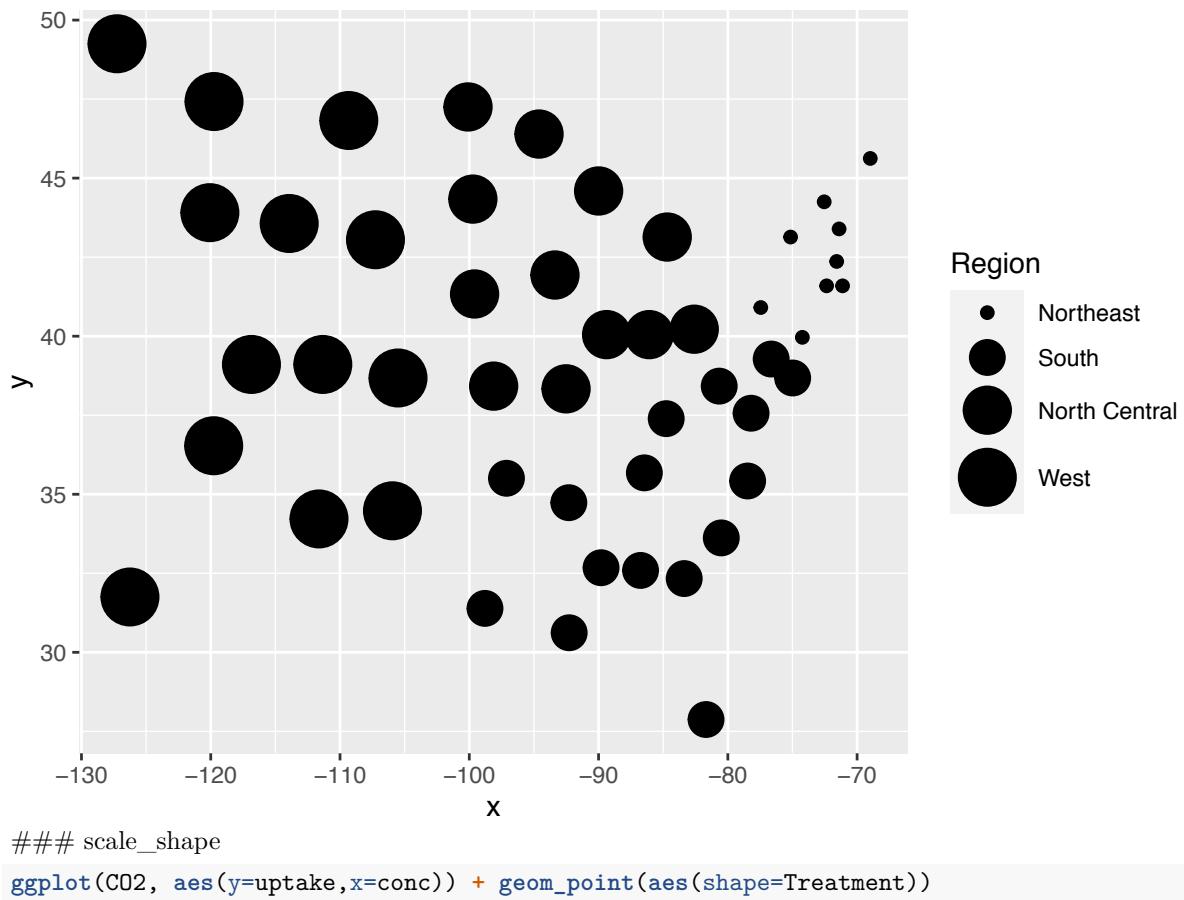
You can also dictate R to create discrete sizes ranging from 2 to 4

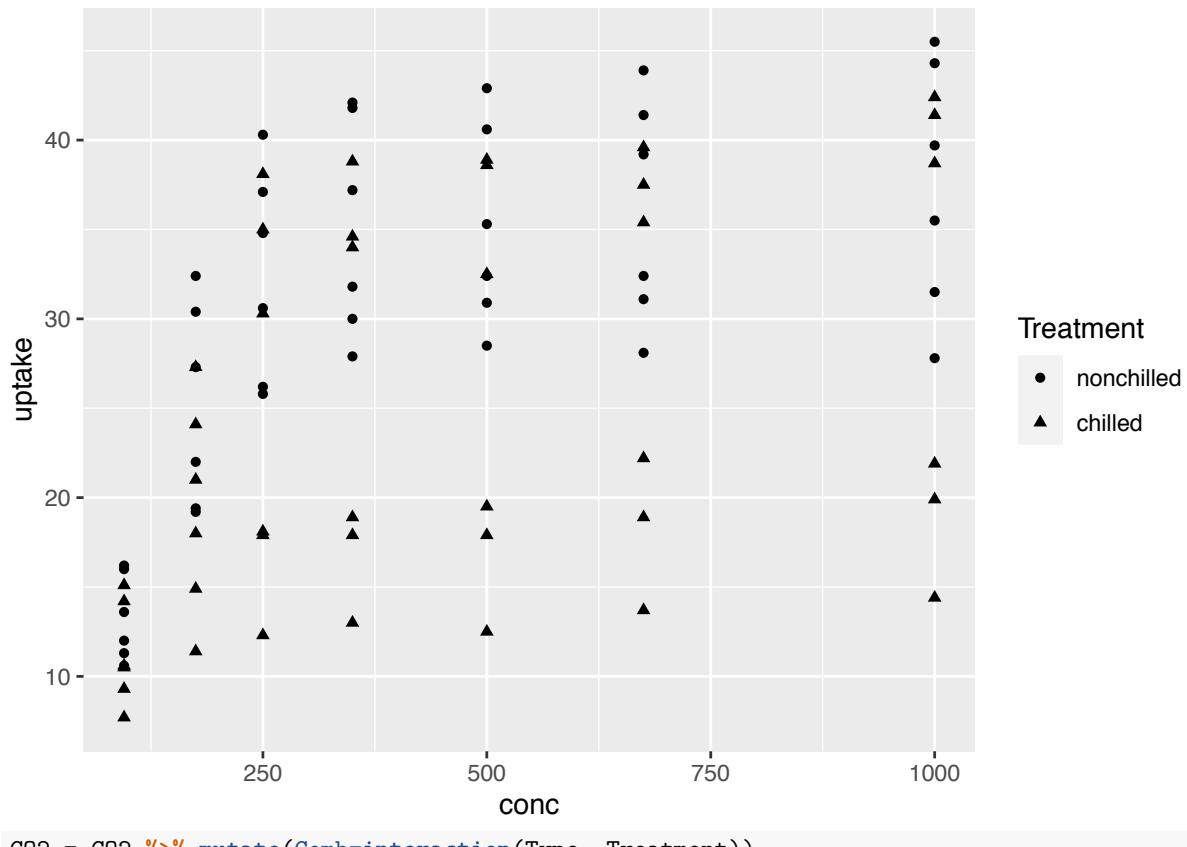
```
head(state, 2)

##          Illiteracy state.region      x      y
## Alabama     2.1       South -86.7509 32.5901
## Alaska      1.5       West -127.2500 49.2500

ggplot(state, aes(y=y,x=x)) + geom_point(aes(size=state.region))+ 
  scale_size_discrete(name="Region", range=c(2,10))

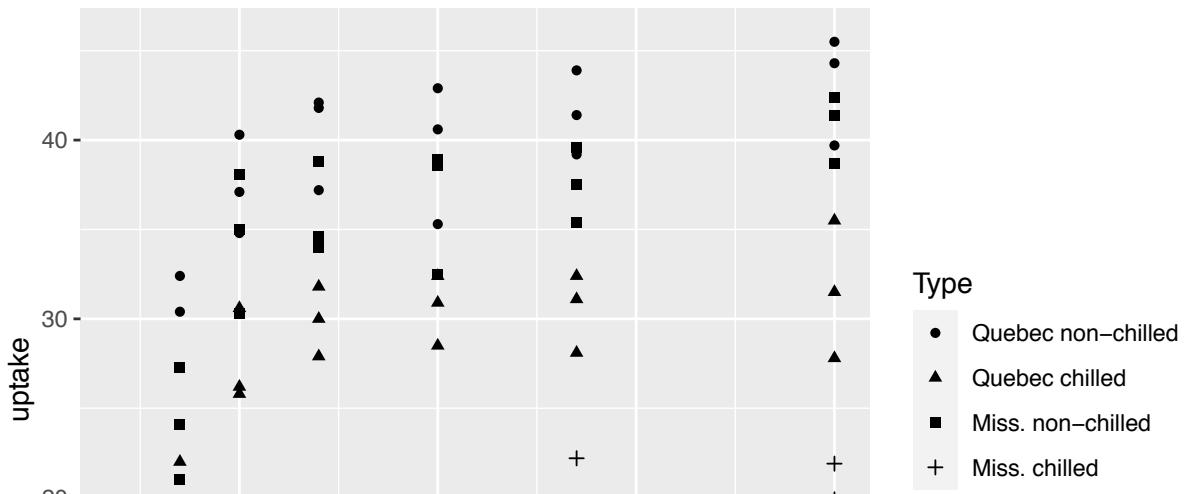
## Warning: Using size for a discrete variable is not advised.
```





```
CO2 = CO2 %>% mutate(Comb=interaction(Type, Treatment))
CO2 %>% pull(Comb) %>% levels
```

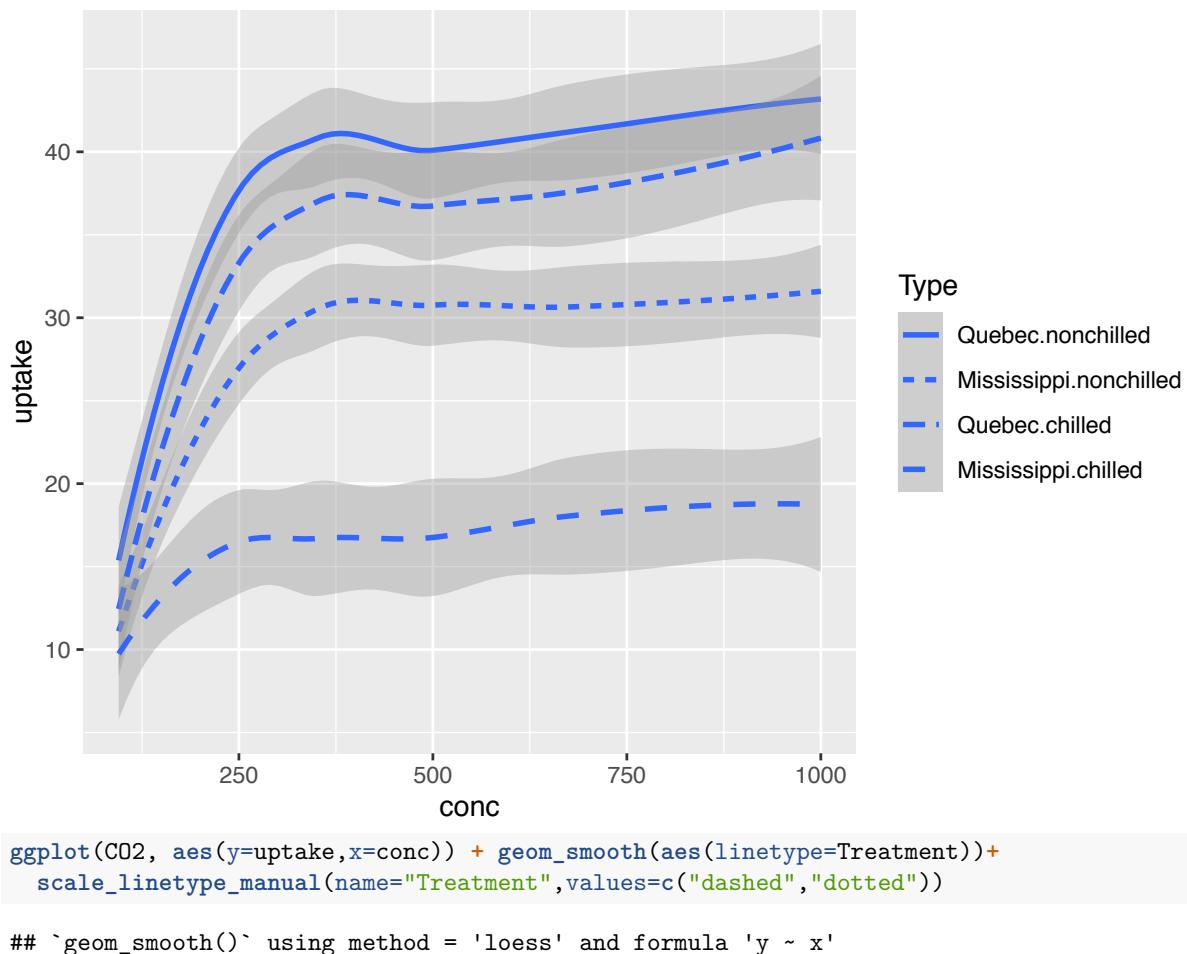
```
## [1] "Quebec.nonchilled"      "Mississippi.nonchilled" "Quebec.chilled"
## [4] "Mississippi.chilled"
ggplot(CO2, aes(y=uptake,x=conc)) + geom_point(aes(shape=Comb)) +
  scale_shape_discrete(name="Type",
    labels=c("Quebec non-chilled","Quebec chilled",
            "Miss. non-chilled","Miss. chilled"))
```

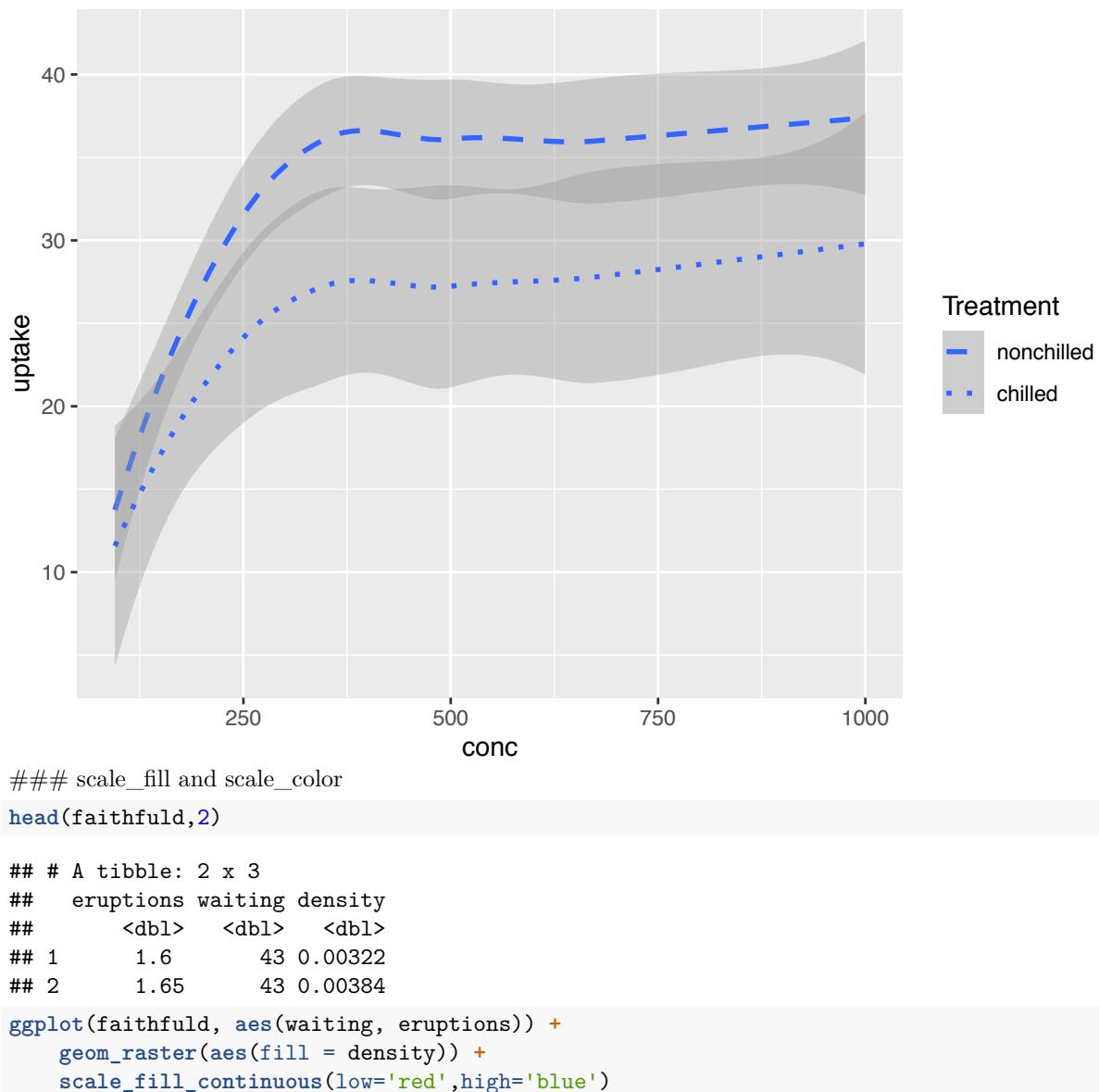


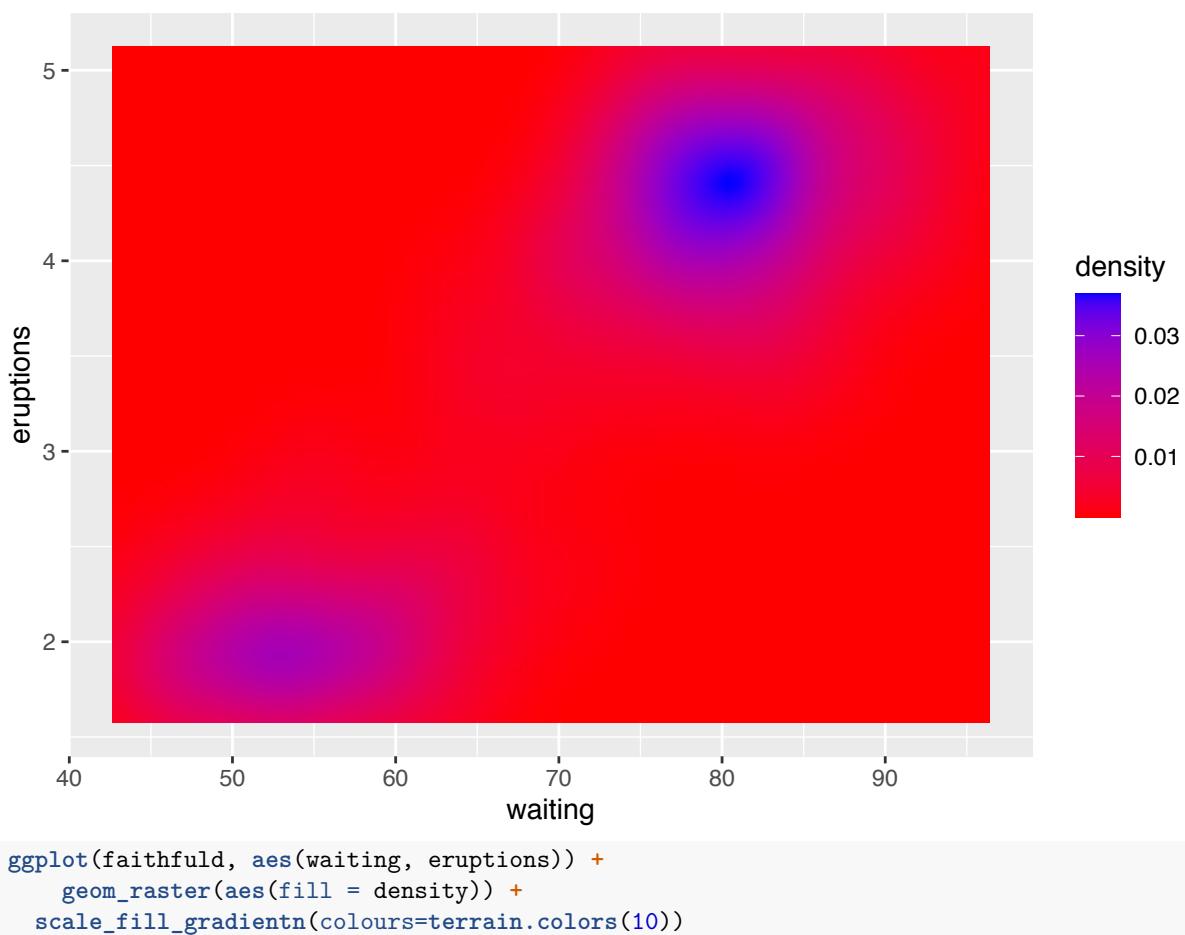
```
## scale_linetype
head(CO2, 2)

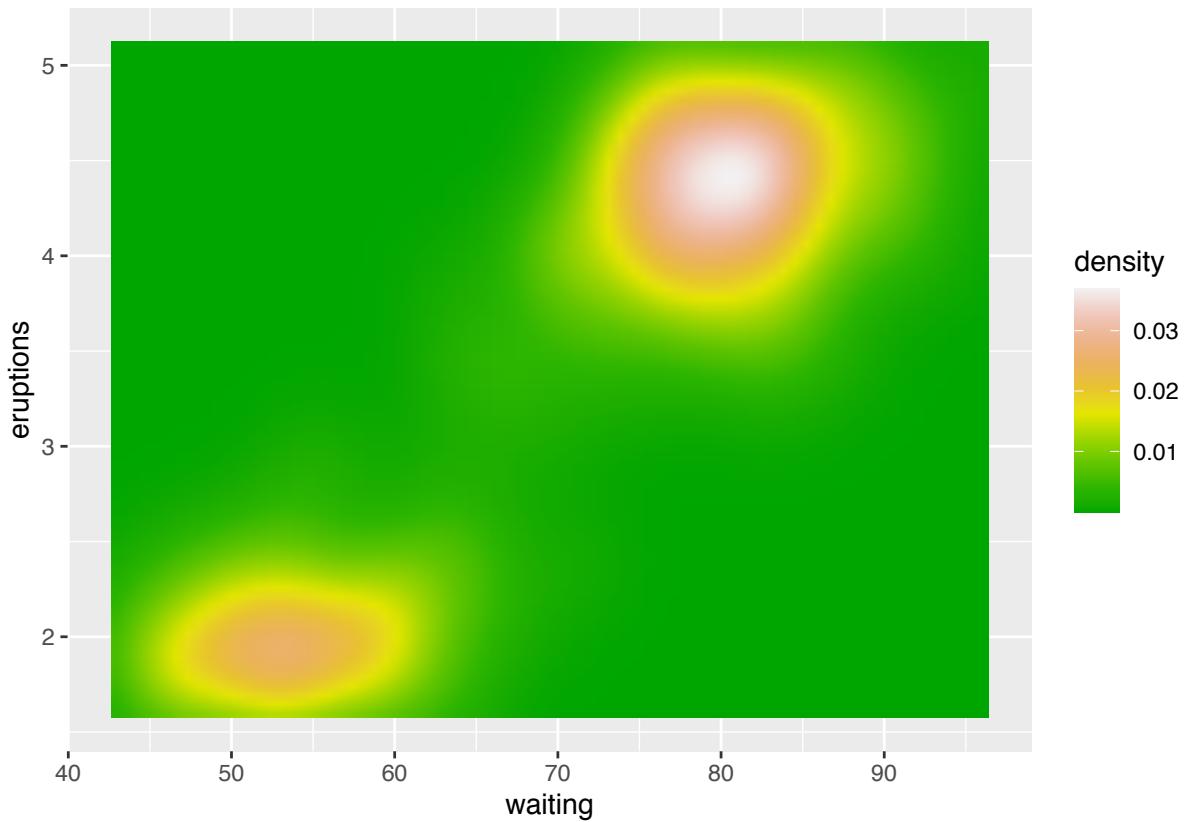
## Grouped Data: uptake ~ conc | Plant
##   Plant Type Treatment conc uptake           Comb
## 1  Qn1 Quebec nonchilled  95  16.0 Quebec.nonchilled
## 2  Qn1 Quebec nonchilled 175 30.4 Quebec.nonchilled
ggplot(CO2, aes(y=uptake, x=conc)) + geom_smooth(aes(linetype=Comb)) +
  scale_linetype_discrete(name="Type")

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

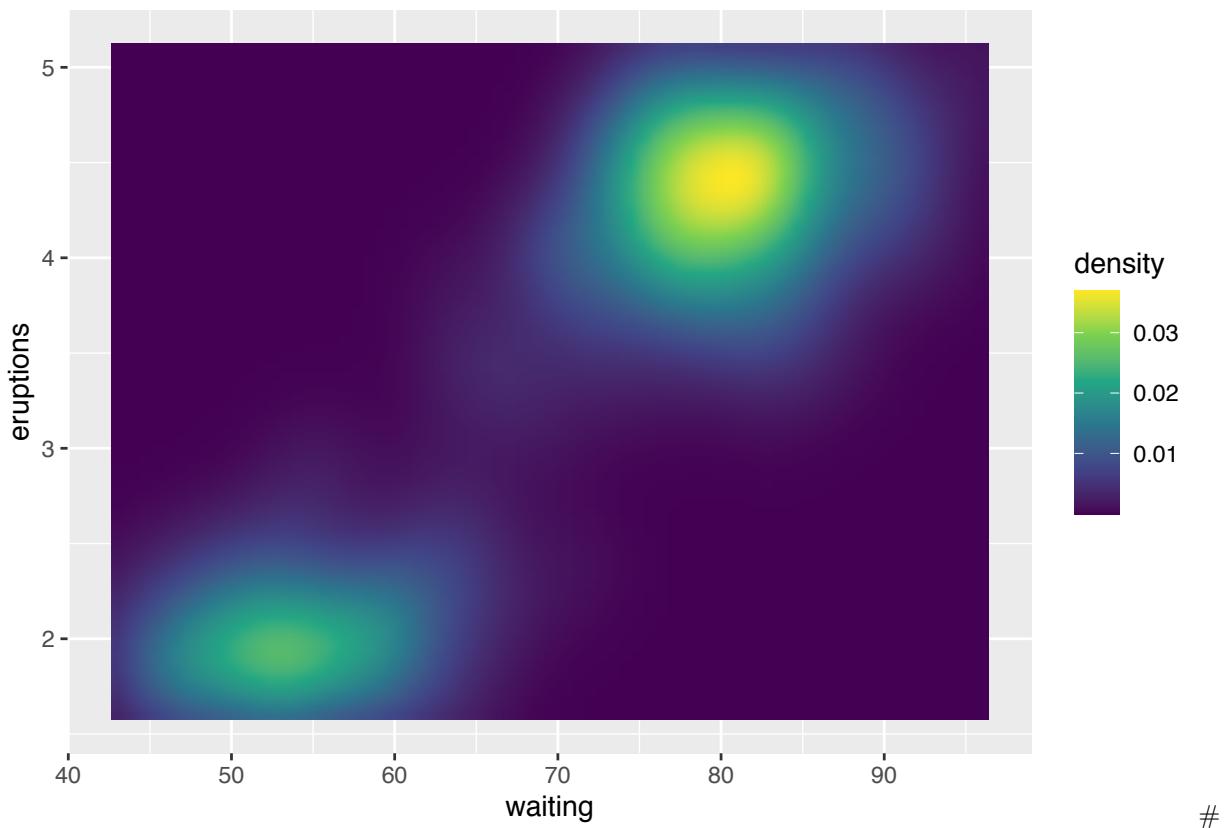






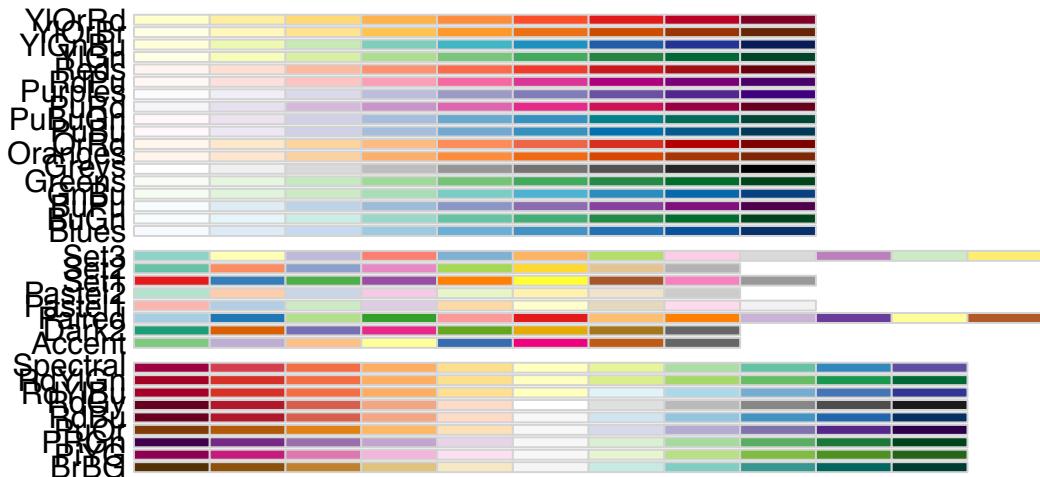


```
ggplot(faithful, aes(waiting, eruptions)) +  
  geom_raster(aes(fill = density)) +  
  scale_fill_viridis_c(option='D') #also try scale_fill_viridis_b
```



COLOR PALETTES

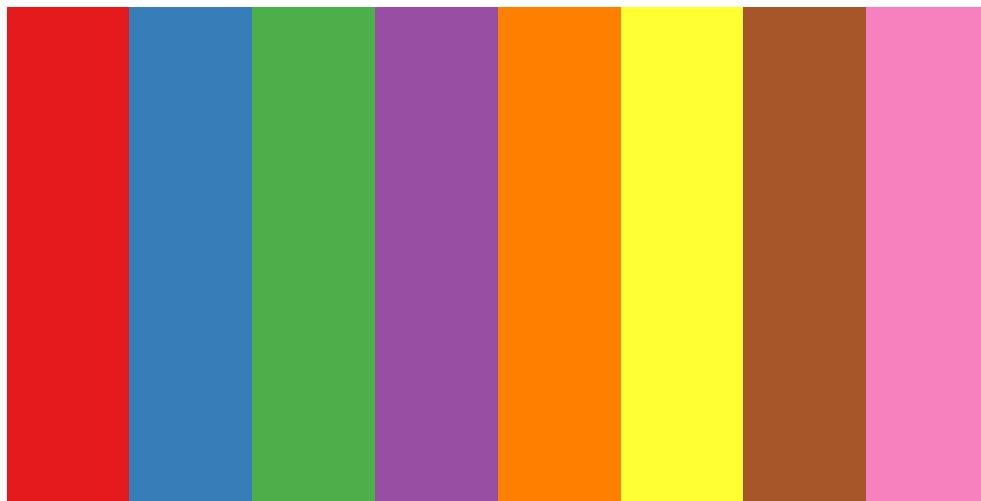
```
RColorBrewer::display.brewer.all()
```



```
RColorBrewer::brewer.pal(n=8, name='Set1')
```

```
## [1] "#E41A1C" "#377EB8" "#4DAF4A" "#984EA3" "#FF7F00" "#FFF33" "#A65628"
## [8] "#F781BF"
```

```
RColorBrewer::display.brewer.pal(n=8, name='Set1')
```



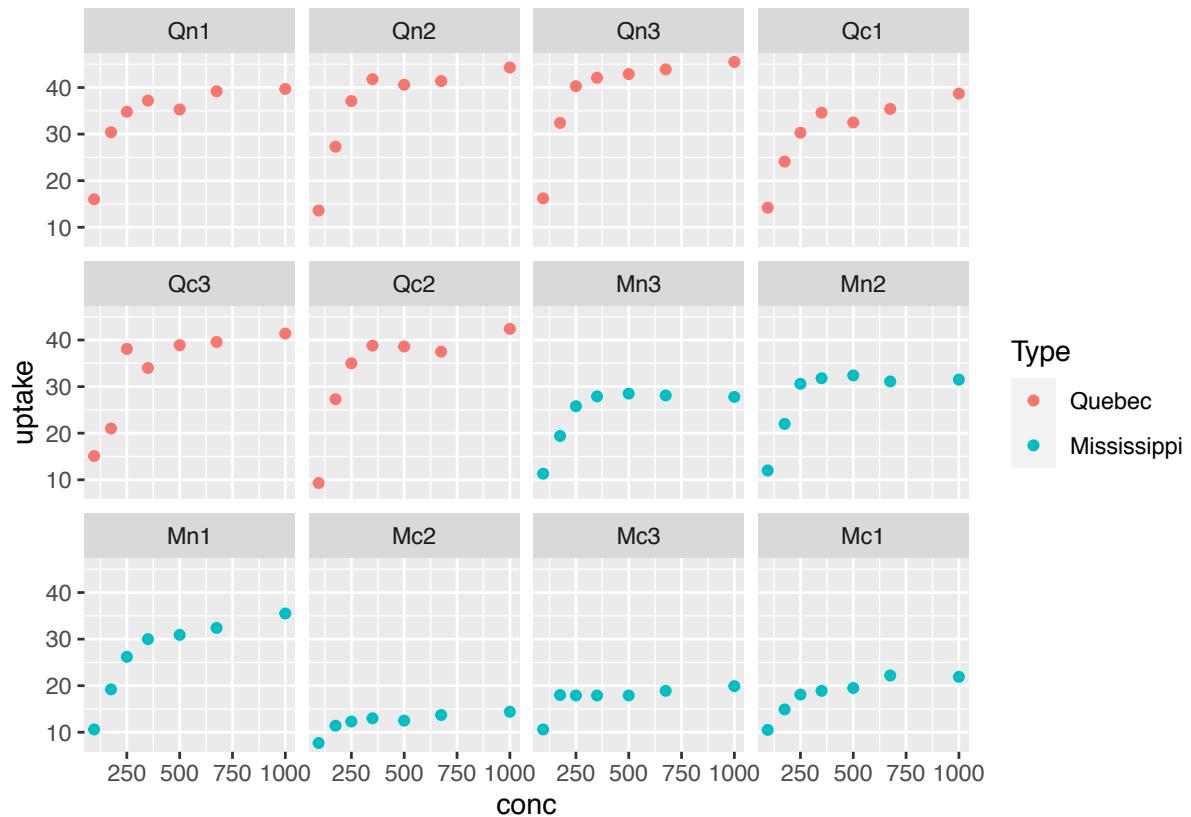
Set1 (qualitative)

FACETS They

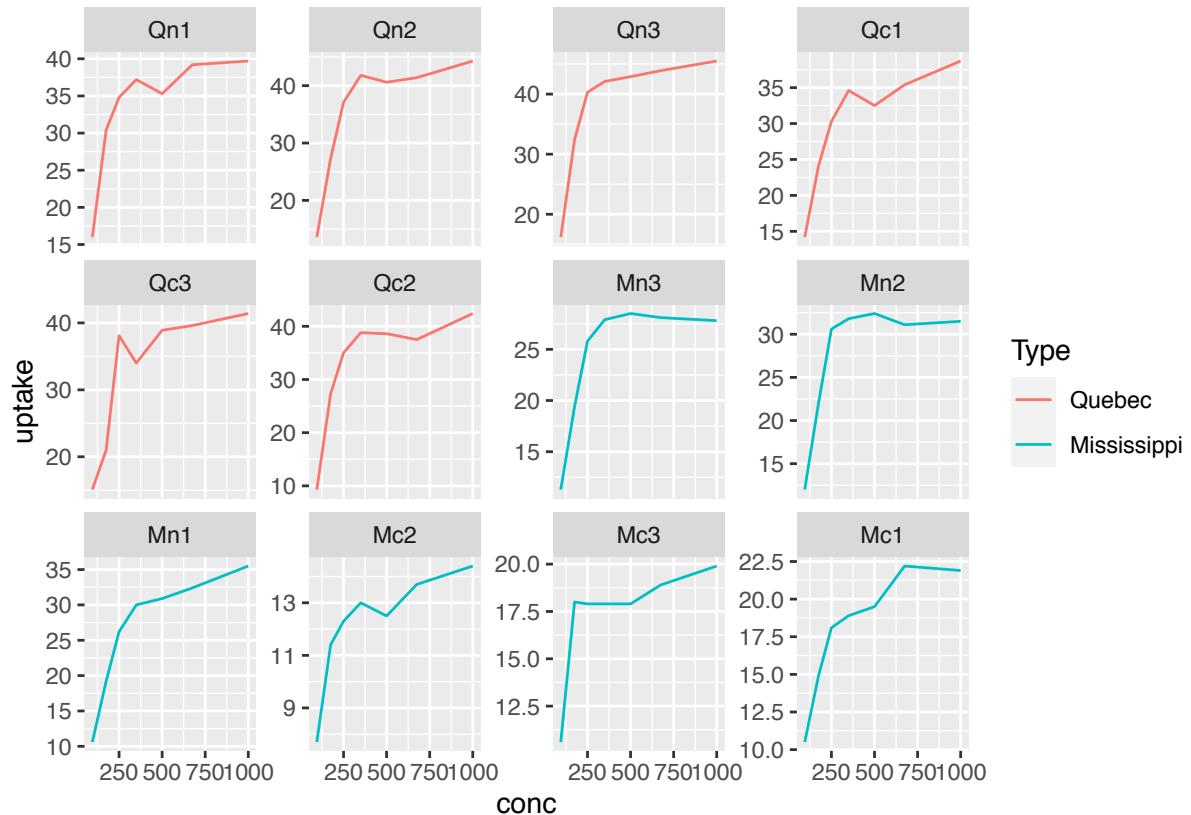
create panels of graphs; ie. matrices of plots - facet_wrap - facet_grid

Facetting by plants

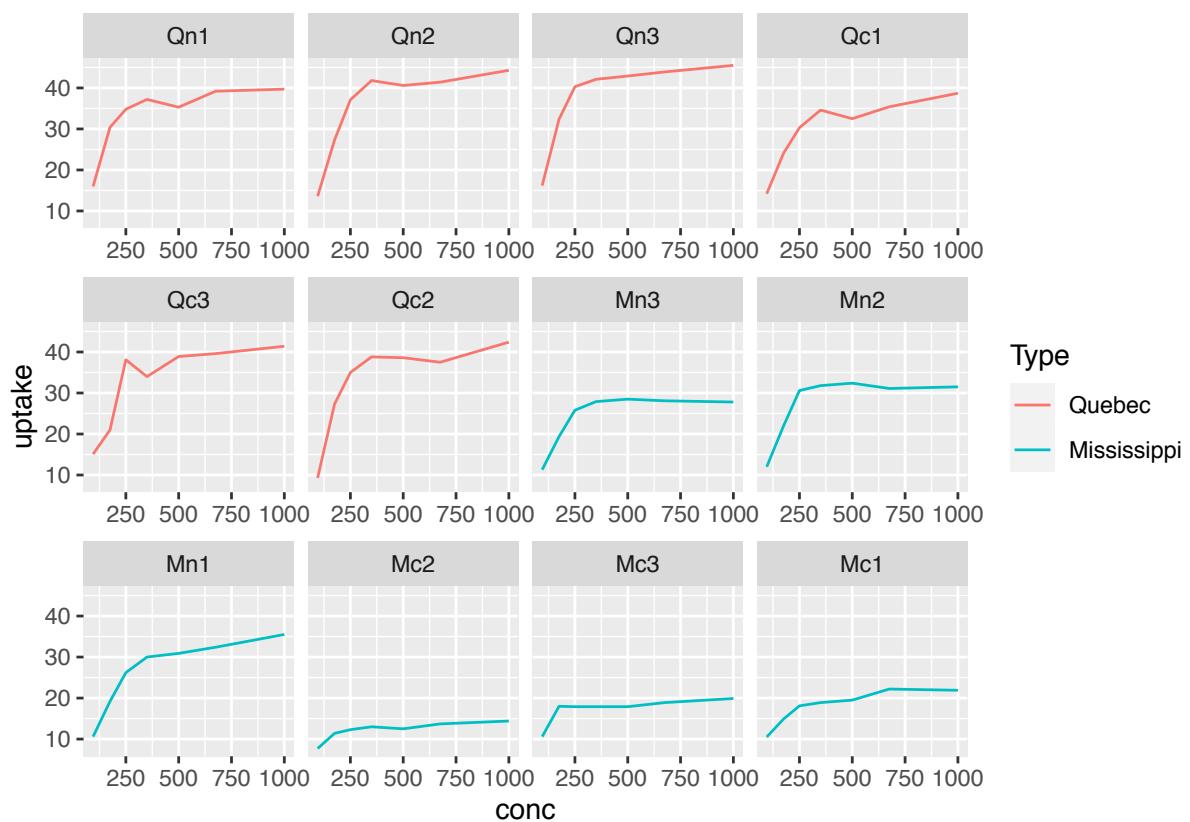
```
ggplot(CO2)+geom_point(aes(x=conc,y=uptake, colour=Type))+  
facet_wrap(~Plant)
```



```
ggplot(CO2)+geom_line(aes(x=conc,y=uptake, colour=Type))+  
facet_wrap(~Plant, scales='free_y')
```

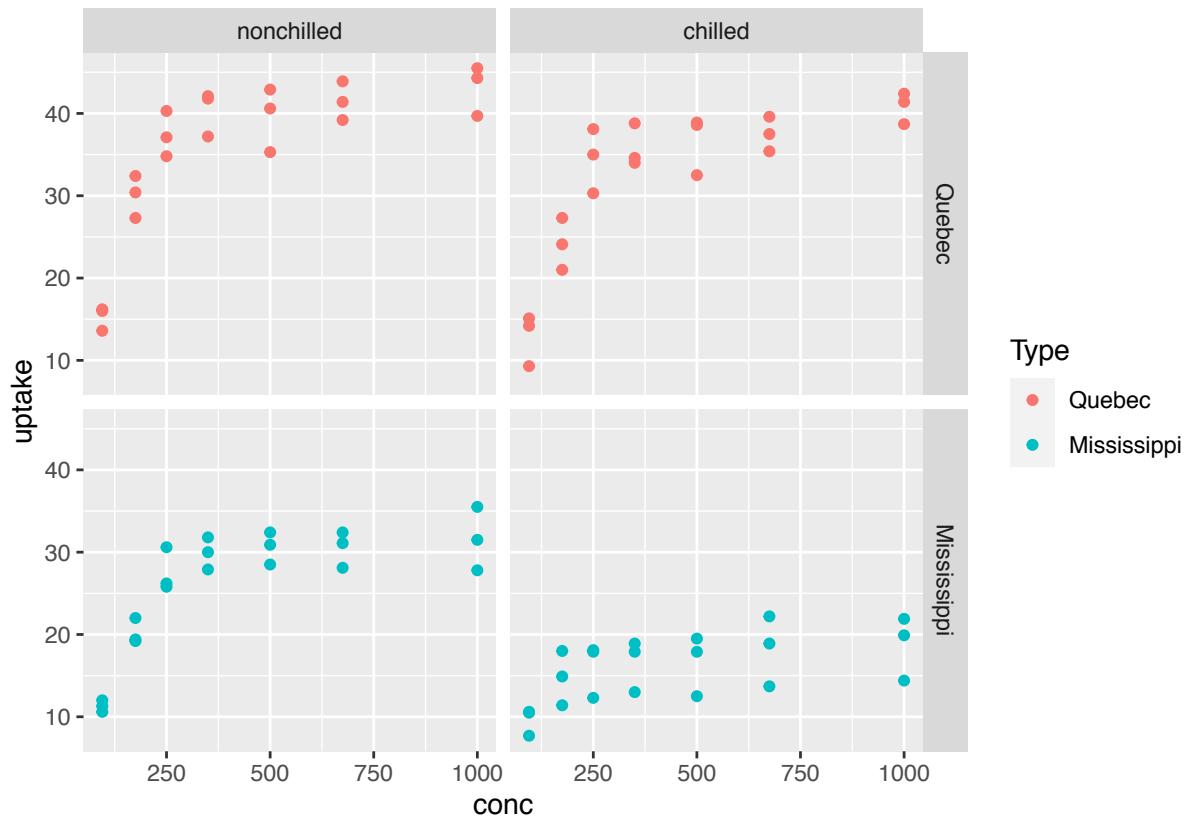


```
ggplot(CO2)+geom_line(aes(x=conc,y=uptake, colour=Type))+  
facet_wrap(~Plant, scales='free_x')
```



facet_grid gives us a grid of panels, rather than a grid that is wrapped around

```
ggplot(CO2)+geom_point(aes(x=conc,y=uptake, colour=Type))+  
facet_grid(Type~Treatment)
```

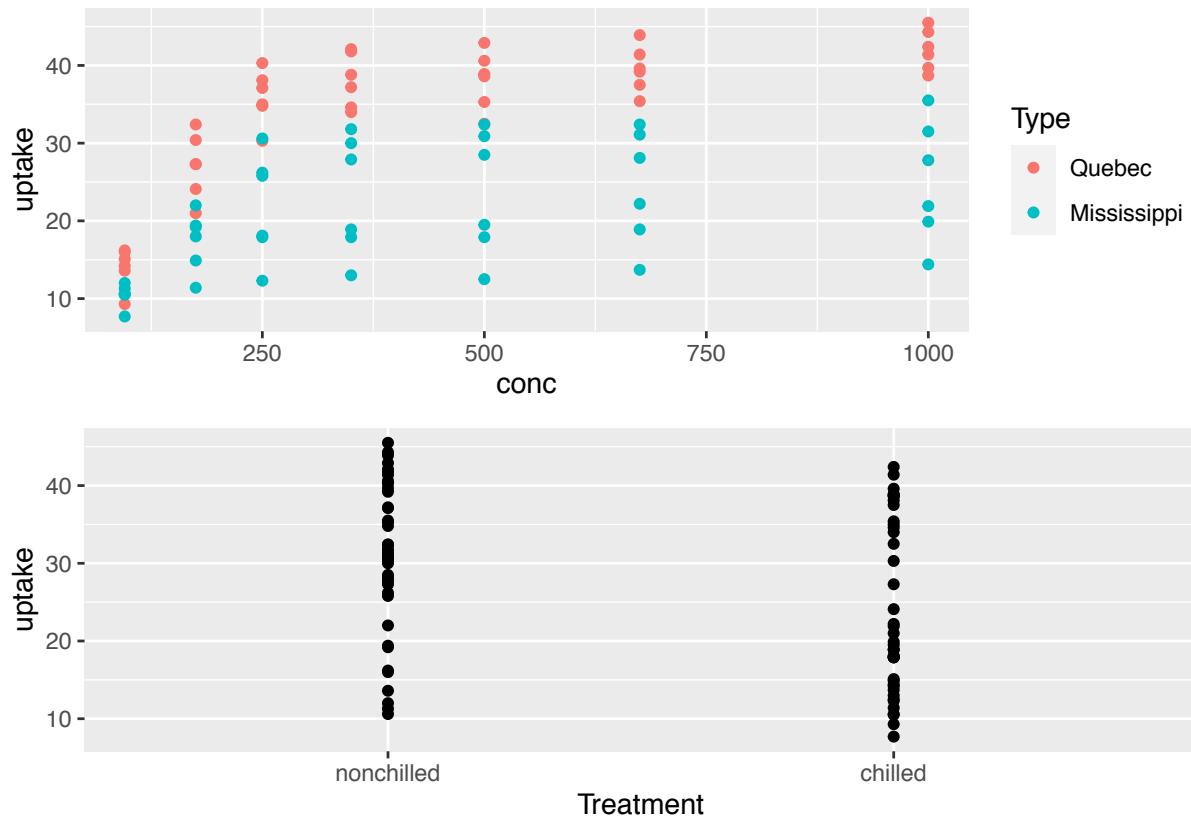


Multiple plots in one figure

We have to create each plot separately, and then use the command `grid.arrange(plot1,plot2)`

```
g1 <- ggplot(CO2)+geom_point(aes(x=conc,y=uptake, colour=Type))
g2 <- ggplot(CO2)+geom_point(aes(x=Treatment,y=uptake))

grid.arrange(g1, g2)
```

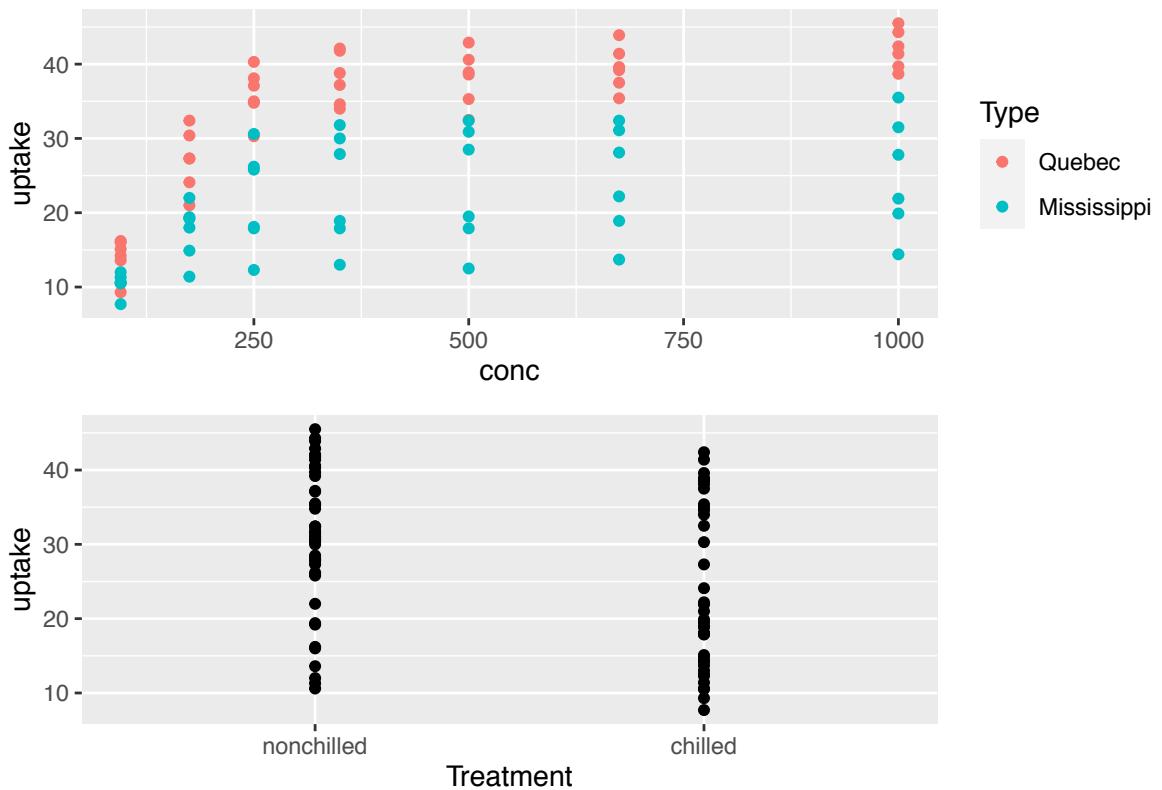


```
### Patchwork package (plot1/plo2 or plot1+plot2)
```

```
plot1/plot2
```

```
g1 <- ggplot(CO2)+geom_point(aes(x=conc,y=uptake, colour=Type))
g2 <- ggplot(CO2)+geom_point(aes(x=Treatment,y=uptake))
```

```
g1/g2
```

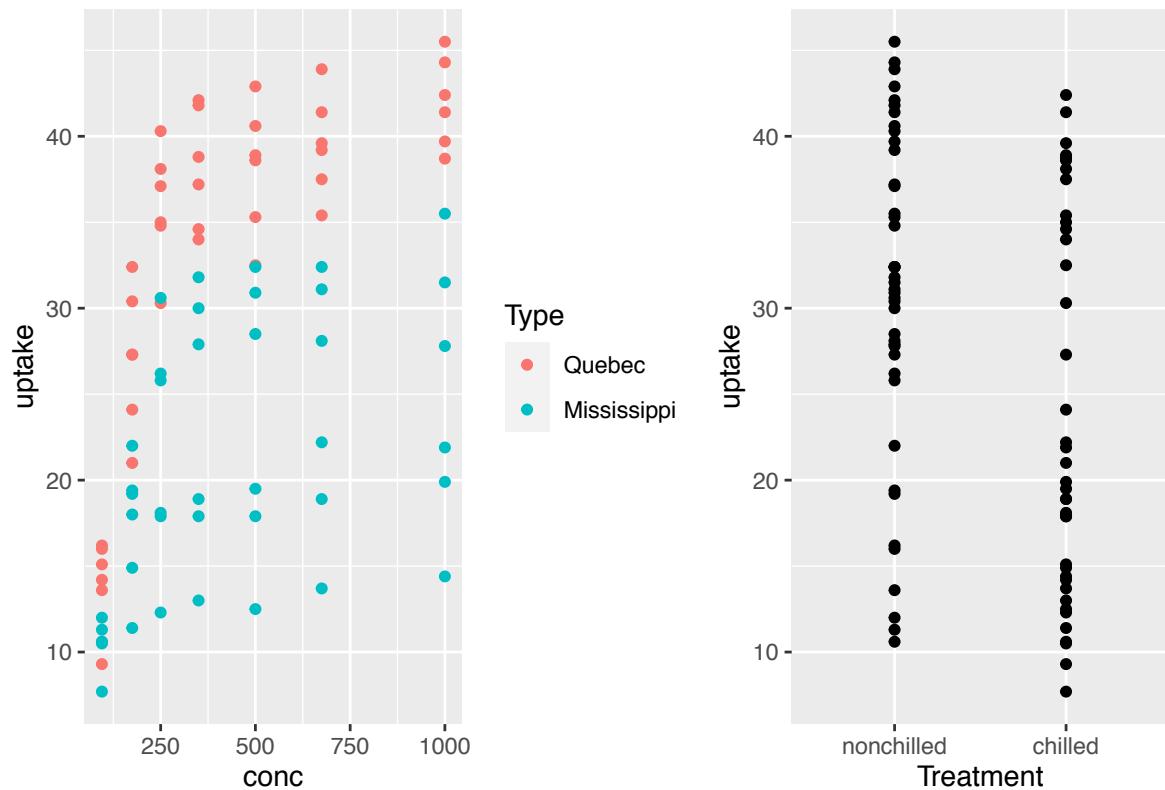


Patchworks lines up the x axes

plot1+plot2

```
g1 <- ggplot(CO2)+geom_point(aes(x=conc,y=uptake, colour=Type))
g2 <- ggplot(CO2)+geom_point(aes(x=Treatment,y=uptake))

g1+g2
```

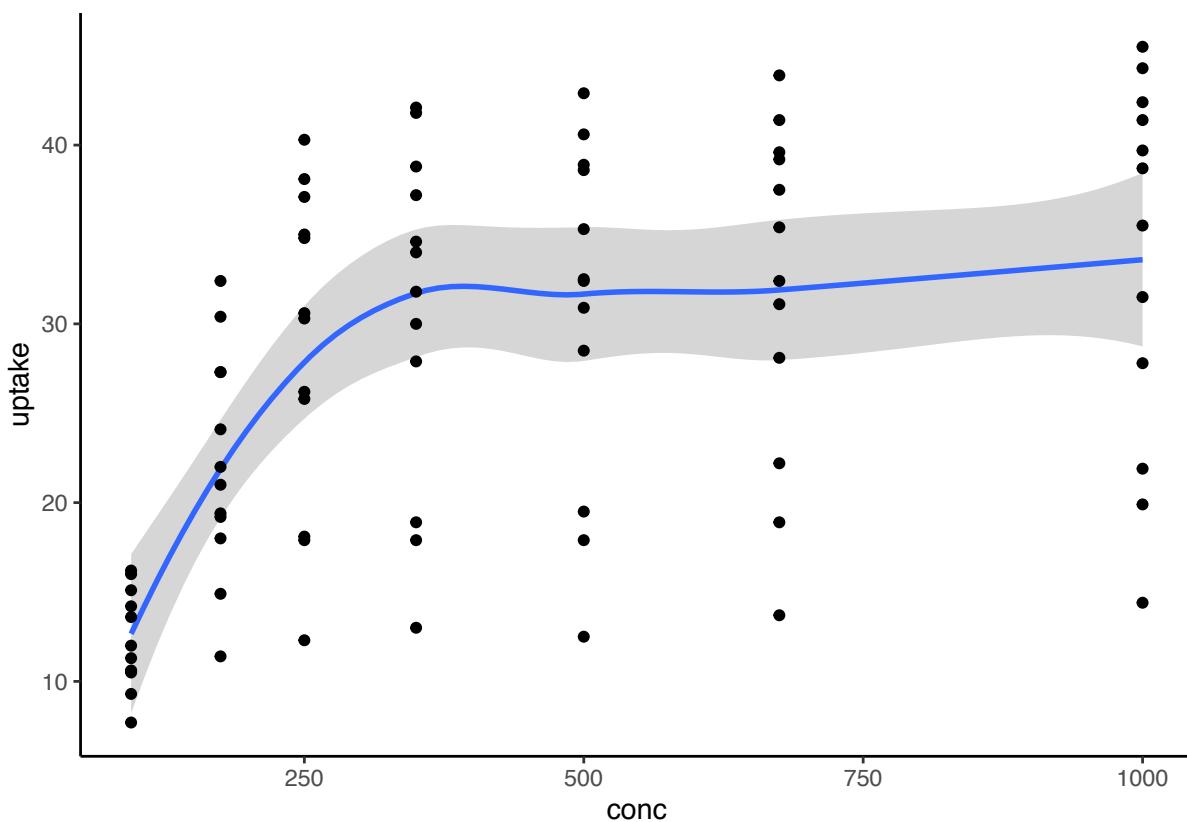


THEMES

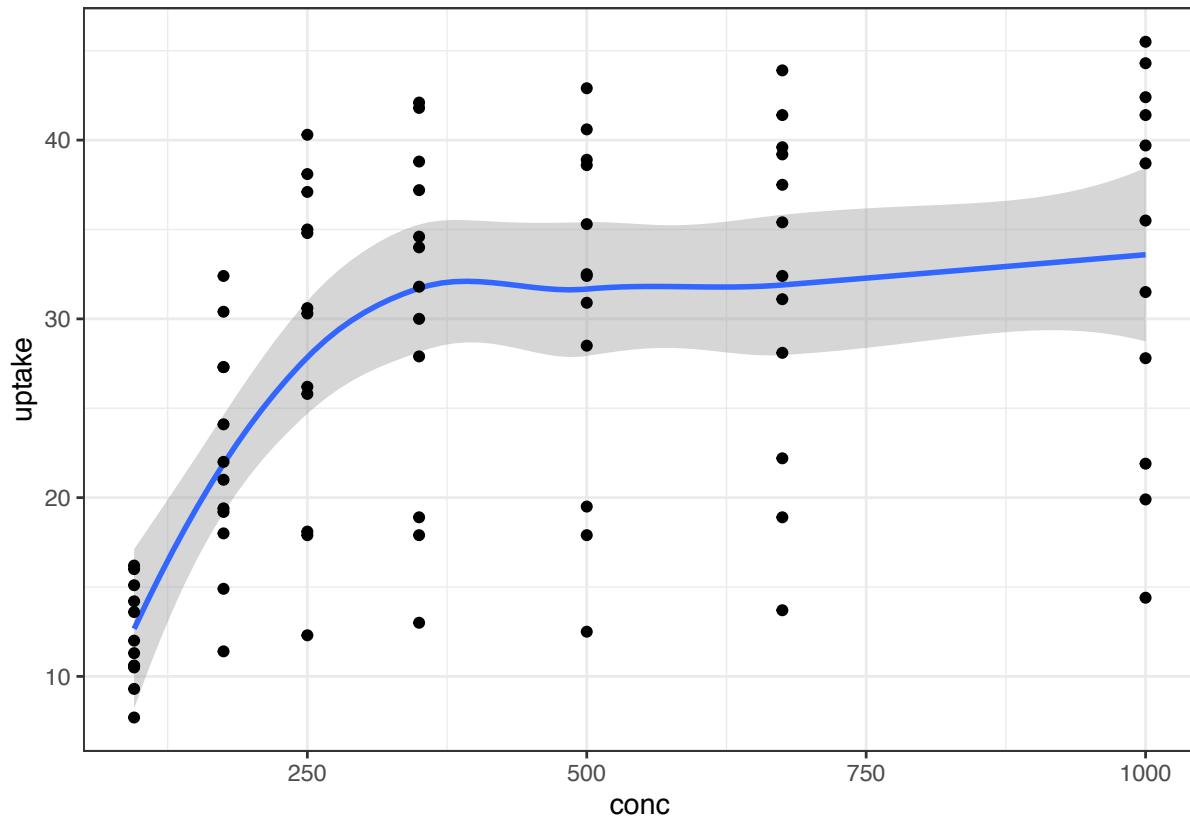
`theme_classic`

```
ggplot(CO2, aes(y = uptake, x = conc)) + geom_smooth() +
  geom_point() + theme_classic()

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

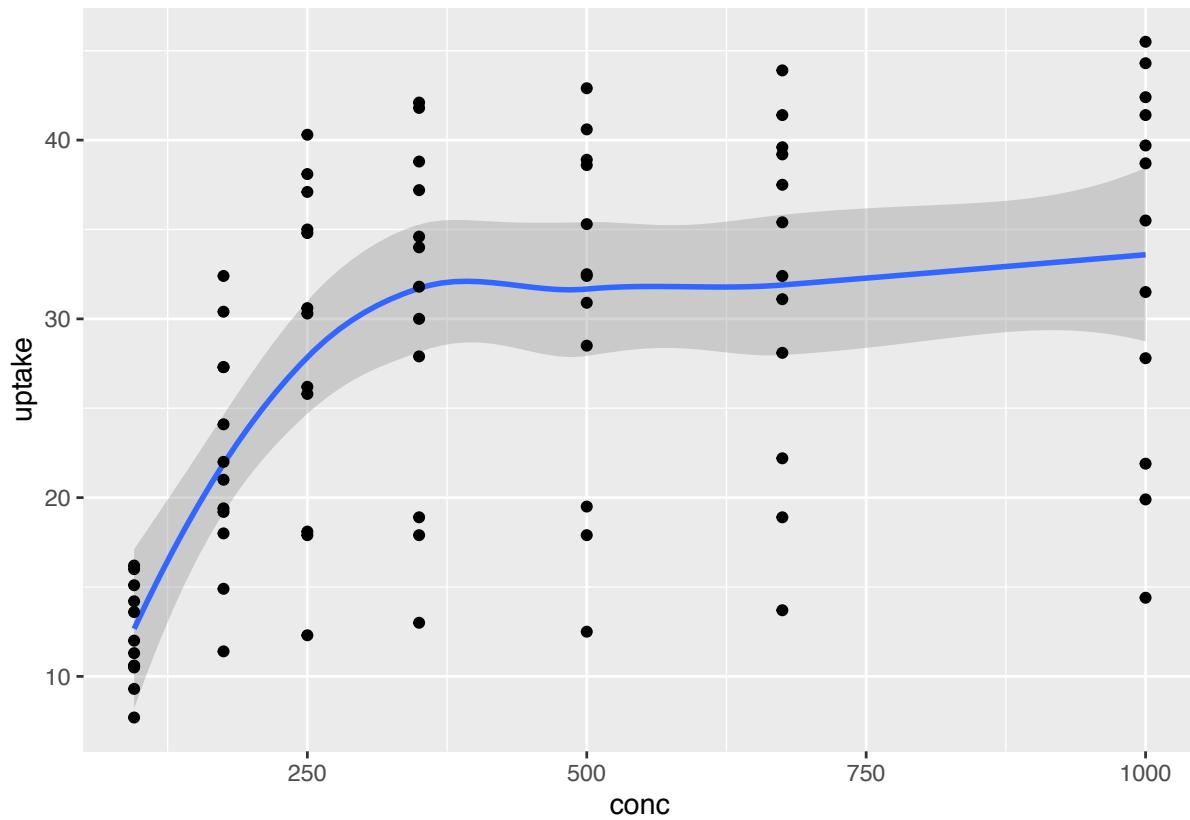


```
## theme_bw  
ggplot(CO2, aes(y = uptake, x = conc)) + geom_smooth() +  
  geom_point() + theme_bw()  
  
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

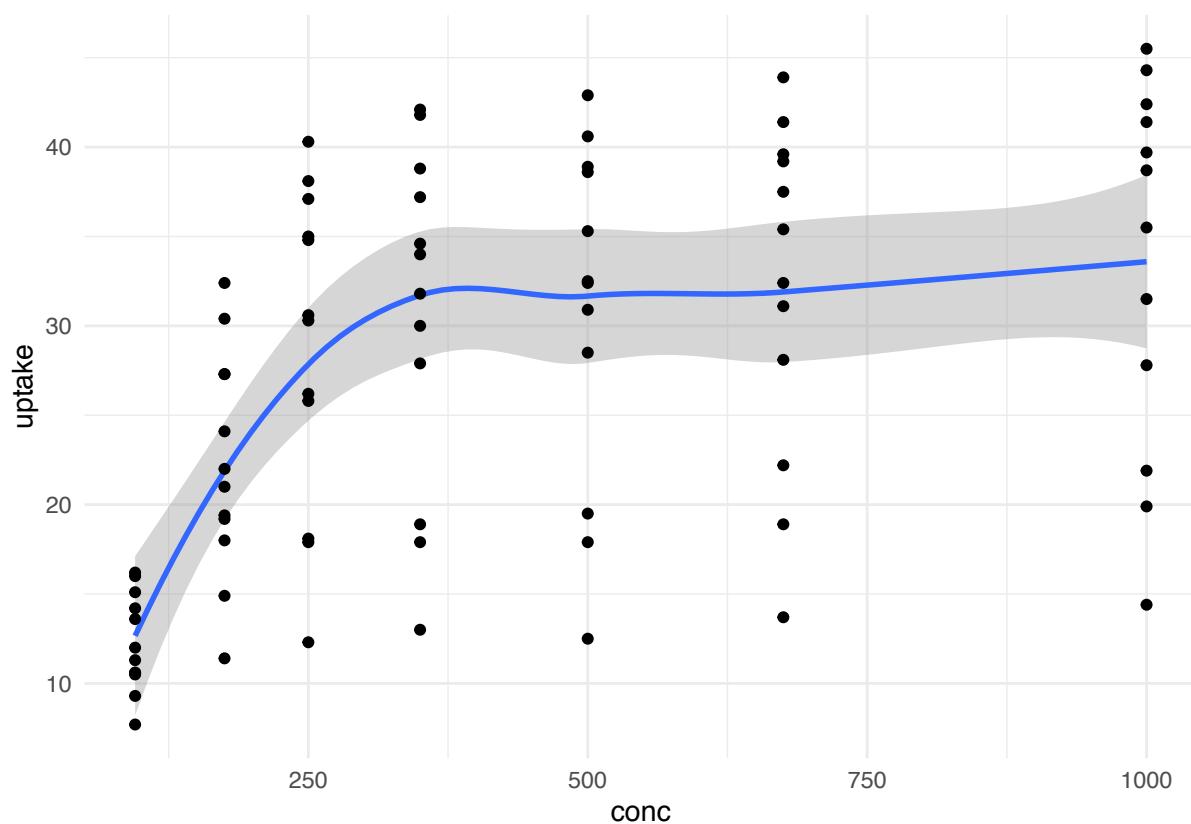


```
## theme_grey
ggplot(CO2, aes(y = uptake, x = conc)) + geom_smooth() +
  geom_point() + theme_grey()

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

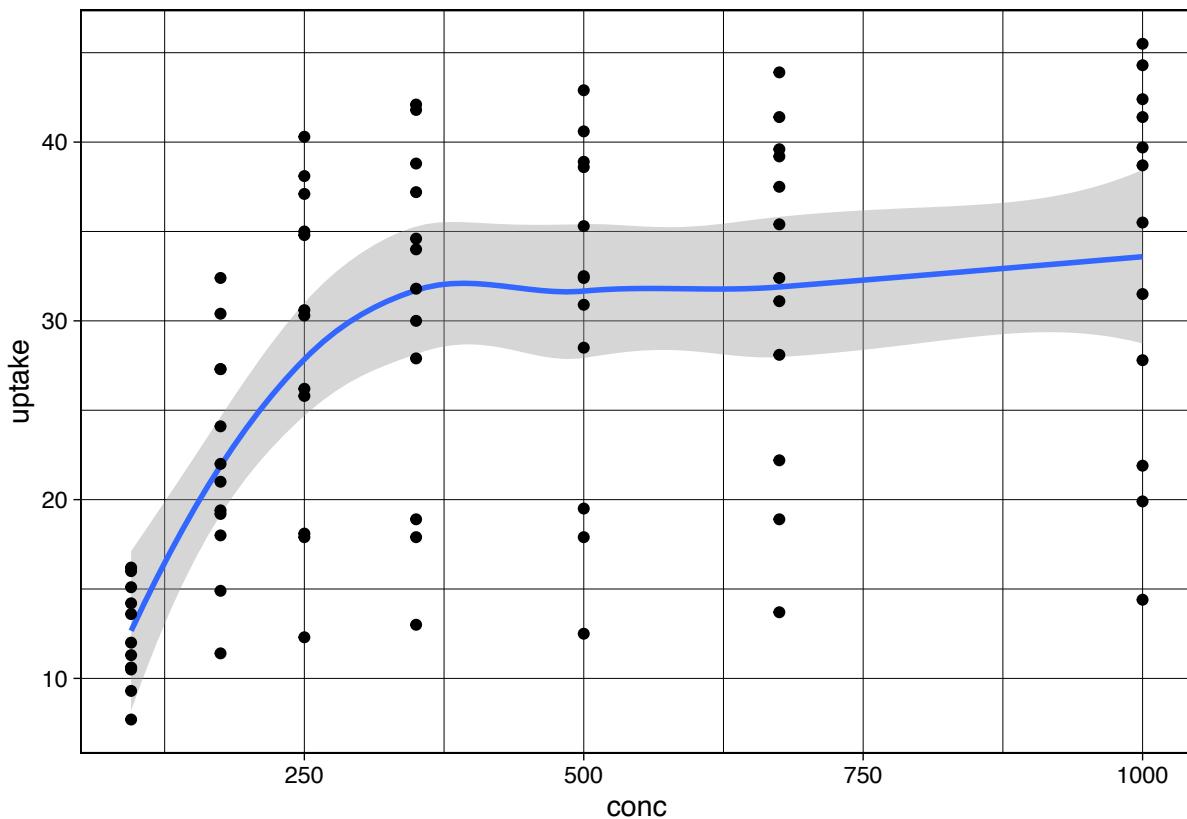


```
## theme_minimal  
ggplot(CO2, aes(y = uptake, x = conc)) + geom_smooth() +  
  geom_point() + theme_minimal()  
  
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

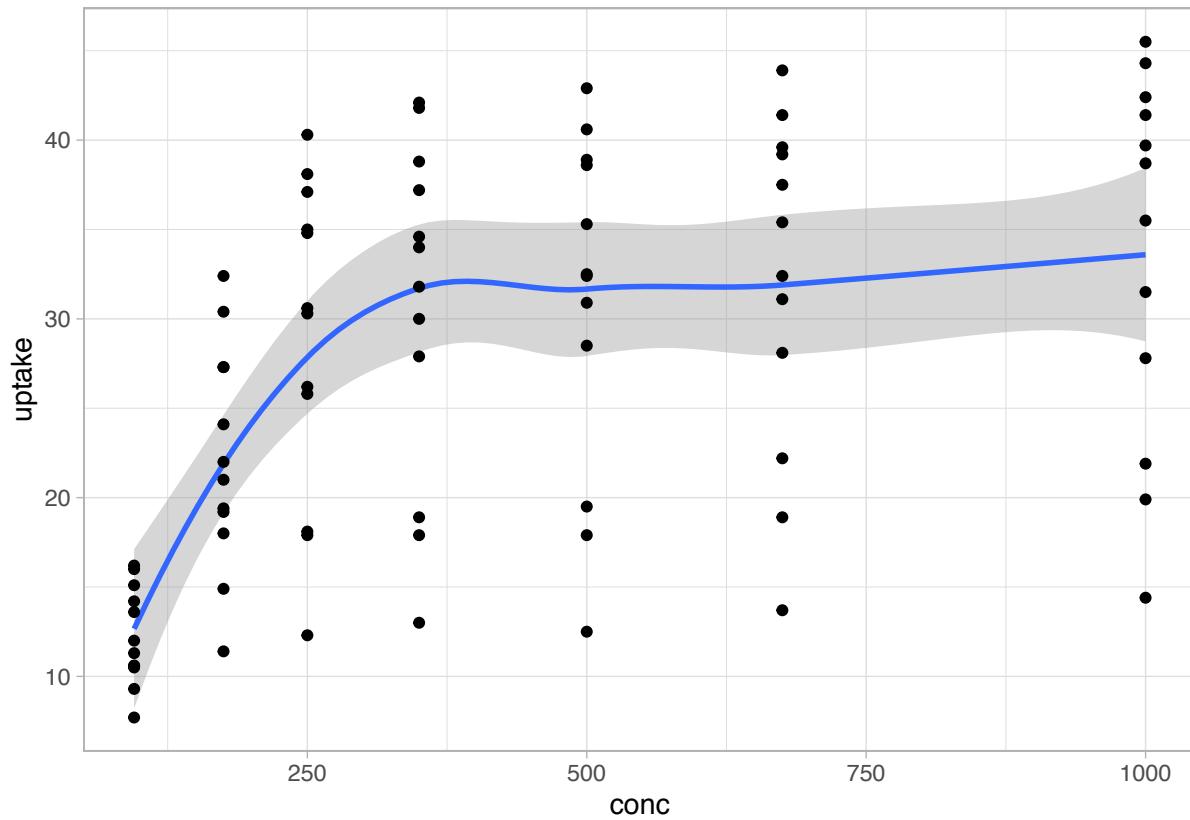


```
## theme_linedraw
ggplot(CO2, aes(y = uptake, x = conc)) + geom_smooth() +
  geom_point() + theme_linedraw()

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

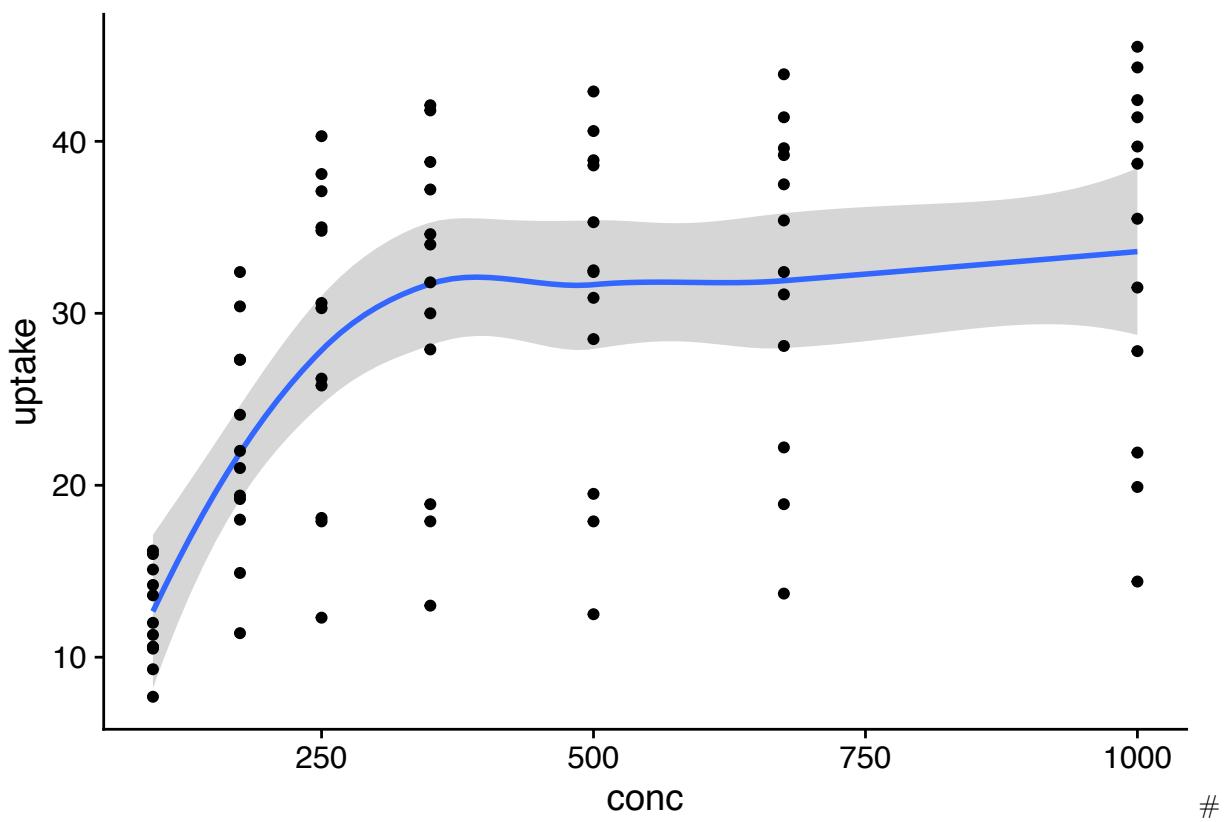


```
## theme_light  
ggplot(CO2, aes(y = uptake, x = conc)) + geom_smooth() +  
  geom_point() + theme_light()  
  
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```
## theme_cowplot (cowplot package)
ggplot(CO2, aes(y = uptake, x = conc)) + geom_smooth() +
  geom_point() + theme_cowplot()

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



Other personalisation options

SAVING GGPLOT FIGURES

```
g1 <- ggplot(CO2) + geom_point(aes(x=conc, y=uptake, colour=Type))  
ggsave(filename='figure1.pdf', width=7, height=5)
```

3_Linear_modelling

Sara Kophamel

01/12/2020

Presentation 7.1.

file:///Users/sara/Documents/PhD/HELP/STATISTICS%20and%20GRAPHS/-%20R%20and%20STATS%20COURSES/2020%20Murray%20Logan%20R%20Course/Logan_Course/docs/pres7.1.html#1

1- THEORY

All models that we will try to fit are low dimensional summaries. They do not try to predict anything and to capture all sources of uncertainty. They only will try to give you an insight on what is going on into your specific model. A linear model is not necessarily give you a better explanation of the world. It is better to keep the model as simple as possible.

A linear model implies that our data has been generated by a Gaussian-like process, by a normal distribution. We therefore need to consider the underlying generation process, and to test the normality assumptions.

How do we estimate model parameters?

```
# yi~b0+b1xi  
# b0=Intercept  
# b1=Slope
```

If we assume y_i is drawn from a normal (gaussian) distribution, we use Ordinary Least Squares OLS

If we assume another distribution (binomial, poisson, etc), we will have to create a model that fits this distribution.

Murray won't distinguish between LM and GLM (it's pretty much the same).

The purpose of statistical modelling are to: - Describe relationships / effects - Estimate effects - Predict outcomes

What criterion do we use to assess best fit? Depends on how we assume Y is distributed. The best line of best fit is the one with less distance between all data points. This process can be easily done by hand.

ESTIMATION (slides >11)

Parameters needed

- location (mean)
- spread (variance) = uncertainty

Least squares estimates

(Check presentation)

It minimizes the sum of the squared residuals, and is calculated by a simple matrix algebra.

The least squares regression line could be checked by trial and error, as long as the data is normal. But what if this is not the case? We would violate the assumptions of normality, and the test won't be reliable. There are alternatives for these cases.

Assumptions to be met

The residuals have to be... - Gaussian (normally distributed) - Homogeneity of variance (equally varied) - Linearity -> We are fitting a straight line through the data - Independent variables. Otherwise, we would not represent the populations as intended. Sometimes, you can't really know if the data was collected independently, or if there might have been a bias in collection (eg. picking sick animals, which will have altered blood values)

Gaussian (normal) distribution

Properties that determine a Gaussian distribution: - mean - variance or SD (how wide it is)

What do we do if the assumptions are not met?

- Identifying outliers and getting rid of them if both the boxplots and the outlier tests identify them so, and if they are not representing the real biological variation in the data (common sense)
- Scale transformations: Fit the data into shape so that it is normally distributed. For example, log-transformation if right skew, square root transformation if left skew. However, it is better that the model reflects the data distribution process. If the data is skewed, our model should also reflect that. So scale transformations are ok, but they should not be used as a first option.
- Using another model that fits the non-normal distribution!

DATA TYPES (see slide 33)

Distribution types

Measurement data

- (a) Gaussian or Normal: It is defined by a mean and a variance/spread

(b) logNormal

(c) Gamma:

Reaches from 0 to infinitive. It defines everything above 0, and can't be negative. It is defined by a shape and a scale (also related to mean and variance). Unlike a Gaussian, where we assume that mean and variance are UNRELATED, in a Gamma distribution we assume that shape and scale ARE related. As the mean decreases, the distribution will become smaller, and lower (cause it can't go below 0). The bigger the mean, the more it will look like a Gaussian distribution. If the lengths are >100, the Gamma and Gaussian are super similar and will converge, and you might as well use the Gaussian distribution for your model. Example: How long did it take for something to reach a specific level?

Count data

(a) Poisson: For count data, a natural distribution is a Poisson. distribution, which defines from 0 to infinity, but with discrete numbers (no decimals). It is ideal for counts. It is defined by the parameter Lambda, which is the mean, variance AND the df. The larger the mean, the larger the variance, and the more symmetrical it will become (once the mean is >30, it is pretty much a Gaussian, which can then be used instead). In a Poisson distribution, we assume that the Mean and the Variance ARE exactly the same. We define this situation by "dispersion", which is =1. If you take the variance and divide it by the mean, you should get =1.

So what if our data is more varied than a Poisson distribution?

(b) Negative binomial distribution: That is when a negative binomial comes in. A negative binomial is defined by 2 parameters:

- mean
- dispersion/variance The mean and variance are NOT related anymore. The variance is estimated separately.

If you were to fit a negative binomial to a Poisson data, a negative binomial would equal a Poisson. The more dispersion increases, the more it will differ from a Poisson. A neg binomial equation is the same as for a binomial distribution, just that one of the cases has a -.

Example: Number of successes until we get a failure (eg. flipping a coin x times until we get a tail, and not a head)

Binary data

(a) Binomial: You can only be in one of two states (zero or one, yes or no).

(b) Beta: What about percentage data? Some people like to use the binomial for percentage cover, but otherwise there is the BETA distribution. A beta distribution could look like a normal or exponential distribution (increase or decrease).

For that reason, you need quite a bit of data, as the shape determines what the parameters are gonna be. But otherwise, it is appropriate for percentage cover. It does NOT define

anything that is =0 or 1 (or 0 or 100 percent). For those cases, you'd have to do some tweaking.

Categorical data

= Ordinal data

GENERALIZED LINEAR MODELS

The equation of the GLM can predict ANY number (see: Systematic part of the equation, slide 43). But if we want a Poisson distribution, which only goes from 0-inf, we would need to link the scale of data on the right side of the equation, to what is appropriate to the left side of the equation (eg. zero). For example, we can use the log link function ($g()$). This link maps the scale of data that we got with the systematic part (right side of eq).

This creates an expected response, according to a link. So what links are appropriate?

- For a Gaussian distribution, we use an identity link (one to one mapping)
- Gamma: The default is to use an inverse link. That's fine for time, but for measurement data, a log link is better. -- Poisson: Log link is also used for Poisson and neg binomials.
- Neg binomials: Log link is also used for Poisson and neg binomials.
- Binomial: Logit (odds ratio scale)

ORDINARY LEAST SQUARES

To create this model, we use the maximum likelihood way of calculating the model, based on a Gaussian distribution.

Slide 48: What is the likelihood that our datapoints came from a distribution as shown on the graph? The maximum likelihood estimate for our mean will tell us that. In this case, it is 10. That allowed us to find the maximum likelihood estimate. There are algorithms that we can apply to speed up that intuitive process. For example, we might start off at the intercept, and keep following the curve (see plot), until we find the maximum likelihood parameter.

Let's say we calculated every combination of mean and variance. We could calculate the likelihood for those pairs, and look for the maximum likelihood = what is the highest value? Well, if you got >2 parameters, that might be a lot of combinations, and very hard to get there. So yes, we can use brute force and try every combination, or we can be smarter:

- Nelger-Mead algorithm: We could use a simplex calculation for arriving to the maximum calculation (slide 50): This algorithm tests all possible options, until it finds the maximum likelihood.
- Newton-Raphson method: Looks at where the likelihood function will become zero.

In summary, you just need to know that there are several ways of finding the maximum likelihood.

2- EXAMPLE 1 - GAUSSIAN DISTRIBUTION

In the ws folder, open: "glm_example1.Rmd"

And go down to line 54, and install and load the packages listed (BUT I copy pasted that Markdown file below, so just follow this file instead)

Packages

```
library(car)      #for regression diagnostics
library(broom)    #for tidy output
library(ggfortify) #for model diagnostics
library(DHARMa)   #for residual diagnostics
library(performance) #for residuals diagnostics
library(see)      #for plotting residuals
library(MASS)     #for glm.nb
library(sjPlot)   #for outputs
library(cowplot)  # for graph theme
library(knitr)    #for kable
library(effects)  #for partial effects plots
library(ggeffects) #for partial effects plots
library(emmeans)  #for estimating marginal means
library(modelr)   #for auxillary modelling functions
library(tidyverse) #for data wrangling
```

Scenario

Here is an example from @Fowler-1998-1998. An agriculturalist was interested in the effects of fertilizer load on the yield of grass. Grass seed was sown uniformly over an area and different quantities of commercial fertilizer were applied to each of ten 1 m² randomly located plots. Two months later the grass from each plot was harvested, dried and weighed. The data are in the file **fertilizer.csv** in the **data** folder.

FERTILIZER	YIELD
25	84
50	80
75	90
100	154
125	148
...	...

Explanation: First row: The first unit had 25 gr of fertiliser added, and after harvesting the grass a year later, 84 gr of grass were yielded. Another patch of grass had 50 gr of fertiliser, and 80 gr of grass were yielded, and so on.

So year is the response, and fertiliser is the predictor. We are therefore looking at a linear regression.

Yield is gonna some sort of measurement. We could use a Gaussian, providing the assumptions are met, or potentially a Gamma, cause the yield can be <0.
be Even a log normal might be useful.

FERTILIZER:	Mass of fertilizer (g.m^{-2}) - Predictor variable
YIELD:	Yield of grass (g.m^{-2}) - Response variable

The aim of the analysis is to investigate the relationship between fertilizer concentration and grass yield.

Read in the data

```
fert = read_csv('data/fertilizer.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   FERTILIZER = col_double(),
##   YIELD = col_double()
## )

glimpse(fert)

## Rows: 10
## Columns: 2
## $ FERTILIZER <dbl> 25, 50, 75, 100, 125, 150, 175, 200, 225, 250
## $ YIELD      <dbl> 84, 80, 90, 154, 148, 169, 206, 244, 212, 248

## Explore the first 6 rows of the data
head(fert)

## # A tibble: 6 x 2
##   FERTILIZER YIELD
##       <dbl> <dbl>
## 1        25     84
## 2        50     80
## 3        75     90
## 4       100    154
## 5       125    148
## 6       150    169

str(fert)

## tibble [10 × 2] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
## $ FERTILIZER: num [1:10] 25 50 75 100 125 150 175 200 225 250
## $ YIELD      : num [1:10] 84 80 90 154 148 169 206 244 212 248
## - attr(*, "spec")=
##   .. cols(
##     .. FERTILIZER = col_double(),
```

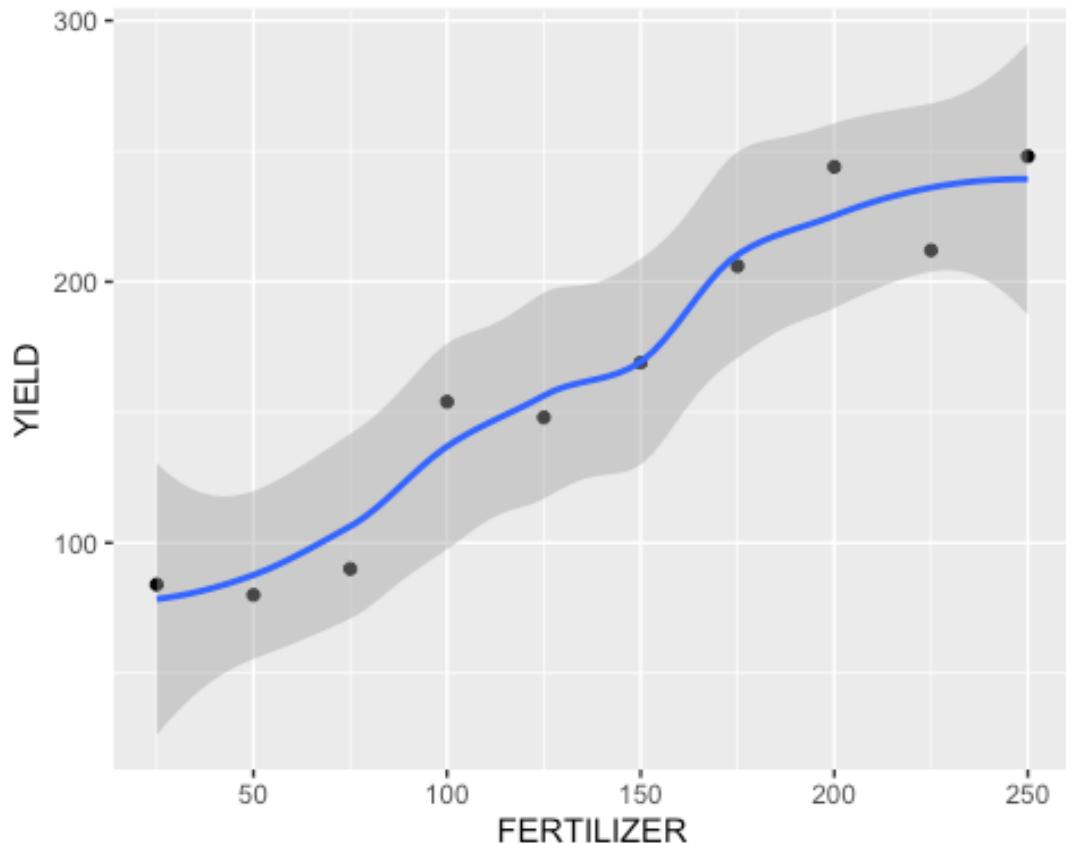
```
## .. YIELD = col_double()
## .. )
```

2.1. Exploratory data analysis

We are first gonna assume normality and homogeneity of variance

Diagnostic plots

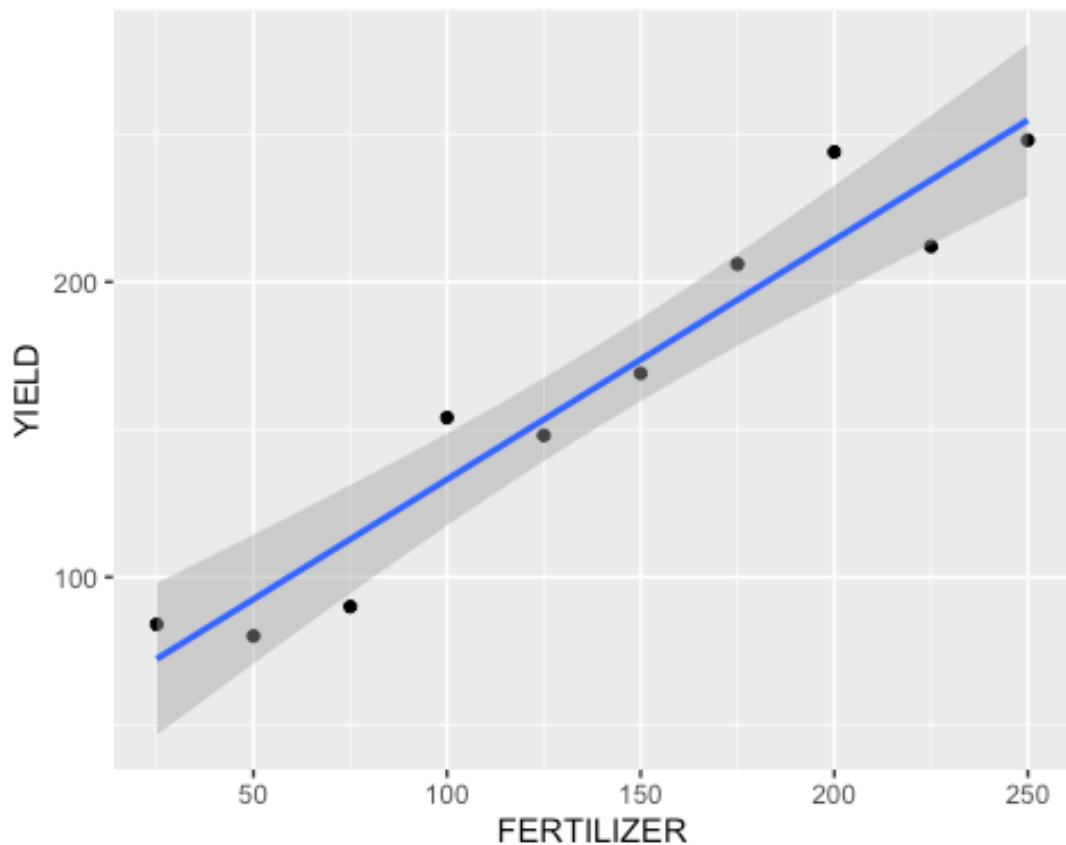
```
ggplot(fert, aes(y=YIELD, x=FERTILIZER)) +
  geom_point() +
  geom_smooth()
```



Linearity: We see the data points and a lower smoother (`#lm smoother`). It seems like the smoother sort of follows the trajectory of the data points / that each data point follows a normal distribution. That kind of suggests that a linear model might be suitable. If this wasn't the case, fitting a linear line might be an oversimplification. In this case, linearity looks fine.

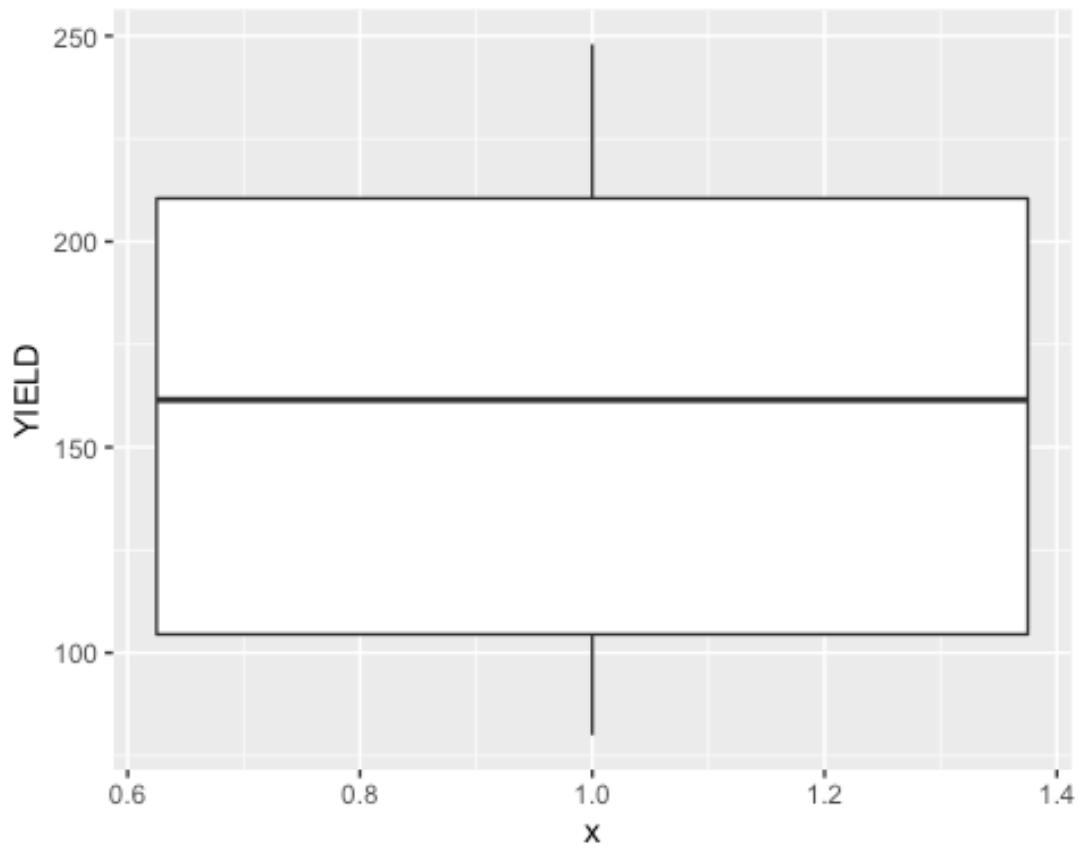
- Homogeneity of variance: There is no pattern amongst the noise (=the error bands). The bands seem homogeneous along the axes. The plot below will look closer into that:

```
ggplot(fert, aes(y=YIELD, x=FERTILIZER)) +
  geom_point() +
  geom_smooth(method='lm')
```



Homogeneity of variance: This lm plot shows that the error bands are more or less homoscedastic along the line.

```
ggplot(fert, aes(y=YIELD)) +  
  geom_boxplot(aes(x=1))
```



The

boxplot looks vaguely normal. We can't tell whether it is uniform, but it is clearly not skewed (=the upper and lower lines have an even length).

In summary, our assumptions do not seem to be violated. We could therefore use Least Squares, Linear Regression OR Maximum Likelihood (doesn't matter)

Fit the model

Model formula:

$$y_i \sim \mathcal{N}(\mu_i, \sigma^2) \quad \mu_i = \beta_0 + \beta_1 x_i$$

where y_i represents the i observed values, β_0 and β_1 represent the intercept and slope respectively, and σ^2 represents the estimated variance.

Each observation is drawn from a normal distribution, each with its own mean (eg. observation 1 was drawn from a normal distribution where y1 amount of fertiliser was used, and has a mean x1; observation 2 has a mean2, and so on).

Now let's think about fitting our model. Given that we can use Linear Least Squares Regression (lm), we will.

The function lm fits linear model by Least Squares Regression. We have to specify a formula, the intercept, and the predictor variable. We also specify what dataset to use to create the lm.

```
fert.lm<-lm(YIELD~1+FERTILIZER, data=fert)
# We specify the intercept (YIELD) and the slope is calculated by the
# predictor variable FERTILISER. We usually get rid of the ~1 symbol, which
# essentially indicates the intercept (see below)

fert.lm<-lm(YIELD~FERTILIZER, data=fert)
```

If we didn't want an intercept (do not do this), it would look like this

```
# fert.Lm<-Lm(YIELD~0+FERTILIZER, data=fert)
# fert.Lm<-Lm(YIELD~(-1)+FERTILIZER, data=fert)

# In this case, we would be placing the model on a single observation (0).
# And when we are fitting models, we want all observations to contribute to the
# model, and not to rely on a single observation,
```

To check all objects that we created, we write:

```
ls()
## [1] "fert"     "fert.lm"
```

So what is exactly fert.lm?

It is a linear model object. And we can have a look at what it contains.

```
class(fert.lm)
## [1] "lm"

To show all the information that the lm contains:
attributes(fert.lm)

## $names
##  [1] "coefficients"    "residuals"      "effects"       "rank"
##  [5] "fitted.values"   "assign"        "qr"            "df.residual"
##  [9] "xlevels"         "call"          "terms"         "model"
##
## $class
## [1] "lm"
```

We could have a look at each of those attributes

```
fert.lm$coefficients # The $ sign gets to components within an object
## (Intercept)  FERTILIZER
## 51.9333333  0.8113939
```

```
# Tells us the Intercept and the slope
# The equation would be:
# yield(gr)=51.93+0.81*fertiliser(gr)
```

To see all the summary values:

```
summary(fert.lm)

##
## Call:
## lm(formula = YIELD ~ FERTILIZER, data = fert)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -22.79 -11.07   -5.00   12.00   29.79
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 51.93333 12.97904  4.001 0.00394 ***
## FERTILIZER   0.81139   0.08367  9.697 1.07e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 19 on 8 degrees of freedom
## Multiple R-squared:  0.9216, Adjusted R-squared:  0.9118
## F-statistic: 94.04 on 1 and 8 DF,  p-value: 1.067e-05
```

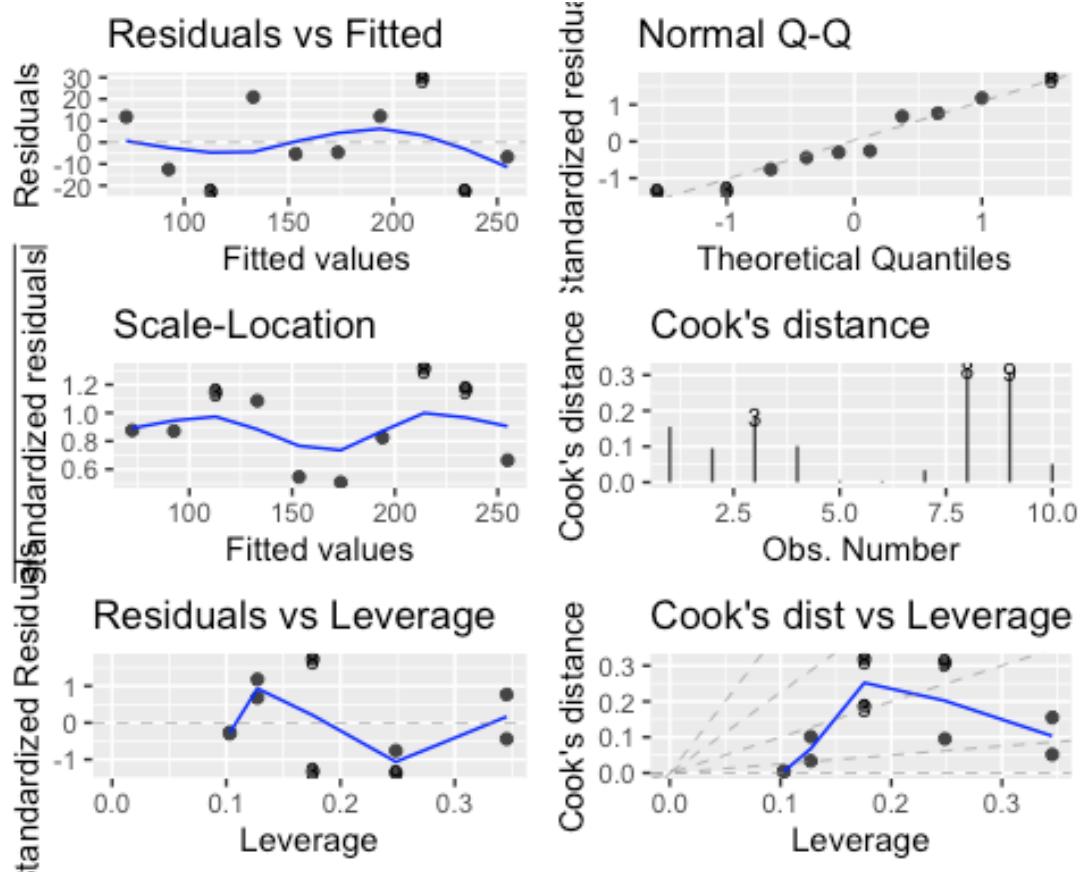
2.2. Model validation

autoplot

We will use the function “`autoplot`”, which gives a bunch of diagnostic plots. Autoplot adjusts to the type of model we are using (eg. if we use `lm`, it will check the assumptions for linear models)

```
autoplot(fert.lm,
         which=1:6, # Plots diagnostic plots 1-6 (by default it would only
show 4)
         ncol=2, # Displays the plots in 2 columns
         label.size=3) # Increases the size of the labels

## Warning: `arrange_()` is deprecated as of dplyr 0.7.0.
## Please use `arrange()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```



This plot tells us the following:

- **Residuals vs Fitted:** We don't want to see noise, patterns. We don't want to see some sort of curve, cause that would imply that we have fitted a straight line through non-straight data. In this case it looks like there is a little bit of a pattern, but that's because we got a small dataset. It essentially looks straight. We don't wanna see a funnel shape created by the dot points either, which would mean that the noise is getting bigger as the x axis (fitted values) increases = The test would have been violated. Residuals also show outliers: A very large residual is an outlier.
- **Normal Q-Q:** Compares the distribution of our data to what you'd expect if you'd follow the expected distribution (i.e. a Gaussian distribution). Murray says that a visual interpretation is enough for ecological data, and that the statistical tests (Shapiro Wilk, Levene test, etc) are usually too rigid, and sometimes even limit what we can realistically analyse (eg. the QQ plot looks fine, but the Shapiro says it's non-parametric -> Murray would rather follow the plot, especially in ecological studies, or we might use non-parametric stats, which might not be suitable)
- **Scale-Location:** It's also called Spread-Location plot. This plot shows if residuals are spread equally along the ranges of predictors. This is how you can check the assumption of equal variance (homoscedasticity). It's good if you see a horizontal line with equally (randomly) spread points

The remaining three plots show the influence of individual observations:

- **Residuals vs Leverage:** How much of an outlier is on the x axis -> The combination of Residuals and Leverage is Cooks distance, which is the influence of any given point. Cook's d has to be <0.8 (!) It provides us with a guide of how far the residuals need to be. In this case, it is 0.3,

so our observations are <that trigger. - Cook's distance - Cook's distance vs Leverage:
Another way of visualising the same problem.

Overall, all seems to be well on well. We do not seem to be violating any assumptions.

BUT there are other things we could look at:

influence.measures

Produces the Cook's distance for each of the data points. It will put a star next to the most influencial data pointsl If there is no star, ignore it!

```
influence.measures(fert.lm)

## Influence measures of
##   lm(formula = YIELD ~ FERTILIZER, data = fert) :
##
##      dfb.1_ dfb.FERT   dffit cov.r cook.d   hat inf
## 1    0.5391 -0.4561  0.5411 1.713 0.15503 0.345
## 2   -0.4149  0.3277 -0.4239 1.497 0.09527 0.248
## 3   -0.6009  0.4237 -0.6454 0.969 0.18608 0.176
## 4    0.3803 -0.2145  0.4634 1.022 0.10137 0.127
## 5   -0.0577  0.0163 -0.0949 1.424 0.00509 0.103
## 6   -0.0250 -0.0141 -0.0821 1.432 0.00382 0.103
## 7    0.0000  0.1159  0.2503 1.329 0.03374 0.127
## 8   -0.2193  0.6184  0.9419 0.623 0.31796 0.176
## 9    0.3285 -0.6486 -0.8390 1.022 0.30844 0.248
## 10   0.1512 -0.2559 -0.3035 1.900 0.05137 0.345  *
```

performance

We can also use the package “performance”, which is similar to autoplot

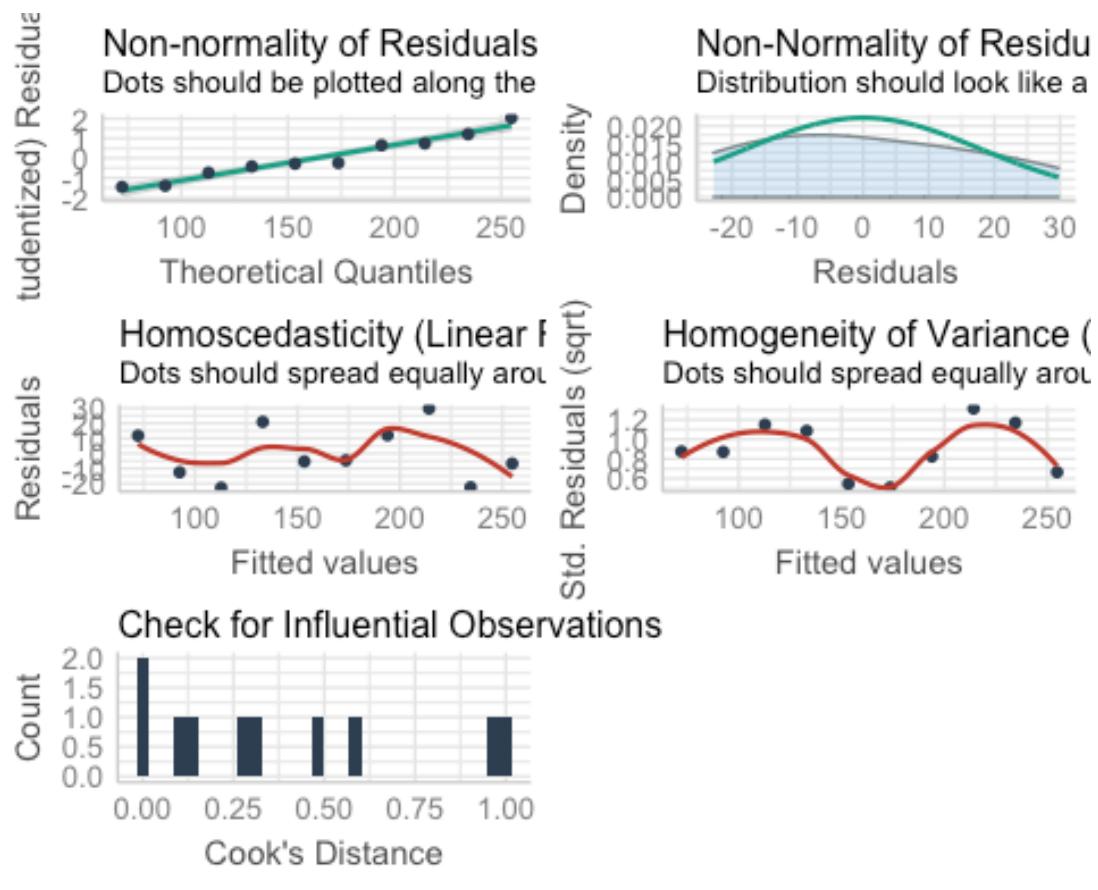
```
performance::check_model(fert.lm)

## Not enough model terms in the conditional part of the model to check for
## multicollinearity.

## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 10 rows containing missing values (geom_text_repel).
```



plot (first to the left): Normality plot. Shows the distribution of our data vs the theoretical distribution (Gaussian in this case) - Homoscedasticity: Our Residual plot - Cook's Distance plot: Ignore it, there are no values that are =1.00 It essentially shows the same as autoplot - Q-Q

Outlier detection

```
performance::check_outliers(fert.lm)
```

```
## OK: No outliers detected.
```

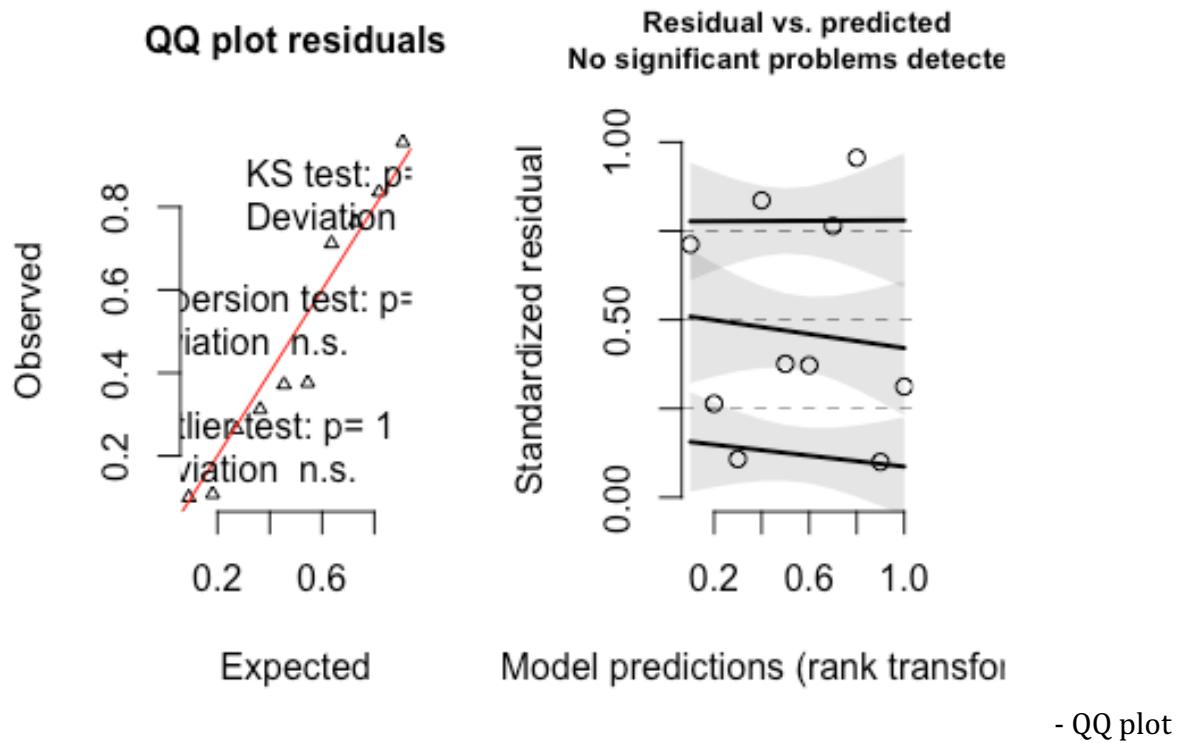
DHARMA (!)

BUT what we should rather use is the package called “DHARMA”.

DHARMA takes the residuals and makes the QQ plot that can be interpreted no matter which model we use, It does so by simulating residuals from our model, rather than calculating them directly. We can have a look at our simulated residuals:

```
fert.resid<-simulateResiduals(fert.lm, plot=TRUE)
```

DHARMA residual diagnostics

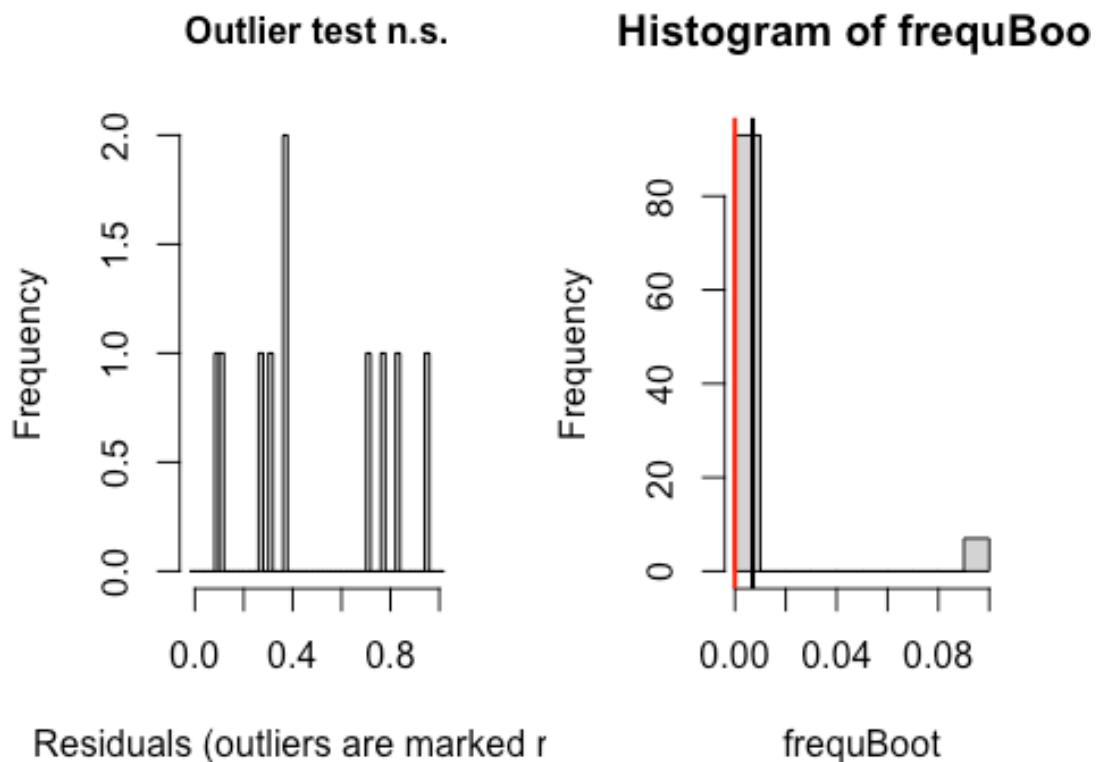


residuals: We want the points to follow the reference line, which they vaguely do.. Along with that plot, it produces a Smirnoff test for normality, which in this case is non-significant. Take this test with caution, cause it is a stricter test than the rules it is trying to protect. This test is a bit over-protective.. Ignore the Dispersion test for now . Outlier test: Tells us there is no outliers

- Residual vs. predicted: Ignore the lines in grey. Focus on the plots. The residuals are on the y axis, the expected values on the x. To identify the trends in the data, it fits not one, but THREE regression lines over our data. Ideally, you wanna see 3 straight lines, which we have (once you account for the uncertainty in them). If they weren't straight, they'd be red (warning sign). In this case, the residuals look great.

If we wanted to look at the Outlier test, for example, we can ask for it:

```
testOutliers(fert.resid)
```

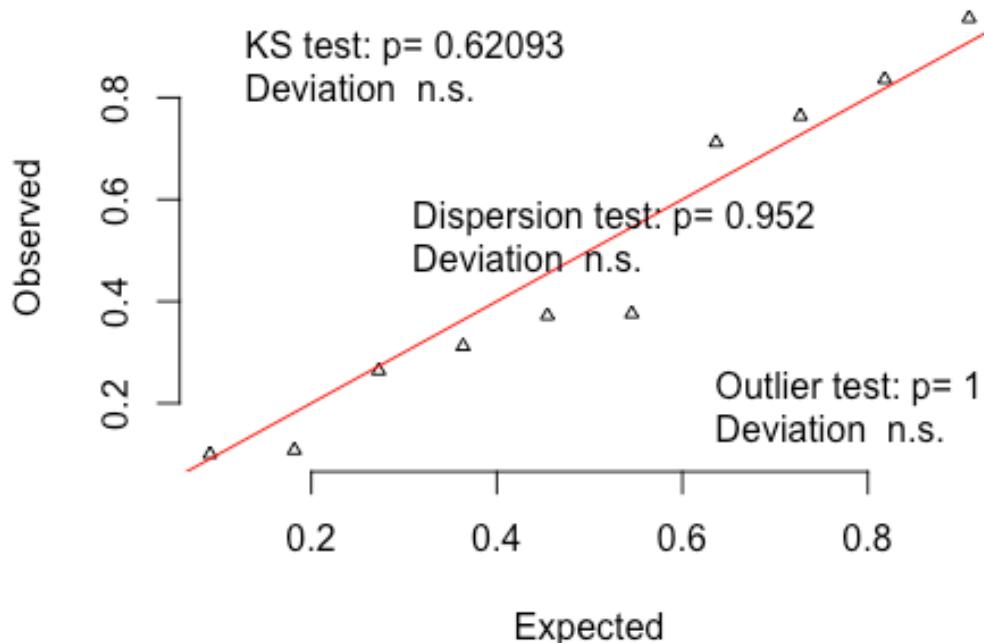


```
##  
## DHARMA bootstrapped outlier test  
##  
## data: fert.resid  
## outliers at both margin(s) = 0, observations = 10, p-value = 1  
## alternative hypothesis: two.sided  
## percent confidence interval:  
## 0.0 0.1  
## sample estimates:  
## outlier frequency (expected: 0.007 )  
## 0
```

Or have a look at the Smirnoff test for Normality

```
testUniformity(fert.resid)
```

QQ plot residuals



```
##  
## One-sample Kolmogorov-Smirnov test  
##  
## data: simulationOutput$scaledResiduals  
## D = 0.224, p-value = 0.6209  
## alternative hypothesis: two-sided
```

In summary, there are a bunch of tests you can perform from the DHARMA package.

We are now ready to test hypotheses from our model. Murray likes having a partial plot that shows what the model is expecting. This is helpful for interpreting any of the parameters.

2.3. Model investigation / hypothesis testing

Partial plots (sjPlot package)

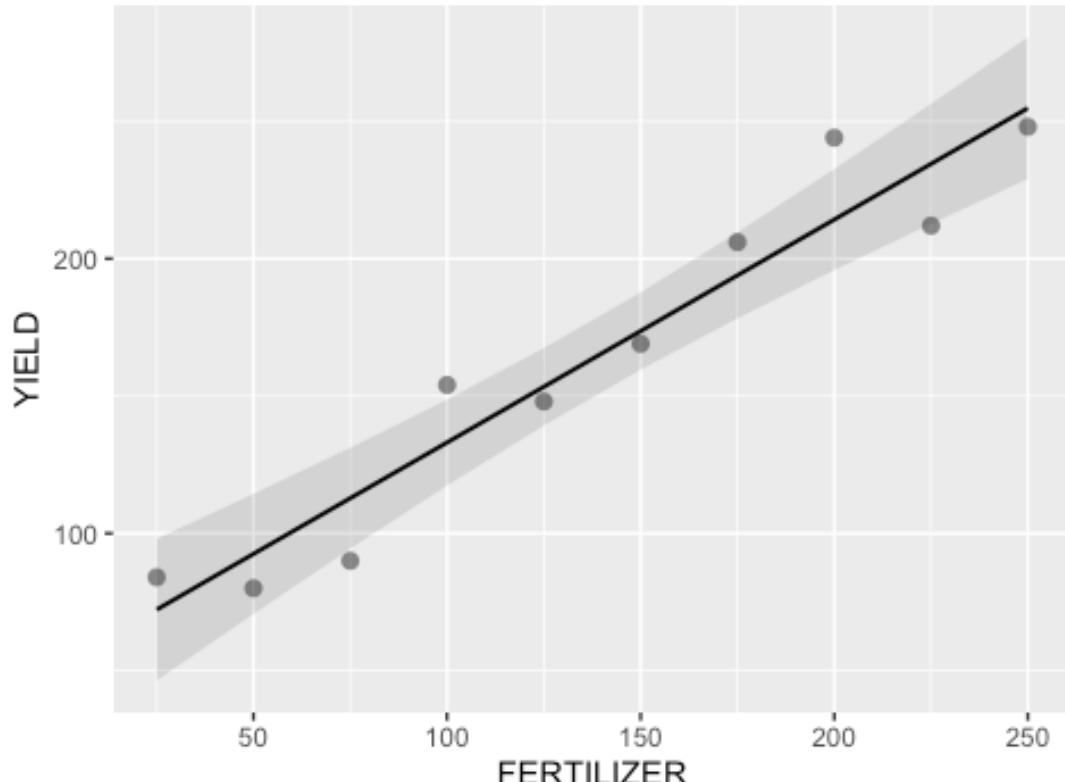
Before continuing, we have to check how our model might look like. Having some quick, dirty, "partial plots" helps with this.

Option 1

```
fert.lm %>%  
  plot_model(type="eff", show.data=TRUE)
```

```
## $FERTILIZER
```

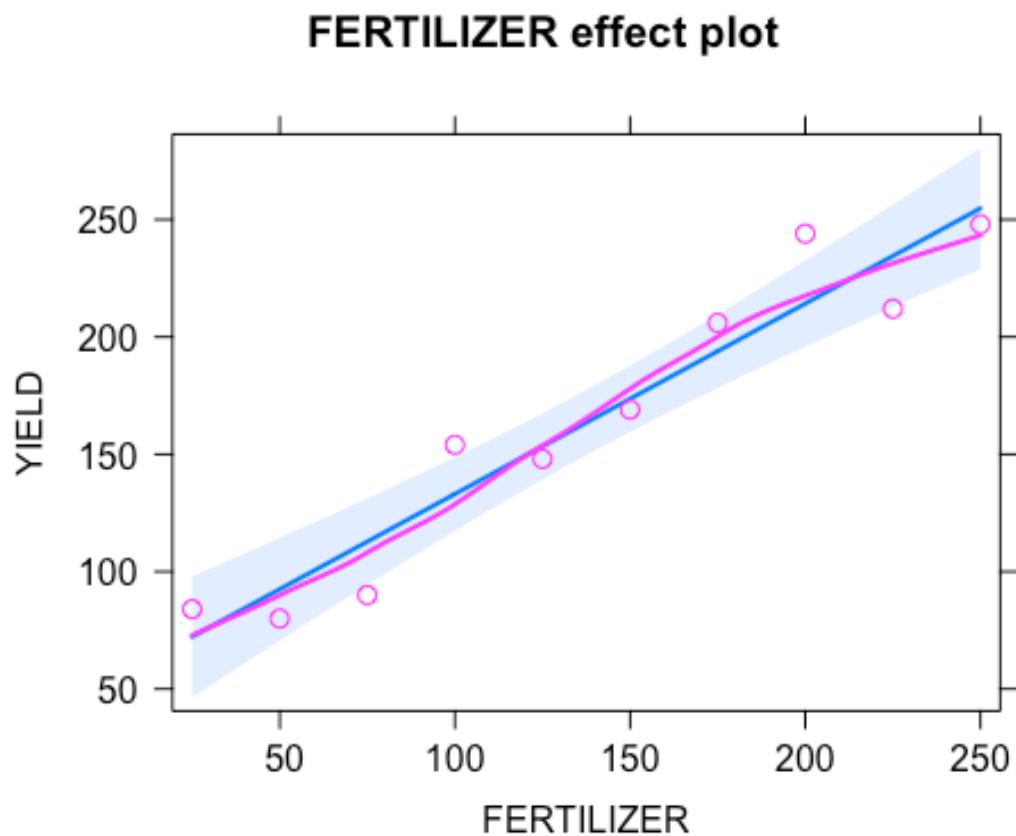
Predicted values of YIELD



Tells us what our predicted trend looks like from our model. It overlays the raw data. We expect a positive slope, and that the intercept will be <100. So we got some notion on what to expect.

Option 2:

```
plot(allEffects(fert.lm, residuals=TRUE))
```

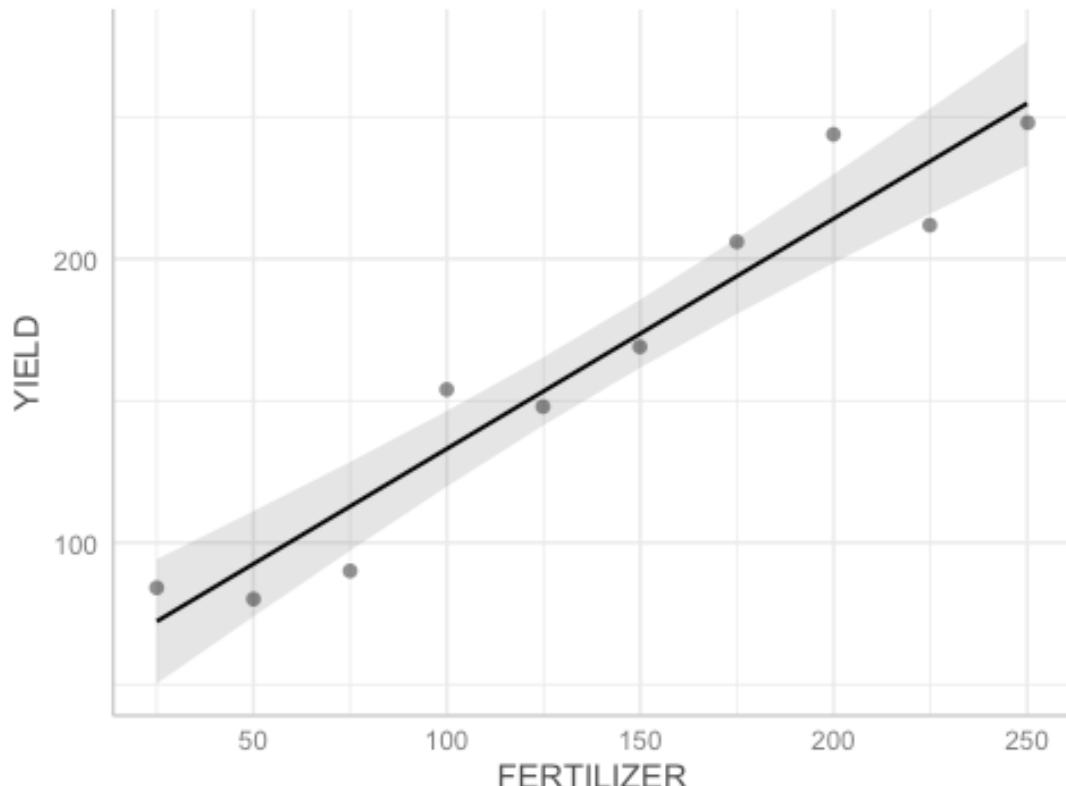


This plot does not use ggplot, and is therefore less flexible. To add the raw data, you tell it "residuals=TRUE". Otherwise, it shows the same as the above plot.

Option 3

```
fert.lm %>%  
  ggpredict() %>%  
  plot(add.data=TRUE)  
  
## $FERTILIZER
```

Predicted values of YIELD

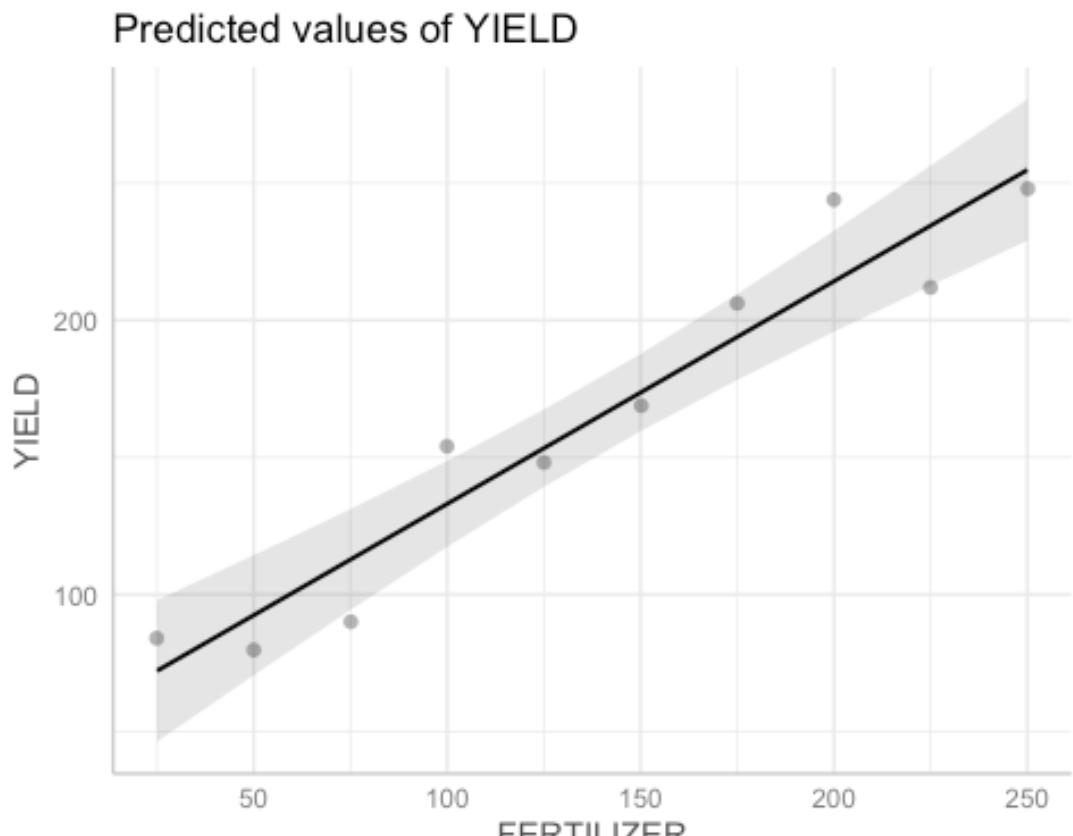


is = Option 1

This plot

Option 4

```
fert.lm %>%
  ggemmeans(~FERTILIZER) %>%
  plot(add.data=TRUE)
```



= as

Option 1 and 3

2.4. Model outputs

Model summary

```
summary(fert.lm)
```

```
##
## Call:
## lm(formula = YIELD ~ FERTILIZER, data = fert)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -22.79 -11.07  -5.00  12.00  29.79
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 51.93333  12.97904   4.001  0.00394 **
## FERTILIZER   0.81139   0.08367   9.697 1.07e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 19 on 8 degrees of freedom
```

```
## Multiple R-squared:  0.9216, Adjusted R-squared:  0.9118
## F-statistic: 94.04 on 1 and 8 DF,  p-value: 1.067e-05
```

Tells us all about the model (intercept, slope, and associated uncertainties (std.error), degrees of freedom, and the p-values ($\text{Pr}(>|t|)$))

Our conclusion indicates that there IS evidence of a relationship (p value). We treat that p-value as "Is it less than 0.05, or not?" Don't treat it as a measure of an effect (and forget about looking into if it is 0.04 or 0.03, 0.0687... just use it as a boundary line). The magnitude of an effect is the slope, (the value 0.811). What it says is that for each unit change in fertiliser concentrations, we get a 0.8 unit change in yield. And that is the magnitude of our effect.

The multiple R-sq is a measure of the relationship = What proportion of the variation can be explained by its relationship with x? In this case, 92% can be explained by the amount of fertiliser (x). It also in theory explains the complexity in our model. The more predictors there are in the model, the more it will be penalised, and the lower the R sq. BUT Murray says to ignore it as a complexity measure, and to NOT use it to compare models.

For a simple linear regression, the F-statistic is the t value sq. It does not tell us anything new, so ignore it.

DF: Complexity of our model

p-value: The probability of rejecting a zero intercept (ignore it)

Confidence intervals (95% CI - 2.5% on each side)

```
confint(fert.lm)
```

```
##              2.5 %    97.5 %
## (Intercept) 22.0036246 81.863042
## FERTILIZER   0.6184496  1.004338
```

Intercepts: THESE ARE IMPORTANT. They are like the p-values, but much better. If the confint does not include zero, it is = to a $p < 0.05$. So we can still say that there is a positive relationship. But it tells us even more about the magnitude of the effect: - Describes the implications of having a slope of 0.8 - Explains the implications of the slope being as low as 0.1, and as high as 1. A p-value won't tell you this.

Model summary - consistent across models

Every model you fit (lm, glm, lmm, etc) have a slightly different format on how they present their summary (summary(fert.lm)). They are kind of inconsistent. Instead, we can produce a tidy output that will be consistent across different types of models:

```
tidy(fert.lm, conf.int=TRUE)

## # A tibble: 2 x 7
##   term      estimate std.error statistic  p.value conf.low conf.high
##   <chr>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
```

```
## 1 (Intercept) 51.9 13.0 4.00 0.00394 22.0 81.9
## 2 FERTILIZER 0.811 0.0837 9.70 0.0000107 0.618 1.00
```

This is helpful when you are working with different models, and want to show them in a consistent manner. We got the same info as with the `summary(fert.lm)` command. We can take this one step further with Markdown:

```
tidy(fert.lm, conf.int=TRUE) %>% kable
```

term	estimate	std.error	statistic	p.value	conf.low	conf.high
(Intercept)	51.933333	12.979035	4.00132	0.003942	22.003624	81.86304
	3	2	5	5	6	2
FERTILIZE	0.8113939	0.0836704	9.69749	0.000010	0.6184496	1.004338
R			9	7		

Produces the table as if it was for a report. It is part of the package knitr.

Alternative table:

```
sjPlot::tab_model(fert.lm, show.se=TRUE, show.aic=TRUE)
```

```

YIELD
Predictors
Estimates
std. Error
CI
p
(Intercept)
51.93
12.98
22.00 – 81.86
0.004
FERTILIZER
0.81
0.08
0.62 – 1.00
<0.001
Observations
10
R2 / R2 adjusted
0.922 / 0.912
AIC
91.035
```

This is another version of a table.

2.5. Predictions

We already know our equation The equation would be:

$$\text{yield(gr)} = 51.93 + 0.811 * \text{fertiliser(gr)}$$

What is the difference btw Prediction intervals and Conf int? - Prediction Interval = What interval do I expect if I took a single observation using a fertiliser conc of 110? - Conf int: Predicts the AVERAGE of a lot of observations, so it will be smaller than the prediction interval. We usually focus on conf int, rather than on pred int. If we divide conf int / sigma (=residuals sq), it gives us the pred int.

Let's do some predictions: We have now established a dataset that predicts our y values.

Option 1

Let's predict it for a fertiliser conc of 110:

```
newdata=data.frame(FERTILIZER=110)

newdata

##   FERTILIZER
## 1      110
```

We can start with a function called "predict":

```
predict(fert.lm, newdata=newdata)

##       1
## 141.1867
```

We can also tell it to give us the 95% conf int

```
predict(fert.lm, newdata=newdata, interval='confidence')

##       fit      lwr      upr
## 1 141.1867 126.3506 156.0227
```

The function predict works well for simpler models, but however is crap for more complex models, so Murray does NOT recommend using it.

Option 2 - emmeans (!)

First, we create the new data of fertilizer=110

```
newdata=list(FERTILIZER=110)

emmeans(fert.lm, ~FERTILIZER, at=newdata)
```

```
##  FERTILIZER emmean    SE df lower.CL upper.CL
##      110     141 6.43   8      126      156
##
## Confidence level used: 0.95
```

The table has rounded the numbers, but they will be the same as the numbers produced in Option 1. This will work with any model. Murray highly recommends using emmeans.

2.6. Additional analyses

Accounting for uncertainty

What if someone wanted to know how much yield will be produced if we increased fertiliser conc from 100->200gr, we also have to account for UNCERTAINTY:

We first create a list with the desired fertiliser concentrations:

```
newdata=list(FERTILIZER=c(200,100)) # write the higher number first,
# otherwise we'll get a negative output
```

And then,

```
emmeans(fert.lm,pairwise~FERTILIZER,at=newdata) %>% confint

## $emmeans
##  FERTILIZER emmean    SE df lower.CL upper.CL
##      200     214 7.97   8      196      233
##      100     133 6.78   8      117      149
##
## Confidence level used: 0.95
##
## $contrasts
##  contrast estimate    SE df lower.CL upper.CL
##  200 - 100    81.1 8.37   8      61.8      100
##
## Confidence level used: 0.95
```

In this case, emmeans contrasts the 200 fert conc against 100 fert conc. increasing the fertiliser from 100 to 200 will increase the yield on avg by 81 units. It is still a sign change, as the CI do not include 0 (=p<0.05).

Let's create all conf int for each point across the line, at 100 points

```
fert_grid=with(fert, # with() tells R where to look for
               list(FERTILIZER=seq(min(FERTILIZER),
                                     max(FERTILIZER), # Sequence of num btw min
                                     and max fertiliser
                                     len=100)))
```

fert_grid

```
## $FERTILIZER
## [1] 25.00000 27.27273 29.54545 31.81818 34.09091 36.36364
38.63636
## [8] 40.90909 43.18182 45.45455 47.72727 50.00000 52.27273
54.54545
## [15] 56.81818 59.09091 61.36364 63.63636 65.90909 68.18182
70.45455
## [22] 72.72727 75.00000 77.27273 79.54545 81.81818 84.09091
86.36364
## [29] 88.63636 90.90909 93.18182 95.45455 97.72727 100.00000
102.27273
## [36] 104.54545 106.81818 109.09091 111.36364 113.63636 115.90909
118.18182
## [43] 120.45455 122.72727 125.00000 127.27273 129.54545 131.81818
134.09091
## [50] 136.36364 138.63636 140.90909 143.18182 145.45455 147.72727
150.00000
## [57] 152.27273 154.54545 156.81818 159.09091 161.36364 163.63636
165.90909
## [64] 168.18182 170.45455 172.72727 175.00000 177.27273 179.54545
181.81818
## [71] 184.09091 186.36364 188.63636 190.90909 193.18182 195.45455
197.72727
## [78] 200.00000 202.27273 204.54545 206.81818 209.09091 211.36364
213.63636
## [85] 215.90909 218.18182 220.45455 222.72727 225.00000 227.27273
229.54545
## [92] 231.81818 234.09091 236.36364 238.63636 240.90909 243.18182
245.45455
## [99] 247.72727 250.00000
```

If we gave it a value of 1000, it will predict a number. However, we SHOULD NOT predict outside the data that we have, as the model might change, without us knowing it.

```
newdata=emmeans(fert.lm, # predict values for our lm fert.lm
                ~FERTILIZER, # for the variable FERTILIZER
                at=fert_grid) %>% # at the specified 100 data points that we
specified previously
as.data.frame # and save it as data frame

newdata

## # FERTILIZER emmean      SE df lower.CL upper.CL
## 1 25.00000 72.21818 11.166947 8 46.46715 97.96921
## 2 27.27273 74.06226 11.007132 8 48.67977 99.44475
## 3 29.54545 75.90634 10.848295 8 50.89012 100.92255
## 4 31.81818 77.75041 10.690481 8 53.09812 102.40271
## 5 34.09091 79.59449 10.533736 8 55.30365 103.88533
## 6 36.36364 81.43857 10.378107 8 57.50661 105.37053
## 7 38.63636 83.28264 10.223647 8 59.70687 106.85842
```

```

## 8   40.90909 85.12672 10.070408 8   61.90432 108.34913
## 9   43.18182 86.97080 9.918448 8   64.09882 109.84278
## 10  45.45455 88.81488 9.767826 8   66.29023 111.33952
## 11  47.72727 90.65895 9.618605 8   68.47841 112.83950
## 12  50.00000 92.50303 9.470850 8   70.66321 114.34285
## 13  52.27273 94.34711 9.324633 8   72.84447 115.84975
## 14  54.54545 96.19118 9.180026 8   75.02201 117.36036
## 15  56.81818 98.03526 9.037107 8   77.19566 118.87487
## 16  59.09091 99.87934 8.895956 8   79.36523 120.39345
## 17  61.36364 101.72342 8.756660 8   81.53052 121.91631
## 18  63.63636 103.56749 8.619309 8   83.69133 123.44365
## 19  65.90909 105.41157 8.483996 8   85.84744 124.97570
## 20  68.18182 107.25565 8.350821 8   87.99862 126.51268
## 21  70.45455 109.09972 8.219888 8   90.14463 128.05482
## 22  72.72727 110.94380 8.091306 8   92.28522 129.60239
## 23  75.00000 112.78788 7.965188 8   94.42012 131.15564
## 24  77.27273 114.63196 7.841654 8   96.54907 132.71484
## 25  79.54545 116.47603 7.720827 8   98.67177 134.28029
## 26  81.81818 118.32011 7.602837 8   100.78794 135.85228
## 27  84.09091 120.16419 7.487817 8   102.89725 137.43113
## 28  86.36364 122.00826 7.375907 8   104.99939 139.01714
## 29  88.63636 123.85234 7.267251 8   107.09403 140.61065
## 30  90.90909 125.69642 7.161995 8   109.18083 142.21201
## 31  93.18182 127.54050 7.060293 8   111.25943 143.82156
## 32  95.45455 129.38457 6.962301 8   113.32948 145.43967
## 33  97.72727 131.22865 6.868176 8   115.39061 147.06669
## 34  100.00000 133.07273 6.778080 8   117.44245 148.70301
## 35  102.27273 134.91680 6.692176 8   119.48462 150.34899
## 36  104.54545 136.76088 6.610628 8   121.51675 152.00502
## 37  106.81818 138.60496 6.533597 8   123.53846 153.67146
## 38  109.09091 140.44904 6.461247 8   125.54937 155.34870
## 39  111.36364 142.29311 6.393735 8   127.54913 157.03709
## 40  113.63636 144.13719 6.331217 8   129.53738 158.73700
## 41  115.90909 145.98127 6.273842 8   131.51376 160.44877
## 42  118.18182 147.82534 6.221752 8   133.47796 162.17273
## 43  120.45455 149.66942 6.175081 8   135.42966 163.90918
## 44  122.72727 151.51350 6.133952 8   137.36858 165.65842
## 45  125.00000 153.35758 6.098479 8   139.29446 167.42069
## 46  127.27273 155.20165 6.068759 8   141.20707 169.19624
## 47  129.54545 157.04573 6.044878 8   143.10622 170.98524
## 48  131.81818 158.88981 6.026905 8   144.99174 172.78787
## 49  134.09091 160.73388 6.014893 8   146.86352 174.60425
## 50  136.36364 162.57796 6.008878 8   148.72146 176.43446
## 51  138.63636 164.42204 6.008878 8   150.56554 178.27854
## 52  140.90909 166.26612 6.014893 8   152.39575 180.13648
## 53  143.18182 168.11019 6.026905 8   154.21213 182.00826
## 54  145.45455 169.95427 6.044878 8   156.01476 183.89378
## 55  147.72727 171.79835 6.068759 8   157.80376 185.79293
## 56  150.00000 173.64242 6.098479 8   159.57931 187.70554
## 57  152.27273 175.48650 6.133952 8   161.34158 189.63142

```

```

## 58 154.54545 177.33058 6.175081 8 163.09082 191.57034
## 59 156.81818 179.17466 6.221752 8 164.82727 193.52204
## 60 159.09091 181.01873 6.273842 8 166.55123 195.48624
## 61 161.36364 182.86281 6.331217 8 168.26300 197.46262
## 62 163.63636 184.70689 6.393735 8 169.96291 199.45087
## 63 165.90909 186.55096 6.461247 8 171.65130 201.45063
## 64 168.18182 188.39504 6.533597 8 173.32854 203.46154
## 65 170.45455 190.23912 6.610628 8 174.99498 205.48325
## 66 172.72727 192.08320 6.692176 8 176.65101 207.51538
## 67 175.00000 193.92727 6.778080 8 178.29699 209.55755
## 68 177.27273 195.77135 6.868176 8 179.93331 211.60939
## 69 179.54545 197.61543 6.962301 8 181.56033 213.67052
## 70 181.81818 199.45950 7.060293 8 183.17844 215.74057
## 71 184.09091 201.30358 7.161995 8 184.78799 217.81917
## 72 186.36364 203.14766 7.267251 8 186.38935 219.90597
## 73 188.63636 204.99174 7.375907 8 187.98286 222.00061
## 74 190.90909 206.83581 7.487817 8 189.56887 224.10275
## 75 193.18182 208.67989 7.602837 8 191.14772 226.21206
## 76 195.45455 210.52397 7.720827 8 192.71971 228.32823
## 77 197.72727 212.36804 7.841654 8 194.28516 230.45093
## 78 200.00000 214.21212 7.965188 8 195.84436 232.57988
## 79 202.27273 216.05620 8.091306 8 197.39761 234.71478
## 80 204.54545 217.90028 8.219888 8 198.94518 236.85537
## 81 206.81818 219.74435 8.350821 8 200.48732 239.00138
## 82 209.09091 221.58843 8.483996 8 202.02430 241.15256
## 83 211.36364 223.43251 8.619309 8 203.55635 243.30867
## 84 213.63636 225.27658 8.756660 8 205.08369 245.46948
## 85 215.90909 227.12066 8.895956 8 206.60655 247.63477
## 86 218.18182 228.96474 9.037107 8 208.12513 249.80434
## 87 220.45455 230.80882 9.180026 8 209.63964 251.97799
## 88 222.72727 232.65289 9.324633 8 211.15025 254.15553
## 89 225.00000 234.49697 9.470850 8 212.65715 256.33679
## 90 227.27273 236.34105 9.618605 8 214.16050 258.52159
## 91 229.54545 238.18512 9.767826 8 215.66048 260.70977
## 92 231.81818 240.02920 9.918448 8 217.15722 262.90118
## 93 234.09091 241.87328 10.070408 8 218.65087 265.09568
## 94 236.36364 243.71736 10.223647 8 220.14158 267.29313
## 95 238.63636 245.56143 10.378107 8 221.62947 269.49339
## 96 240.90909 247.40551 10.533736 8 223.11467 271.69635
## 97 243.18182 249.24959 10.690481 8 224.59729 273.90188
## 98 245.45455 251.09366 10.848295 8 226.07745 276.10988
## 99 247.72727 252.93774 11.007132 8 227.55525 278.32023
## 100 250.00000 254.78182 11.166947 8 229.03079 280.53285

```

The lowest and highest values shown in the table above are our min and max CI.

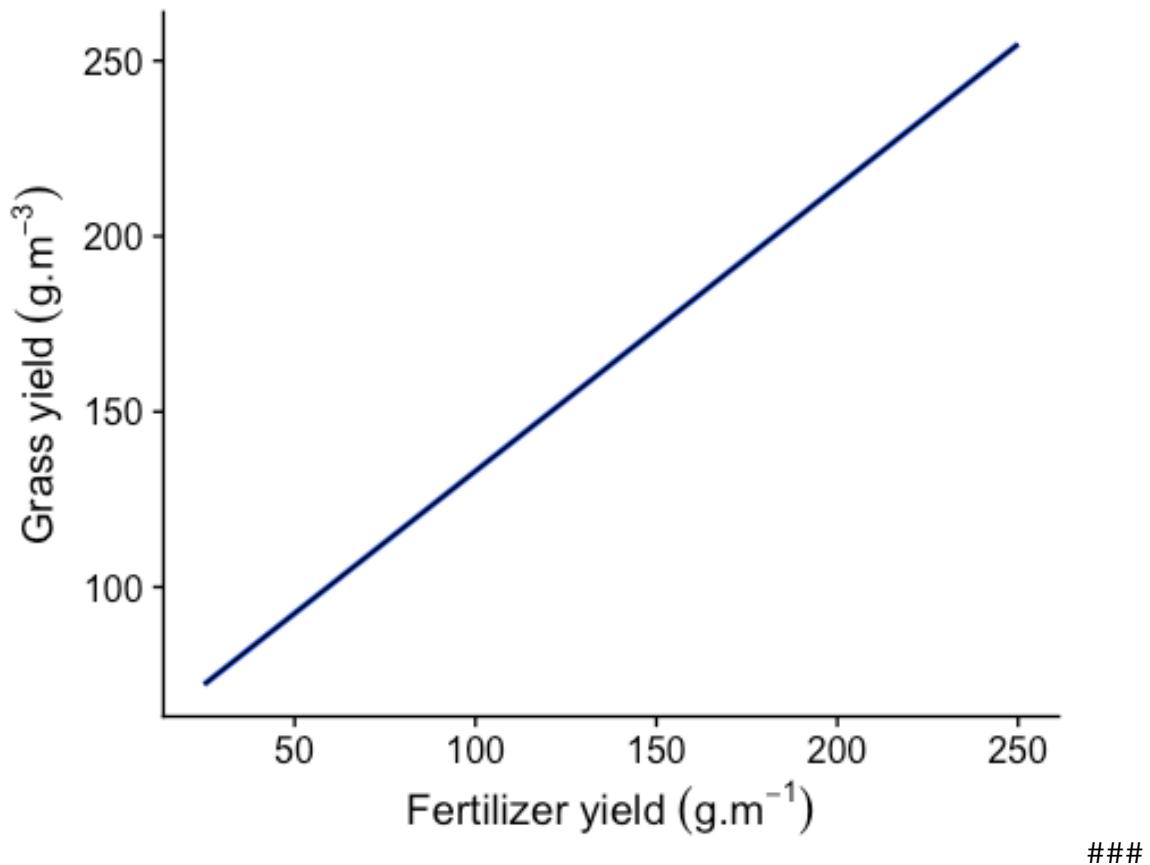
2.7. Summary figures

Option 1: Showing the relationship btw FERTILIZER and yield.

(= emmeans on y axis, and FERTILIZER on the x axis)

```
newdata %>%
  ggplot(aes(x=FERTILIZER,y=emmmean))+
  geom_smooth() # Not really needed, as the predicted values have no error
bands, really. In this case it's the blue line
  geom_line()+
  scale_x_continuous(expression(Fertilizer~yield~(g.m^-1)))+
  scale_y_continuous(expression(Grass~yield~(g.m^-3)))+
  #ylab("Predicted yield (gr)")+ # This would have been the alternative,
simplified y label
  #xLab("Fertilizer (gr)")+ # This would have been the alternative,
simplified x label
  theme_cowplot()

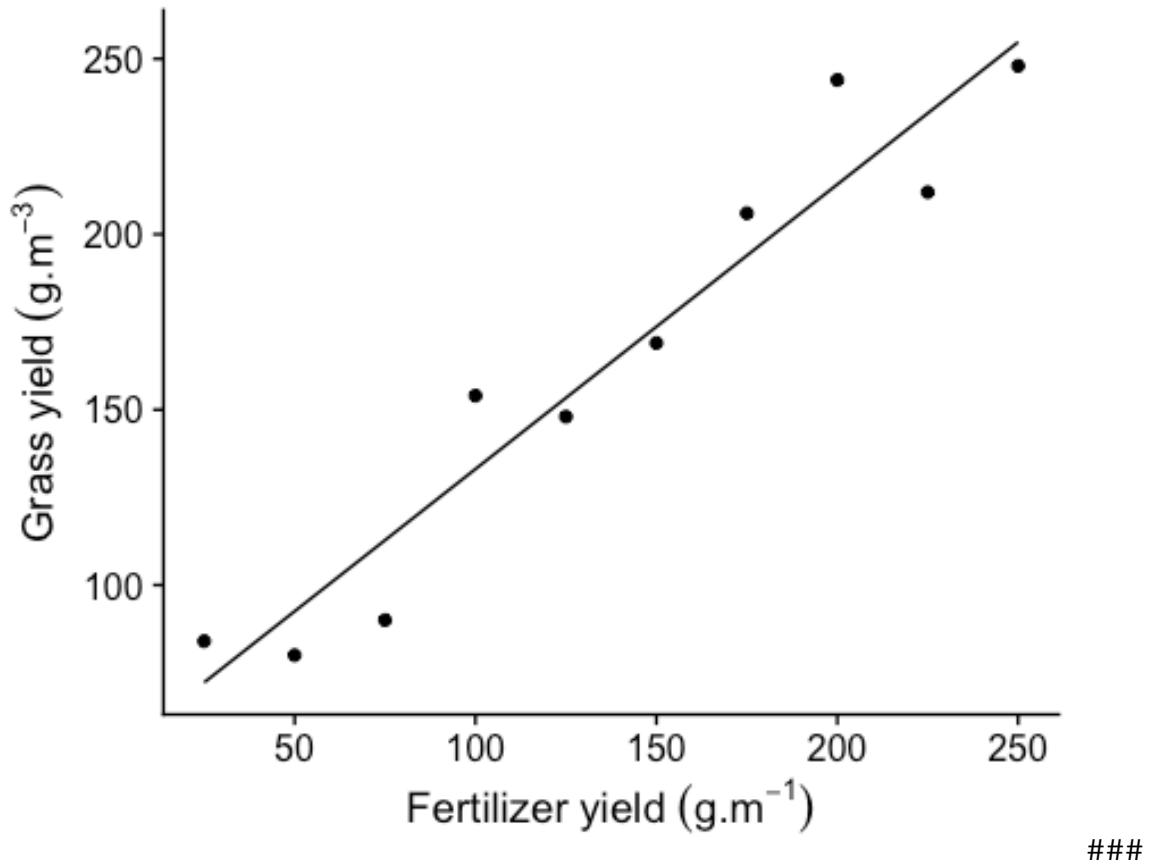
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Option 2: With the raw data

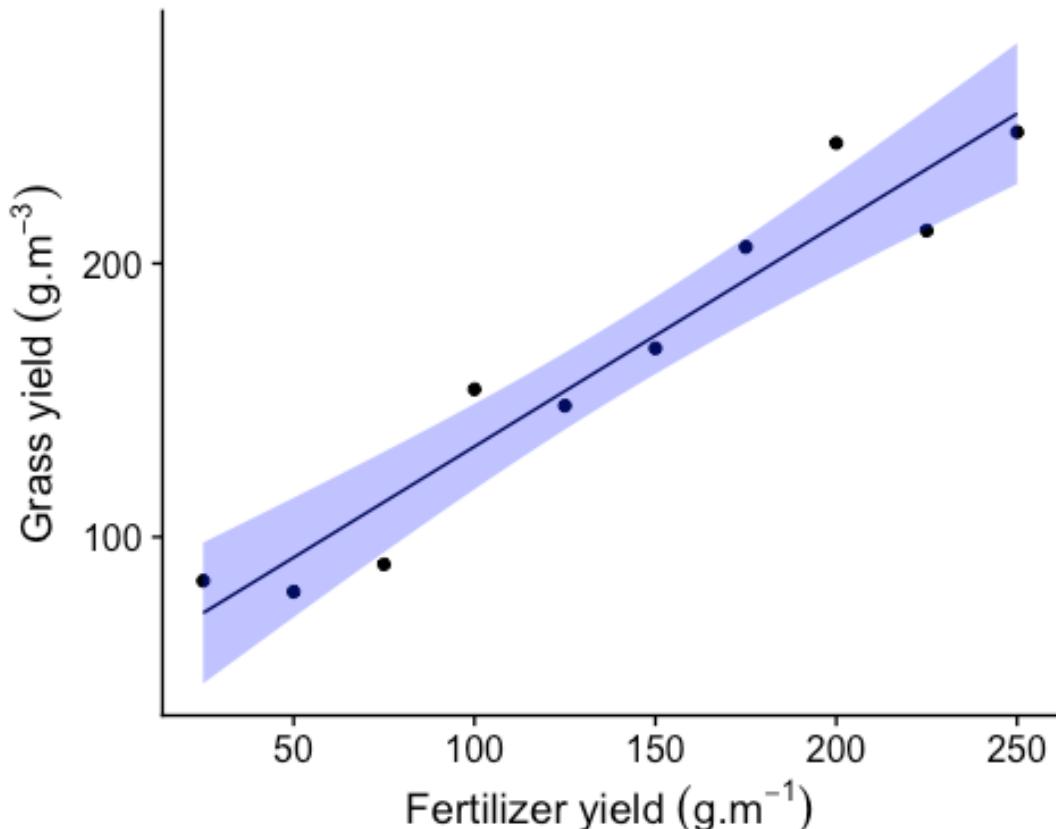
###

```
newdata %>%
  ggplot(aes(x=FERTILIZER,y=emmmean))+
  geom_line()+
  geom_point(data=fert,aes(y=YIELD))+ # plots the RAW data points (YIELD),
from the dataset fert
  scale_x_continuous(expression(Fertilizer~yield~(g.m^-1)))+ # Label names
for special characters
  scale_y_continuous(expression(Grass~yield~(g.m^-3)))+ # Label names for
special characters
  theme_cowplot()
```



Option 3: With raw data and conf int ribbon

```
newdata %>%
  ggplot(aes(x=FERTILIZER,y=emmmean))+
  geom_line()+
  geom_point(data=fert,aes(y=YIELD))+ # plots the RAW data points (YIELD),
from the dataset fert
  geom_ribbon(aes(ymin=lower.CL,ymax=upper.CL), fill="blue", alpha=0.3)+
  scale_x_continuous(expression(Fertilizer~yield~(g.m^-1)))+
  scale_y_continuous(expression(Grass~yield~(g.m^-3)))+
  theme_cowplot()
```



Essentially, the plots show us that we got a trend. Plotting the raw data on top is ok when you have a single predictor, and the model is pretty straightforward. BUT when you are creating lmm, and models with several predictors, random effects... you might wanna plot the partial data/residuals.

2.8. References

3- EXAMPLE 2 - BINOMIAL DISTRIBUTION

Scenario

@Polis-1998-490 were interested in modelling the presence/absence of lizards (*Uta* sp.) against the perimeter to area ratio of 19 islands in the Gulf of California.



In summary, you wanna test on which islands you find a specific lizard. The researchers measured the perimeter/area ratio. The higher the perimeter/area ratio, then clearer the longer the shape. And the researchers also measured the presence/absence of lizards → This already tells us that we are working with a BINOMIAL DISTRIBUTION.

Format of polis.csv data file

ISLAND	RATIO	PA
Bota	15.41	1
Cabeza	5.63	1
Cerraja	25.92	1
Coronadito	15.17	0
..
ISLAND	Categorical listing of the name of the 19 islands used - variable not used in analysis.	
RATIO	Ratio of perimeter to area of the island.	
PA	Presence (1) or absence (0) of <i>Uta</i> lizards on island.	

The aim of the analysis is to investigate the relationship between island parameter to area ratio and the presence/absence of Uta lizards.

Read in the data

```
polis = read_csv('data/polis.csv', trim_ws=TRUE) # Polis is the researcher
# Looking at lizards

## Parsed with column specification:
## cols(
##   ISLAND = col_character(),
##   RATIO = col_double(),
##   PA = col_double()
## )
```

Let's have a quick glimpse at the data

```
glimpse(polis)

## Rows: 19
## Columns: 3
## $ ISLAND <chr> "Bota", "Cabeza", "Cerraja", "Coronadito", "Flecha",
```

```

"Gemelose...
## $ RATIO  <dbl> 15.41, 5.63, 25.92, 15.17, 13.04, 18.85, 30.95, 22.87,
12.01, ...
## $ PA      <dbl> 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1
head(polis)

## # A tibble: 6 x 3
##   ISLAND     RATIO     PA
##   <chr>     <dbl> <dbl>
## 1 Bota      15.4     1
## 2 Cabeza    5.63     1
## 3 Cerraja   25.9     1
## 4 Coronadito 15.2     0
## 5 Flecha    13.0     1
## 6 Gemelose  18.8     0

str(polis)

### tibble [19 × 3] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##$ ISLAND: chr [1:19] "Bota" "Cabeza" "Cerraja" "Coronadito" ...
##$ RATIO : num [1:19] 15.41 5.63 25.92 15.17 13.04 ...
##$ PA   : num [1:19] 1 1 1 0 1 0 0 0 0 1 ...
## - attr(*, "spec")=
##   .. cols(
##     .. ISLAND = col_character(),
##     .. RATIO = col_double(),
##     .. PA = col_double()
##   .. )

```

We see the perimeter/area ratio of the island (RATIO), and the presence/abscence (PA) of the lizard. And it seems like there is a relation btw the two (see table above, head(polis))

Because PA is composed by 0s and 1s, there is no way we can do a stat. test.

3.1. Exploratory data analysis

Model formula:

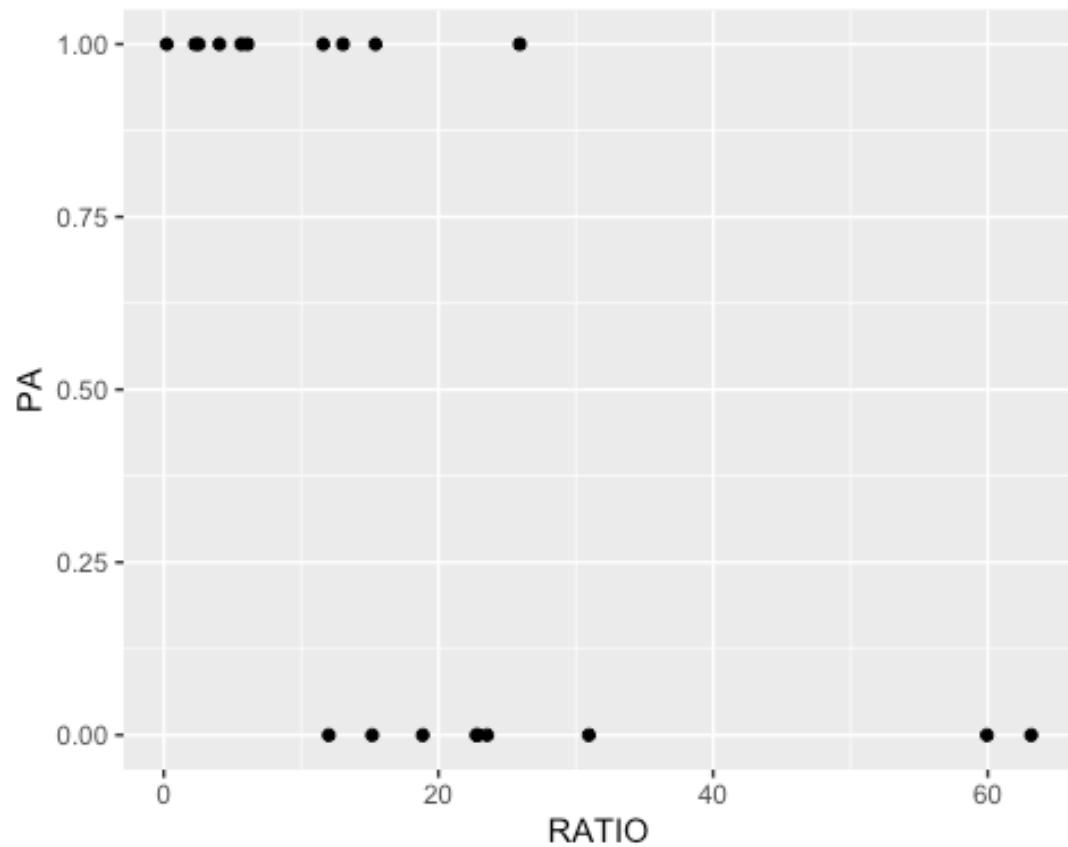
$$\text{y}_i \sim \text{Bin}(n, p_i) \ln(\text{left}(\frac{p_i}{1-p_i})) = \beta_0 + \beta_1 x_i$$

where y_i represents the i observed values, n represents the number of trials (in the case of logistic, this is always 1), p_i represents the probability of lizards being present in the i^{th} poluation, and β_0 and β_1 represent the intercept and slope respectively.

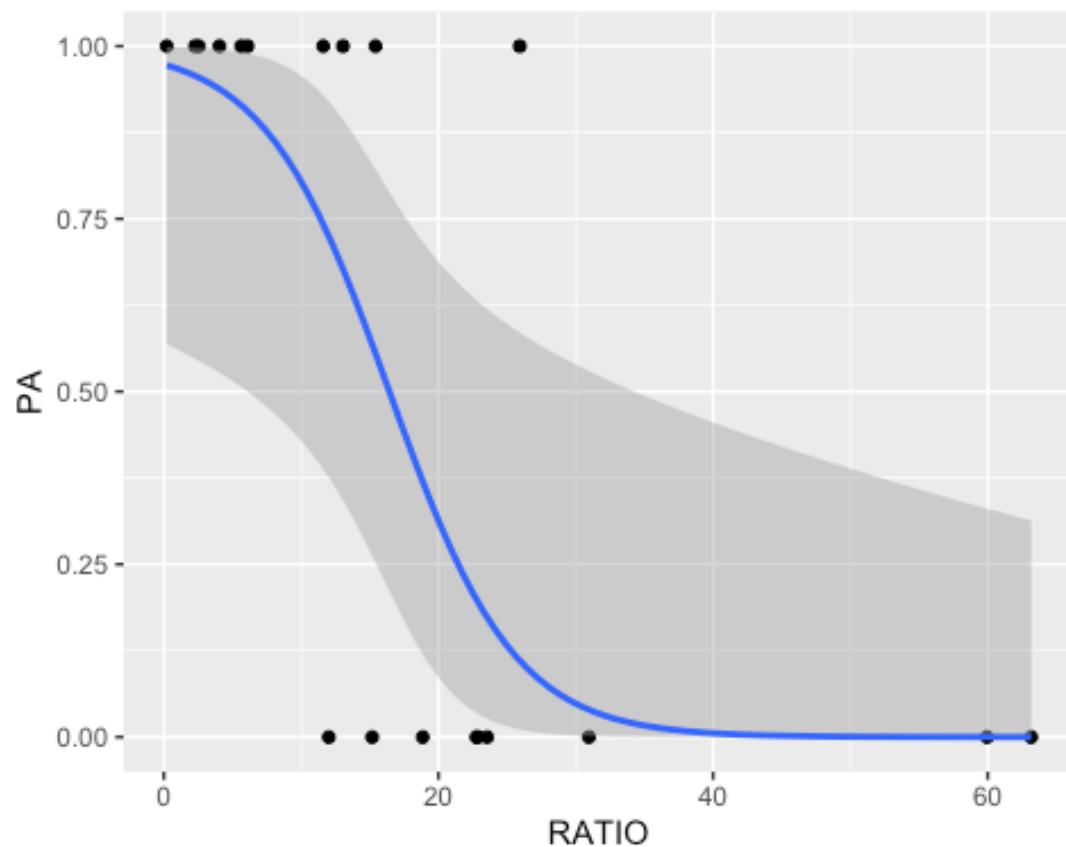
```

ggplot(polis, aes(y=PA, x=RATIO))+
  geom_point()

```



```
ggplot(polis, aes(y=PA, x=RATIO))+
  geom_point()+
  geom_smooth(method='glm', formula=y~x,
              method.args=list(family='binomial'))
```



It looks

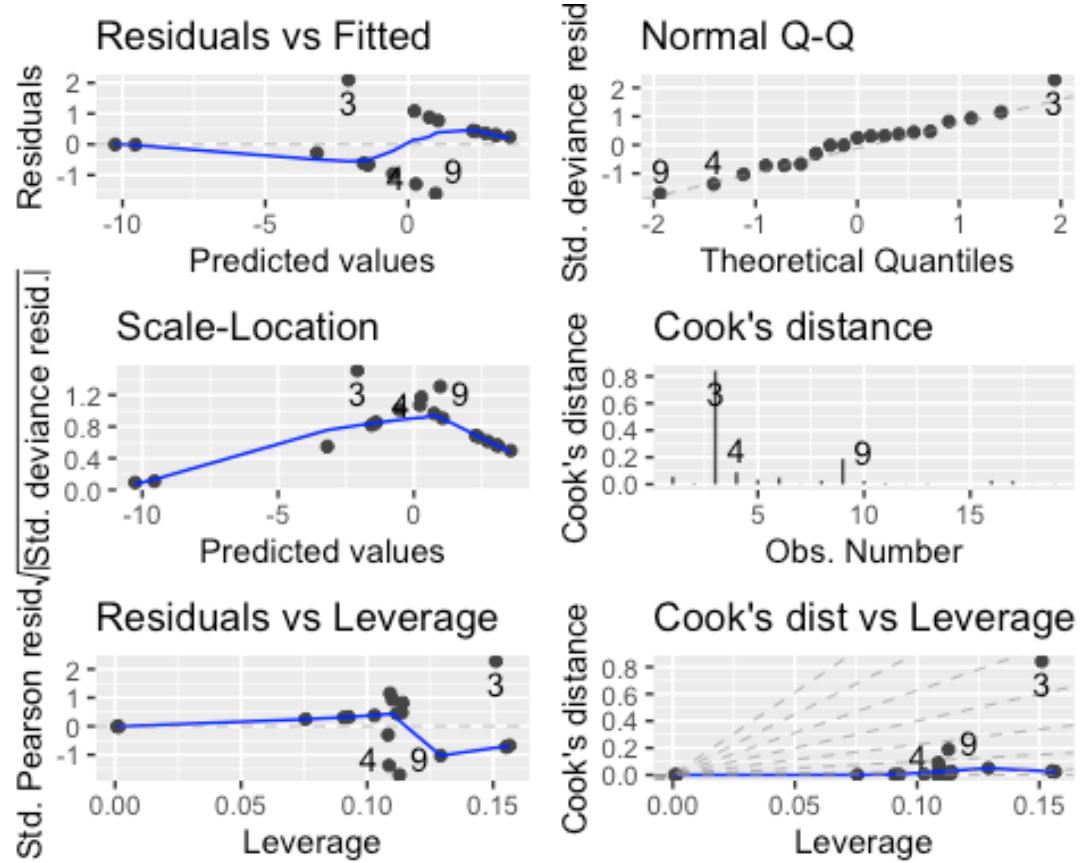
like the higher the ratio, the lower the presence

3.2. Fit the model

```
polis.glm<-glm(PA~RATIO,family=binomial(link="logit"),data=polis) # We  
nominate the Link of the expected values to the model predictor (to map the  
values to a scale of 1; Logit = Log odds ratio scale, using maximum  
Likelihood)
```

3.3. Model validation

```
autoplot  
autoplot(polis.glm,which=1:6,label.repel=TRUE)
```



These plots cannot be interpreted, except from the Normal Q-Q plot, which compares our model to a binomial distribution (what we nominated it to). We see that our model follows the line, which indicates that it actually IS a binomial distribution. - The Cook's distance shows that Observation 3 is quite influential and might be an outlier.

Let's have a quick look at that observation

```
head(polis)
```

```
## # A tibble: 6 x 3
##   ISLAND      RATIO     PA
##   <chr>      <dbl>  <dbl>
## 1 Bota       15.4    1
## 2 Cabeza     5.63   1
## 3 Cerraja    25.92   1
## 4 Coronadito 15.2    0
## 5 Flecha     13.0    1
## 6 Gemelose   18.8    0
```

Observation 3 (Cerraja) has a ratio of 25.92, and is thus the highest value in our data. The plot suggests that it is an outlier. Let's have a look at our plotted observation nr. 3:

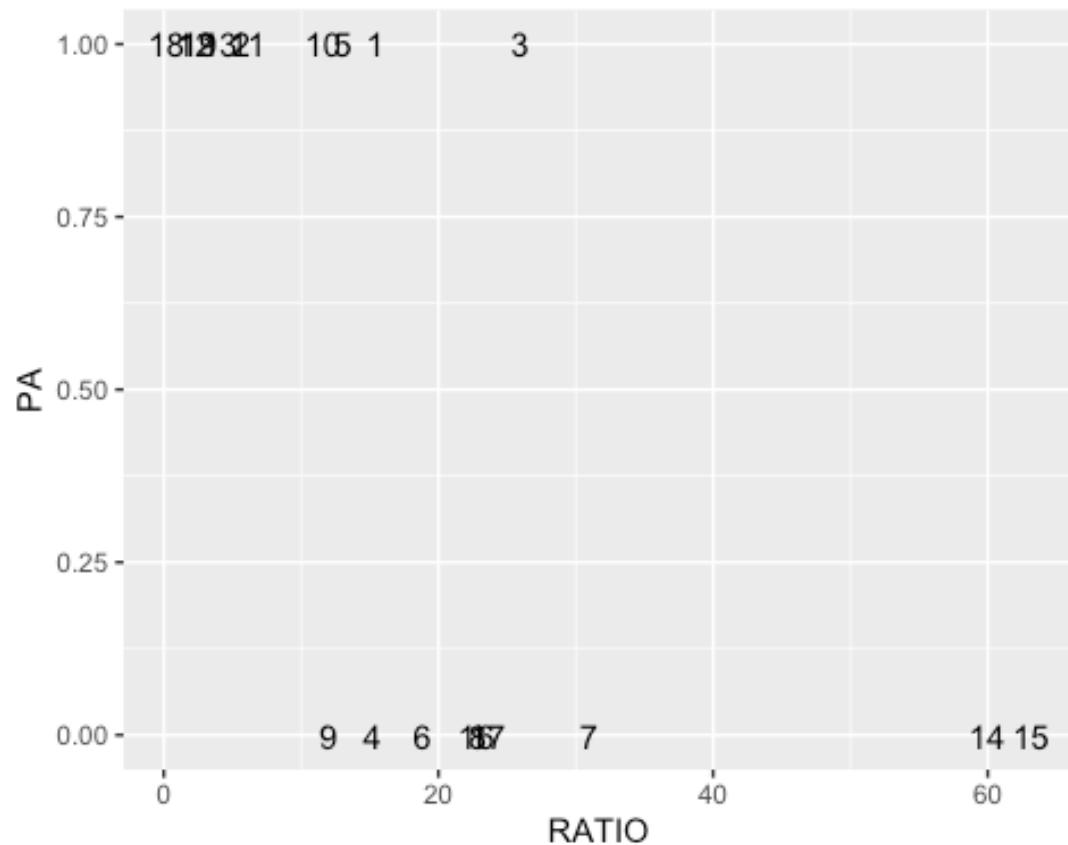
Let's first name our observations

```
polis %>%
  mutate(n=1:n()) # Adds a column to polis, with labeled rows

## # A tibble: 19 x 4
##   ISLAND    RATIO     PA     n
##   <chr>     <dbl>   <dbl> <int>
## 1 Bota      15.4     1     1
## 2 Cabeza    5.63    1     2
## 3 Cerraja   25.9     1     3
## 4 Coronadito 15.2     0     4
## 5 Flecha    13.0     1     5
## 6 Gemelose  18.8     0     6
## 7 Gemelosw  31.0     0     7
## 8 Jorabado  22.9     0     8
## 9 Mitlan    12.0     0     9
## 10 Pata     11.6     1    10
## 11 Piojo    6.09    1    11
## 12 Smith    2.28    1    12
## 13 Ventana  4.05    1    13
## 14 Bahiaan  59.9    0    14
## 15 Bahiaas  63.2    0    15
## 16 Blanca   22.8    0    16
## 17 Pescador 23.5    0    17
## 18 Angeldlg  0.21    1    18
## 19 Mejia    2.55    1    19
```

And let's plot the observations

```
polis %>%
  mutate(n=1:nrow(.)) %>% # Adds a column to polis, with labeled rows (same
# as n=1:n())
  ggplot(aes(y=PA,x=RATIO))+
  geom_text(aes(label=n)) # Plots the labeled rows
```

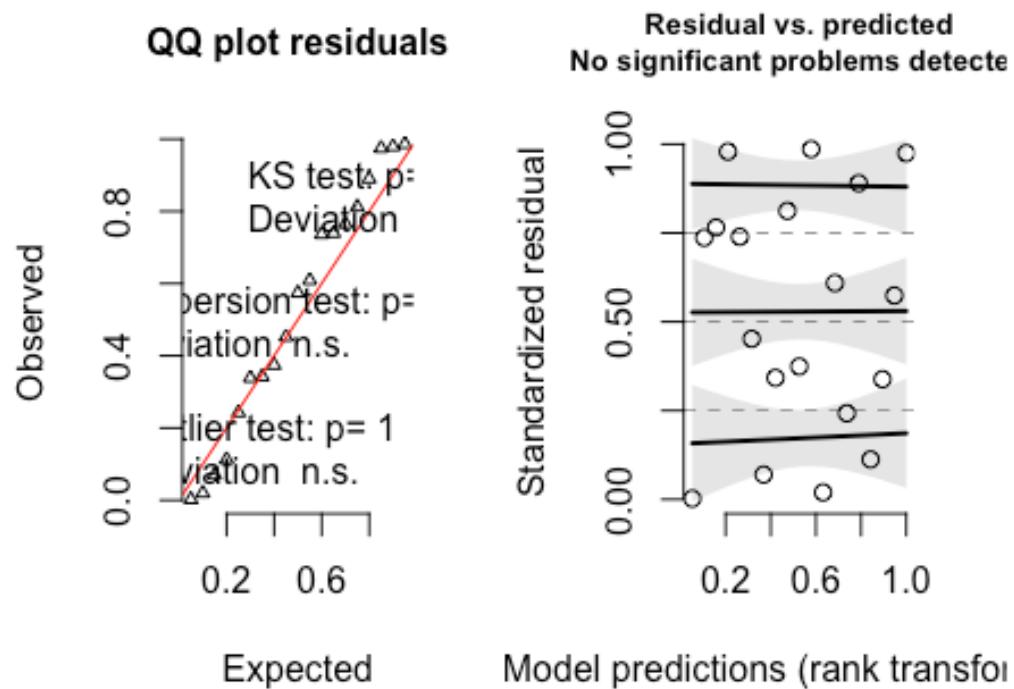


So that observation nr 3 shows an island with a very high ratio (very long). That outlier indicates that that island is quite different from the others, eg. it might indicate an unpopulated island. Because we don't know why that value is so high, we keep it in the dataset (no need to kick it out)

DHARMA residuals

```
polis.resid<-simulateResiduals(polis.glm,plot=TRUE)
```

DHARMA residual diagnostics



Our residuals plot looks quite different from the autoplot residuals plot. In this case, we don't see a pattern (so that's good). The stat.tests are also happy with the model (ns)

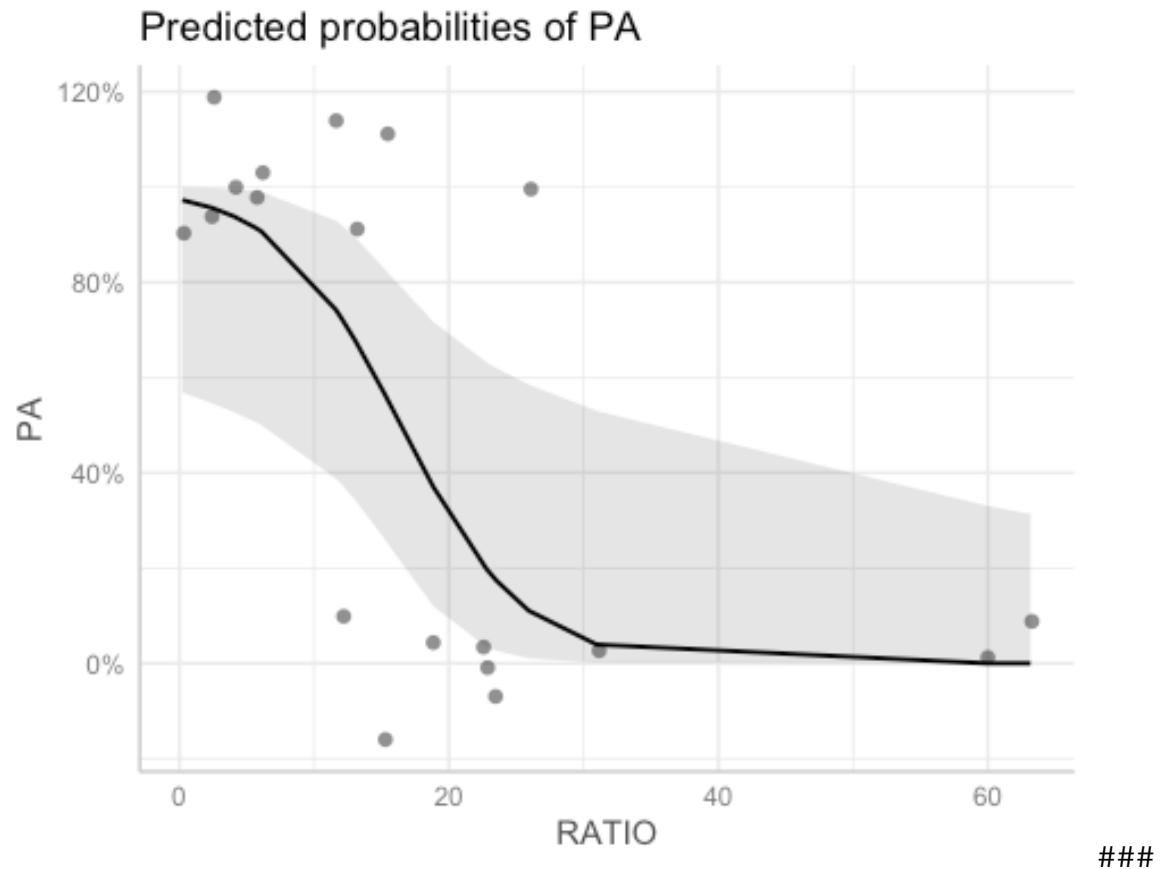
3.4. Partial plots

We could explore plotting the partial plots, like ggpredict, allEffects, etc (see previous example)

Option 1

```
polis.glm %>%
  ggpredict() %>%
  plot(add.data=TRUE)

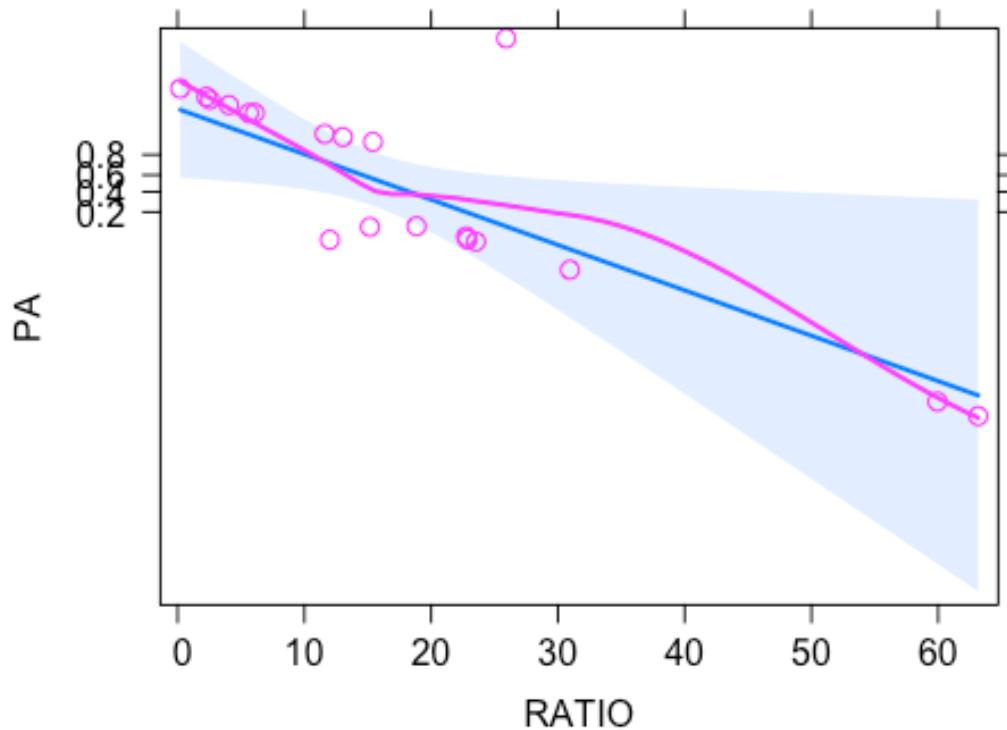
## $RATIO
```



Option 2

```
plot(allEffects(polis.glm, residuals=TRUE))
```

RATIO effect plot



These models essentially tell us that, as the ratio of the island increases, the likelihood of having lizards decreases.

3.5. Model investigation / hypothesis testing

Summarising the model

```
summary(polis.glm)

##
## Call:
## glm(formula = PA ~ RATIO, family = binomial(link = "logit"),
##      data = polis)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.6067  -0.6382   0.2368   0.4332   2.0986
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 3.6061    1.6953   2.127   0.0334 *
## RATIO      -0.2196    0.1005  -2.184   0.0289 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##  
## (Dispersion parameter for binomial family taken to be 1)  
##  
## Null deviance: 26.287 on 18 degrees of freedom  
## Residual deviance: 14.221 on 17 degrees of freedom  
## AIC: 18.221  
##  
## Number of Fisher Scoring iterations: 6
```

The output looks similar to our previous example. The values displayed are on the logit scale. A logit is the prob of sth occurring to the prob of it not occurring, or to the natural log: $\log(p/1-p)$

Our intercept is 3.6. If we back-transform that number from the logit scale to odds-ratio, we get the following: $\log(p/1-p) \rightarrow (p/1-p)$ This is the logit scale back-transformed to an odds ratio

And in R, you would write this as:

```
exp(3.61)  
## [1] 36.96605
```

This means that the odds of being present are 37x higher of not being present when the x axis is zero. When the ratio is zero, we are 37x likelier to have lizards. In other words, the smaller the island, the more lizards there will be.

We could also back transform it to a probability scale: $\log(p/1-p) \rightarrow (p/1-p) \rightarrow p$

```
plogis(3.61)  
## [1] 0.9736607
```

At a perimeter area/ratio of 0, there is a 97% of chance of the lizards being present.

The odds ratio at 0 is 37. For ev 1 unit change in perim/area ratio (when we move from 0 to 1), the odds decline by 20%, and are 80% of what they used to be. The odds are changing by that fraction for ev 1 unit change.

```
exp(-0.22)  
## [1] 0.8025188
```

So after 1 unit it goes from being 37 (times more likely to be present) to 30x more likely to be present; then 24x more likely, and so on

```
37*0.8  
## [1] 29.6  
37*0.8*0.8  
## [1] 23.68
```

= The odds are declining by 20% per unit perimeter area

```
tidy(polis.glm, conf.int=TRUE, exponentiate=TRUE)

## # A tibble: 2 × 7
##   term      estimate std.error statistic p.value conf.low conf.high
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept) 36.8      1.70      2.13  0.0334    2.73   3109.
## 2 RATIO       0.803     0.101     -2.18  0.0289    0.616   0.936
```

- Conf int: The lizards could be declining by as much as 40% per unit change, or it could be as low as 0.6 % change (???)
- We get similar info from the p value. If you took the exponential out, you'd still be checking if the zero was represented.

Calculating the surrogate of R sq

Because this model calculated from a max likelihood, you can't really rely on the R sq value that the summary(lm) might show us. We can't rely on R sq because the model was never optimising the residuals, but the likelihood.

```
1-(polis.glm$deviance/ # how far the values deviate from the expected = how
much we can't explain
polis.glm>null)

## [1] 0.4590197
```

This is the null model, without a predictor. There is only an intercept. That is why we write 1-lm In this case, around 46% of the variability is gonna be explained. We use this value as surrogate to R sq. If it was 90%, we wouldn't have to worry, but if it was 20%, our model would only explain 20% of the lizard's presence/abscence, which is not much. In that case we might have missed some important geological/biological/ecological info that might explain presence/abscence better than our perimeter/area ratio.

3.6. Predictions

Lethal dose 50

When does the probability of presence decline by 50%? When is the tipping point from “>50% more likely to have lizards” to “>50% less likely to have lizards”?

You get it by dividing the intercept by slope

```
(ld50<- - polis.glm$coeff[1]/polis.glm$coef[2]) # don't forget the (-)

## (Intercept)
##      16.4242

# We divide the intercept polis.glm$coeff[1] by the slope polis.glm$coef[2]
(first is the intercept, second is the slope)
```

```
ld50
## (Intercept)
##      16.4242
```

This tells us that at a area/perimeter ratio of 16.42, there is the same probability of presence then of absence (see plot below, at y axis = 0.5)

What if we wanted to calculate the LD50, but at other levels? Let's calculate the lethal dose where 50%, and 25% of the population will "die"

```
ld=MASS::dose.p(polis.glm,p=c(0.5,0.25))
ld

##           Dose       SE
## p = 0.50: 16.42420 3.055921
## p = 0.25: 21.42794 4.061348
```

- p=0.5 → The probability that 50% of the population is present (\neq dead), occurs at a perimeter/area ratio of 16.4 (pre-defined as "Dose" because this is usually used in medical studies)
- p=0.25 → The probability that 25% of the population is present (\neq dead), occurs at a perimeter/area ratio of 21.4 → See plot below to understand this better (!)

To display the 50LD at different "doses":

```
ld.SE=attr(ld,"SE")
ld=data.frame(LD=attr(ld,"p"),
              Dose=as.vector(ld),
              SE=ld.SE) %>%
  mutate(lower=Dose-SE*qnorm(0.975),
        upper=Dose+SE*qnorm(0.975))

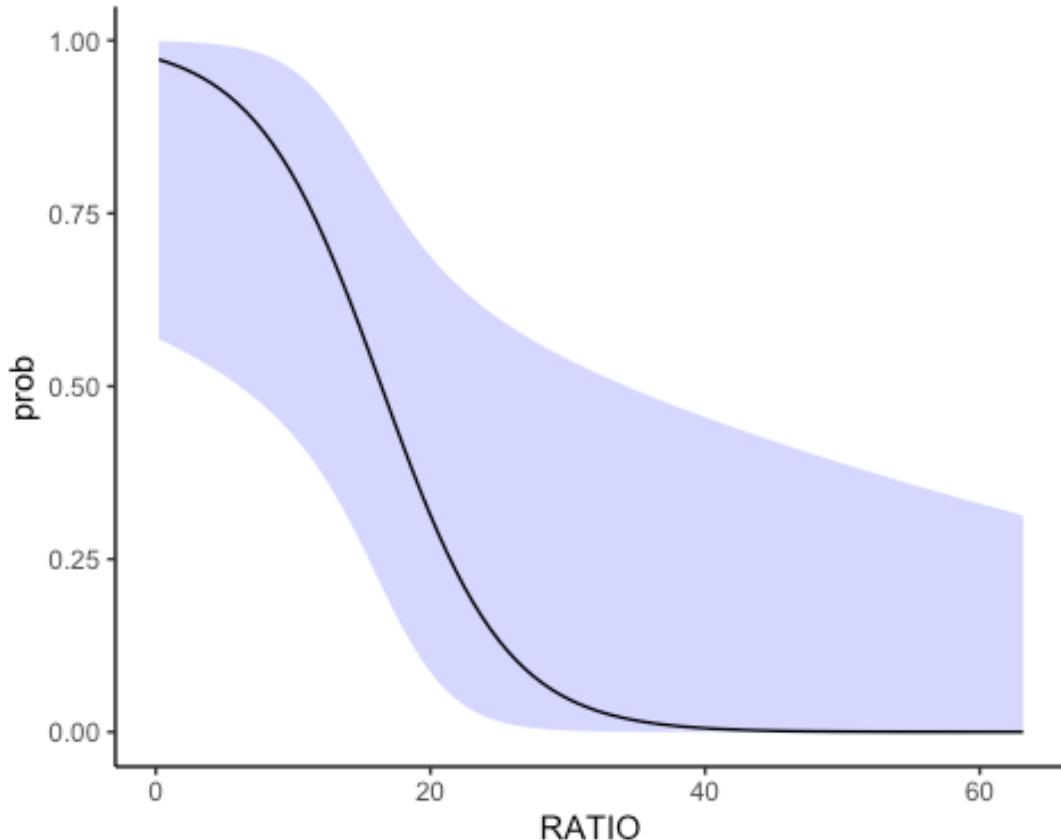
ld
##      LD      Dose       SE    lower    upper
## 1 0.50 16.42420 3.055921 10.43471 22.41370
## 2 0.25 21.42794 4.061348 13.46785 29.38804
```

3.7. Summary figures

```
# First, we create a grid with 100 values on the x axis (RATIO)
polis.grid=with(polis, list(RATIO=seq(min(RATIO),max(RATIO),len=100)))

# Second, we define a new dataset with our predicted values (emmeans)
newdata=emmeans(polis.glm, # predict values for our lm polis.glm
                ~RATIO, # for the variable RATIO
                at=polis.grid,# at the specified 100 data points that we
specified previously
                type='response') %>%
  as.data.frame # and save it as data frame
```

```
# Finally, we plot the predicted values
ggplot(newdata,aes(y=prob,x=RATIO))+ # newdata=predicted values
  geom_ribbon(aes(ymin=asymp.LCL,ymax=asymp.UCL),fill="blue",alpha=0.2)+ 
  geom_line()+
  theme_classic()
```



What are those asymp.LCL and asymp.UCL of the graph?

They are another way of plotting conf intervals (defined by the degrees of freedom)

The traditional conf int are defined by the df, which is essentially your sample size.
Problem is, conf int keep changing up to a specific df, and at some point are not really telling us anything,

```
qt(0.975,df=12)
```

```
## [1] 2.178813
```

With df=12, the upper conf int (97.5%) is 2.7

```
qt(0.975,df=19)
```

```
## [1] 2.093024
```

With df=19, the upper conf int (97.5%) is 2.1

```
qt(0.975,df=5000000)
```

```
## [1] 1.959964
```

With df=5000000, the upper conf int (97.5%) is 2.7

```
qt(0.975, df=500000000000000000000000)
```

```
## [1] 1.959964
```

At some point, the conf int (defined by the df=sample size) don't change anymore, so are not really showing a real scenario.

That is why people like to use asymptotic conf int instead of the traditional conf int. We can check them looking at the predicted data:

```
newdata %>% head
```

```

##          RATIO      prob        SE   df asympt.LCL asympt.UCL
## 1 0.2100000 0.9723466 0.04506174 Inf 0.5684028 0.9989359
## 2 0.8458586 0.9683346 0.04959335 Inf 0.5622597 0.9986284
## 3 1.4817172 0.9637623 0.05445905 Inf 0.5558908 0.9982335
## 4 2.1175758 0.9585581 0.05965685 Inf 0.5492722 0.9977274
## 5 2.7534343 0.9526432 0.06517769 Inf 0.5423765 0.9970797
## 6 3.3892929 0.9459317 0.07100404 Inf 0.5351723 0.9962525

```

If this option does not work, try adding the following code to your newdata formula ->
`lmer.df = "asymptotic"` Example: `newdata=emmeans(polis.glm, # predict values for our lm
polis.glm "wavesymbol" RATIO, # for the variable RATIO at=polis.grid,# at the specified 100
data points that we specified previously lmer.df = "asymptotic", type='response') %>%
as.data.frame # and save it as data frame`

3.8. References

4_Multiple_linear_regression

Sara Kophamel

02/12/2020

Presentation:

file:///Users/sara/Documents/PhD/HELP/STATISTICS%20and%20GRAPHS/-%20R%20and%20STATS%20COURSES/2020%20Murray%20Logan%20R%20Course/Logan_Course/docs/pres7.2.html#4

Multiple linear models have numerous predictors in a model. Try to keep it as simple as possible though. If the model seems too complicated, and you really need to use it, use regression trees.

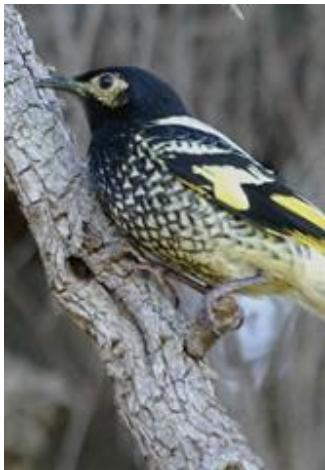
PACKAGES

```
library(car)      #for regression diagnostics
library(broom)    #for tidy output
library(ggfortify) #for model diagnostics
library(sjPlot)   #for outputs
library(knitr)    #for kable
library(effects)  #for partial effects plots
library(MuMIn)    #for interfacing with STAN
library(emmeans)  #for estimating marginal means
library(ggeffects) #for partial effects plots
library(MASS)     #for glm.nb
library(MuMIn)    #for AICc
library(DHARMa)   #for residual diagnostics plots
library(modelr)   #for auxillary modelling functions
library(performance) #for residuals diagnostics
library(see)       #for plotting residuals
library(patchwork) #for grids of plots
library(tidyverse) #for data wrangling
```

Example: glm_example4.Rmd Avg count of birds per grazing patch area, with different We are having averages, which tend to follow a Gaussian distribution (#Poisson)

Scenario

@Loyn-1987-1987 modeled the abundance of forest birds with six predictor variables (patch area, distance to nearest patch, distance to nearest larger patch, grazing intensity, altitude and years since the patch had been isolated).



Regent honeyeater

Format of loyn.csv data file

ABUND	DIST	LDIST	AREA	GRAZE	ALT	YR.ISOL
..
ABUND	Abundance of forest birds in patch-response variable					
DIST	Distance to nearest patch - predictor variable					
LDIST	Distance to nearest larger patch - predictor variable					
AREA	Size of the patch - predictor variable					
GRAZE	Grazing intensity (1 to 5, representing light to heavy) - predictor variable					
ALT	Altitude - predictor variable					
YR.ISOL	Number of years since the patch was isolated - predictor variable					

The aim of the analysis is to investigate the effects of a range of predictors on the abundance of forest birds.

Read in the data

```
loyn = read_csv('data/loyn.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   ABUND = col_double(),
##   AREA = col_double(),
##   YR.ISOL = col_double(),
##   DIST = col_double(),
```

```

##   LDIST = col_double(),
##   GRAZE = col_double(),
##   ALT = col_double()
## )

glimpse(loyn)

## Rows: 56
## Columns: 7
## $ ABUND    <dbl> 5.3, 2.0, 1.5, 17.1, 13.8, 14.1, 3.8, 2.2, 3.3, 3.0, 27.6,
1...
## $ AREA     <dbl> 0.1, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0,
2.0, 2...
## $ YR.ISOL <dbl> 1968, 1920, 1900, 1966, 1918, 1965, 1955, 1920, 1965,
1900, 1...
## $ DIST     <dbl> 39, 234, 104, 66, 246, 234, 467, 284, 156, 311, 66, 93,
39, 4...
## $ LDIST    <dbl> 39, 234, 311, 66, 246, 285, 467, 1829, 156, 571, 332, 93,
39, ...
## $ GRAZE   <dbl> 2, 5, 5, 3, 5, 3, 5, 4, 5, 3, 5, 2, 1, 5, 5, 3, 3, 3,
2, 2...
## $ ALT      <dbl> 160, 60, 140, 160, 140, 130, 90, 60, 130, 130, 210, 160,
210, ...

```

1- Exploratory data analysis

Model formula:

$$\$y_i \sim \mathcal{N}(\mu_i, \sigma^2) \log(\mu_i) = \boldsymbol{\beta} \bf{X}_i$$

where β is a vector of effects parameters and \bf{X} is a model matrix representing the additive effects of the scaled versions of distance (ln), distance to the nearest large patch (ln), patch area (ln), grazing intensity, year of isolation and altitude on the abundance of forest birds. The bold X is a set of betas, one for each vector, and a matrix of x's.

Assumptions

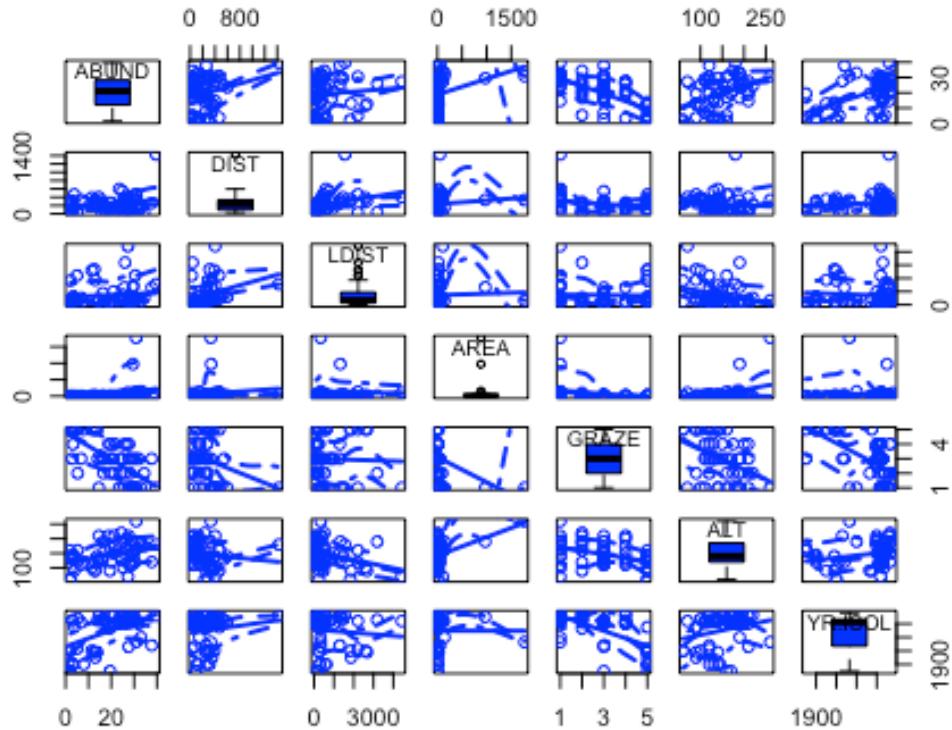
- Independence
- Normally distributed
- Homogeneity of variance
- Predictors not correlated with one another: If you put two variables in the model to be correlated, it is not going to distinguish the combined impact of the two. The model will define which effect is stronger, so they will compete against each other.

Don't use linear models to predict data (!)

Assumption testing

To test the assumptions: Create a scatterplot matrix, which plots the variable predictors against each other. By doing so, it will also indicate if there is correlation.

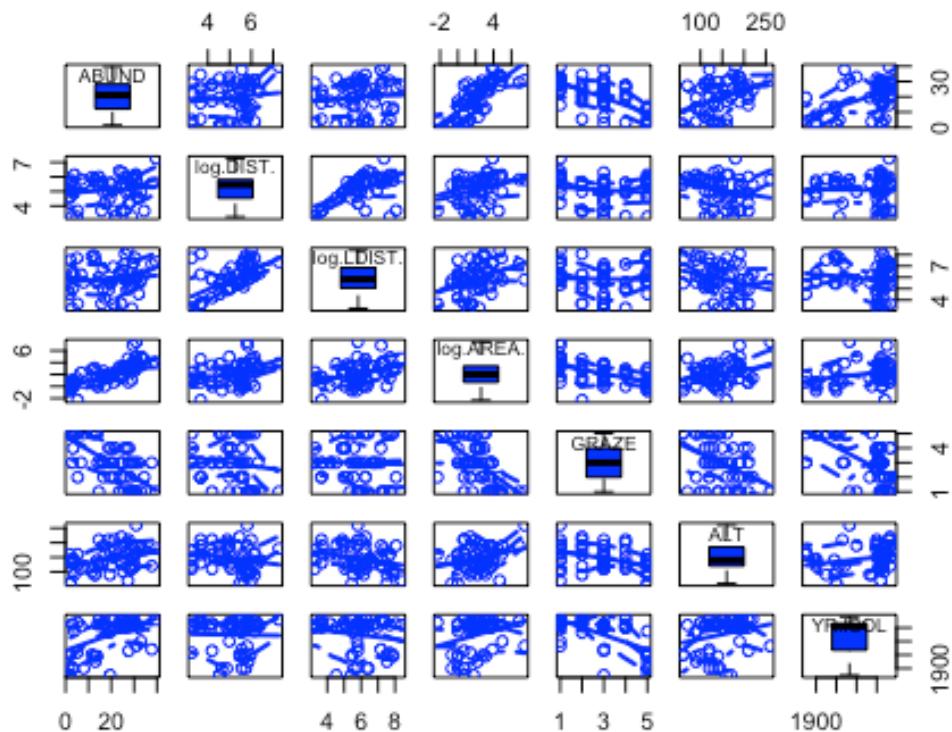
```
scatterplotMatrix(~ABUND+DIST+LDIST+AREA+GRAZE+ALT+YR.ISOL, data=loyn,
diagonal=list(method="boxplot"))
```



Here you can see all variables plotted pairwise against each other, all treated equally. By default, R would create a density plot, but because we got fairly small amounts of data, we define R to plot a boxplot. -> Abundance is going to be on the y axis of the first row. The second left graph from the first row for example has abundance on the y, and distance on the x axis.

The first thing you should address is the Normality of your predictors and your response. Once we addressed the normality issues, we can re-plot the graph. In this case, the following variables do not seem to be normally distributed: DIST, LDST, AREA. To fix this problem, we could: - normalise them with log-transformation (when skewed to the right / larger line on TOP of the boxplot) - YR.ISOL also seems to be slightly skewed to the left / larger line on BOTTOM of the boxplot. In this case, we'd have to do a sq root. However, this is very complex especially if we are using other transformations too. Murray reckons it does not look TOO paranormal, and would leave this variable as it is. -> We decide to transform the skewed variables to natural log (#log to the base 10):

```
scatterplotMatrix(~ABUND+log(DIST)+log(LDIST)+log(AREA)+GRAZE+ALT+YR.ISOL,
data=loyn, diagonal=list(method="boxplot"))
```



Now, all our predictors seem to have been normalised.

Homogeneity of variance: Looks good. There does not seem to be a relationship btw mean and variance.

The top row shows the linearity of ABUND. There seems to be a relationship btw ABUND and log.AREA, ABUND and GRAZE, ABUND and ALT, and maybe also ABUND and YR.ISOL.

Let's have a closer look at the GRAZE variable. R assumes that the diff btw grazing intensity 1 and 2 is similar to grazing 2 and 3... We will convert GRAZE into a categorical (factor) variable.

```
loyn=loyn %>% mutate(factor(GRAZE))

# Alternative transformation -->
loyn=loyn %>% mutate(fGRAZE=ordered(factor(GRAZE))) # We turn GRAZE into an
# ordered factor (if GRAZE had been coded g1,g2,g3... instead of 1,2,3, R would
# have recognised it as factor from the beginning). I tried using this one, but
# this is a polynomial contrast model (whatever it is), and the diagnostic
# plots would not match Murray's. So better ignore the ordered command for now.
```

2- Fit the model

Transforming data

If we fit a model where we perform the transformation online, then the model will know that we performed a transf, and can perform back-transf later on. IF we supply the model with ALREADY transformed data, it will not be able to back-transform it later on (!)

In the case of grazing now, however, we perform it outside.

Centering the data

Before we fit the model, we will have to centre our data.

Centering will do two things: - The intercept will become the mean bird abundance across the whole data (instead of when area is 0, grazing is 0, etc) - The max likelihood can find a peak much better when we centre the data

```
scale(1:3,scale=FALSE) # It is important that we write scale=FALSE, so that  
# the data is centred around 0
```

```
##      [,1]  
## [1,]    -1  
## [2,]     0  
## [3,]     1  
## attr(,"scaled:center")  
## [1] 2
```

After scaling, the numbers are turned into neg, and are set around 0

FYI: If we hadn't set scale=TRUE, we would NOT centre around zero:

```
scale(1:3,scale=TRUE) # This would NOT centre around zero, but scale the data  
# instead
```

```
##      [,1]  
## [1,]    -1  
## [2,]     0  
## [3,]     1  
## attr(,"scaled:center")  
## [1] 2  
## attr(,"scaled:scale")  
## [1] 1  
  
#attr(,"scaled:center")  
#attr(,"scaled:scale")
```

Model creation

Now let's create the model, log-transforming AND centering the data

```
loyn.glm<-
  glm(ABUND~scale(log(DIST), scale=FALSE)+scale(log(DIST), scale=FALSE)+
    scale(log(AREA), scale=FALSE)+
    fGRAZE+scale(ALT, scale=FALSE)+scale(YR.ISOL, scale=FALSE),
    data=loyn, family=gaussian())
```

We should now validate the model, check that it is ok... And if it is not, we might need another distribution (log, gamma, etc).

Option 2: Gaussian distribution with a link=log, indicating that it is a log-normal distribution. The link log is logging the EXPECTED abundance (not the abundance itself). It logs the expectations, which is why it is better than logging abundance per se. In other words, the expected values won't have any zeros.

```
loyn.glm1<-
  glm(ABUND~scale(log(DIST), scale=FALSE)+scale(log(DIST), scale=FALSE)+
    scale(log(AREA), scale=FALSE)+
    fGRAZE+scale(ALT, scale=FALSE)+scale(YR.ISOL, scale=FALSE),
    data=loyn, family=gaussian(link="log"))
```

Option 3: Gamma distribution, also with a log-link

```
loyn.glm2<-
  glm(ABUND~scale(log(DIST), scale=FALSE)+scale(log(DIST), scale=FALSE)+
    scale(log(AREA), scale=FALSE)+
    fGRAZE+scale(ALT, scale=FALSE)+scale(YR.ISOL, scale=FALSE),
    data=loyn, family=Gamma(link="log"))
```

Theory suggests that a Gaussian distribution will be fine, but we should always have different models at hand, just in case.

3- MCMC sampling diagnostics

Checking for correlated predictors vif()

We have to make sure there are no issues of correlated predictors in the model. We could have a look at the diagnostic plots to see if we find predictors competing (eg. log.DIST and log.LDIST seem to be correlated, competing against each other). The metrics to suggest a relationship are: - R square -> Murray does not recommend using it. Theory is that, if you had an R sq > 0.6 (or >0.8, depending on the strength you want to accept), it would indicate a correlation. - Variance inflation -> You calculate it with vif. If there is a vif >3, it would indicate a correlation.

```
vif(loyn.glm)
##                                     GVIF Df GVIF^(1/(2*Df))
## scale(log(DIST), scale = FALSE) 1.518627  1      1.232326
## scale(log(AREA), scale = FALSE) 1.760947  1      1.327007
## fGRAZE                           7.141590  4      1.278570
```

```
## scale(ALT, scale = FALSE)      1.530143  1      1.236990
## scale(YR.ISOL, scale = FALSE) 3.252124  1      1.803365
```

If we look at the first column GVIF, there are two values >3, fGRAZE and scale(YR.ISOL, scale = FALSE). If we got cat predictors in a model, we are NOT dealing with just one predictor. A cat pred is essentially the amount of levels -1 of that predictor. If the pred has the levels high-low-med, there is actually two predictors (the third object would be the intercept). However, it is the RIGHT hand column that you have to pay attention to, as it accounts for the DF (!). And in this case, all variables are <3, so we would not assume correlation btw any of the variables. If it does not manifest as a problem in the model, even though it might in the plot, we assume the model is right.

But what if we HAD predictors competing in the model? - We would remove one of them, if possible. - If we can't remove the correlated predictor(s), we might be able to combine them with a principal components/coordinates analysis. Getting a proxy for the distance btw them might be sufficient. In other words, we would reduce the amount of correlations, merging them. - If we had interactions, centering is NOT optional. You MUST center if you have two continuous variables in a multiple regression model. They WILL interact. Otherwise, you might end up with problems -> Ex: We got two continuous variables temp and time, and their interaction. temp and time WILL be correlated to the interaction, unless you centre. And then they won't be correlated. If we are working with a continuous var and a factor/cat var, don't bother. If we are working with two cat var, don't bother either.

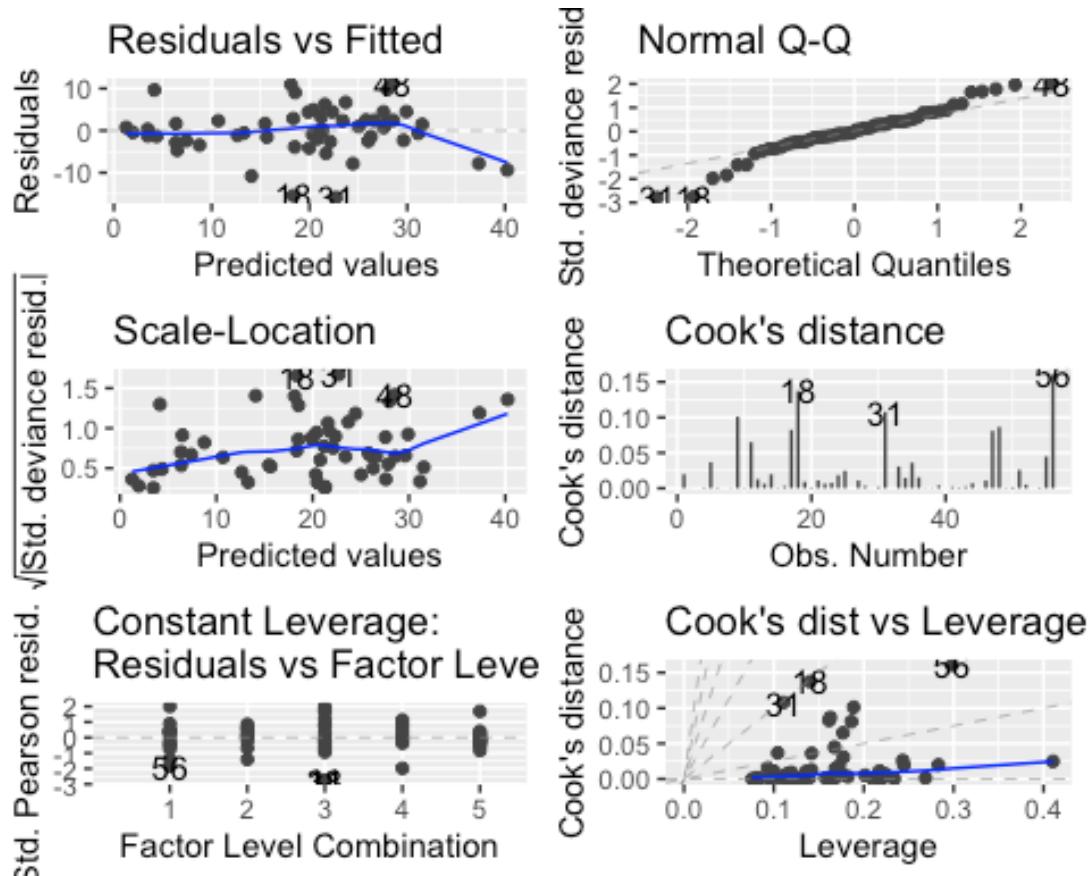
4- Model validation

Diagnostic plots

autoplot

```
autoplot(loyn.glm, which=1:6)

## Warning: `arrange_()` is deprecated as of dplyr 0.7.0.
## Please use `arrange()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```



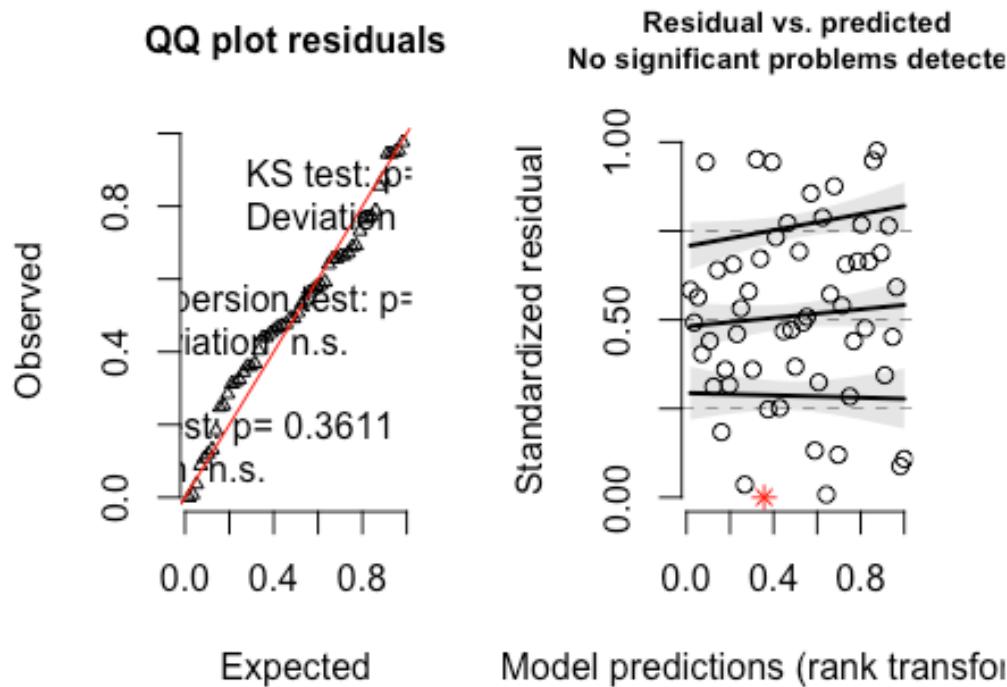
Residuals vs Fitted: Looks acceptable - Nornal QQ plot: There is less mass on the left handside of the tail, which prob means it's a bit truncated on the L handside. And it is a bit fatter than a regular normal distrib (=there is more mass on the R handside) - Cook's distance: Looks ok. If a data point was >0.8 , it is likely an outlier. In general, our diagnostic plots look ok.

The authors of this research analysed it as Gaussian (which is ok), but they also sq root transformed Abundance!! Back transf from sq roots are really NOT appropr, particularly if you got values that span <1 and >1 . The transf value, if it was -0, it suddenly becomes positive. So things btw 0-1 get smaller if you back transform. Murray would NEVER recommend root sq the data, if that can be avoided.

DHARMA residual diagnostics

```
loyn.resid<-simulateResiduals(loyn.glm, plot=TRUE)
```

DHARMA residual diagnostics



The DHARMA residuals suggests that there is no real issue. A potential outlier has been identified (red star), even though the outlier test is negative. The DHARMA residuals are not THAT good for small datasets (<30 observations per predictor and per level of grazing). This here is effectively a small dataset, so we should also check the following:

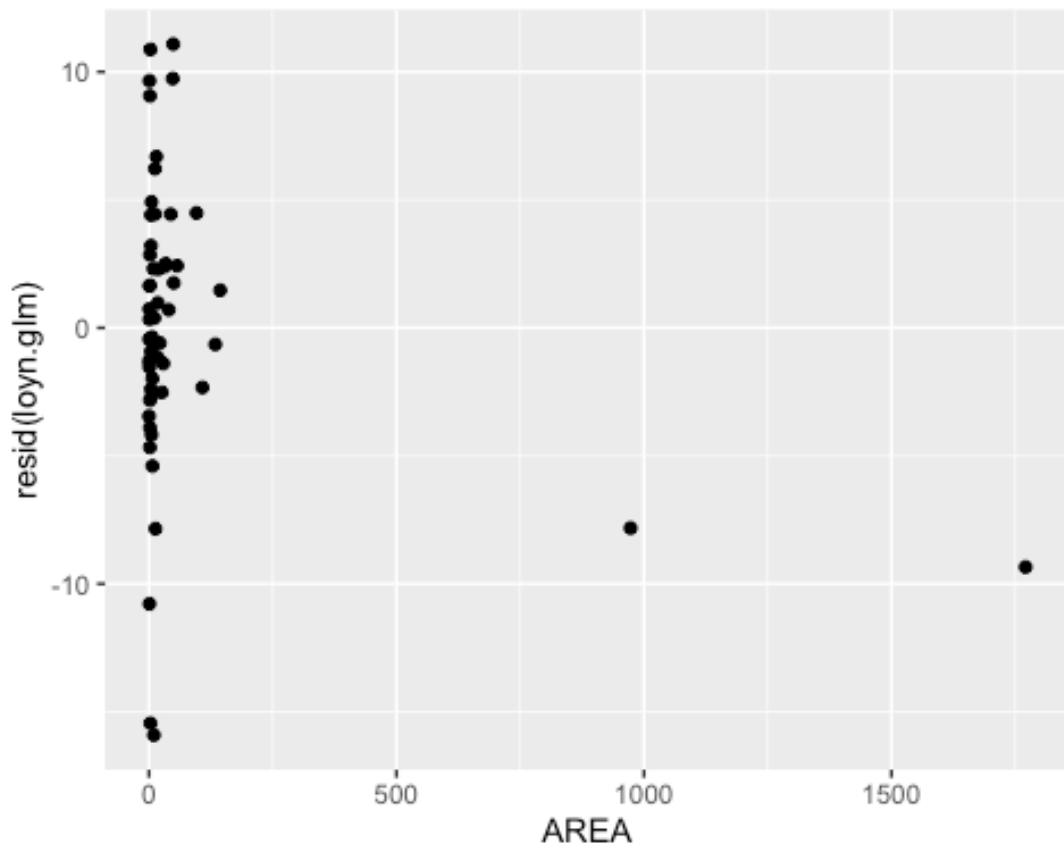
Residuals plotted against each of the predictors

When running anything with multiple pred, we should both look at the residual plot, but also at the residuals plotted against each of the predictors!

y axis = Residuals from our model, x axis = log Area. I could do this for EACH of my predictors to make sure there are no patterns. Some sort of wedge would be bad, or if one group would be -0, and the other +0, or if one was spread to one side, and the other to the other side.

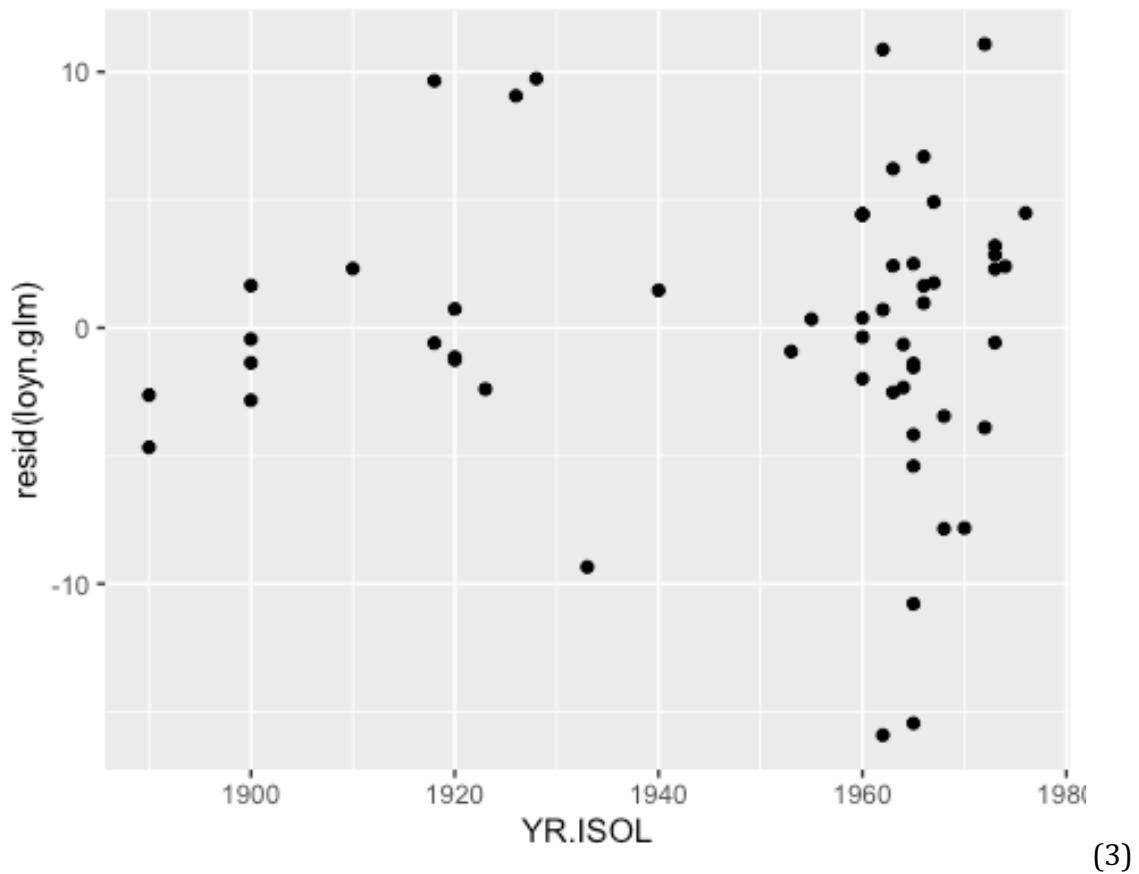
(1) Residuals against area:

```
ggplot(data=loyn)+geom_point(aes(y=resid(loyn.glm),x=AREA))
```



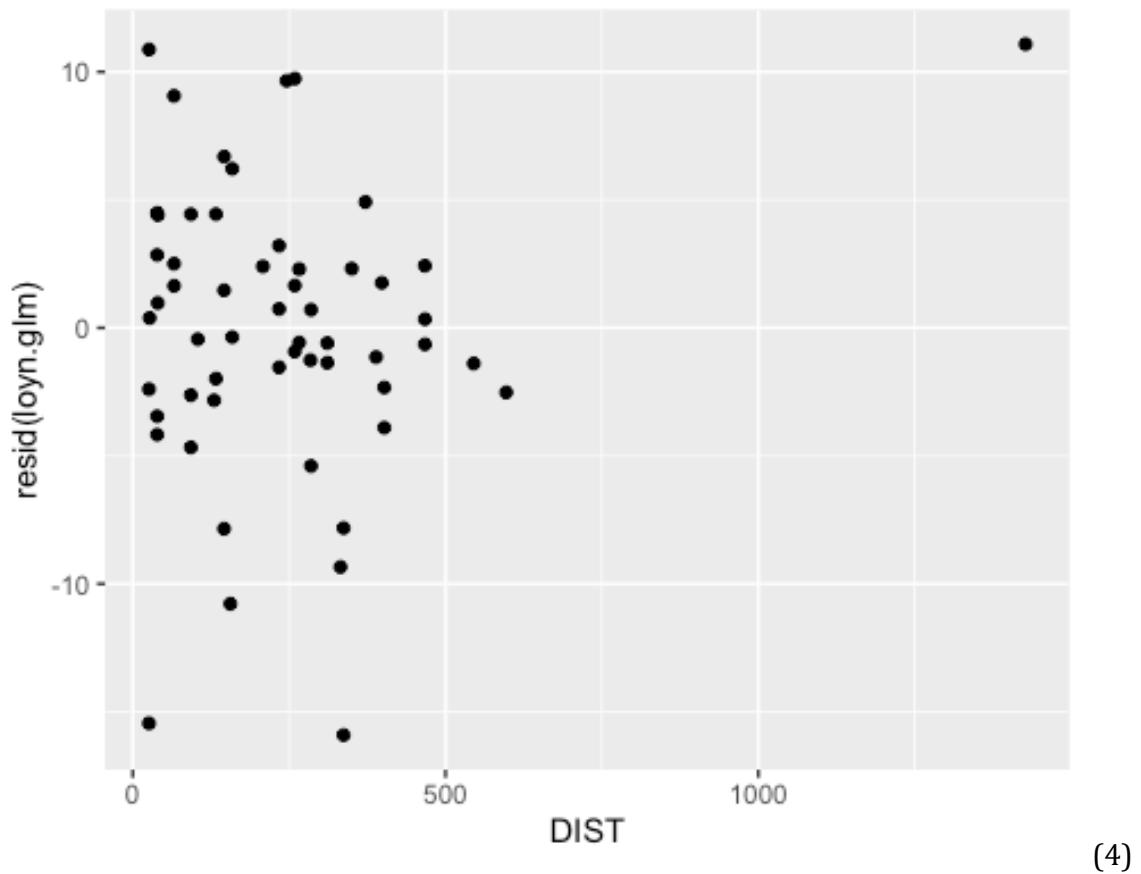
(2) Residuals against years in isolation:

```
ggplot(data=loyn)+geom_point(aes(y=resid(loyn.glm),x=YR.ISOL))
```



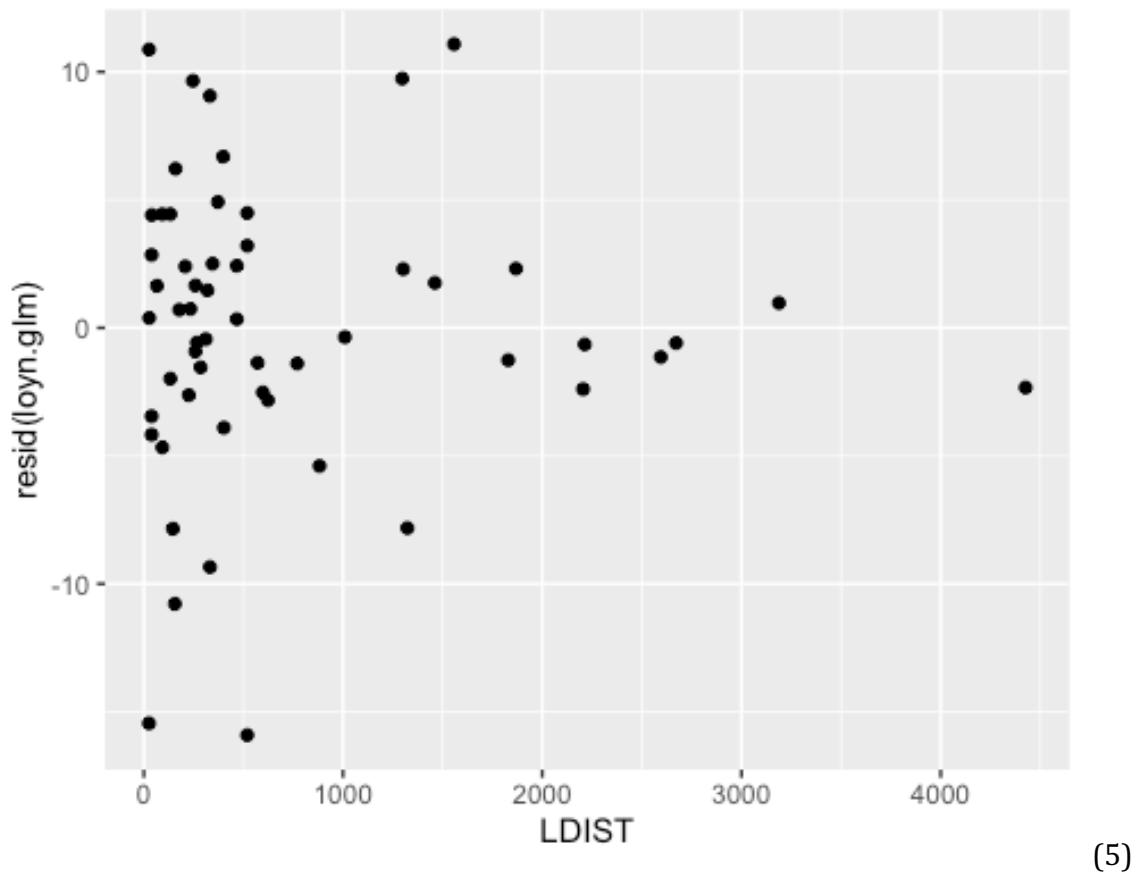
Residuals against dist:

```
ggplot(data=loyn)+geom_point(aes(y=resid(loyn.glm),x=DIST))
```



Residuals against log dist:

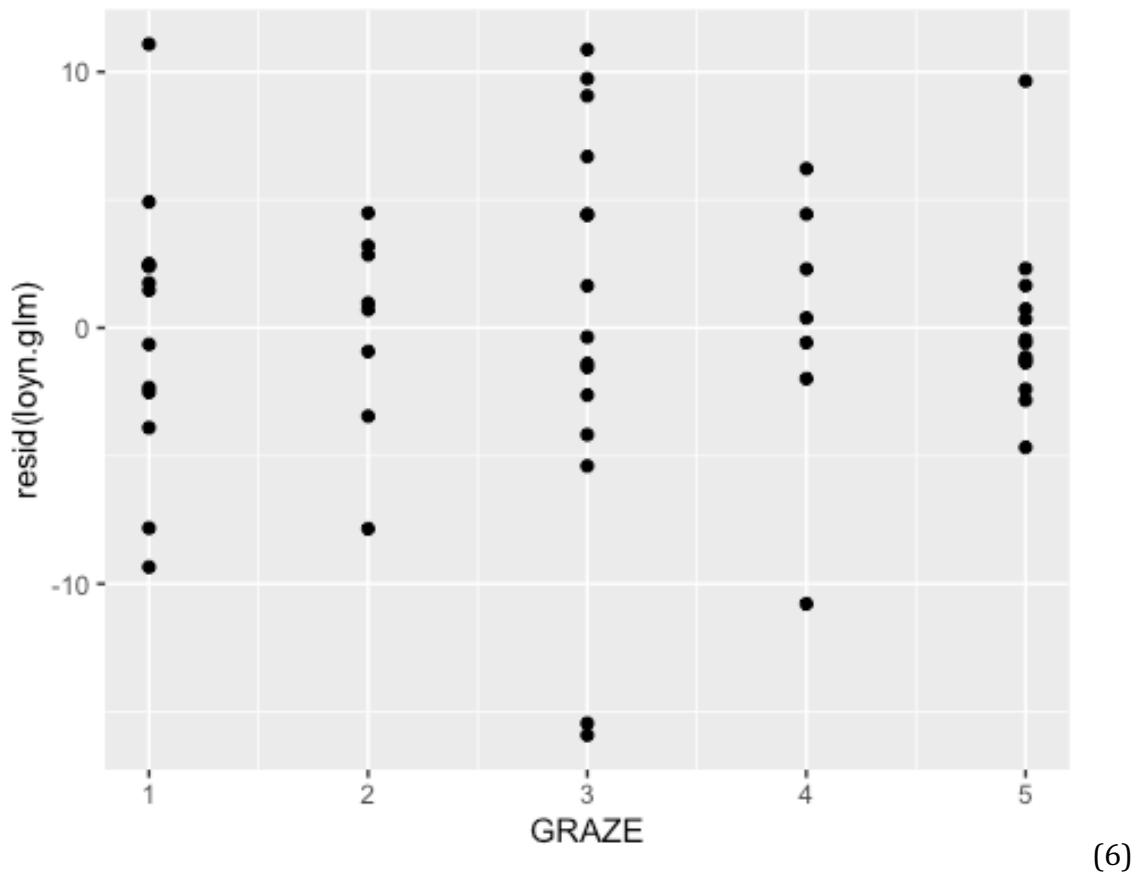
```
ggplot(data=loyn)+geom_point(aes(y=resid(loyn.glm),x=LDIST))
```



(5)

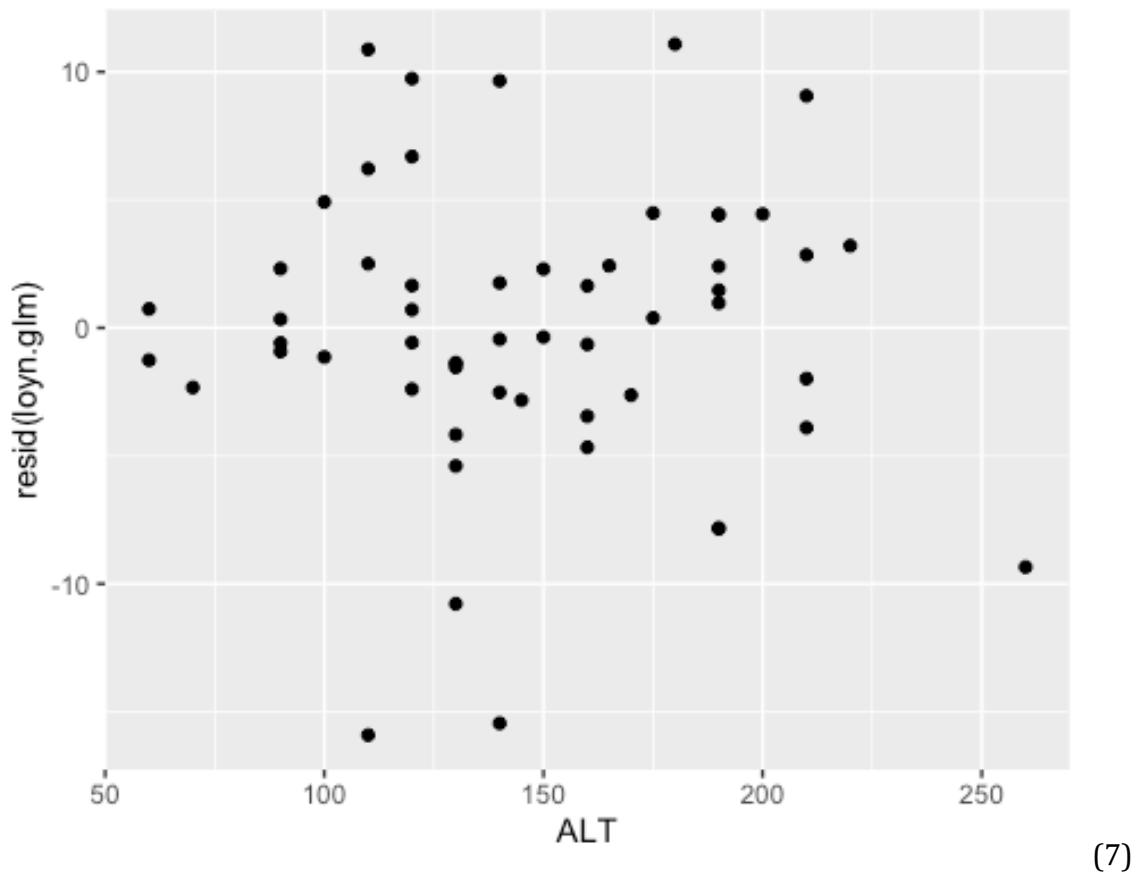
Residuals against grazing patch category:

```
ggplot(data=loyn)+geom_point(aes(y=resid(loyn.glm),x=GRAZE))
```



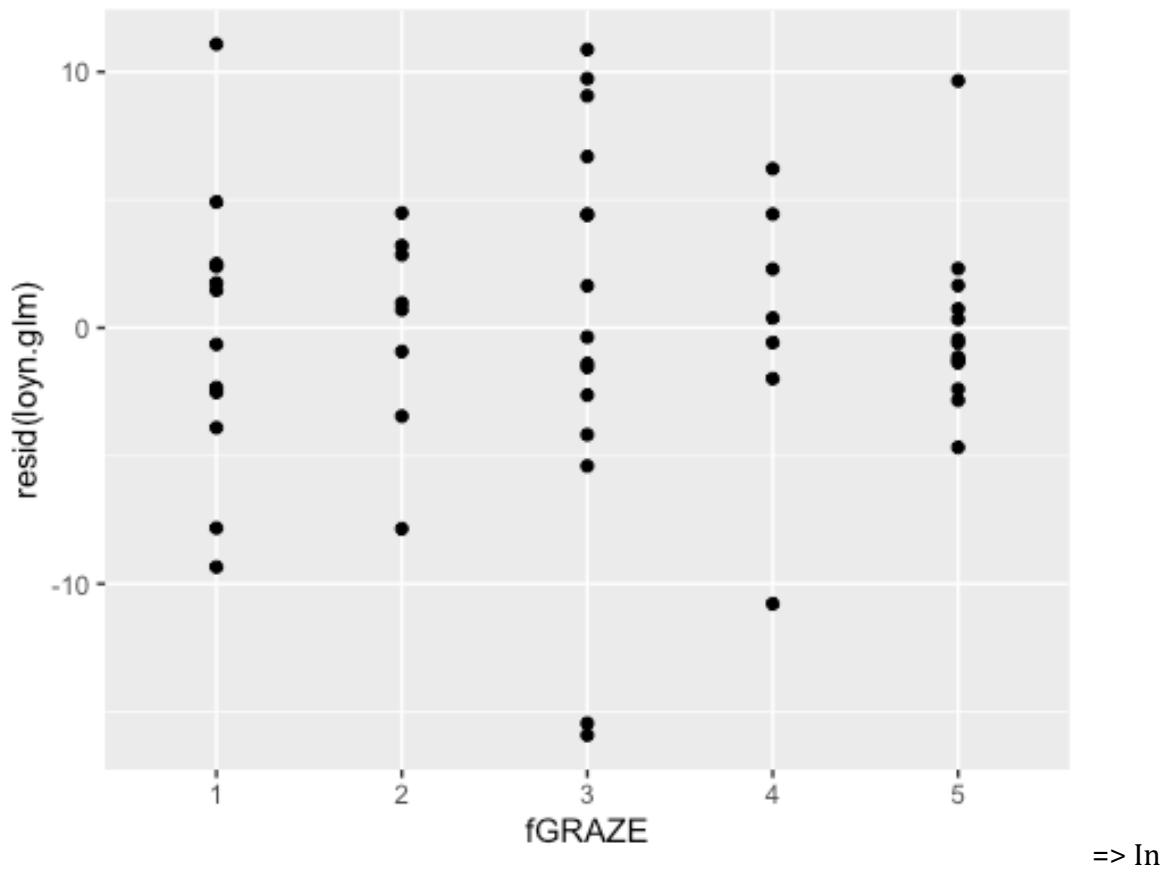
Residuals against altitude:

```
ggplot(data=loyn)+geom_point(aes(y=resid(loyn.glm),x=ALT))
```



Residuals against logged grazing patch category:

```
ggplot(data=loyn)+geom_point(aes(y=resid(loyn.glm),x=fGRAZE))
```



summary, all residuals look ok.

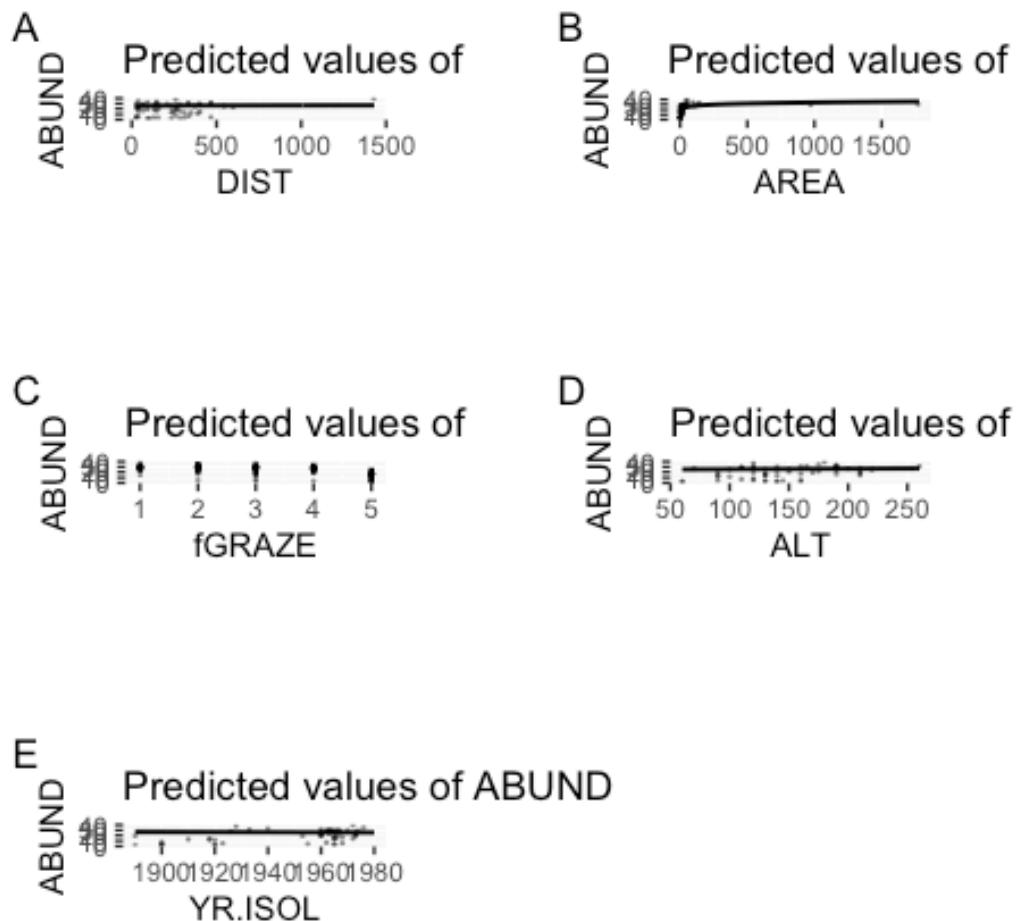
5- Partial effects plots

Quick summary graphs

To show the partial effects for each of the predictors. If the plot is too huge to be displayed, write in the curly brackets {r fig.width=5,fig.height=5} (fiddle with the numbers, in this case 5 was fine); OR run the command in the console so that the plot shows up in the right tab window.

Option 1 - plot_model

```
plot_model(lyn.glm,type='eff',show.data=TRUE,dot.size=0.5) %>% plot_grid  
## Warning in plot_grid(.): Not enough tags labels in list. Using letters instead.
```



Option 2 - allEffects

```
plot(allEffects(loyn.glm,residuals=TRUE),type="response")
### Warning in Analyze.model(focal.predictors, mod, xlevels, default.levels, :
the
## predictors scale(log(DIST), scale = FALSE), scale(log(AREA), scale =
FALSE),
## scale(ALT, scale = FALSE), scale(YR.ISOL, scale = FALSE) are one-column
matrices
## that were converted to vectors

## Warning in Analyze.model(focal.predictors, mod, xlevels, default.levels, :
the
## predictors scale(log(DIST), scale = FALSE), scale(log(AREA), scale =
FALSE),
## scale(ALT, scale = FALSE), scale(YR.ISOL, scale = FALSE) are one-column
```

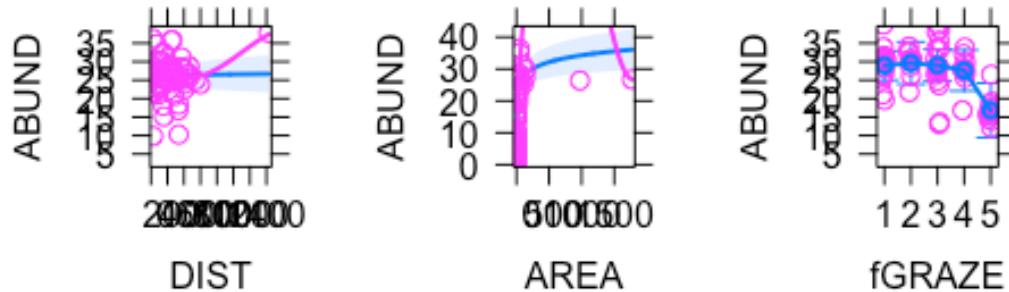
```
matrices
## that were converted to vectors

## Warning in Analyze.model(focal.predictors, mod, xlevels, default.levels, :
the
## predictors scale(log(DIST), scale = FALSE), scale(log(AREA), scale =
FALSE),
## scale(ALT, scale = FALSE), scale(YR.ISOL, scale = FALSE) are one-column
matrices
## that were converted to vectors

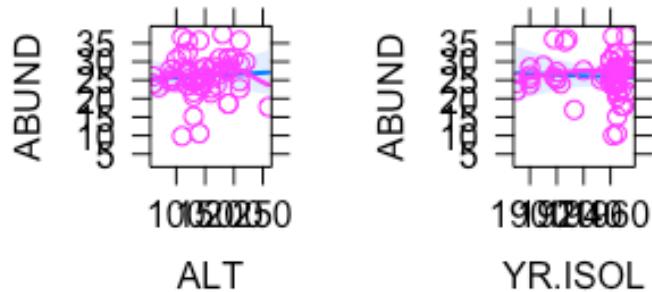
## Warning in Analyze.model(focal.predictors, mod, xlevels, default.levels, :
the
## predictors scale(log(DIST), scale = FALSE), scale(log(AREA), scale =
FALSE),
## scale(ALT, scale = FALSE), scale(YR.ISOL, scale = FALSE) are one-column
matrices
## that were converted to vectors

## Warning in Analyze.model(focal.predictors, mod, xlevels, default.levels, :
the
## predictors scale(log(DIST), scale = FALSE), scale(log(AREA), scale =
FALSE),
## scale(ALT, scale = FALSE), scale(YR.ISOL, scale = FALSE) are one-column
matrices
## that were converted to vectors
```

DIST effect plot AREA effect plot GRAZE effect plot



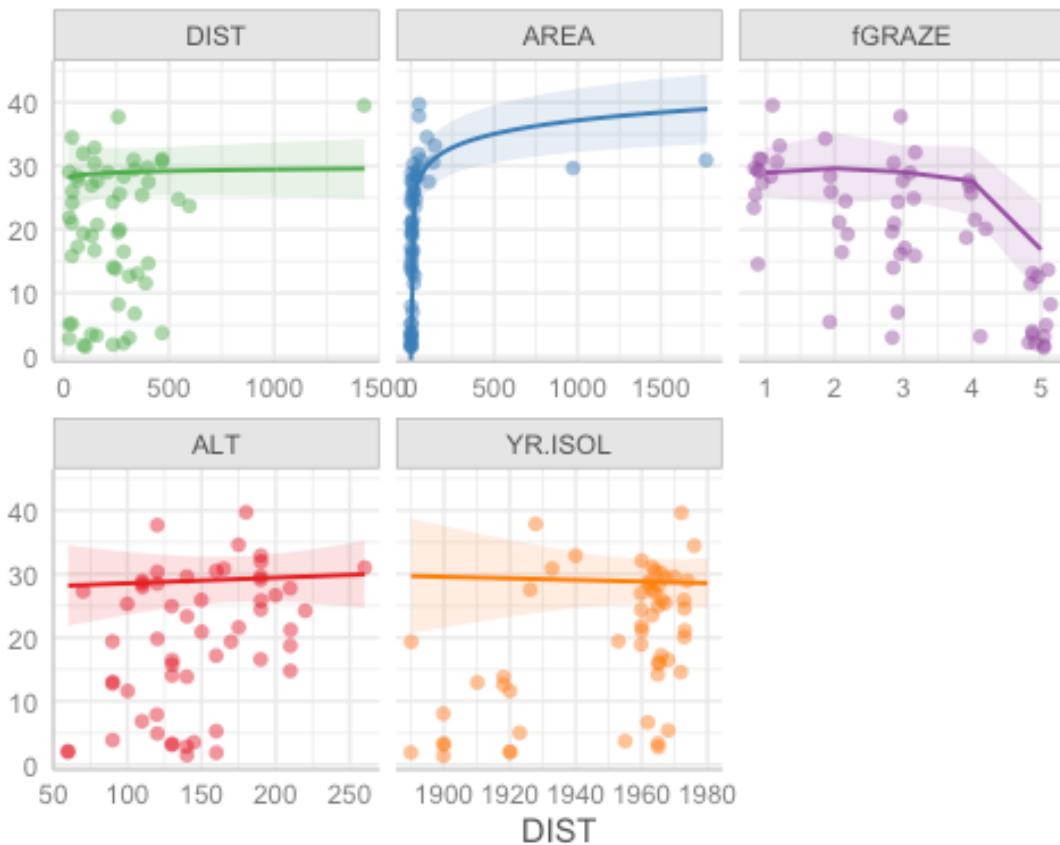
ALT effect plot YR.ISOL effect plot



###

Option 3 - ggpredict

```
ggpredict(loyn.glm) %>% plot(add.data=TRUE, facet=TRUE)
```

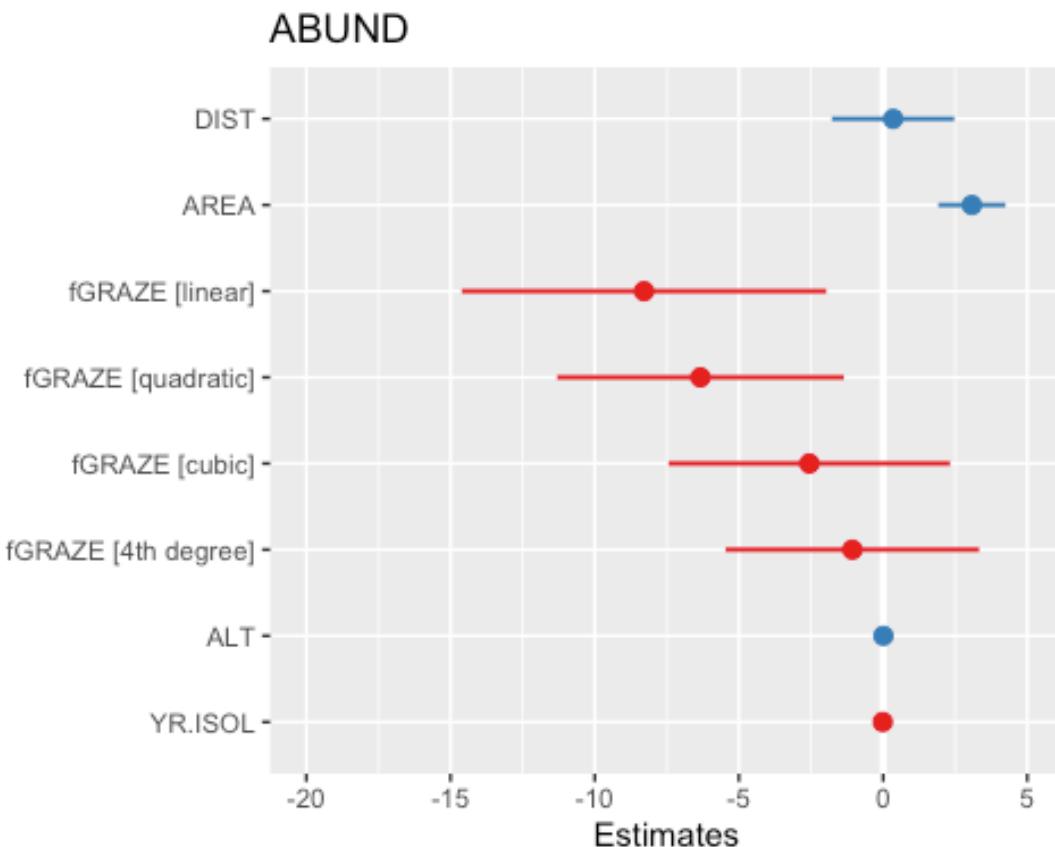


Distance has no effect on the amount of birds - The higher the area, the more birds - If grazing level is really high, the birds decline - Altitude does not impact that much the amount of birds - Years of isolation does not seem to impact the amount of birds either BUT the dots don't seem to follow the trends very well. Why? Because this diagnostic plot has plotted the predicted values (lines and error bars), and has plotted the RAW data points on top.

What you would do is to run the model like this, and then run a particular contrast.

Option 4 - caterpillar plot

```
plot_model(lyn.glm, type='est')
```



What we should see here is: - Along the y axis are each of the model terms (distance, log dist, area, and the four graze terms/categories, altitude and yrs of isol) - Red to the left = Negative impact on the amount of birds \rightarrow Grazing areas 4 and 5 - We got a ref line of zero = No impact on the amount of birds \rightarrow Altitude and years of isolation - Blue ones are to the right = Positive impact on the amount of birds \rightarrow Distance and area, and grazing areas 2 and 3

Model investigation

```
summary(loyn.glm)

##
## Call:
## glm(formula = ABUND ~ scale(log(DIST), scale = FALSE) + scale(log(DIST),
##     scale = FALSE) + scale(log(AREA), scale = FALSE) + fGRAZE +
##     scale(ALT, scale = FALSE) + scale(YR.ISOL, scale = FALSE),
##     family = gaussian(), data = loyn)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -15.9124   -2.4264   -0.0115    2.5923   11.0863
##
## Coefficients:
```

```

##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                19.83573  0.86483 22.936 < 2e-16 ***
## scale(log(DIST), scale = FALSE) 0.34569  1.05488  0.328  0.7446
## scale(log(AREA), scale = FALSE) 3.07649  0.57859  5.317 2.86e-06 ***
## fGRAZE.L                  -8.29109  3.14131 -2.639  0.0112 *
## fGRAZE.Q                  -6.33684  2.47070 -2.565  0.0136 *
## fGRAZE.C                  -2.56071  2.42507 -1.056  0.2964
## fGRAZE^4                  -1.06793  2.18535 -0.489  0.6273
## scale(ALT, scale = FALSE)    0.00887  0.02318  0.383  0.7037
## scale(YR.ISOL, scale = FALSE) -0.01252  0.05750 -0.218  0.8286
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 36.58765)
##
## Null deviance: 6337.9  on 55  degrees of freedom
## Residual deviance: 1719.6  on 47  degrees of freedom
## AIC: 370.69
##
## Number of Fisher Scoring iterations: 2

```

Let's interpret the bits and pieces here:

- Intercept: It is the value of y when all of the axes = 0. Because we CENTERED each of the predictors, it is the response when all the predictors are at their average. Across our study, if the grazing level = 1 (the first level, grazing 1, becomes the reference level), we have 22.4 birds. That intercept has a meaning, and we can use it for our ecological interpretations.
- The number of birds, on avg, on grazing level 2, has increased by 0.7 in comparison to grazing level 1 (but that is not significant)
- Grazing level 4 has -1.33 birds in comparison to grazing level 1, but it is not sign
- Grazing level 5 has -12 birds in comparison to grazing level 1, and it is a sign diff (!)
- For ev 1 unit change in log distance, the birds increase in grazing level 1 by 0.34 of a bird. Ev unit, you get 0.34 more birds in grazing 1. That is ns. (see: scale(log(DIST), scale = FALSE) -> ESTIMATE)
- We don't know what the rate of grazing is for grazing level 2, 3, 4... because we assumed that it was an ADDITIVE model, without interactions. We assume that the increase is teh SAME for each level
- For ev 1 unit increase in log area, we get 3 more birds (see: scale(log(AREA), scale = FALSE) 3.07649 -> ESTIMATE)
- No evidence of Altitude or Year effect (see: scale(ALT, scale = FALSE) ; and scale(YR.ISOL, scale = FALSE))

What if we had put scale predictors in our model?

That would put ALL of these coefficients on the same scale, and therefore we could say which ones would be more important and had a stronger effect. However, these are all on diff scales. You woudl take a predictor and multipliy it by the ration of its variance to the total variance of the model. We don't want to do this by hand, so we will use the "standard coefficient" function. It will allow us to compare each predictor to get a relative effect, and to define which has what magnitude. It is ok to do that at this point in our Model investigation.

```
std.coef(loyn.glm,partial.sd=TRUE)
```

	Estimate*	Std. Error*	df
## (Intercept)	0.00000	0.00000	47
## scale(log(DIST), scale = FALSE)	0.28611	0.87307	47
## scale(log(AREA), scale = FALSE)	4.64232	0.87307	47
## fGRAZE.L	-2.30435	0.87307	47
## fGRAZE.Q	-2.23923	0.87307	47
## fGRAZE.C	-0.92190	0.87307	47
## fGRAZE^4	-0.42665	0.87307	47
## scale(ALT, scale = FALSE)	0.33410	0.87307	47
## scale(YR.ISOL, scale = FALSE)	-0.19004	0.87307	47

What it says here is that the most important variables are AREA (positive effect of 4.6) and GRAZE 5 (neg effect of 2.3). Note: here, we only look at the absolute values, not the log transformed.

All it does is take the coefficients, multiply it by the sd of that pred, and divide it by the total variation of the model. And that is how it makes it more relative.

Now, do we need ALL of these variables in the model? YES

BUT an alternative approach (not recommended by Murray), is to remove the least significant term, and keep going until we are only left with significant terms. But what is exactly significant / non significant? And why should we assume that small effects are negligible?

How to choose the best model? → Go to Option 3

AIC (MuMIn)

Instead of doing that, we can use information criteria (AICs). We could compare any combination of terms, but we could NOT compare different types of models (eg lm vs lmm). And we do this using the MuMIn package

What is an AIC?

```
AIC(loyn.glm)
## [1] 370.6935
```

The number itself is POINTLESS, as it depends on scale of variables, number of observations, etc. It means nothing, UNTIL you compare it to another one.

The MuMIn package provides an AIC calculator for small sample sizes (recommended) - AICc:

```
AICc(loyn.glm)
## [1] 375.5824
```

Penalises for the number of parameters (the more parameters, the more complex) and sample size. That is called “parsimony”.

Option 1 - dredging

There is a function called “dredge” (which Murray does not recommend using) that does the following:

The models omit any missing values by default (gets rid of that row). However, that is NOT appropriate for dredge. When a particular predictor is within a model you wanna fit, and has an NA, that row is gone. But if you are fitting a different model that does not have that predictor, it won’t take any NAs out, and the total sample size will be different. And you CANNOT compare models with different sample sizes. So the way this protects you from running models with NAs, is by telling you that you should have defined in your model creation na.action=na.fail. THEN, you could run the dredge, and it would be fine:

So let’s update the model with na.action=na.fail

```
loyn.glm=update(loyn.glm,na.action=na.fail)
```

And let’s run the dredge command again (works now)

```
options(width=300) # displays all characters, without trying to wrap them
# around
dredge(loyn.glm,rank="AICc")

## Fixed term is "(Intercept)"

## Global model call: glm(formula = ABUND ~ scale(log(DIST), scale = FALSE) +
scale(log(DIST),
##      scale = FALSE) + scale(log(AREA), scale = FALSE) + fGRAZE +
##      scale(ALT, scale = FALSE) + scale(YR.ISOL, scale = FALSE),
##      family = gaussian(), data = loyn, na.action = na.fail)
## ---

## Model selection table
##   (Int) fGR scl(ALT,F) scl(log(ARE),F) scl(log(DIS),F) scl(YR.ISO,F) df
logLik AICc delta weight
## 6  19.81  +          3.147                   7 -
175.522 367.4  0.00  0.473
## 8  19.79  +  0.008227  3.128                   8 -
175.435 369.9  2.56  0.132
## 22 19.86  +          3.123                 -0.01664  8 -
175.467 370.0  2.62  0.127
## 14 19.82  +          3.129          0.1999       8 -
175.499 370.1  2.68  0.124
## 16 19.80  +  0.010200  3.093          0.3338       9 -
175.375 372.7  5.29  0.034
## 24 19.82  +  0.006935  3.115                 -0.01154  9 -
175.411 372.7  5.36  0.032
## 30 19.87  +          3.098          0.2429     -0.01832  9 -
175.434 372.8  5.40  0.032
## 23 19.51          0.036230  3.540          0.12480   5 -
181.428 374.1  6.68  0.017
## 21 19.51          3.732          0.13520    4 -
```

```

182.993 374.8 7.39 0.012
## 32 19.84 + 0.008870            3.076      0.3457    -0.01252 10 -
175.347 375.6 8.21 0.008
## 31 19.51          0.034130            3.602     -0.3024    0.12420 6 -
181.380 376.5 9.10 0.005
## 29 19.51          3.867     -0.7971    0.13190 5 -
182.638 376.5 9.10 0.005
## 7 19.51          0.048620            3.935           4 -
187.342 383.5 16.09 0.000
## 15 19.51          0.044900            4.039     -0.5192    5 -
187.226 385.7 18.28 0.000
## 5 19.51          4.247           3 -
189.659 385.8 18.40 0.000
## 13 19.51          4.432     -1.2010    4 -
189.012 386.8 19.43 0.000
## 2 19.74          +
189.827 393.4 25.99 0.000
## 10 19.80          +
189.374 395.1 27.70 0.000
## 18 19.90          +
189.404 395.1 27.76 0.000
## 4 19.69          + 0.018850           7 -
189.550 395.4 28.06 0.000
## 12 19.74          + 0.026890            1.4470    8 -
188.837 396.7 29.36 0.000
## 26 19.98          +
188.842 396.7 29.37 0.000
## 20 19.84          + 0.013120           1.2450    -0.06614 8 -
189.279 397.6 30.25 0.000
## 28 19.89          + 0.021040            1.4720    -0.05157 9 -
188.534 399.0 31.60 0.000
## 27 19.51          0.081670            2.3400    0.18060 5 -
198.902 409.0 41.63 0.000
## 19 19.51          0.070060           0.18350   4 -
200.670 410.1 42.75 0.000
## 17 19.51          0.21120   3 -
203.690 413.8 46.47 0.000
## 25 19.51          1.5390    0.21230   4 -
202.981 414.7 47.37 0.000
## 11 19.51          0.107100            2.5000    4 -
205.772 420.3 52.95 0.000
## 3 19.51          0.095150           3 -
207.358 421.2 53.80 0.000
## 1 19.51          2 -
211.871 428.0 60.59 0.000
## 9 19.51          1.4280           3 -
211.418 429.3 61.92 0.000
## Models ranked by AICc(x)

```

This command ranks 64 different models. The model #6 was the best (see the little number), with an intercept of 22.7. Model 6 had GRAZING in it (cat var get a + if they are there, and nothing if they are not). The model does NOT incl Altitude. It DOES have area, and we can check its parameter estimate on the log scale. That model consumed 7 df (one for each parameter, remembering that there is multiple parameters for grazing, cause it's categorical). In the dredge command, we nominated it to produce the AICc (for small sample sizes). The lower the AIC value, the better. The model with the lowest AIC is displayed above. The second best model has Altitude AND grazing, and used 8 df. The diff in it's AIC is 2.56 (this is the delta / the diff btw it's AIC and the best AIC's). If the AICs are within 2 units of difference, they are considered equally. THEN, you would not necessarily choose the top one, but rather the one with less df, cause it's gonna be the least complex. With the deltas, models that are within 5-10 units are reasonable models. But models that are >10 unit of difference (eg. 23-25) are considered pretty poor models.

In summary dredging has identified that the upper model is considered the best model.

BUT Murray does not like dredging. Why do we have to follow these numbers? Why categorising the models? We need to apply common sense as well, and not be slave to the numbers. At some points, you need to be the adult in the room and be mindful of what the models are proposing you if they are even sensible, and if there are even more sensible models. You may even find that models with dredging are hard to publish. These are just windows that might give you a HINT on the best models, but don't follow them like they were the Bible.

So what are the alternatives? Maybe I could avg all reasonable models, and take the avg Altitude effect, the avg Distance effect, etc. That is called Model averaging. It is the acknowledgment that not ONE model will be THE best, but that a collection of models will give you the best representation.

Murray would like to run a machine learning approach to identify the important variables, and then model THOSE.

Option 2 - model averaging

Let's say we go for the scheme to use the reasonable models, those within 10 AICc. We might wanna weigh them differently according to their AIC -> see column "weight" in the dredge output (it's their avg AIC weights)

What about models that were NOT included, and we still wanna avg them -> see column YR.ISOL, first row: Was not included, so would you have them as a zero instead of omitting them? Ben Hirsch reckons we should not turn these empty rows into zeros, as we would bias the model to zero. BUT what if we only pick the values showing up (and not the empty rows)? Then we would ALSO bias towards the higher values. Murray would rather include those empty values as 0s, and average them as well.

```
loyn.av<-model.avg(dredge(loyn.glm, rank="AICc"),
                      subset=delta<=10)
```

```

## Fixed term is "(Intercept)"
## Fixed term is "(Intercept)"

summary(loyn.av)

##
## Call:
## model.avg(object = dredge(loyn.glm, rank = "AICc"), subset = delta <=
##           10)
##
## Component model call:
## glm(formula = ABUND ~ <12 unique rhs>, family = gaussian(), data = loyn,
## na.action = na.fail)
##
## Component models:
##      df  logLik  AICc delta weight
## 13    7 -175.52 367.38  0.00   0.47
## 123   8 -175.43 369.93  2.56   0.13
## 135   8 -175.47 370.00  2.62   0.13
## 134   8 -175.50 370.06  2.68   0.12
## 1234  9 -175.37 372.66  5.29   0.03
## 1235  9 -175.41 372.73  5.36   0.03
## 1345  9 -175.43 372.78  5.40   0.03
## 235   5 -181.43 374.06  6.68   0.02
## 35    4 -182.99 374.77  7.39   0.01
## 12345 10 -175.35 375.58  8.21   0.01
## 2345   6 -181.38 376.47  9.10   0.01
## 345   5 -182.64 376.48  9.10   0.00
##
## Term codes:
##                               fGRAZE      scale(ALT, scale = FALSE)
## scale(log(AREA), scale = FALSE) scale(log(DIST), scale = FALSE)
## scale(YR.ISOL, scale = FALSE)
##                                     1                      2
## 3                           4                      5
##
## Model-averaged coefficients:
## (full average)
##                               Estimate Std. Error Adjusted SE z value
## Pr(>|z|)
## (Intercept)                 19.806072  0.834094  0.854893 23.168
## < 2e-16 ***
## fGRAZE.L                  -7.887807  2.753956  2.800144  2.817
## 0.00485 **
## fGRAZE.Q                  -5.781742  2.179297  2.218472  2.606
## 0.00916 **
## fGRAZE.C                  -2.442985  2.221937  2.274998  1.074
## 0.28289
## fGRAZE^4                  -0.967038  1.879808  1.926389  0.502
## 0.61567

```

```

## scale(log(AREA), scale = FALSE) 3.153818 0.559354 0.572918 5.505
< 2e-16 ***
## scale(ALT, scale = FALSE)      0.002500 0.011881 0.012106 0.206
0.83640
## scale(YR.ISOL, scale = FALSE) 0.001785 0.036552 0.037005 0.048
0.96153
## scale(log(DIST), scale = FALSE) 0.040840 0.472411 0.483730 0.084
0.93272
##
## (conditional average)
##                                     Estimate Std. Error Adjusted SE z value
Pr(>|z|)
## (Intercept)                  19.80607 0.83409 0.85489 23.168 <
2e-16 ***
## fGRAZE.L                   -8.20365 2.30150 2.35875 3.478
0.000505 ***
## fGRAZE.Q                   -6.01326 1.88344 1.93043 3.115
0.001839 **
## fGRAZE.C                   -2.54081 2.21046 2.26590 1.121
0.262150
## fGRAZE^4                  -1.00576 1.90689 1.95464 0.515
0.606868
## scale(log(AREA), scale = FALSE) 3.15382 0.55935 0.57292 5.505 <
2e-16 ***
## scale(ALT, scale = FALSE)      0.01099 0.02296 0.02347 0.468
0.639637
## scale(YR.ISOL, scale = FALSE) 0.00750 0.07464 0.07557 0.099
0.920940
## scale(log(DIST), scale = FALSE) 0.19750 1.02387 1.04911 0.188
0.850679
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Here, you see two outputs: - Full average = Where it substitutes in 0s where you did not have any. This is the one where it is gonna be biased towards 0. - Conditional average = Where it ONLY included the terms if there was a number. Does not substitute missing terms for 0s. Otherwise, you interpret the value as you would with any other table (sd, p values, conf int, etc)

For the manuscript: If we are gonna show a set of top models, show THE top model, OR the model average (but do not show a set of models if you are not explaining what they actually mean).

Option 3 - pick this one

This option, rather than blindly going through every combination of model, a priori you nominate certain models that speak to particular aspects of the ecology or the system

We could eg propose a model about connectivity, including DIST and LDIST. We could fit a model that includes just those variables, and see what it tells us about Connectivity. We

could then propose a model related to the Habitats, including the Grazing areas. That would be a climate type model. In summary, we should propose models related to different ecological aspects. And we know what they are even before seeing the data. We ask ourselves "Is the grazing area important for the birds?" "Is the climate important?" etc. You'd create as many models as you need (maybe around 10), and then you'd compare them.

When doing this, it is useful to create a NULL model, a CONTROL or benchmark that you compare all models against.

Considering that we got small sample sizes and predictors (not so complex models), we will create few models with interactions. And we will use the function UPDATE, so that we don't have to type everything again. It will just update the formula.

Model 1 - Connectivity model (Distance)

```
loyn.glm1<-  
update(loyn.glm,.~scale(log(DIST),scale=FALSE)*scale(log(LDIST),scale=FALSE))
```

The dot indicates to retain what was on the LEFT side of the model. And on the right handside of the dot, we tell R to replace it by what we have written above

Model 2 - Habitat use (Area and grazing intensity)

```
loyn.glm2<-update(loyn.glm,.~scale(log(AREA),scale=FALSE)*fGRAZE)
```

Model 3 - Habitat use * yrs in isolation

```
loyn.glm3<-  
update(loyn.glm,.~scale(log(AREA),scale=FALSE)*fGRAZE*scale(YR.ISOL,scale=FAL  
SE))
```

Model 4 - Altitude

```
loyn.glm4<-update(loyn.glm,.~scale(ALT,scale=FALSE))
```

Model Null - Null/Control model

```
loyn.null<-update(loyn.glm,.~1)
```

AICc for all created models

Rather than fitting 64 models (Option 2), we have created 5. Now we will compare their AICc's.

```
AICc(loyn.glm1,loyn.glm2,loyn.glm3,loyn.glm4,loyn.null)
```

```
##          df      AICc  
## loyn.glm1  5  433.2034  
## loyn.glm2 11  370.1623  
## loyn.glm3 21  406.9398  
## loyn.glm4  3  421.1775  
## loyn.null  2  427.9690
```

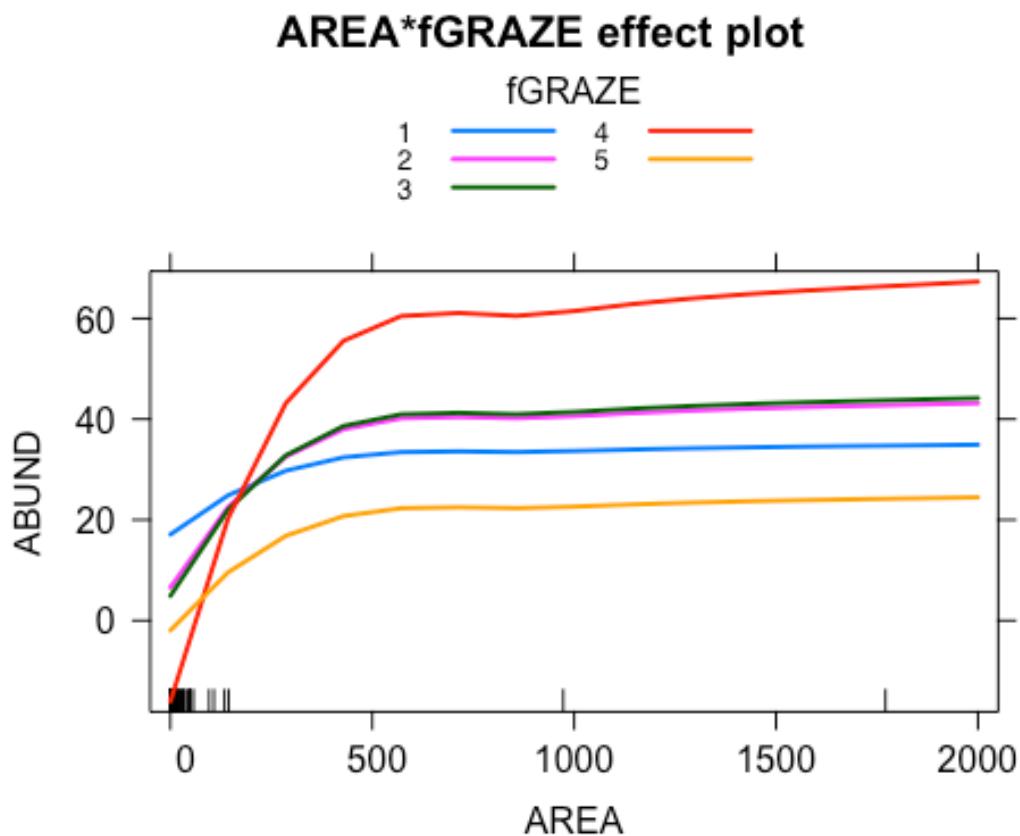
Let's check the results - The 1st model is the connectivity model, which is POORER than the Null. So you would suggest that the model 1 is not the best one (maybe because birds got wings and don't care about connectivity) - The 2nd model seems to be the best model. It has got the lowest AIC, and is better than the NULL model by >10 units. - The third model looks too complex (df=21) - The fourth model's AIC is also too high

Diagnostic plot

Let's create a partial plot to understand the winning model better

```
plot(allEffects(loyn.glm2), multiline=TRUE)

## Warning in Analyze.model(focal.predictors, mod, xlevels, default.levels, :
# the predictor scale(log(AREA), scale = FALSE) is a one-column matrix that was
# converted to a vector
```



Now, the plot has un-done the transformation we did when creating the model. It is telling us that the magnitude of the effect of the trends is different across grazing levels, but that all of the grazing levels affect abundance. Level 4 differs from the rest though, and seems to have the greatest magnitude. Grazing level 4 has a much deeper slope than the others

ANOVA (ignore)

(Not really needed:) We can check with an ANOVA how much the predictors influence bird abundance (dependent variable).

```
anova(loyn.glm2)

## Analysis of Deviance Table
##
## Model: gaussian, link: identity
##
## Response: ABUND
##
## Terms added sequentially (first to last)
##
##
##                                     Df Deviance Resid. Df Resid. Dev
## NULL                               55   6337.9
## scale(log(AREA), scale = FALSE)    1     3471.0      54   2866.9
## fGRAZE                            4     1136.5      50   1730.4
## scale(log(AREA), scale = FALSE):fGRAZE 4     253.8       46   1476.6
```

Murray normally does not do ANOVA tables as he finds that he gets enough out of the summary(lm) table alone. The ANOVA would ONLY say that there was a sign interaction, but you could not see where that interaction is.

summary() of the model

Let's have a closer look at what seems to be the best model

```
summary(loyn.glm2)

##
## Call:
## glm(formula = ABUND ~ scale(log(AREA), scale = FALSE) + fGRAZE +
##       scale(log(AREA), scale = FALSE):fGRAZE, family = gaussian(),
##       data = loyn, na.action = na.fail)
##
## Deviance Residuals:
##      Min        1Q        Median         3Q        Max 
## -16.3615   -2.3807   -0.2449    2.6181   11.3529
##
## Coefficients:
##                                     Estimate Std. Error t value
## Pr(>|t|)                                 20.3941   0.9423  21.643 <
## (Intercept)                           4.1092   0.6607   6.220
## scale(log(AREA), scale = FALSE)          1.35e-07 *** 
## fGRAZE.L                                -10.1142   2.3108  -4.377
## 6.86e-05 ***
```

```

## fGRAZE.Q           -5.1929   2.1142  -2.456
0.0179 *
## fGRAZE.C          -3.7784   2.1475  -1.759
0.0851 .
## fGRAZE^4          -0.8784   1.8268  -0.481
0.6329
## scale(log(AREA), scale = FALSE):fGRAZE.L   2.0406   1.2998   1.570
0.1233
## scale(log(AREA), scale = FALSE):fGRAZE.Q   -2.9704   1.2848  -2.312
0.0253 *
## scale(log(AREA), scale = FALSE):fGRAZE.C   -2.7107   1.7290  -1.568
0.1238
## scale(log(AREA), scale = FALSE):fGRAZE^4   -2.4128   1.5491  -1.558
0.1262
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 32.1006)
##
## Null deviance: 6337.9 on 55 degrees of freedom
## Residual deviance: 1476.6 on 46 degrees of freedom
## AIC: 364.16
##
## Number of Fisher Scoring iterations: 2

```

- Intercept Estimate: Avg nr of birds in Grazing 1 = 25. It says it's sign bec it just checks if that Intercept value is sign diff from zero (it always will be). (!) IGNORE THE FIRST P-VALUE
- In grazing 2, there is 2 less birds in comparison to grazing 1 (NOT a multiplier 2*Intercept, as this is not log scale - I THINK)
- In grazing 3, there is 2.5 less birds in comparison to grazing 1 etc
- scale(log(AREA), scale = FALSE) tells us that there is an effect of area on grazing 1 (not 2, 3, etc; but we assume it's the same)
- The trend/effect btw abundance and area of grazing 3 is NOT diff to grazing 1, nor for 1-3. It IS for 1-4.
- The p value tells us that the trend WAS significant (see Intercept row: 4.74e-14 ***), but it does not change that much across grazing levels (that's why it is ns in the following grazing categories).

emmeans - Pairwise comparisons

```
emtrends(loyn.glm2, pairwise~fGRAZE, var="log(AREA)")
```

```

## $emtrends
## fGRAZE log(AREA).trend    SE  df asympt.LCL asympt.UCL
## 1             1.80 0.893 Inf   0.0486     3.55
## 2             3.70 1.012 Inf   1.7137     5.68
## 3             3.97 1.246 Inf   1.5238     6.41
## 4             8.42 2.413 Inf   3.6871    13.15

```

```

##   5          2.67 1.309 Inf    0.1005      5.23
##
## Confidence level used: 0.95
##
## $contrasts
##   contrast estimate   SE  df z.ratio p.value
##   1 - 2       -1.897 1.35 Inf -1.405  0.6241
##   1 - 3       -2.167 1.53 Inf -1.413  0.6192
##   1 - 4       -6.617 2.57 Inf -2.572  0.0757
##   1 - 5       -0.867 1.58 Inf -0.547  0.9824
##   2 - 3       -0.270 1.61 Inf -0.168  0.9998
##   2 - 4       -4.719 2.62 Inf -1.804  0.3713
##   2 - 5        1.030 1.65 Inf  0.623  0.9715
##   3 - 4       -4.450 2.72 Inf -1.638  0.4727
##   3 - 5        1.300 1.81 Inf  0.719  0.9521
##   4 - 5        5.750 2.75 Inf  2.095  0.2223
##
## P value adjustment: tukey method for comparing a family of 5 estimates

```

This shows us what the slope is at the diff grazing levels (=“trend” column in \$emtrends). If we want to compare the slopes to each other, “contrasts” provides multiple, pairwise comparisons. Here, we also see that grazing level 4, although it got the most extreme slope, is not sign diff from the other grazing levels

So now we are not only able to get the individual slopes, but also to check how much each grazing level differs from one another in relation to the abundance of birds.

Plot

We start by creating the grid of prediction values that we want to predict for, the 5 grazing levels, and area from min to max.

```

loyn.grid<-with(loyn,list(fGRAZE=levels(fGRAZE),
                           AREA=seq(min(AREA),max(AREA),len=100)))

loyn.grid

## $fGRAZE
## [1] "1" "2" "3" "4" "5"
##
## $AREA
##   [1] 0.10000 17.98788 35.87576 53.76364 71.65152 89.53939
107.42727 125.31515 143.20303 161.09091 178.97879 196.86667 214.75455
232.64242 250.53030 268.41818 286.30606 304.19394 322.08182 339.96970
357.85758 375.74545 393.63333 411.52121 429.40909 447.29697
##  [27] 465.18485 483.07273 500.96061 518.84848 536.73636 554.62424
572.51212 590.40000 608.28788 626.17576 644.06364 661.95152 679.83939
697.72727 715.61515 733.50303 751.39091 769.27879 787.16667 805.05455
822.94242 840.83030 858.71818 876.60606 894.49394 912.38182

```

```
## [53] 930.26970 948.15758 966.04545 983.93333 1001.82121 1019.70909
1037.59697 1055.48485 1073.37273 1091.26061 1109.14848 1127.03636 1144.92424
1162.81212 1180.70000 1198.58788 1216.47576 1234.36364 1252.25152 1270.13939
1288.02727 1305.91515 1323.80303 1341.69091 1359.57879 1377.46667
## [79] 1395.35455 1413.24242 1431.13030 1449.01818 1466.90606 1484.79394
1502.68182 1520.56970 1538.45758 1556.34545 1574.23333 1592.12121 1610.00909
1627.89697 1645.78485 1663.67273 1681.56061 1699.44848 1717.33636 1735.22424
1753.11212 1771.00000
```

Now we will calculate the predictions for ALL combinations with emmeans

```
newdata=emmeans(loyn.glm2, ~AREA|fGRAZE, at=loyn.grid, type="response") %>%
  as.data.frame
```

```
head(newdata)
```

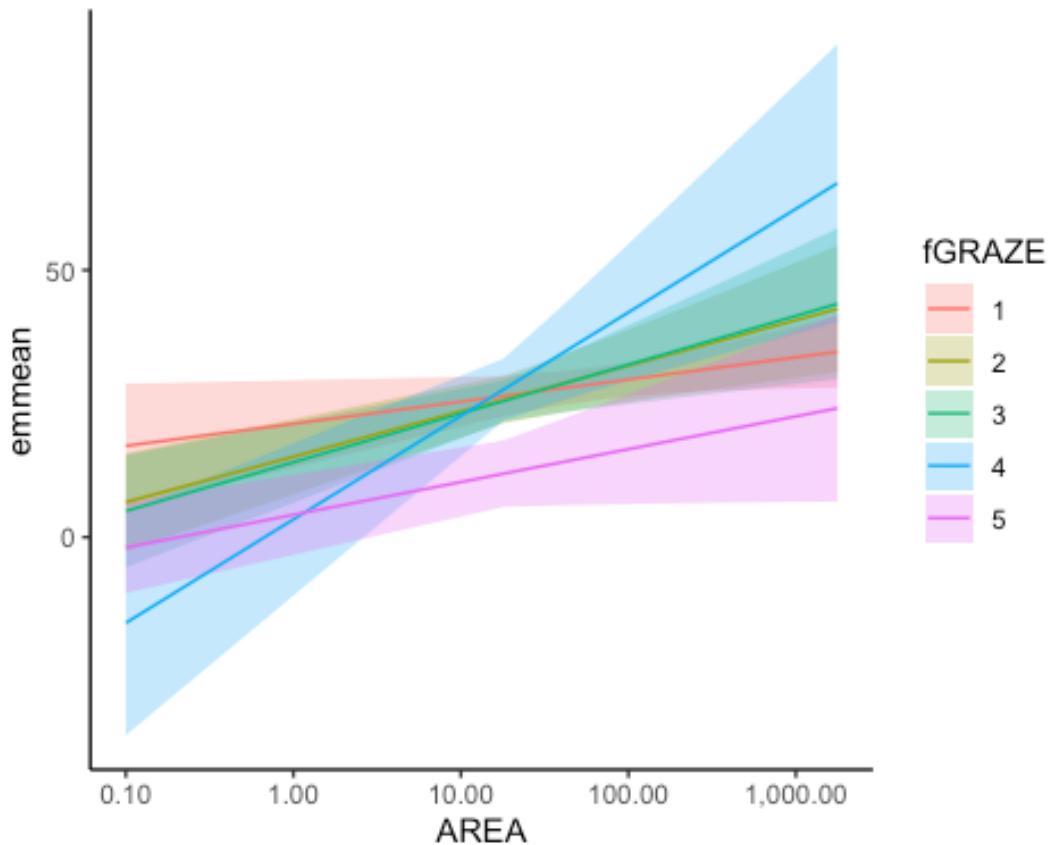
	AREA	fGRAZE	emmean	SE	df	asymp.LCL	asymp.UCL
## 1	0.10000	1	17.09903	5.932885	Inf	5.470792	28.72727
## 2	17.98788	1	26.44362	1.907861	Inf	22.704283	30.18296
## 3	35.87576	1	27.68608	1.638797	Inf	24.474092	30.89806
## 4	53.76364	1	28.41412	1.574814	Inf	25.327543	31.50070
## 5	71.65152	1	28.93103	1.578813	Inf	25.836611	32.02544
## 6	89.53939	1	29.33212	1.610335	Inf	26.175919	32.48832

- The predictions are called emmean
- We also got asymptotic upper and lower conf bands

Let's create the plot

```
plot<-ggplot(newdata, aes(y=emmean, x=AREA, color=fGRAZE, fill=fGRAZE))+#
  geom_ribbon(aes(ymin=asymp.LCL,ymax=asymp.UCL),color=NA,alpha=0.3)+ #
  color=NA removes the outline of the ribbon
  geom_line()+
  scale_x_log10(labels=scales::comma)+ # We are creating a Log10 axis because
we had log transformed area previously, as the raw data was right-skewed (see
Line 110)
  # "labels=scales::comma" displays the x axis values in full (otherwise it
would shorten them into 10*e whatever)
  theme_classic()

plot
```



PROBLEM: We are predicting negative birds for some of the grazing levels! Look at grazing level 4! So, our model met all the assumptions, and was seemingly performing well, but WTF. This is not realistic. Our model is predicting <0. What can we do about it? Way back at the start, we fitted two other models, one log gaussian, and one gamma model. Either of those models would have prevented this.

SO, if we ever get sth like this with OUR data, and get predictions <0, go for a log gaussian, or a gamma model, which prevent <0 values every time we back-transform them (eg which happens automatically for plotting)

A Gamma model would be the same procedure, but “emmean” (Gaussian model) would be called “response”, so we would have to replace the “emmean” for “response” (for example for plotting).

HOMEWORK (IF WANTED): RE-RUN THE MODEL WITH A GAMMA DISTRIBUTION

5_LM_CatVar

Sara Kophamel

02/12/2020

Straight multiple linear regression with categorical variables

Presentation:

file:///Users/sara/Documents/PhD/HELP/STATISTICS%20and%20GRAPHS/-%20R%20and%20STATS%20COURSES/2020%20Murray%20Logan%20R%20Course/Logan_Course/docs/pres7.2.html#1

Rmd file (copy pasted below): glm_example5

We are dealing with categorical variables. In this example, we see that Treat has got 3 categories. We have to transform them into numerical variables (see Dummy 1,2,3) Treat -> Dummy 1 Dummy 2 Dummy 3 A 1 0 0 A 1 0 0 B 0 1 0 C 0 1 0 C 0 0 1

We got an intercept being the mean of the 1st group, and then we got two effects: B to A, and C to A, which are called "treatment effects". They are applied to an analysis when you run for example an ANOVA.

Preparations

Load the necessary libraries

```
library(car)      #for regression diagnostics
library(broom)    #for tidy output
library(ggfortify) #for model diagnostics
library(sjPlot)   #for outputs
library(knitr)    #for kable
library(effects)  #for partial effects plots
library(emmeans)  #for estimating marginal means
library(ggeffects) #for plotting marginal means
library(MASS)     #for glm.nb
library(MuMIn)    #for AICc
library(tidyverse) #for data wrangling
library(modelr)   #for auxillary modelling functions
library(performance) #for residuals diagnostics
library(see)      #for plotting residuals
library(DHARMa)   #for residual diagnostics plots
library(patchwork) #grid of plots
library(scales)   #for more scales
```

Scenario

Here is a modified example from @Quinn-2002-2002. Day and Quinn (1989) described an experiment that examined how rock surface type affected the recruitment of barnacles to a rocky shore. The experiment had a single factor, surface type, with 4 treatments or levels: algal species 1 (ALG1), algal species 2 (ALG2), naturally bare surfaces (NB) and artificially scraped bare surfaces (S). There were 5 replicate plots for each surface type and the response (dependent) variable was the number of newly recruited barnacles on each plot after 4 weeks.

Six-plated barnacle

Six-plated barnacle

Format of day.csv data files

TREAT	BARNACLE
ALG1	27
..	..
ALG2	24
..	..
NB	9
..	..
S	12
..	..
TREAT	Categorical listing of surface types. ALG1 = algal species 1, ALG2 = algal species 2, NB = naturally bare surface, S = scraped bare surface.
BARNACLE	The number of newly recruited barnacles on each plot after 4 weeks.

Read in the data

As we are going to treat Treatment as a categorical predictor, we will specifically declare it as such straight after importing the data.

```
day = read_csv('data/day.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   TREAT = col_character(),
##   BARNACLE = col_double()
## )

glimpse(day) # Here, Treatment is considered a character
```

```

## Rows: 20
## Columns: 2
## $ TREAT    <chr> "ALG1", "ALG1", "ALG1", "ALG1", "ALG1", "ALG2", "ALG2",
## $ ALG...
## $ BARNACLE <dbl> 27, 19, 18, 23, 25, 24, 33, 27, 26, 32, 9, 13, 17, 14,
22, 1...
day <- day %>%
  mutate(TREAT = factor(TREAT)) # So we convert Treatment to a factor

glimpse(day)

## Rows: 20
## Columns: 2
## $ TREAT    <fct> ALG1, ALG1, ALG1, ALG1, ALG1, ALG2, ALG2, ALG2,
ALG2, ...
## $ BARNACLE <dbl> 27, 19, 18, 23, 25, 24, 33, 27, 26, 32, 9, 13, 17, 14,
22, 1...

```

Exploratory data analysis

Model formula:

$$\$y_i \sim \text{Pois}(\lambda_i) \quad \mu_i = \boldsymbol{\beta} \bf{X}_i$$

where $\boldsymbol{\beta}$ is a vector of effects parameters and \bf{X} is a model matrix representing the intercept and treatment contrasts for the effects of Treatment on barnacle recruitment.

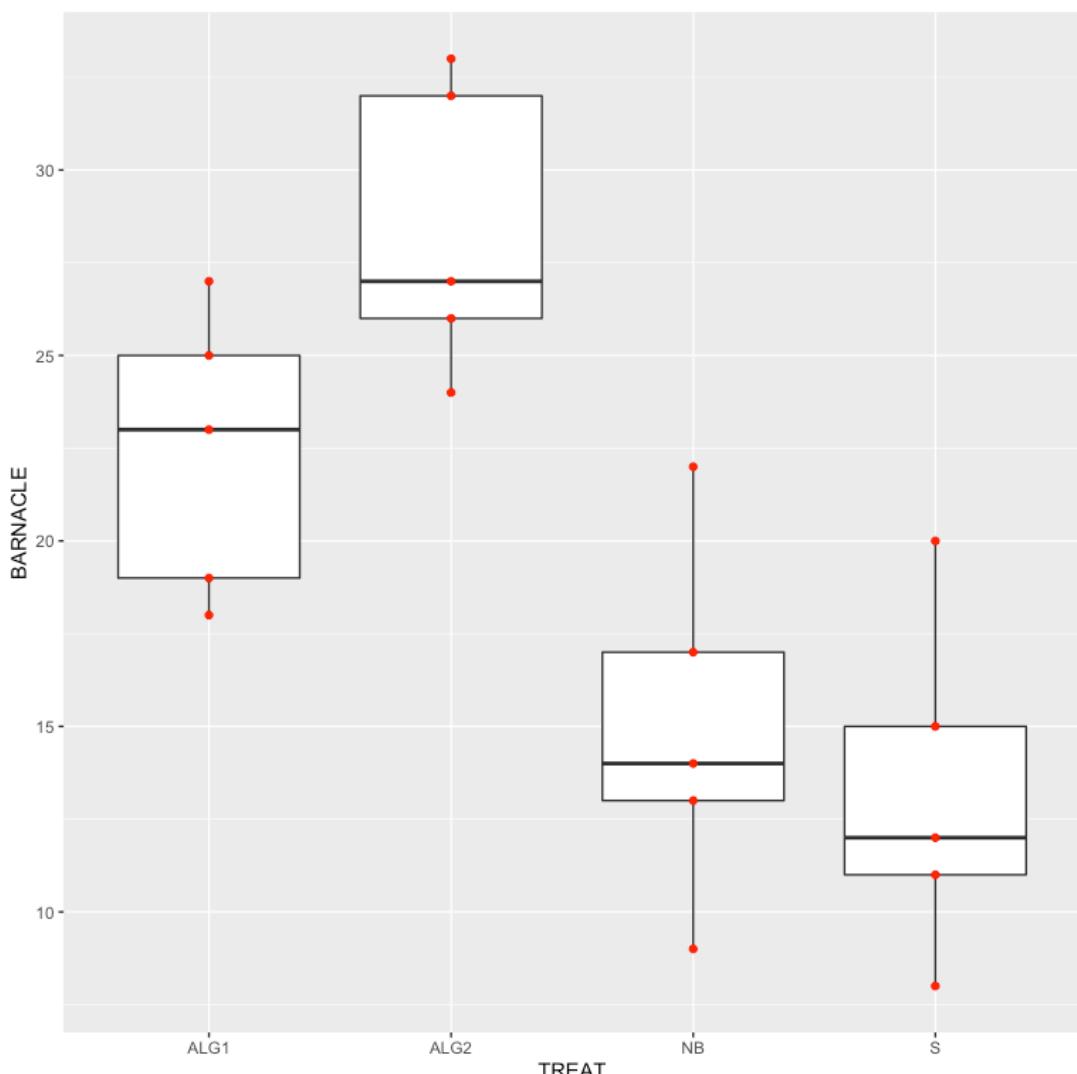
Diagnostic plots

Unlike previous examples, where we use scatterplots as diagnostic plots, here we use boxplots instead

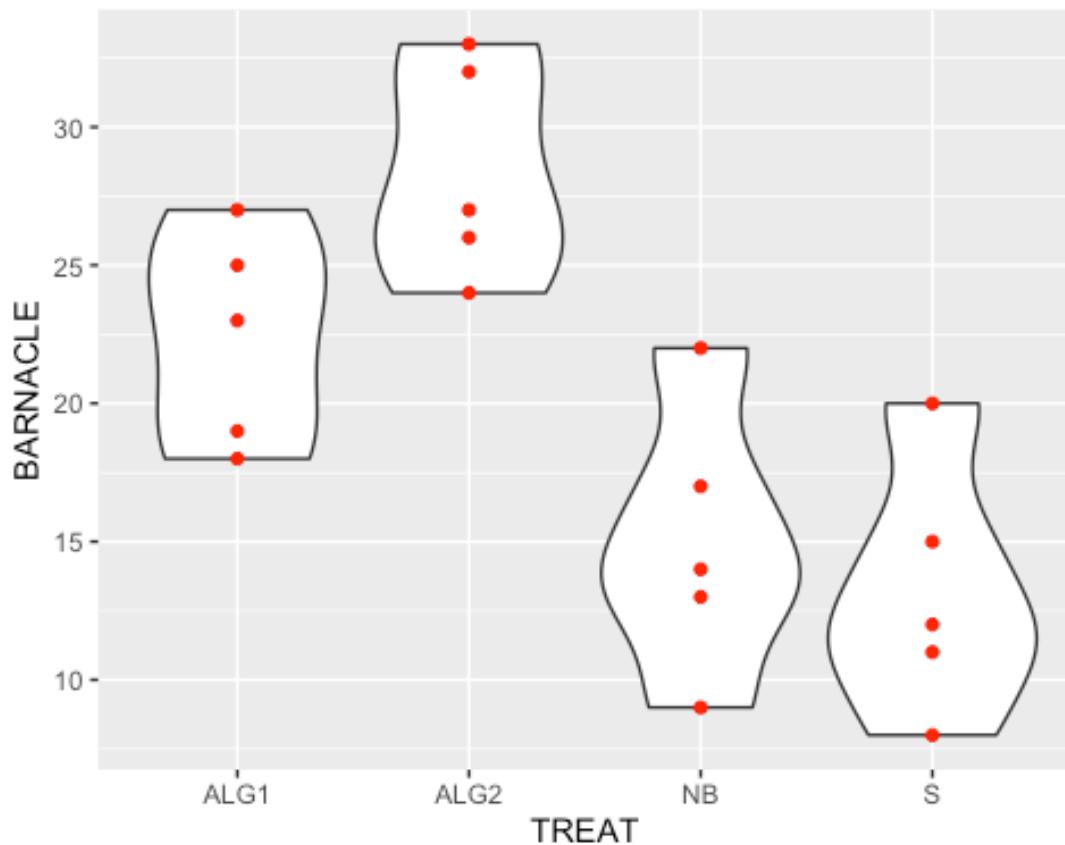
```

ggplot(day, aes(y=BARNACLE, x=TREAT)) +
  geom_boxplot() +
  geom_point(color='red')

```



```
ggplot(day, aes(y=BARNACLE, x=TREAT)) +  
  geom_violin() +  
  geom_point(color='red')
```



Fit the model

Let's go for a Poisson distribution to avoid <0 values when transforming.

Assumptions

- Dispersion: We consider that mean and variance ARE related (\neq Gaussian distribution). So we have to make sure that is the case.
- Normality and homogeneity of variance don't matter, as we are not using numerical data

Let's fit our Poisson model:

When we have been looking at parameters / coefficients that we estimate, different scales would make it a bit complicated. So we could in theory fit it first as a Gaussian, and then as a Poisson (and handle the back transformations), and see how it goes. For the purpose of this example though, we go with the Poisson only.

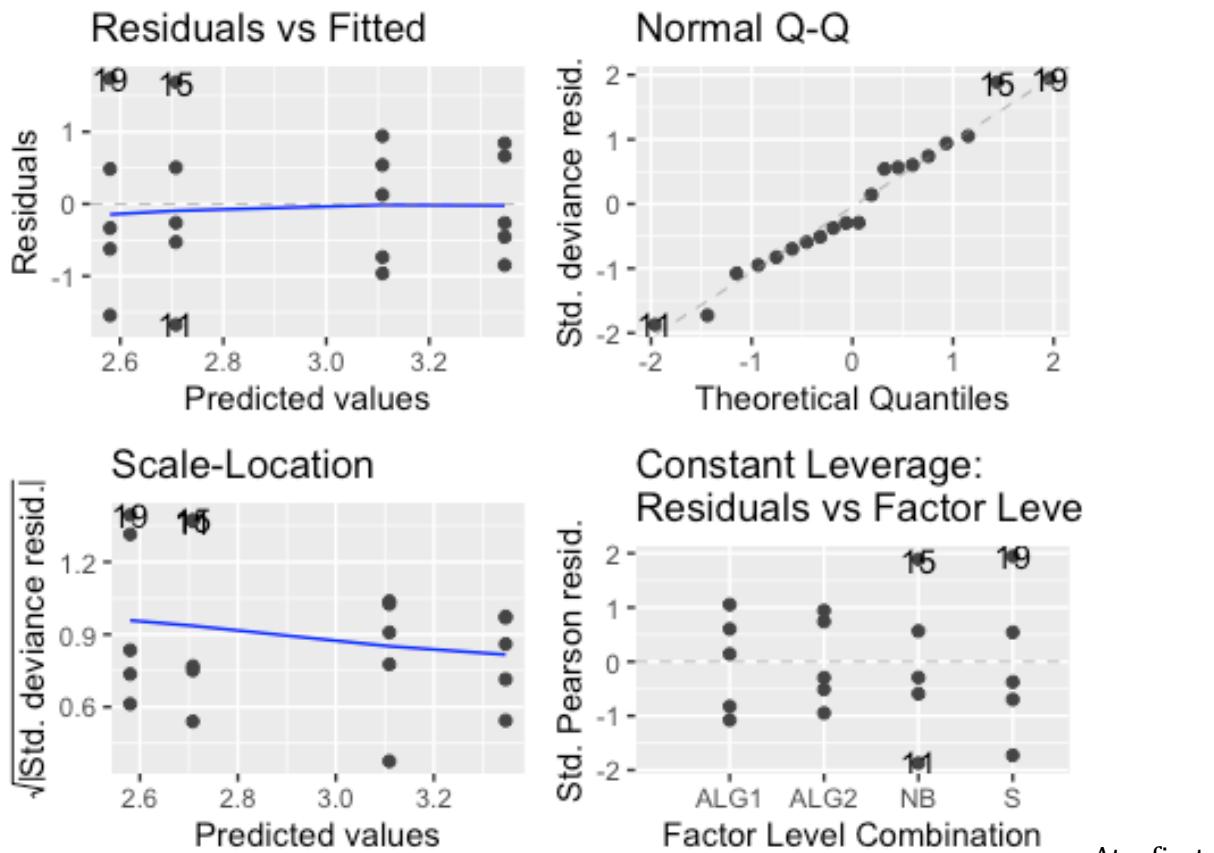
```
# day.glm1<-glm(BARNACLE~TREAT,data=day,family="poisson") # Don't forget the
# Link=log (see below)
day.glm1<-glm(BARNACLE~TREAT,data=day,family="poisson"(link="log"))
```

Model validation

autoplot

```
autoplot(day.glm1)
```

```
## Warning: `arrange_()` is deprecated as of dplyr 0.7.0.
## Please use `arrange()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```



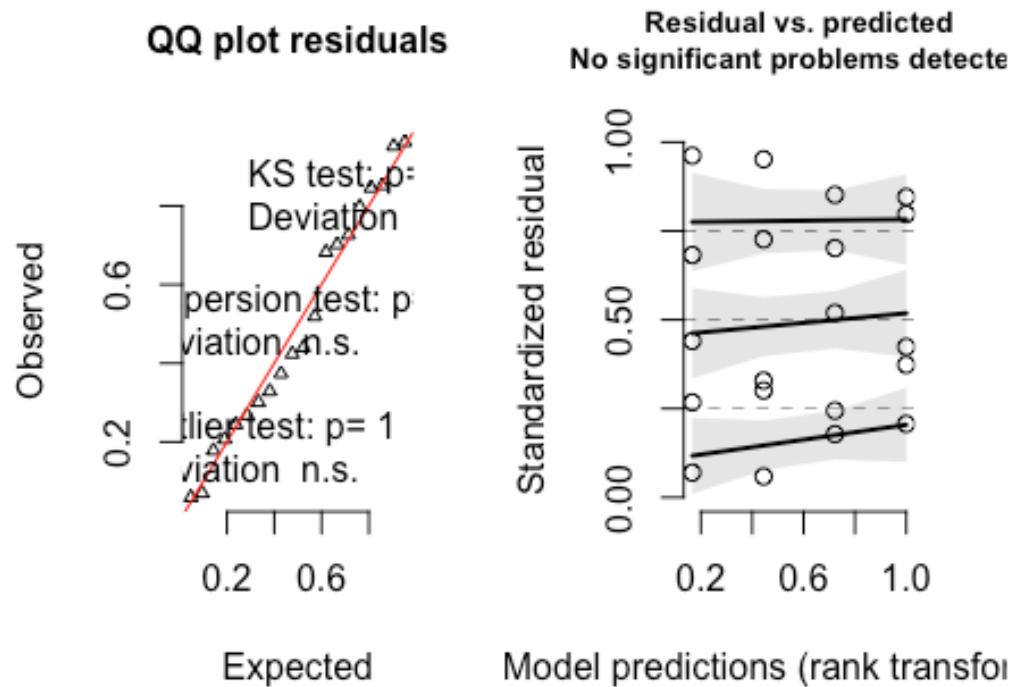
sight, nothing looks alarming

DHARMA residuals

```
day.resids<-simulateResiduals(day.glm1, plot=TRUE)
```

At a first

DHARMA residual diagnostics

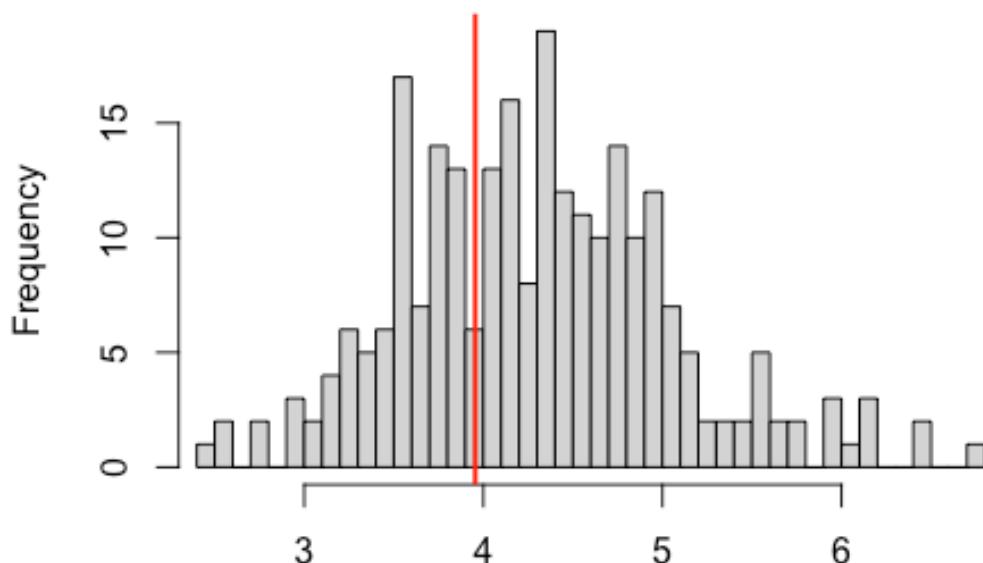


In addition to the QQ normal plot and the residual plot, it has provided a Dispersion test for overdispersion. Dispersion = Variance/mean. We want it to be 1. So the closer to 1, the better. In this case, the test says that it does not deviate from 1 (no overdispersed data). If the p value was <0.05 in any of the tests, you'd be concerned.

Double-checking the dispersion test

```
testDispersion(day.resids)
```

DHARMA nonparametric dispersion test via sd of residuals fitted vs. simulated



Simulated values, red line = fitted model. p-value (two.sided) = 0.

```
##  
## DHARMA nonparametric dispersion test via sd of residuals fitted vs.  
## simulated  
##  
## data: simulationOutput  
## ratioObsSim = 0.91946, p-value = 0.68  
## alternative hypothesis: two.sided
```

The test says that the data is slightly overdispersed, but that is an artefact.

How do you get overdispersion? - We have nominated a Poisson distribution, which expects a certain amount of variance. If our data is more varied than that, we have over dispersion - We know that these models are low-dimensional representations - Substrate is not the only reason for which the data is overdispersed. The model we account for does not account for ALL sorts of variances, so our model might be more varied than if it was a Poisson distribution - We got a lot more 0s than a Poisson distribution would expect. In this case, we might not have detected all the barnacles, so imperfect detection can also lead to extra 0s, which leads to overdispersion. - Clumpiness / Aggregation of the barnacles (so you either get lots or you get any). That's an additional source of variability.

How we account for overdispersion depends on what we think the cause is. - If we think the cause is zero inflation (additional 0s), then we tackle it with a model that can deal with 0 inflation, like a negative binomial - If we have more variables we think could lead to a

response of our dependent variable, we can add the, to the model - We can create an observational unit random level effect = We create an additional variable that sucks up all the noise. This, however, might lead to statistical shrinkage (when your values shrink to zero). But you could still work with this data.

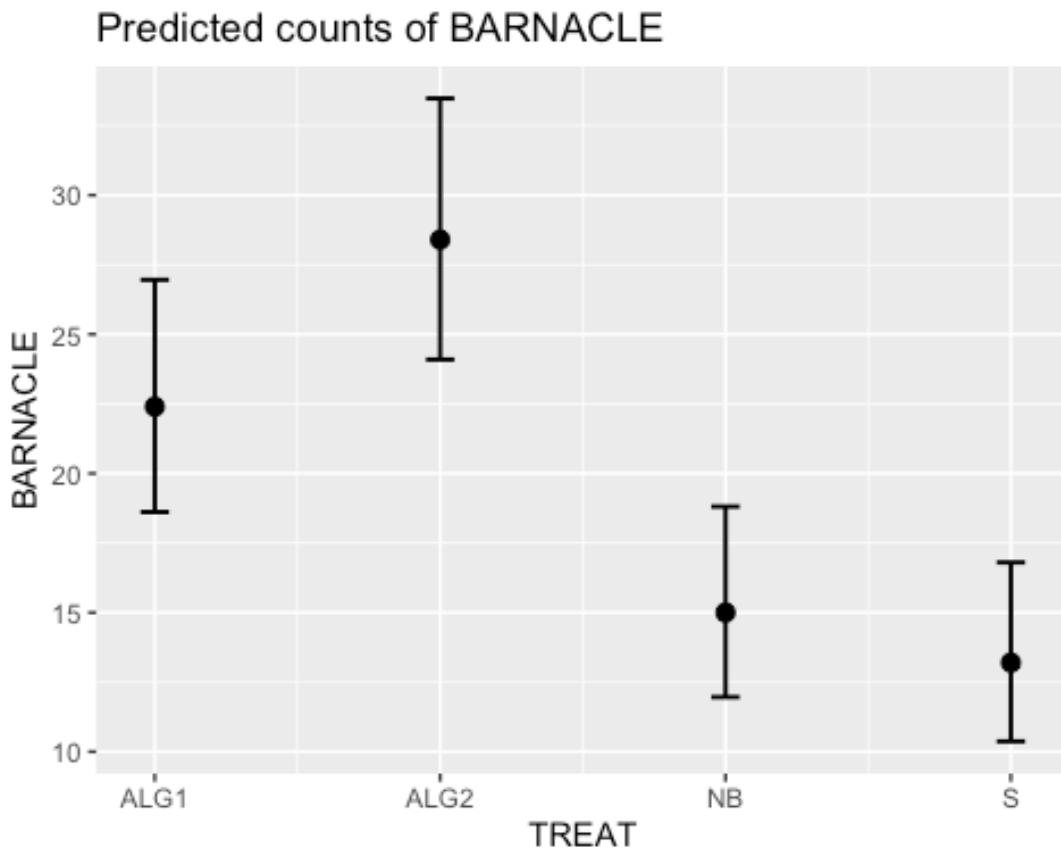
What are the consequences of overdispersion? - The SEs will be underestimated / will be too small, which will lead to a more powerful test than needed = Things will appear to be more significant than they really are. You can't work with this.

Models that are underdispersed are a LOT rarer, and are usually much more conservative than overdispersed models. So you usually don't have to worry about them.

Essentially, we look at the results and hope they are reliable.

Partial plots

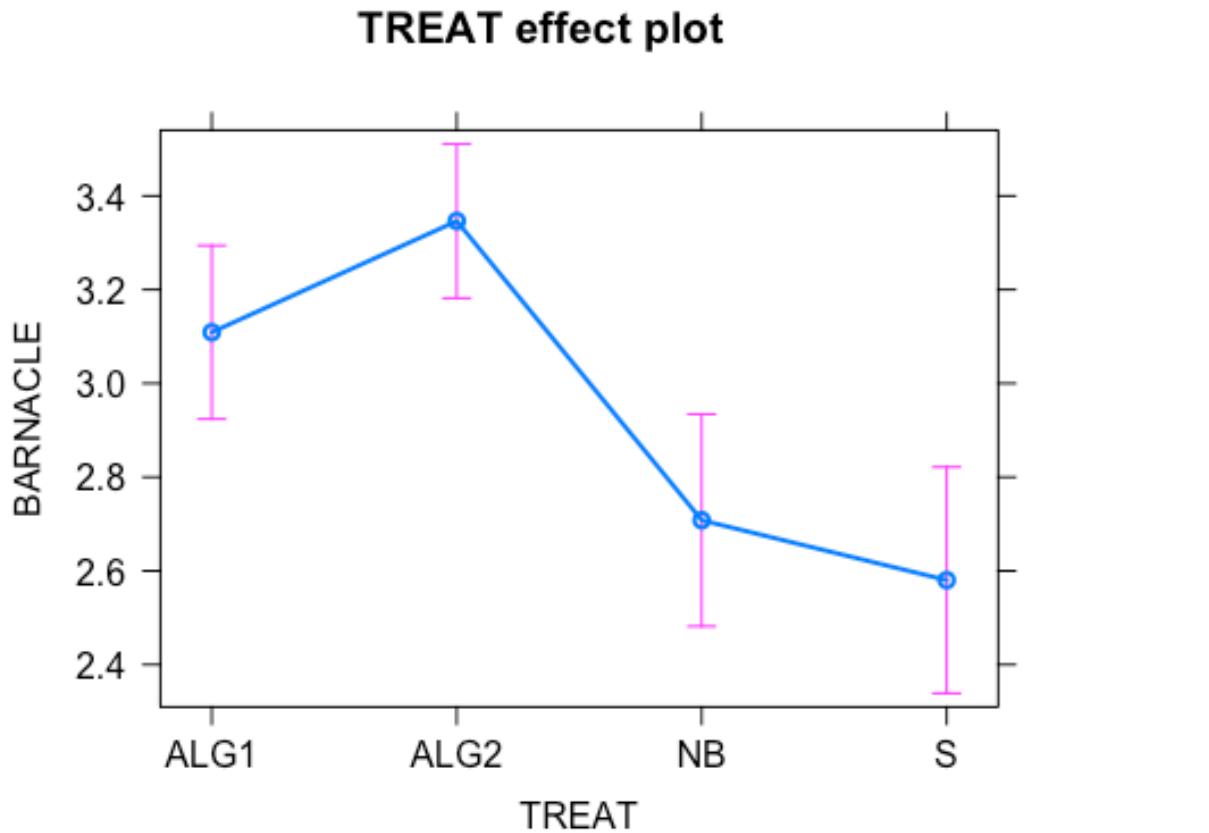
```
plot_model(day.glm1, type="eff")  
## $TREAT
```



This plot shows us our predicted values from the model. It has already back transformed the data, which is usually a useful thing. BUT in this case, our results will actually be on a log-scale, so

we would like to have the graph on a log-scale. We want to check the coefficients, which are better displayed on a log scale. We will thus plot the model in a slightly diff way:

```
allEffects(day.glm1, transformation=NULL) %>% plot # The plot_model() command
wouldn't allow for transformation=NULL, which undoes the log-transformation
```



investigation / hypothesis testing {.tabset .tabset-faded}

```
summary(day.glm1)
## Model
## Call:
## glm(formula = BARNACLE ~ TREAT, family = poisson(link = "log"),
##      data = day)
##
## Deviance Residuals:
##      Min      1Q   Median      3Q      Max
## -1.6748 -0.6522 -0.2630  0.5699  1.7380
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 3.10906   0.09449 32.903 < 2e-16 ***
## TREATALG2    0.23733   0.12638  1.878 0.060387 .
## TREATNB     -0.40101   0.14920 -2.688 0.007195 **
##
```

```

## TREATS      -0.52884    0.15518   -3.408 0.000654 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
## Null deviance: 54.123  on 19  degrees of freedom
## Residual deviance: 17.214  on 16  degrees of freedom
## AIC: 120.34
##
## Number of Fisher Scoring iterations: 4

```

- The Intercept is the avg for the 1st group, the ALG1 group. The avg is 3.1 (double check on graph). If we hadn't undone the transformation, we wouldn't have been able to compare the intercept with the plot (it would have been in raw data x axis). It says it's sign bec it just checks if that Intercept value is sign diff from zero (it always will be). (!) IGNORE THE FIRST P-VALUE
- The Estimate value of TREATALG2, which is 0.23733 is the distance btw ALG1 and ALG2
- TREATS is 0.52 units less than ALG1 = It's gone down 0.53 units on the log scale (y axis, double check on graph to understand better)

What we can say from this analysis is that substrate type seems to affect the recruitment of barnacles.

Predictions

We could now ask for an ANOVA table, but it would tell us a lot less, as it does not do pairwise post hoc comparisons. Plus ANOVA tables can be heavily affected by imbalances in sample sizes (they also don't affect NAs). Let's do an alternative method.

Option 1

So let's go for a proper check, the 95% conf int:

```

confint(day.glm1)

## Waiting for profiling to be done...

##           2.5 %     97.5 %
## (Intercept) 2.917958584 3.2887098
## TREATALG2   -0.009471895 0.4865512
## TREATNB     -0.696806649 -0.1108759
## TREATS      -0.837588039 -0.2280991

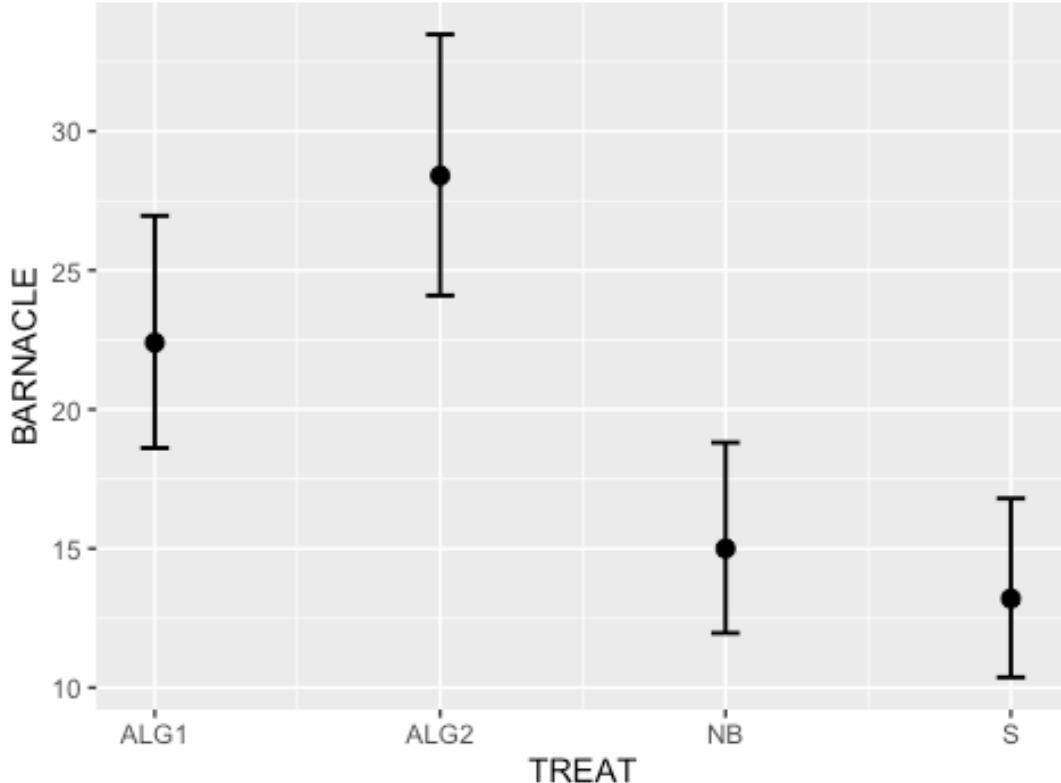
```

IF the conf int are around 0 (in log transformed data), or around 1 (in back-transformed data), we have a sign effect (see below, Option 2 explanation)

```
plot_model(day.glm1,type="eff")
```

```
## $TREAT
```

Predicted counts of BARNACLE



The function `tidy` get's the bits of the model and back transforms them, and also displays the CI

Option 2 (recommended)

```
tidy(day.glm1, conf.int=TRUE, exponentiate=TRUE)
```

```
## # A tibble: 4 x 7
##   term      estimate std.error statistic p.value conf.low conf.high
##   <chr>     <dbl>    <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept) 22.4     0.0945    32.9  1.98e-237   18.5     26.8
## 2 TREATALG2    1.27     0.126     1.88  6.04e- 2    0.991     1.63
## 3 TREATNB     0.670     0.149    -2.69  7.20e- 3    0.498     0.895
## 4 TREATS      0.589     0.155    -3.41  6.54e- 4    0.433     0.796
```

gathers the data into a table, puts CI, and allows to back transform from a Log by exponentiating the data.

- Intercept estimate: 22.4 is the actual nr of newly recruited barnacles on the ALG1 surface. It has a conf int of 18.5-27ish (see graph)
- ALG2 estimate has 27%ish more barnacles ($1.27 \times 22.4 = \text{nr of barnacles in ALG2}$) (!) This is a MULTIPLIER. There is 27% more recruitment on ALG2 surface than ALG1 (but it is still ns)

- NB has 33% decline
- S has 42% decline

The p values are not as clear as the conf int But when back transforming from a log, we are not looking for intervals that include a 0, but for intervals that include a 1. So the ALG2 includes a 1, and NB and S don't include a 1. That makes NB and S sign from ALG1. ALG2 is not sign diff from ALG1

Post-hoc test (Tukey's)

Where are the differences btw groups?

We can now do the following - Compare ev group to every other group in a pairwise test manner - We could specifically suggest some comparisons - we'd have to account for increases in Type 1 error effects though. Ev hypothesis we test, there is a 5% chance that you make an error if you reject the H0. If for example p value was 0.03 and you reject H0, there is a 5% chance that you have made an error, and that your sample unfortunately very different from the population by chance. The more times you check this, the lower the error rate becomes. - Tukey's test: Will compare all groups, and insure that the error rate does not inflate >0.05. It will not inflate the error rate. The cost of fixing the family error rate to 0.05 though is that each family will be less identifiable; ie. it will be harder to identify sign effects.

```
day.glm1 %>% emmeans(pairwise~TREAT, type="response") # pairwise~ calculates
pairwise tests, in addition to calculating the marginal means

## $emmeans
##   TREAT rate    SE  df asymp.LCL asymp.UCL
##   ALG1  22.4 2.12 Inf     18.6      27.0
##   ALG2  28.4 2.38 Inf     24.1      33.5
##   NB    15.0 1.73 Inf     12.0      18.8
##   S     13.2 1.62 Inf     10.4      16.8
##
## Confidence level used: 0.95
## Intervals are back-transformed from the log scale
##
## $contrasts
##   contrast    ratio    SE  df z.ratio p.value
##   ALG1 / ALG2 0.789 0.0997 Inf -1.878  0.2375
##   ALG1 / NB   1.493 0.2228 Inf  2.688  0.0362
##   ALG1 / S    1.697 0.2633 Inf  3.408  0.0037
##   ALG2 / NB   1.893 0.2703 Inf  4.472  <.0001
##   ALG2 / S    2.152 0.3205 Inf  5.143  <.0001
##   NB / S     1.136 0.1918 Inf  0.757  0.8735
##
## P value adjustment: tukey method for comparing a family of 4 estimates
## Tests are performed on the log scale
```

The contrasts compare each group in a pairwise manner - ALG1 vs NB has a p value<0.05. They are nearly 50% different. - ALG2 vs NB has nearly a 90% diff (and is sign diff) - ALG2 and S are nearly 100% diff (and is sign diff) - NB vs S do not seem sign diff

The alternative that will not loose power, and allows you to do other things, is called "Planned Contrast".

Planned contrasts

Example: If we got A,B,C,D, we would test for sign diff btw A-B B-C NOT for A-C, as this would not be necessary after knowing what A-B and B-C are

Define your own matrix for the pairwise comparisons:

We will define the following:

Comp1	Comp2	Comp3
ALG1 1 0 0.5		
ALG2 -1 0 0.5	NB 0 1 -0.5	S 0 -1 -0.5

The (-)1 and define "we want to compare these" (must add up to 0). The 0 define "We don't want to compare these". In Comp3, the 0.5s mean that we are creating the AVG btw ALG1 and ALG2 (so 0.5 and 0.5), and we are creating the AVG btw NB and S (NB+S="Bare substrate") (so -0.5 and -0.5). Note that we are comparing ALG1+ALG2 against NB+S.

Each column must add up to 0. The positives must add up to 1, and the negatives must add up to 1 too.

The matrix determines what we want to compare. We need it to tell R, when it is doing the pairwise comp, to know where those comparisons are. We are making them independent

...now in code:

```
cmat<-cbind("Alg-Alg2"=c(1,-1,0,0),
             "NB_S"=c(0,0,1,-1),
             "Alg_Bare"=c(0.5,0.5,-0.5,-0.5))

cmat

##      Alg-Alg2 NB_S Alg_Bare
## [1,]      1    0     0.5
## [2,]     -1    0     0.5
## [3,]      0    1    -0.5
## [4,]      0   -1    -0.5
```

We essentially define the following:

We tell R to compare:

- a) ALG1 vs ALG2

- b) NB vs S
- c) average of ALG1+ALG2 vs NB+S

By doing only 3 comparisons (and not comparing ALL groups against each other), we reduce the likelihood of inflating the Type 1 error too much (cause if ALG1>ALG2, and ALG2>NB, then we already know that ALG1>NB, which would inflate the Type 1 error unnecessarily. So we don't have to tell R to test ALG1>NB)

```
coef(day.glm1) # FYI
## (Intercept) TREATALG2 TREATNB TREATS
## 3.1090610 0.2373282 -0.4010108 -0.5288441
```

Displaying the cross product of the matrix

```
crossprod(cmat)
##          Alg-Alg2 NB_S Alg_Bare
## Alg-Alg2      2    0      0
## NB_S          0    2      0
## Alg_Bare      0    0      1
```

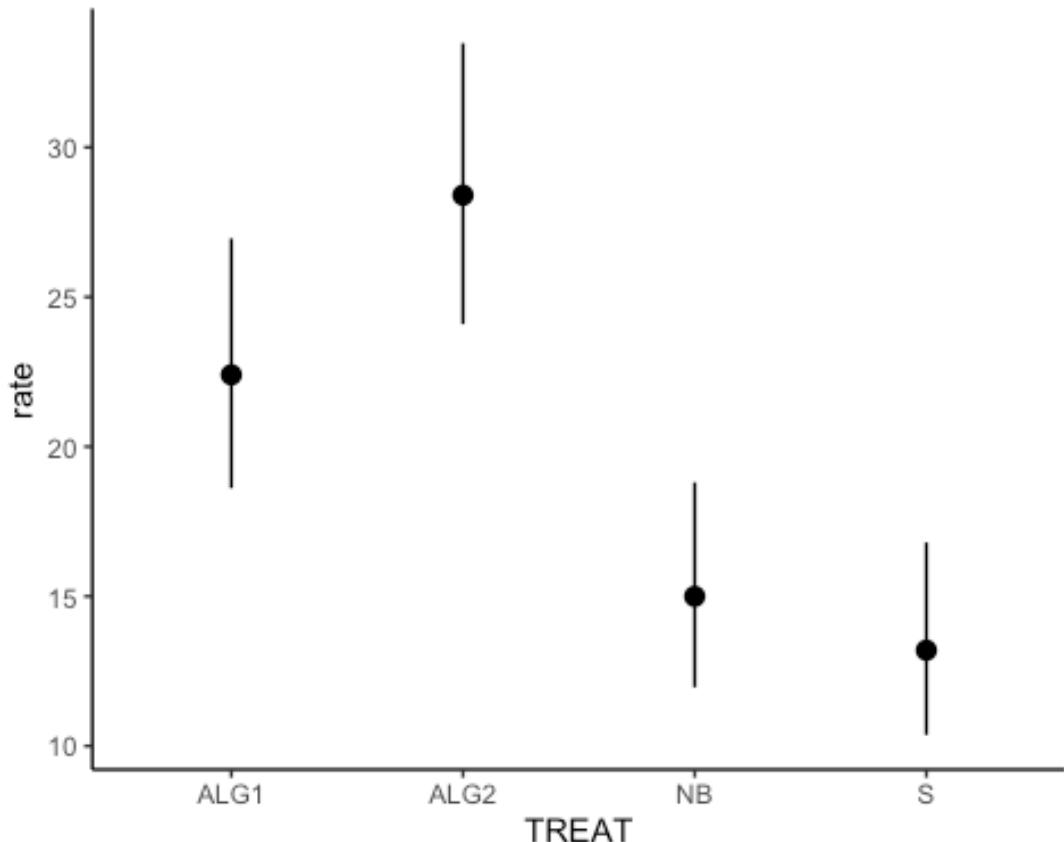
If there was non independence, we would get non-zeros. We want the triangles on each side to be zero (left-lower edge and upper-right edge, made up by 0s). In other words, we only want that there are numbers in the diagonal line. Murray said: We are cutting a cake, and we are defining where to cut. We don't want to cut where there is no cake (=where there is zeros). If we didn't that, we might accumulate Type 1 errors.

```
day.glm1 %>% emmeans(~TREAT, contr=list(TREAT=cmat), type="response")
## $emmeans
##   TREAT rate   SE df  asymp.LCL  asymp.UCL
##   ALG1  22.4 2.12 Inf   18.6     27.0
##   ALG2  28.4 2.38 Inf   24.1     33.5
##   NB    15.0 1.73 Inf   12.0     18.8
##   S     13.2 1.62 Inf   10.4     16.8
##
## Confidence level used: 0.95
## Intervals are back-transformed from the log scale
##
## $contrasts
##   contrast      ratio      SE  df z.ratio p.value
##   TREAT.Alg-Alg2 0.789 0.0997 Inf -1.878  0.0604
##   TREAT.NB_S     1.136 0.1918 Inf  0.757  0.4488
##   TREAT.Alg_Bare 1.792 0.1890 Inf  5.536 <.0001
##
## Tests are performed on the log scale
```

Constraints: - TREAT.Alg-Alg2 => Alg1 is 21% lower than Alg2 ($1 - 0.789 = 0.21$) => Alg1 = $0.789 * \text{Alg2}$ - TREAT.NB_S => NB is 13% higher than S => NB = $1.136 * S$ - TREAT.Alg_Bare => Alg1 is 79% higher than the combi of NB+S ("Bare substrate") => Alg1 = $1.792 * \text{Bare}$

Summary figures

```
newdata=emmeans(day.glm1,~TREAT,type="response") %>%  
  as.data.frame  
  
ggplot(newdata,aes(y=rate,x=TREAT)) +  
  geom_pointrange(aes(ymin=asymp.LCL,ymax=asymp.UCL)) +  
  theme_classic()
```



References

6_MLR_Poisson

Sara Kophamel

02/12/2020

Preparations

Load the necessary libraries

```
library(car)      #for regression diagnostics
library(broom)    #for tidy output
library(ggfortify) #for model diagnostics
library(sjPlot)   #for outputs
library(knitr)    #for kable
library(effects)  #for partial effects plots
library(emmeans)  #for estimating marginal means
library(ggeffects) #for plotting marginal means
library(MASS)     #for glm.nb
library(MuMIn)    #for AICc
library(tidyverse) #for data wrangling
library(modelr)   #for auxillary modelling functions
library(DHARMa)   #for residual diagnostics plots
library(performance) #for residuals diagnostics
library(see)       #for plotting residuals
library(patchwork) #grid of plots
library(scales)    #for more scales
```

Scenario

An ecologist studying a rocky shore at Phillip Island, in southeastern Australia, was interested in how clumps of intertidal mussels are maintained [@Quinn-1988-137]. In particular, he wanted to know how densities of adult mussels affected recruitment of young individuals from the plankton. As with most marine invertebrates, recruitment is highly patchy in time, so he expected to find seasonal variation, and the interaction between season and density - whether effects of adult mussel density vary across seasons - was the aspect of most interest.

The data were collected from four seasons, and with two densities of adult mussels. The experiment consisted of clumps of adult mussels attached to the rocks. These clumps were then brought back to the laboratory, and the number of baby mussels recorded. There were 3-6 replicate clumps for each density and season combination.

Format of quinn.csv data files

SEASON	DENSITY	RECRUITS	SQRTRECRUITS	GROUP
Spring	Low	15	3.87	SpringLow
..
Spring	High	11	3.32	SpringHigh
..
Summer	Low	21	4.58	SummerLow
..
Summer	High	34	5.83	SummerHigh
..
Autumn	Low	14	3.74	AutumnLow
..
SEASON	Categorical listing of Season in which mussel clumps were collected independent variable			
DENSITY	Categorical listing of the density of mussels within mussel clump independent variable			
RECRUITS	The number of mussel recruits response variable			
SQRTRECRUITS	Square root transformation of RECRUITS - needed to meet the test assumptions			
GROUPS	Categorical listing of Season/Density combinations - used for checking ANOVA assumptions			



Mussel

Read in the data

```
quinn = read_csv('data/quinn.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   SEASON = col_character(),
##   DENSITY = col_character(),
##   RECRUITS = col_double(),
##   SQRTRECRUITS = col_double(),
##   GROUP = col_character()
## )

glimpse(quinn)

## Rows: 42
## Columns: 5
## $ SEASON      <chr> "Spring", "Spring", "Spring", "Spring", "Spring",
## "Sprin...
## $ DENSITY     <chr> "Low", "Low", "Low", "Low", "Low", "High", "High",
## "High...
## $ RECRUITS    <dbl> 15, 10, 13, 13, 5, 11, 10, 15, 10, 13, 1, 21, 31, 21,
## 18...
## $ SQRTRECRUITS <dbl> 3.872983, 3.162278, 3.605551, 3.605551, 2.236068,
## 3.3166...
## $ GROUP       <chr> "SpringLow", "SpringLow", "SpringLow", "SpringLow",
## "Spr...

summary(quinn)
```

```

##      SEASON          DENSITY        RECRUITS      SQRTRECRUITS
##  Length:42          Length:42       Min.   : 0.00  Min.   :0.000
##  Class  :character  Class  :character  1st Qu.: 9.25  1st Qu.:3.041
##  Mode   :character  Mode   :character  Median  :13.50  Median  :3.674
##                                         Mean   :18.33  Mean   :3.871
##                                         3rd Qu.:21.75 3rd Qu.:4.663
##                                         Max.   :69.00  Max.   :8.307
##
##      GROUP
##  Length:42
##  Class  :character
##  Mode   :character
##
##
```

Since we intend to model both SEASON and DENSITY as categorical variables, we need to explicitly declare them as factors.

```

quinn = quinn %>% mutate(SEASON = factor(SEASON, levels=c('Spring', 'Summer',
'Autumn', 'Winter')),
                           DENSITY = factor(DENSITY))

```

This will affect which is the ref level, and the order by which they are produced on graphs. It won't affect the stats though

Exploratory data analysis

Model formula:

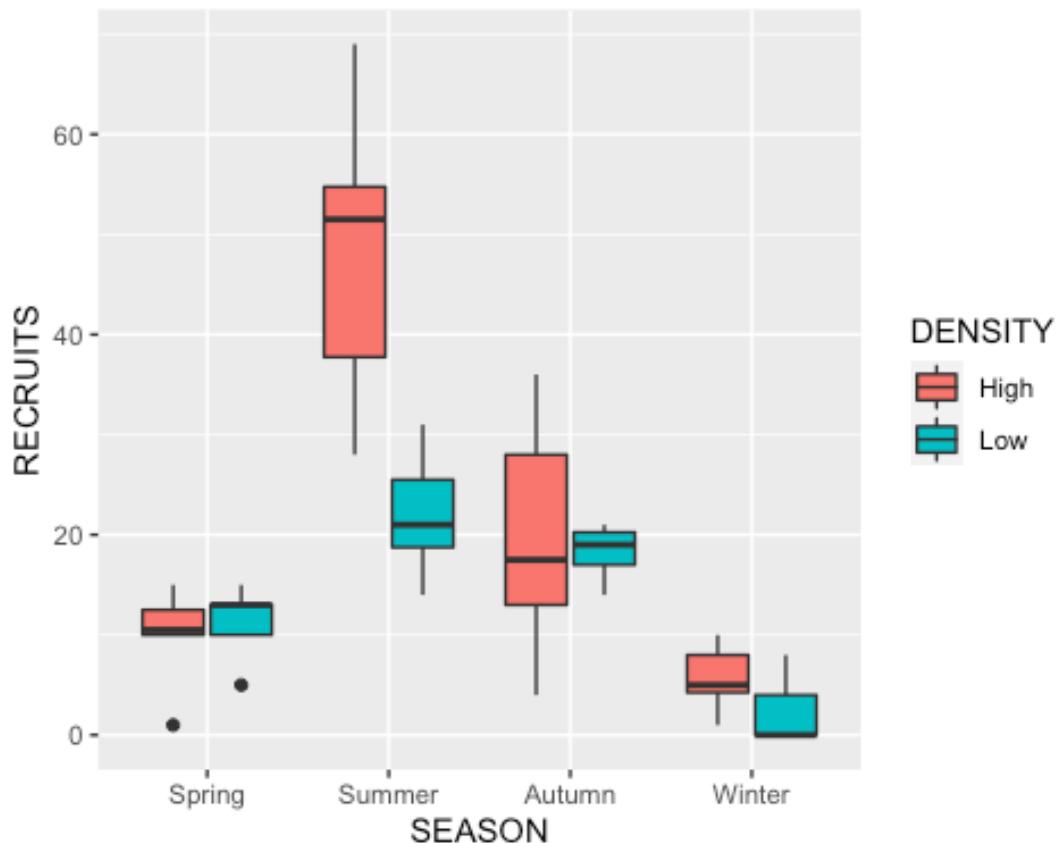
$$\$ \$ \begin{aligned} y_i &\sim \text{Pois}(\lambda_i) \\ \boldsymbol{\beta} \bf{X}_i &\end{aligned} \$ \$$$

where β is a vector of effects parameters and \bf{X} is a model matrix representing the intercept and effects of season, density and their interaction on mussel recruitment.

```

ggplot(quinn, aes(y=RECRUITS, x=SEASON, fill=DENSITY)) +
  geom_boxplot()

```



Conclusions:

- there is clear evidence of non-homogeneity of variance. The middle of the boxplot is the median, but if it was the mean, it would also be symmetrical. The variance is gonna be proportional to the size of the boxplots. So “High density boxplot in Summer” has a high mean, and high variance, and “High density boxplot in Winter” has a low mean, and a low variance.
- specifically, there is evidence that the variance is related to the mean in that boxplots that are lower on the y-axis (low mean) also have lower variance (shorter boxplots)
- this might be expected for count data and we might consider that a Poisson distribution (which assumes that mean and variance are equal - and thus related in a very specific way).

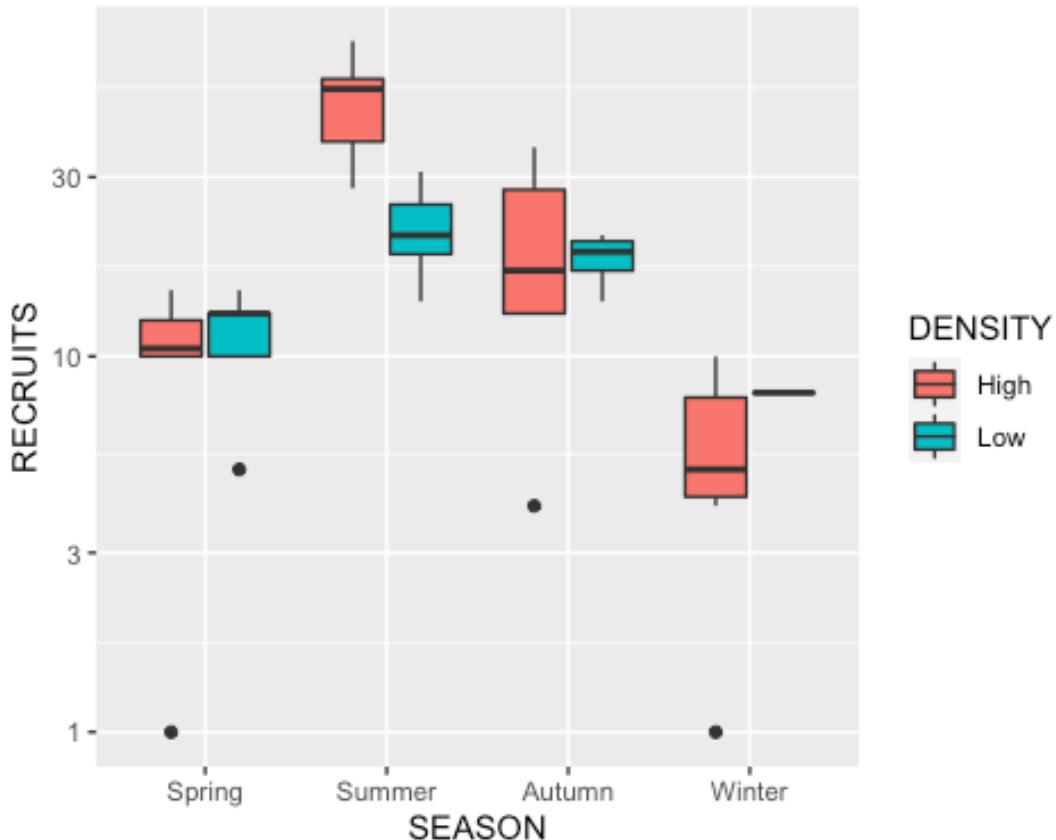
We expect the mean to be related to the variance in a Poisson distribution, which is the case. The authors of this study did a sq root transformation, which Murray does NOT recommend to do.

Lets mimic what the data is gonna be like from a Poisson, cause we know that a Poisson has a log link. We can thus mimic the effect of using a log link, by using log scaled y-axis.

```
ggplot(quinn, aes(y=RECRUITS, x=SEASON, fill=DENSITY)) +
  geom_boxplot() +
  scale_y_log10()
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

```
## Warning: Removed 2 rows containing non-finite values (stat_boxplot).
```



Conclusions: - The Winter low density group had quite a few zeros, so they have now been removed from the data from the perspective of this plot. - Otherwise, the log transformation is an improvement, whereas a Gaussian would not be useful unless we log-transformed the data. We decide to go for a Poisson distribution.

Fit the model

```
quinn.glm<-glm(RECRUITS~DENSITY*SEASON,data=quinn,
                  family=poisson(link="log"))
```

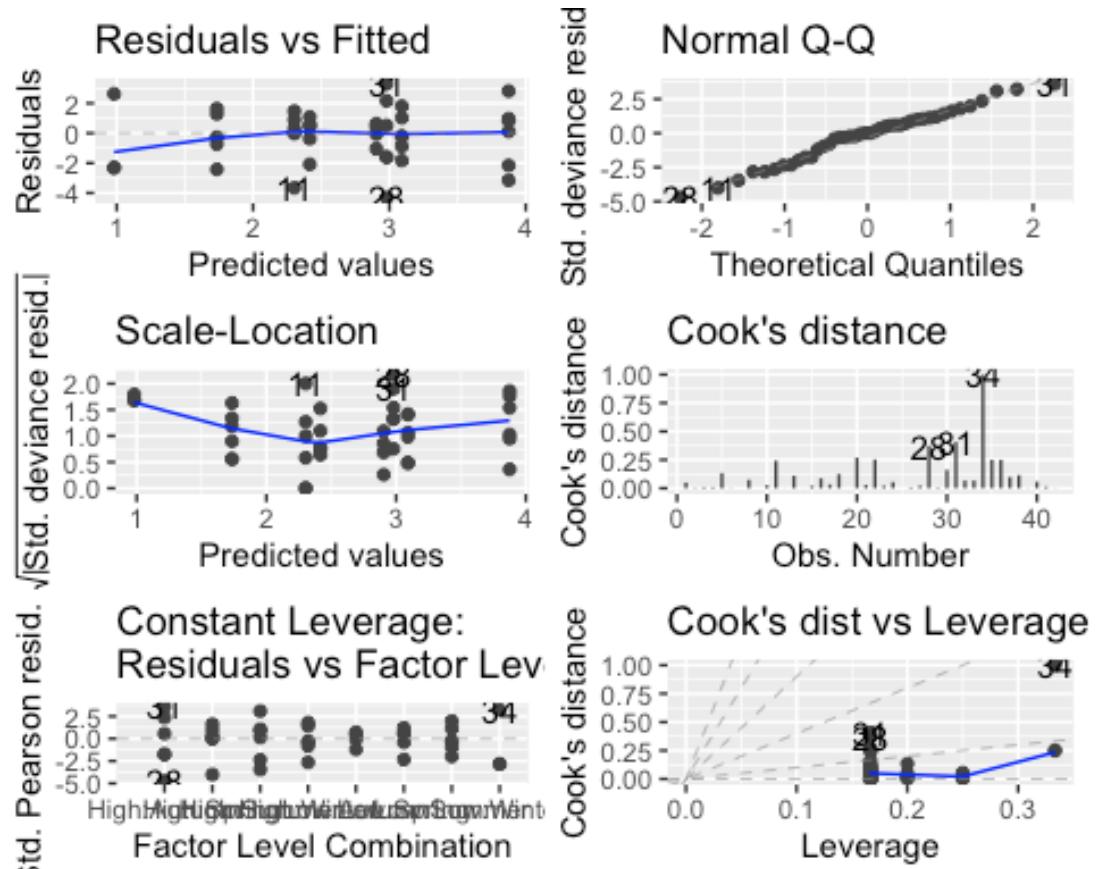
Why did Murray not centre the variables? Cause they are categorical, there is nothing to centre (eg what is the centre of season?) Also, NEVER centre a response variable (there are only few exceptions to do this)

Model validation

Option 1: autoplot

```
autoplot(quinn.glm,which=1:6)
```

```
## Warning: `arrange_()` is deprecated as of dplyr 0.7.0.
## Please use `arrange()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle:::last_warnings()` to see where this warning was generated.
```

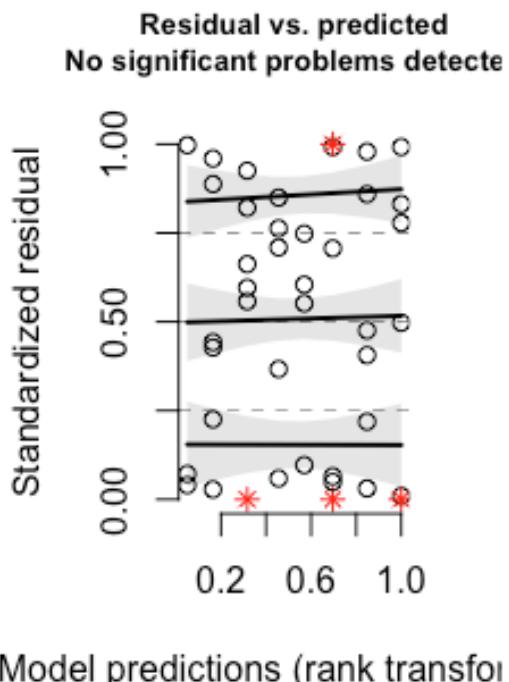
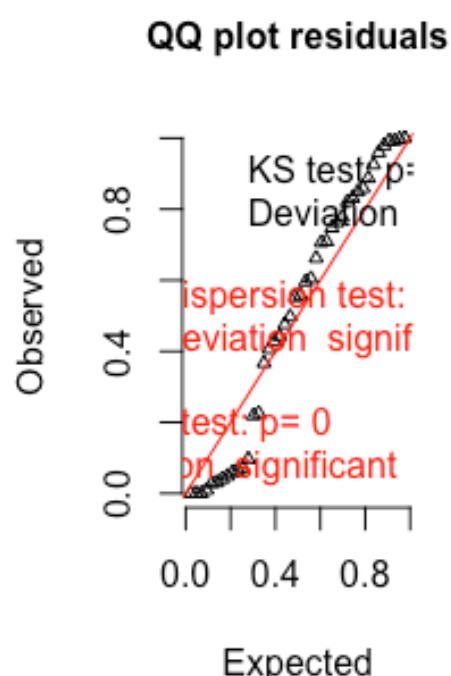


- If a data point had a Cook's distance >0.8, that observation might be an outlier. In this case, observation 34 is identified as an outlier - Otherwise, the remaining plots look ok - Leverage (two bottom plots) is in this case nonsense, as we got categorical variables

Option 2

```
quinn.resid=simulateResiduals(quinn.glm, plot=TRUE)
```

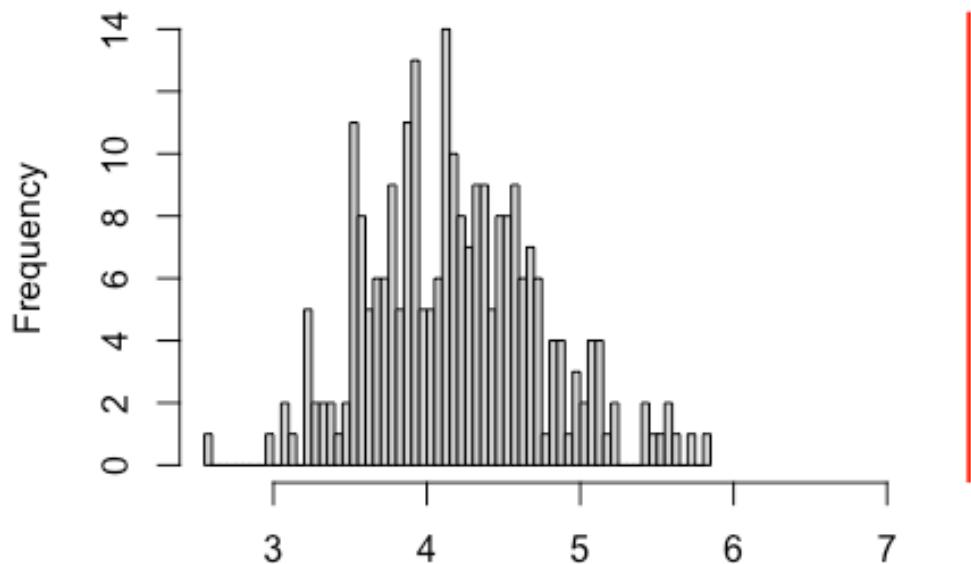
DHARMA residual diagnostics



We got a problem here. - Evidence of a dispersion problem - Outlier problems It would appear that our data are over- or underdispersed

```
testDispersion(quinn.resid)
```

DHARMA nonparametric dispersion test via sd of residuals fitted vs. simulated



Simulated values, red line = fitted model. p-value (two.sided) = 1

```
## 
## DHARMA nonparametric dispersion test via sd of residuals fitted vs.
## simulated
## 
## data: simulationOutput
## ratioObsSim = 1.797, p-value < 2.2e-16
## alternative hypothesis: two.sided
```

Overdispersion test

```
performance::check_overdispersion(quinn.glm)

## # Overdispersion test
## 
##      dispersion ratio = 3.309
##      Pearson's Chi-Squared = 112.497
##                  p-value = < 0.001

## Overdispersion detected.
```

The dispersion value is >1 . There is also another measure of overdispersion, which shows that we got 3.3x more variance than the Poisson would expect (if it is $>3x$, it definitely needs to be addressed)

Different model

Let's try using a different model, as we were having issues with Poisson. A negative binomial is a good distrib for aggregated data. It converges on a Poisson if conversion is = 1, but it does not ASSUME dispersion of 1. It actually measures the dispersion (which is good in this case, as we got over-dispersion with the Poisson).

We want a model that understands that we got a data generation (count process), and a data detection process occurring. A zero inflation model has those two models involved, the count process and the detection process.

Zero inflation testing

We got a couple of tests for zero inflation:

Option 1

```
performance::check_zeroinflation(quinn.glm)

## # Check for zero-inflation
##
##   Observed zeros: 2
##   Predicted zeros: 0
##             Ratio: 0.00

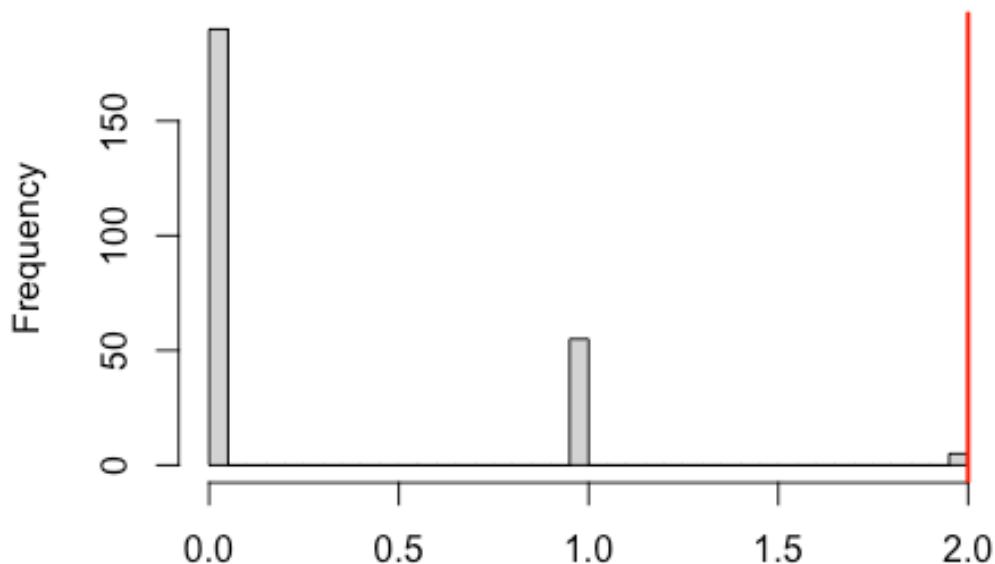
## Model is underfitting zeros (probable zero-inflation).
```

This test says that we are inflating zeros (we got 2 inflated zeros)

Option 2

```
testZeroInflation(quinn.resid)
```

DHARMA zero-inflation test via comparison to expected zeros with simulation under H0 = fitted model



Simulated values, red line = fitted model. p-value (two.sided) = 0.

```
##  
## DHARMA zero-inflation test via comparison to expected zeros with  
## simulation under H0 = fitted model  
##  
## data: simulationOutput  
## ratioObsSim = 7.6923, p-value = 0.04  
## alternative hypothesis: two.sided
```

This second test simulates Residuals. It says that there is actually not much to worry about. Murray says that if we got <30% of our data inflated with 0s, it's fine.

Neg binomial model fitting

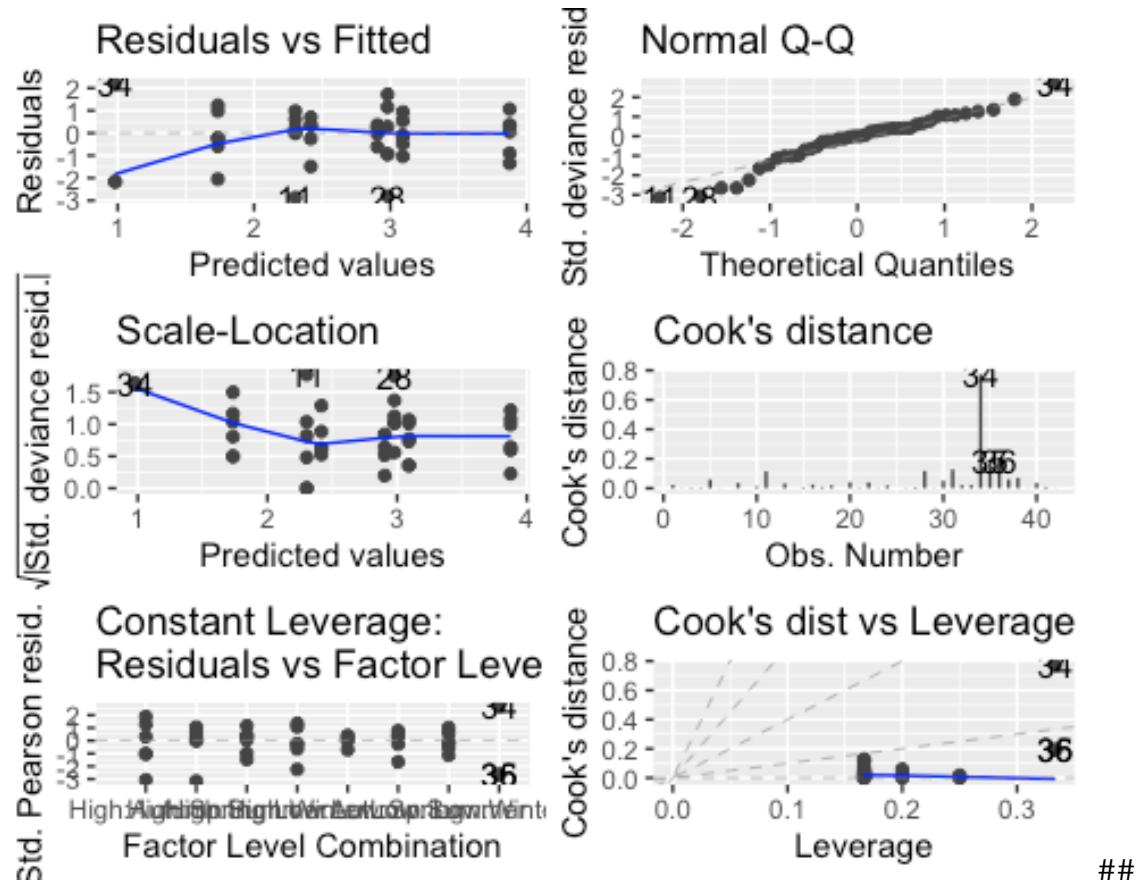
The neg binomial test is often not included in the common R packages. We will need a special function. And annoyingly enough, packages use diff names of writing binomials

```
quinn.nb=g1m.nb(RECRUITS~DENSITY*SEASON,data=quinn)
```

Partial plots

Option 1: autoplot

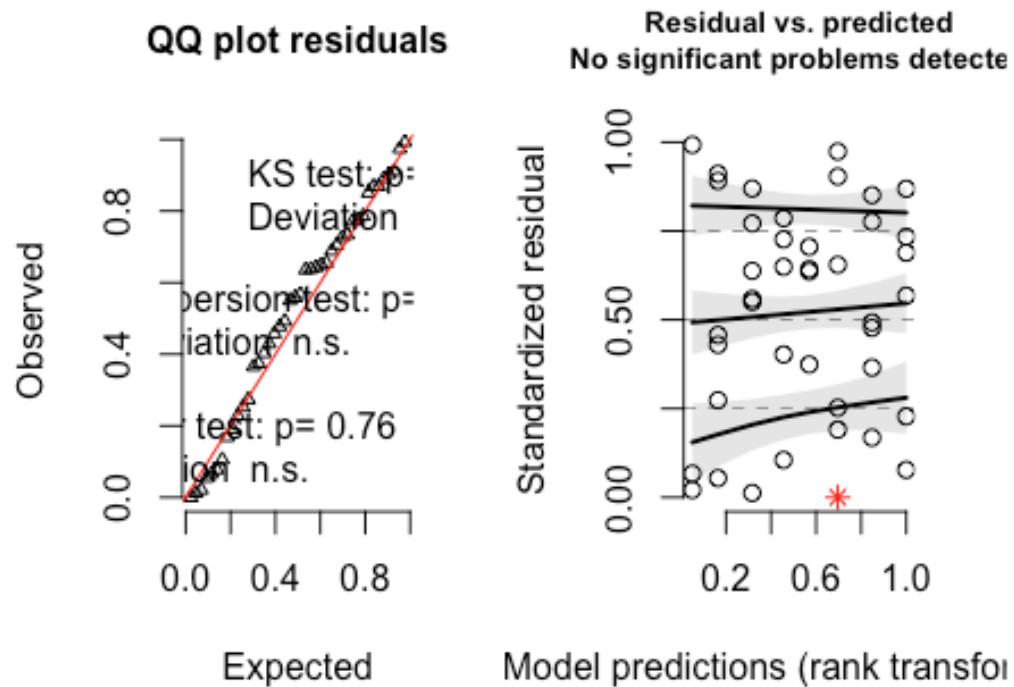
```
autoplot(quinn.nb,which=1:6)
```



Option 1: autoplot

```
quinn.resid=simulateResiduals(quinn.nb,plot=TRUE)
```

DHARMA residual diagnostics



In general, this model looks good. We no longer have overdispersion (and that one outlier is not a big issue).

BUT We have to be aware that just because we run a neg binomial does not mean that we addressed zero inflation. And because we apparently don't have zero inflation does not mean that we don't have overdispersion.

```
plot_model(quinn.nb, type="eff", terms=c("SEASON", "DENSITY"))
```



Model investigation / hypothesis testing

```
summary(quinn.nb)
```

```
##
## Call:
## glm.nb(formula = RECRUITS ~ DENSITY * SEASON, data = quinn, init.theta =
## 9.022467857,
##         link = log)
##
## Deviance Residuals:
##      Min        1Q     Median        3Q       Max
## -2.8704   -0.8274    0.0000    0.4999   2.1866
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 2.3026    0.1875 12.283 < 2e-16 ***
## DENSITYLow  0.1133    0.2742  0.413  0.67934
## SEASONSummer 1.5721    0.2389  6.581 4.69e-11 ***
## SEASONAutumn 0.6763    0.2492  2.714  0.00664 **
## SEASONGWinter -0.5680   0.2881 -1.971  0.04870 *
## DENSITYLow:SEASONSummer -0.8970   0.3509 -2.556  0.01059 *
## DENSITYLow:SEASONAutumn -0.1881   0.3788 -0.496  0.61955
```

```

## DENSITYLow:SEASONWinter -0.8671      0.5338 -1.624  0.10432
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(9.0225) family taken to be 1)
##
## Null deviance: 183.269  on 41  degrees of freedom
## Residual deviance: 54.883  on 34  degrees of freedom
## AIC: 293.09
##
## Number of Fisher Scoring iterations: 1
##
##
##          Theta:  9.02
##          Std. Err.: 3.69
##
## 2 x log-likelihood: -275.095

```

Let's have a closer look at the output: - Density is the adult density of the resident population, and whether it was high or low - Intercept represents the combination of Spring-High Density (it's the first red dot in the plot above). - If the intercept / our first level had the LEAST data, DON'T make it the ref group / the intercept. Because then all the other observations would link back to that group. Make the intercept the group with the MOST observations. Here, we re-arranged seasons for aesthetic reasons, but it also helps when comparing the groups against the intercept (=first season defined) - Densitylow Estimate: Diff btw low density and high density in Spring (Intercept) - Summer Estimate 1.6 -> Summer HIGH is 1.6x higher than Spring high - DENSITYLow:SEASONSummer - 0.8970 => In an additive model, we'd expect this to be 0 if the model had no interactions = We'd expect Summer Low to be at the same level as Summer high, but it is 0.89x times LOWER than summer high! And it is a sign diff

An ANOVA table would use a sums of squares (#type 1 or 2 sums of sq) and conclude that the diff are sign, although they aren't.

The magnitude of the effect depends on the season. In summer, it looks like density has got a big impact, but not in Spring or Winter. How diff the seasons are depends on the density. It looks like the seasons are quite diff for high density, and quite distinguished for low density.

Predictions

So we just interpreted this output on a log scale. Might be worth checking the raw data (exponentiate the transformation):

```

tidy(quinn.nb, conf.int=TRUE, exponentiate=TRUE)

## # A tibble: 8 x 7
##   term           estimate std.error statistic p.value conf.low

```

```

conf.high
##   <chr>          <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
<dbl>
## 1 (Intercept)      10.       0.187    12.3   1.11e-34   6.93
14.5
## 2 DENSITYLow       1.12     0.274    0.413  6.79e- 1   0.654
1.92
## 3 SEASONSummer     4.82     0.239    6.58   4.69e-11   3.02
7.72
## 4 SEASONAutumn     1.97     0.249    2.71   6.64e- 3   1.21
3.21
## 5 SEASONWinter     0.567    0.288   -1.97  4.87e- 2   0.320
0.993
## 6 DENSITYLow:SEASONSummer 0.408    0.351   -2.56  1.06e- 2   0.205
0.811
## 7 DENSITYLow:SEASONAutumn 0.829    0.379   -0.496 6.20e- 1   0.394
1.74
## 8 DENSITYLow:SEASONWinter 0.420     0.534   -1.62  1.04e- 1   0.142
1.17

```

SEASONSummer 4.8166667 => $4.82 - 1 * 100 =$ High Summer is 382% higher than High Spring (our intercept, which has a value of 10) => $4.81 \times 10 =$ High Summer value.

Pairwise comparisons

We are gonna look at the effect of Density now separate to each season.

```

quinn.nb %>% emmeans(pairwise~DENSITY|SEASON, type="response")

## $emmeans
## SEASON = Spring:
##  DENSITY response   SE  df asympt.LCL asympt.UCL
##  High      10.00 1.87 Inf    6.93    14.44
##  Low       11.20 2.24 Inf    7.57    16.58
##
## SEASON = Summer:
##  DENSITY response   SE  df asympt.LCL asympt.UCL
##  High      48.17 7.13 Inf   36.03    64.39
##  Low       22.00 3.55 Inf   16.03    30.19
##
## SEASON = Autumn:
##  DENSITY response   SE  df asympt.LCL asympt.UCL
##  High      19.67 3.23 Inf   14.26    27.13
##  Low       18.25 3.71 Inf   12.25    27.19
##
## SEASON = Winter:
##  DENSITY response   SE  df asympt.LCL asympt.UCL
##  High      5.67 1.24 Inf    3.69    8.70
##  Low       2.67 1.07 Inf    1.21    5.87
##

```

```

## Confidence level used: 0.95
## Intervals are back-transformed from the log scale
##
## $contrasts
## SEASON = Spring:
##   contrast ratio    SE df z.ratio p.value
##   High / Low 0.893 0.245 Inf -0.413  0.6793
##
## SEASON = Summer:
##   contrast ratio    SE df z.ratio p.value
##   High / Low 2.189 0.480 Inf  3.577  0.0003
##
## SEASON = Autumn:
##   contrast ratio    SE df z.ratio p.value
##   High / Low 1.078 0.282 Inf  0.286  0.7749
##
## SEASON = Winter:
##   contrast ratio    SE df z.ratio p.value
##   High / Low 2.125 0.973 Inf  1.646  0.0999
##
## Tests are performed on the log scale

```

Let's focus on the contrasts: Is there any evidence of density effect in Spring? NO ...And in Summer? YES. The density is nearly 120% higher btw low and high summer (low summer * 2.2 = high summer) ...In Autumn? No ...In Winter? No. The ration is nearly the same magnitude. There is just a bit more noise there. In absolute terms it's a small change, but in % terms, which is what we are working with, it's a big change.

Summary figures

```

newdata=emmeans(quinn.nb, ~DENSITY|SEASON,type="response") %>% as.data.frame

head(newdata)

##   DENSITY SEASON response      SE  df asympt.LCL asympt.UCL
## 1     High Spring 10.00000 1.874542 Inf  6.925302 14.43980
## 2     Low  Spring 11.20000 2.240673 Inf  7.567050 16.57713
## 3     High Summer 48.16667 7.133321 Inf 36.031846 64.38826
## 4     Low  Summer 22.00000 3.550677 Inf 16.034060 30.18574
## 5     High Autumn 19.66667 3.228389 Inf 14.256124 27.13064
## 6     Low Autumn 18.25000 3.713650 Inf 12.247680 27.19392

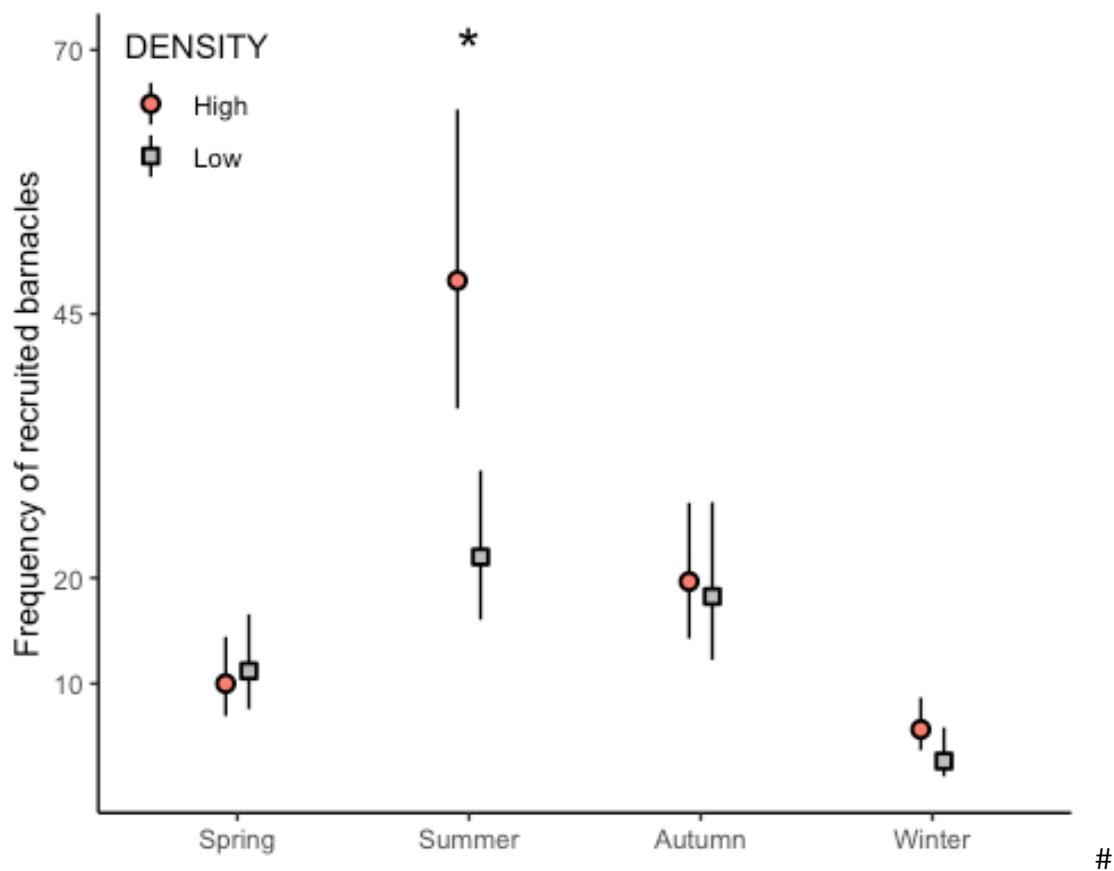
```

In this case, we don't need to create a grid, as we are working with pre-defined categorical variables. If we were working with numeric data, we would have to define where we want the predictions to be made. Here it is clear that we want ev Season category. As a rule of thumb, don't create grids with categorical data (useless most of the time, unless you want specific categories within those categories)

```

ggplot(newdata,aes(y=response, x=SEASON, fill=DENSITY))+ # fill defines the
  colors of the dots
  geom_pointrange(aes(ymax=asymp.UCL,ymin=asymp.LCL,shape=DENSITY),
    position=position_dodge(width=0.2))+ # separates the data
  points by 0.2
  scale_shape_manual(values=c(21,22))+ # sets the shapes of the data points
  (21=circle, 22=square)
  scale_fill_manual(breaks=c('High','Low'),
    values=c('salmon','grey')) + #defines the levels and
  colors (High-black, Low-grey)
  scale_y_continuous(name='Frequency of recruited barnacles', # name="" adds
  a y axis label
    breaks=c(10,20,45,70)) + # sets the breaks of the y axis
  (if we had a discrete scale, we would write scale_y_discrete )
  theme_classic()+
  theme(axis.title.x=element_blank(),
    legend.position=c(0.01,1),# refers to the x and to the y axis (1st
  and 2nd value)
    legend.justification=c(0,1))+ # justifies the box to the top left
  tick
  annotate(geom='text',x='Summer',y=70,label='*',size=7) # Don't "" the
  numbers! # The star symbol defines the text, which he then changes with
  size=7

```



Murray did not go through the following

References

7_GLMM_Gaussian

Sara Kophamel

02/12/2020

PRESENTATION:

file:///Users/sara/Documents/PhD/HELP/STATISTICS%20and%20GRAPHS/-%20R%20and%20STATS%20COURSES/2020%20Murray%20Logan%20R%20Course/Logan_Course/docs/pres11.html#1

THEORY

Assumptions

- All obs are from the same variance
- The off diagonals of obs 1 and 2 are independent, there is no covariance btw them

To maximise power, you can: - increase sample size or replication - add another variable that might explain variability, reducing noise (eg. adding body mass to the model) - block (control the conditions)

(Slide 6) We could for example add sub-replicates to increase the power (eg. instead of sampling QLD and NSW, I sample TSV, Cairns, BNE, and Sydney, BlueMountains, and BondiBeach). This is called a nested design. If we include these extralevels in our models, we have pseudoreplicated it. But we also add an extra model. These observations are NOT independent from one another (Cairns, TSV and BNE are part of QLD; and Sydney, BlueMountains, and BondiBeach are part of NSW). Location for example might have a strong effect on the response variable. To increase power, we could litter the entire place full of these sampling units, or we could be more clever and put each treatment combination in each plot. In that way, they all become their own controls (Slide 7).

The idea is to create a nested and blocked design to reduce the unexplained variability by controlling overspace and by making the observations more homogeneous. But by doing so, our model is not independent anymore, and has to account for both the blocking and lack of independence.

The block itself is a random effect (this is not the case in Bayesian though, they don't use fixed and random effects). Your ind replicates are random effects. They are randomly located. You are not interested in that specific location, it is just a unit that we are using for sampling.

A blocking design is where the fixed effects occur within the blocks. Each level of the fixed effect occurs within each block. In the hierarchy, the highest level is the block, and then you got a factor, the fixed effects (slide 9, randomized block design).

The fixed effects are the levels you have measured (eg. seasons). They are the levels we are interested in concluding about. Random effects however, were not sampled to explain a response variable. We want to extrapolate from the random effects to serve as a model for our data (eg. animal ID is just a model for the experiment, and can be extrapolated to other animals). On the other hand, we don't want to extrapolate for our fixed effects (eg we want to know the specific effect of each season, without extrapolating to the other seasons).

In a nested design instead, the treatments occur at a higher level. The fixed effects occur above the level of the random effect.

If you combine the Nested AND Block design, you get a Split-Block design (the terms come from agricultural aggregations, as they were literally blocks of land).

Slide 14: Illustrates random effects. If we account for the blocks, the uncertainty has been dramatically reduced.

Slide 15: Illustrates random effects. We don't have one intercept, but a bunch of them. A random effects model is a random intercepts model, where each block is allowed to have its own intercept. And of that, what we actually record is the slope for each block, and the variance in the intercepts. This is another important property that we will collect, as it tells you how varied the blocks are. If the intercept is diff from each block, the slopes can also be diff from each block.

Having fit several intercepts and/or slopes, it will then calc the avg intercept and avg slope, and that is what R will present us with (slide 15). But as you can imagine, if we were to put purely parallel lines here, not allowing the slope to be diff for each group, the noise will be slightly more. Random intercepts will cut down on the noise the most, and therefore require more data to be able to fit.

To summarise, the mixed effects models are made for increasing power without increasing resources by making the units that I am sampling from a more homogenous situation. It is almost as if we had done lab experiments. Internally, it is equivalent to allowing diff intercepts, and slopes for each group, as well as accounting for the lack of independence.

We are trying to integrate this slope over all possible blocks, not just over the one we measured. We want to extrapolate beyond the actual quadrates we measured. That is a maximum likelihood process, and cannot be done by ordinary least squares regression. In this case, it is always better using a residual maximum likelihood (REML) instead of the standard max likelihood in the presence of random effects. -> It maximises the likelihood of the residuals, instead of the likelihood of the data (I think he said that)

PRACTISE

Load the example: glmm_example1

Packages

```
library(lme4)      #for Lmer
```

```

## Loading required package: Matrix

library(lmerTest) #for satterthwaite p-values with lmer

##
## Attaching package: 'lmerTest'

## The following object is masked from 'package:lme4':
## 
##     lmer

## The following object is masked from 'package:stats':
## 
##     step

library(glmmTMB) # for fitting the random intercept model
library(car)      #for regression diagnostics

## Loading required package: carData

## Registered S3 methods overwritten by 'car':
##   method           from
##   influence.merMod lme4
##   cooks.distance.influence.merMod lme4
##   dfbeta.influence.merMod    lme4
##   dfbetas.influence.merMod   lme4

library(broom)     #for tidy output
library(broom.mixed)

## Registered S3 method overwritten by 'broom.mixed':
##   method   from
##   tidy.gamlss broom

library(ggfortify) #for model diagnostics

## Loading required package: ggplot2

library(sjPlot)    #for outputs
library(knitr)     #for kable
library(effects)   #for partial effects plots

## lattice theme set by effectsTheme()
## See ?effectsTheme for details.

library(emmeans)   #for estimating marginal means
library(ggeffects) #for partial effects plots
library(MASS)       #for glm.nb
library(MuMin)     #for AICc
library(nlme)

##
## Attaching package: 'nlme'

```

```

## The following object is masked from 'package:lme4':
##
##     lmList

library(lme4)      #for Lmer
library(lmerTest)   #for satterthwaite p-values with Lmer
library(performance) #for residuals diagnostics
library(see)        #for plotting residuals
#library(pbkrttest) #for kenward-roger p-values with Lmer
library(glmmTMB)    #for glmmTMB
library(DHARMa)     #for residuals and diagnostics

## This is DHARMa 0.3.3.0. For overview type '?DHARMa'. For recent changes,
type news(package = 'DHARMa') Note: Syntax of plotResiduals has changed in
0.3.0, see ?plotResiduals for details

library(tidyverse) #for data wrangling
## — Attaching packages
----- tidyverse 1.3.0 —

## ✓ tibble  3.0.3      ✓ dplyr   1.0.2
## ✓ tidyr   1.1.2      ✓ stringr 1.4.0
## ✓ readr   1.3.1      ✓ forcats 0.5.0
## ✓ purrr   0.3.4

## — Conflicts
----- tidyverse_conflicts() —

## x dplyr::collapse() masks nlme::collapse()
## 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 dplyr::recode()  masks car::recode()
## x dplyr::select()  masks MASS::select()
## x purrr::some()    masks car::some()
## x tidyr::unpack()  masks Matrix::unpack()

# To install a package from the source -->
install.packages('glmmTMB', type='source')
# To unload/detach a package --> detach(package:glmmTMB)
# To deinstall a package --> remove.packages('glmmTMB')

```

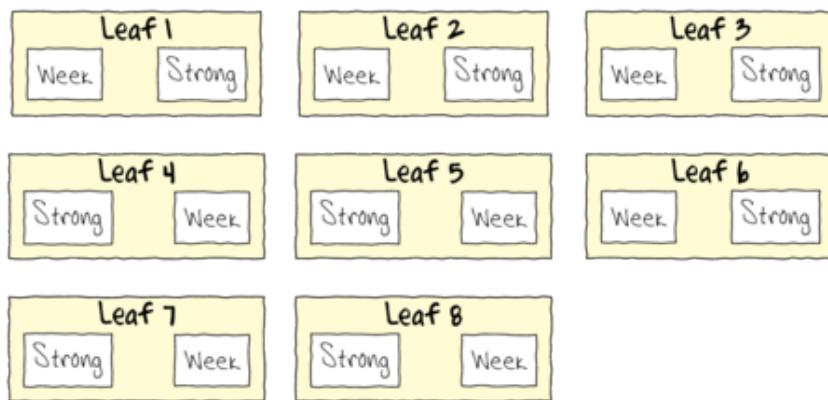
Scenario

A plant pathologist wanted to examine the effects of two different strengths of tobacco virus on the number of lesions on tobacco leaves. She knew from pilot studies that leaves

were inherently very variable in response to the virus. In an attempt to account for this leaf to leaf variability, both treatments were applied to each leaf. Eight individual leaves were divided in half, with half of each leaf inoculated with weak strength virus and the other half inoculated with strong virus. So the leaves were blocks and each treatment was represented once in each block. A completely randomised design would have had 16 leaves, with 8 whole leaves randomly allocated to each treatment.



Tobacco plant



The leaves are the blocks, and we have applied each treatment within each leaf. That means that each leaf can act as its own control. We need to make sure that we alternate the treatment on each side of the leaf (sometimes R, sometimes L...)

If possible, you wanna randomise your treatments within the box. Otherwise, you need to consider autocorrelation (sorry, this explanation was quite a brainfuck; not that clear)

$x \ 0 \ 0 \ 0 \ x \ x \ 0 \ x \ 0 \ 0 \rightarrow p \ x \rightarrow p^1 \ x$
 $0 \ 0 \ x \ 0 \ p \ p \ x \ p^2 \ p^2 \ x \ 0 \ 0 \ 0 \ x \ p \ p \ x \ p^3 \ p^3 \ x$ variance-covar mixed effects model
 first order autocorrelation structure (constant degree of correlation) (reflected in
 summary(lmer)output: Correlation of fixed effects: -0.507)

The p's are gonna be constant if we randomise the order. The objects closer in time will be more similar than those further apart. If p=0.5, the ps further apart will be higher. We will use the autocor str if we can't randomise our trial.

Format of tobacco.csv data files

LEAF	TREAT	NUMBER
1	Strong	35.898
1	Weak	25.02
2	Strong	34.118
2	Weak	23.167
3	Strong	35.702
3	Weak	24.122
...
LEAF	The blocking factor - Factor B	
TREAT	Categorical representation of the strength of the tobacco virus - main factor of interest Factor A	
NUMBER	Number of lesions on that part of the tobacco leaf - response variable	

Read in the data

```
tobacco = read_csv('data/tobacco.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   LEAF = col_character(),
##   TREATMENT = col_character(),
##   NUMBER = col_double()
## )

glimpse(tobacco)

## Rows: 16
## Columns: 3
## $ LEAF      <chr> "L1", "L1", "L2", "L2", "L3", "L3", "L4", "L4", "L5",
## "L5",...
## $ TREATMENT <chr> "Strong", "Weak", "Strong", "Weak", "Strong", "Weak",
## "Stro...
## $ NUMBER    <dbl> 35.89776, 25.01984, 34.11786, 23.16740, 35.70215,
## 24.12191,...
```

Our random effects are the leaves, so we will convert it into factors. We will also convert treatment, our main fixed effect, into factors.

Our response (number of lesions) has already been aggregated up, which is not ideal. We will create a model assuming a Gaussian distribution.

Exploratory data analysis

Model formula:

$$\$\$ y_i \sim N(\mu_i, \sigma^2) \\ \mu_i = \boldsymbol{\beta} X_i + \boldsymbol{\gamma} Z_i \$\$$$

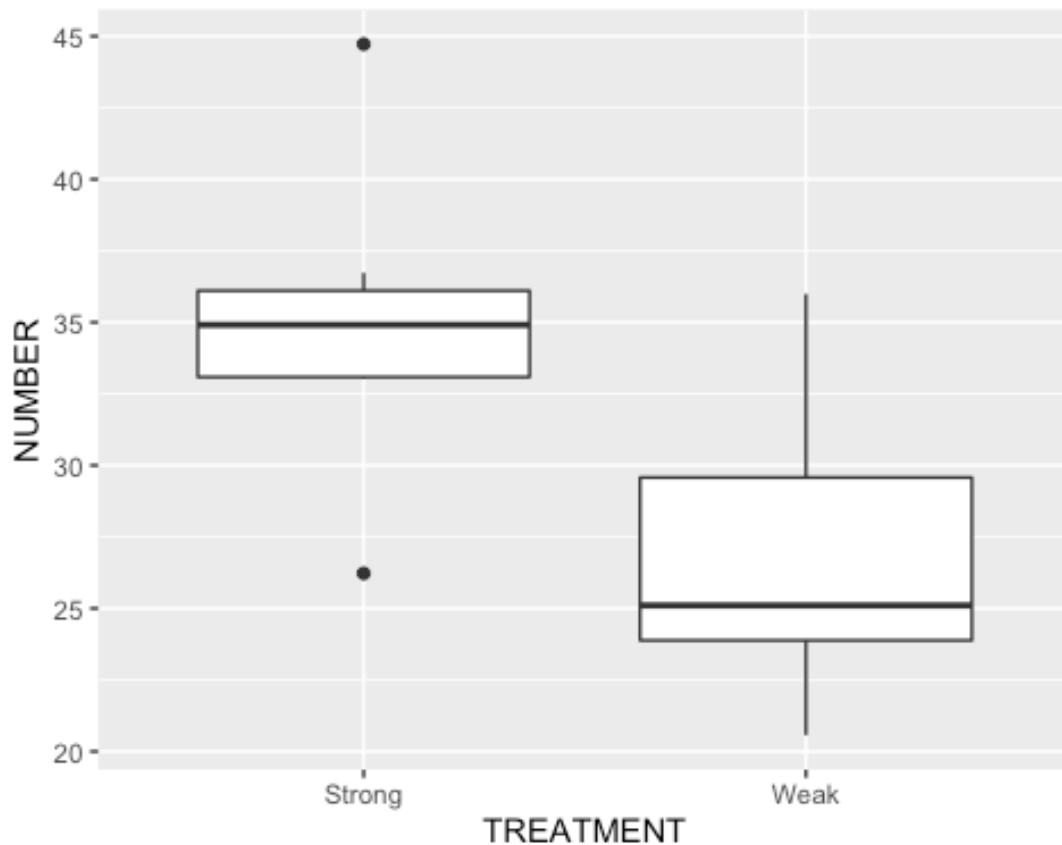
where β and γ are vectors of the fixed and random effects parameters respectively and X is the model matrix representing the overall intercept and effects of the treatment on the number of lesions. Z represents a cell means model matrix for the random intercepts associated with leaves.

We will start by explicitly declaring the categorical variable (TREATMENT) as a factor. In addition, random effects (in this case LEAF) should also be declared as factors.

```
tobacco = tobacco %>%
  mutate(LEAF=factor(LEAF),
        TREATMENT=factor(TREATMENT))
```

To explore the assumptions of homogeneity of variance and normality, a boxplot of each Treatment level is appropriate.

```
ggplot(tobacco, aes(y=NUMBER, x=TREATMENT)) +
  geom_boxplot()
```



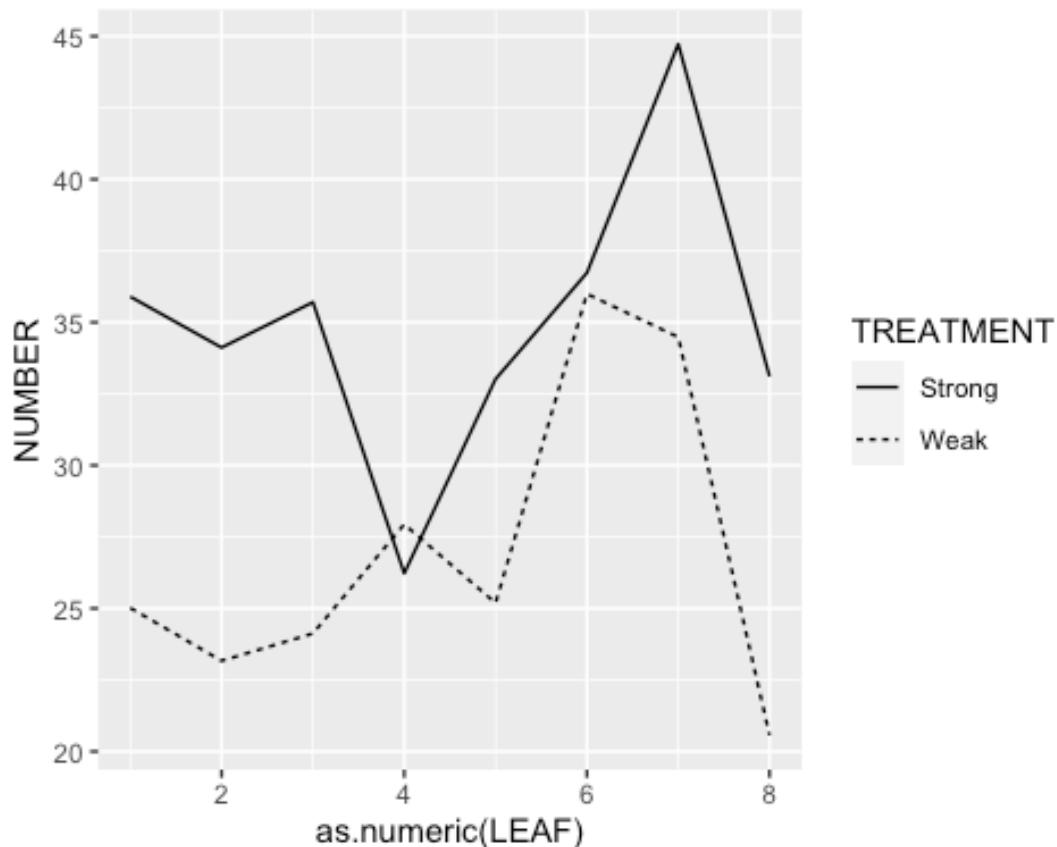
We are looking for gross non-normality and non-equal variance, which does not seem to be the case.

Conclusions:

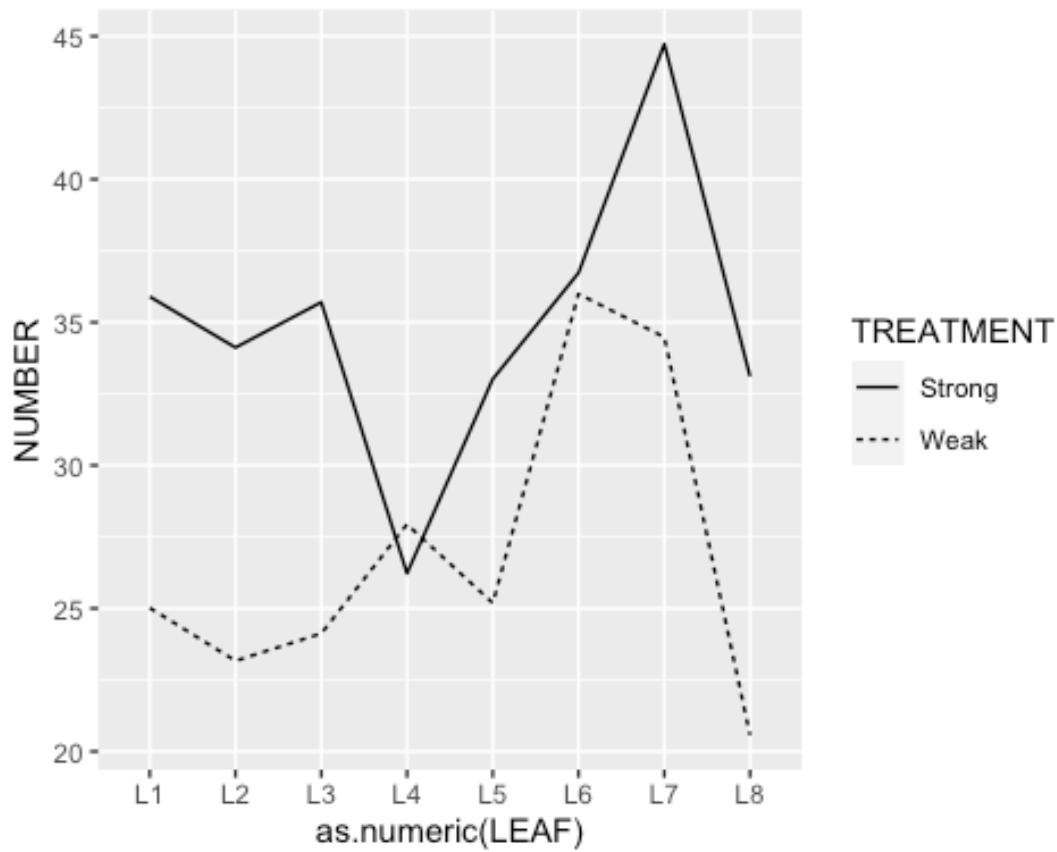
- both normality and homogeneity of variance seem satisfied

It can also be useful to get a sense of the consistency across blocks (LEAF). That is, do all Leaves have a similar baseline level of lesion susceptibility and do they respond similarly to the treatment.

```
ggplot(tobacco, aes(y=NUMBER, x=as.numeric(LEAF))) +  
  geom_line(aes(linetype=TREATMENT))
```



```
## If we want to retain the original LEAF Labels  
ggplot(tobacco, aes(y=NUMBER, x=as.numeric(LEAF))) +  
  geom_blank(aes(x=LEAF)) +  
  geom_line(aes(linetype=TREATMENT))
```

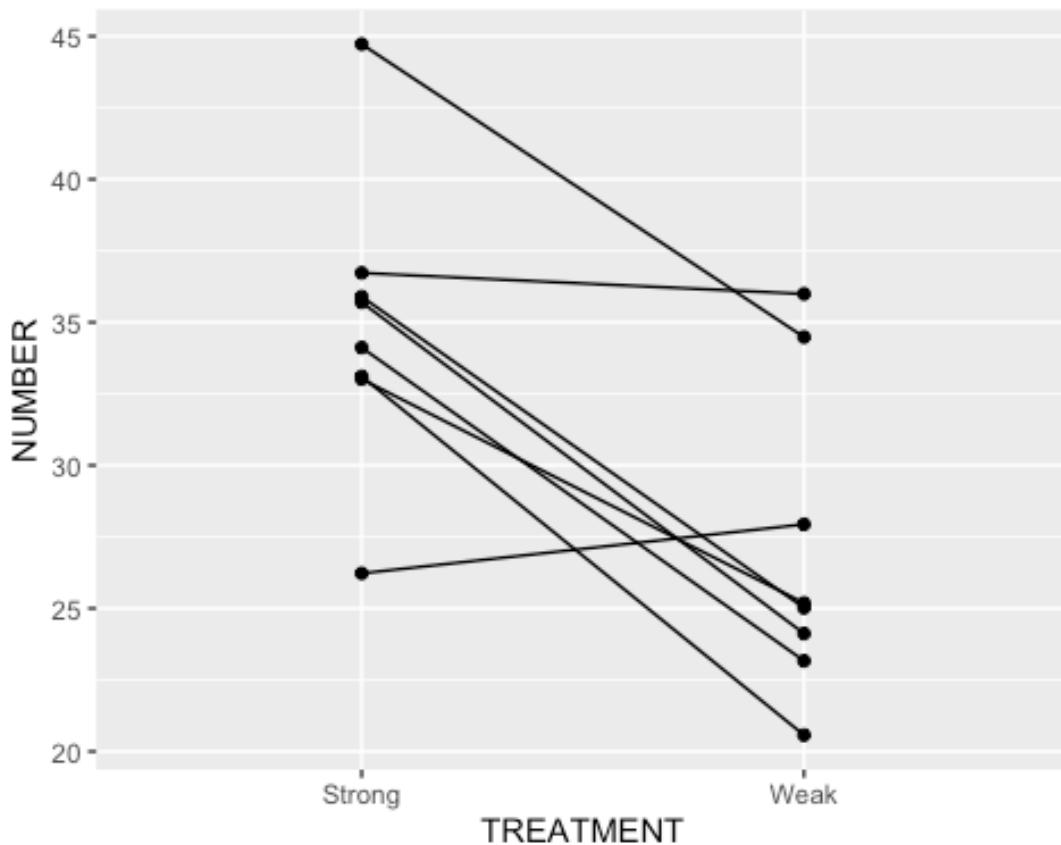


Conclusions:

- it is clear that some leaves are more susceptible to lesions (e.g. Leaf 7) than other leaves (e.g. Leaf 4)
- most leaves (other than Leaf 4 and 6) have a similar response to the Treatments - that is most have higher number of lesions from the Strong Treatment than the Weak Treatment.

Given that there are only two levels of Treatment (Strong and Weak), it might be easier to visualise the differences in baselines and effect consistency by plotting as:

```
ggplot(tobacco, aes(y=NUMBER, x=TREATMENT, group=LEAF)) +
  geom_point() +
  geom_line(aes(x=as.numeric(TREATMENT)))
```



Conclusions:

- this figure reiterates the points made earlier about the varying baselines and effect consistency.

The above figure also serves as a good way to visualise certain aspects of mixed effects models. When we fit a mixed effects model that includes a random blocking effect (in this case LEAF), we are indicating that we are allowing there to be a different intercept for each block (LEAF). In the current case, the intercept will represent the first Treatment level (Strong). So the random effect is specifying that the intercept can vary from Leaf to Leaf.

Most leaves seem to have the same pattern. This figure helps us decide about the kind of model we want to fit. A mixed effects model is essentially a random effects model. A special type of mixed effects model is a random intercept slope model. This is also worth considering, however it is a much more complicated model which (I think) assumes that the lines are parallel (so we would have to take some out??)

We can think of the model as having two tiers (a hierarchy), where the tiers of the hierarchy represent progressively smaller (typically) spatial scales. In the current example, the largest spatial units are the leaves (blocking factor). Within the leaves, there are the two Treatments (Strong and Weak) and within the Treatments are the individual observations.

We tend to represent this hierarchy upside down in the model formula:

$$y_i \sim N(\mu_i, \sigma^2) \quad \mu_i = \beta_0 + \boldsymbol{\beta} X_i \quad \beta_0 = \boldsymbol{\gamma} Z_i$$

In addition to allowing there to be a different intercept per leaf, we could allow there to be a different magnitude of effect (difference between Strong and Week Treatment) per leaf. That is considered a random slope. From the figure above, there is some evidence that the effects (slopes) may vary from Leaf to Leaf.

Incorporating a random slope (in addition to a random intercept), may reduce the amount of unexplained variance and thus improve the power of the main effect (Treatment effect).

Fit the model

Fitting the random intercept model within lme4

```
tobacco.lmer<-lmer(NUMBER~TREATMENT+(1|LEAF), data=tobacco, REML=TRUE)
# NUMBER is the response variable
# TREATMENT is our fixed effect
# The randomised effect needs to be written Like (1/effect) --> The 1 stands
for the intercept (you could have also written 1+TREATMENT). In this case,
the intercept varies with each Leaf, so we have to specify that with the "1|"
```

These models are residual maximum likelihood models (REML). Rather than trying to maximise the likelihood of the data, it does it for the residuals. In the presence of random effects, max likelihood produces biased estimates for the random effects. To get unbiased effects, we use residual max likelihood. If you Don't have any effects, max likelihood and residual max likelihood are the same. But when you are actually estimating random effects and want them to be unbiased, we need to use REML. Murray thinks that REML is the default for lmer, but it does not hurt writing it down.

To fit a random intercept random slope model, we write the following:

The slope and the intercept vary according to the leaf. The thing is that R can't run this model because we haven't got enough observations. The model needs to estimate too many things, and can't do it with so very few samples.

There is a modification that requires less data

It's got two || instead of one. The diff btw lmer1 and lmer2 is that our regular random intercept slope model (lmer1) will calculate the degree of correlation btw the intercepts and the slopes. As the slope lessens, the intercept gets lower. But this is not the case of lmer2.

lmer2: random intercept uncorrelated slope model -> We tell R: "The correlation btw intercept and slope is =0, so don't estimate it". It is a slightly simpler model, but it is dangerous saying that. Cause when the slope changes, the intercept usually changes too! If that was not the case, the slope would be pivoting at a fixed point on the y axis. This model

can be used when the other models won't work. It is not incorrect using it though, but it will create more noise than the other models. It does not work with categorical data. Murray tried fitting the lmer2 because we did not have enough data to fit the lmer1.

Random intercept random slope model: glmmTMB provides an alternative lmer, but is more powerful than lmer:

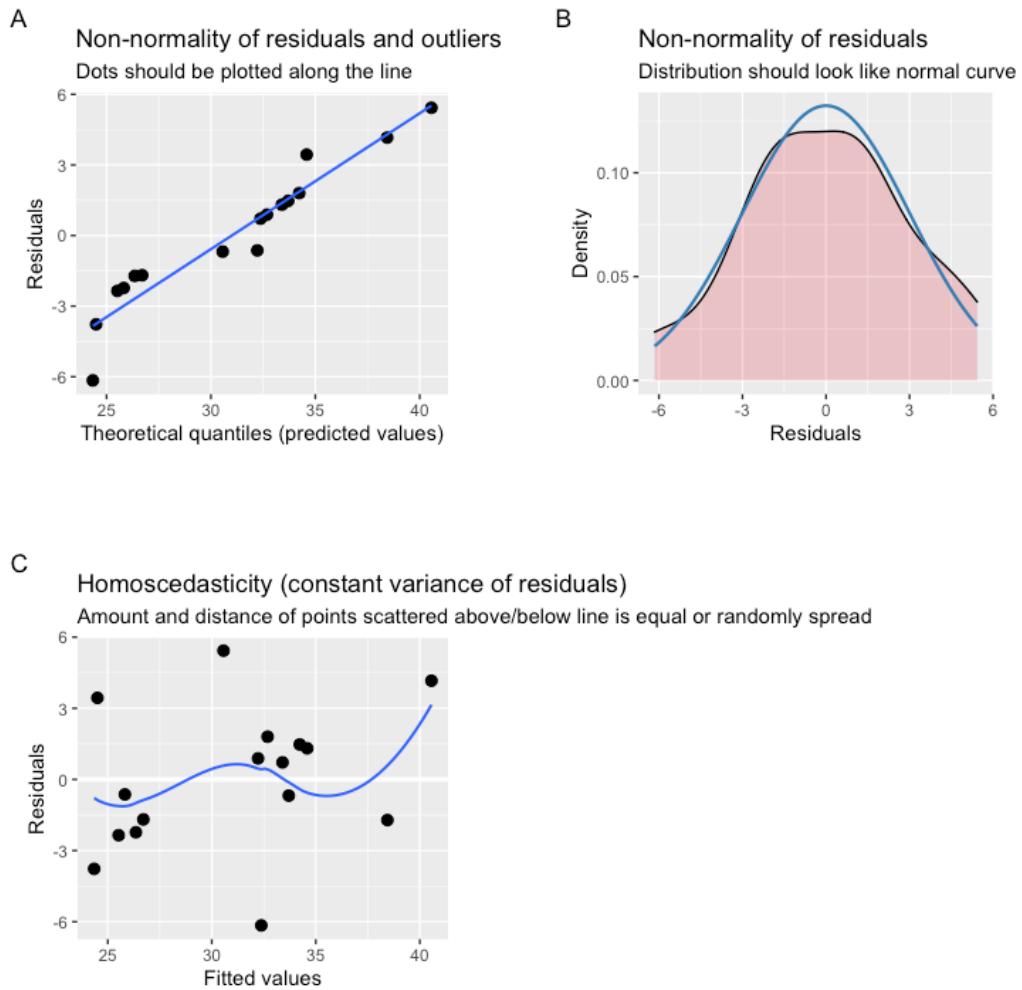
Obstructs way the vast diff btw lmer and glmm. But from a user perspective, they are very similar (and that was intentional). The syntax is exactly the same for both. You normally choose one or the other. Murray prefers using glmmTMB. It might create bugs when using it together with lmer though, so load either one or the other package.

Model validation

```
plot_grid(plot_model(tobacco.lmer, type = "diag")[-2]) # Diagnostic plot

## Warning in plot_grid(plot_model(tobacco.lmer, type = "diag")[-2]): Not
enough
## tags labels in list. Using letters instead.

## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
```



all good.

Looks

```
tobacco.resid=simulateResiduals(tobacco.lmer,plot=TRUE)

## Warning in checkModel(fittedModel): DHARMA: fittedModel not in class of
## supported models. Absolutely no guarantee that this will work!

## Warning in smooth.construct.tp.smooth.spec(object, dk$data, dk$knots):
basis dimension, k, increased to minimum possible

## Unable to calculate quantile regression for quantile 0.25. Possibly to few
(unique) data points / predictions. Will be omitted in plots and significance
calculations.

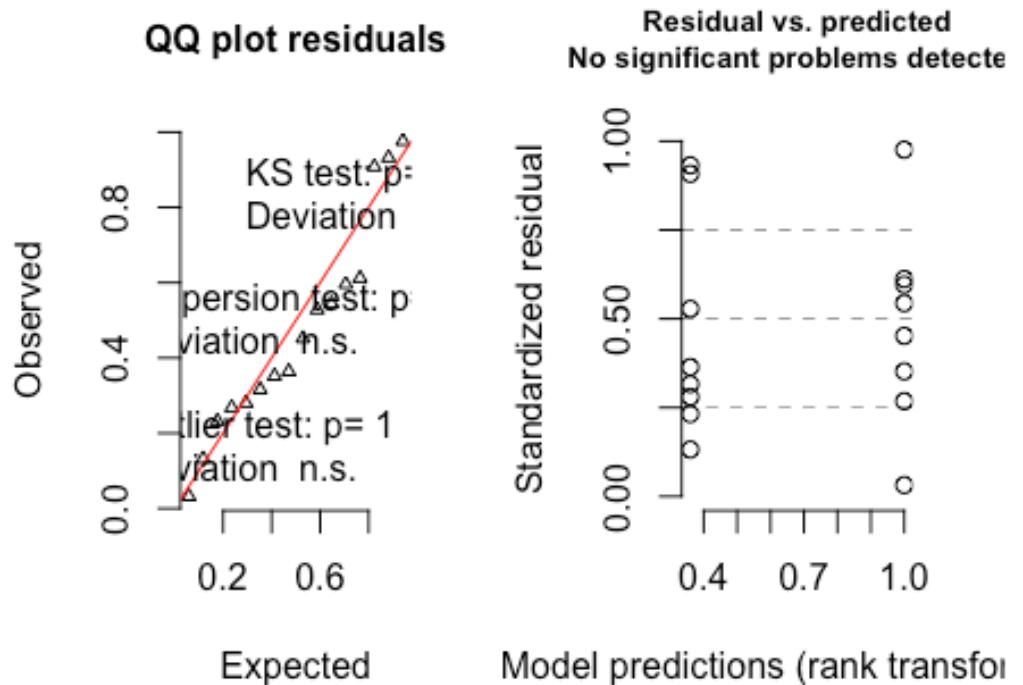
## Warning in smooth.construct.tp.smooth.spec(object, dk$data, dk$knots):
basis dimension, k, increased to minimum possible
```

```
## Unable to calculate quantile regression for quantile 0.5. Possibly to few
(unique) data points / predictions. Will be omitted in plots and significance
calculations.

## Warning in smooth.construct.tp.smooth.spec(object, dk$data, dk$knots):
basis dimension, k, increased to minimum possible

## Unable to calculate quantile regression for quantile 0.75. Possibly to few
(unique) data points / predictions. Will be omitted in plots and significance
calculations.
```

DHARMA residual diagnostics

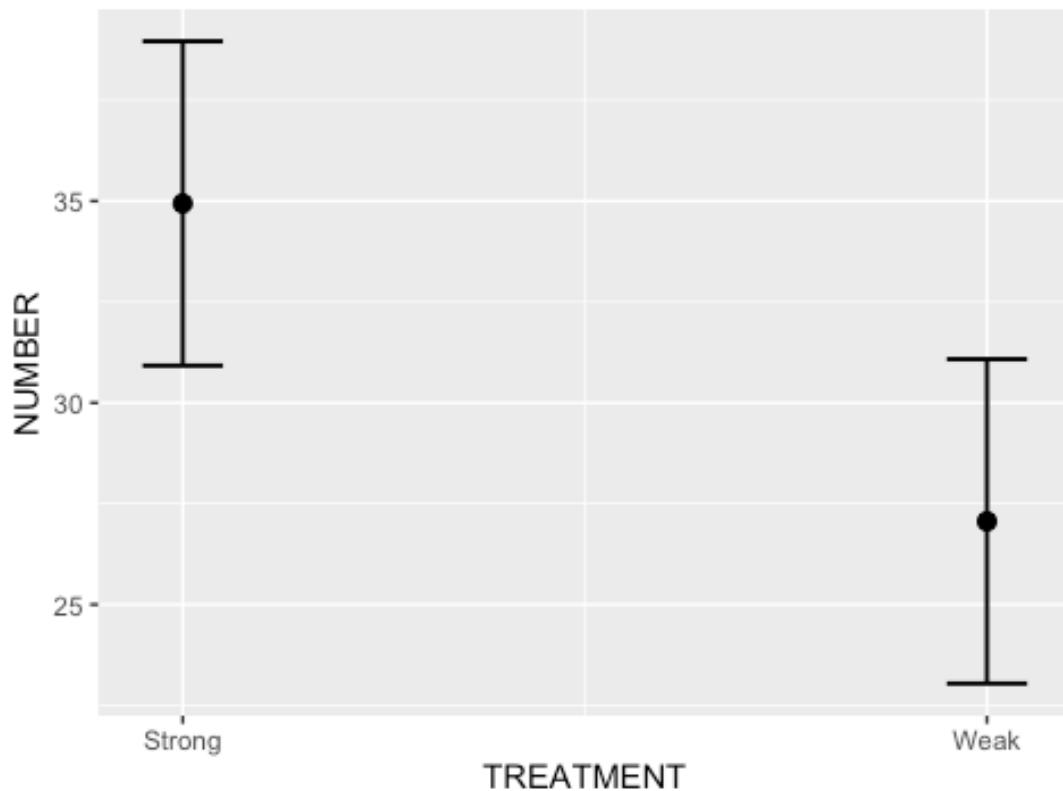


The DHARMA residuals have revealed no additional issues, so we can be confident that our outputs from the models will be reliable. We have satisfied the model assumptions.

Partial plots

```
plot_model(tobacco.lmer, type="eff") # Effect plot
## $TREATMENT
```

Predicted values of NUMBER



the mean values of lesions for strong and weak treatments, along with the 95% CI for lesions.

It plots

Model investigation / hypothesis testing

`summary(tobacco.lmer)`

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [  
## lmerModLmerTest]  
## Formula: NUMBER ~ TREATMENT + (1 | LEAF)  
##   Data: tobacco  
##  
## REML criterion at convergence: 88.7  
##  
## Scaled residuals:  
##     Min      1Q  Median      3Q     Max  
## -1.61850 -0.48453  0.01133  0.40900  1.42778  
##  
## Random effects:  
##   Groups    Name        Variance Std.Dev.  
##   LEAF      (Intercept) 13.63    3.692  
##   Residual              14.46    3.803  
## Number of obs: 16, groups: LEAF, 8
```

```

## 
## Fixed effects:
##           Estimate Std. Error     df t value Pr(>|t|)    
## (Intercept) 34.940     1.874 11.332 18.645 7.38e-10 ***
## TREATMENTWeak -7.879     1.901  7.000 -4.144  0.00433 **  
## ---      
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Correlation of Fixed Effects:
##          (Intr) 
## TREATMENTWk -0.507

```

Let's go through the output:

(a) Random effects paragraph:

- Tells us that LEAF is our random effect, and shows the variance btw leaves, ie. the variance in leave intercepts (13.63). The variance is also expressed as SD (3.692).
- Residual: R is also showing us the variance in the actual observed data, and it is about the same amount of SDs as there is btw leaves (upper row)

When lme4 was written, it happened during the p value and df controversy in the late 2000s. This was particularly the case for mixed effects models. People were saying that you cannot calculate df for mixed effects models, and many people still think like that. So the authors of the lme4 package refused to calculate p values and df. So some other people wrote a package called "lmerTest" that satisfied the Kenwood Roger's p value calculations. If you include that package after lmer4, you will receive the same output, but with p values (included in this case).

(b) Fixed effects paragraph:

- (Intercept) 34.940 => Nr of lesions in the strong inoculation group (ignore this p value, as it is compared to 0)
- TREATMENTWeak -7.879 => Diff btw nr of lesions btw strong and weak treatment. There is evidence that there is a sign decline (p value)

Correlation of Fixed Effects: (Intr) TREATMENTWk -0.507

This is the correlation degree btw treatments (strong vs weak)

```

confint(tobacco.lmer)

## Computing profile confidence intervals ...

##           2.5 %    97.5 %    
## .sig01      0.000000 7.258832
## .sigma      2.333872 6.361451
## (Intercept) 31.216405 38.663855
## TREATMENTWeak -11.829135 -3.929629

```

Reporting this:

"There was sign fewer lesions on the weaker treatment (gr) (conf int -11.8 to -3.9)".

Equivalent for the tidy package (to create a nice table)

```
tidy(tobacco.lmer, effects="fixed", conf.int=TRUE) %>% kable
```

effe	ct	term	estimat	std.err	statistic	df	p.value	conf.lo	conf.hig
			e	or			w		h
fixe	(Intercept)		34.9401	1.8739	18.6447	11.331	0.00000	31.267	38.6130
d		30	89	93		89	00	18	80
fixe	TREATMENT		-	1.9014	-	7.0000	0.00432	-	-
d	Weak		7.87938	60	4.14385	0	82	11.606	4.15258
		1			8			18	8

Examining the strength of the relationship:

This package produces R sq for mixed effects models

```
r.squaredGLMM(tobacco.lmer)
```

```
## Warning: 'r.squaredGLMM' now calculates a revised statistic. See the help page.
```

```
##          R2m          R2c
## [1,] 0.3707882 0.6761026
```

- R2m are marginal R sq values. That is the R sq purely due to the fixed effects (ie. treatment). They explain about 37% of the variability
- R2c are the conditional R sq values. These are the Rsq due to BOTH the fixed and the random effects (so the whole model). The whole model explains about 68% of the variability.
- R sq of the random effects = Diff between the two: 68-37=31% of the variability is caused by the random effects (leaves)

If we wouldn't have added the random effect, 1/3 of the variance would have not been explained, and would have shown up as noise on the residuals, making the test less powerful.

Predictions

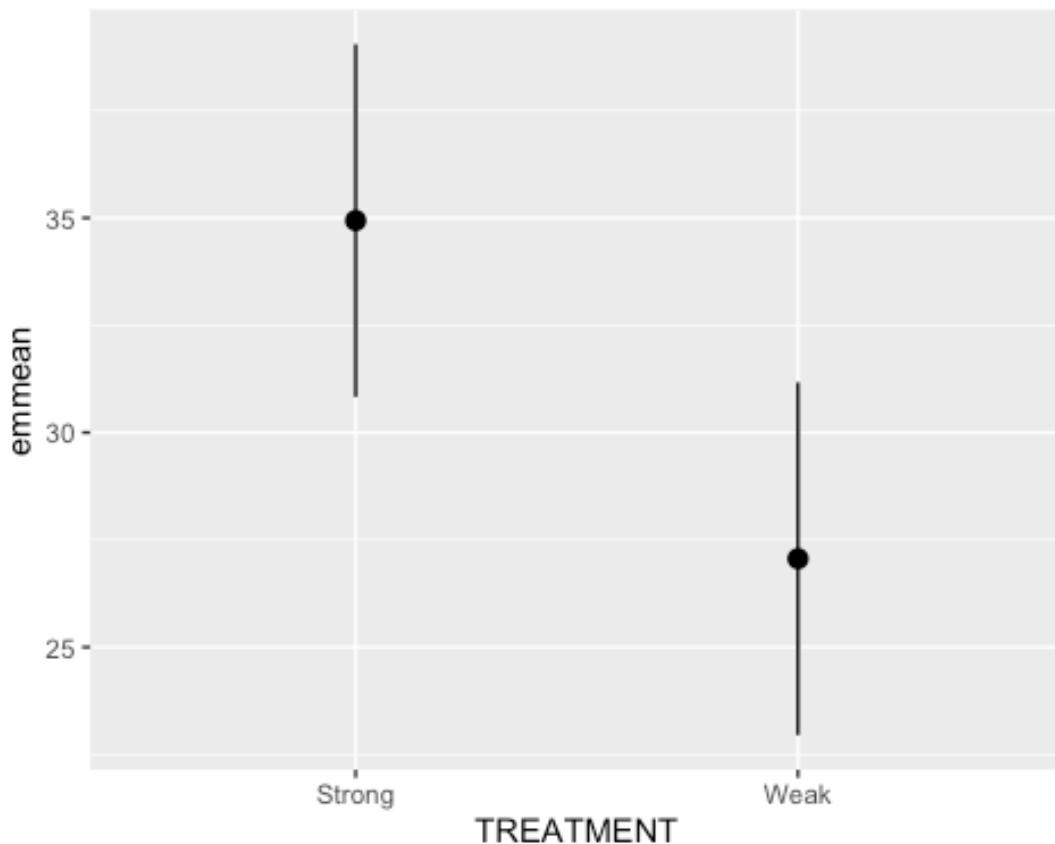
Summary figures

```
newdata<-emmeans(tobacco.lmer, ~TREATMENT) %>%
  as.data.frame()
```

```
newdata

##   TREATMENT    emmean       SE      df lower.CL upper.CL
## 1     Strong 34.94013 1.873989 11.33189 30.83020 39.05006
## 2      Weak 27.06075 1.873989 11.33189 22.95082 31.17068

ggplot(data=newdata)+  
  geom_pointrange(aes(y=emmean,x=TREATMENT,ymin=lower.CL,ymax=upper.CL))
```



References

8_GLMM_Random

Sara Kophamel

03/12/2020

Load glmm_example2

```
library(car)      #for regression diagnostics
## Loading required package: carData

library(broom)    #for tidy output
library(ggfortify) #for model diagnostics

## Loading required package: ggplot2

library(sjPlot)   #for outputs

## Registered S3 methods overwritten by 'lme4':
##   method                  from
##   cooks.distance.influence.merMod car
##   influence.merMod            car
##   dfbeta.influence.merMod     car
##   dfbetas.influence.merMod    car

library(knitr)    #for kable
library(effects)  #for partial effects plots

## lattice theme set by effectsTheme()
## See ?effectsTheme for details.

library(ggeffects) #for partial effects plots
library(emmeans)   #for estimating marginal means
library(MASS)       #for glm.nb
library(MuMIn)     #for AICc
library(tidyverse)  #for data wrangling

## — Attaching packages


---


tidyverse 1.3.0 —

## ✓ tibble  3.0.3    ✓ dplyr   1.0.2
## ✓ tidyr   1.1.2    ✓ stringr 1.4.0
## ✓ readr   1.3.1    ✓ forcats 0.5.0
## ✓ purrr   0.3.4

## — Conflicts


---


— tidyverse_conflicts() —
```

```

## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x dplyr::recode() masks car::recode()
## x dplyr::select() masks MASS::select()
## x purrr::some()   masks car::some()

library(nlme)

##
## Attaching package: 'nlme'

## The following object is masked from 'package:dplyr':
## 
##     collapse

library(lme4)

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyR':
## 
##     expand, pack, unpack

##
## Attaching package: 'lme4'

## The following object is masked from 'package:nlme':
## 
##     lmList

library(glmmTMB)
library(broom.mixed)

## Registered S3 method overwritten by 'broom.mixed':
##   method      from
##   tidy.gamlss broom

library(glmmTMB)      #for glmmTMB
library(DHARMa)       #for residuals and diagnostics

## This is DHARMa 0.3.3.0. For overview type '?DHARMa'. For recent changes,
## type news(package = 'DHARMa') Note: Syntax of plotResiduals has changed in
## 0.3.0, see ?plotResiduals for details

library(performance) #for diagnostic plots
library(see)          #for diagnostic plots

```

DEBUGGING

If we are having issues with packages / bugs -> delete the file ending in .Rdata

I was having issues with the glmmTMB package. In that case, Murray recommend sdoing the following: 1-Unload/detach a package -> detach(package:glmmTMB) 2-Deinstall a package -> remove.packages('glmmTMB') 3-Install a package from the source -> install.packages('glmmTMB',type='source') 4-Restart R 5-And click on the Broom icon on the Environment window to clean our workspace.

NEVER save the workspace if R prompts you to do so.

Scenario

To investigate differential metabolic plasticity in barramundi (*Lates calcarifer*), @Norin-2016-369 exposed juvenile barramundi to various environmental changes (increased temperature, decreased salinity and increased hypoxia) as well as control conditions. Metabolic plasticity was calculated as the percentage difference in standard metabolic rate between the various treatment conditions and the standard metabolic rate under control conditions. They were interested in whether there was a relationship between metabolic plasticity and typical (control) metabolism and how the different treatment conditions impact on this relationship.

A total of 60 barramundi juveniles were subject to each of the three conditions (high temperature, low salinity and hypoxia) in addition to control conditions. Fish mass was also recorded as a covariate as this is known to influence metabolic parameters.

Barramundi

Barramundi

Sampling design

Sampling design

Format of norin.csv data files

FISHID	MASS	TRIAL	SMR_contr	CHANGE
1	35.69	LowSalinity	5.85	-31.92
2	33.84	LowSalinity	6.53	2.52
3	37.78	LowSalinity	5.66	-6.28
..
1	36.80	HighTemperature	5.85	18.32
2	34.98	HighTemperature	6.53	19.06
3	38.38	HighTemperature	5.66	19.03
..

1	45.06	Hypoxia	5.85	-18.61
2	43.51	Hypoxia	6.53	-5.37
3	45.11	Hypoxia	5.66	-13.95
FISHID	Categorical listing of the individual fish that are repeatedly sampled			
MASS	Mass (g) of barramundi. Covariate in analysis			
TRIAL	Categorical listing of the trial (LowSalinity: 10ppt salinity; HighTemperature: 35 degrees; Hypoxia: 45% air-sat. oxygen.			
SMR_contr	Standard metabolic rate (mg/h/39.4 g of fish) under control trial conditions (35 ppt salinity, 29 degrees, normoxia)			
CHANGE	Percentage difference in Standard metabolic rate (mg/h/39.4 g of fish) between Trial conditions and control adjusted for 'regression to the mean'.			

Each fish is measured under each condition, so it is a BLOCKED DESIGN. Fish = random effect Treatment = fixed effect, stressor Other variables: mass, met rate?

The authors wanted to see if there was a relationship btw change and treatment, and what impact does treatment have on metabolism.

We see that we got CHANGE nr from <0 to >0, so we will need to use a GAUSSIAN distribution.

Read in the data

```
norin = read_csv('data/norin.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   FISHID = col_double(),
##   MASS = col_double(),
##   TRIAL = col_character(),
##   SMR_contr = col_double(),
##   CHANGE = col_double()
## )

glimpse(norin)
```

```
## Rows: 180
## Columns: 5
## $ FISHID      <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, ...
## $ MASS        <dbl> 35.69, 33.84, 37.78, 26.58, 37.62, 37.68, 30.62, 50.37,
24...
## $ TRIAL       <chr> "LowSalinity", "LowSalinity", "LowSalinity",
"LowSalinity",...
## $ SMR_contr   <dbl> 5.847466, 6.530707, 5.659556, 6.278200, 4.407336,
4.818589, ...
## $ CHANGE      <dbl> -31.919389, 2.520929, -6.284968, -4.346675, -3.071329, -
15...
```

The categorical var are fish ID (random effect) and trial. Let's transform them to factors

```
norin=norin %>% mutate(FISHID=factor(FISHID),
                         TRIAL=factor(TRIAL))
```

Exploratory data analysis

What does the data look like and are our assumptions met?

Model formula:

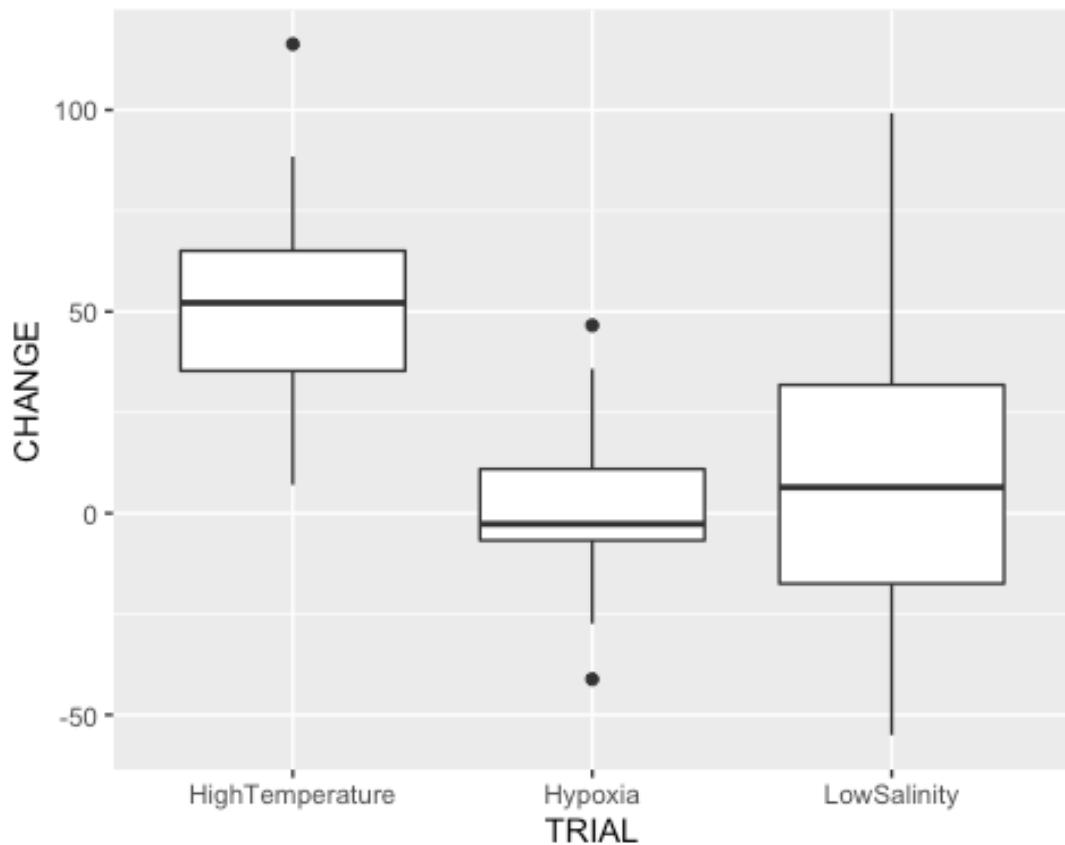
$$\$y_i \sim \mathcal{N}(\mu_i, \sigma^2) \\ \mu_i = \boldsymbol{\beta} \bf{X}_i + \boldsymbol{\gamma} \bf{Z}_i$$

where β and γ are vectors of the fixed and random effects parameters respectively and \bf{X} is the model matrix representing the overall intercept and effects of temperature and (centered) mean fish size on SDA peak. \bf{Z} represents a cell means model matrix for the random intercepts associated with individual fish.

Quick diagnostic plots

All assumptions - boxplots

```
ggplot(norin, aes(y=CHANGE, x=TRIAL))+geom_boxplot()
```

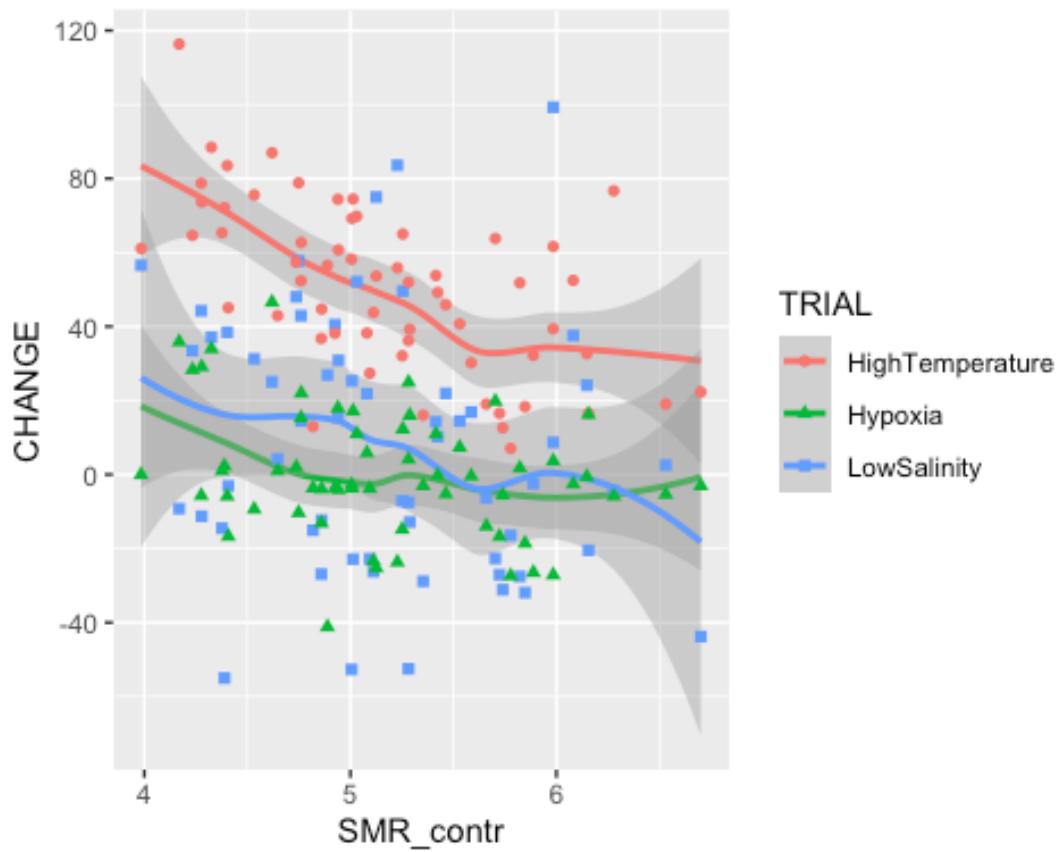


Normality looks alright. The diff btw mean and median is not massively different, and small boxplots have small lines, and big boxplots have big lines. And homogeneity of variance (homogeneous lines) looks ok too. Murray would ignore the outliers until you go further through the model. If there is a problem with the outliers, and they actually change what the data itself represents, you take them out. Otherwise, leave them in (it's just a slightly unusual value).

Linearity

Checking the linearity in relation to control metabolic rate (SMR)

```
ggplot(norin, aes(y=CHANGE, x=SMR_contr, shape=TRIAL, color=TRIAL))+
  geom_smooth() + geom_point()
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



We are

looking for:

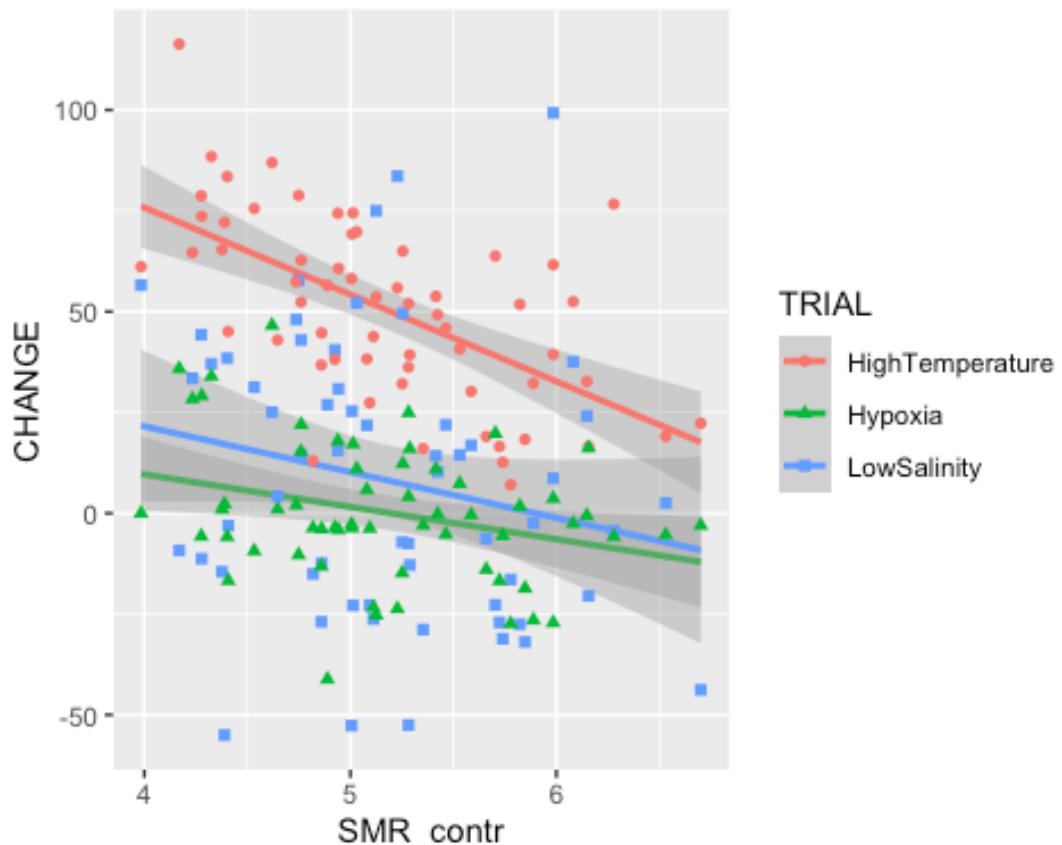
- Changes in linearity. The assumption of linearity in this case seems reasonable for all treatments
- Homogeneity of slopes. If it looks like the direction of the lines look quite different, we will need to set an interaction. Otherwise, the model will just define “one slope to rule them all”. Murray suspects we don’t need an interaction btw SMR and Trial, but we will test that.

Homogeneity of variance

Given that we decided that the data is more or less linear, let’s switch the smoother to linear trends

```
ggplot(norin, aes(y=CHANGE, x=SMR_contr, shape=TRIAL, color=TRIAL))+
  geom_smooth(method="lm") + geom_point()

## `geom_smooth()` using formula 'y ~ x'
```



This plot

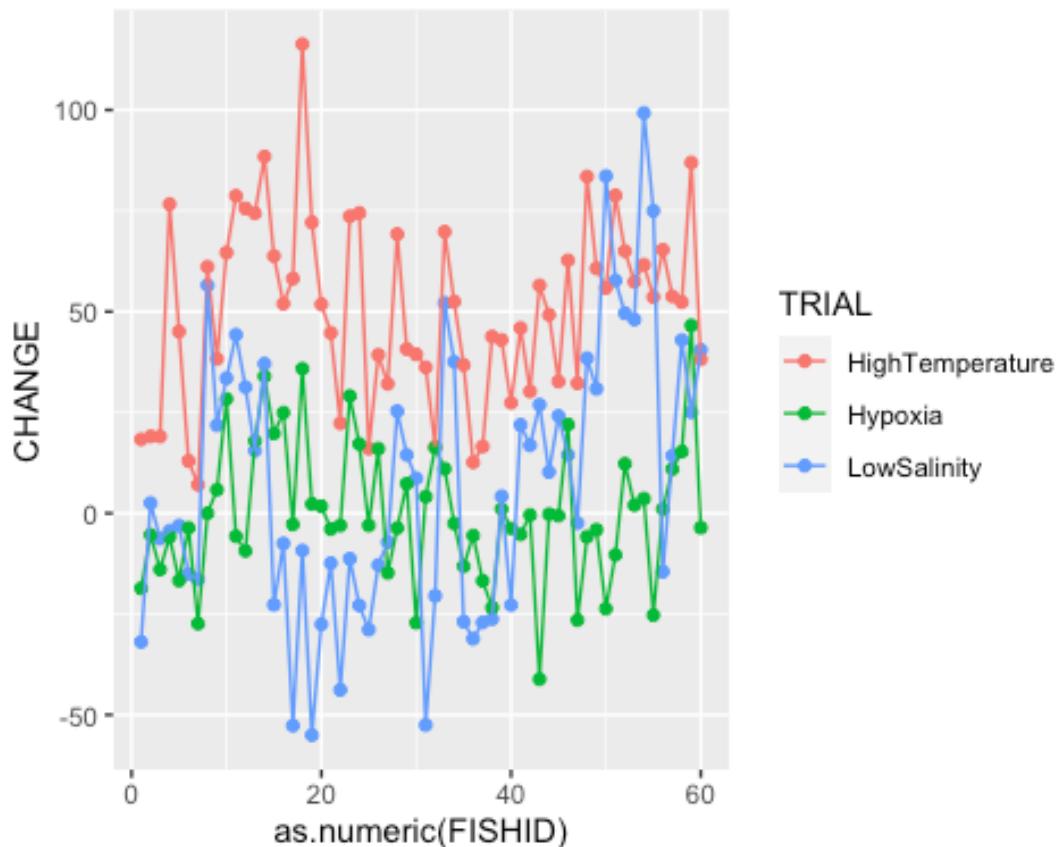
is useful for double-checking the homogeneity of variance. The DOTS indicate if there is homogeneity of variance. They should be homogeneously sprinkled along the lines, which seems to be the case.

It looks like our data, as is, is gonna be fine for modelling.

Random effects (fish)

How do we know how consistent each fish is?

```
ggplot(norin, aes(y=CHANGE, x=as.numeric(FISHID), color=TRIAL))+
  geom_point() + geom_line()
```



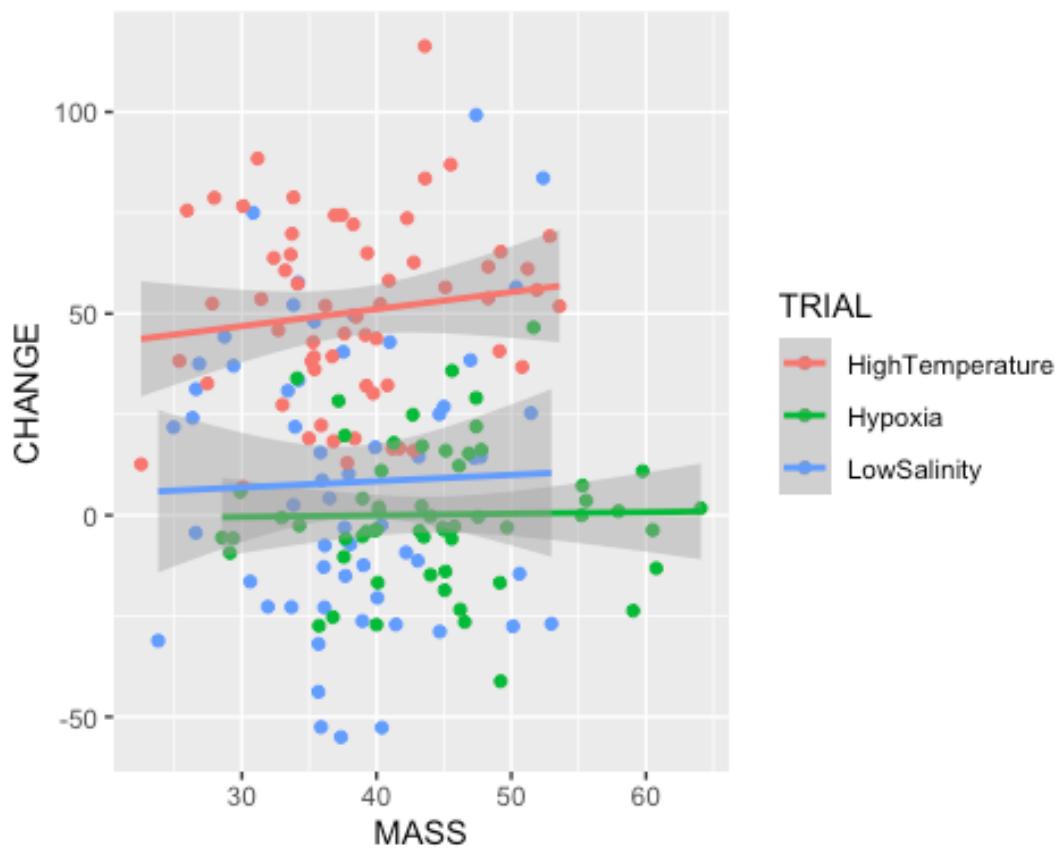
We can see that most of the time, the highest temp has the greatest change in metabolism for most fish. There is a fair degree in consistency on how the fish respond. But it is not completely consistent.

- The change of metabolism DOES vary fish to fish. Our random intercept is therefore gonna be important
- Some fishes have a massive change (fish #18 I think), some respond with lesser change
- Some fish respond more towards trial 1, others towards trial 2 or trial 3

Overall, it looks like our data will satisfy - normality - homogeneity of variance - homogeneity of slopes (perhaps, we'll test it) - We need random intercepts (cause fish ID has got high variability), random slopes (cause slopes change) model

Mass of fish in relation to change in metabolism

```
ggplot(norin, aes(y=CHANGE, x=MASS, color=TRIAL))+
  geom_point()+
  geom_smooth(method="lm")
## `geom_smooth()` using formula 'y ~ x'
```



There does not seem to be a particular association btw change in metabolism, and mass. If there was an interaction, there would be a slope.

Fit the model

For each of the following modelling alternatives, we will:

- establish whether MASS is a useful covariate (based on AICc). This step involves evaluating alternative fixed effects structures. When doing so, we must use Maximum Likelihood (ML) rather than Residual Maximum Likelihood (REML).
- establish whether a random intercept/slope model is useful (based on AICc). This step involves evaluating alternative random effects structures. When doing so, we should use Restricted Maximum Likelihood (REML)

lmer (lme4)

fixed structure

What is the fixed effect we wanna use?

To find out which are the best one(s) for our model, we will create some lmers with different variables as fixed effects

Option 1

```
##Compare models that estimate partial slope for MASS vs an offset for MASS
##must use ML to compare models that vary in fixed effects
norin.lmer1 <- lmer(CHANGE ~ TRIAL*scale(SMR_contr, scale=FALSE) +
offset(MASS) + (1|FISHID), data=norin, REML=FALSE)
```

The response variable is CHANGE.

In this case, the fixed effect is TRIAL.

What about MASS? offset(MASS) -> We have included MASS as an offset (\neq full-on parameter) = it can be used to standardise your model, but it does not have a parameter associated to it. So it does not have df. If you had used a log link model, you would need to write log(MASS)! In this case, we are using a Gaussian distribution. It is an Identity link, with NO transformation. So we leave it as it is.

Example when we'd use this: Let's say we had measured counts of turtles, and we had counted those turtles from transects. And we really wanna express them as density. We got area and counts = density. But density is hard to model, counts aren't. So you model counts against a Poisson distrib with an offset for area -> This is as if you were modelling it for density, which cannot be modelled otherwise.

If we expect that MASS is gonna be a simple 1:1 relationship (eg if I double the mass, I will double the response), go for it as an offset. If you are not sure, or don't think it's gonna be a 1:1 change, DON'T use it as an offset.

$(1|FISHID)$ -> Random intercept for fish

Why are we saying REML=FALSE? Because if we need to compare 2 models that vary ONLY in their fixed effects, we HAVE to use maximum likelihood. They will both be biased to the same degree (so it won't matter). But REML is inappropriate whenever comparing models that VARY in their FIXED effects. We will just be using maximum likelihood.

Murray's annotations to the above chunk (don't get them): - Unfortunately, update() does not remove offset(). - We will just have to write the other model out in full as well.

Option 2

```
norin.lmer2 <- lmer(CHANGE ~ TRIAL*scale(SMR_contr, scale=FALSE) + MASS +
(1|FISHID), data=norin, REML=FALSE)
```

Option 3

Without MASS altogether

```
norin.lmer3 <- lmer(CHANGE ~ TRIAL*scale(SMR_contr, scale=FALSE)+ (1|FISHID),
data=norin, REML=FALSE)
```

Compare these models via AICc

```
AICc(norin.lmer1,norin.lmer2, norin.lmer3)
```

```
##          df     AICc
## norin.lmer1 8 1673.943
## norin.lmer2 9 1666.596
## norin.lmer3 8 1664.473

## Alternatively, we can use sequential Likelihood Ratio Tests (LRT)
```

The model without mass (lmer3) has the same AICc than lmer2. But because lmer3 has lower df, it is less complex. The first model, lmer1, with mass as an offset, has a worse AIC. So far, lmer3 (without MASS) seems to be the best structure for our model.

```
anova(norin.lmer1, norin.lmer2, norin.lmer3)

## Data: norin
## Models:
## norin.lmer1: CHANGE ~ TRIAL * scale(SMR_contr, scale = FALSE) +
## offset(MASS) +
## norin.lmer1: (1 | FISHID)
## norin.lmer3: CHANGE ~ TRIAL * scale(SMR_contr, scale = FALSE) + (1 |
## FISHID)
## norin.lmer2: CHANGE ~ TRIAL * scale(SMR_contr, scale = FALSE) + MASS + (1 |
## norin.lmer2: FISHID)
##      npar    AIC    BIC  logLik deviance   Chisq Df Pr(>Chisq)
## norin.lmer1  8 1673.1 1698.6 -828.55    1657.1
## norin.lmer3  8 1663.6 1689.2 -823.82    1647.6 9.4701  0
## norin.lmer2  9 1665.5 1694.3 -823.77    1647.5 0.0936  1     0.7597
```

Conclusions:

- on the basis of AICc, the model without MASS is considered the most parsimonious (lowest AICc)
- we will proceed with the fixed structure of model 3

random structure - Random slope

Now we can focus on the random structure. Do we want a random slope as well as a random intercept (FISHID)?

Because we will now focus on the random effects, which are biased under max likelihood, we will have to use REML. Let's pick our winning model lmer3, but add REML=TRUE

Option 1 - with random intercept/slope

Option 2 - with random intercept only

```
norin.lmer3a = update(norin.lmer3, REML=TRUE)
```

This model, without random slopes, can be fit.

Let's check which one is best, if lmer3 or rlmer3

```
AICc(norin.lmer3a, norin.lmer3)

##          df      AICc
## norin.lmer3a 8 1637.191
## norin.lmer3 8 1664.473
```

`norin.lmer3a` (with FISHID as random effect) is the winning model.

```
anova(norin.lmer3a, norin.lmer3)

## refitting model(s) with ML (instead of REML)

## Data: norin
## Models:
## norin.lmer3a: CHANGE ~ TRIAL * scale(SMR_contr, scale = FALSE) + (1 | FISHID)
## norin.lmer3: CHANGE ~ TRIAL * scale(SMR_contr, scale = FALSE) + (1 | FISHID)
##          npar     AIC     BIC   logLik deviance Chisq Df Pr(>Chisq)
## norin.lmer3a     8 1663.6 1689.2 -823.82    1647.6
## norin.lmer3     8 1663.6 1689.2 -823.82    1647.6      0   0
```

Conclusions:

- unfortunately, we are unable to fit a model with random intercept/slope
- therefore we will proceed with a random intercept model

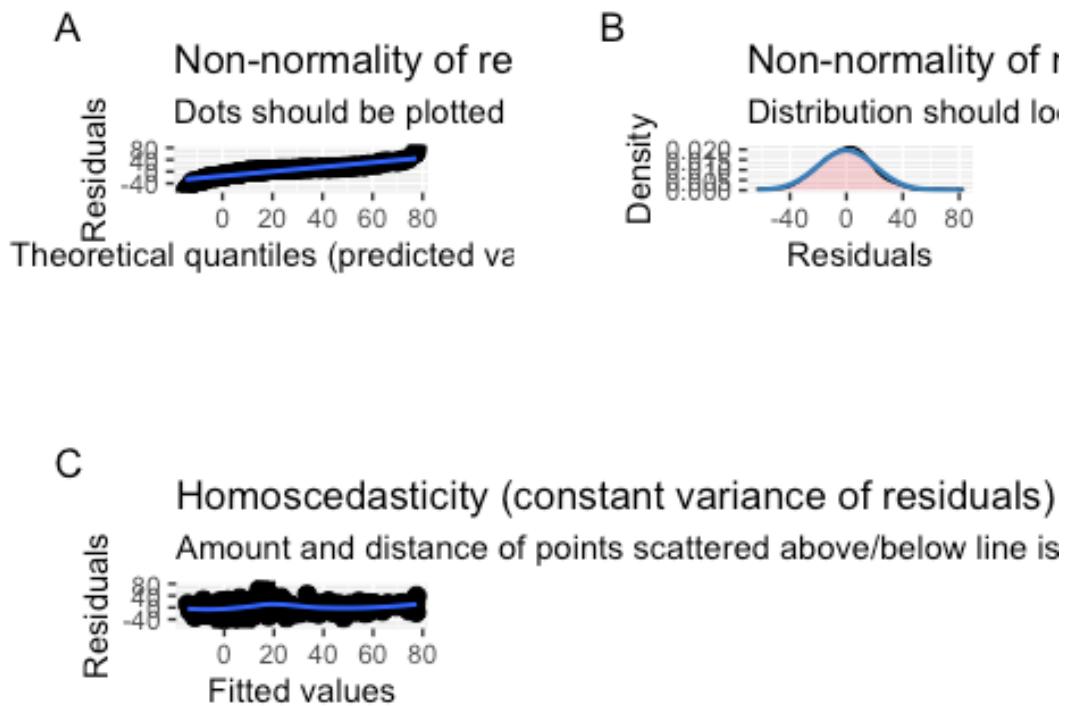
Model validation

Diagnostic plots

```
plot_grid(plot_model(norin.lmer3a, type = "diag")[-2])

## Warning in plot_grid(plot_model(norin.lmer3a, type = "diag")[-2]): Not enough
## tags labels in list. Using letters instead.

## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
```



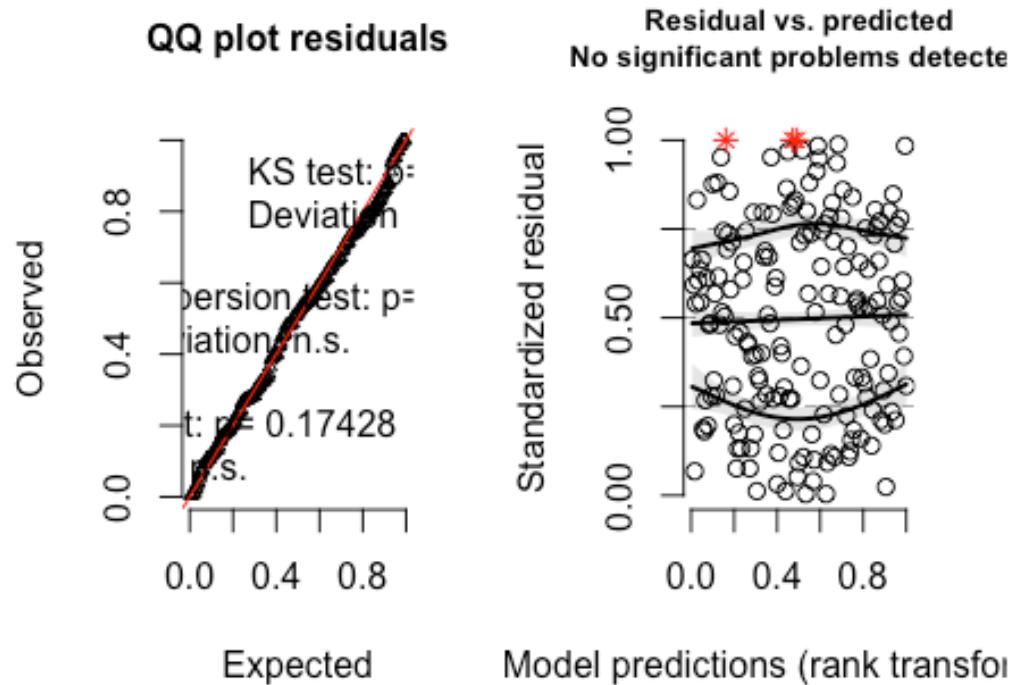
diagnostic plots seem to be ok

All the

DHARMA residuals

```
norin.resid=simulateResiduals(norin.lmer3a,plot=TRUE)
```

DHARMA residual diagnostics



DHARMA residuals look fantastic.

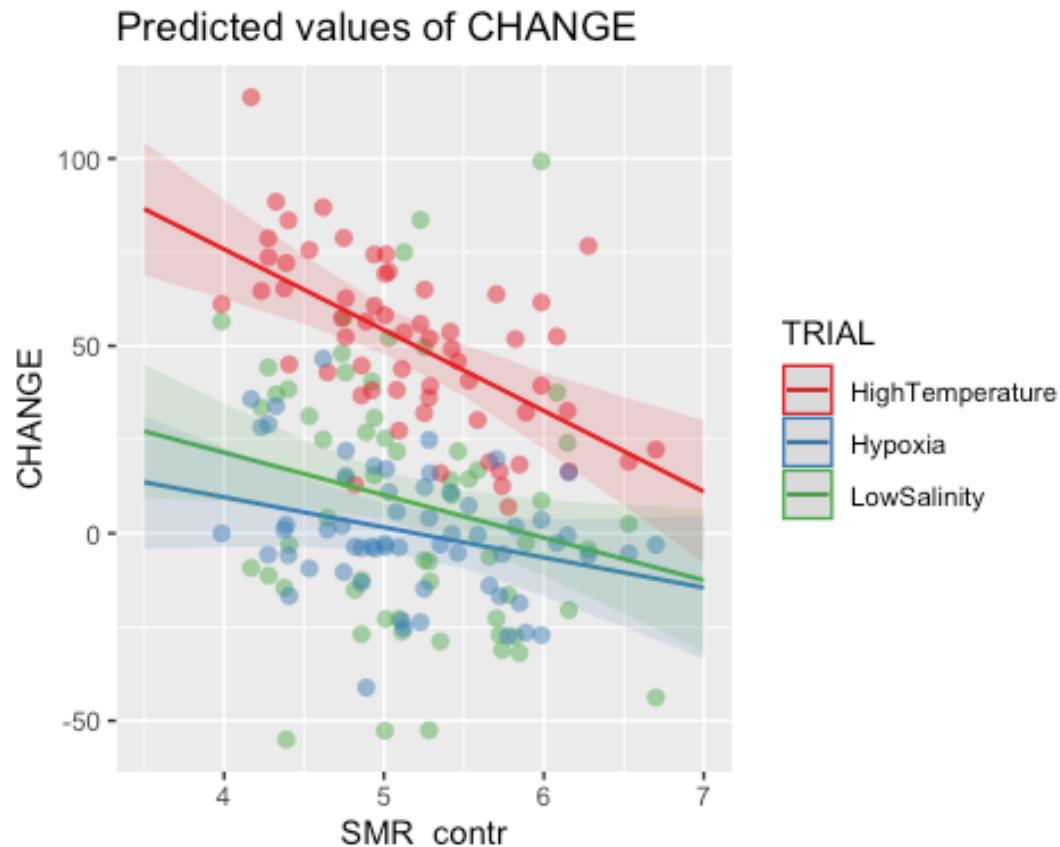
Our model is likely to be reliable (=there is nothing to suggest that it won't be reliable)

The

Partial plots model

Because there IS an interaction, we plot them on ONE graph

```
plot_model(norin.lmer3a, type="eff", terms=c("SMR_contr", "TRIAL"), show.data=TRUE)
```



This is what our model is predicting. We are interested in the effects and in metabolic rate effects. We also overlayed the raw data.

It DOES suggest a trend in change due to SMR control. There is also differences btw trials. Particularly high temp has a strong slope. If the metabolism of the fishes was really high, perhaps we wouldn't see any diff in response of the trials. Whether there is an effect of the trial might depend on where the metabolism starts at.

Model investigation / hypothesis testing

`summary(norin.lmer3a)`

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: CHANGE ~ TRIAL * scale(SMR_contr, scale = FALSE) + (1 | FISHID)
##   Data: norin
##
## REML criterion at convergence: 1620.3
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max 
## -2.8551 -0.6207  0.0224  0.5421  3.7421 
##
## Random effects:
```

```

##  Groups   Name      Variance Std.Dev.
##  FISHID  (Intercept) 99.93    9.996
##  Residual          487.77   22.085
## Number of obs: 180, groups: FISHID, 60
##
## Fixed effects:
##                                     Estimate Std. Error t
## value
## (Intercept)                   50.382    3.130
16.098
## TRIALHypoxia                 -50.226   4.032 -
12.456
## TRIALLowSalinity              -42.223   4.032 -
10.471
## scale(SMR_contr, scale = FALSE) -21.553   5.013 -
4.300
## TRIALHypoxia:scale(SMR_contr, scale = FALSE) 13.511   6.458
2.092
## TRIALLowSalinity:scale(SMR_contr, scale = FALSE) 10.189   6.458
1.578
##
## Correlation of Fixed Effects:
##           (Intr) TRIALH TRIALL s(Ss=F TRIALHs=F
## TRIALHypoxi -0.644
## TRIALLwSlnt -0.644  0.500
## s(SMR_,s=FA  0.000  0.000  0.000
## TRIALH:(s=F  0.000  0.000  0.000 -0.644
## TRIALLS:s=F  0.000  0.000  0.000 -0.644  0.500

```

Let's have a closer look:

Random effects: Groups Name Variance Std.Dev. FISHID (Intercept) 99.93 9.996

Residual 487.77 22.085

=> Variability btw fish is nearly 5x more (from 99.93 to 487.77). This is MASSIVE.

I might need to decide: DO I need more sites, transects, quadrants? Where is the best scale to apply my resources?

Let's have a look at the other outputs:

Fixed effects: Estimate Std. Error df t value Pr(>|t|)

(Intercept) 50.382 3.130 164.489 16.098 < 2e-16 *** Intercept = High temp group at the avg control metabolism (cause that was centered). In the graph, it's the value at high temp at about 5 SMR. The Intercept refers to the MIDDLE of that high temp group. It should be about 50.

TRIALHypoxia -50.226 4.032 116.000 -12.456 < 2e-16 *** = Diff btw High temp group and Hypoxia group

TRIALLowSalinity -42.223 4.032 116.000 -10.471 < 2e-16 *** = Diff btw High temp group and low salinity group

scale(SMR_contr, scale = FALSE) -21.553

=> Shows the initial SMR value The diff btw SMR (middle metabolic rate) and High temp SMR is 21.2 units (double check on graph). It is the slope of the high temp line. If all of the slopes were parallel, the following two numbers (13.51 and 10.18) would be zero

TRIALHypoxia:scale(SMR_contr, scale = FALSE) 13.511 6.458 116.000 2.092 0.0386 *

TRIALLowSalinity:scale(SMR_contr, scale = FALSE) 10.189 6.458 116.000 1.578 0.1174

The slope associated with high temp is -21.

The low salinity slope is 10 units GREATER than the SMR slope, so it would be -21.553+10.189=11. It is a gentler slope. The low salinity slope is not considered sign diff to the high temp slope.

The hypoxia slope has a sign change in comparison to the high temp slope.

The natural next step here is to - There is a neg relationship btw starting metabolism and change of metabolism (what the slopes tell us) - At the avg metabolism, there is a diff btw hypoxia and temp, and hypoxia and low salinity

But is there a diff btw hypoxia and low salinity? We don't know. We might wanna describe how that differs btw diff starting metabolism. We might wanna test that across a couple of diff metabolism parameters.

Predictions / further analyses

To check that, we have to: 1) Look at a Tukey's test for the 3 treatments (temp, hypoxia, low salinity), averaging/marginalising across the whole graph

```
emmeans(norin.lmer3a, pairwise~TRIAL)

## NOTE: Results may be misleading due to involvement in interactions

## $emmeans
##   TRIAL      emmean    SE  df lower.CL upper.CL
##   HighTemperature 50.382 3.13 164     44.20    56.56
##   Hypoxia        0.156 3.13 164     -6.02     6.34
##   LowSalinity    8.159 3.13 164      1.98    14.34
##
## Degrees-of-freedom method: kenward-roger
## Confidence level used: 0.95
##
## $contrasts
##   contrast           estimate    SE  df t.ratio p.value
##   HighTemperature - Hypoxia      50.2 4.03 116 12.456 <.0001
##   HighTemperature - LowSalinity    42.2 4.03 116 10.471 <.0001
##   Hypoxia - LowSalinity       -8.0 4.03 116 -1.985  0.1205
```

```
##  
## Degrees-of-freedom method: kenward-roger  
## P value adjustment: tukey method for comparing a family of 3 estimates
```

Here, we can see the diff btw treatments. There IS a diff sign response btw hypoxia-temp, and btw temp-salinity; but NOT btw hypoxia-salinity. This is arginalised across the whole range of metabolism, rather than at one particular point.

If we want to know what the actual slopes are, we know the slope of the high treatment, and how much the other slopes differ. Let's calculate the slope of each of those trends (temp, hypoxia and salinity), and then comparing each of them.

```
emtrends(norin.lmer3, pairwise~TRIAL, var="SMR_contr")  
  
## $emtrends  
##   TRIAL          SMR_contr.trend    SE  df lower.CL upper.CL  
##   HighTemperature      -21.55 5.01 176     -31.4    -11.66  
##   Hypoxia            -8.04 5.01 176     -17.9     1.85  
##   LowSalinity        -11.36 5.01 176     -21.3    -1.47  
##  
## Degrees-of-freedom method: kenward-roger  
## Confidence level used: 0.95  
##  
## $contrasts  
##   contrast           estimate    SE  df t.ratio p.value  
##   HighTemperature - Hypoxia     -13.51 6.46 124  -2.092  0.0956  
##   HighTemperature - LowSalinity -10.19 6.46 124  -1.578  0.2591  
##   Hypoxia - LowSalinity       3.32 6.46 124   0.514  0.8645  
##  
## Degrees-of-freedom method: kenward-roger  
## P value adjustment: tukey method for comparing a family of 3 estimates
```

No slope is considered diff from one another. There is no trend across SLOPES (#avg values, see previous emmeans)

We will now compare these three treatments at a couple of diff places along this trend = Is there a diff in treatments in very low, medium, and very high metabolism? In other words, we will do a Tukey's test at minimum SMR, avg SMR, and maximum SMR (see graph)

```
# First we define the three SMR points to examine  
norin.grid<-  
with(norin, list(SMR_contr=c(min(SMR_contr), mean(SMR_contr), max(SMR_contr))))  
  
# Then we repeat the emmean, at min, avg and max SMR  
emmmeans(norin.lmer3a, pairwise~TRIAL | SMR_contr, at=norin.grid)  
  
## $emmmeans  
## SMR_contr = 3.98:  
##   TRIAL          emmean    SE  df lower.CL upper.CL  
##   HighTemperature 76.128 6.76 164    62.79    89.47  
##   Hypoxia         9.763 6.76 164   -3.58    23.10
```

```

## LowSalinity      21.734 6.76 164      8.39      35.07
##
## SMR_contr = 5.18:
##   TRIAL          emmean    SE  df lower.CL upper.CL
##   HighTemperature 50.382 3.13 164     44.20     56.56
##   Hypoxia         0.156 3.13 164     -6.02      6.34
##   LowSalinity     8.159 3.13 164      1.98     14.34
##
## SMR_contr = 6.70:
##   TRIAL          emmean    SE  df lower.CL upper.CL
##   HighTemperature 17.586 8.24 164      1.31     33.87
##   Hypoxia        -12.081 8.24 164     -28.36      4.20
##   LowSalinity     -9.133 8.24 164     -25.41      7.15
##
## Degrees-of-freedom method: kenward-roger
## Confidence level used: 0.95
##
## $contrasts
## SMR_contr = 3.98:
##   contrast           estimate    SE  df t.ratio p.value
##   HighTemperature - Hypoxia      66.37  8.70 116  7.624 <.0001
##   HighTemperature - LowSalinity  54.39  8.70 116  6.249 <.0001
##   Hypoxia - LowSalinity       -11.97  8.70 116 -1.375  0.3572
##
## SMR_contr = 5.18:
##   contrast           estimate    SE  df t.ratio p.value
##   HighTemperature - Hypoxia      50.23  4.03 116 12.456 <.0001
##   HighTemperature - LowSalinity  42.22  4.03 116 10.471 <.0001
##   Hypoxia - LowSalinity        -8.00  4.03 116 -1.985  0.1205
##
## SMR_contr = 6.70:
##   contrast           estimate    SE  df t.ratio p.value
##   HighTemperature - Hypoxia      29.67 10.62 116  2.793  0.0167
##   HighTemperature - LowSalinity  26.72 10.62 116  2.515  0.0351
##   Hypoxia - LowSalinity        -2.95 10.62 116 -0.278  0.9584
##
## Degrees-of-freedom method: kenward-roger
## P value adjustment: tukey method for comparing a family of 3 estimates

```

At low SMR, there IS a diff btw temp-hypoxia, temp-salinity, but not with hypoxia-salinity
 At medium and high SMR, the magnitude changes, but the sign is the same (we can make the same conclusions throughout the SMR) = The effects are diminishing.

Summary figures

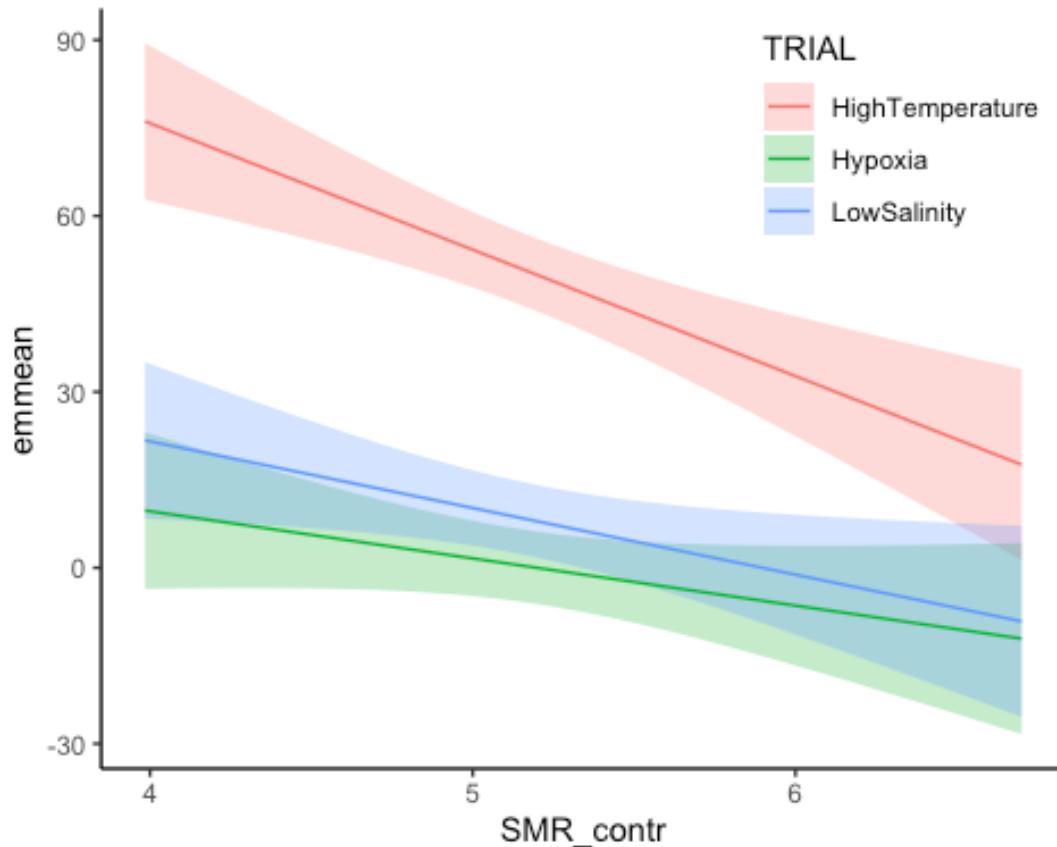
```

norin.grid=with(norin, list(SMR_contr=seq(min(SMR_contr),max(SMR_contr),len=100)))
newdata=emmeans(norin.lmer3a,~SMR_contr|TRIAL,at=norin.grid) %>%

```

```
as.data.frame
```

```
ggplot(data=newdata,aes(y=emmmean,x=SMR_contr))+  
  geom_ribbon(aes(ymin=lower.CL,ymax=upper.CL,fill=TRIAL),alpha=0.3)+  
  geom_line(aes(color=TRIAL))+  
  theme_classic() +  
  theme(legend.position = c(0.99,0.99),  
        legend.justification=c(1,1))
```



References

IGNORE THE FOLLOWING CODE CHUNKS (NOT DISCUSSED IN CLASS)

Additional models that can be created instead (were not discussed in the class)

glmmTMB

fixed structure

Conclusions:

- on the basis of AICc, the model without MASS is considered the most parsimonious (lowest AICc)
- we will proceed with the fixed structure of model 3

random structure

Conclusions:

- unfortunately, we are unable to fit a model with random intercept/slope
- therefore we will proceed with a random intercept model

9_GLMM_Binomial

Sara Kophamel

03/12/2020

Example: glmm_example4

Preparations

Load the necessary libraries

```
library(car)      #for regression diagnostics
library(broom)    #for tidy output
library(broom.mixed)
library(ggfortify) #for model diagnostics
library(sjPlot)   #for outputs
library(knitr)    #for kable
library(effects)  #for partial effects plots
library(ggeffects) #for effects plots in ggplot
library(emmeans)  #for estimating marginal means
library(dfoptim)
library(MASS)     #for glm.nb
library(MuMIn)   #for AICc
library(tidyverse) #for data wrangling
library(DHARMa)   #for residuals and diagnostics
library(nlme)     #for lme
library(optimx)
library(lme4)     #for glmer
library(glmmTMB)  #for glmmTMB
library(performance) #for diagnostic plots
library(see)      #for diagnostic plots
```

Scenario

To investigate synergistic coral defence by mutualist crustaceans, @McKeon-2012-1095 conducted an aquaria experiment in which colonies of a coral species were placed in a tank along with a preditory seastar and one of four symbiont combinations:

- no symbiont,
- a crab symbiont
- a shrimp symbiont
- both a crab and shrimp symbiont.

The experiments were conducted in a large octagonal flow-through seawater tank that was partitioned into eight sections, which thereby permitted two of each of the four symbiont combinations to be observed concurrently. The tank was left overnight and in the morning, the presence of feeding scars on each coral colony was scored as evidence of predation. The experiments were repeated ten times, each time with fresh coral colonies, seastars and symbiont.

The ten experimental times represent blocks (random effects) within which the symbiont type (fixed effect) are nested.

The response is whether they are exposed to predation (1) or not (0).

Each coral is exposed to each symbiont treatment

Read in the data

```
mckeon = read_csv('data/mckeon.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   BLOCK = col_double(),
##   PREDATION = col_double(),
##   SYMBIONT = col_character()
## )

glimpse(mckeon)

## Rows: 80
## Columns: 3
## $ BLOCK      <dbl> 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9,
## 10, 1...
## $ PREDATION <dbl> 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## 0, ...
## $ SYMBIONT  <chr> "none", "none", "none", "none", "none", "none", "none",
## "no..."
```

We want treatment and symbiont to be a categorical (factor) variable, NOT a numeric variable. But let's ahve a look at the SYMBIONT categories first.

```
unique(mckeon$SYMBIONT) # unique() also corrects for duplicates (see
# 10_GLMM_Poisson.Rmd for an example)

## [1] "none"    "shrimp"  "crabs"   "both"
```

Factor conversion and ordering

These are the current unique levels / categories of the SYMBIONT variable (none=No symbiont). If we turn this into a categorical variable, the order of the levels will be ALPHABETICAL: both->crabs->none->shrimp. Does not make sense. We'd like to have "none" as the first one:

```
mckeon=mckeon %>%
  mutate(BLOCK=factor(BLOCK),
        SYMBIONT=factor(SYMBIONT, levels=c("none", "crabs", "shrimp", "both")))

head(mckeon)

## # A tibble: 6 x 3
##   BLOCK PREDATION SYMBIONT
##   <fct>    <dbl> <fct>
## 1 1          0 none
## 2 1          1 none
## 3 2          1 none
## 4 2          1 none
## 5 3          1 none
## 6 3          1 none
```

Naming of random effects

BLOCK is a random effect, so we should NOT name them 1,2,3,4... R could easily recognise them as numeric. We should call them B1,B2,B3,B4... adding a B We do it this way:

```
mckeon=mckeon %>% mutate(CORAL=paste("B",BLOCK))

head(mckeon)

## # A tibble: 6 x 4
##   BLOCK PREDATION SYMBIONT CORAL
##   <fct>    <dbl> <fct>   <chr>
## 1 1          0 none    B 1
## 2 1          1 none    B 1
## 3 2          1 none    B 2
## 4 2          1 none    B 2
## 5 3          1 none    B 3
## 6 3          1 none    B 3
```

If we wanted to call them with number first (eg 1B) instead, we'd write paste(BLOCK,"B") paste0(BLOCK,"B") would remove the space.

Another example about naming random effects (eg in a nested experiment) Imagine you are collecting corals from Reef -> Sites -> Transects

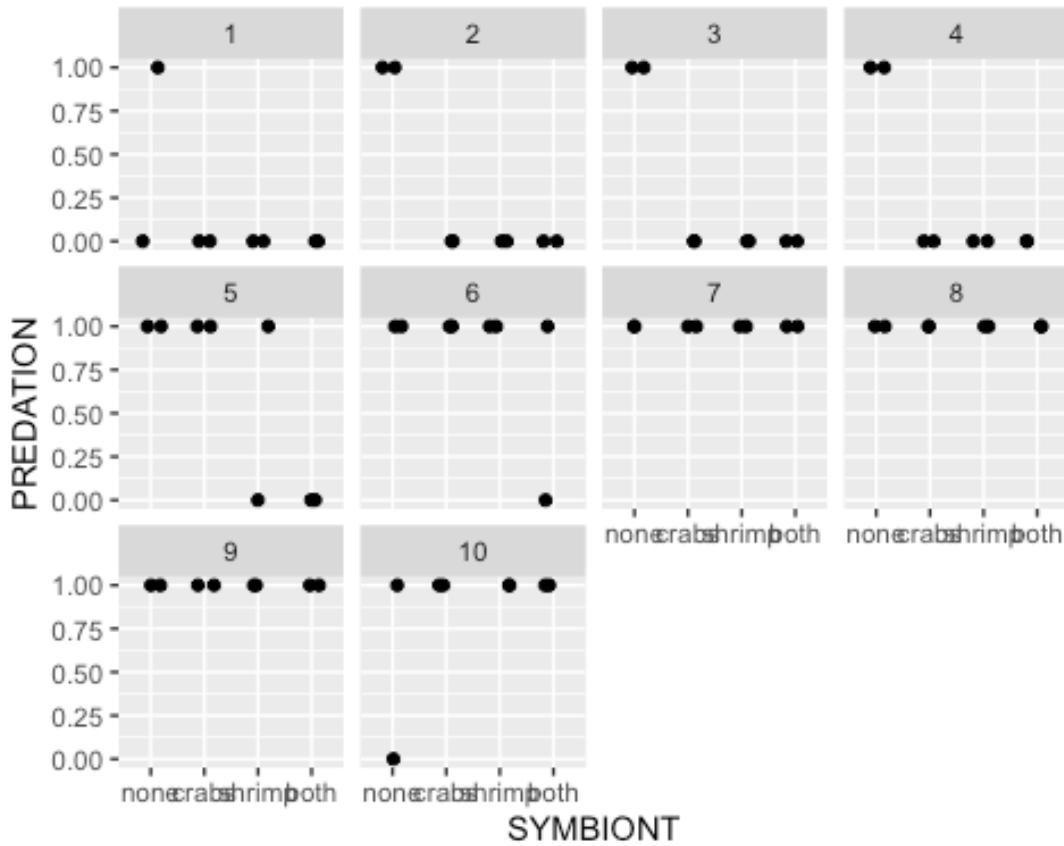
You should not name the areas Reef: A A A B B B Sites: 1 2 3 1 2 3 -> R will think that the 1s are linked, when they really are not. Add a letter in front! (A1, A2, B1...)

Exploratory data analysis

Model formula:

$$\text{where } \beta \text{ and } \gamma \text{ are vectors of the fixed and random effects parameters respectively and } \mathbf{X} \text{ is the model matrix representing the overall intercept and effects of symbionts on the probability of the colony experiencing predation. } \mathbf{Z} \text{ represents a cell means model matrix for the random intercepts associated with individual coral colonies.}$$

```
ggplot(mckeon, aes(y=PREDATION, x=SYMBIONT)) +
  geom_point(position=position_jitter(width=0.2, height=0))+
```



This figure tells us that whenever we have random effects, we should do a plot that shows that treatments are diff for individuals. We should explore each of our animals (ie blocks). In the plot above, each number is a colony of corals. The none group as a whole, very rarely did have no predation. Where there was a symbiont, there sometimes was no predation. The point is: There is some consistency, but there is also a LOT of inconsistency. So we need stats to recheck this.

Fit the model

We will use a binomial family with a logit link (or a binoli).

We will use a generalised linear mixed effects model (needed for binomial models, presence/absence data). Our model, instead of using lmer it uses glmer. Otherwise, the structure and syntax are the same.

```
mckeon.glmer1=glmer(PREDATION~SYMBIONT+(1|BLOCK), data=mckeon,  
family=binomial(link="logit"))
```

We fit a random intercept model. We haven't changed the fixed effects (cause it's only of them, SYMBIONT)

Fitting a slope model

We might maybe wanna fit a random slope model. We'd have to do the following

This model won't work because the model is taking too much time to run.

What if we can't fit the model?

Let's try running it with an optimiser (can be used in small sample sizes)

The optimiser won't work either. A boundary means that when it is testing the SDs, by definition we are testing on a boundary cause $SD \neq 0$. If our SD are close to 0, we get a singularity issue.

Let's try another optimiser

NONE of the optimiser allows the model for conversion. So yeah, you can try running diff optimisation engines, but in the end you are unlikely to convert them.

Now that the optimisers didn't work, we wanna try one last technique. There is a function called allFit. It will try ALL optimisation engines it can find and check if it can somehow convert our model.

```
# mckeon.allFit<-Lme4::allFit(mckeon.glmer2)
```

It turns out none of the optimisation engines are any good. If it DID find one, you could assess which ones are ok. ([OK]≠converged, just OK). In this case, it suggests several optimiser models with OK (eg bobyqa), which we can try fitting. In theory, we'd do that, but in this case Murray tells us that none will work.

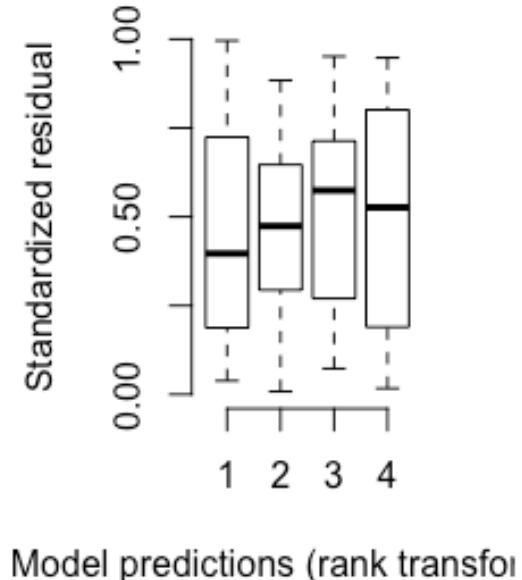
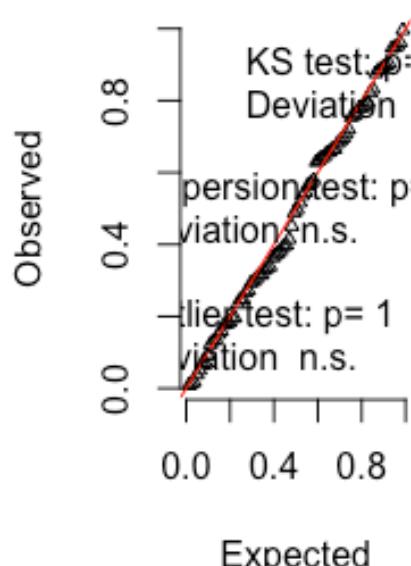
We'll therefore forget about fitting a slope model, and take our initial random intercept model (mckeon.glmer1) and start with the Model Validation

Model validation

```
mckeon.resid=simulateResiduals(mckeon.glmer1, plot=TRUE)
```

DHARMA residual diagnostics

QQ plot residuals

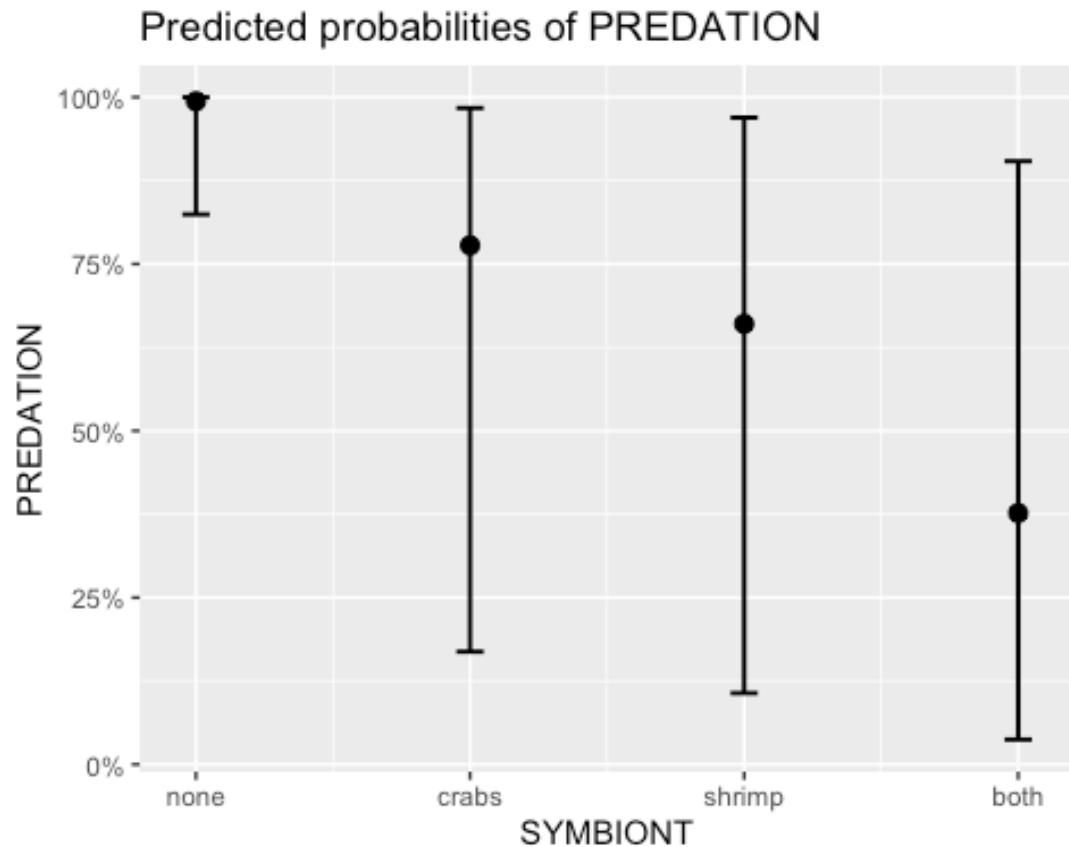


residuals look great.

The

Partial plots

```
plot_model(mckeon.glmer1,type="eff")  
## $SYMBIONT
```



When there is none symbiont, predation is very likely gonna happen. With one symbiont, predation is lower, and with both symbionts, predation is even lower.

Model investigation / hypothesis testing

`summary(mckeon.glmer1)`

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
## Approximation) [glmerMod]
## Family: binomial ( logit )
## Formula: PREDATION ~ SYMBIONT + (1 | BLOCK)
## Data: mckeon
##
##      AIC      BIC      logLik deviance df.resid
##    70.7     82.6     -30.4      60.7       75
##
## Scaled residuals:
##      Min      1Q  Median      3Q      Max
## -23.2076 -0.1730  0.0976  0.2980  0.8944
##
## Random effects:
## Groups Name        Variance Std.Dev.
## BLOCK  (Intercept) 11.81    3.437
```

```

## Number of obs: 80, groups: BLOCK, 10
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 5.096     1.811   2.813 0.00490 **
## SYMBIONTcrabs -3.842    1.465  -2.623 0.00871 **
## SYMBIONTshrimp -4.431    1.552  -2.856 0.00429 **
## SYMBIONTboth   -5.599    1.724  -3.247 0.00117 **
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##          (Intr) SYMBIONTc SYMBIONTs
## SYMBIONTcrb -0.625
## SYMBIONTshr -0.652  0.736
## SYMBIONTbth -0.687  0.732    0.767

```

Let's have a look at the output

Fixed effects: Estimate Std. Error z value Pr(>|z|)
 (Intercept) 5.096 1.811 2.813 0.00490 **

The Estimate numbers are log-transf. It is an odds ratio. We'd have to back-transform them with the exponential

`exp(5.096)`

```
## [1] 163.3671
```

In the absence of any symbionts, corals are 163x more likely to be predated than not predated

What if we transformed it even further back?

`plogis(5.096)`

```
## [1] 0.9939161
```

Here, we are back transforming from logit to probability. In the absence of any symbionts, corals have a 0.994 probability to be predated

What about the crab treatment?

`Estimate`

SYMBIONTcrabs -3.842

This is the diff btw the crab and the none group.

If we back transform this number:

`exp(-3.842)`

```
## [1] 0.02145066
```

$\rightarrow 163.3671 \times 0.021 = 3.430709$ In the presence of crabs, the odds ratio has declined. The prob of being predated is now 3.4x lower than in the none group

And what about both?

```
exp(-5.599)
```

```
## [1] 0.003701563
```

$\rightarrow 163.3671 \times 0.003701563 = 0.6047136$ The odds of being predated is 0.6 = They are more likely to not be predated (<50% chance, 0.6 to 1, which is less than 1 to 1)

We want to ask ourselves: - Is only one symbiont any good? - Is there any diff btw crab and shrimp? - Is there any diff btw one symbiont, and two?

To make all these comparisons, let's define them to minimise them, to not get penalised when doing the t-test. We got 4 groups, so it is enough to make 3 comparisons: - crab vs shrimp - one (crab or shrimp) vs both (crab and shrimp) - none vs symbiont (crab,shrimp and both)

In a matrix, this would look like this

```
cmat=cbind(
  crab_vs_shrimp=c(0,1,-1,0),
  one_vs_both=c(0,1/2,1/2,-1),
  symbiont=c(1,-1/3,-1/3,-1/3)
)
cmat

##      crab_vs_shrimp one_vs_both   symbiont
## [1,]            0        0.0  1.0000000
## [2,]            1        0.5 -0.3333333
## [3,]           -1        0.5 -0.3333333
## [4,]            0       -1.0 -0.3333333
```

Murray emphasizes to do this instead of just running emmeans and do all comparisons, because: - The fewer comparisons we made (\neq all possible comparisons), the more power we will have - emmeans would NOT have merged crab+shrimp, and compared it against each other - emmeans would also have NOT compared none vs any symbiont combination

We are now testing if there is an overlap in comparisons ($\neq 0$), eg AvsB, BvsC and AvsC(this last one is not needed). We want to see a diagonal line ($=$ no overlap). We are essentially testing if our matrix was created properly.

```
round(crossprod(cmat),1)
```

```
##      crab_vs_shrimp one_vs_both   symbiont
## crab_vs_shrimp            2        0.0     0.0
```

```

## one_vs_both          0       1.5    0.0
## symbiont            0       0.0    1.3

crossprod(cmat) # gives you the same output as the chunk above, but with
# commas

##               crab_vs_shrimp one_vs_both symbiont
## crab_vs_shrimp           2       0.0 0.000000
## one_vs_both              0       1.5 0.000000
## symbiont                0       0.0 1.333333

```

We want to see all zeros in the corners, which is the case.

Predictions

```

emmeans(mckeon.glmer1, ~SYMBIONT, contr=list(SYMBIONT=cmat), type="response")

## $emmeans
## SYMBIONT prob   SE  df asymp.LCL asymp.UCL
## none     0.994 0.011 Inf   0.8243   1.000
## crabs    0.778 0.251 Inf   0.1690   0.984
## shrimp   0.660 0.319 Inf   0.1068   0.969
## both     0.377 0.329 Inf   0.0373   0.904
##
## Confidence level used: 0.95
## Intervals are back-transformed from the logit scale
##
## $contrasts
## contrast          odds.ratio      SE  df z.ratio p.value
## SYMBIONT.crab_vs_shrimp    1.80  1.98 Inf  0.536  0.5921
## SYMBIONT.one_vs_both       4.32  4.42 Inf  1.429  0.1530
## SYMBIONT.symbiont        101.91 146.78 Inf  3.210  0.0013
##
## Tests are performed on the log odds ratio scale

```

- no diff btw crab and shrimp
- no diff btw 1 or 2 symbionts
- sign diff about having a symbiont at all (odds ratio 101, which is quite substantial) -> created by comparing none to the avg of shrimp+crab+both

R square

```

r.squaredGLMM(mckeon.glmer1)

## Warning: 'r.squaredGLMM' now calculates a revised statistic. See the help
page.

## Warning: The null model is correct only if all variables used by the
original
## model remain unchanged.

```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =  
control$checkConv, :  
## Model failed to converge with max|grad| = 0.0307141 (tol = 0.002,  
component 1)  
  
## R2m R2c  
## theoretical 0.2282003 0.8318487  
## delta 0.2166278 0.7896639  
  
R2m R2c
```

- theoretical 0.2282003 0.8318487 are the original R sq calc, which is fine for GAUSSIAN distributions.
- But if you got a diff distribution, then you are better off using the delta distribution:
R2m R2c delta 0.2166278 0.7896639

Conclusion: We can explain around 22% from the fixed effect (types of symbiont), and around 80% if we take into acc the individual blocks as well (random intercept). The random intercept effect itself would explain 80-22=58%.

10_GLMM_Poisson_and_NegBinomial

Sara Kophamel

03/12/2020

Preparations

Load the necessary libraries

```
library(car)      #for regression diagnostics
library(broom)    #for tidy output
library(broom.mixed) #for tidy output
library(ggfortify) #for model diagnostics
library(sjPlot)   #for outputs
library(knitr)    #for kable
library(effects)  #for partial effects plots
library(ggeffects) #for effects plots in ggplotjk
library(emmeans)  #for estimating marginal means
library(MASS)     #for glm.nb
library(MuMIn)    #for AICc
library(tidyverse) #for data wrangling
library(DHARMa)   #for assessing dispersion etc
library(glmmTMB)  #for glmmTMB
library(performance) #for diagnostic plots
library(see)       #for diagnostic plots
library(lme4)      #for glmer
library(glmmTMB)  #for glmmTMB
```

Scenario

Some ornithologists were interested in the degree of sibling negotiations in owl chicks. Specifically, they wanted to explore how sibling negotiations were affected by feeding satiety and the sex of the parent returning to the nest. The ornithologists had accessed to a number of owl nests and were able to count (via recording equipment) the number of sibling negotiations (calls) that the owl chicks made when the parent returned to the nest.

We could hypothesise that the chicks might call more if they were hungry. As part of the investigation, the researchers were able to provided supplementary food. As such, they were able to manipulate the conditions such that sometimes the chicks in a nest would be considered deprived of supplementary food and at other times they were satiated.

As a parent returned, the researchers recorded the number of sibling negotiations (calls) along with the sex of the parent. Since the number of calls is likely to be a function of the

number of chicks (the more chicks the more calls), the researchers also counted the number of siblings in the brood.

Each nest was measured on multiple occasions. Hence, we must include the nest as a random effect to account for the lack of independence between observations on the same set of siblings.

In summary: - We want to see if our chicks argue more under specific circumstances. - Data was collected from semi-captive popul, where they supplemented the food. - They had two treatments: Supplemented_Food (satiated) vs NonSupplemented_Food (deprived). - They also were interested in whether the chicks called more or less depending on which parent returned -> this is another variable, the sex of the parent returning. - They reused the nests for different breeding pairs, so that every nest was its own control -> BLOCKED design = Each nest was subjected to both treatments, and we had both parent in all nests - They also looked at brood size, cause the more chicks, the more calls there will be. So we need the brood size as an offset, as a covariate. It will determine the amount of calls per NEST (#chick) - We are not gonna use the variable calls/chick -> Only useful for plotting, but not for the model (because of the above)

A POISSON distribution seems reasonable. # PRETTY MUCH ALL ECOLOGICAL DATA IS GONNA BE POISSON DISTRIBUTIONS (!)

Read in the data

```
owls = read_csv('data/owls.csv', trim_ws=TRUE)
## Parsed with column specification:
## cols(
##   Nest = col_character(),
##   FoodTreatment = col_character(),
##   SexParent = col_character(),
##   ArrivalTime = col_double(),
##   SiblingNegotiation = col_double(),
##   BroodSize = col_double(),
##   NegPerChick = col_double()
## )
glimpse(owls)
## Rows: 599
## Columns: 7
## $ Nest              <chr> "AutavauxTV", "AutavauxTV", "AutavauxTV",
## "Autavau...
## $ FoodTreatment     <chr> "Deprived", "Satiated", "Deprived", "Deprived",
## "D...
## $ SexParent         <chr> "Male", "Male", "Male", "Male", "Male",
## "M...
## $ ArrivalTime       <dbl> 22.25, 22.38, 22.53, 22.56, 22.61, 22.65,
## 22.76, 2...
```

```
## $ SiblingNegotiation <dbl> 4, 0, 2, 2, 2, 2, 18, 4, 18, 0, 0, 3, 0, 3, 3,
6, ...
## $ BroodSize           <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, ...
## $ NegPerChick         <dbl> 0.8, 0.0, 0.4, 0.4, 0.4, 0.4, 3.6, 0.8, 3.6,
0.0, ...
```

(!) Declare variables as categorical that should be categorical: Nest - FoodTreatment and SexParent should be ok, as it's only very few, but we'll change them either way

Data preparation

```
owls=owls %>% mutate(Nest=factor(Nest),
                      FoodTreatment=factor(FoodTreatment),
                      SexParent=factor(SexParent),
                      NCalls=SiblingNegotiation) # we also give this weird
variable name a more understandable one. This is the amount of calls per nest

head(owls)

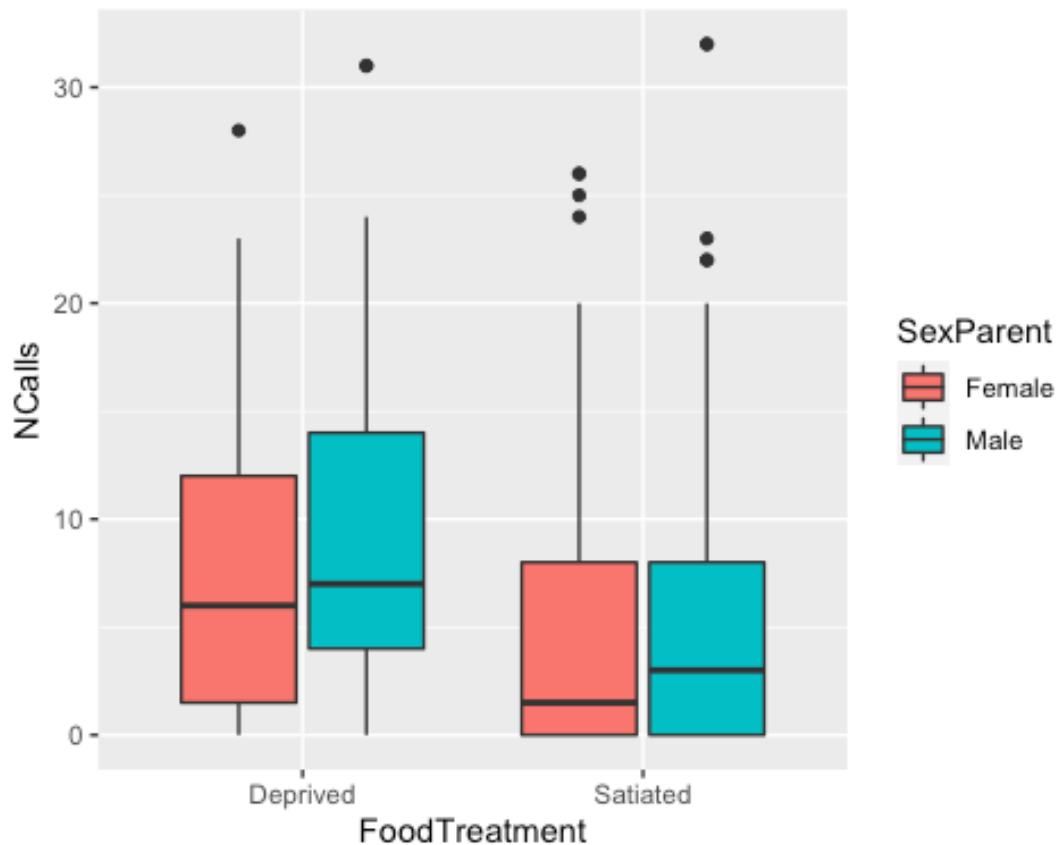
## # A tibble: 6 x 8
##   Nest  FoodTreatment SexParent ArrivalTime SiblingNegotiat... BroodSize
##   <fct> <fct>        <fct>     <dbl>            <dbl>      <dbl>
## 1 Auta... Deprived    Male       22.2             4          5
## 2 Auta... Satiated    Male       22.4             0          5
## 3 Auta... Deprived    Male       22.5             2          5
## 4 Auta... Deprived    Male       22.6             2          5
## 5 Auta... Deprived    Male       22.6             2          5
## 6 Auta... Deprived    Male       22.6             2          5
## # ... with 2 more variables: NegPerChick <dbl>, NCalls <dbl>
```

Exploratory data analysis

Exploratory plots

x axis for food treatm or sex of parent color or fill or sth for the other

```
owls %>%
  ggplot(aes(FoodTreatment,NCalls,fill=SexParent))+#
  geom_boxplot()
```



There is probably even a relationship btw mean and variance. There is quite a few 0s in our dataset, and no bottom whiskers for the satiated group, suggesting there MUST be some zeros (check it in the dataset).

The more chicks there are in the nest, the more calls there will be, so we wanna check that

```
owls %>%
  ggplot(aes(BroodSize, NCalls, color=SexParent))+
  geom_point()+
  geom_smooth(method="lm")+
  facet_grid(~FoodTreatment)+
  coord_equal() # allows us to see whether the Line is one on one (whether there is a 45° angle btw the x and y axes in the lines)
## `geom_smooth()` using formula 'y ~ x'
```



The more broods, the more calls. The lines in the satiated group have vaguely a 45° angle, so it is a one on one relationship (same slope behaviour for x and y axis). They do not seem parallel in the Satiated group. The 1:1 line tells us that we will need an offset for BroodSize (!)

For the fixed str, we will include Sex and FoodTreatment, and get them to interact. We will also get an offset for BroodSize.

Model formula:

$$\text{y}_i \sim \text{Pois}(\lambda_i) \quad \ln(\lambda_i) = \boldsymbol{\beta} X_i + \boldsymbol{\gamma} Z_i$$

where $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are vectors of the fixed and random effects parameters respectively and X is the model matrix representing the overall intercept and effects of food treatment, sex of parent, arrival time (and various interactions) on the number of sibling negotiations. Brood size was also incorporated as an offset. Z represents a cell means model matrix for the random intercepts associated with individual nests.

Fit the model

In theory, you should fit a NULL model first (with no interactions or additions, the most basic one). But we will skip that step for the purpose of the course.

Model 1 - with interaction

```
owls.glmmTMB1<-glmmTMB(NCalls~FoodTreatment*SexParent+
                           offset(log(BroodSize))+(1|Nest),data=owls,
                           family=poisson(link="log"),REML=FALSE)
```

If you are gonna use an offset \rightarrow offset(log(BroodSize)) \rightarrow You have to transform it using the log link function (!) Otherwise, it is not gonna be 1 on 1. An because we are gonna compare a couple fixed models, we are gonna use REML=FALSE.

We have to decide btw an additive model (that includes the interaction), and one that does not include the interaction.

Model 2 - without interaction, just addition

```
owls.glmmTMB2<-update(owls.glmmTMB1,~.-FoodTreatment:SexParent)
# The - removes the interaction component, leaving the additive component
```

Model validation

```
AICc(owls.glmmTMB1,owls.glmmTMB2)

##           df      AICc
## owls.glmmTMB1  5 5212.877
## owls.glmmTMB2  4 5214.320
```

The models are within two units, but model 2 has fewer df. Now we could fit this model, and continue with the stats. If we did this, we would have got a non sign interaction btw NCalls across Treatments (we won't do it, as it is the same procedure as always).

Alternative model validation - likelihood test

```
anova(owls.glmmTMB1,owls.glmmTMB2)

## Data: owls
## Models:
## owls.glmmTMB2: NCalls ~ FoodTreatment + SexParent + (1 | Nest) +
##                   offset(log(BroodSize)), zi=~0, disp=~1
## owls.glmmTMB1: NCalls ~ FoodTreatment * SexParent + offset(log(BroodSize))
##                   + , zi=~0, disp=~1
## owls.glmmTMB1:     (1 | Nest), zi=~0, disp=~1
##                   Df      AIC      BIC    logLik deviance Chisq Chi Df Pr(>Chisq)
## owls.glmmTMB2  4 5214.3 5231.8 -2603.1    5206.3
## owls.glmmTMB1  5 5212.8 5234.8 -2601.4    5202.8 3.4766      1   0.06224 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Says that there is no sign diff btw them ($\text{Pr}(>\text{Chisq})$), so we go for the one with lower df.

Conclusions:

- there is evidence that the model does not fit that well. It is evidently zero inflated and possibly also overdispersed.
- it would seem that a zero-inflated Poisson or even a zero-inflated Negative Binomial would be a sensible next step.
- zero-inflated models cannot be fit in `glmer()`, so we will proceed with `glmmTMB()` only.

Model refit and validation

We want to include RANDOM effects, so let's refit our model 2 (winning model), but including random effects (so `REML=TRUE`)

Model 2a (winning model) for fixed effects comparisons

```
owls.glmmTMB2a<-update(owls.glmmTMB2, REML=TRUE)
```

Because Lorenzo wanted to know what happens if we actually PICK the more complex model, Murray said to check it and to do the exercise using the MORE COMPLEX model (`glmm1`); damn.

Model 1a (more complex model) for fixed effects comparisons

SO this is model 1, but with random effects included (\neq model 2)

```
owls.glmmTMB1a<-update(owls.glmmTMB1, REML=TRUE)
```

Model 1b with random int random slope term for FoodTreatment

```
owls.glmmTMB1b<-update(owls.glmmTMB1a, ~.-(1|Nest)+(FoodTreatment|Nest))
```

Model 1c with random int random slope term for SexParent

```
owls.glmmTMB1c<-update(owls.glmmTMB1a, ~.-(1|Nest)+(SexParent|Nest))
```

Model 1d with random int random slope term for FoodTreatment*SexParent interaction

```
owls.glmmTMB1d<-update(owls.glmmTMB1a, ~.-(1|Nest)+(FoodTreatment*SexParent|Nest))
```

Now if it turned out that we went with the simpler, additive model, we probably wouldn't have gone and made model 1d (with the interaction). *We would have done:*

```
owls.glmmTMB1d<-update(owls.glmmTMB1a, ~.-(1/Nest)+(FoodTreatment+SexParent/Nest))
```

instead, as our fixed effects were already telling us that there was NO interaction (it's nonsense putting a in that case)

```
AICc(owls.glmmTMB1a,owls.glmmTMB1b,owls.glmmTMB1c,owls.glmmTMB1d)

##          df     AICc
## owls.glmmTMB1a 5 5228.924
## owls.glmmTMB1b 7 4913.153
## owls.glmmTMB1c 7 5138.496
## owls.glmmTMB1d 14 4782.402
```

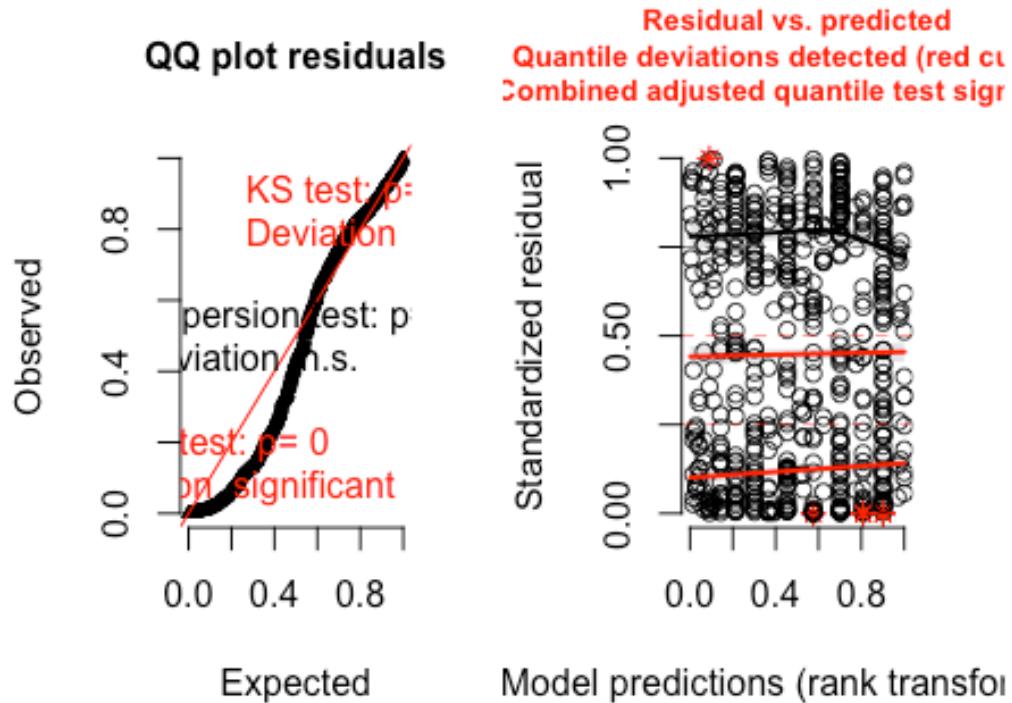
It seems like model 1d is the best model. It tells us that the random effect (FoodTreatment*SexParent|Nest) creates a lot of variability. By including that interaction in our model, we will REDUCE variability of the residuals.

Partial plots

```
owls.resid=simulateResiduals(owls.glmmTMB1d,plot=TRUE,integerResponse=TRUE)

## DHARMA:plot used testOutliers with type = binomial for computational
## reasons (nObs > 500). Note that this method may not have inflated Type I
## error rates for integer-valued distributions. To get a more exact result, it
## is recommended to re-run testOutliers with type = 'bootstrap'. See
## ?testOutliers for details
```

DHARMA residual diagnostics



The plot is NOT good! There is outliers, and pretty much all tests are giving us warning signs.

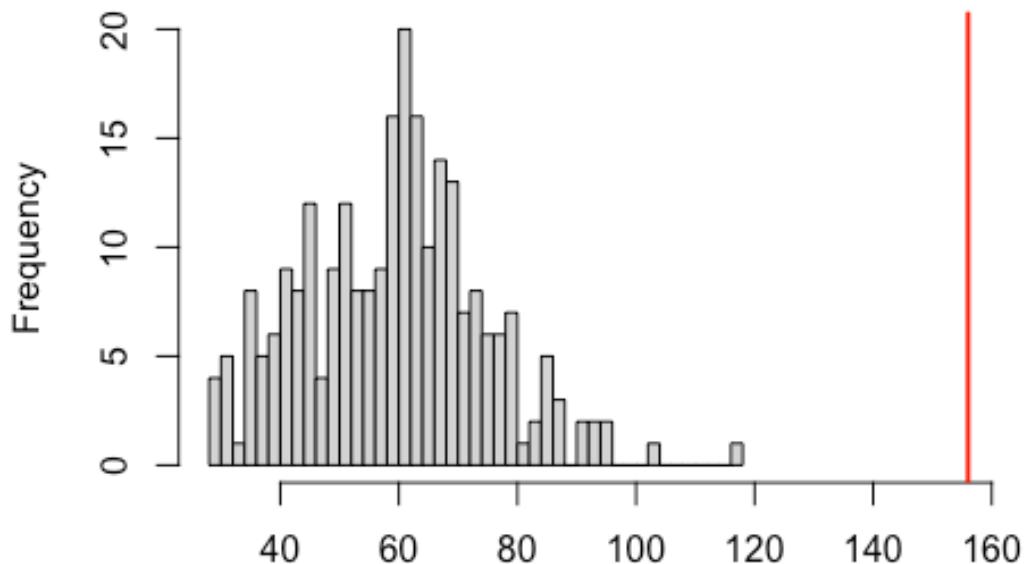
WHy is that model NOT a good fit? First of all, cause we had some zeros in our data.

Zero inflation test

Maybe we had TOO MANY zeroes. We can query that with a zero inflation test.

```
testZeroInflation(owls.resid) # bases the tst on the residuals (better test than the next one, but doesn't tell you how many zeros you got). This test tells you whether it is substantial or not.
```

DHARMA zero-inflation test via comparison to expected zeros with simulation under H0 = fitted model



Simulated values, red line = fitted model. p-value (two.sided) = 1

```
##  
## DHARMA zero-inflation test via comparison to expected zeros with  
## simulation under H0 = fitted model  
##  
## data: simulationOutput  
## ratioObsSim = 2.6152, p-value < 2.2e-16  
## alternative hypothesis: two.sided
```

The zero inflation test suggests that the model is highly zero inflated = sign p value. We have to consider this.

We could also use the performance zero inflation test

```
performance::check_zeroinflation(owls.glmmTMB1d) # bases the test on the  
# model, not on the residuals, which is not great (but it tells you how many  
# zeros to expect, so that's good)  
  
## # Check for zero-inflation  
##  
## Observed zeros: 156  
## Predicted zeros: 44  
## Ratio: 0.28  
  
## Model is underfitting zeros (probable zero-inflation).
```

There should be 44 zeros in the dataset, but we got 156! That's heaps.

Why do we have so many zeros? The calls were recorded by a tape recorder. BUT an owl might have been calling really quietly / not audible for the recorder. We call those zeros "zero inflated", cause there is an inflated number of zeros in the data. Some of these zeros are not true zeros (they would have been a 1,2...). We probably did not detect them correctly. That distribution SHOULD be a POISSON. But on top of that, there is the ability to detect the calls. There is a probability of detection that leads to what our data really are, what we actually record.

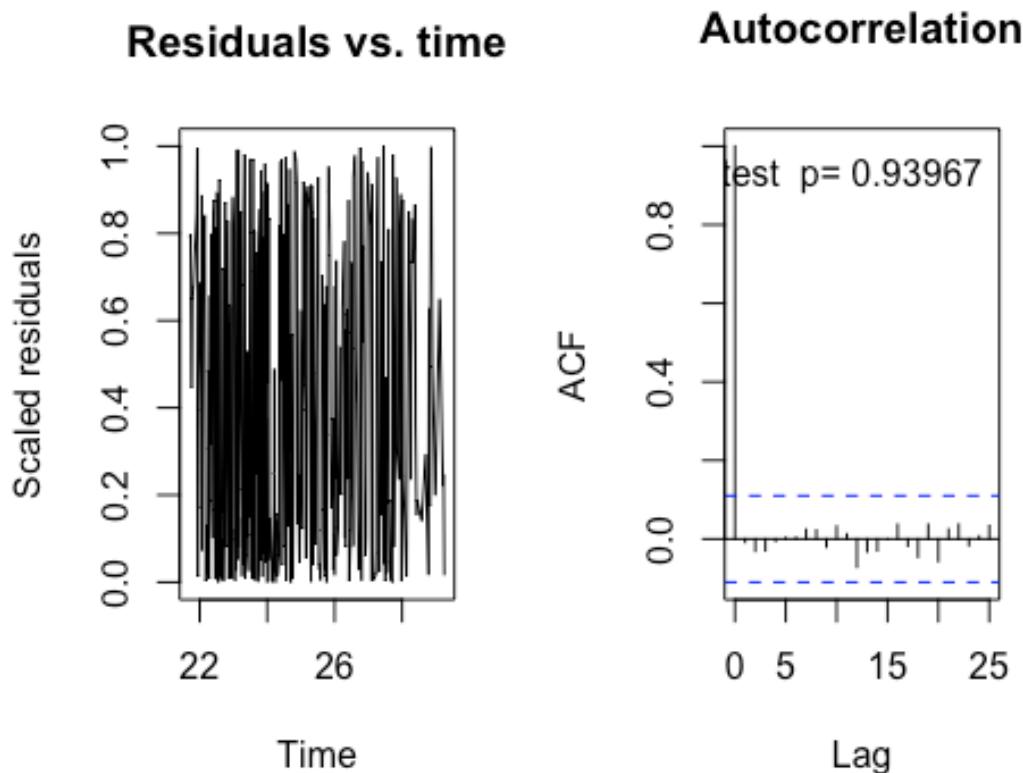
We therefore got: - the count process - a binomial process that determines whether we can detect the calls or not In other words, we have to fit a model which has got those two processes embedded. glmmTMB actually allows to fit zero inflated models. This is NOT the case in lmer models.

Before we go and fit a zero inflated model, we also have to look at the following: We have to randomise things. These data were collected in the evening, and there might be an autocorrelation pattern due to that time, when the parents were returning.

So we should explore autocorrelation. DHARMA residuals allow you to do that (you usually don't have to do this):

```
# We redefine the residuals so that they include the time
owls.resid1<-
recalculateResiduals(owls.resid, group=interaction(owls$ArrivalTime, owls$Nest),
,
aggregateBy=mean)

# Then, we run the autocorrelation test for ArrivalTime included
testTemporalAutocorrelation(owls.resid1, time=unique(owls$ArrivalTime)) # we
have to specify time=unique to correct for the duplicates!!
```



```
##  
## Durbin-Watson test  
##  
## data: simulationOutput$scaledResiduals ~ 1  
## DW = 2.0085, p-value = 0.9397  
## alternative hypothesis: true autocorrelation is not 0
```

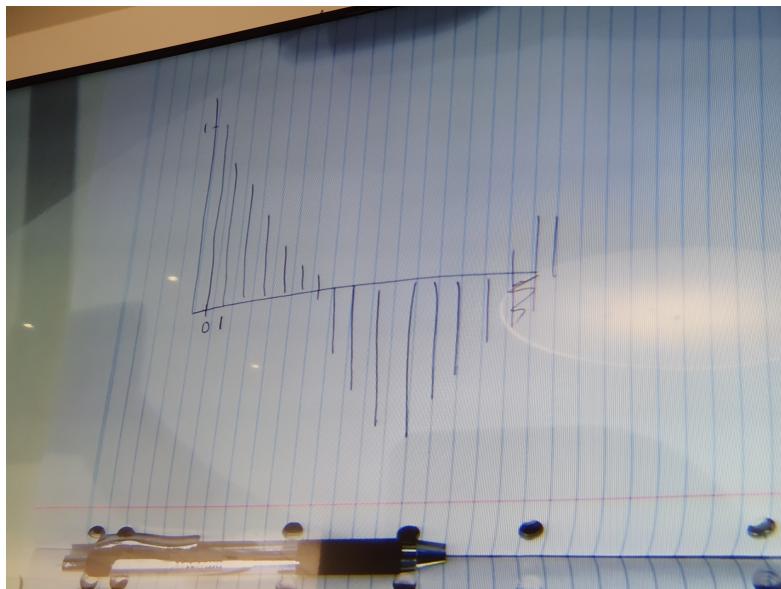
We tell R to provide the residuals and a variable that indicates the time.

We are testing whether the residuals are more similar if we are close together in time. If residuals that are closer together in time are more similar to one another. Because we always assumed that the residuals are independent.

If we look at the R plot, the autocorrelation function plot, it plots the correlation btw a residual and a residual at a certain lag (=a certain time distance apart). You will always have a spike and a lag of zero (where it correlates with itself). If we shift them one unit apart, we see that they are NOT correlated.

In other words, if the lines protrude outside of the blue dotted lines, we have a problem. We would then need to fit a first order autocorrelation structure

If they were correlated, we would see the following:



After a lag of 1, they are correlated. After a lag of 2, they are still correlated...until they come back to normal after a while. That would indicate some sort of autocorrelation structure (second pic)

We could impose that structure on our data rather than the default (the one with the ps and no 1,2,3)

But that's not the focus of the class. We will continue as if we never chose the more complex model, and had picked the simpler, additive model

Let's just continue with the simpler, additive model

Model 3a - zero inflated Poisson

```
owls.glmmTMB3<-update(owls.glmmTMB1d, ziformula=~1) # ziformula indicates 0 inflation  
# the count model we used previously (owls.glmmTMB1d, the simpler model)  
models the count of calls against SexParent, FoodTreatment, etc  
# then there is the detection model (this model). All we are doing with  
this model is to account for detection. We are telling R that there is a  
detection issue with the ziformula part
```

This model says "model the imperfect detection" = Try and partition the 0s in what we expect and what we don't. "Tell me the rate of false detection"

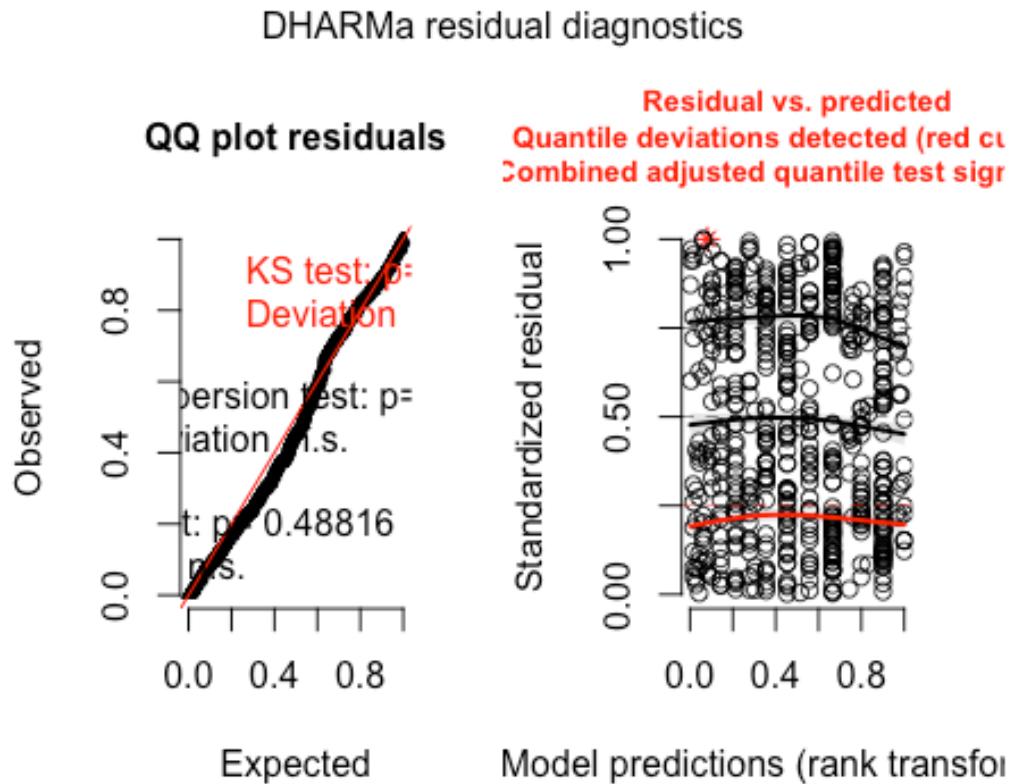
Model 4 - zero inflated Poisson with interactions

```
owls.glmmTMB4<-update(owls.glmmTMB1d, ziformula=~FoodTreatment*SexParent) # ziformula indicates 0 inflation
```

Checks whether the rate of false detection is related to Treatment and SexParent. Perhaps when the chicks are satiated they don't call as much, and the detection is less precise.

We now want to see if we have corrected the problem of zero inflation. We will do this with DHARMA residuals.

```
owls.resid=simulateResiduals(owls.glmmTMB4,plot=TRUE,integerResponse=TRUE)
```



residuals are better than the first time we tested them, but still not great.

Zero inflated negative binomial model

We will therefore try a zero inflated negative binomial:

```
owls.glmmTMB6<-glmmTMB(NCalls~FoodTreatment*SexParent+offset(log(BroodSize))+  
# don't forget the log  
                                (FoodTreatment+SexParent|Nest),  
                                ziformula=~FoodTreatment*SexParent,data=owls, #  
accounts for zero inflation  
                                family=nbinom2(link="log"),REML=TRUE)  
# REML=TRUE cause we are not comparing models with diff fixed effects  
# nbinom2 is how you fit a neg binomial (and the nr 2 is the one that's  
sensible for ecology)
```

The

Let's check which one is the better model that actually accounts for zero inflation

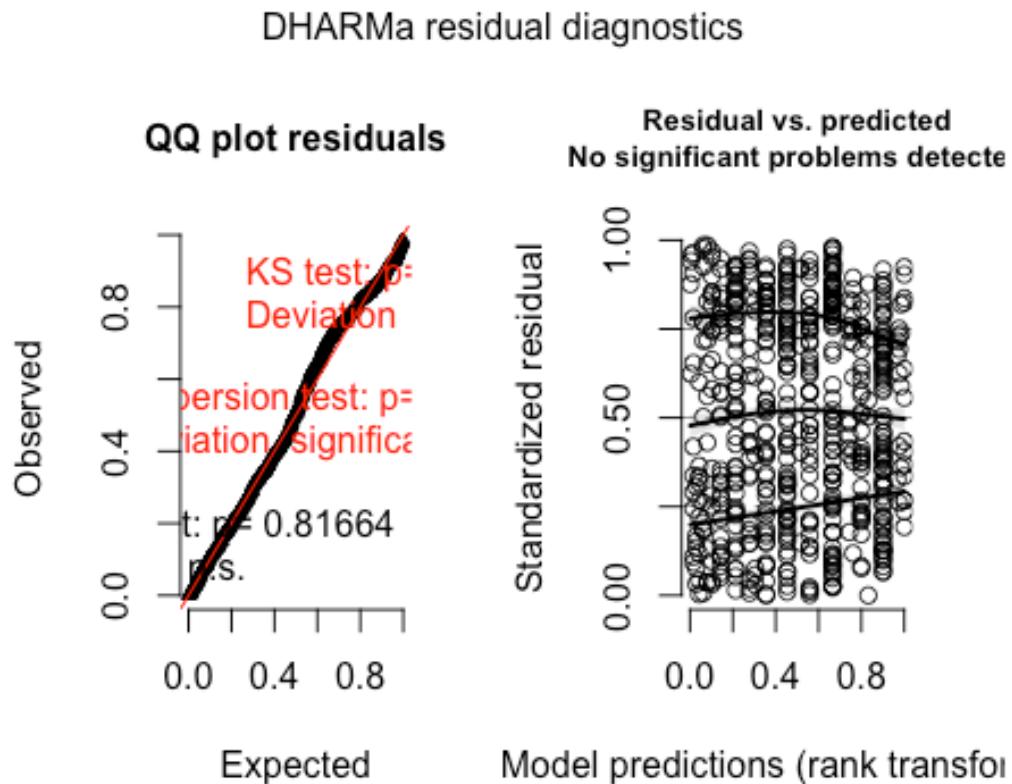
```
AICc(owls.glmmTMB3,owls.glmmTMB4,owls.glmmTMB6)
```

```
##          df      AICc
## owls.glmmTMB3 15 3873.651
## owls.glmmTMB4 18 3850.323
## owls.glmmTMB6 15 3373.525
```

The last model, the one that is a Zero inflated negative binomial model, is actually the better model

Let's check the DHARMA residuals

```
owls.resid=simulateResiduals(owls.glmmTMB6,plot=TRUE,integerResponse=TRUE)
```



DHARMA residuals are still not good...

What can we do now? - Look at the AICc and pick the one with slightly better test results -> take the Zero inflated Poisson (model 4) - Ignore the DHARMA tests as the graphs look pretty good -> take model 6

The

Let's go with model 6 (and fuck the DHARMA tests)

Model investigation / hypothesis testing

`summary(owls.glmmTMB6)`

```
## Family: nbinom2 ( log )
## Formula:
## NCalls ~ FoodTreatment * SexParent + offset(log(BroodSize)) +
##          (FoodTreatment + SexParent | Nest)
## Zero inflation: ~FoodTreatment * SexParent
## Data: owls
##
##      AIC      BIC    logLik deviance df.resid
##  3372.7  3438.6 -1671.4   3342.7      588
##
## Random effects:
##
## Conditional model:
## Groups Name             Variance Std.Dev. Corr
## Nest   (Intercept)     0.11739  0.3426
##        FoodTreatmentSatiated 1.24417  1.1154   0.18
##        SexParentMale       0.03551  0.1884  -0.66 -0.72
## Number of obs: 599, groups: Nest, 27
##
## Overdispersion parameter for nbinom2 family (): 2.72
##
## Conditional model:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 0.8367    0.1059   7.905 2.69e-15 ***
## FoodTreatmentSatiated      -0.8273    0.2691  -3.074  0.00211 **
## SexParentMale                -0.1027    0.1070  -0.959  0.33735
## FoodTreatmentSatiated:SexParentMale  0.1838    0.1742   1.055  0.29160
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Zero-inflation model:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                -1.5276    0.2580  -5.921 3.2e-09 ***
## FoodTreatmentSatiated      0.7200    0.3582   2.010  0.0444 *
## SexParentMale                -0.9590    0.4056  -2.364  0.0181 *
## FoodTreatmentSatiated:SexParentMale  0.8334    0.5177   1.610  0.1074
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This has been interpreted as a neg binomial model. Let's examine it:

Random effects: Conditional model: Estimate Std. Error z value Pr(>|z|)
(Intercept) 0.8367 = Nr of calls in the deprived female group, on a log scale

FoodTreatmentSatiated -0.8273 0.2691 -3.074 0.00211 **

=> Diff btw the deprived and satiated female group. When the group is satiated, there are significantly fewer calls (-0.8273 fewer calls in comparison to the Intercept, on a log scale)

FoodTreatmentSatiated:SexParentMale 0.1838 0.1742 1.055 0.29160

=> No evidence of interaction btw SexParent and FoodTreatment

SexParentMale -0.1027 0.1070 -0.959 0.33735

=> No important variable (ns p value)

Zero-inflation model: Estimate Std. Error z value Pr(>|z|)

(Intercept) -1.5276 0.2580 -5.921 3.2e-09 *** => Detectability of calls for the deprived female group

```
exp(-1.5276) # Intercept
```

```
## [1] 0.217056
```

```
plogis(-1.5276) # Intercept
```

```
## [1] 0.1783451
```

In a female deprived group, the prob of a zero being false is 1-0.18 => 80% of zeros in that group are false zeros (!)

And:

```
exp(0.72) # sorry, missed where he got the number from
```

```
## [1] 2.054433
```

We are more likely to miss zeros in the satiated group (twice as likely)

```
exp(-0.959) # sorry, missed where he got the number from
```

```
## [1] 0.383276
```

We are less likely to miss a call if the returning animal is a male. The calls are more obvious when the males are returning.

R square for zero inflated models

```
performance::r2_nakagawa(owls.glmmTMB6)
```

```
## Warning in fitTMB(TMBStruc): Model convergence problem; non-positive-
```

```
definite
```

```
## Hessian matrix. See vignette('troubleshooting')
```

```
## Warning in fitTMB(TMBStruc): Model convergence problem; false convergence
(8).
## See vignette('troubleshooting')

## # R2 for Mixed Models
##
## Conditional R2: 0.608
## Marginal R2: 0.101
```

The error tells us that the calculation made some assumptions which we cannot trust. HOWEVER, the output R sq can still be trusted as an estimation.

- Conditional R2: The fixed effects of our model do not explain a great deal, only around 10%.
- Marginal R2: With random effects included, we explain around 60%. Random effects therefore explain around 60-10=50%

Predictions

We did not do this, but Murray says there is a .code hidden folder in SUYR_public.Rproj where we can access ALL THE CODE and explanations.

Predictions

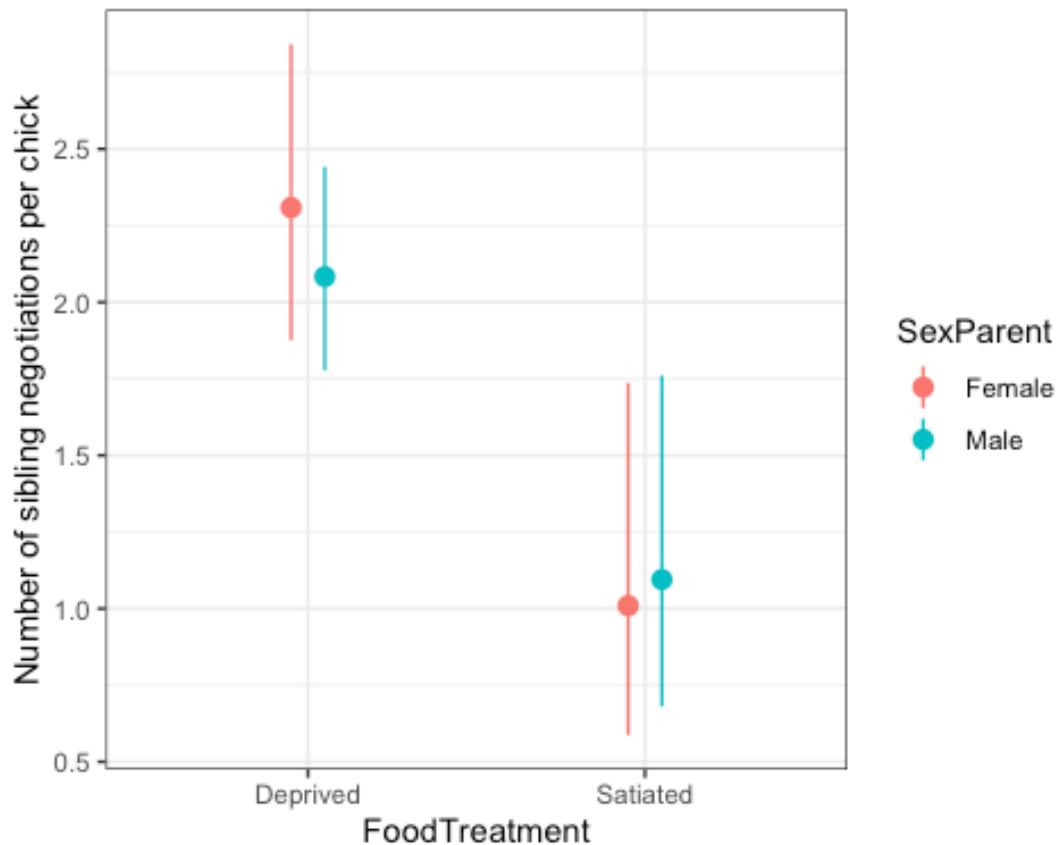
```
owls.grid = with(owls, list(FoodTreatment=levels(FoodTreatment),
                           SexParent=levels(SexParent)))
newdata = emmeans(owls.glmmTMB6, ~FoodTreatment+SexParent, at=owls.grid,
                  offset=0, type='response') %>%
  as.data.frame
head(newdata)

##   FoodTreatment SexParent response      SE df lower.CL upper.CL
## 1     Deprived    Female  2.308829 0.2443986 588 1.8754393 2.842371
## 2    Satiated    Female  1.009514 0.2786735 588 0.5870245 1.736077
## 3     Deprived     Male  2.083487 0.1685550 588 1.7774040 2.442280
## 4    Satiated     Male  1.094754 0.2649406 588 0.6806016 1.760922

# If we had chosen model4, we'd have "rate" instead of "response" (but it is
# the same)
```

Summary figures

```
ggplot(newdata, aes(y=response, x=FoodTreatment)) +
  geom_pointrange(aes(ymin=lower.CL, ymax=upper.CL, color=SexParent),
                  position=position_dodge(width=0.2)) +
  scale_y_continuous('Number of sibling negotiations per chick') +
  theme_bw()
```

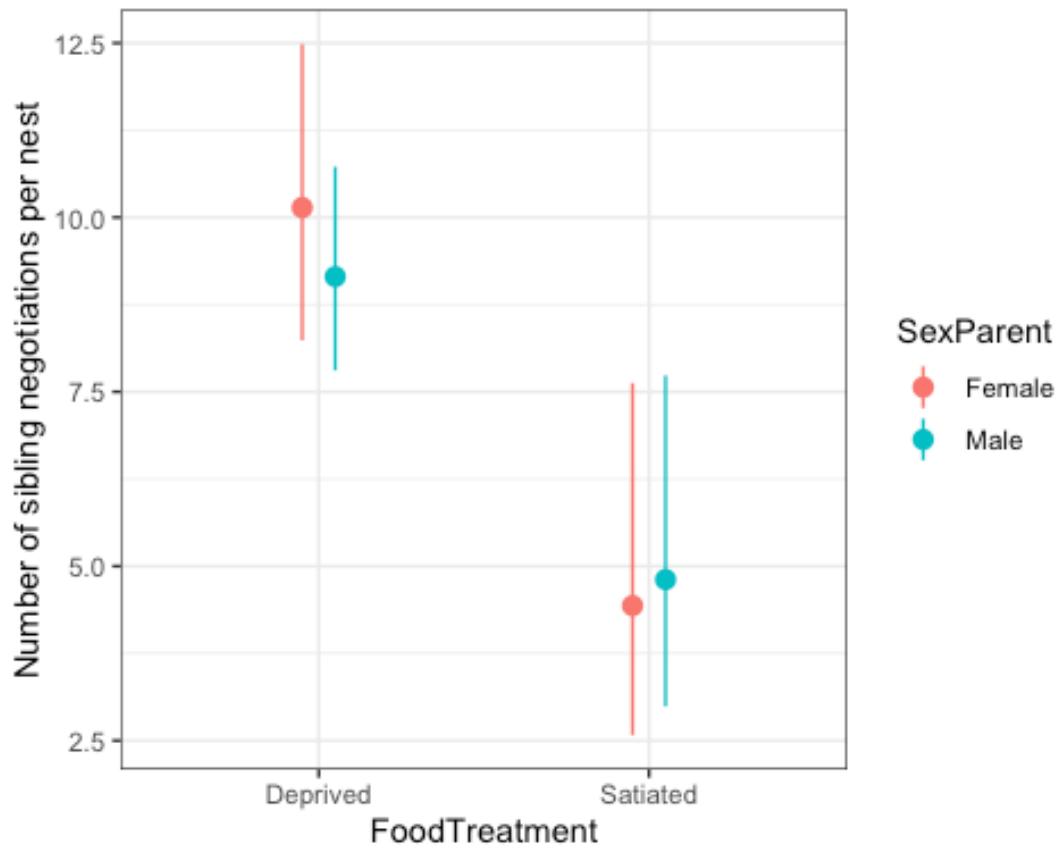


Alternative figure

```
##OR if we want to express this for the average brood size
owls.grid = with(owls, list(FoodTreatment=levels(FoodTreatment),
                           SexParent=levels(SexParent)))
newdata = emmeans(owls.glmmTMB6, ~FoodTreatment+SexParent, at=owls.grid,
                  offset=log(mean(owls$BroodSize)), type='response') %>%
  as.data.frame
head(newdata)

##   FoodTreatment SexParent  response           SE  df lower.CL upper.CL
## 1      Deprived    Female 10.141119 1.0734769 588 8.237530 12.484602
## 2      Satiated    Female  4.434111 1.2240233 588 2.578400  7.625405
## 3      Deprived     Male  9.151342 0.7403475 588 7.806928 10.727275
## 4      Satiated     Male  4.808510 1.1637038 588 2.989421  7.734532

ggplot(newdata, aes(y=response, x=FoodTreatment)) + # If we had chosen
model4, we'd have "rate" instead of "response" (but it is the same)
  geom_pointrange(aes(ymin=lower.CL, ymax=upper.CL, color=SexParent),
                  position=position_dodge(width=0.2)) +
  scale_y_continuous('Number of sibling negotiations per nest') +
  theme_bw()
```



Alternative model - Herdle model Murray suggests to have used a Herdle model instead of model 6. A Herdle model splits it into presence/abscence, and counts (if it is present). It asks only two questions. It is used for negative binomial distributions

IGNORE BELOW - We did not go through this

plot_model (Murray had used model 4 as best model, so if you wanted to do the following with the last model, use model 6 instead)

These predictions appear to be based on the mean BroodSize of approximately 4.39.

allEffects

These predictions also appear to be based on the mean BroodSize, although the documentation seems to suggest that `allEffects()` might not deal with the offsets the way we have used them (as a function in the formula) correctly.

ggpredict

This seems to deal with the offset incorrectly. For the purpose of prediction, the offset seems to be set at the value of the first BroodSize (on the response scale). This is incorrect for two reasons:

1. it should be on the log scale
2. it would be better to use either the mean BroodSize (then on the link scale) or a value of 0 (so as to reflect the per unit BroodSize prediction).

ggemmeans

ggemmeans() can accommodate the offset correctly. There are two sensible choices:

- set the offset to the (log) of the mean BroodSize (similar to other partial effects), hence giving predictions appropriate for the average brood size conclusion.
- set the offset to 0. This results in predictions appropriate for a per owl chick conclusion.

Model investigation / hypothesis testing

Predictions

Summary figures

References

11_GAM_Gaussian

Sara Kophamel

14/12/2020

GENERALISED ADDITIVE MODELS (GAM)

Presentation:

file:///Users/sara/Documents/PhD/HELP/STATISTICS%20and%20GRAPHS/-%20R%20and%20STATS%20COURSES/2020%20Murray%20Logan%20R%20Course/Logan_Course/docs/pres.8.html#1

THEORY

GAM do not assume linearity. They are for models with non-linear shapes (\neq linear models). When we get more complicated shapes, we are not trying to extract what the slope is. We want to draw the relationship so that we can describe it.

Examples of non-linear models: - Logarithmic model $\rightarrow y=b^x$ - Polynomial models allow for a straight line, or a curved line, and are within a linear model context. Murray does not recommend going over a third order polynomial (more curvy), as it might not explain the real distribution of the data anymore. - Piecewise regression: Fitting TWO models through ONE line. This helps to find abrupt changes. Often, because it is fitting two separate models, we also have two sep pred at the value where the curve changes, at the value of 4.2 (see slide 8). This model would force the slope and curvature at the meeting point, blending the two curves together, forcing it to have the same trend / rate of curvature. That is, piecewise regressions must have the same 1st and 2nd order derivatives.

Piecewise regression - KNOTS and SPLINES - Knots: The KNOTS define how many models we are fitting (one at the start and one at the end). One model fits btw two nods. On slide 9, we got three nods, which is equivalent to having two models. - Splines (slide 10): Top left: A basis function for a simple regression has a single line, and we only need to know the slope of that line. We also got an intercept. Top right: We ask what the value of the intercept, and the slope is. The slope is determined to be a negative slope. The resulting line is the sum of the slope and the intercept. As you keep moving along the line, you end up with the predicted trend. For these sorts of models, we are only trying to describe the trends. When we get more complicated shapes, we are not trying to extract what the slope is. We want to draw the relationship so that we can describe it.

What a spline does, is, it takes your data, and depending on how many nods and orders of a polynomial, it converts the dataset to be appropriate to fit the model.

A first order polynomial fits two knots (one model), so it would have TWO columns assigned to it. A second order polynomial fits three knots (two models, nods in our example

are at x values 0, 4.2 and 9.4), so it would have THREE columns assigned to it. When you add all intercepts to the model, you end up with the piecewise regression (black line).

Slides 11-12: - If we had two separated polynomials being fit, that met in the middle and blend together, we'd have the top right model. - A first(?) order polynomial (bottom left). - A third order polynomial: bottom right We don't want to go above a third order polynomial

Slide 14: Depending on how many knots, we would get a wobblier line. We do NOT want to reconstruct the perfect equation, but to create a model that describes the data well, without overfitting (USUALLY this is the case of models >3 knots). Models with heaps of knots overfit the data, as they try to reconstruct it, which we don't want it to do. Models with too few knots underfit the data.

How do we know if we have an overfit or an underfit? (Slide 15) - With Balance = reduction of bias / prediction error, and by creating a good explanation of the model with RSS. - With Cross-validation = Withholding some data and seeing how well it is predicted. Each point is left out as a way of doing this. Another option is to consider the number of dots as a parameter of the model, and estimate it. However, this has been shown to not be very accurate, and is not used anymore.

All of these models are estimated by a Maximal Likelihood Model, so we can use REML to estimate the wigglyness.

Generalised additive models (GAMs) -> slide 17

We need to weight each of the curves / knots and sum them up. Then, R will penalise for overcomplexity and will define how many knots actually have to be used (slide 18). A GAM fits lines and works out the ideal number of knots.

In this example, the graph shows a cubic regression, which has heaps of third order polynomials. But there are other models (slide 19): - Thin plate spline: Does not need all the data to work out how many knots it needs. It is convenient for large datasets. - Penalised spline: Will apply more knots where there is more data, so it does not mind if the data is not uniformly distributed across the x axis. - Cubic regression spline: Data / knots are evenly spaced. The data is uniformly distributed across the x axis. - Cyclic cubic regression line: Cubic regression lines with ends that meet (as they have the same 2nd order derivatives). They are also very useful. - Spline for random effects (slide 20): The random effects have their own spline, which handles the dependency structure. These are special ones for interactions: Seems to contradict on the surface what a GAM is (in theory it is an ADDITIVE model, without interactions). BUT you can actually fit interactions :-)

PRACTICE

Open gam_example1.Rmd

Preparations: Packages

Load the necessary libraries

```
library(mgcv)      #for GAMs. "mgcv = multiple generalised cross validation".  
Since we have decided that cross validation is not the best approach, it was  
updated to GAMs  
library(broom)      #for tidy results  
library(gratia)     #for GAM plots  
library(randomForest) # for performance / checking the GAM model  
library(DHARMa)      #for residual diagnostics  
library(performance) #for residuals diagnostics  
library(see)         #for plotting residuals  
library(emmeans)     #for marginal means etc  
library(MuMIn)       #for model selection and AICc  
library(tidyverse)    #for data wrangling
```

Scenario

This is an entirely fabricated example (how embarrassing). So here is a picture of some Red Grouse Chicks to compensate..



Red grouse chicks

Format of data.gp.csv data file

x	y
2	3
4	5
8	6
10	7
14	4

x - a continuous predictor

y - a continuous response

Read in the data

```
data_gam = read_csv('data/data_gam.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   x = col_double(),
##   y = col_double()
## )

glimpse(data_gam)

## Rows: 5
## Columns: 2
## $ x <dbl> 2, 4, 8, 10, 14
## $ y <dbl> 3, 5, 6, 7, 4
```

Exploratory data analysis

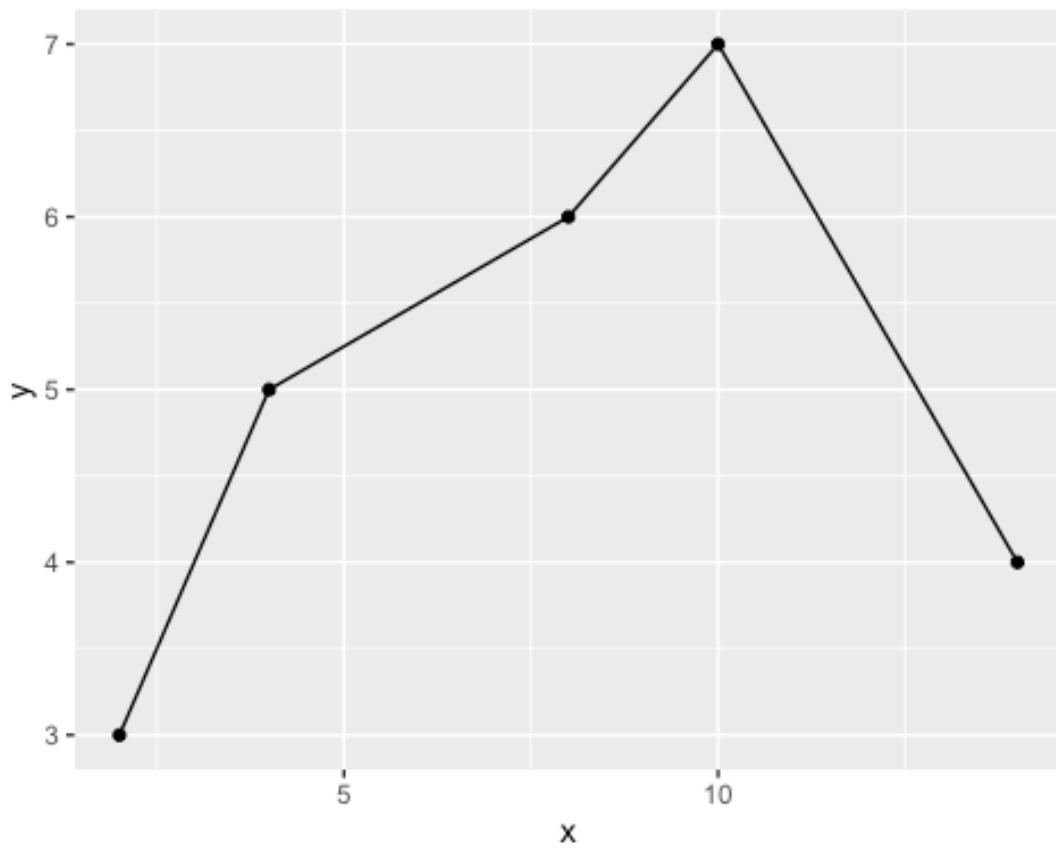
Model formula:

$$\$ y_i \sim N(\mu_i, \sigma^2) \quad \mu_i = \beta_0 + f(x_i) \quad f(x_i) = \sum_{j=1}^k b_j(x_i) \beta_j$$

where β_0 is the y-intercept, and $f(x)$ indicates an additive smoothing function of x .

Exploratory plot

```
ggplot(data_gam, aes(y=y, x=x)) +
  geom_point() +
  geom_line()
```



What we see is that our data is NOT linear. Had we used a smoother, we'd have also noticed that the trend is not linear

```
ggplot(data_gam, aes(y=y, x=x))+
  geom_point()+
  geom_smooth()

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

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : span too small. fewer data values than degrees of freedom.

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : pseudoinverse used at 1.94

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : neighborhood radius 6.06

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : There are other near singularities as well. 36.724
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 5

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 5

## Warning in sqrt(sum.squares/one.delta): NaNs produced

## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object))), : span too small.
fewer
## data values than degrees of freedom.

## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object))), : pseudoinverse used
at
## 1.94

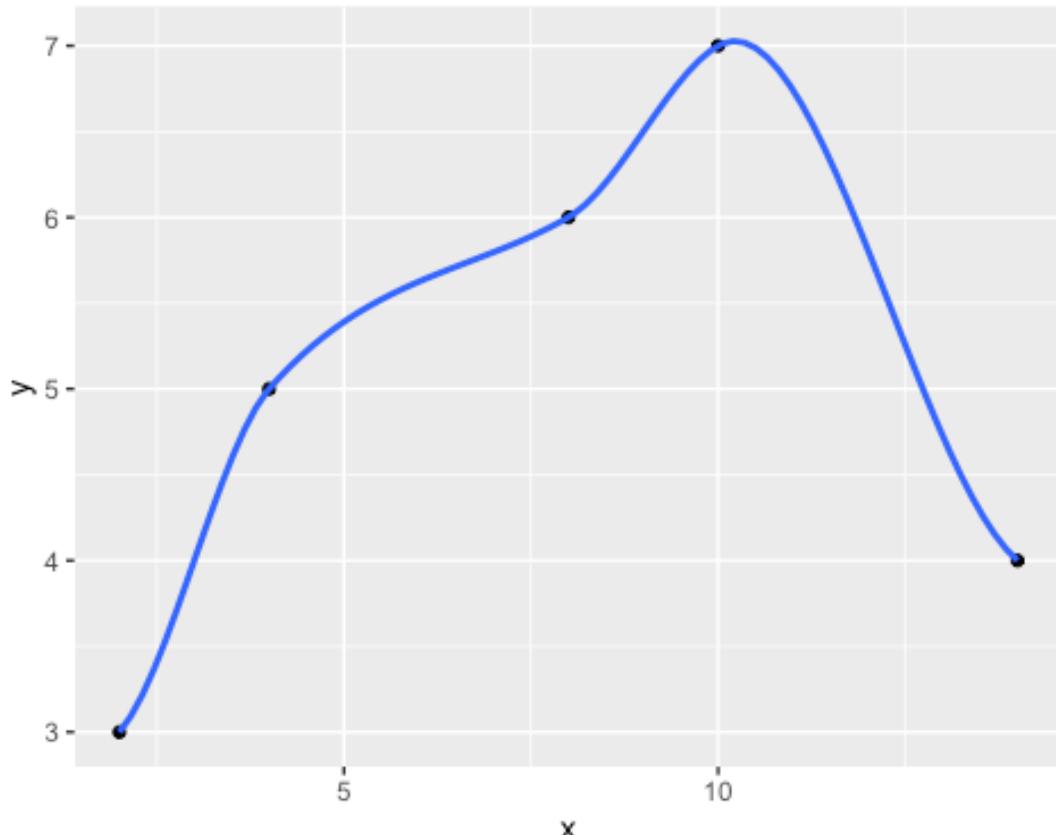
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object))), : neighborhood
radius 6.06

## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object))), : reciprocal
condition
## number 0

## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object))), : There are other
near
## singularities as well. 36.724

## Warning in stats::qt(level/2 + 0.5, pred$df): NaNs produced

## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max;
returning -
## Inf
```



A lower smoother, rather than fitting one regression, will fit several through the datapoints. This is the lowest smoother, which already indicates that the data is not linear.

Fit the model

What if we fitted a polynomial?

```
data_gam.lm<-lm(y~x+I(x^2)+I(x^3), data=data_gam)
```

The formula here is the most directly relatable to the way we might write a polynomial. We have always got a y intercept, but we also got x sq and x rooted. This would be the direct translation, but it is NOT useful. That is because, when we summarise the model, ...

```
summary(data_gam.lm)

##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3), data = data_gam)
##
## Residuals:
##      1       2       3       4       5 
## -0.13784  0.33083 -0.55138  0.41353 -0.05514
##
## Coefficients:
```

```

##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.423559  2.798574  0.509   0.700
## x           0.876566  1.524581  0.575   0.668
## I(x^2)     -0.003133  0.215476 -0.015   0.991
## I(x^3)     -0.003289  0.008729 -0.377   0.771
##
## Residual standard error: 0.7788 on 1 degrees of freedom
## Multiple R-squared:  0.9393, Adjusted R-squared:  0.7574
## F-statistic: 5.163 on 3 and 1 DF,  p-value: 0.3104

```

... we get the intercept along with the partial slopes for each of these. BUT they are NOT independent (!) They allow us to repopulate the equation, but they don't make ANY conclusions about the hypothesis test. They don't mean anything AT ALL.

A BETTER way of fitting a polynomial model is to do the following:

```

data_gam.lm1<-lm(y~poly(x,3),data=data_gam)
# x,3 = defines however many orders you want.

```

In this case, up to a third order polynomial. In this case, the three terms are INDEPENDENT of one another. Therefore, we can ask: "Is there a linear, quadratic or cubic component to this trend?"

```

summary(data_gam.lm1)

##
## Call:
## lm(formula = y ~ poly(x, 3), data = data_gam)
##
## Residuals:
##      1       2       3       4       5 
## -0.13784  0.33083 -0.55138  0.41353 -0.05514
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 5.0000    0.3483 14.356   0.0443 *  
## poly(x, 3)1 1.0471    0.7788  1.345   0.4071    
## poly(x, 3)2 -2.8655    0.7788 -3.679   0.1689    
## poly(x, 3)3 -0.2935    0.7788 -0.377   0.7706    
## ---    
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7788 on 1 degrees of freedom
## Multiple R-squared:  0.9393, Adjusted R-squared:  0.7574
## F-statistic: 5.163 on 3 and 1 DF,  p-value: 0.3104

```

The stats show that there is a significant optimum value of the model.

```

data.frame(smoothCon(s(x,k=3),data=data_gam)[[1]]$X) %>%
  bind_cols(data_gam)

```

```
##          X1   X2          X3   x   y
## 1  1.3554342  1 -1.31122014  2  3
## 2  0.9289363  1 -0.84292723  4  5
## 3  0.4755086  1  0.09365858  8  6
## 4  0.5780165  1  0.56195149 10  7
## 5  1.3189632  1  1.49853730 14  4
```

If we had nominated three knots for our data, we would end up with three columns

```
# data_gam.gam<-mgcv::gam(y~s(x),data=data_gam) # s indicates the smoother to
# fit basic functions like the splines (s is NOT a variable in the data)
```

It wants to try and fit ten knots to the data. We will have to reduce the number of knots:

```
data_gam.gam<-gam(y~s(x, k=3),data=data_gam, method="REML")
# s indicates the smoother to fit basic functions like the splines
# method = REML makes sure our model is correct
```

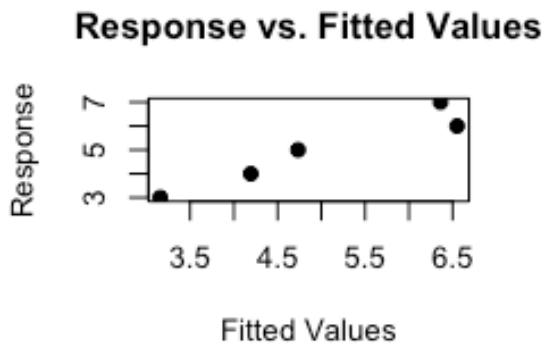
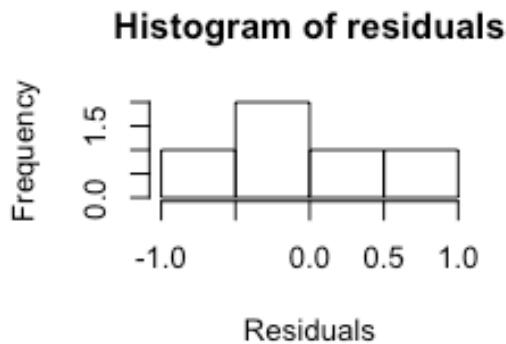
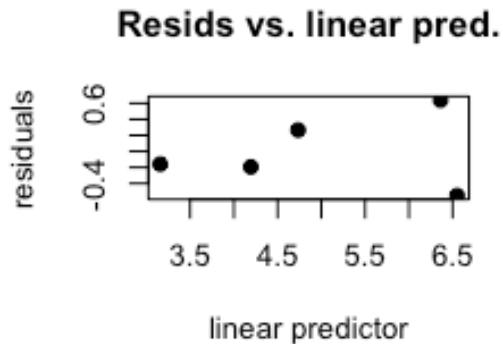
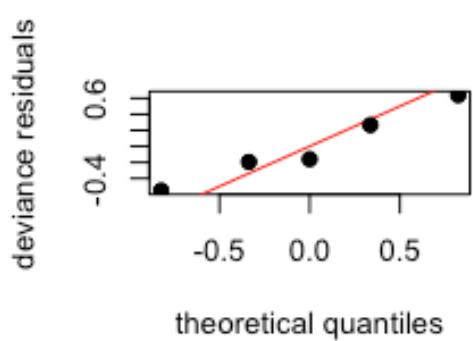
Model validation

The following packages can be used to assess our model:

Option 1: gamcheck

The package “gamcheck” checks if we fitted the correct GAM

```
gam.check(data_gam.gam, pch=19)
```



```
##
## Method: REML   Optimizer: outer newton
## full convergence after 5 iterations.
## Gradient range [-4.151024e-06,2.176788e-06]
## (score 6.024247 & scale 0.4120262).
## Hessian positive definite, eigenvalue range [0.2676093,1.6827].
## Model rank = 3 / 3
##
## 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(x) 2.00 1.95    2.19    0.98
```

This package... - Produces a statistical output (see copy pasted below) - Produces a series of figures

Output: All of these metrics are combined, you cannot explain them in isolation. k' edf k -index p -value $s(x)$ 2.00 1.95 2.19 0.97 - k' prime (k') is the number of knots minus 1 = 2 knots. It is equivalent to the df . The lower the value is <1, the more likely it is we MISSED some real patterns = we have (probably) OVERSMOOTHED our model. - edf is the estimated df after penalising for wigginess, so 1.95 (not 2). $df \neq \text{integer}$ is not common, but this is only because of penalising for complexity. We want edf to be a LOT lower than k

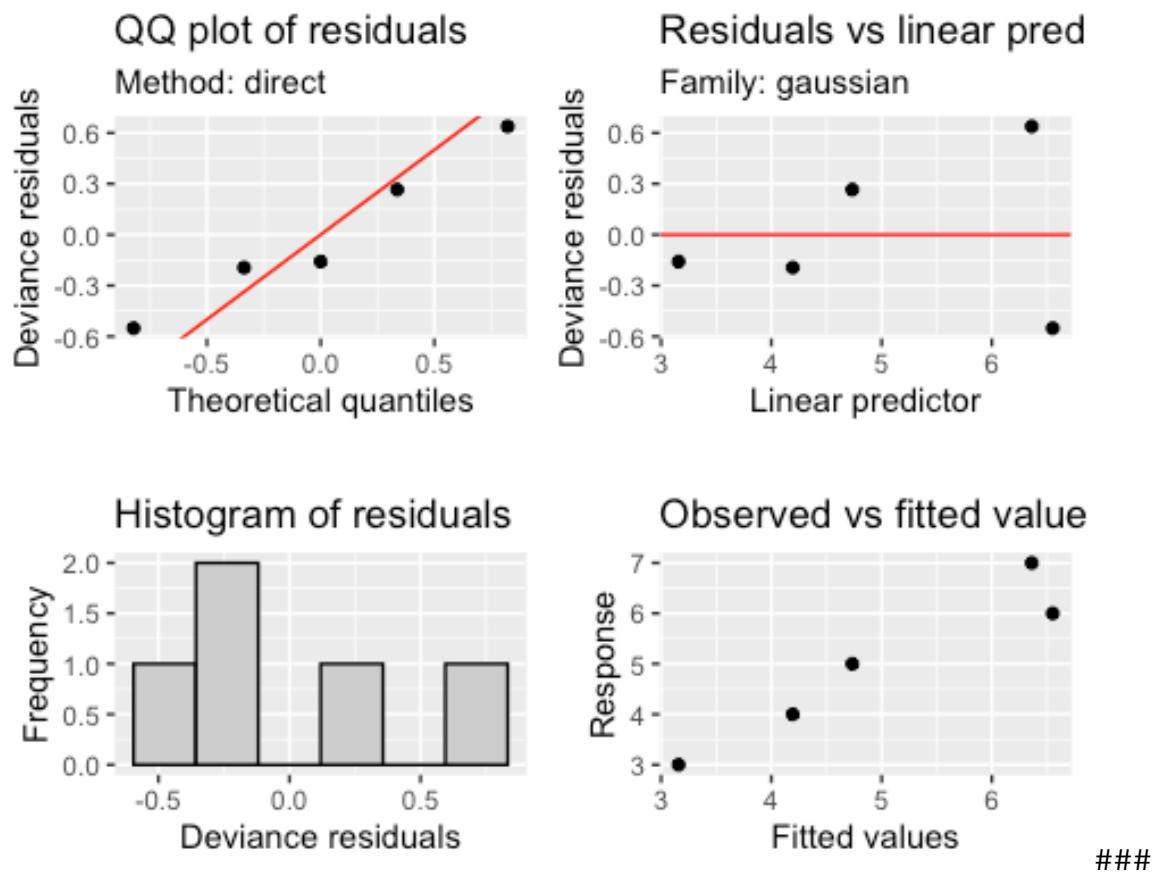
prime. 1.95 is not great, but our dataset is also really small, so we can't expect much more. edf will never be higher than k' If edf is close to k' (lets say edf=1.95) AND the k-index is <1, the optimum smoothing is possibly around 3 knots, and we have CONSTRAINED the model too much. - k-index: Has to be >1. If k-index <1 AND edf close to k', then we have constrained the data. - p-value: In this case, it is not significant. You want it to be non significant. It indicates if we got a significantly overconstrained k, which is not the case here.

The p value indicates that, whenever we fit our GAM, whether we had to set k or not, we need to look at these diagnostics. R will always set k to 10 by default, which might be too much in some cases. If we get a sign p value, and we think that we over- or underfitted our data based on the output above, we will have to define another number of knots.

Figures: - Response vs Fitted values: Pred values are on x and obs. on y. We want it to have a straight line.

Option 2: appraise (same as gamcheck, but for plots, package gratia)

```
appraise(data_gam.gam)
```



Option 3: k.check (alternative checking package, not sure which one?)

```
k.check(data_gam.gam)
```

```
##      k'      edf k-index p-value
## s(x) 2 1.949003 2.186796 0.9825
```

Shows the output that gamcheck also produced

Checking the type of model

```
performance::check_distribution(data_gam.gam)

## # Distribution of Model Family
##
## Predicted Distribution of Residuals
##
## Distribution Probability
##     normal        44%
##     beta          31%
##     exponential    9%
##
## Predicted Distribution of Response
##
## Distribution Probability
##     binomial       41%
##     beta-binomial  22%
##     uniform        22%
```

Tells you which should be the best model distribution, eg. normal, binomial, etc. (Murray does not really recommend using it, but many researchers find it comforting)

Option 4: DHARMa residuals

```
resids<-DHARMa::simulateResiduals(data_gam.gam, plot=T)

## Warning in gam.fit4(x, y, sp, Eb, UrS = UrS, weights = weights, start =
## start, :
## Non-finite coefficients at iteration 4

## Unable to calculate quantile regression for quantile 0.25. Possibly to few
## (unique) data points / predictions. Will be omitted in plots and significance
## calculations.

## Unable to calculate quantile regression for quantile 0.5. Possibly to few
## (unique) data points / predictions. Will be omitted in plots and significance
## calculations.

## Warning in gam.fit4(x, y, sp, Eb, UrS = UrS, weights = weights, start =
## start, :
## Non-finite coefficients at iteration 16

## Warning in gam.fit4(x, y, sp, Eb, UrS = UrS, weights = weights, start =
## start, :
## Non-finite coefficients at iteration 16

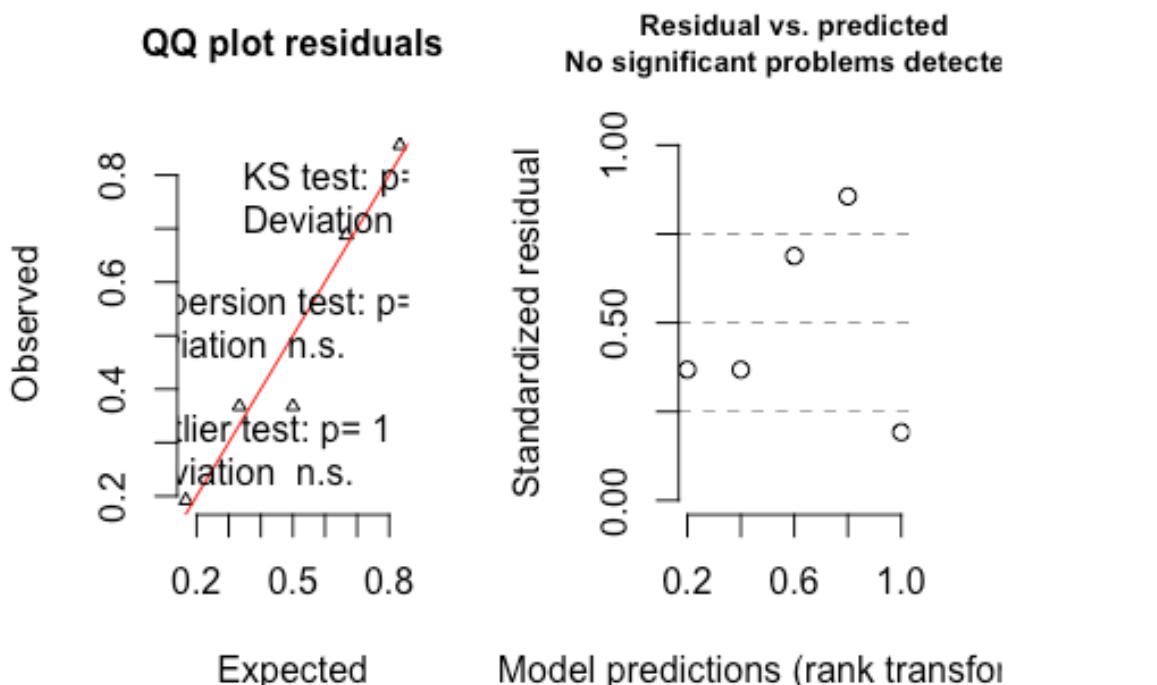
## Warning in gam.fit4(x, y, sp, Eb, UrS = UrS, weights = weights, start =
```

```

start, :
## Non-finite coefficients at iteration 16
## qu = 0.75, log(sigma) = -2.364669 : outer Newton did not converge fully.
## qu = 0.75, log(sigma) = -2.660058 : outer Newton did not converge fully.
## qu = 0.75, log(sigma) = -3.173699 : outer Newton did not converge fully.
## qu = 0.75, log(sigma) = -3.476143 : outer Newton did not converge fully.
## qu = 0.75, log(sigma) = -3.478656 : outer Newton did not converge fully.
## qu = 0.75, log(sigma) = -3.480209 : outer Newton did not converge fully.
## We had to increase `err` for some of the quantiles. See fit$calibr$err
## Unable to calculate quantile regression for quantile 0.75. Possibly to few
(unique) data points / predictions. Will be ommited in plots and significance
calculations.

```

DHARMA residual diagnostics



(Ignore the warning messages because we got a small dataset; none of them are concerning for diagnostics. Worry about warning messages when FITTING models)

The figures show... - QQ plot on left + tests - Residuals plot on right Everything seems ok.

Checking for Concurvity

```
concurvity(data_gam.gam)
```

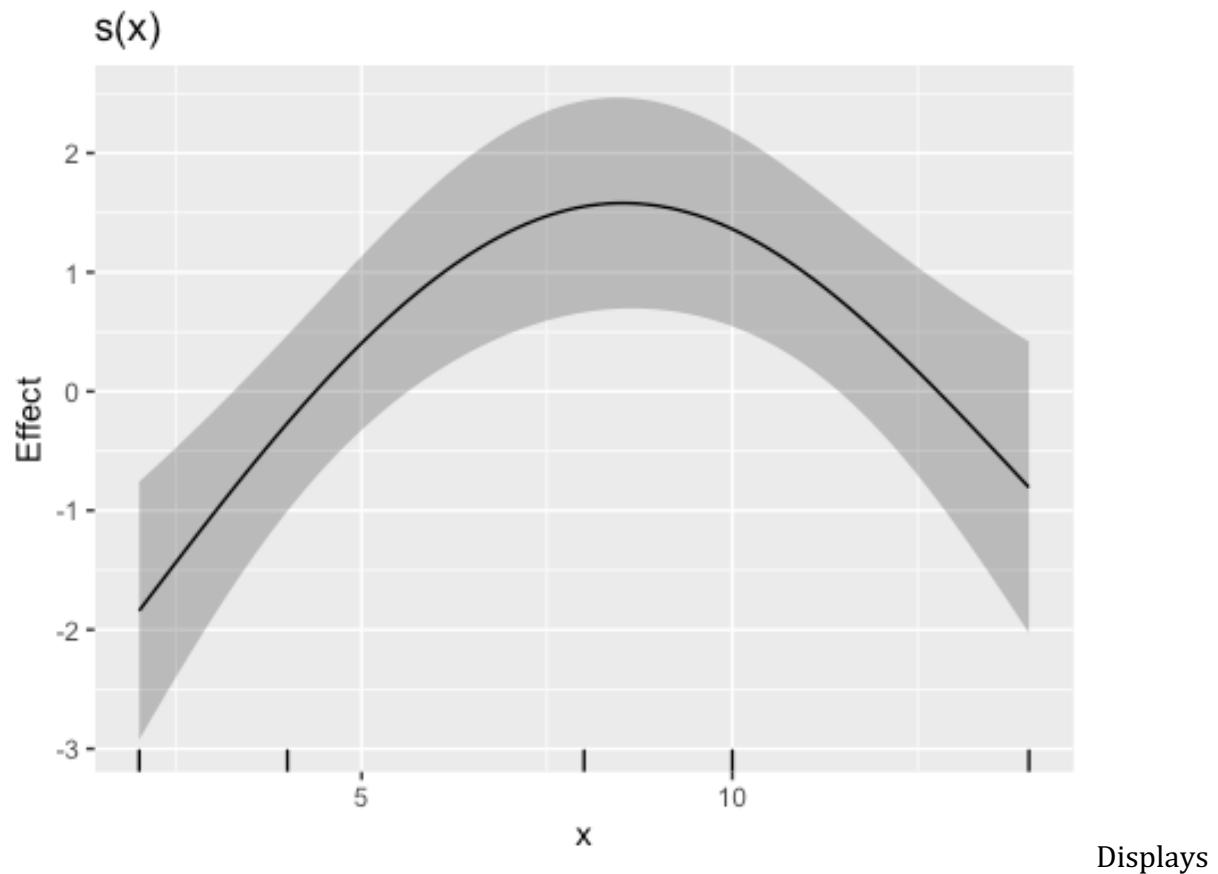
```
##          para      s(x)
## worst  3.194887e-30 3.487872e-30
## observed 3.194887e-30 3.148783e-30
## estimate 3.194887e-30 2.393887e-31
```

When we have multiple predictors, we can have collinearity. GAMs have smoothers for one predictor, but not for the other predictor. In a multiple regr, we can't have multiple pred, as one will compete against the other, and the patterns won't be true. If we have only one predictor, it does not make sense using this package, but it is useful when we got two or more predictors

It provides case scenarios, a worst case (ignore), and observed (mah, not too important either) and an estimated (look into this one). Murray will explain them in another example, when they actually say sth meaningful.

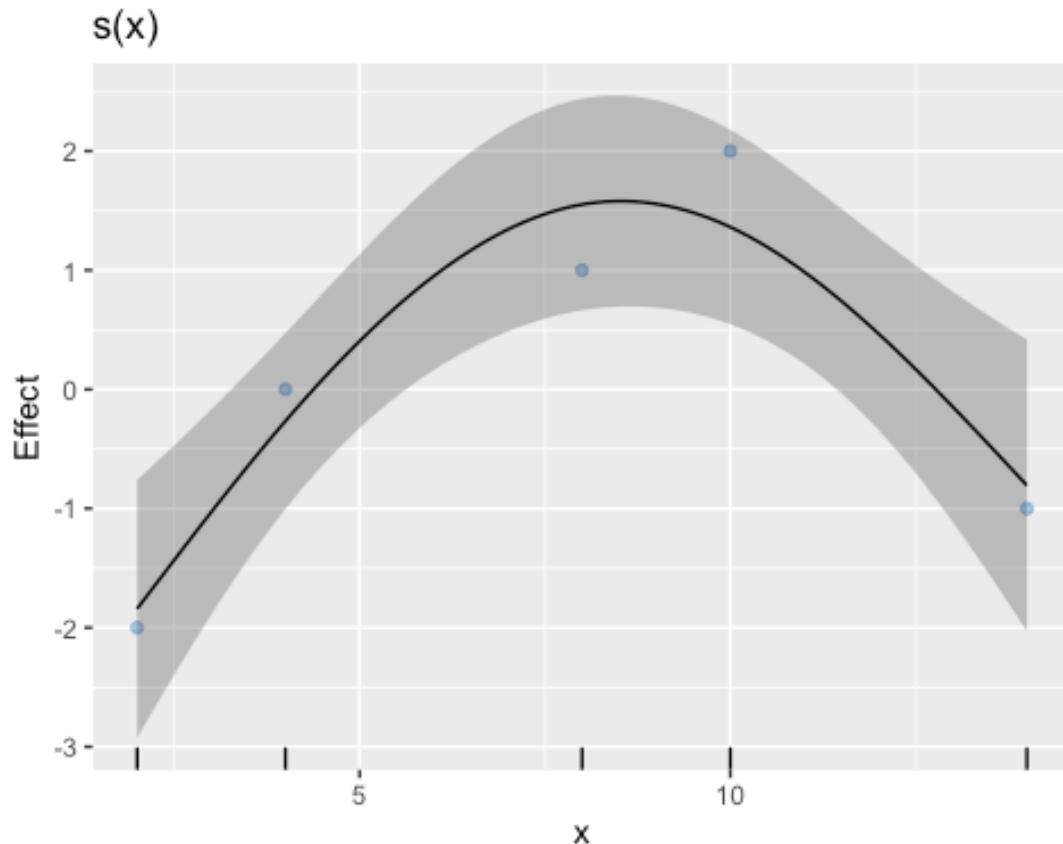
Partial plots

```
draw(data_gam.gam)
```



the trend of our model. We can even put the data back on, adding residuals=TRUE (=the raw data)

```
draw(data_gam.gam, residuals=TRUE)
```



The

problem with this plot is that it is centered at 0 (y-axis). This is an intentional decision for GAMs. R does that because we can talk about the trend (goes up and down), but in the presence of other predictors, you can't say what the absolute is. That is why R shows the centered trend instead. If we changed the y axis to an absolute value, it won't give us an estimate.

Model investigation / hypothesis testing

```
summary(data_gam.gam)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x, k = 3)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 5.0000    0.2871   17.42  0.00293 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
## 
## Approximate significance of smooth terms:
##          edf Ref.df      F p-value
## s(x) 1.949 1.997 10.88 0.0826 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## R-sq.(adj) =  0.835  Deviance explained = 91.5%
## -REML = 6.0242  Scale est. = 0.41203 n = 5
```

The output is split into two: - Parametric terms: This is the regular regression stuff. Here, we can see what the intercept is. We could actually fit a LR with GAM, but you'd be just fitting a GLM. We can't draw any real interpretation from this example. - Smooth terms:

Approximate significance of smooth terms: edf Ref.df F p-value

s(x) 1.949 1.997 10.88 0.0826 . - edf = estimated df. The higher the number, the higher the wigginess - Ref df = Ignore this one, as it is sort of a throwback and does not have any meaning. The author of this package would say that it means nothing (was used about 10 yrs ago) - F-statistic - associated p value = Does the line you have fit differ sign from a straight, flat line? = Is it sign wiggly? In this case, it is not. In our case, it could be a straight line. However, this might be because of the low power (sample size related) our example has. - Measures of the explained variance: R-sq.(adj) = 0.835 Deviance explained = 91.5% - R-sq(adj) = Adjusted R sq, calculated the same way that you calculate an R sq, but adjusted for sample size. Murray recommends NOT using it, as an adj R sq does NOT indicate the variance explained, as it is penalised by sample size. It is gonna be LESS than the amount your model explains. They used it to compare models, but it's outdated now. - INSTEAD, use the pseudo-R sq, the "Deviance explained". It is the (Dev / Null Dev)-1. In our case, 91% of the variance is explained by the model (this is a lot).

Table output (tidy)

```
tidy(data_gam.gam)

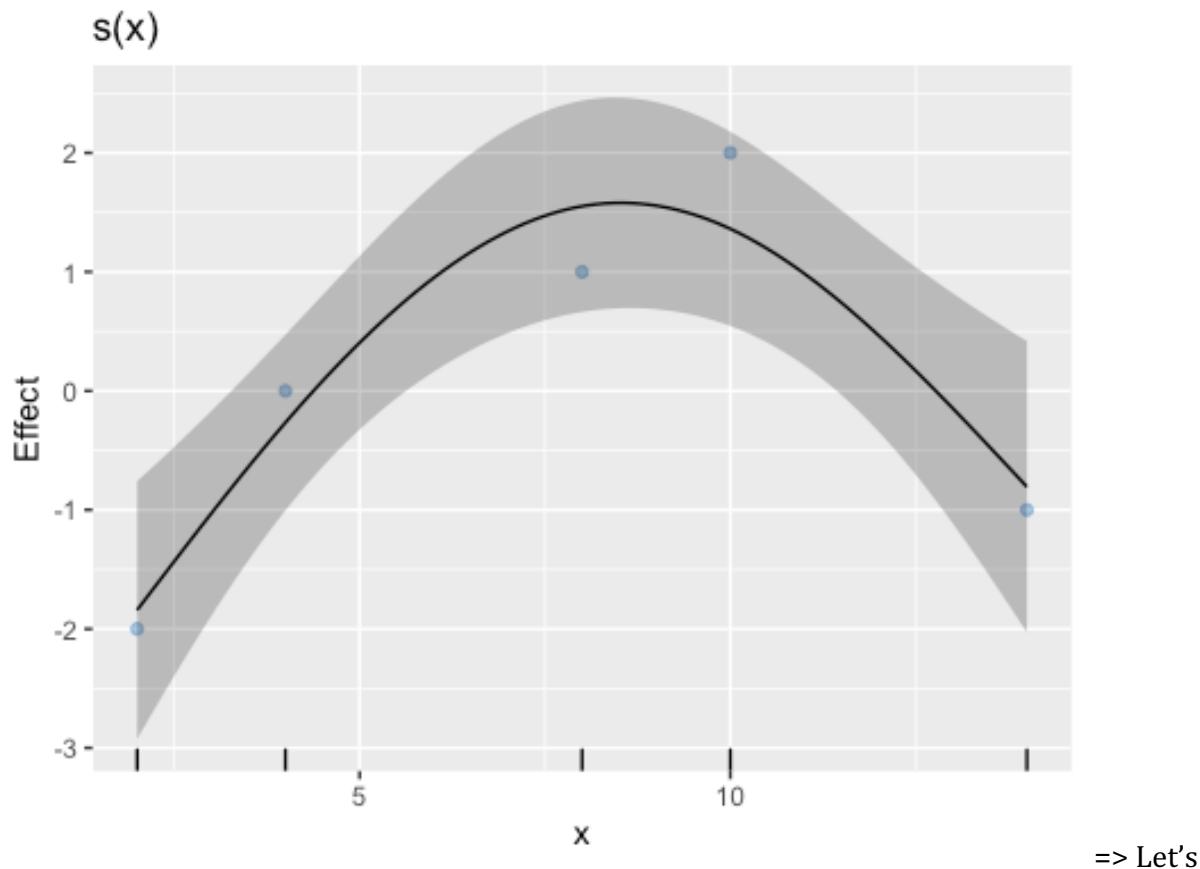
## # A tibble: 1 x 5
##   term    edf ref.df statistic p.value
##   <chr>  <dbl>  <dbl>     <dbl>    <dbl>
## 1 s(x)    1.95   2.00     10.9    0.0826
```

This output is easier to compare to other models, and much more useful for tabulating (creating tables for publication)

Further analyses

Checking the optimum

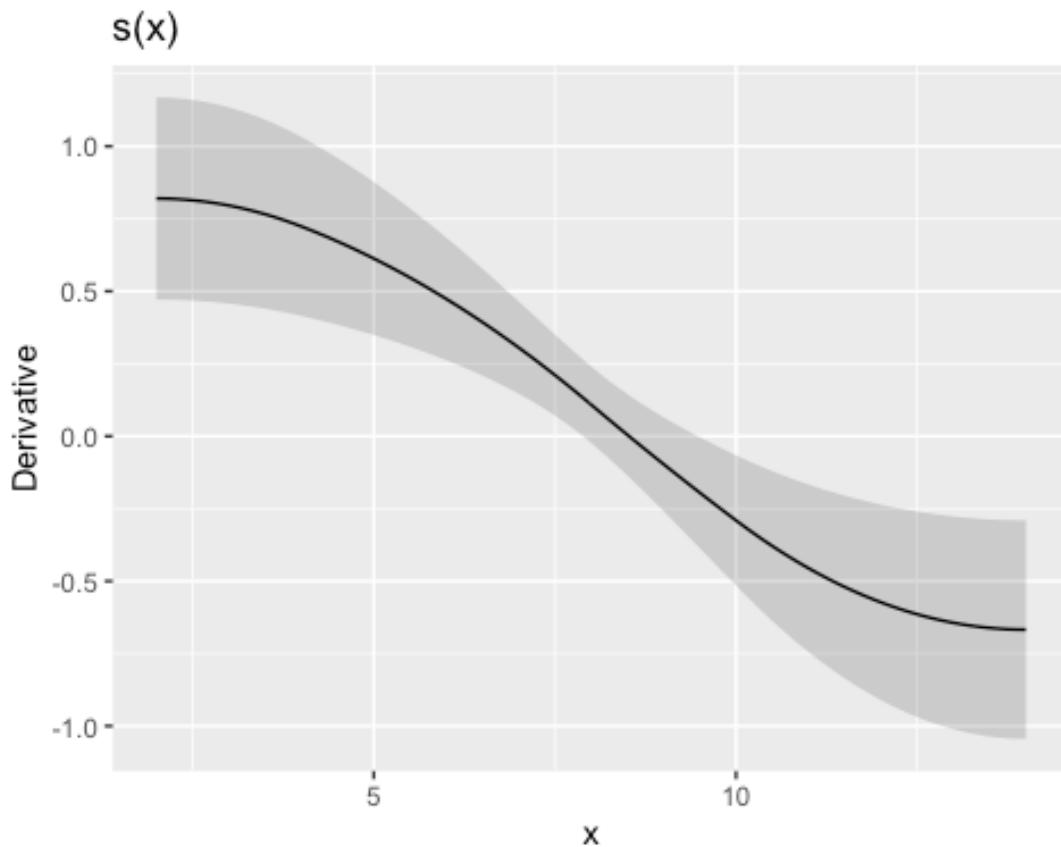
```
draw(data_gam.gam, residuals=TRUE)
```



assume our model was sign wiggly (because we had a low df, it might as well be a polynomial, but let's assume it's a GAM). We see that at some point, our trend reaches an optimum (max value) at around 8, so we might want to describe that. We could explain the optimum if we had the first order derivative, which can be calculated if we have the equation. However, this is NOT the case, as we got splines.

Step 1: We check the first order derivatives, and plot them

```
derivatives(data_gam.gam, order=1) %>% draw
```



The trend changes around the value of $x=8$. This equals to $y=0$, which is = derivative closer to zero.

Let's capture these derivatives:

```
d=derivatives(data_gam.gam, order=1)

head(d)

## # A tibble: 6 x 8
##   smooth var    data derivative    se  crit lower upper
##   <chr>  <chr> <dbl>      <dbl> <dbl> <dbl> <dbl>
## 1 s(x)    x     2       0.820 0.178 1.96 0.471 1.17
## 2 s(x)    x     2.06    0.820 0.178 1.96 0.471 1.17
## 3 s(x)    x     2.12    0.819 0.178 1.96 0.471 1.17
## 4 s(x)    x     2.18    0.819 0.178 1.96 0.471 1.17
## 5 s(x)    x     2.24    0.818 0.178 1.96 0.470 1.17
## 6 s(x)    x     2.30    0.818 0.177 1.96 0.470 1.17
```

We got a whole bunch of data, starting at 2. Its derivative was 0.82, and then we move along. If we want to know what the value of x associated with the derivative of ZERO was, we go and find the row closest to zero derivative (column=derivative):

```
d %>% summarise(Value=data[which.min(abs(derivative))],
lower=data[which.min(abs(lower))],
upper=data[which.min(abs(upper))])

## # A tibble: 1 x 3
##   Value lower upper
##   <dbl> <dbl> <dbl>
## 1 8.51  7.91  9.48
```

This tells us that the trend changes at x=8.512563, and indicates the upper and lower conf int. Indicating the trend is important, as it might indicate the optimal

Summary figures

We careate table with predicted values

```
data_gam.list=with(data_gam, list(x=seq(min(x), max(x),len=100))) # We create
100 rows of predicted values (100 is usually a good number)

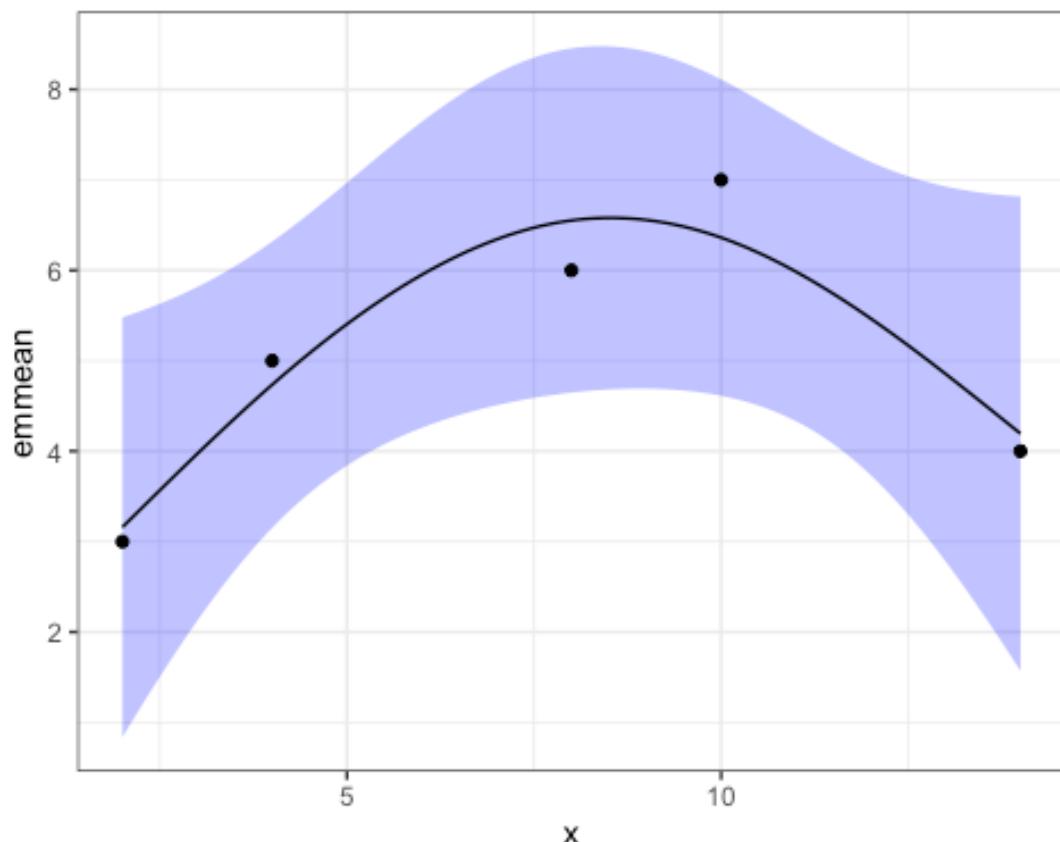
newdata=emmeans(data_gam.gam, ~x, at=data_gam.list) %>%
  as.data.frame

head(newdata)

##          x     emmean       SE      df lower.CL upper.CL
## 1 2.000000 3.158610 0.5522989 2.050997 0.8379937 5.479227
## 2 2.121212 3.257965 0.5366681 2.050997 1.0030250 5.512905
## 3 2.242424 3.357234 0.5214687 2.050997 1.1661572 5.548310
## 4 2.363636 3.456331 0.5067463 2.050997 1.3271145 5.585547
## 5 2.484848 3.555171 0.4925469 2.050997 1.4856168 5.624725
## 6 2.606061 3.653668 0.4789174 2.050997 1.6413817 5.665955
```

Now we can plot the predicted values

```
ggplot(newdata, aes(y=emmean, x=x))+
  geom_ribbon(aes(ymin=lower.CL,ymax=upper.CL),fill="blue",alpha=0.3)+
  geom_line()+
  geom_point(data=data_gam, aes(y=y,x=x))+
  theme_bw()
```



IGNORE BELOW

References

12_GAM_temporal

Sara Kophamel

14/12/2020

Open: gam_example3.Rmd

Preparations: Packages

Load the necessary libraries

```
library(mgcv)      #for GAMs
library(gratia)    #for GAM plots
library(emmeans)   #for marginal means etc
library(broom)      #for tidy output
library(MuMIN)     #for model selection and AICc
library(lubridate)  #for processing dates
library(tidyverse)  #for data wrangling
library(DHARMA)    #for residuals diagnostics
library(performance) #for residual diagnostics
library(see)        # to visualize residual diagnostics
library(patchwork)  #for grids of plots
```

Scenario

The Australian Institute of Marine Science (AIMS) have a long-term inshore marine water quality monitoring program in which water samples are collected and analysed from sites (reef.alias) across the GBR numerous times per year. The focus of this program is to report long-term condition and change in water quality parameters.

Although we do have latitude and longitudes, the nature of the spatial design predominantly reflects a series of transects that start near the mouth of a major river and extend northwards, yet mainly within the open coastal zone. As a result, this design is not well suited to any specific spatial analyses (since they are mainly one dimensional).

Format of aims.wq.csv data file

LATITUDE	LONGITUDE	reef.alias	Water_Samples	Region	Subregion	Season	waterYear	NOx
-16.1	145.	Cape Trib...	AIMS	Wet T...	Barron D...	Dry	2008	0.830
-16.1	145.	Cape Trib...	AIMS	Wet T...	Barron D...	Wet	2008	0.100
-16.1	145.	Cape Trib...	AIMS	Wet T...	Barron D...	Dry	2009	0.282
-16.1	145.	Cape Trib...	AIMS	Wet T...	Barron D...	Wet	2009	1.27
-16.1	145.	Cape Trib...	AIMS	Wet T...	Barron D...	Dry	2009	0.793
-16.1	145.	Cape Trib...	AIMS	Wet T...	Barron D...	Dry	2010	0.380
...



Figure 1: AIMS water quality monitoring

LATITUDE	- Latitudinal coordinate
LONGITUDE	- Longitudinal coordinate
reef.alias	- Internal AIMS reef name
Water_Samples	- Categorical label of who collected the data
Region	- The MMP region (a management region)
Subregion	- The MMP subregion (a management subregion)
Season	- A categorical listing of Wet or Dry
waterYear	- The water year (1st Oct - 30 Sept) to which the data are attached
Date	- The date the sample was collected
NOx	- Nitrite and Nitrate

For the purpose of this example, we will focus on the variable NOx. We want to see what the temporal changes of NOx could be over the next 50 yrs. NOx parameters are measured few times per year, but not too frequently. In other words, the data is gonna be messy, influenced by season, etc. and it's gonna be difficult to find the patterns in the data, as there are multiple variables. Temporal analysis data is one of those.

Temporal data usually requires a full and unborken dataset, but with GAMs we can work with missing data points (!)

Read in the data

```
wq = read_csv('data/aims.wq.csv', trim_ws=TRUE)

## Warning: Duplicated column names deduplicated: 'Dt.num' => 'Dt.num_1' [25]

## Parsed with column specification:
## cols(
##   .default = col_double(),
##   reef.alias = col_character(),
##   Water_Samples = col_character(),
##   Region = col_character(),
##   Subregion = col_character(),          2
##   Season = col_character(),
##   Date = col_date(format = ""),
##   Source = col_character(),
##   ...)
```

```

## $ Mnth          <dbl> 10, 3, 10, 2, 6, 10, 3, 6, 10, 2, 6, 10, 2, 6, 10, ...
## $ Dt.num        <dbl> 2007.784, 2008.243, 2008.779, 2009.153, 2009.452, ...
## $ Source        <chr> "AIMS Niskin", "AIMS Niskin", "AIMS Niskin", "AIMS...
## $ CDOM_443      <lgl> NA, NA...
## $ CHL_TSS       <dbl> 0.3101798, 0.1077403, 0.1927524, 0.4841206, 0.2699...
## $ DIN           <dbl> NA, 1.7603107, 1.8912551, 1.5700109, 0.9340067, 0....
## $ DIP           <dbl> 3.2430236, 0.0000000, 8.4301889, 0.2818657, 3.3818...
## $ DOC_DON       <dbl> 9.894332, 18.838502, 10.247938, 8.853276, 35.17359...
## $ DOC_DOP       <dbl> 249.57930, 164.16894, 232.26052, 138.28572, 143.74...
## $ DOC           <dbl> 568.7456, 800.0811, 665.5129, 790.8947, 709.3114, ...
## $ DON_DOP       <dbl> 25.371405, 8.672342, 21.385520, 15.773521, 4.21869...
## $ DON           <dbl> 58.39177, 42.59066, 66.56530, 89.86323, 21.04308, ...
## $ DOP           <dbl> 2.360146, 5.676724, -3.369768, 6.108445, 5.245931, ...
## $ DRIFTCHL_UGPERL <dbl> 0.2348550, 0.3407700, 0.3229300, 0.3026219, 0.3904...
## $ DRIFTPHAE_UGPERL <dbl> 0.10667250, 0.23786250, 0.14444250, 0.11588125, 0....
## $ Dt.num_1       <dbl> 2007.784, 2008.243, 2008.779, 2009.153, 2009.452, ...
## $ Dtt.num        <dbl> 1192330200, 1206843900, 1223763420, 1235609580, 12...
## $ HAND_NH4       <dbl> NA, 1.6579284, 1.1876348, 0.2446500, 0.1862600, 0....
## $ NH4            <dbl> 3.321632, 1.013287, 2.984873, 5.136503, 3.343649, ...
## $ NO2            <dbl> 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.1400...
## $ NO3            <dbl> 0.8225237, 0.0000000, 0.2744196, 1.2644041, 0.6533...
## $ NOx_P04        <dbl> 0.233605384, Inf, 0.166195654, Inf, 0.207299638, 0...
## $ NOx            <dbl> 0.8300237, 0.0100000, 0.2819196, 1.2675291, 0.7934...
## $ PN_CHL         <dbl> 50.81076, 70.97502, 43.20272, 32.14920, 40.25010, ...
## $ PN_PP          <dbl> 5.457474, 4.841050, 5.852297, 5.035881, 5.801272, ...
## $ PN_SHIM        <dbl> 18.99411, 29.22387, 13.21873, 14.93224, 18.73589, ...
## $ PN_TSS         <dbl> 0.017091541, 0.007517002, 0.008352101, 0.014976245...
## $ PN             <dbl> 9.556838, 17.876092, 13.877616, 9.648177, 13.33542...
## $ P04            <dbl> 3.2430236, 0.0000000, 8.4301889, 0.2818657, 3.3818...
## $ POC_CHL        <dbl> 643.2646, 594.1710, 320.9770, 300.8186, 334.7438, ...
## $ POC_PN          <dbl> 13.802976, 8.211349, 7.611680, 9.477119, 8.455543, ...
## $ POC_PP          <dbl> 78.17430, 39.91906, 43.86293, 46.88621, 48.09479, ...
## $ POC_TSS         <dbl> 0.28442285, 0.06276370, 0.06193767, 0.14314580, 0...
## $ POC             <dbl> 126.47385, 147.34368, 103.58926, 89.97550, 110.714...
## $ PP_CHL          <dbl> 9.073836, 13.775650, 7.352966, 6.429233, 6.769279, ...
## $ PP_TSS          <dbl> 0.002787327, 0.001460411, 0.001423642, 0.003094262...
## $ PP              <dbl> 2.025051, 3.678694, 2.371837, 1.922521, 2.336996, ...
## $ Salinity        <lgl> NA, NA...
## $ SECCHI_DEPTH    <dbl> 11.0, 4.0, 3.8, 6.0, 6.0, 11.0, 7.0, 3.5, 5.5, 6.5...
## $ SI_NOx          <dbl> 5345.75834, 32763.52291, 8925.66237, 5725.21940, 3...
## $ SI_P04          <dbl> 20.99143, Inf, 42.09525, Inf, 59.47313, 10.50605, ...
## $ SI              <dbl> 70.47872, 327.63523, 119.92590, 196.88465, 160.969...
## $ TDN             <dbl> 62.53592, 43.60394, 69.82460, 96.26414, 25.18013, ...
## $ TDP             <dbl> 5.603170, 5.676724, 5.060421, 6.390311, 8.627777, ...
## $ TN_TP           <dbl> 9.567956, 6.958258, 11.240651, 13.392960, 3.586795...
## $ Total1N         <dbl> 72.09276, 61.48004, 83.70221, 105.91231, 38.51556, ...
## $ Total1P         <dbl> 7.628221, 9.355418, 7.432258, 8.312832, 10.964774, ...
## $ TSS_MGPERL      <dbl> 1.052500, 3.120000, 1.695000, 0.761250, 1.505000, ...

```

Data preparations

Setting variables as factors

We will prepare the data as if we were fitting a random effects model, so we have to make sure that the variables we include in the model are defined as either numerical or factors. In this case, the Regions (the reefs) are defined as categories/factors; same as if we defined Region as random effect in a blocking design. We also need to make sure other variables we might include are defined as factors.

```
wq=wq %>%
  mutate(reef.alias=factor(reef.alias),
    Region=factor(Region),
    Subregion=factor(Subregion),
    Season=factor(Season))
```

Dates

Date HAS to be in standard US format (see above) to be recognised as such by R. If we do have dates in our data which are not in that format (e.g., format “14 Dec 2020”), we do the following:

```
d<-"14 Dec 2020"
as.Date(d,format='%d %b %Y') # you could add here --> %>% lubridate::decimal_date()

## [1] "2020-12-14"

# or: as.Date(d,format='%d/%b/%Y') # would separate the date as day/month/yr
# Pay attention to leave the spaces
# lubridate = for manipulating Dates. Turns the date into the decimal number of it.
# %b = month
```

The function ?strptime will tell you what all of these %s are.

```
?strptime
```

If I turned Date into numeric, R would tell me how many days have passed since 1.Jan 1970 (don't ask me why)

```
# as.Date(d,format='%d %b %Y') %>% as.numeric
```

(!) Pay attention that you export your data as the streamed format (2020-12-14) = Don't let Excel (common bug) or R turn your dates as NUMBERS !!

Exploratory data analysis

Let's see what NOx is like over time

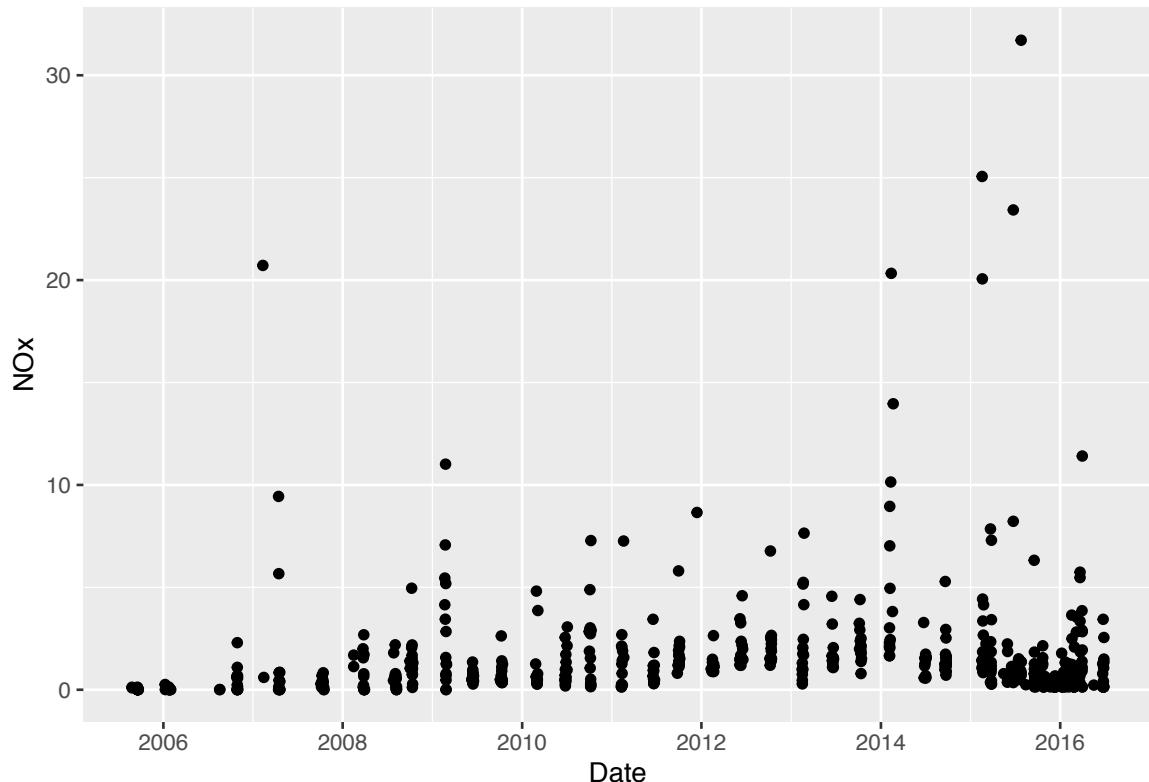
Model formula:

$$y_i \sim \mathcal{N}(\mu_i, \sigma^2) \mu_i = \beta_0 + f(Date_i) + f(Month_i)$$

where β_0 is the y-intercept. $f(Date)$ and $f(Month)$ indicate the additive smoothing functions of the long-term temporal trends and the annual seasonal trends respectively.

```
ggplot(wq, aes(y=NOx,x=Date))+
  geom_point()

## Warning: Removed 50 rows containing missing values (geom_point).
```

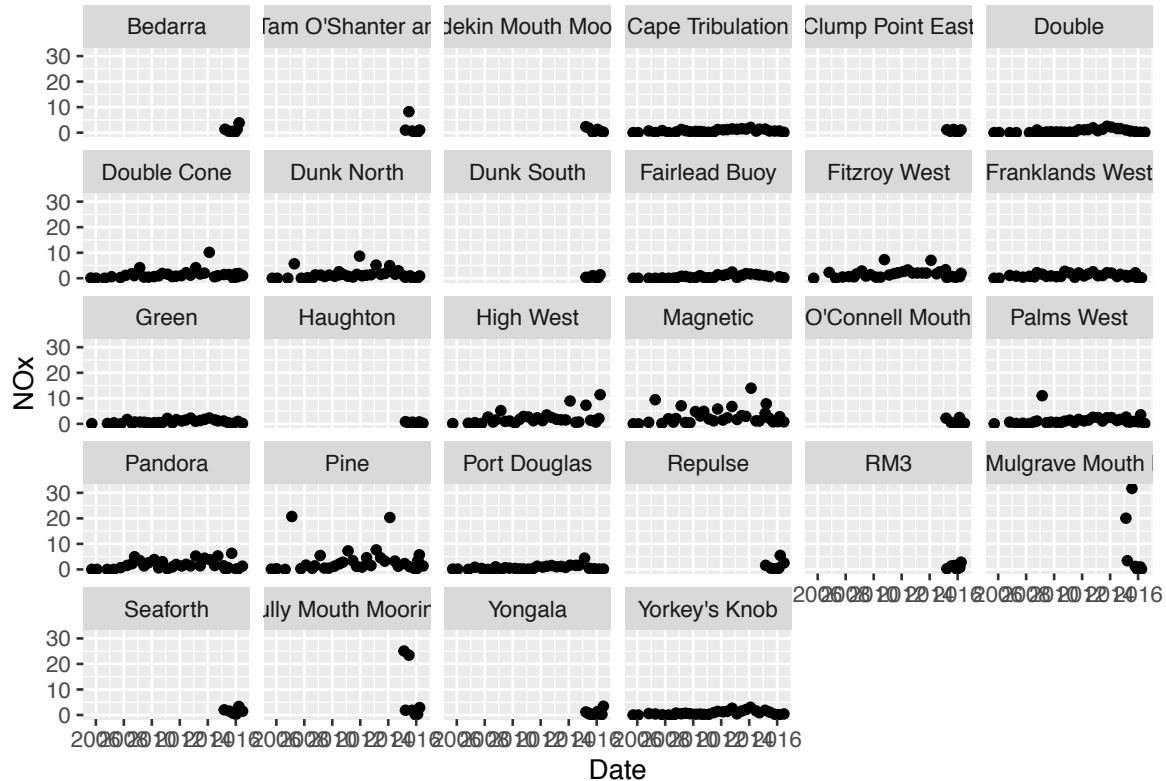


The first record was in 2006, and it stops around the end of 2016. Because the x variable was a date, ggplot will provide an axis that is already drafted for years. If we for example had only months within a year, ggplot would plot Months to the axis. So it is an advantage to use actual dates.

These data come from a bunch of locations, so it might be difficult to find patterns if we combine them all together. Let's divide by location:

```
ggplot(wq, aes(y=NOx,x=Date))+
  geom_point()+
  facet_wrap(~reef.alias)

## Warning: Removed 50 rows containing missing values (geom_point).
```



Facet_wrap will plot the combinations altogether. Facet_grid will use either rows and columns, according to your categories (eg one for summer, one for winter)

In this case, we can see that some of the sites are relatively recent, and that others have longer time series.

We will start off by looking at a long time series location. Let's choose Pandora. We want to fit a model that explains temporal changes in NOx for Pandora.

Pseudo log scale

Log transformations are quite common, so there is a good chance, given that our values for NOx are not gonna be normally distributed (they are sparser as you move up the y axis), they might either be Gamma or Log-normal (with log links). So let's look at a pseudo log scale. A pseudo log scale is used when we got zeros, but still want to display a log scale. It essentially leaves the zeros on the plot if there are some on the data, but logs all other values.

```
ggplot(wq, aes(y=NOx,x=Date))+
  geom_point()+
  geom_smooth()+
  scale_y_continuous(trans=scales::pseudo_log_trans())+ # pseudo log transformation command
  facet_wrap(~reef.alias, scales="free_y") # scales="free_y" zooms into the data, making it easier to v
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## Warning: Removed 50 rows containing non-finite values (stat_smooth).
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6
```

```

## Warning in sqrt(sum.squares/one.delta): NaNs produced
## Warning in stats::qt(level/2 + 0.5, pred$df): NaNs produced
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6

## Warning in sqrt(sum.squares/one.delta): NaNs produced
## Warning in stats::qt(level/2 + 0.5, pred$df): NaNs produced
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6

## Warning in sqrt(sum.squares/one.delta): NaNs produced
## Warning in stats::qt(level/2 + 0.5, pred$df): NaNs produced
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6

## Warning in sqrt(sum.squares/one.delta): NaNs produced
## Warning in stats::qt(level/2 + 0.5, pred$df): NaNs produced
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : Chernobyl! trL>n 6

## Warning in sqrt(sum.squares/one.delta): NaNs produced
## Warning in stats::qt(level/2 + 0.5, pred$df): NaNs produced
## Warning: Removed 50 rows containing missing values (geom_point).

## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -
## Inf

## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -
## Inf

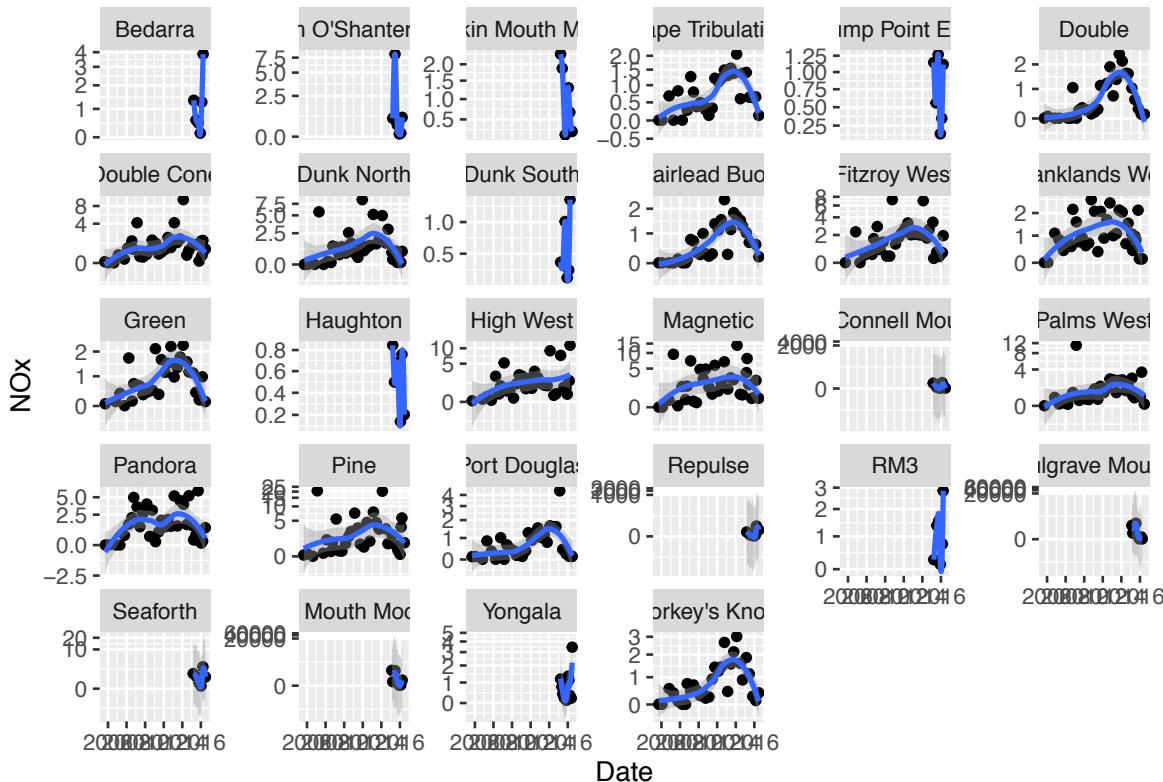
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -
## Inf

## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -
## Inf

```

```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -
## Inf
```

```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -
## Inf
```



It does look like our model is not linear. Every y axis is different, which makes it easier to see the patterns.

Simple model (Pandora only)

Filtering the Pandora rows out of the wq dataset

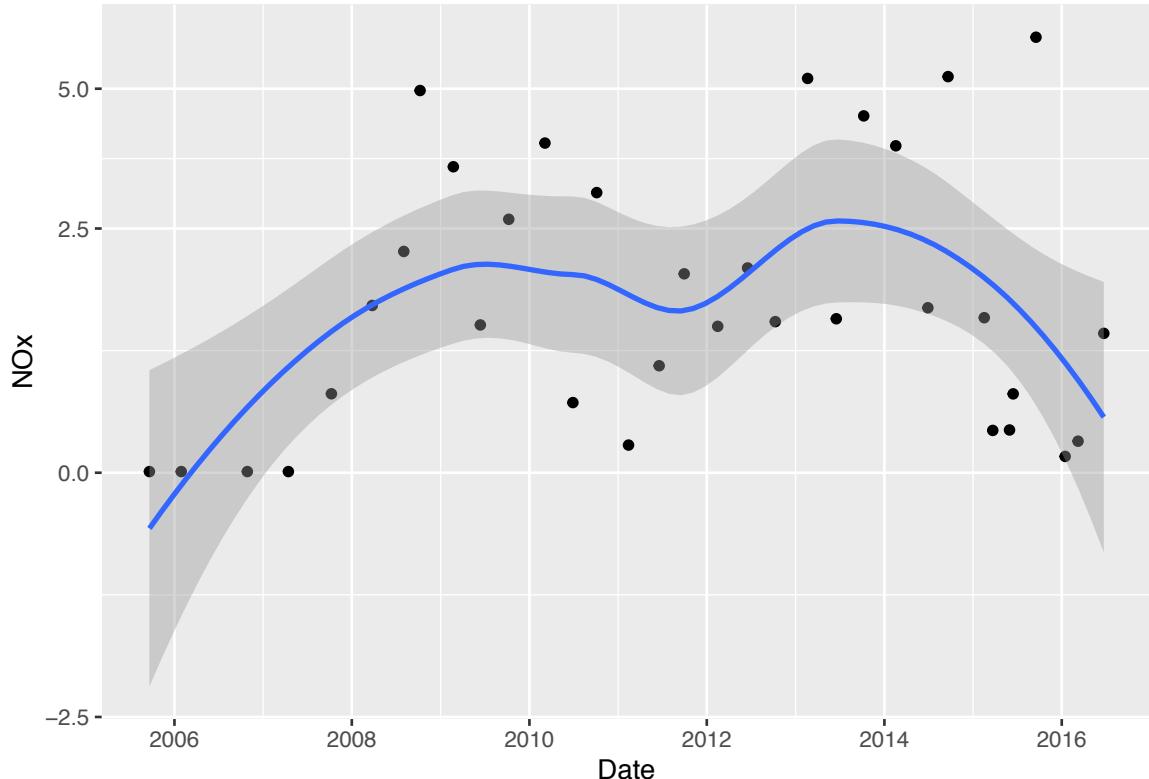
As we only want the Pandora region, we have to determine what rows we want

```
wq.pandora<-wq %>%
  filter(reef.alias=="Pandora", !is.na(NOx))
# we also tell R to only include those rows that are NOT NA values
```

Exploratory Pandora plot

Let's plot the Pandora region

```
ggplot(wq.pandora, aes(y=NOx, x=Date))+
  geom_point()+
  geom_smooth()+
  scale_y_continuous(trans=scales::pseudo_log_trans())
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Deciding on the best distribution The $x=0$ values must be an instrumentation error, when NOx was not detectable. However, in reality there should be no zeros. We could do the log of the response, but we should choose a more sensible distribution, if possible. Sensible distributions would be - A log-like distribution (Gaussian with log-link) - A Gamma (also with log-link) - A Tweedy distribution, which is somewhere btw Poisson and Gamma => It allows zeros, but is quite Gamma like, as it does not require integers. In this case, it is useful, as it is all real numbers, plus zeros.

All of these are sensible enough candidates. We will fit each of those and see which one satisfies the distributions of the data. For fun, we will also fit a normal Gamma (without log link), although this one is very unlikely gonna be a sensible distribution.

Exploring more models

Model 1: Gaussian normal distribution

```
wq.gam1<-gam(NOx~s(Dt.num), data=wq.pandora, family="gaussian", method="REML")
# s fits a smoother, and defines by default 10 knots
# Dt.num = Date converted to a numeric fraction. If we used "Date" only, we had days elapsed since 1970
```

This is a straight normal model, which we think is NOT gonna be sensible. Murray is fitting it only for comparison purposes

Model 2: Gaussian with log link

```
wq.gam2<-gam(NOx~s(Dt.num), data=wq.pandora, family="gaussian"(link="log"), method="REML")
```

Model 3: Gamma with log link

```
wq.gam3<-gam(NOx~s(Dt.num), data=wq.pandora, family="Gamma"(link="log"), method="REML")
```

Model 4: Tweedy distribution with log link

```
wq.gam4<-gam(NOx~s(Dt.num), data=wq.pandora, family="tw"(link="log"), method="REML")
```

Comparing models

```
AICc(wq.gam1,wq.gam2,wq.gam3,wq.gam4)
```

```
##          df      AICc
## wq.gam1 4.681877 140.14932
## wq.gam2 4.866293 140.79455
## wq.gam3 8.073340  98.91884
## wq.gam4 9.004837 102.20140
```

wq.gam3 (the gamma model) seems to be the best fitting model.

Model investigation / hypothesis testing

Now we want to make sure this model is actually sensible, and if it follows the assumptions. The max number of knots is 10. We don't know if this is enough. Let's check it.

k check

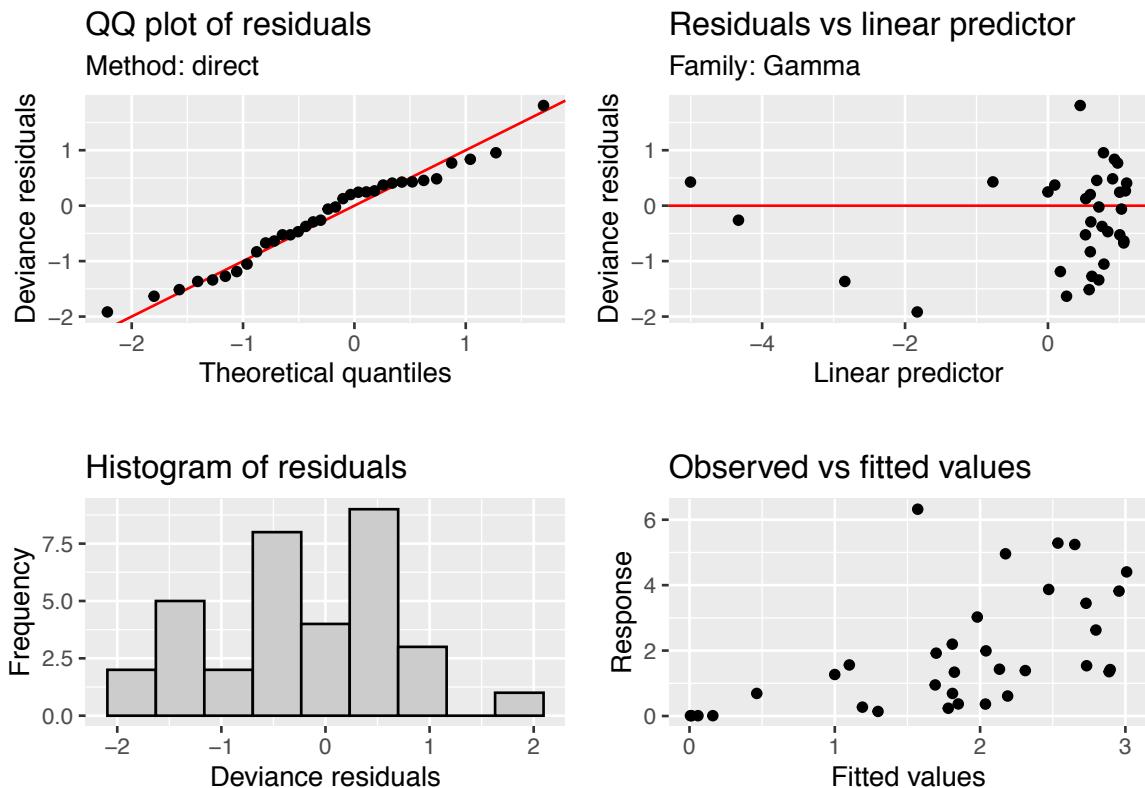
```
k.check(wq.gam3)
```

```
##          k'      edf  k-index p-value
## s(Dt.num) 9 5.076167 1.019582  0.6475
```

- k' = We got 9 df
- edf = Our estim df is 5.08. We wanted it to be <8, so that's good. This gives us a sense for how wiggly the line is gonna be. A straight line is = 1.
- k-index = Has to be >1. It's ok. Below 1, or substantially below 1, would be bad.
- p-value = It's ns, which is good. It indicates if we got a significantly overconstrained k, which is not the case here.

appraise

```
appraise(wq.gam3)
```

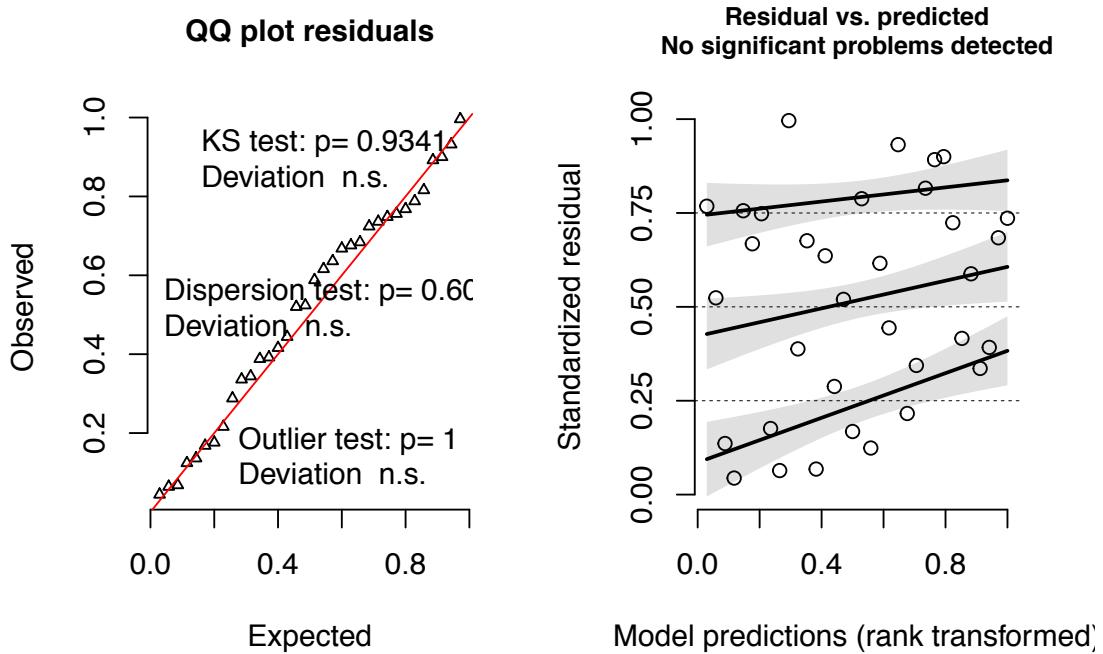


The QQ plot is ok - The residual model does not necessarily predict the data, so there is a lot we can't explain

DHARMA residuals

```
wq.resid<-simulateResiduals(wq.gam3,plot=TRUE)
```

DHARMA residual diagnostics

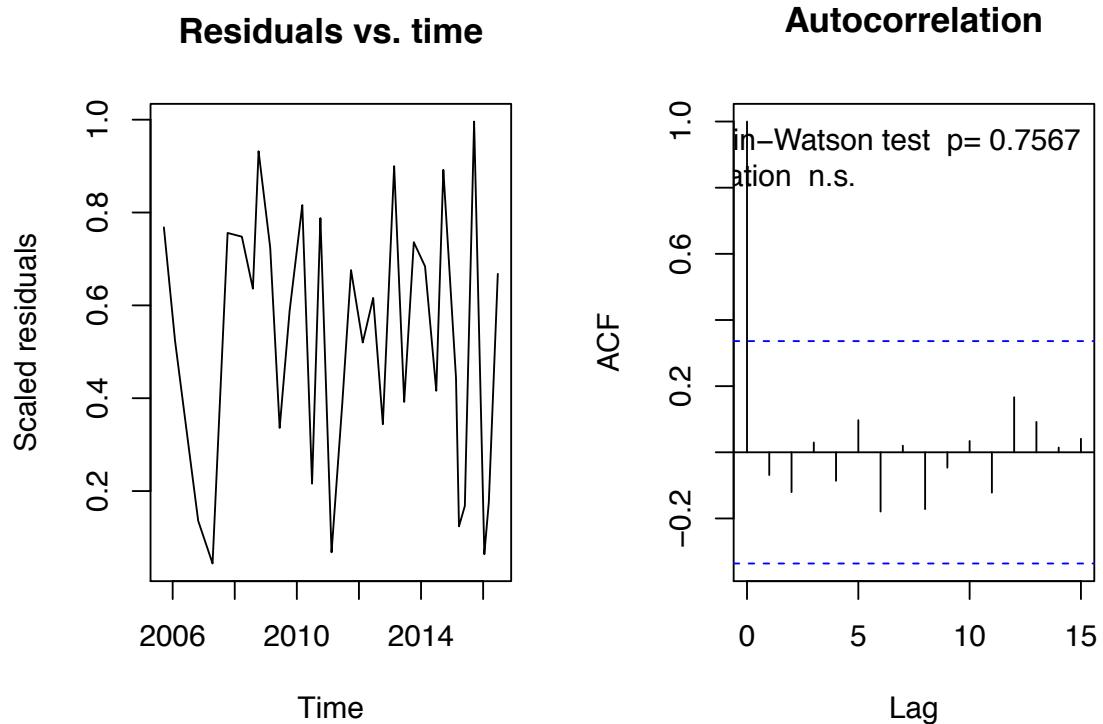


DHARMA residuals look ok.

Autocorrelation test

Given that this is a temporal dataset, we wanna see if there is autocorrelation, if the noise/variability changes over time. We can see this from the residuals we just calculated.

```
testTemporalAutocorrelation(wq.resid, time=wq.gam3$model1$Dt.num)
```



```
##  
## Durbin-Watson test  
##  
## data: simulationOutput$scaledResiduals ~ 1  
## DW = 2.1053, p-value = 0.7567  
## alternative hypothesis: true autocorrelation is not 0
```

For most models, we do need to assume independence from our residuals. BUT if there is an autocorrelation patterns not explained by time alone, we will be violating this assumption. In that case, our explained variability will be reduced, and we'd have more power than assumed. We don't want autocorrelation in our model. If you put time in the model, it will account for temporal autocorrelation, but it does not like if the noise btw the extremes is not about the same as the variance. The effect would be x wide, plus some sort of noise. And it is that noise that needs to be consistent throughout.

If you are measuring the SAME ind, you'd expect an autocorrelation problem. IF we had autocorrelation, you'd have to include it in the model with correlation=corAR1(~Dt.num), like this:

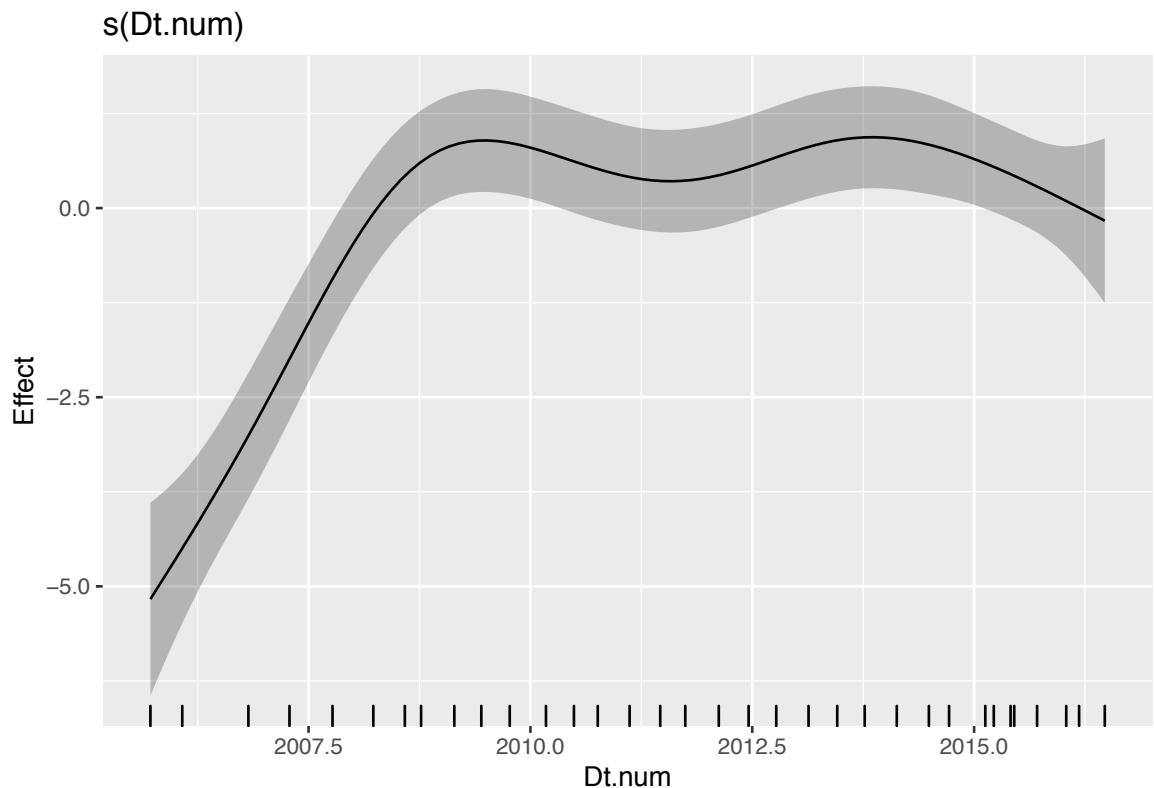
```
# wq.gam3<-gam(NOx~s(Dt.num), data=wq.pandora, correlation=corAR1(~Dt.num), family="Gamma"(link="log"), meti
```

Don't do this if you did not find any autocorrelation, as you would decrease the power.

We can see that in our example, there is no evidence of autocorrelation, so we keep going

Exploratory plot

```
draw(wq.gam3)
```



Model output

```
summary(wq.gam3)
```

```
##
## Family: Gamma
## Link function: log
##
## Formula:
## NOx ~ s(Dt.num)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.1664    0.1520   1.095   0.283
##
## Approximate significance of smooth terms:
##          edf Ref.df    F p-value
## s(Dt.num) 5.076  6.178 15.48 8.18e-10 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.18 Deviance explained = 55.7%
## -REML = 47.788 Scale est. = 0.78522 n = 34
```

Parametric coefficients: Estimate Std. Error t value Pr(>|t|) (Intercept) 0.1664 0.1520 1.095 0.283 -> Indicates the Intercept, which is on a LOG scale. If we took the x of this value, we'll get sth more sensible.

s(Dt.num) 5.076 6.178 15.48 8.18e-10 *** -> df = 5 -> p value <0.05, so we got wigginess in our data

R-sq.(adj) = 0.18 Deviance explained = 55.7% \rightarrow We are explaining around 56% of the variability

Model 5: Interaction with Season

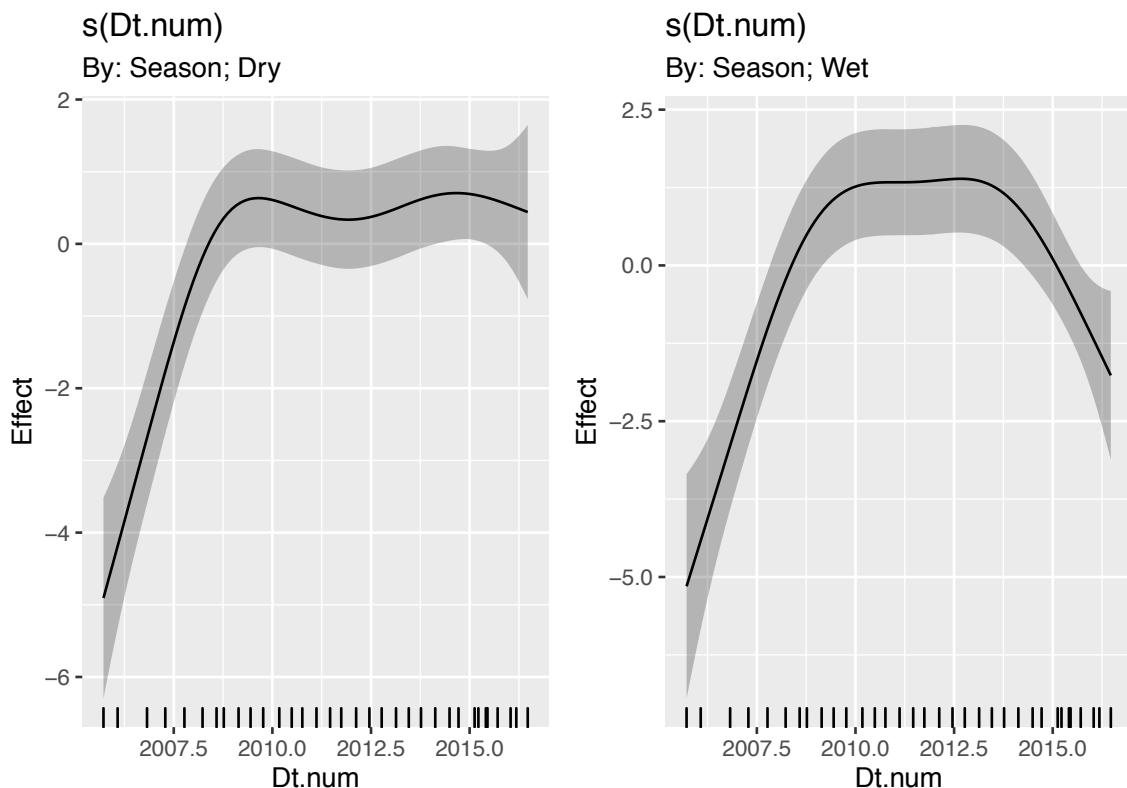
We know that our data was collected at diff seasons, so we might wanna fit two separate trends: wet and dry seasons. We might also want to take out an entire seasonality from it. We know the dates, but we may wanna see if there are more trends within a same year, on top of a long-term trend (over several years).

A GAM can only work on continuous data.

```
wq.gam5<-gam(NOx~s(Dt.num,by=Season),
  data=wq.pandora,
  family=Gamma(link="log"), method="REML")

# If Season was not declared as CATEGORICAL variable in the beginning, we could get an error message. W

draw(wq.gam5)
```



The wet season shows a dip in the most recent years, the dry season does not. This is how you add an interaction (in this case, with Season) to your model.

To say that the Seasons are different, you'd have to do a Bayesian analysis. You CAN'T do it for frequentist analysis. The only thing you can do here is describe the plot.

We should have gone through all diagnostic checks as well, but let's assume they were ok.

```
summary(wq.gam5)
```

```
##
## Family: Gamma
## Link function: log
```

```

## 
## Formula:
## NOx ~ s(Dt.num, by = Season)
## 
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  0.1785    0.1439   1.24    0.226    
## 
## Approximate significance of smooth terms:
##                  edf Ref.df      F p-value    
## s(Dt.num):SeasonDry 4.130 5.104 11.14 2.22e-06 ***
## s(Dt.num):SeasonWet 3.776 4.651 11.48 3.25e-06 ***  
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## R-sq.(adj) = -0.0865  Deviance explained = 59.4% 
## -REML = 48.752  Scale est. = 0.69628 n = 34

```

As you can see, we get one row explaining the Dry and one row explaining the Wet season edf Ref.df F p-value

`s(Dt.num):SeasonDry 4.130 5.104 11.14 2.22e-06 s(Dt.num):SeasonWet 3.776 4.651 11.48 3.25e-06`

Model 6: Identifying short and long-term trends

What if we wanted to know how NOx changes over the year? And once we take that signal out, what does our long-term trend (over the years) look like? Can we take our trend and split it into long-term trend, and short-term seasonal periodicity?

Let's create the model to test this:

```
wq.gam6crap<-gam(NOx~s(Dt.num)+  
                     s(Mnth,bs="cc",k=5), # The smoother "cc" is a cyclical smoother. The time of the year is  
                     data=wq.pandora,  
                     family=Gamma(link="log"), method="REML")
```

BUT this model has a problem, as we are assessing the following points of the year:

```
summary(wq.pandora$Mnth)
```

```
##   Min. 1st Qu. Median Mean 3rd Qu. Max. 
## 1.000 3.000 6.000 5.794 9.000 10.000
```

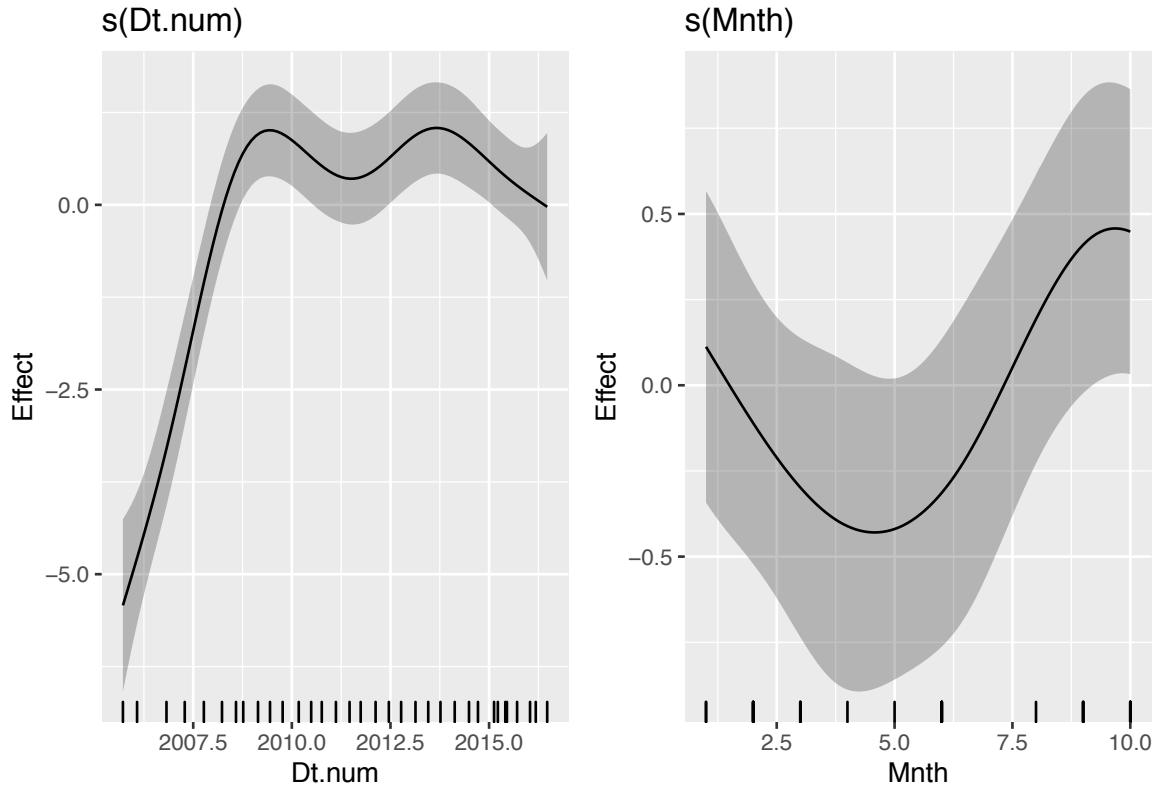
If we look at the actual data, the minimum month was 1 (Jan), but no data was sampled in Nov (11) and Dec (12). The model would therefore link Jan (1) to Oct (10), because it has NO notion that there are two months there.

A way to fix this is by adding the following

```
wq.gam6<-gam(NOx~s(Dt.num)+  
                 s(Mnth,bs="cc",k=5), # we are fitting a cyclical smoother  
                 knots=list(Mnth=seq(1,12,length=5)), # Accounts for 12 months, as we are MISSING Nov and D  
                 data=wq.pandora,  
                 family=Gamma(link="log"), method="REML")
```

Partial plots

```
draw(wq.gam6)
```



The left plot indicates the LONG TERM trend, after de-trending the seasonal trend. - The one on the right is the trend within a year. Because we haven't got months 11 and 12, that plot finishes at Month 10.

Our plots are centered on the y axis at 0, which R always does for partial plots. It essentially shows the trend over time when R does not know what month we are interested in. In other words, R does not consider the month (the absolutes), but only shows the trend. You would have to put all months together to know the full trend.

Summary

```
summary(wq.gam6)

##
## Family: Gamma
## Link function: log
##
## Formula:
## NOx ~ s(Dt.num) + s(Mnth, bs = "cc", k = 5)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.07122   0.13274   0.537   0.596
##
## Approximate significance of smooth terms:
##             edf Ref.df    F p-value
## s(Dt.num) 5.566 6.704 20.320 7.07e-13 ***
## s(Mnth)   1.658 3.000  2.455   0.0179 *
## ---
```

```
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.325 Deviance explained = 66.7%
## -REML = 46.3 Scale est. = 0.59909 n = 34
```

Explanation of short and long-term trends - Check the following row: Approximate significance of smooth terms: edf Ref.df F p-value
 s(Dt.num) 5.566 6.704 20.320 7.07e-13 *** -> Indicates a sign long-term trend (over several years) s(Mnth)
 1.658 3.000 2.455 0.0179 * -> Indicates a sign short-term, seasonal trend (over one year)

Quick summary plot: Long-term trend

-> You'd also do these steps for the short-term trend info, but as an example, we'll do it on the long term trend.

First, we predict our data

```
wq.list=with(wq.pandora,list(Dt.num=seq(min(Dt.num),max(Dt.num),len=100)))
# Here, we are marginalising the data to show the long-term trend irrespective of the month

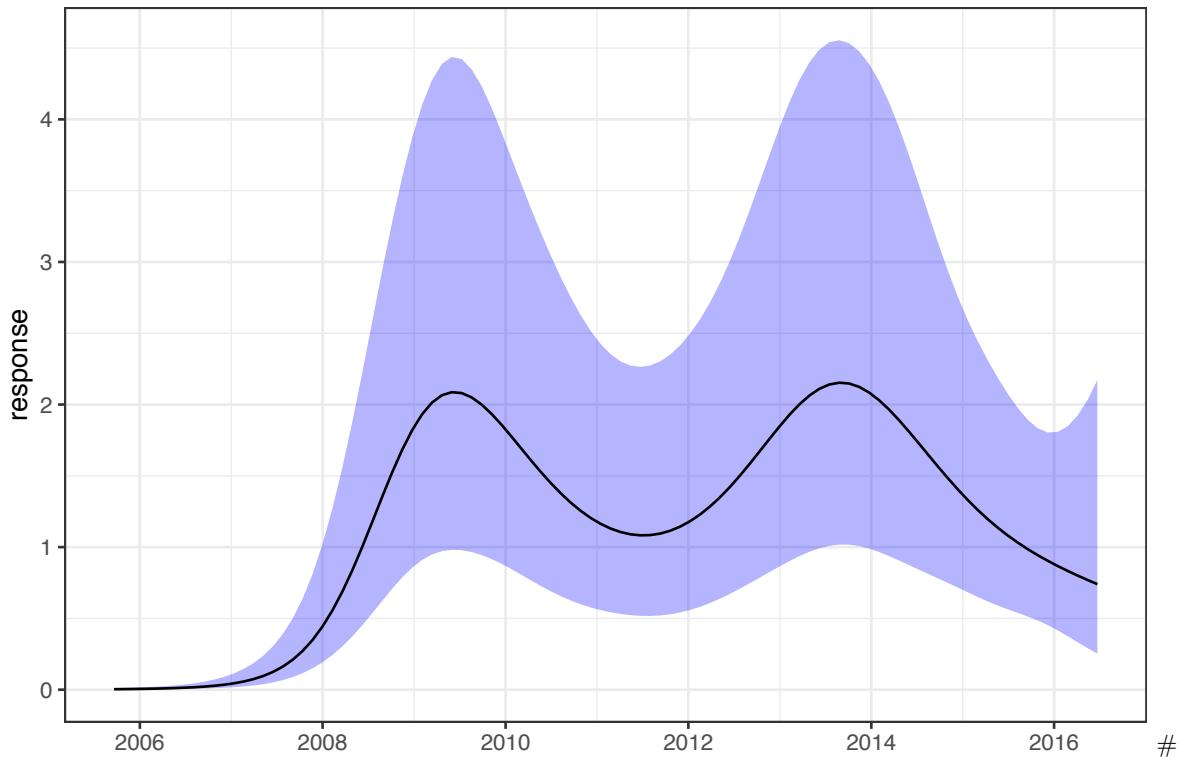
newdata=emmeans(wq.gam6,-Dt.num, at=wq.list, type="response") %>% #type=response back transforms our l
as.data.frame

head(newdata)

##      Dt.num     response        SE       df   lower.CL   upper.CL
## 1 2005.718 0.003359915 0.002176721 25.77557 0.0008866333 0.01273246
## 2 2005.826 0.004073045 0.002451198 25.77557 0.0011815521 0.01404060
## 3 2005.935 0.004942854 0.002775161 25.77557 0.0015579620 0.01568190
## 4 2006.044 0.006011341 0.003172633 25.77557 0.0020306007 0.01779583
## 5 2006.152 0.007334280 0.003677146 25.77557 0.0026157563 0.02056448
## 6 2006.261 0.008984412 0.004332737 25.77557 0.0033327266 0.02422030
```

Now, we can plot the predicted, back-transformed data

```
ggplot(newdata,aes(y=response, x=date_decimal(Dt.num)))+ # we back transform the numeric scale back into
geom_ribbon(aes(ymax=lower.CL, ymin=upper.CL), fill="blue", alpha=0.3)+
geom_line()+
scale_x_datetime("")+
theme_bw()
```



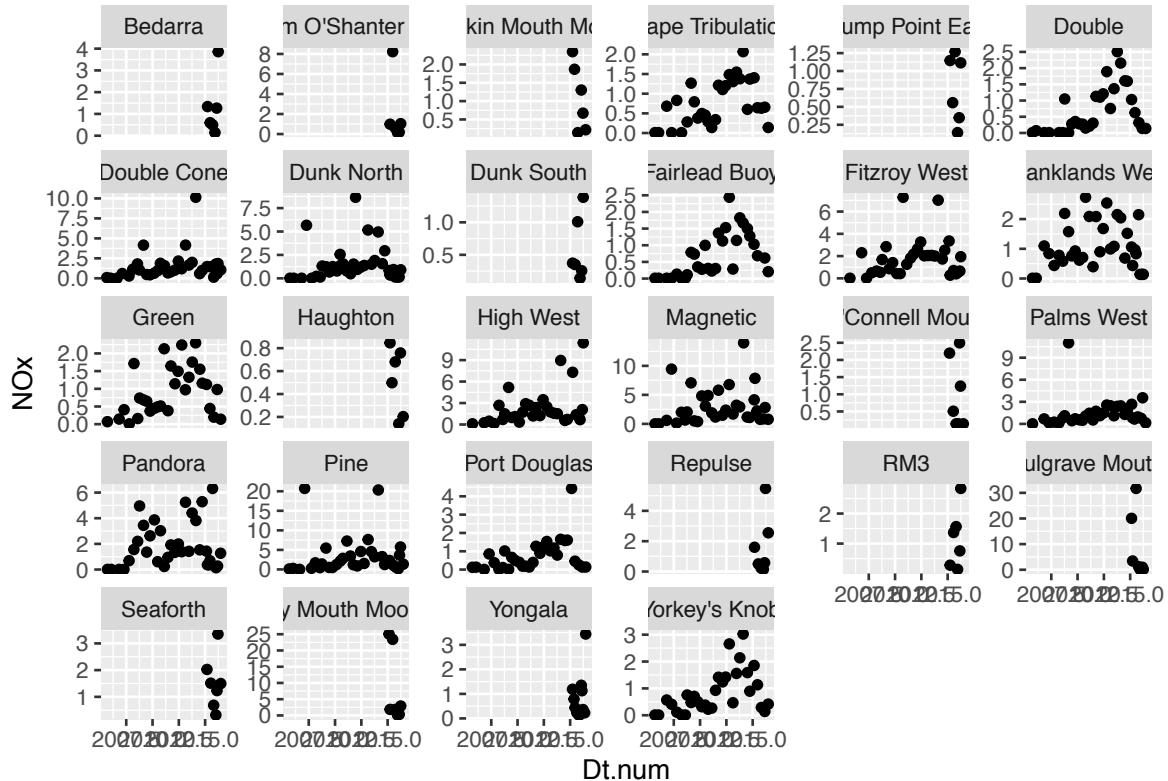
ASYMPTOTIC CONF INT QUESTION Why do we not use asymptotic conf int (lmer.df = "asymptotic") instead of the standard upper and lower CIs? Because it can calculate the df (it would say NA if ti couldn't). And because it would actually show the asymp CI by default if it needed them (all LM do that)

Mixed model (all reefs)

We throw everything we learned from Pandora in this model, extrapolating the previously established model to all other locations

Exploratory plot of all locations

```
ggplot(wq,aes(y=N0x,x=Dt.num))+
  geom_point()+
  facet_wrap(~reef.alias,scales="free_y")
## Warning: Removed 50 rows containing missing values (geom_point).
```



Let's fit a generalised additive mixed model for all locations then.

We could write:

Model 1: AVOID

```
wq.gamm1<-gamm(NOx~s(Dt.num), random=list(reef.alias=~1), #random=list defines the random intercept mod
  data=wq,
  family=Gamma(link="log"),
  method='REML')

## Maximum number of PQL iterations: 20
## iteration 1
## iteration 2
## iteration 3
## iteration 4
## iteration 5
## iteration 6
## iteration 7
## iteration 8
## iteration 9
## iteration 10
```

```
## iteration 11
```

BUT we got fitted a PQL model by default. PQL indicates quasi penalised likelihood. If you can avoid a model with quasi penalised likelihood, AVOID it, as you can't do AIC and model validations (!) Some cases where you might not be able to avoid PQL are where you got unequal variance and you wanna fit a model which has got separate variances for separate groups. But this is VERY rare.

Rather than using GAM under the hood, this model is using a PQL for random effects, that is a maximum likelihood because of the way it works. It USED to be the only way to fit a GAMM model, but it is totally outdated.

Another option (not to be used) is...

Model 2 (package gamm4): AVOID

```
wq.gamm2<-gamm4::gamm4(NOx~s(Dt.num), random=~(1|reef.alias), #random=list defines the random intercept
                         data=wq,
                         family=Gamma(link="log"),
                         REML=TRUE)
```

Does NOT use a smoother, so don't pursue this model either

Model 3: USE THIS ONE

```
wq.gamm3<-gam(NOx~s(Dt.num)+s(reef.alias,bs="re"),
                 data=wq,family=Gamma(link="log"), method="REML")
```

Number of knots = number of locations (set by default), up to a maximum of 10 = The maximum number of

This model uses GLM (I think), being more stable

Model validation

```
k.check(wq.gamm3)

##          k'      edf  k-index p-value
## s(Dt.num)    9  8.090924 0.7049284     0
## s(reef.alias) 28 22.801446        NA      NA
```

edf is close to k' k index is <1 p value is <0.5

Model 3a with more knots

=> We therefore have to INCREASE the number of knots. Let's DOUBLE the knots.

```
wq.gamm3a<-gam(NOx~s(Dt.num, k=20)+s(reef.alias,bs="re"),
                  data=wq,family=Gamma(link="log"), method="REML")
```

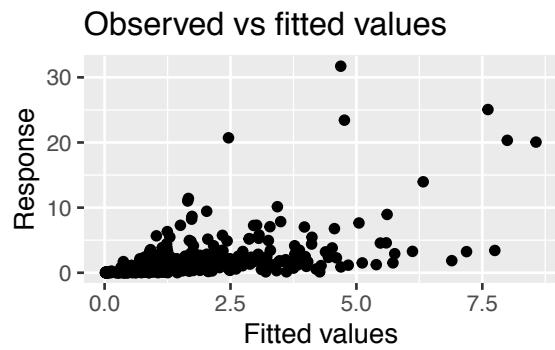
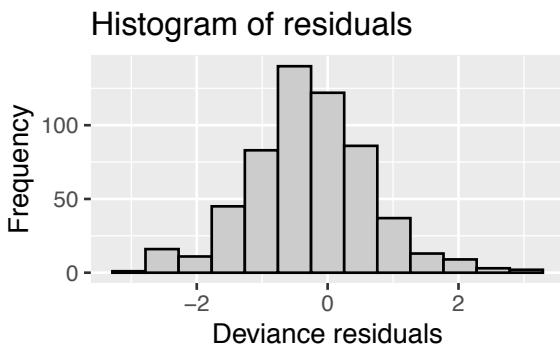
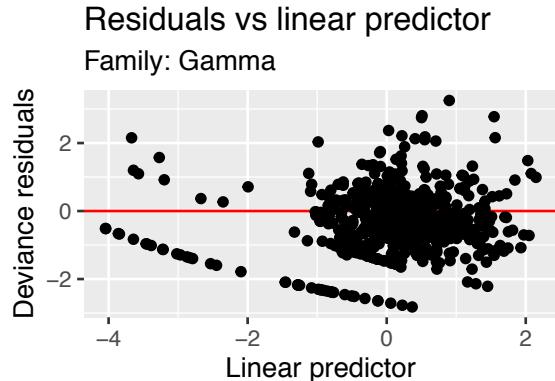
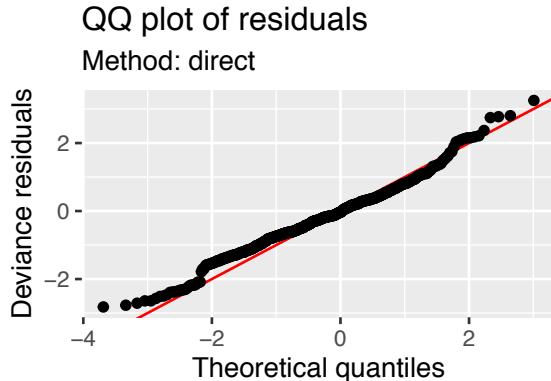
```
k.check(wq.gamm3a)

##          k'      edf  k-index p-value
## s(Dt.num)   19 14.94452 0.7543448     0
## s(reef.alias) 28 22.82756        NA      NA
```

Even increasing the knots, it did not make any difference.

What do we do now? Let's check the rest of our diagnostics #### appraise

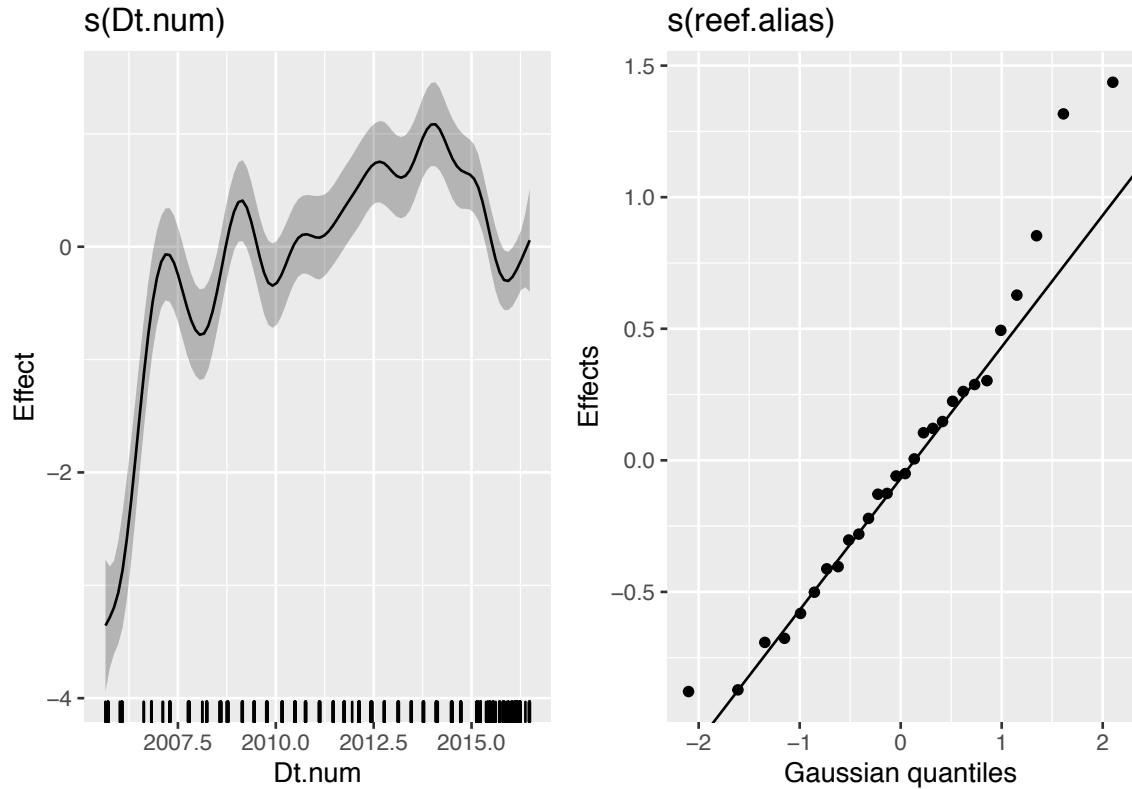
```
appraise(wq.gamm3a)
```



QQ is not disastrously bad, so we assume it's ok - The residuals are more or less ok - Histogram: The distribution of residuals looks pretty good, sort of normal - Our prediction against the observed has got a lot of noise in the data, and is not fantastic. By making the model a bit more complex, we might be able to reduce that noise.

Partial plot

```
draw(wq.gamm3a)
```



On the left is our long term trend, displayed 0-centered. - The right is like a QQ plot for random effects. What we wanna see here are our data points being on a straight line.

Model summary

```
summary(wq.gamm3a)

##
## Family: Gamma
## Link function: log
##
## Formula:
## NOx ~ s(Dt.num, k = 20) + s(reef.alias, bs = "re")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.1505    0.1477   1.019   0.309
##
## Approximate significance of smooth terms:
##             edf Ref.df      F p-value
## s(Dt.num)    14.94    17.2 20.448 <2e-16 ***
## s(reef.alias) 22.83    27.0  6.069 <2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.246 Deviance explained = 46.3%
## -REML = 684.43 Scale est. = 1.1184 n = 568
```

Previously, our model included the cyclic smoother for months, so let's include it:

Model 3b with cyclic smoother

```
wq.gamm3b<-gam(NOx~s(Dt.num, k=20)+ # For every smoother, you define its own k. In this case, we defin
  s(Mnth,bs="cc",k=5)+ # For the cyclic smoother, k=5 is what we had defined initially,
  s(reef.alias,bs="re"),
  knots=list(Mnth=seq(1,12,length=5)), # here, we define length as the k from the cyclic s
  family=Gamma(link="log"),
  data=wq, method="REML")
```

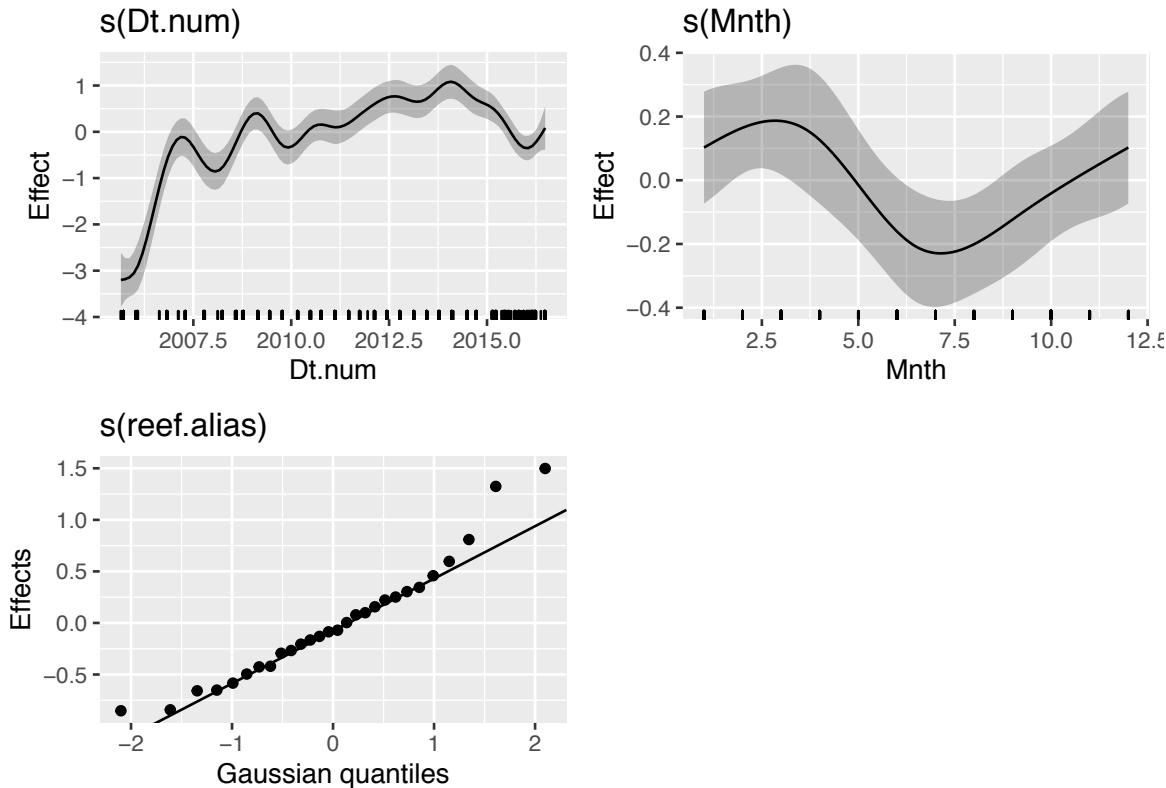
Let's check our model

```
k.check(wq.gamm3b)
```

```
##          k'      edf  k-index p-value
## s(Dt.num)   19 14.941010 0.7621963 0.0000
## s(Mnth)      3  2.027341 0.8520794 0.0375
## s(reef.alias) 28 22.854635       NA       NA
```

The k check values still don't look awesome, so we might fit a third model.

```
draw(wq.gamm3b)
```



Model 3c with region partition We might wanna partition our model for different regions separately, so let's define it in the model

```
wq.gamm3c<-gam(NOx~s(Dt.num, by=Region, k=20)+
  s(Mnth,bs="cc",by=Region,k=5)+
  s(reef.alias,bs="re"),
```

```

knots=list(Mnth=seq(1,12,length=5)),
data=wq,family=Gamma(link="log"), method="REML")

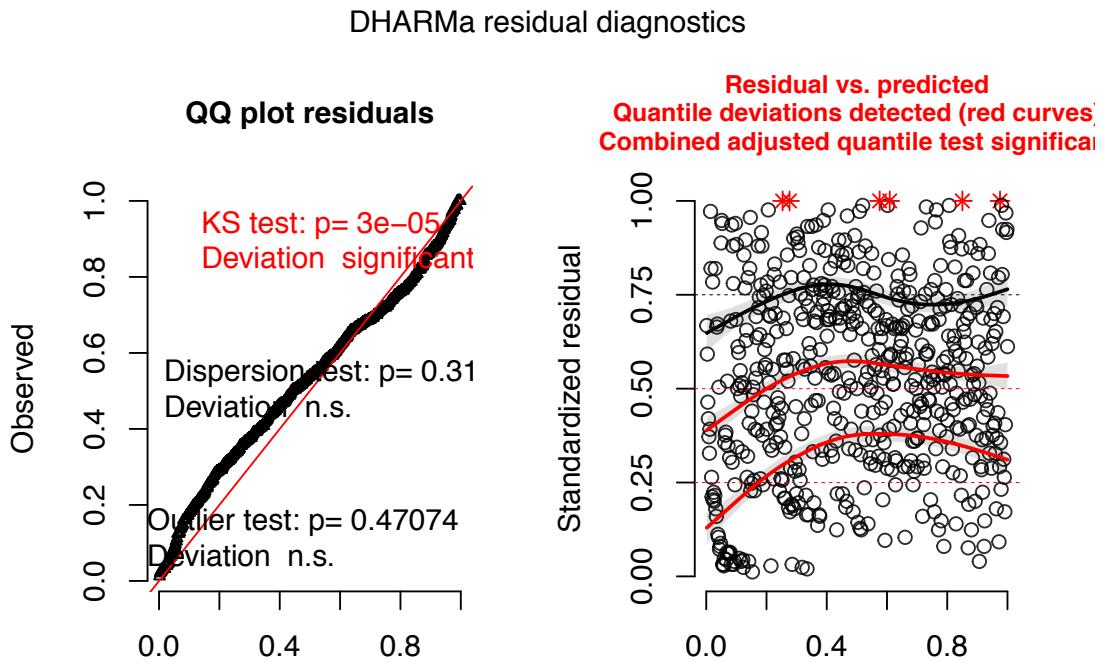
k.check(wq.gamm3c)

##                                     k'      edf   k-index p-value
## s(Dt.num):RegionBurdekin       19 6.886987e+00 0.7588018 0.0000
## s(Dt.num):RegionMackay Whitsunday 19 4.235635e+00 0.7588018 0.0000
## s(Dt.num):RegionWet Tropics     19 8.888705e+00 0.7588018 0.0000
## s(Mnth):RegionBurdekin        3 1.541850e+00 0.8727208 0.1500
## s(Mnth):RegionMackay Whitsunday 3 2.322221e+00 0.8727208 0.1300
## s(Mnth):RegionWet Tropics      3 6.590147e-04 0.8727208 0.1575
## s(reef.alias)                  28 2.241896e+01      NA      NA

```

The p values will be slightly different if we run it several times. The k strap does the bootstrap.

```
wq.resids<-DHARMa::simulateResiduals(wq.gamm3c, plot=T)
```

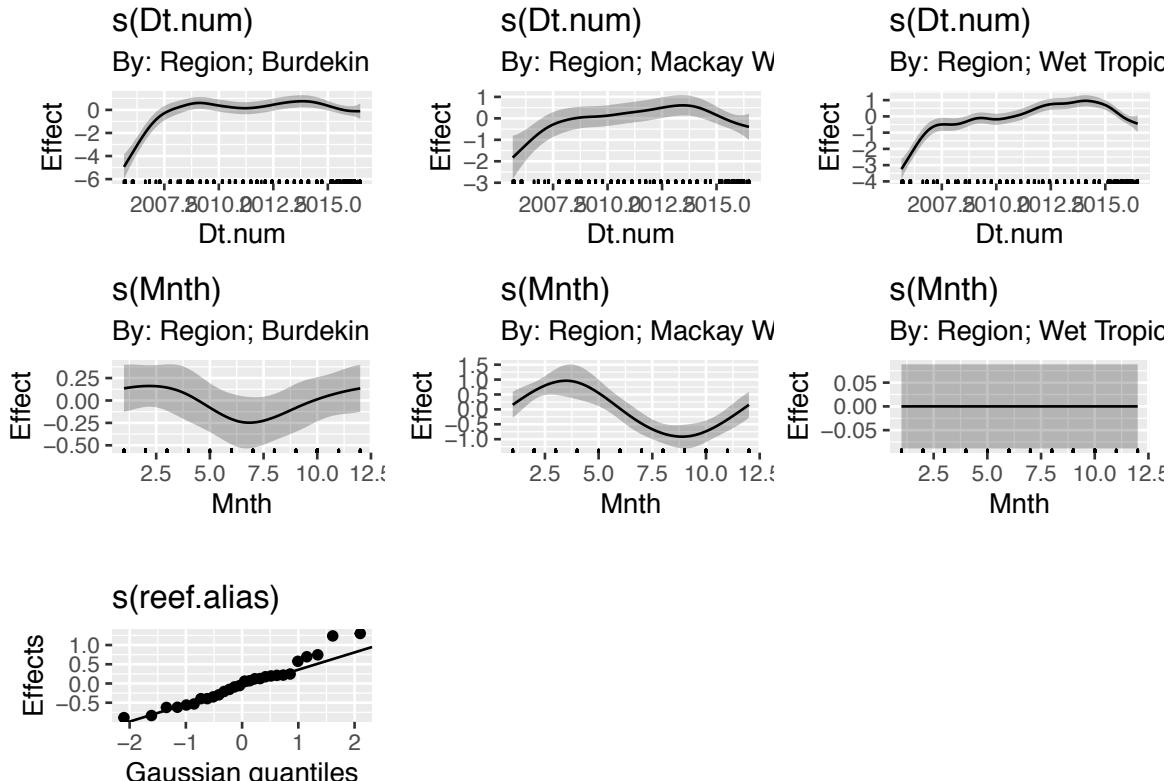


some problems here: - Smirnoff test has failed: Our QQ plot is not straight. BUT this is the least important assumption. It is like saying in a regression that normality and homogeneity of variance are equally important. THEY ARE NOT. Hom of var is far more important than normality. - There seems to be non-linearity in the residuals, which is normal as we are working with a non-linear model

We could now either: - switch to a log-normal distribution, which might do a better job to the full dataset - switch to a tweedy distribution (possibly this one is the better option) For the sake of the course, we are not gonna do this here as it won't change the conclusions. But if this was your data, you'd explore transforming the data first before continuing.

Partial plot of 3c

```
draw(wq.gamm3c)
```



Seasonality is not as evident in the wet tropics, and is most evident in Mackay. We can see that there has been a decline in Mackay and Wet tropics from about 2015.

(!) Ignore the initial increasing line, as it is an artifact (!)

Model summary of 3c

Let's have a look at the output

```
summary(wq.gamm3c)
```

```
##
## Family: Gamma
## Link function: log
##
## Formula:
## NOx ~ s(Dt.num, by = Region, k = 20) + s(Mnths, bs = "cc", by = Region,
##       k = 5) + s(reef.alias, bs = "re")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.1199    0.1464   0.819   0.413
##
## Approximate significance of smooth terms:
##                               edf Ref.df      F p-value
## s(Dt.num):RegionBurdekin     6.886987 8.496 11.926 < 2e-16 ***
```

```

## s(Dt.num):RegionMackay Whitsunday 4.235635 5.263 3.620 0.0027 **
## s(Dt.num):RegionWet Tropics      8.888705 10.930 18.653 < 2e-16 ***
## s(Mnth):RegionBurdekin        1.541850 3.000 1.099 0.1107
## s(Mnth):RegionMackay Whitsunday 2.322221 3.000 9.877 2.65e-07 ***
## s(Mnth):RegionWet Tropics      0.000659 3.000 0.000 0.7082
## s(reef.alias)                  22.418958 27.000 5.265 < 2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.243 Deviance explained = 47.6%
## -REML = 684.18 Scale est. = 1.1562 n = 568

```

These are each of our smoothers. R tells us which ones are sign wiggly, and which aren't. Approximate significance of smooth terms: edf Ref.df F p-value

```

s(Dt.num):RegionBurdekin 6.886987 8.496 11.926 < 2e-16 s(Dt.num):RegionMackay Whitsunday
4.235635 5.263 3.620 0.0027 s(Dt.num):RegionWet Tropics 8.888705 10.930 18.653 < 2e-16
s(Mnth):RegionBurdekin 1.541850 3.000 1.099 0.1107
s(Mnth):RegionMackay Whitsunday 2.322221 3.000 9.877 2.65e-07 s(Mnth):RegionWet Tropics
0.000659 3.000 0.000 0.7082
s(reef.alias) 22.418958 27.000 5.265 < 2e-16 **

```

Long term trends (Dt.num) -> All three regions (Burdeking, Mackay and Wet tropics) are wiggly.

Short term trends (Mnth) -> The Burdekin and Wet Tropics, once you account for uncertainty, could be very well a straight line.

Describing changes between years

What if we wanted to show how much NOx has changed between 2014 and 2016?

```

emmeans(wq.gamm3c,
         pairwise~Dt.num, # for comparing the two years, 2014 and 2016
         at=list(Dt.num=c(2014,2016)),
         type="response") %>%
confint

## NOTE: A nesting structure was detected in the fitted model:
##       reef.alias %in% Region

## $emmeans
##   Dt.num response    SE  df lower.CL upper.CL
##   2014     2.355 0.406 521     1.678     3.31
##   2016     0.933 0.115 521     0.732     1.19
##
## Results are averaged over the levels of: reef.alias, Region
## Confidence level used: 0.95
## Intervals are back-transformed from the log scale
##
## $contrasts
##   contrast    ratio    SE  df lower.CL upper.CL
##   2014 / 2016 2.52 0.475 521     1.74     3.65
##
## Results are averaged over the levels of: reef.alias, Region
## Confidence level used: 0.95
## Intervals are back-transformed from the log scale

```

- response: 2014 was higher in 2016 => There is a 60% decline (2.355 -> 0.933).

```

emmeans(wq.gamm3c,pairwise~Dt.num|Region, at=list(Dt.num=c(2014,2016)),
        type="response") %>%
confint

## NOTE: A nesting structure was detected in the fitted model:
##      reef.alias %in% Region

## $emmeans
## Region = Burdekin:
##   Dt.num response    SE df lower.CL upper.CL
##   2014     1.923 0.634 521     1.006     3.67
##   2016     0.840 0.195 521     0.533     1.32
##
## Region = Mackay Whitsunday:
##   Dt.num response    SE df lower.CL upper.CL
##   2014     2.421 0.825 521     1.239     4.73
##   2016     1.080 0.276 521     0.653     1.79
##
## Region = Wet Tropics:
##   Dt.num response    SE df lower.CL upper.CL
##   2014     2.805 0.538 521     1.925     4.09
##   2016     0.895 0.118 521     0.690     1.16
##
## Results are averaged over the levels of: reef.alias
## Confidence level used: 0.95
## Intervals are back-transformed from the log scale
##
## $contrasts
## Region = Burdekin:
##   contrast ratio    SE df lower.CL upper.CL
##   2014 / 2016 2.29 0.863 521     1.09     4.80
##
## Region = Mackay Whitsunday:
##   contrast ratio    SE df lower.CL upper.CL
##   2014 / 2016 2.24 0.763 521     1.15     4.37
##
## Region = Wet Tropics:
##   contrast ratio    SE df lower.CL upper.CL
##   2014 / 2016 3.13 0.755 521     1.95     5.03
##
## Results are averaged over the levels of: reef.alias
## Confidence level used: 0.95
## Intervals are back-transformed from the log scale

```

The \$contrasts tell us that 2014 was... - Burdekin region: 2.29x higher than 2016 (you can double check this in the response columns [=predicted log values] above) - Whitsunday region: 2.24x higher than 2016 (you can double check this in the response columns [=predicted log values] above) - Wet tropics region: 3.13x higher than 2016 (you can double check this in the response columns [=predicted log values] above)

Further analyses

Summary figures

References

13_GAM_maps

Sara Kophamel

14/12/2020

Open: gam_example4

Preparations

Load the necessary libraries

```
library(mgcv)      #for GAMs
library(gratia)    #for GAM plots
library(emmeans)   #for marginal means etc
library(broom)     #for tidy output
library(MuMIn)    #for model selection and AICc
library(lubridate) #for processing dates
library(mapdata)
library(maps)
library(rgeos) # for downloading maps
library(tidyverse) #for data wrangling
library(DHARMa)   #for residual diagnostics
library(performance)
library(see)
library(sf) # represents simple features as native R objects
library(stars) # for rasters (sth where you have a colour representing sth in
each x-y cell)
library(rnaturalearth) # to extract major geographical datasets from a
particular server (eg. a map of a country). Some of them are preloaded in
the rnaturalearthdata package (below)
library(rnaturalearthdata)
library(raster)
library(ggspatial) # implements arrows and error bars on maps
library(patchwork)
```

Scenario

@Paruelo-1996-1212 analyzed the geographic distribution and the effects of climate variables on the relative abundance of a number of plant functional types (PFT's) including shrubs, forbs, succulents (e.g. cacti), C3 grasses and C4 grasses. They used data from 73 sites across temperate central North America (see pareulo.csv) and calculated the relative abundance of C3 grasses at each site as a response variable



grass

Format of paruelo.csv data file

C3	LAT	LONG	MAP	MAT	JJAMAP	DJFMAP
...
C3	- Relative abundance of C3 grasses at each site - response variable					
LAT	- Latitudinal coordinate					
LONG	- Longitudinal coordinate					
MAP	- Mean annual precipitation					
MAT	- Mean annual temperature					
JJAMAP	- Mean annual precipitation in June, July, August					
DJFMAP	- Mean annual precipitation in December, January, February					

Read in the data

```
paruelo = read_csv('data/paruelo.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   C3 = col_double(),
##   LAT = col_double(),
##   LONG = col_double(),
##   MAP = col_double(),
##   MAT = col_double(),
##   JJAMAP = col_double(),
```

```
##   DJFMAP = col_double()
## )

glimpse(paruelo)

## #> Rows: 73
## #> Columns: 7
## #> $ C3      <dbl> 0.65, 0.65, 0.76, 0.75, 0.33, 0.03, 0.00, 0.02, 0.05, 0.05,
## #> 0...
## #> $ LAT     <dbl> 46.40, 47.32, 45.78, 43.95, 46.90, 38.87, 32.62, 36.95,
## #> 35.30, ...
## #> $ LONG    <dbl> 119.55, 114.27, 110.78, 101.87, 102.82, 99.38, 106.75,
## #> 96.55, ...
## #> $ MAP     <dbl> 199, 469, 536, 476, 484, 623, 259, 969, 542, 421, 446, 376,
## #> 66...
## #> $ MAT     <dbl> 12.4, 7.5, 7.2, 8.2, 4.8, 12.0, 14.5, 15.3, 13.9, 8.5, 5.1,
## #> 11...
## #> $ JJAMAP <dbl> 0.12, 0.24, 0.24, 0.35, 0.40, 0.40, 0.47, 0.30, 0.44, 0.31,
## #> 0...
## #> $ DJFMAP <dbl> 0.45, 0.29, 0.20, 0.15, 0.14, 0.11, 0.17, 0.14, 0.13, 0.14,
## #> 0...
```

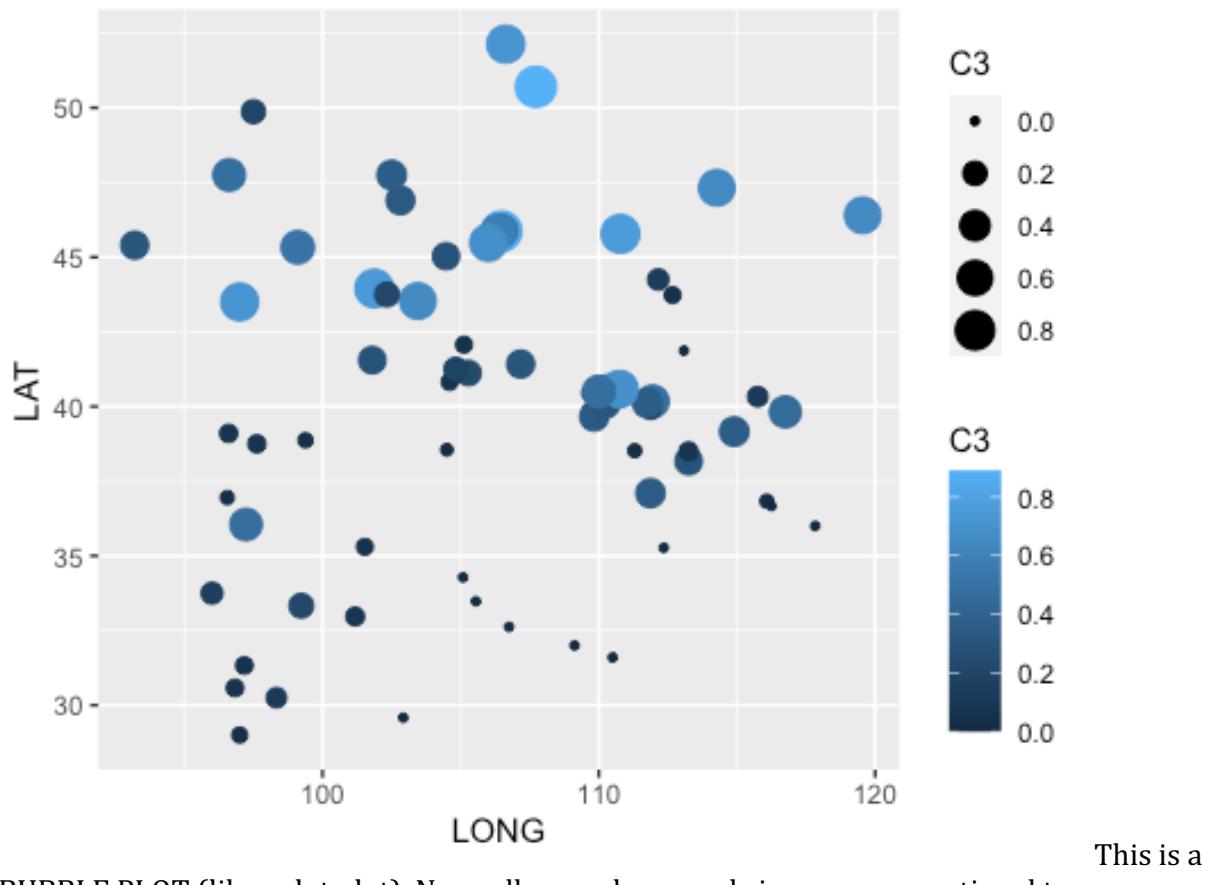
We don't have any categorical data to transform into factors, so we don't worry about that.

Exploratory data analysis

Option 1 - bubble plot

We could just do a scatterplot

```
ggplot(paruelo, aes(y=LAT, x=LONG, color=C3, size=C3))+
  geom_point()
```



This is a BUBBLE PLOT (like a dot plot). Now, all our colours and sizes are proportional to our response (Abundance of C3 grasses). At a first glimpse, there seem to be more C3 grasses at higher latitudes than at lower latitudes. It looks like there is a North-South gradient

Option 2 - map

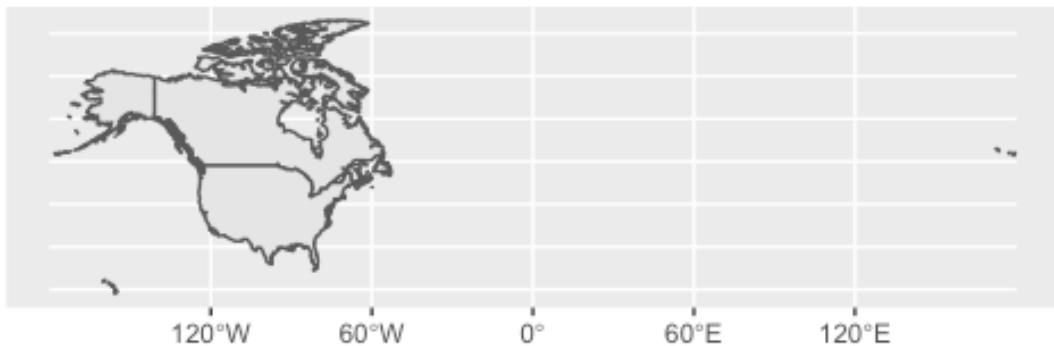
We will focus on the spatial components now.

Plotting the North America map

Because our data is from North America, let's get a map of US (+Canada) - package rgeos

```
usa<-ne_countries(country=c("united states of america","canada"),
                  scale="medium", returnclass="sf") # returnClass allows to
plot polygons (I think)

ggplot()+
  geom_sf(data=usa)
```



This plot has got all states for US and Canada, and therefore includes Hawaii and Guam. BUT we don't need that. So what we might do is to CROP this map to a particular range, removing the outer territories.

Cropping the North America map

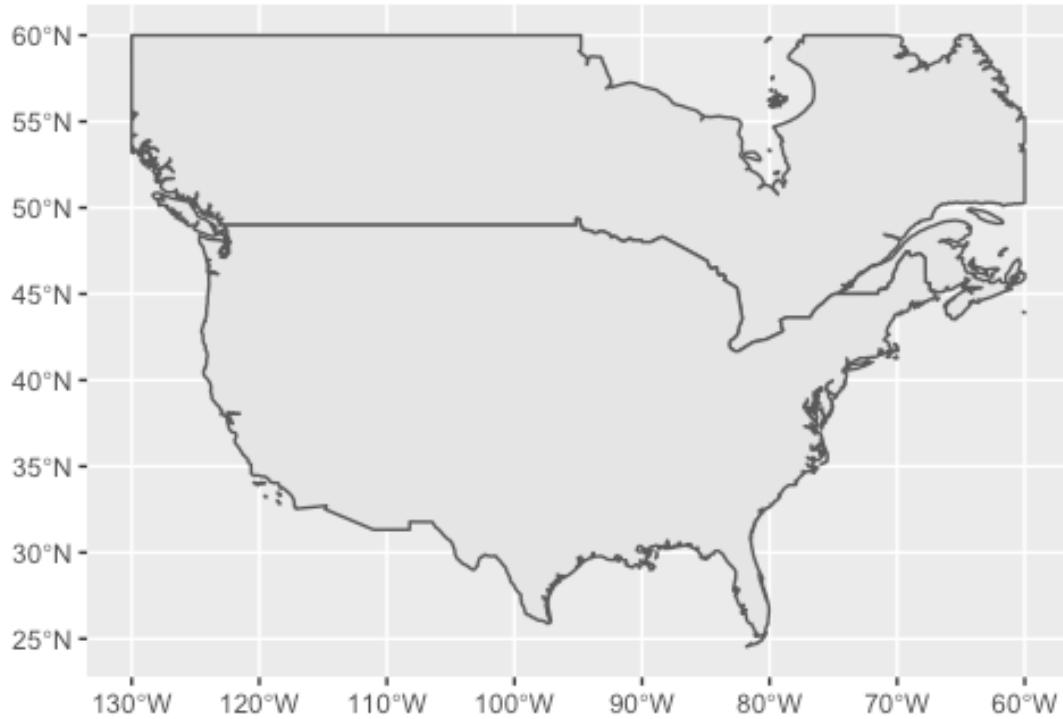
```
usa=usa %>%
  st_crop(xmin=-130, xmax=-60,
          ymin=0,ymax=60)

## although coordinates are longitude/latitude, st_intersection assumes that
## they are planar

## Warning: attribute variables are assumed to be spatially constant
## throughout all
## geometries
```

After cropping, we plot US again

```
ggplot(usa) +
  geom_sf()
```



The x and y axis ticks have been neatly formatted for you. R also ensures that the aspect ratio is EXACTLY right, even if we stretch the map.

Defining the coordinates and removing zeros

Now, we could plot our bubble plot on top to see where they sampled BUT the LAT and LON supplied were not in westernly method, which was not same latitude method than the US map (easterly method). So we first have to modify the coordinates.

```
paruelo=paruelo %>%
  mutate(mC3=ifelse(C3==0, 0.01, C3), # Turns any cover that is =0 into 0.01
(explanation below)
  LONG=-1*LONG) # transforms western into eastern coordinates
```

The distribution of grass in this dataset has been recorded as %. The prob with % cover as a variable is that the only suitable distribution is a BETA distribution, which does not define ANY mass at 0 or 1 = you would not be allowed to have covers with 0 or 100%. An alternative distribution would be BINOMIAL, which measures integers (number of successes out of the number of trials). So it is also NOT useful for % cover. If they had recorded the number of points in a quadrate (eg how many patches did the grass cover), you could have used a binomial distribution. We can't go back and reconstruct the data, so we will stick to a BETA distribution. So we have to tell R: If C3 is equal to 0, make it 0.01, otherwise keep it as it was.

```
paruelo
```

```
## # A tibble: 73 x 8
##      C3    LAT   LONG   MAP   MAT JJAMAP DJFMAP   mC3
##      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  0.65  46.4 -120.   199  12.4  0.12  0.45  0.65
## 2  0.65  47.3 -114.   469   7.5  0.24  0.290  0.65
## 3  0.76  45.8 -111.   536   7.2  0.24  0.2   0.76
## 4  0.75  44.0 -102.   476   8.2  0.35  0.15  0.75
## 5  0.33  46.9 -103.   484   4.8  0.4   0.14  0.33
## 6  0.03  38.9 -99.4  623   12   0.4   0.11  0.03
## 7  0     32.6 -107.   259  14.5  0.47  0.17  0.01
## 8  0.02  37.0 -96.6  969  15.3  0.3   0.14  0.02
## 9  0.05  35.3 -102.   542  13.9  0.44  0.13  0.05
## 10 0.05  40.8 -105.   421   8.5  0.31  0.14  0.05
## # ... with 63 more rows
```

There is nothing to tell R how to map the data points, no projection information, etc. WE HAVE TO TELL R THAT IT IS DEALING WITH GEOGRAPHICAL DATA:

We tell R what the coordinates are for LONG and LAT:

```
paruelo.sf=paruelo %>%
  st_as_sf(coords=c("LONG", "LAT"), crs=st_crs(usa)) # crs=st_crs is the
coordinate ref system that came with the us data
```

FYI - To check which coordinate system US used:

```
st_crs(usa)

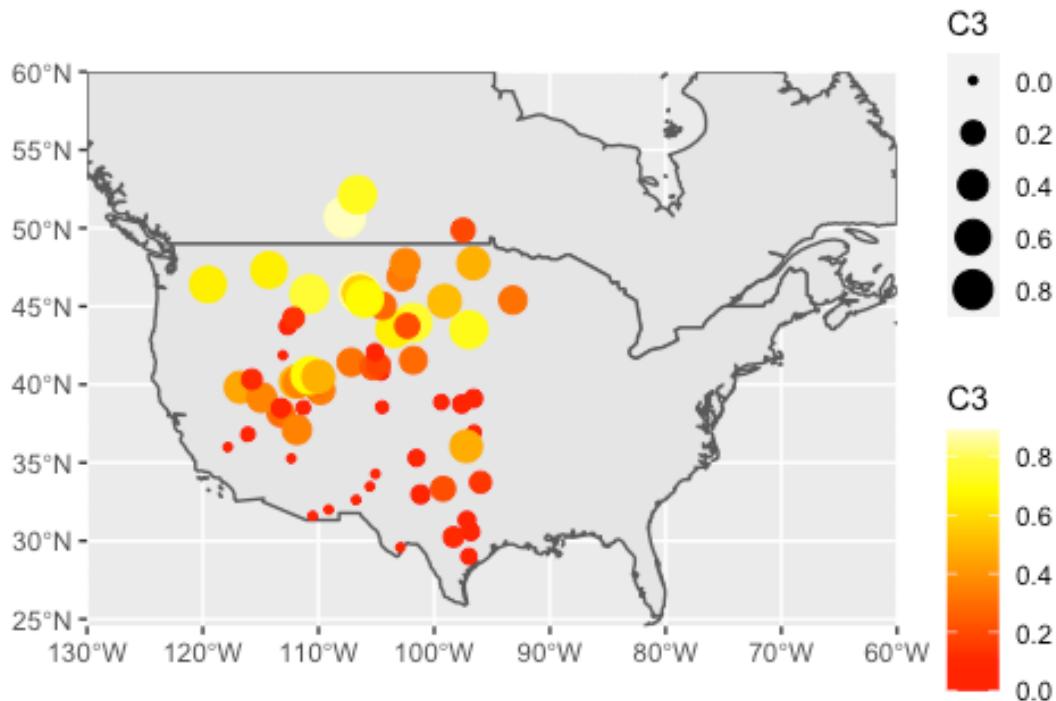
## Coordinate Reference System:
##   User input: +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
+towgs84=0,0,0
##   wkt:
## GEOGCS["WGS 84",
##       DATUM["WGS_1984",
##             SPHEROID["WGS 84",6378137,298.257223563,
##                   AUTHORITY["EPSG","7030"]],
##             AUTHORITY["EPSG","6326"]],
##       PRIMEM["Greenwich",0,
##              AUTHORITY["EPSG","8901"]],
##       UNIT["degree",0.0174532925199433,
##             AUTHORITY["EPSG","9122"]],
##       AUTHORITY["EPSG","4326"]]
```

Plotting the cropped North America map with the data on top

Now we can plot our data on the map

```
ggplot()+
  geom_sf(data=usa)+
  geom_sf(data=paruelo.sf,aes(color=C3,size=C3))+
```

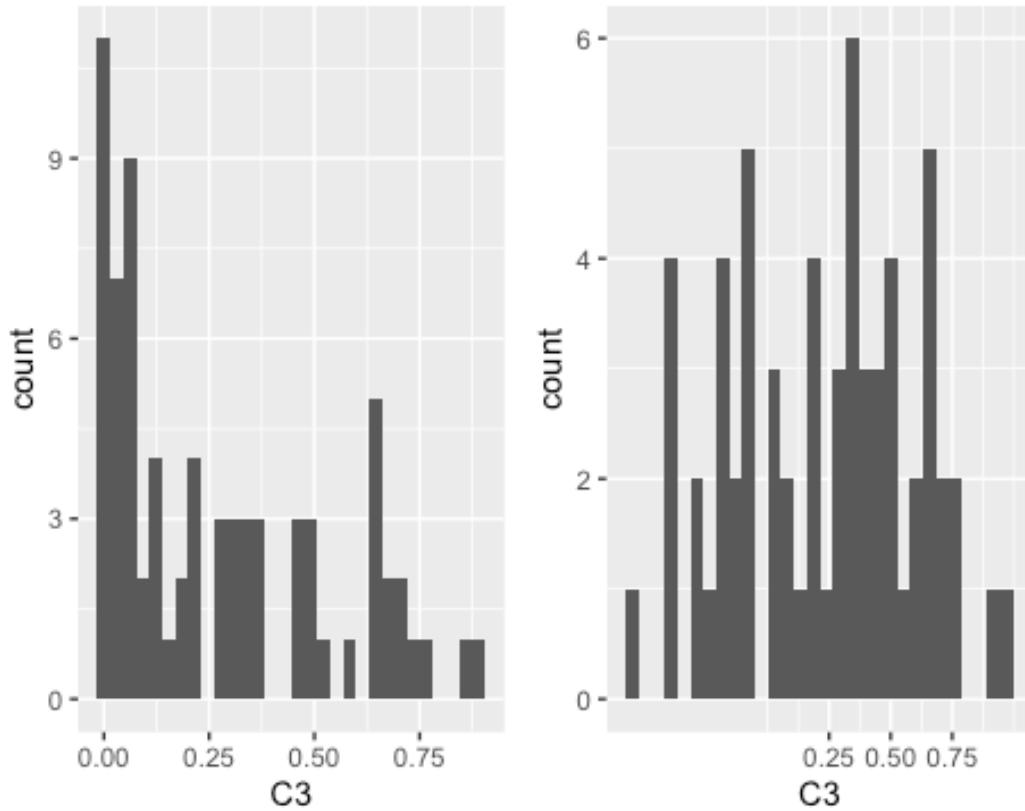
```
scale_color_gradientn(colors=heat.colors(n=10))+  
coord_sf(expand=FALSE)
```



We now managed to see where our data was collected from, and it looks like our researchers did not sample in the east. It also looks like in the north grasses are more widely distributed than in the south.

Checking for normality

```
ggplot(paruelo)+ geom_histogram(aes(x=C3))+  
ggplot(paruelo) +  
geom_histogram(aes(x=C3))+scale_x_continuous(trans=scales::logit_trans())  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## Warning: Transformation introduced infinite values in continuous x-axis  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## Warning: Removed 10 rows containing non-finite values (stat_bin).
```



Here, we have created histograms for both the raw data and the logit transformation of the x axis scale, to mimic what the beta distribution is going to be looking for (normally distributed data).

We can plot two plots together by using the package “patchwork”

Fit the model

Model formula:

$\$ \$ y_i \sim \mathcal{N}(\mu_i, \sigma^2) \\ \mu_i = \beta_0 + f(Long_i) + f(Long_i, Lat_i) \$ \$$

where β_0 is the y-intercept. $f(Lat)$ and $f(Long)$ indicate the additive smoothing functions of the spatial predictors.

We will have to define the smoother for the model. A smoother for LAT and one for LONG would be purely additive, without allowing interaction btw the two, which is not suitable for our data. It would assume a single dimension, and no spacial extent. There are three ways to define this:

Option 1 - with smoother s (AVOID)

```
paruelo.gam1<-gam(mC3~s(LONG, LAT), data=paruelo,
                      family=betar, method="REML" ) # family = betar => BETA distribution
```

Assumes that the wiggliness in LAT is the same than the wiggliness in LONG. It only accounts for wiggliness ONCE, which could be a problem. It also assumes that they are both on the same scale, which is the case here. BUT imagine we had scale and temp. We couldn't fit it then. In this case, we are consider this model not appropriate, because the wiggliness is unlikely gonna be the same for LAT and LONG

Option 2 - with tensor products te (OK, NOT IDEAL)

Instead of smoother, we use tensor products te. They assume that the wiggliness can be different across LAT and LONG. The scales also do not have to be the same; we could for example have scale and temp.

```
paruelo.gam2<-gam(mC3~te(LONG, LAT), data=paruelo,
                      family=betar, method="REML" ) # family = betar => BETA distribution
```

Option 3 - with more flexible tensor products ti (TAKE THIS ONE)

This model fits tensor prod for LAT, one for LONG, and one for the tensor prod of the interaction btw the two.

```
paruelo.gam3<-gam(mC3~ti(LONG)+ti(LAT)+ti(LONG,LAT), data=paruelo,
                      family=betar, method="REML" ) # family = betar => BETA distribution
```

With this model, we will be able to assess whether the LAT change is consistent across LONG = Is there a need for the ti. It also assumes that the wiggliness can be different across LAT and LONG. In this case, the scales also do not have to be the same.

Model check - gam3

k.check

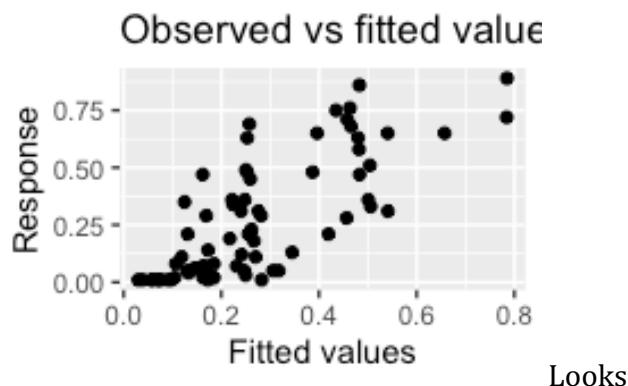
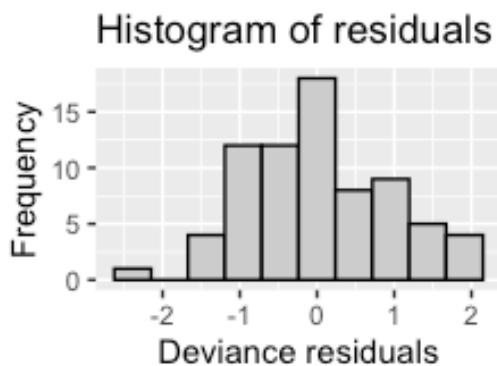
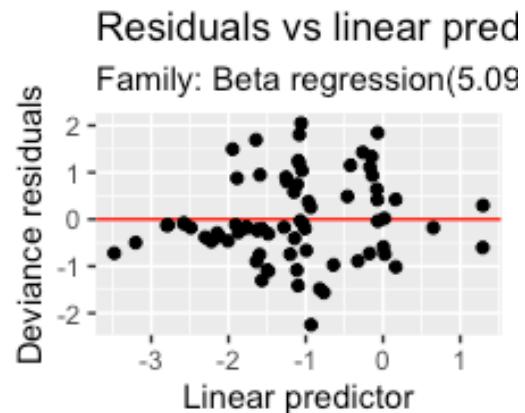
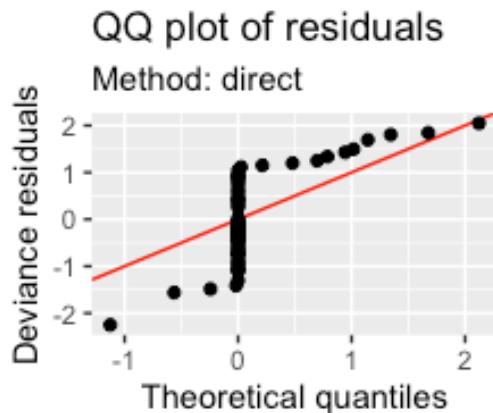
```
k.check(paruelo.gam3)
```

```
##          k'      edf  k-index p-value
## ti(LONG)    4 1.000019  0.8673167  0.1150
## ti(LAT)     4 1.000024  0.9588562  0.3375
## ti(LONG,LAT) 16 3.523281  0.9956490  0.4075
```

The interaction model is not massively constrained, and nor are LAT or LONG.

appraise

```
appraise(paruelo.gam3)
```



ok

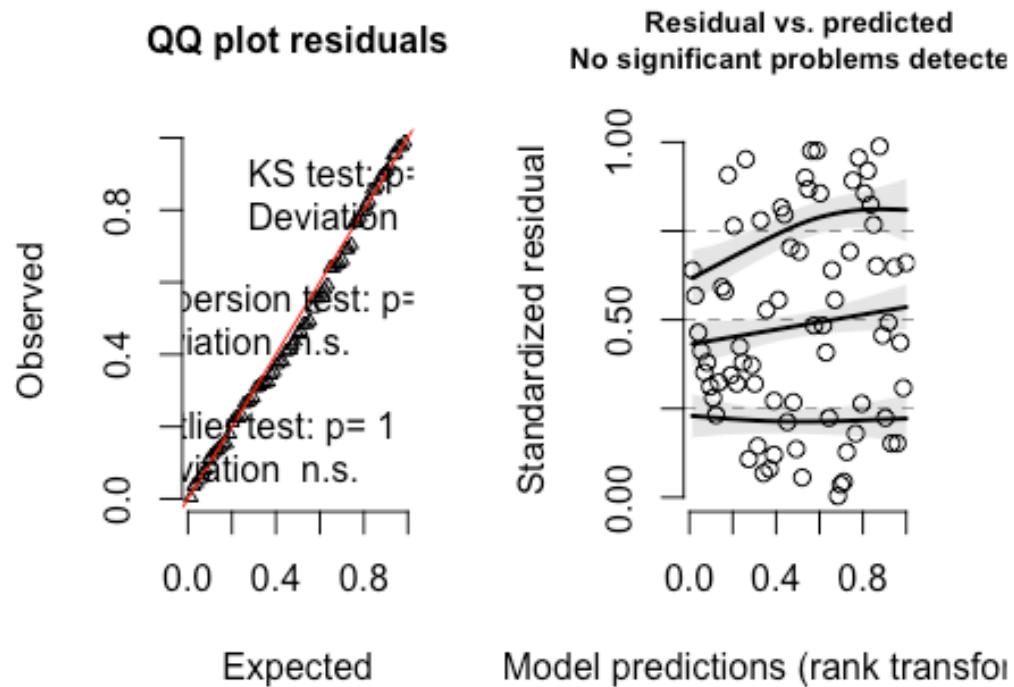
DHARMA residuals

Unfortunately, DHARMA residuals was not created for BETA models. So we have to define them ourselves

```
paruelo.resids<-
  createDHARMA(simulatedResponse=simulate(paruelo.gam3,nsim=250),
               observedResponse=paruelo$mC3,
               fittedPredictedResponse = predict(paruelo.gam3))

plot(paruelo.resids)
```

DHARMA residual diagnostics



Now we can check the DHARMA residuals, and they look fantastic.

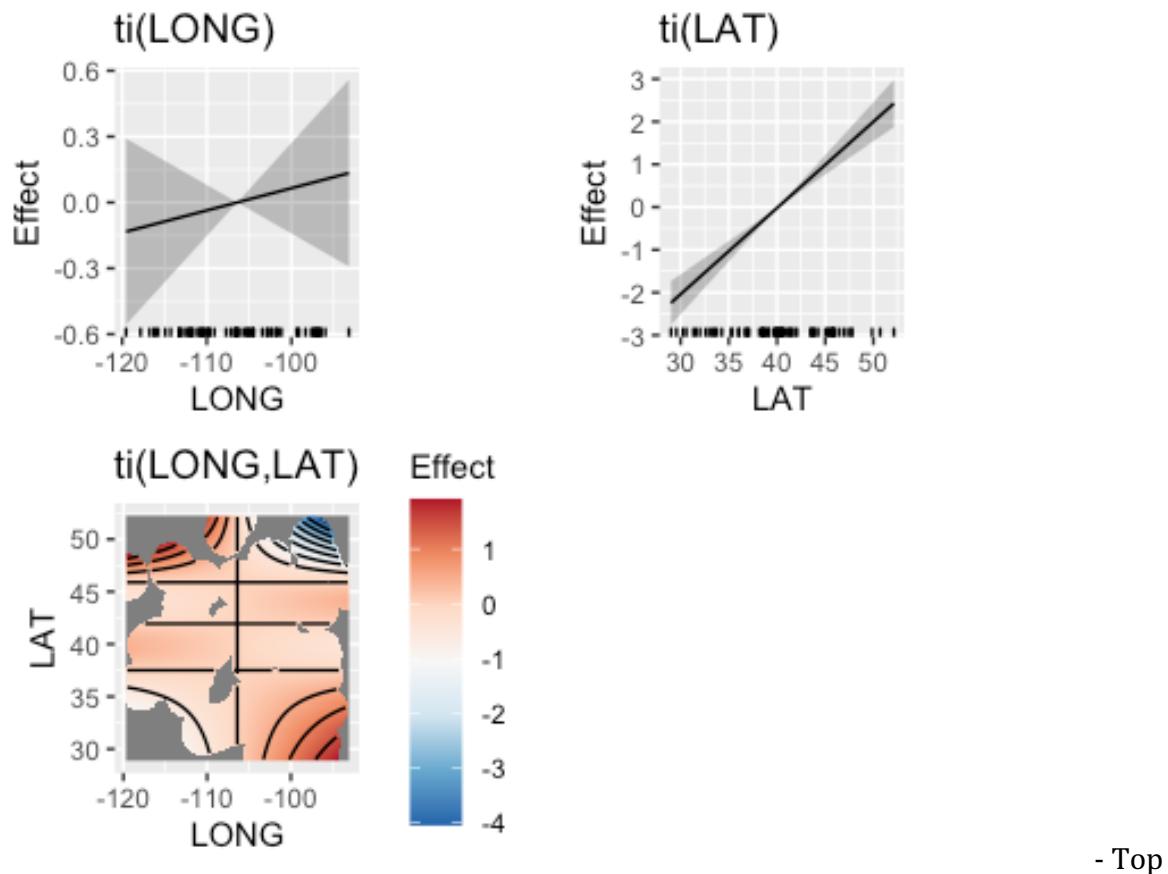
Once we have looked at the model and the residuals, we can continue with our partial plots.

Partial plots

Option 1

```
draw(paruelo.gam3)
```

```
## Warning: Removed 2315 rows containing non-finite values (stat_contour).
```

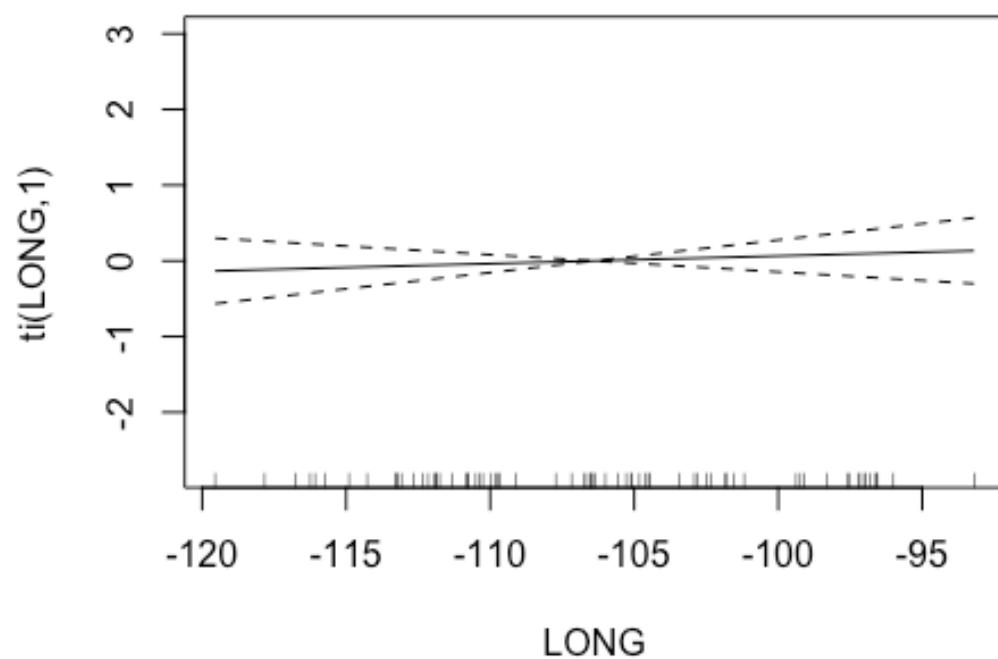


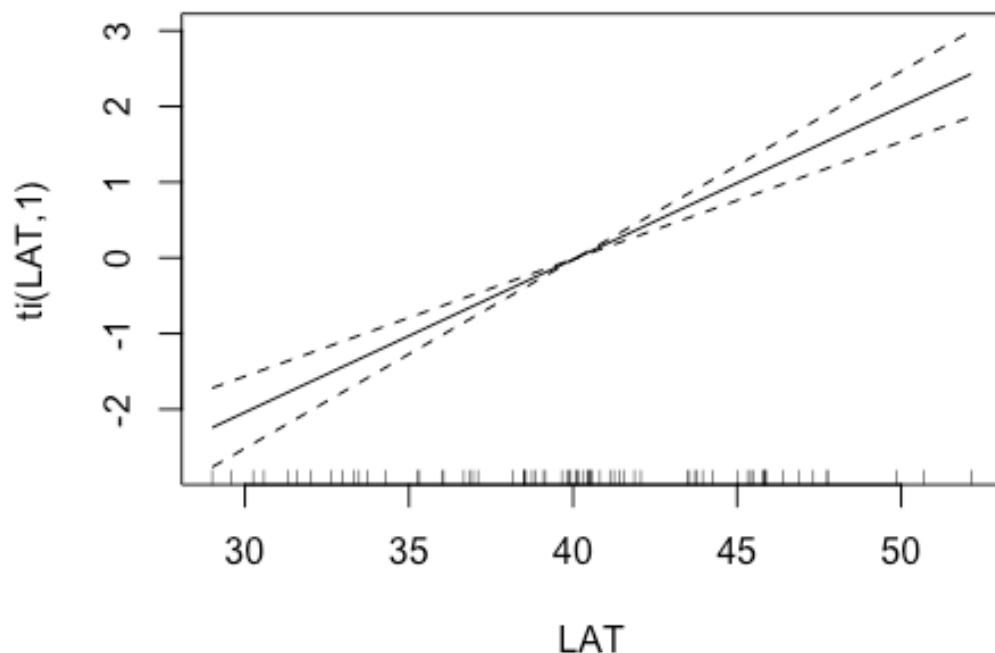
- Top

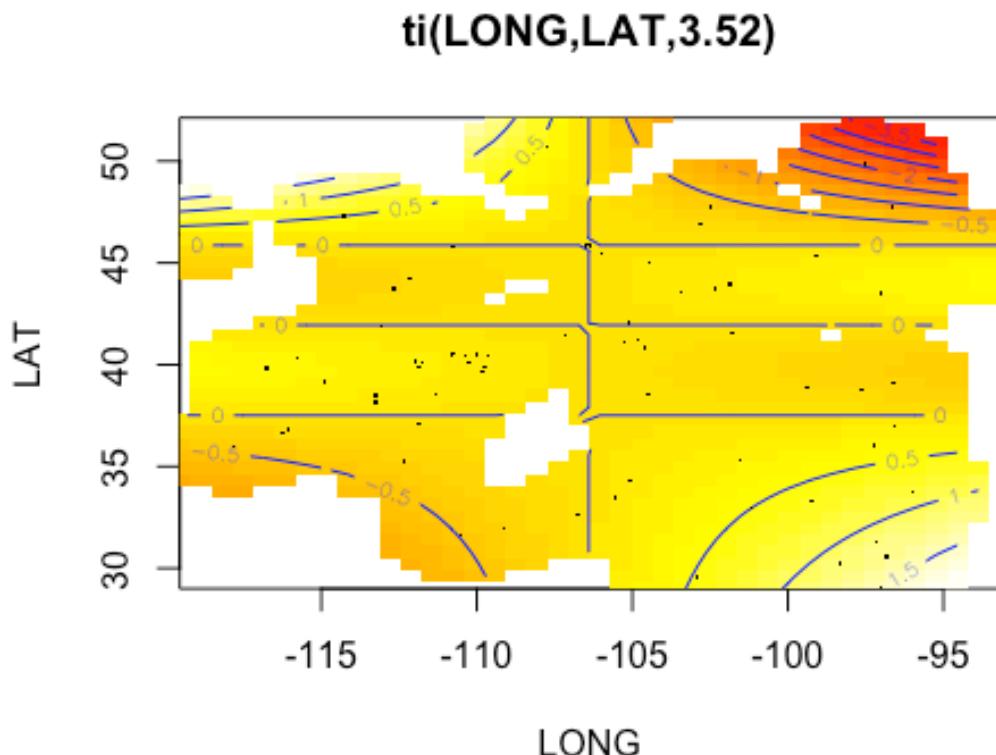
left plot: At the avg latitude, the top left panel reflects the LONG effect. When LAT is at its avg, the left plot shows what the LONG plot looks like, a slight positive response. That means that you get slightly more response (density cover) at the east than at the left, at the AVERAGE LAT. - Top right plot: At the avg LONG, abundance of C3 increases STRONGLY from south to north - Bottom left: Shows the interaction btw LAT and LONG; how it deviates from purely additive. If LAT was persistent across all LONG, the bottom panel should be just one colour, without interaction, without change. BUT we can see that although there is apparently a strong gradient along LAT (top right corner), it is less obvious/positive than at lower LAT (bottom right corner). In fact, it might well be close to negative. The gradient of C3 plants is more extreme at lower LONG than it is at higher LONG. - The grey areas are uncertain areas, that R could not calculate (due to not enough samples there, I guess)

Option 2 (=Option 1, but less pretty)

```
plot(paruelo.gam3, scheme=2)
```





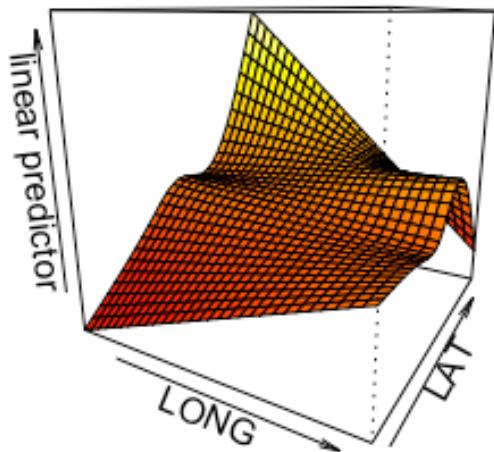


is the same as Option 2, but less pretty.

This plot

Option 3: Provides degrees of rotation

```
vis.gam(paruelo.gam3,theta=30)
```



This plot allows us to visualise the interaction. If there was NO interaction, this plot would look like a sheet of paper (could also be tilted). The fact that it shows peaks and drops = interaction. The LAT effect accelerates/increases at lower LONG, compared to what it is in the middle. At lower LONG, the gradient of C3 plants is more extreme than it is at higher LONG.

```
summary(paruelo.gam3)

##
## Family: Beta regression(5.099)
## Link function: logit
##
## Formula:
## mC3 ~ ti(LONG) + ti(LAT) + ti(LONG, LAT)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.2330    0.1151 -10.71 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##             edf Ref.df Chi.sq p-value
## ti(LONG)     1.000 1.000  0.378 0.53843
## ti(LAT)      1.000 1.000 73.604 < 2e-16 ***
```

```
## ti(LONG,LAT) 3.523 3.875 15.070 0.00294 **
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.535 Deviance explained = 63.6%
## -REML = -52.2 Scale est. = 1 n = 73
```

Predictions

We have to define a sequence for LAT and LONG:

```
paruelo.list=with(paruelo,
  list(LAT=seq(min(LAT), max(LAT), len=100),
       LONG=seq(min(LONG), max(LONG), len=100)))

newdata=emmeans(paruelo.gam3, ~LONG+LAT, at=paruelo.list, type="response") %>%
  as.data.frame

newdata %>% head

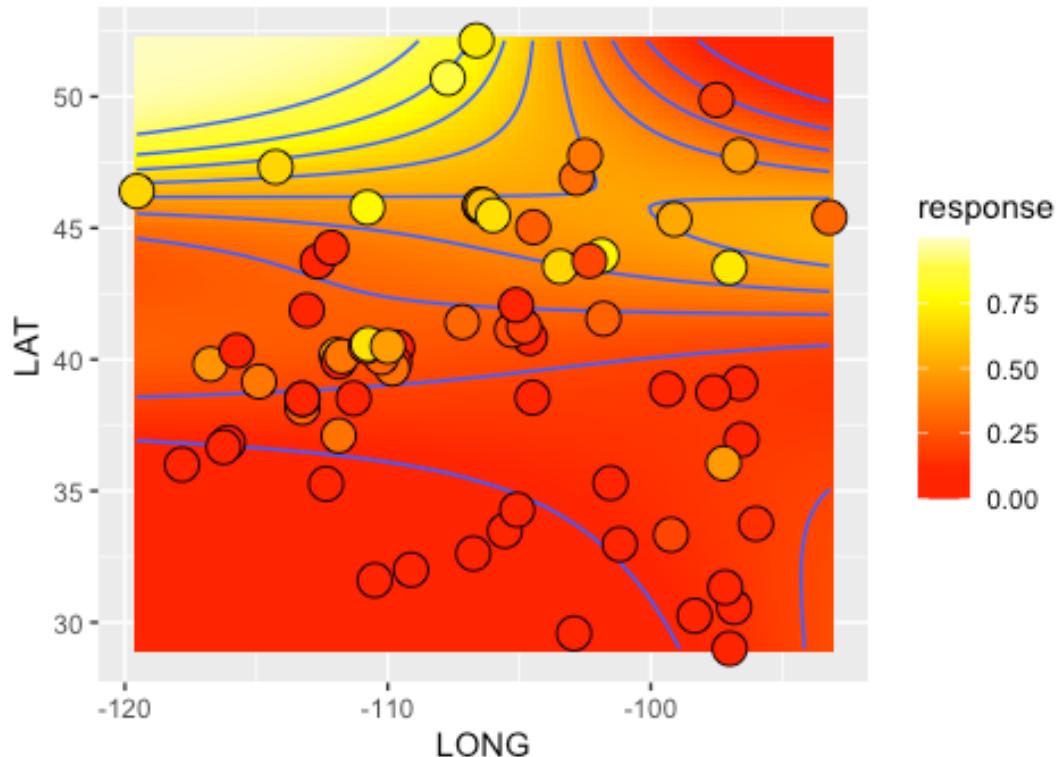
## #> #>   LONG LAT response      SE      df lower.CL upper.CL
## #> 1 -119.5500 29 0.003275951 0.003829316 66.47668 0.0003161297 0.03303197
## #> 2 -119.2838 29 0.003427577 0.003940068 66.47668 0.0003437871 0.03325299
## #> 3 -119.0177 29 0.003586196 0.004052898 66.47668 0.0003738497 0.03347661
## #> 4 -118.7515 29 0.003752128 0.004167767 66.47668 0.0004065246 0.03370294
## #> 5 -118.4854 29 0.003925707 0.004284626 66.47668 0.0004420362 0.03393206
## #> 6 -118.2192 29 0.004107283 0.004403422 66.47668 0.0004806279 0.03416408
```

Plot

Option 1 (AVOID)

We could now go and plot these results without spatial considerations:

```
ggplot(newdata, aes(y=LAT, x=LONG))+
  geom_tile(aes(fill=response))+
  geom_contour(aes(z=response))+
  scale_fill_gradientn(colors=heat.colors(10))+
  geom_point(data=paruelo, aes(fill=C3), shape=21, size=5)+
  coord_equal()
```

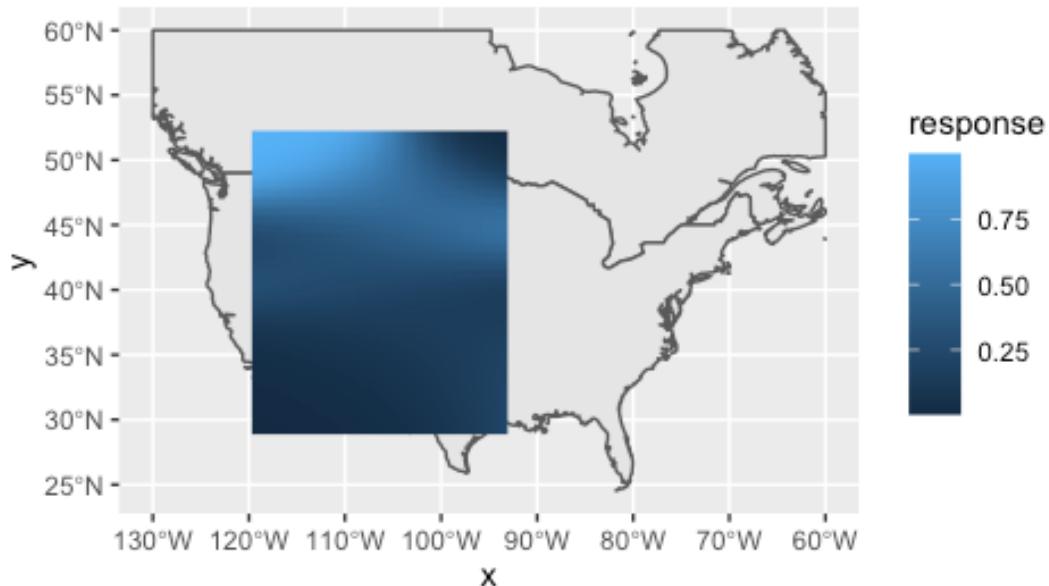


This plot has got the prediction along x and y coordinates, but they are not LAT and LONG. For every coordinate, we have a colour. We can turn this into a real raster which has got coordinate information.

Let's convert our matrix/dataframe/tibble into a raster. You need to define x, y and z when you do that. ## Option 2 (TAKE THIS ONE)

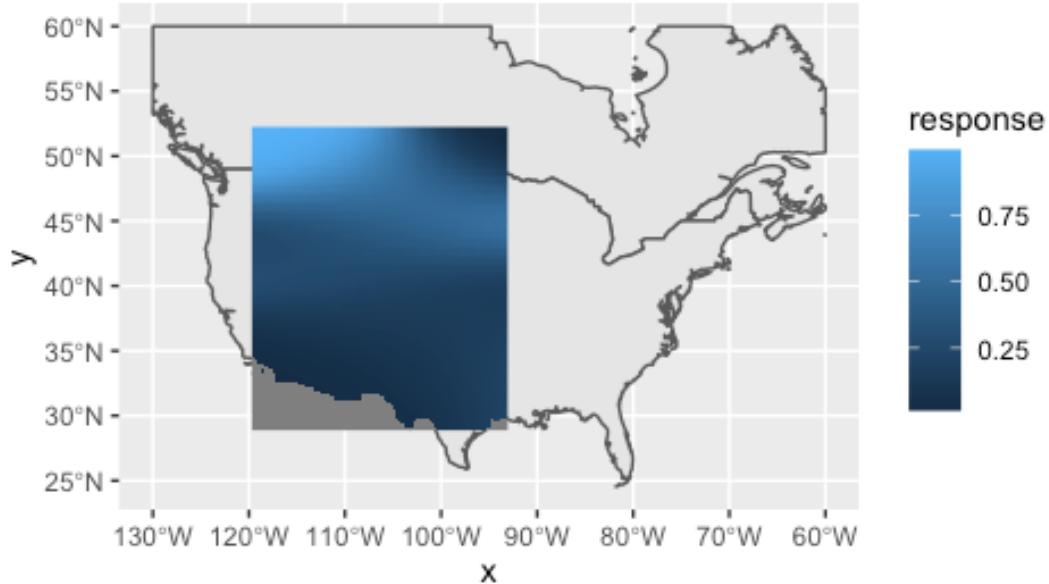
```
newdata.sf<-rasterFromXYZ(newdata[,c("LONG","LAT","response")]) %>% #define
x, y and z
st_as_sf() %>%
st_set_crs(st_crs(usa))

ggplot()+
  geom_sf(data=usa)+
  geom_stars(data=newdata.sf)
```



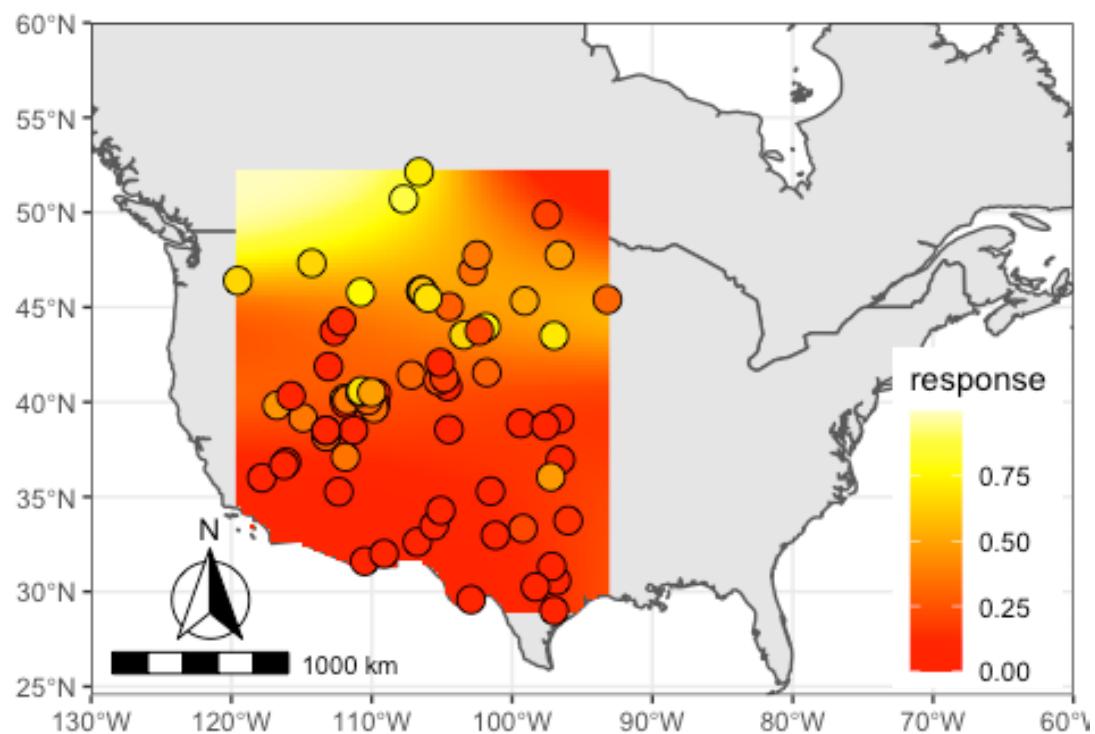
We got some issues here, as the country boundaries have not been taken into account. Let's try again:

```
newdata.sf<-rasterFromXYZ(newdata[,c("LONG","LAT","response")]) %>% #define  
x, y and z  
mask(usa) %>% # we tell R to take the borders of USA into account  
st_as_sf() %>%  
st_set_crs(st_crs(usa))  
  
ggplot() +  
  geom_sf(data=usa) +  
  geom_stars(data=newdata.sf)
```



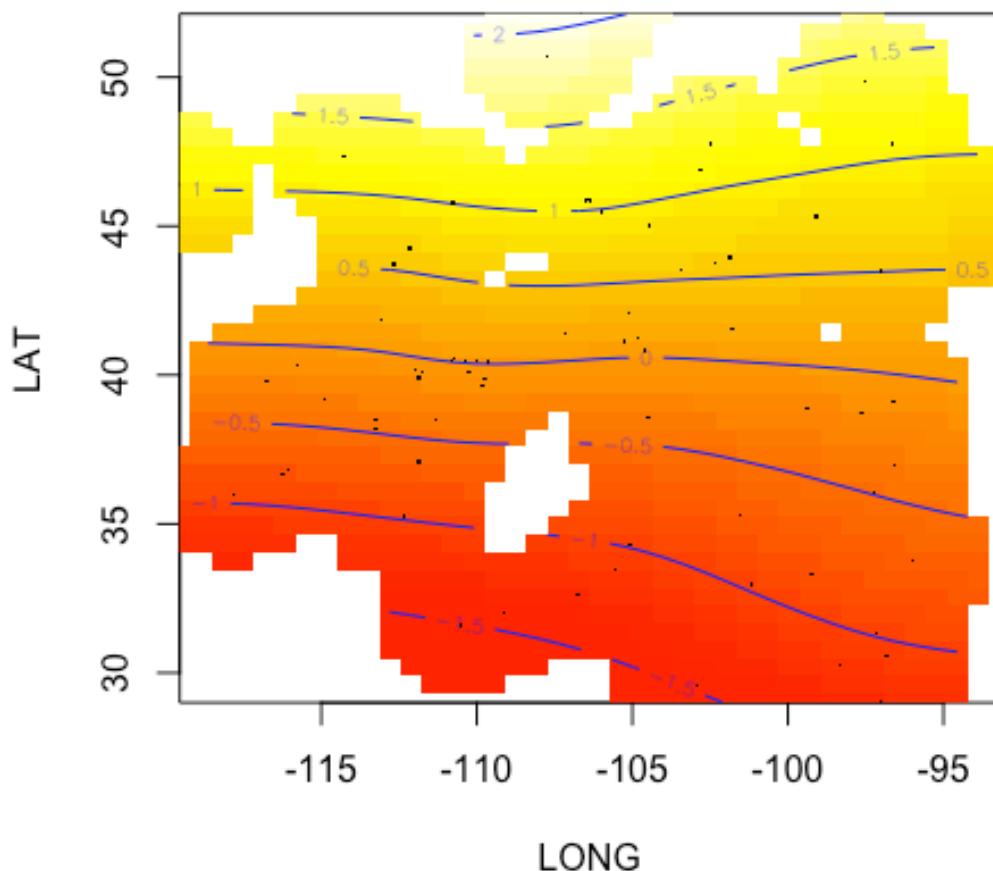
Now, we can see the predicted distribution of grass ("response") in the defined LAT and LON area. We can make this plot a big prettier now:

```
ggplot()+
  geom_sf(data=usa)+
  geom_stars(data=newdata.sf)+
  scale_fill_gradientn(colours=heat.colors(10),na.value=NA)+
  geom_sf(data=paruelo.sf, aes(fill=C3), shape=21, size=4)+
  annotation_scale(location="bl",width_hint=0.25)+ # adds a short ("width")
  scale
  annotation_north_arrow(location="bl",which_north="true",
                          pad_x=unit(0.25,"in"),pad_y=unit(0.2,"in"),
                          style=north_arrow_fancy_orienteering)+ # adds a cool
  arrow pointing north
  coord_sf(expand=FALSE)+
  theme_bw()+
  theme(axis.title=element_blank(), # gets rid of the axis titles
        legend.position=c(0.99, 0.01), legend.justification=c(1,0)) # moves
  the legend to the bottom right corner
## Warning: Removed 839 rows containing missing values (geom_raster).
## Scale on map varies by more than 10%, scale bar may be inaccurate
```



```
plot  
plot(paruelo.gam1, scheme = 2)
```

s(LONG,LAT,4.85)



THEORY: EXTRA INFO FOR GAMS

- For very large datasets, GAM functions will run VERY slowly and will frustrate you a lot. The function bam (big additive models) also uses GAM models, but is designed for large datasets. So the only change you'd have to do is to change the g for a b
- For mapping: soapsmoothers and soapsplines are good for other situations too
- In the mcdp (?) package you can run these models as Bayesian, but Murray does not recommend it, as they run VERY slow. For Bayesian, use the packages: rstanarm, broom mixed, tidybayes, bayesplot, ggmc, coda

```
library("rstanarm")
```

```
## Loading required package: Rcpp
## This is rstanarm version 2.21.1
```

```
## - See https://mc-stan.org/rstanarm/articles/priors for changes to default
priors!

## - Default priors may change, so it's safest to specify priors, even if
equivalent to the defaults.

## - For execution on a local, multicore CPU with excess RAM we recommend
calling

##   options(mc.cores = parallel::detectCores())

##
## Attaching package: 'rstanarm'

## The following object is masked from 'package:performance':
##
##     pp_check

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

library("broom.mixed")

## Registered S3 method overwritten by 'broom.mixed':
##   method      from
##   tidy.gamlss broom

library("tidybayes")

##
## Attaching package: 'tidybayes'

## The following objects are masked from 'package:gratia':
##
##     fitted_samples, predicted_samples

library("bayesplot")

## This is bayesplot version 1.7.2

## - Online documentation and vignettes at mc-stan.org/bayesplot

## - bayesplot theme set to bayesplot::theme_default()
##   * Does _not_ affect other ggplot2 plots
##   * See ?bayesplot_theme_set for details on theme setting

##
## Attaching package: 'bayesplot'
```

```

## The following object is masked from 'package:performance':
##
##     pp_check

library("ggmcmc")

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

library("coda")

# checking if rstanarm works:
stan_glm(y~x,data=data.frame(x=1:10,y=rnorm(10,0,1)))

##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 9.4e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would
## take 0.94 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration: 1 / 2000 [  0%] (Warmup)
## Chain 1: Iteration: 200 / 2000 [ 10%] (Warmup)
## Chain 1: Iteration: 400 / 2000 [ 20%] (Warmup)
## Chain 1: Iteration: 600 / 2000 [ 30%] (Warmup)
## Chain 1: Iteration: 800 / 2000 [ 40%] (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%] (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%] (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%] (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%] (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%] (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%] (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%] (Sampling)
## Chain 1:
## Chain 1: Elapsed Time: 0.077886 seconds (Warm-up)
## Chain 1:           0.07452 seconds (Sampling)
## Chain 1:           0.152406 seconds (Total)
## Chain 1:
## 
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.6e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would
## take 0.16 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration: 1 / 2000 [  0%] (Warmup)

```

```

## Chain 2: Iteration: 200 / 2000 [ 10%] (Warmup)
## Chain 2: Iteration: 400 / 2000 [ 20%] (Warmup)
## Chain 2: Iteration: 600 / 2000 [ 30%] (Warmup)
## Chain 2: Iteration: 800 / 2000 [ 40%] (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%] (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%] (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%] (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%] (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%] (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%] (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%] (Sampling)
## Chain 2:
## Chain 2: Elapsed Time: 0.066793 seconds (Warm-up)
## Chain 2: 0.068833 seconds (Sampling)
## Chain 2: 0.135626 seconds (Total)
## Chain 2:
## 
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.6e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would
## take 0.16 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration: 1 / 2000 [ 0%] (Warmup)
## Chain 3: Iteration: 200 / 2000 [ 10%] (Warmup)
## Chain 3: Iteration: 400 / 2000 [ 20%] (Warmup)
## Chain 3: Iteration: 600 / 2000 [ 30%] (Warmup)
## Chain 3: Iteration: 800 / 2000 [ 40%] (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%] (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%] (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%] (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%] (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%] (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%] (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%] (Sampling)
## Chain 3:
## Chain 3: Elapsed Time: 0.068385 seconds (Warm-up)
## Chain 3: 0.08564 seconds (Sampling)
## Chain 3: 0.154025 seconds (Total)
## Chain 3:
## 
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.5e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would
## take 0.15 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:

```

```
## Chain 4:  
## Chain 4: Iteration: 1 / 2000 [  0%] (Warmup)  
## Chain 4: Iteration: 200 / 2000 [ 10%] (Warmup)  
## Chain 4: Iteration: 400 / 2000 [ 20%] (Warmup)  
## Chain 4: Iteration: 600 / 2000 [ 30%] (Warmup)  
## Chain 4: Iteration: 800 / 2000 [ 40%] (Warmup)  
## Chain 4: Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Chain 4: Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Chain 4: Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Chain 4: Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Chain 4: Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Chain 4: Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Chain 4: Iteration: 2000 / 2000 [100%] (Sampling)  
## Chain 4:  
## Chain 4: Elapsed Time: 0.061815 seconds (Warm-up)  
## Chain 4: 0.065315 seconds (Sampling)  
## Chain 4: 0.12713 seconds (Total)  
## Chain 4:  
  
## stan_glm  
## family: gaussian [identity]  
## formula: y ~ x  
## observations: 10  
## predictors: 2  
## -----  
## Median MAD_SD  
## (Intercept) 0.8 0.6  
## x 0.0 0.1  
##  
## Auxiliary parameter(s):  
## Median MAD_SD  
## sigma 0.9 0.2  
##  
## -----  
## * For help interpreting the printed output see ?print.stanreg  
## * For info on the priors used see ?prior_summary.stanreg
```

14_Bayesian_Intro

Sara Kophamel

15/12/2020

PRESENTATION:

file:///Users/sara/Documents/PhD/HELP/STATISTICS%20and%20GRAPHS/-%20R%20and%20STATS%20COURSES/2020%20Murray%20Logan%20R%20Course/Logan_Course/docs/pres10.html#11

THEORY

Bayesian calculates the LIKELIHOOD of something to happen. Calculating likelihood is VERY easy if you know the equation of the curve (Slide 11). However, we do not know it from scratch.

In frequentist analysis (slide 3), we calculate the probability of the DATA given a HYPOTHESIS. The hypothesis is about the DATA, but we cannot entirely know what happens in the real world. We establish several equations from our data to identify the peak of the curve. And that becomes our best estimate of the parameter. We then use that equation to calculate standard parameters (eg. t, F value...), which is then used to calculate a p value. BUT this p value is kind of being tested to prove that something is wrong (or right). And what we actually want is to know the probability that sth happens. The p values is a kind of link to this probability, but it does not fully explain it. This was great in the ages BEFORE computing. Frequentist analysis is unavoidably concerned with things that are more extreme than what you even got.

Another problem is that p values are unavoidably misused (only treat it as < or > than 0.05 !!!). Modern computing allows to calculate the exact p value, which is bullshit. A p value shows the prob of detecting an effect IF it occurs. That means, the larger the sample size, the more significant the p value. That is why the p value is JUST a reflection of your power / sample size.

Confidence intervals are also frequently misinterpreted. If we got CI of 90-100, it means that the MEAN is floating somewhere btw 90-100. In Bayesian analysis, we do not have CI, but can exactly estimate where the mean is. If we checked 100x where the mean is in freq analysis if the CI are btw 90-100. we would get the mean outside that range 10x. In freq analysis, the parameters are true and fixed, and there is an actual mean, but we don't know where it is exactly.

But NOW, we can actually go MUCH further. Bayesian calculates the prob of a hypothesis given data. Probability is a degree of BELIEF ("I am 95% confident that that will occur"). It is the natural way of how people interpret the results, which is not only "good" and "bad". Bayesian does not treat the true parameter as fixed. It treats it as a random distribution.

And in frequentist, all inferences are about the data, whereas in Bayesian it is about the parameters. Using %s is a far more natural language than focusing on p values.

SLide 6 (Example): Left plot: Sign relationship, although you can't see much from the plot. Middle plot: Not sign relationship, although the magnitude of the effect is the same as on the right plot. This is because of the sample size. The data is too noisy. Right plot: Sign relationship. This plot has the same magnitude than the middle plot, but is using a much larger sample size. That is why the p value says it has a significant decrease. Hence, the p value reflects sample size.

Slide 7 (Same example as on Slide 6): In Bayesian, on the other hand, we calculate the probability that the decline is >0 . And here, we see that the middle plot has got 86% probability of having a decline, whereas the p value for the slope of that plot was non significant!

Slide 9 (Bayesian theory): How do we go to a prob of a hypothesis $P(H)$ (which is the same as the prob of a "parameter"), given data?

$$P(H|D) = (P(D|H)xP(H)) / P(D)$$

The prob of data, given a hypothesis is LIKELIHOOD, which can be calculated. The prob of a hypothesis is our PRIOR BELIEF (what do we believe the true mean is?). Our outcome depends on what we are used to believe, and is not driven by data. Therefore, we could influence the outcome by what we believe. If we believe that the outcome is not important, that's it. And funny things, even in frequentist there are prior beliefs involved (eg. I choose a specific sample size, from X specific locations, etc).

The $P(H|D)$ indicates the posterior belief; the probability based on what you thought previously. The smaller your sample size, the smaller your probability, as your belief in how strong that effect is might be very low. In Bayesian, you can also test Power, but your conclusions are a little bit different. In freq, you'd simulate the data thousands of times, and then calculate the p values for each simulation. In Bayesian, there is no 0.05 standard, and you'd have to come up with it.

The $P(D)$ indicates the prob of data, the normalizing constant. It turns the curve into a prob distribution. But it is not available for us from the beginning. If the area under the curve is $\neq 1$, it is just a distribution (\neq prob distribution). The problem of this equation was that we did not have the tools to perform it, until modern computing appeared.

MCMC sampling (slide 14)

If you've got a histogram, you can calculate probability. E.g., we can calculate how many values are greater than 10. But to get there, I need to have sampled a number of times, and ideally I want to sample in a way that it is proportional to its likelihood. If we knew the formula of that curve, sampling would be trivial. We do NOT know the formula, but we can calculate the points of the curve.

Markov Chain Monte Carlo sampling enable us to reconstruct what the actual reconstruction looks like, without knowing what it looks like; simply by being able to calculate what its value is at any point.

MCMC randomly chooses coordinates for a and b (for an equation $y=a+bx$), defining possible estimates for intercepts and slope (a and b). These estimates might be totally useless, but we got a starting point. Now we start our chain. We now slightly move our estimates of a and b down, which reduces the likelihood. We now estimate HOW MUCH lower the likelihood is. If the initial value had been 10 and it had gone now down to 5, the prob had reduced by half. If it was reduced again, we might choose not to keep that data. However, if it went up, we might keep trying to describe the distribution (slide 17). MCMC sampling determines the best step size. By doing this again and again and again, we are slowly uncovering the likelihood, and discovering the uncertainty. We hereby create (which we want) a lot of noise and uncertainty.

If we continue this logic (slide 18), and repeat this sampling about 1000x, we get a very realistic estimate of the probability distribution. We are constructing this posterior likelihood reasonably well. After 10,000 samples, we pretty much got the prob distribution completely uncovered. BUT we still need other metrics to confirm that this is actually the real distribution.

MCMC diagnostics: Trace plots (slide 21):

R creates heaps of iterations (trace plots) around the midpoint. We have to run this 3-4x, each time from a random start. - If the plot looks like the one on the L, we have likely sampled the right distribution. - The middle plot tells you that you haven't fully identified the distribution. It might be because our data is incredibly noisy. We need to keep running the stats, starting at another point, to see if the distribution changes. - The R plot shows several features, indicating bad mixing. We need to keep running the stats, starting at another point, to see if the distribution changes. What if the diagnostics get a plot that does not change after 3-4x? Then we might not have sampled the right distribution.

MCMC diagnostics: Autocorrelation (slide 22)

Eventually, we want to find the distribution depicted on the slide. We might find several points that are closeby, and that are correlated (R figure). We therefore have to implement an Autocorrelation fix. Essentially, we kick out 9/10 values, keeping only every 10th value (slide 23). That way you break the dependency btw samples. When you thin your sample though, you will also have to keep running iterations until you find the real distribution (slide 24).

Sampler types:

Metropolis-Hastings sampler (slide 26)

You need a millions of samples to get a good chain, as it is checking every direction.

An improvement of this is the Gibbs sampler (slide 27), which will only move in the direction of one or the other parameter. However, it still produces a lot of autocorrelation, so the data has to be thinned a lot.

The NUTS sampler (no U turn sampler) is the current sampler being used. It involves motion physics. It is checking how well the chains are moving, and takes much less to calculate the prob distribution. For a lot of the likelihood, your distribution might not look like anything, but at the end of the sampling, it will become more and more clear, and better shaped. In some cases with very weird (eg. thin) distributions, NUTS (and the other samplers) might sample outside the real distribution, not getting the real points and not really sampling the right area. The metrics would start failing a lot, and the whole sample would get stuck.

In those cases, we might need specific samplers for weird distribution.

In summary, we need to define:

- How long is the chain = how many iterations do we have to perform
- How much thinning do we need to do?
- How much burning (warmup) do we need? At the Beginning, the physics are unreliable, so we toss them away. If we got let's say 1000 iterations, we'd toss away the first 10% (and for complex models, the first 50%).

Software to use for Bayesian analyses:

- MCMCpack: Can be used within R. Is very fast, but can only be used for very simple models, so we are not gonna use it.
- winbugs (R2winbugs): Unbelievably slow, written in a very bizarre language. Does not run in sequence, but as a whole. It is a nightmare to develop. Uses a Gibbs sampler. R could run these, but they are crap.
- jags (R2jags): Much faster than winbugs, but still using the same terrible language. Uses a Gibbs sampler. R could run these, but they are crap.
- stan (rstan, rstanarm, brms): stan uses a NUTS sampler. It is written in some sort of C language, and then compiles the models (turns the human language into machine language) to run the models. The good thing is that R can use the following Bayesian packages based on stan.
 - rstanarm: R takes the model formula, converts it to stan, and runs it. Fits GLM models. We will use this one.

- brms: Supports all models you can think of. Unfortunately, it is not as easy to install, as our computer needs a C++ compiler, which is often not the case. We could use this package for more complicated models.

ILA is a software which does Bayesian approximation within R, and is super fast. The problem is that it is very hard to use, as the language of ILA is not the same as the standard R language.

PRACTICE

OPEN: bglm_example1

See 15-17_Bayesian_LinRegr for additional explanations (I might have missed some things here)

Preparations

Load the necessary libraries

```
library(rstanarm)    #for fitting models in STAN
## Loading required package: Rcpp
## This is rstanarm version 2.21.1
## - See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!
## - Default priors may change, so it's safest to specify priors, even if equivalent to the defaults.
## - For execution on a local, multicore CPU with excess RAM we recommend calling
##   options(mc.cores = parallel::detectCores())
library(brms)        #for fitting models in STAN
## Loading 'brms' package (version 2.14.4). Useful instructions
## can be found by typing help('brms'). A more detailed introduction
## to the package is available through vignette('brms_overview').
##
## Attaching package: 'brms'
```

```

## The following objects are masked from 'package:rstanarm':
##
##     dirichlet, exponential, get_y, lasso, ngrps

## The following object is masked from 'package:stats':
##
##     ar

library(coda)      #for diagnostics
library(bayesplot) #for diagnostics

## This is bayesplot version 1.7.2

## - Online documentation and vignettes at mc-stan.org/bayesplot
## - bayesplot theme set to bayesplot::theme_default()
##   * Does _not_ affect other ggplot2 plots
##   * See ?bayesplot_theme_set for details on theme setting

library(ggmcmc)    #for MCMC diagnostics

## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: tidyverse

## Loading required package: ggplot2

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

library(DHARMa)     #for residual diagnostics

## This is DHARMa 0.3.3.0. For overview type '?DHARMa'. For recent changes,
## type news(package = 'DHARMa') Note: Syntax of plotResiduals has changed in
## 0.3.0, see ?plotResiduals for details

library(rstan)       #for interfacing with STAN

## Loading required package: StanHeaders

```

```

## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend
## calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)

##
## Attaching package: 'rstan'

## The following object is masked from 'package:tidyverse':
## extract

## The following object is masked from 'package:coda':
## traceplot

library(emmeans)    #for marginal means etc
library(broom)       #for tidying outputs
library(tidybayes)   #for more tidying outputs

##
## Attaching package: 'tidybayes'

## The following objects are masked from 'package:brms':
## dstudent_t, pstudent_t, qstudent_t, rstudent_t

library(ggeffects)  #for partial plots
library(tidyverse)   #for data wrangling etc

## — Attaching packages

```

```

----- tidyverse 1.3.0 ----

## ✓ tibble  3.0.3      ✓ stringr 1.4.0
## ✓ readr   1.3.1      ✓forcats 0.5.0
## ✓ purrr   0.3.4

## — Conflicts

```

```

----- tidyverse_conflicts() ----

## x rstan::extract() masks tidyverse::extract()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()

library(broom.mixed) #for summarising models (it's an extension of broom)

```

```
## Registered S3 method overwritten by 'broom.mixed':  
##   method      from  
##   tidy.gamlss broom  
  
library(ggeffects) #for partial effects plots  
theme_set(theme_grey()) #puts the default ggplot theme back
```

Scenario

Here is an example from @Fowler-1998-1998. An agriculturalist was interested in the effects of fertilizer load on the yield of grass. Grass seed was sown uniformly over an area and different quantities of commercial fertilizer were applied to each of ten 1 m² randomly located plots. Two months later the grass from each plot was harvested, dried and weighed. The data are in the file **fertilizer.csv** in the **data** folder.



FERTILIZER	YIELD
25	84
50	80
75	90
100	154
125	148
...	...

FERTILIZER: Mass of fertilizer (g.m⁻²) - Predictor variable

YIELD: Yield of grass (g.m⁻²) - Response variable

The aim of the analysis is to investigate the relationship between fertilizer concentration and grass yield.

Read in the data

```
fert = read_csv('data/fertilizer.csv', trim_ws=TRUE)
```

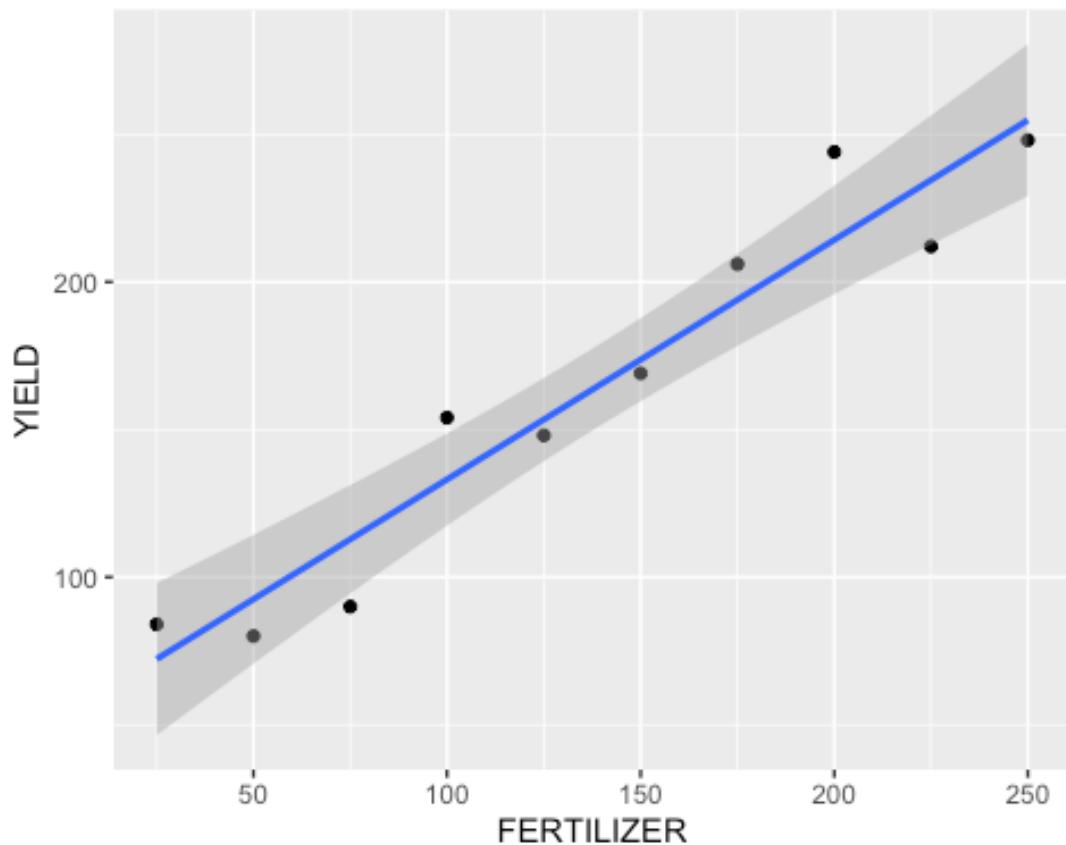
```
## Parsed with column specification:  
## cols(  
##   FERTILIZER = col_double(),  
##   YIELD = col_double()  
## )  
  
glimpse(fert)  
  
## Rows: 10  
## Columns: 2  
## $ FERTILIZER <dbl> 25, 50, 75, 100, 125, 150, 175, 200, 225, 250  
## $ YIELD      <dbl> 84, 80, 90, 154, 148, 169, 206, 244, 212, 248
```

Exploratory data analysis

Bayesian still needs to know if we are dealing with normal, poisson, gaussian, etc. data. We will do some exploratory plots for that purpose.

Exploratory plot

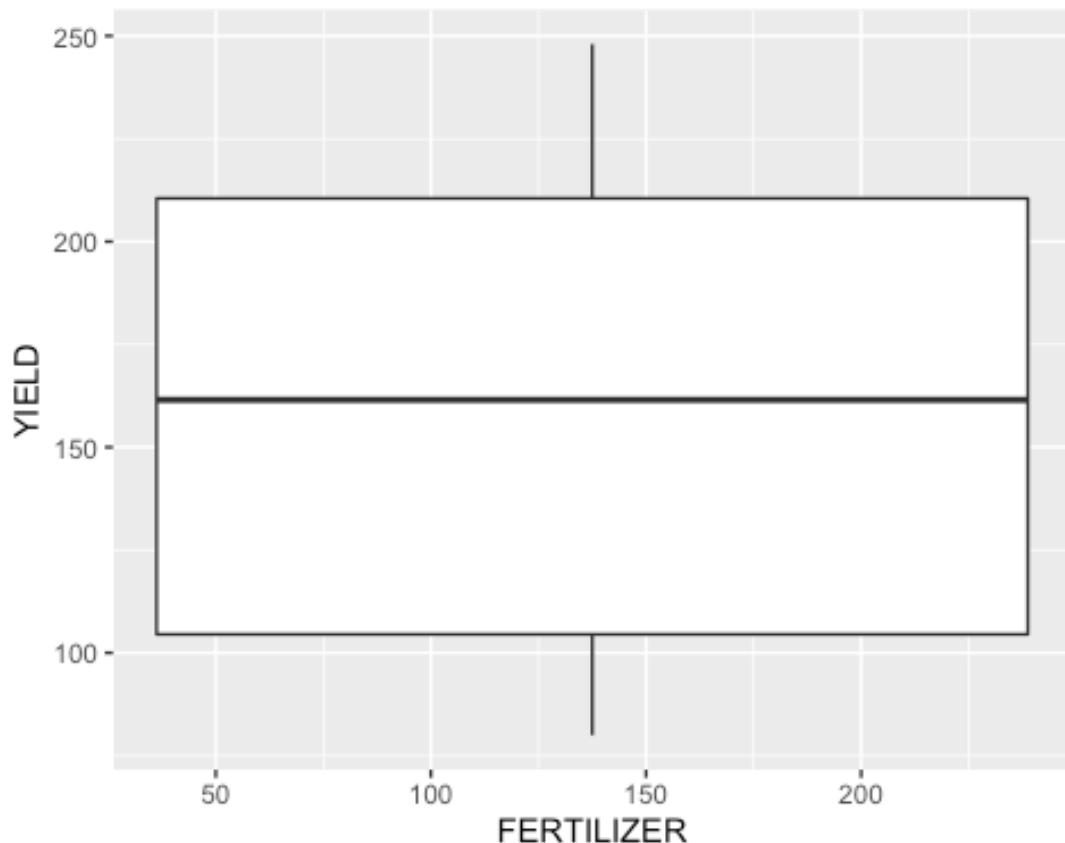
```
ggplot(fert, aes(y=YIELD, x=FERTILIZER)) +  
  geom_point() +  
  geom_smooth(method="lm")  
  
## `geom_smooth()` using formula 'y ~ x'
```



Does this plot look like... - linearity is gonna be sufficient? YES - Homogeneity of variance is sufficient (spread of data around the line)? YES - Normality: YES (can be double checked with box plot - the whiskers have to be same length, and median close to mean; see below)

```
ggplot(fert, aes(y=YIELD, x=FERTILIZER))+
  geom_boxplot()

## Warning: Continuous x aesthetic -- did you forget aes(group=...)?
```



purpose, we stick to a normal distribution.

Model formula:

$$\begin{aligned}
 y_i &\sim \mathcal{N}(\mu_i, \sigma^2) \\
 \mu_i &= \beta_0 + \beta_1 x_i \\
 \beta_0 &\sim \mathcal{N}(0, 100) \\
 \beta_1 &\sim \mathcal{N}(0, 10) \\
 \sigma &\sim \text{cauchy}(0, 5) \\
 \text{OR} \\
 \sigma &\sim \text{Exp}(1)
 \end{aligned}$$

For this

In Bayesian, we need to identify what our parameters are and apply priors to each of those. For our effects parameters (betas), they all follow a normal distribution. If we had prior information, we would use accurate priors. But we don't, so we will start with very vague priors. For analyses with continuous variables, we need a predictor that gives the y intercepts meaning. But this predictor also offers computational efficiency, possibly making a difference in calculation time of minutes. For that reason, the software we are using centres our predictors by default. The priors apply when the data has been centered. Our SD is = to how variable our data are. If our prior is too tight, the posterior will get mainly defined by the prior. We therefore have to set the prior as wide as possible. Often, our prior will look like a cauchy distribution (a normal distribution chopped in half, with the peak as

$y=0$). We could therefore use a cauchy, or an exponential distribution. We also got a slope, which can be pos or neg. We got a certain amount of noise for our prior, which we also have to define. We usually start by providing sth wide, and then narrow it down.

In summary, we have to pre-define the priors to get the best output.

Fit the model

1. Initial model (Model draft)

Equivalent of a GLM, but run in Bayesian

```
fert.rstanarm=stan_glm(YIELD~FERTILIZER,data=fert,
                        iter=5000,
                        warmup=1000,
                        chains=3,
                        thin=5, # we are only gonna keep the 5th value
                        refresh=0)

fert.rstanarm

## stan_glm
## family: gaussian [identity]
## formula: YIELD ~ FERTILIZER
## observations: 10
## predictors: 2
## -----
##           Median MAD_SD
## (Intercept) 52.4   14.2
## FERTILIZER    0.8    0.1
##
## Auxiliary parameter(s):
##           Median MAD_SD
## sigma 20.5    5.3
##
## -----
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg

# The equivalent frequentist model would have been: lm(YIELD~FERTILIZER,
# data=fert)
```

2. Check the default priors that were applied (they used the Adjusted prior within Intercept)

```
prior_summary(fert.rstanarm)

## Priors for model 'fert.rstanarm'
## -----
```

```

## Intercept (after predictors centered)
## Specified prior:
##   ~ normal(location = 164, scale = 2.5)
## Adjusted prior:
##   ~ normal(location = 164, scale = 160)
##
## Coefficients
## Specified prior:
##   ~ normal(location = 0, scale = 2.5)
## Adjusted prior:
##   ~ normal(location = 0, scale = 2.1)
##
## Auxiliary (sigma)
## Specified prior:
##   ~ exponential(rate = 1)
## Adjusted prior:
##   ~ exponential(rate = 0.016)
## -----
## See help('prior_summary.stanreg') for more details

```

Intercept (after predictors centered) Specified prior $\sim \text{normal}(\text{location} = 164, \text{scale} = 2.5)$ –> If we take our SD and multiply by 2.5 we should get 164 (which is the mean, see code below) –> R used the Adjusted Prior

Auxiliary (sigma) Specified prior: $\sim \text{exponential}(\text{rate} = 1)$ Adjusted prior: $\sim \text{exponential}(\text{rate} = 0.016)$ –> This is the alternative prior for a sigma distribution (=exponential?)

Calculate:

The mean for our response; in this case, the mean yield:

```

mean(fert$YIELD)
## [1] 163.5

```

The SD:

```

sd(fert$YIELD)
## [1] 63.97439

```

Adjusted Intercept prior used:

```

sd(fert$YIELD)*2.5 # 2.5=scale
## [1] 159.936

```

Not sure what this was (?):

```

1/sd(fert$YIELD)

```

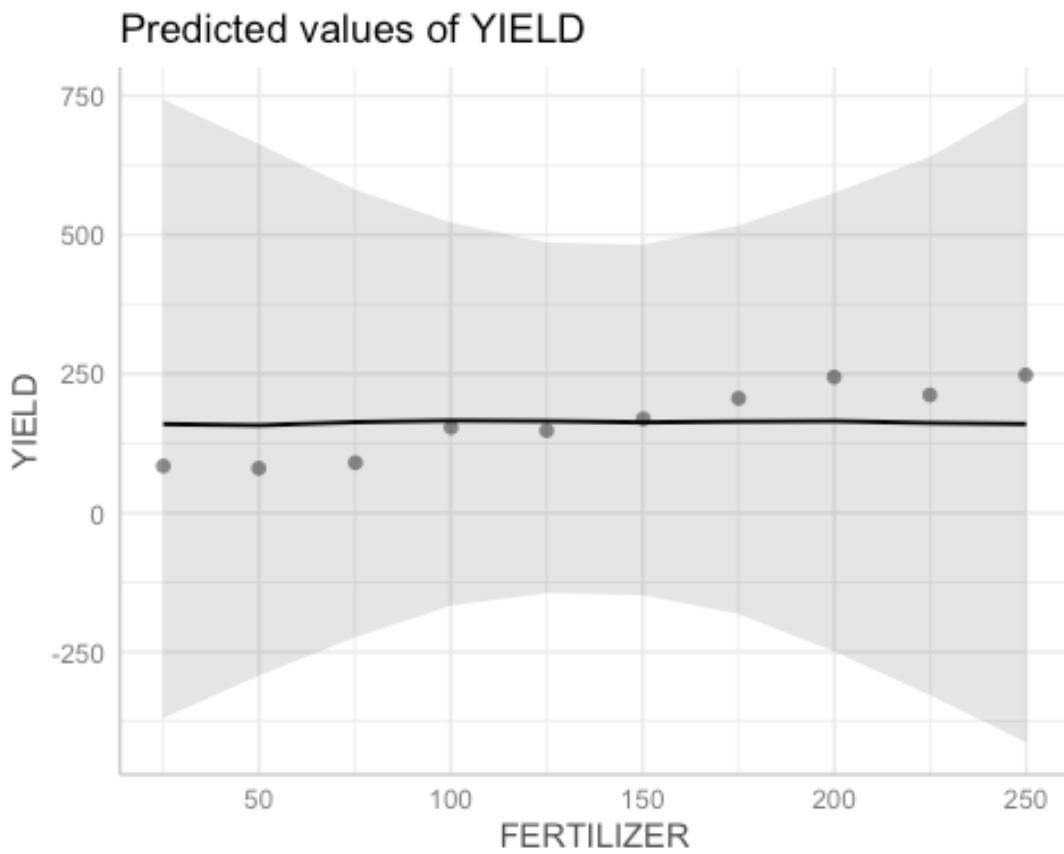
```
## [1] 0.01563126
```

3. Run the model again, only including the prior

We are trying to make sure that the priors we are setting are sensible to the posteriors resulting from the data. You want the priors to be vague, but not too vague.

We tell R to show us only the model, WITHOUT the y axis (it includes x)

```
fert.rstanarm1<-update(fert.rstanarm,prior_PD=TRUE)
ggpredict(fert.rstanarm1) %>% plot(add.data=TRUE)
## $FERTILIZER
```



This graph shows what our prior ALONE would suggest. We have got a mean of 164 (or whatever we nominated). This would allow us to have sth between a strong pos and a strong neg slope. It is NOT gonna have much influence on the outcome. If we had some data, we could define the pattern. You wanna check if there is anything non-sensible here. If for example we are expecting a pos slope, but the prior shows a negative slope, we have chosen a prior that is too strong. However, you also don't want to have a prior with the largest conf bands ever, as it would be too vague. In that case, it may be that we are sampling nowhere near the data (the sampler will be sampling very inefficiently). We

essentially are checking if the bands are massively wide (model runs inefficiently), or massively narrow (too influential)

4. Fit our own priors

We can come up with them any way we like. Murray will make them a bit narrower than the ones we chose. We could also define them as just pos values if we got no - values in our data, for example.

Murray only wants to change the width of the prior, the SD.

We first create the model as if we were only using the priors, not the actual data (see step 5. for final model)

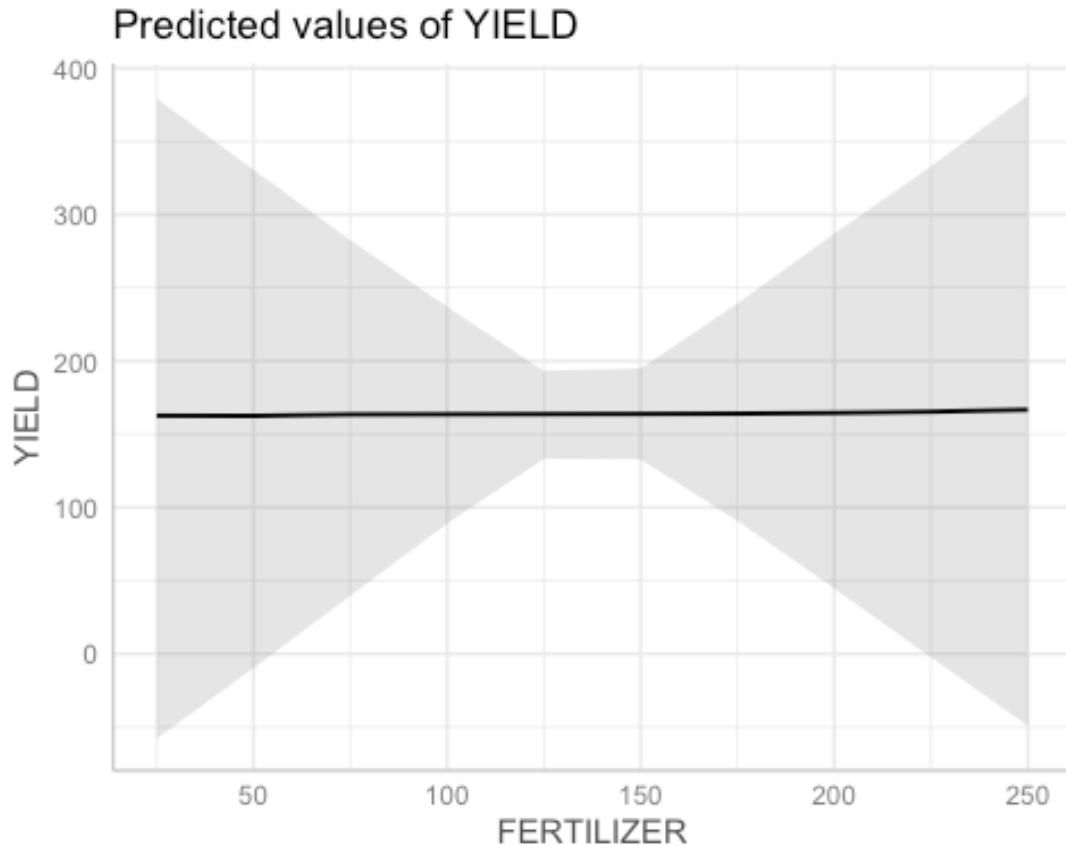
```
fert.rstanarm2=stan_glm(YIELD~FERTILIZER, data=fert,
                         prior_intercept=normal(164,10,autoscale=FALSE), #
autoscale=FALSE as we already considered the priors we want; we don't have to scale them to our data.
                         # 164 = This is the approx mean value (of the first group) on the x axis (see exploratory plot) - if we had a log scale, we'd take the Log number of the raw value
                         # 10 = Approximate orders of magnitude --> let's say we got a max value of ~10, so we'd set this to be 10. However, it could be anything really (in this case Murray took a random number, as far as I know). But we could set it to what the boxplots tell us is the max value.
                         prior=normal(0,1,autoscale=FALSE), # default values
# autoscale=FALSE as we already considered the priors we want; we don't have to scale them to our data
                         # 0 = We don't wanna define a pos or neg interaction (we don't know how the response is gonna change), so we set this value to zero
                         # 1 = we give the priors enough space to develop
                         prior_aux=cauchy(0,2,autoscale=FALSE),
                         # 0=WHY THIS VALUE?
                         # 2=WHY THIS VALUE?
                         prior_PD=TRUE,# only the priors are involved in this model, as we wanna see whether our priors look vaguely sensible (is the scheme not too wide and not too narrow?)
                         iter=5000, warmup=1000, # default values (explained above)
                         chains=3, thin=5, refresh=0) # default values
(explained above)
```

REMINDER: Priors for model 'fert.rstanarm' —— Intercept (after predictors centered)
 Specified prior: $\sim \text{normal}(\text{location} = 164, \text{scale} = 2.5)$ Adjusted prior: $\sim \text{normal}(\text{location} = 164, \text{scale} = 160) \rightarrow 164$ was chosen as the prior_intercept

Coefficients Specified prior: $\sim \text{normal}(\text{location} = 0, \text{scale} = 2.5)$ Adjusted prior: $\sim \text{normal}(\text{location} = 0, \text{scale} = 2.1)$

Auxiliary (sigma) Specified prior: $\sim \text{exponential}(\text{rate} = 1)$ Adjusted prior: $\sim \text{exponential}(\text{rate} = 0.016)$ ——

```
ggpredict(fert.rstanarm2) %>%  
  plot()  
  
## $FERTILIZER
```



We can see that the priors are a lot narrower (possibly even too narrow, but it's ok for this example). They are also not so strong that they yield pos or neg relationships, so that's good.

5. Fit the ACTUAL model

Now that we have checked the priors of rstanarm2, and are happy with priors selected, we tell R that we want to include the data as well (and not only the prior). We therefore update the model, and define prior_PD=FALSE

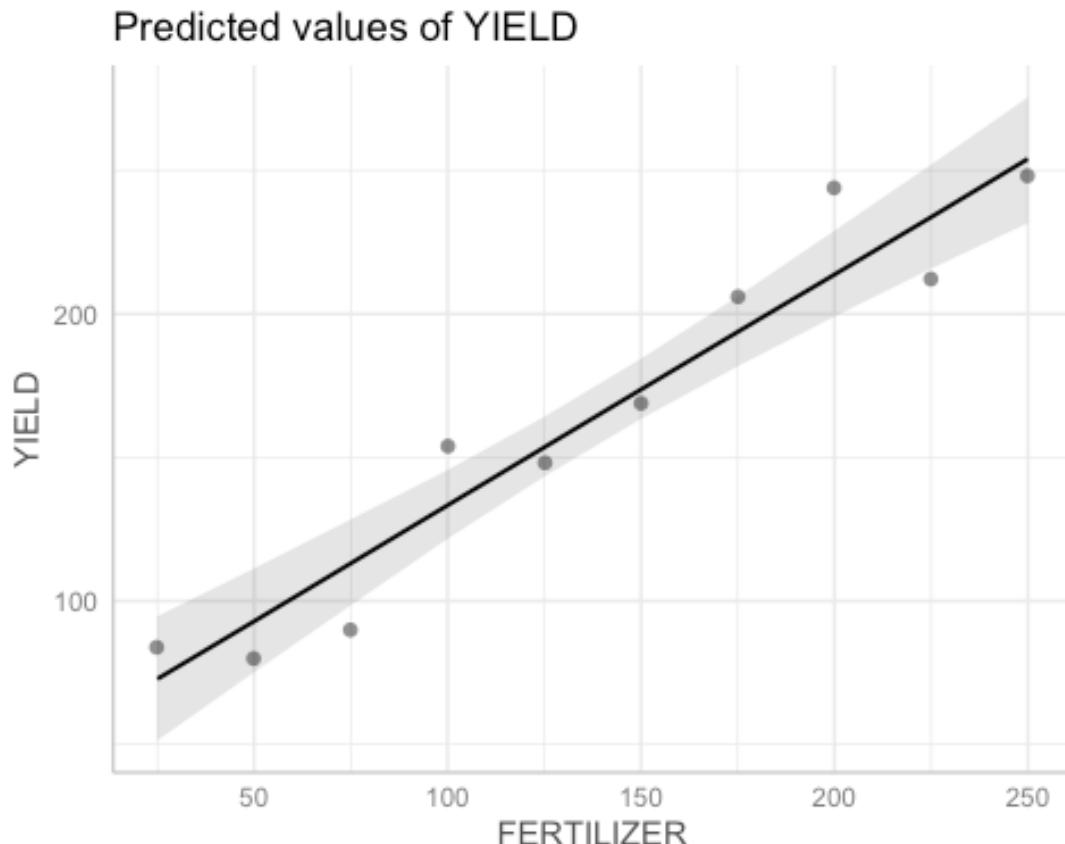
```
fert.rstanarm3<-update(fert.rstanarm2, prior_PD=FALSE)
```

It is like saying: Should we JUST have the prior? NO, include the data

6. Predict the data

```
ggpredict(fert.rstanarm3) %>% plot(add.data=TRUE)
```

```
## $FERTILIZER
```

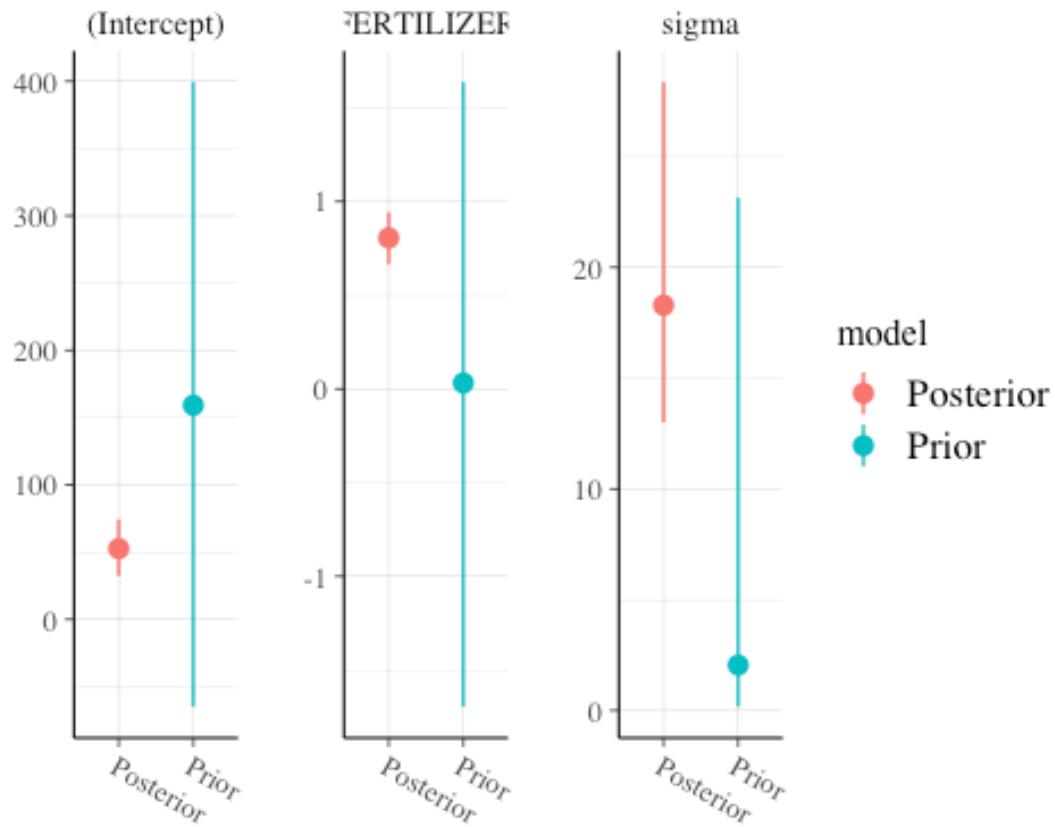


looks pretty much like the real data, so it seems like our priors were great.

This

7. Checking how well our priors fit the actual data

```
posterior_vs_prior(fert.rstanarm3,color_by="vs",group_by=TRUE,  
facet_args=list(scales="free_y"))  
  
##  
## Drawing from prior...
```



We do not want to see that the spread of data for the posteriors is similar to that of the priors, and aligned with the priors. A bad outcome would be: (a) if the Posterior had the same size as prior = Posterior is a reflection of the prior. (b) if the prior (pr) and posteriors (pos) were narrow, and the intercept (int) was narrow too = The posterior can't find the real data, as it is too narrow: I

I I

pr pos int

- if the intercept was really wide, and the posterior was at the same level than the intercept, it would be ok: I I I pr pos int

-> In summary: The priors have to be quite wide, and the posteriors a LOT narrower

8. Model validation

WE NOW CONDUCT SAMPLING DIAGNOSTICS:

1. MCMC sampling diagnostics

We want to know: - is our thinning ok? - have the chains converged after 3x iteration? etc

We have all the following diagnostics available to us:

```
available_mcmc()

## bayesplot MCMC module:
##   mcmc_acf
##   mcmc_acf_bar
##   mcmc_areas
##   mcmc_areas_data
##   mcmc_areas_ridges
##   mcmc_areas_ridges_data
##   mcmc_combo
##   mcmc_dens
##   mcmc_dens_chains
##   mcmc_dens_chains_data
##   mcmc_dens_overlay
##   mcmc_hex
##   mcmc_hist
##   mcmc_hist_by_chain
##   mcmc_intervals
##   mcmc_intervals_data
##   mcmc_neff
##   mcmc_neff_data
##   mcmc_neff_hist
##   mcmc_nuts_acceptance
##   mcmc_nuts_divergence
##   mcmc_nuts_energy
##   mcmc_nuts_stepsize
##   mcmc_nuts_treedepth
##   mcmc_pairs
##   mcmc_parcoord
##   mcmc_parcoord_data
##   mcmc_rank_hist
##   mcmc_rank_overlay
##   mcmc_recover_hist
##   mcmc_recover_intervals
##   mcmc_recover_scatter
##   mcmc_rhat
##   mcmc_rhat_data
##   mcmc_rhat_hist
##   mcmc_scatter
##   mcmc_trace
##   mcmc_trace_data
##   mcmc_trace_highlight
##   mcmc_violin
```

Shiny has got other options to check these diagnostics (Murray will share the code later on)

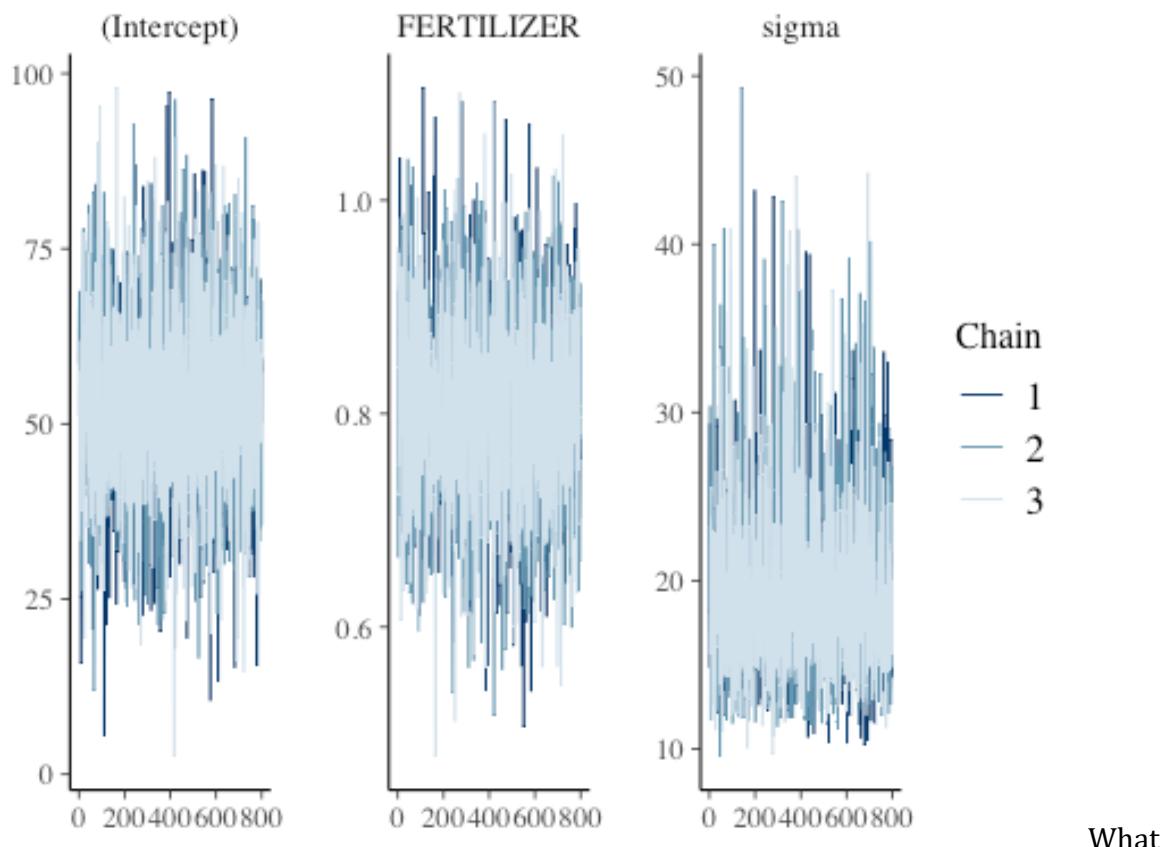
Murray will focus on the following: - Trace plots - Autocorrelation $f(x)$ = Are we thin enough?

We will have to check the following diagnostics:

1. Trace plots

Option 1

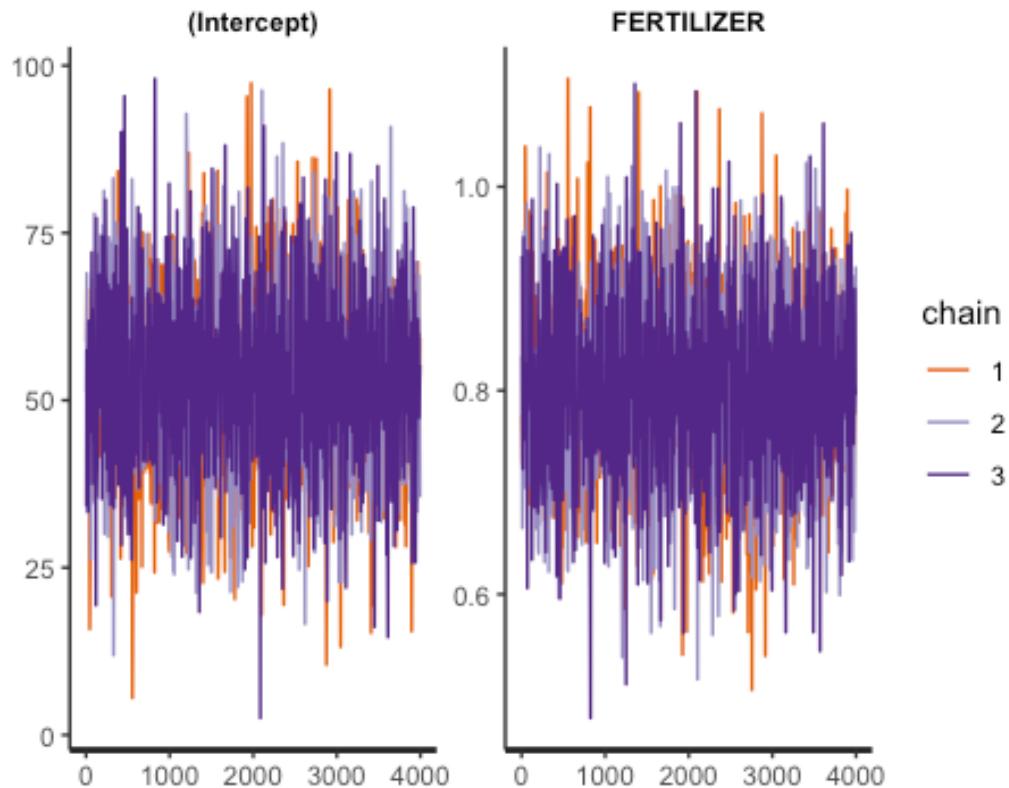
```
plot(fert.rstanarm3, plotfun="mcmc_trace")
```



What we are looking for here is just noise, and that is what we got. There are three chains, all in different blue shades. They appear all mixed together and converged (each of the three chains says pretty much the same thing), so that's good. That is what we want to see.

Option 2

```
stan_trace(fert.rstanarm3)
```



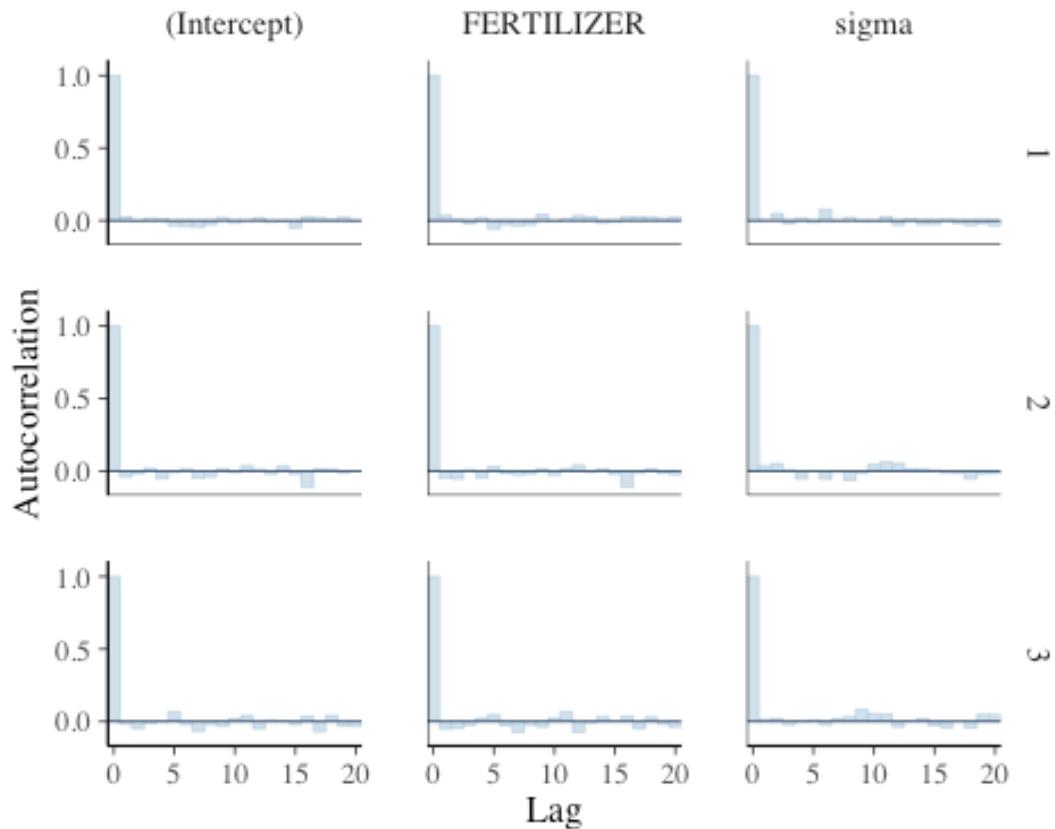
interpretation of this plot is the same as the one above

The

2. Autocorrelation function for thinning

Option 1

```
plot(fert.rstanarm3, plotfun="acf_bar")
```

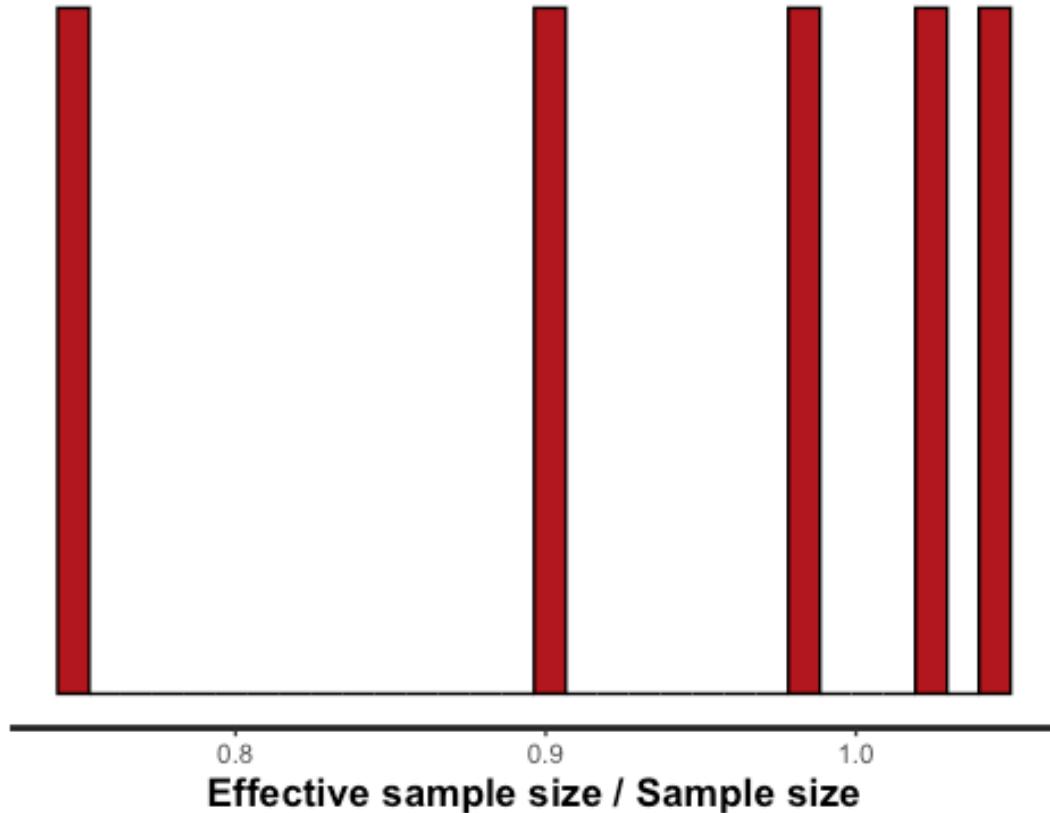


We don't wanna see big bars after the first one. In this case, we don't want to thin any further. 5 is adequate (this number is usually enough for all models).

Option 2

```
stan_ess(fert.rstanarm3)
```

```
## `stat_bin()` using `bins = 30` . Pick better value with `binwidth` .
```

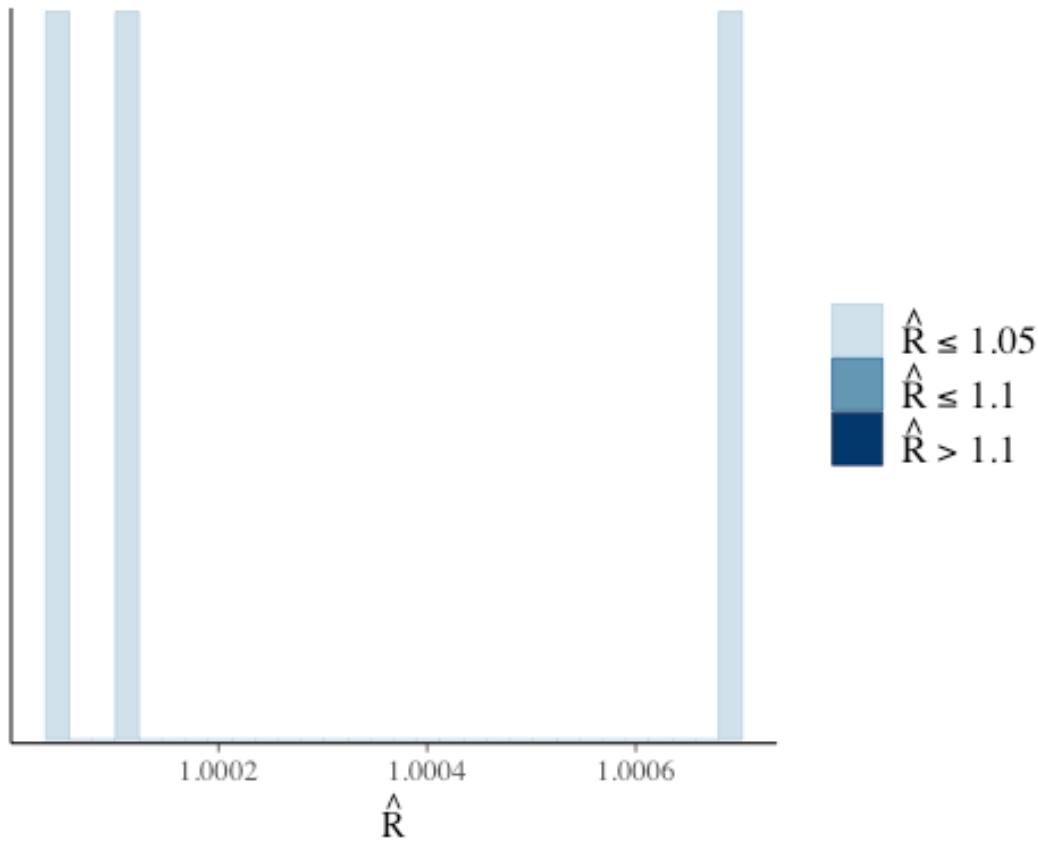


interpretation of this plot is the same as the previous plot. There should be 5 bars (sometimes the bar show on top of each other and it seems as if there were 4).

The

3. Measure of chain convergence - R hat

```
plot(fert.rstanarm3, "rhat_hist")  
## `stat_bin()` using `bins = 30` . Pick better value with `binwidth` .
```



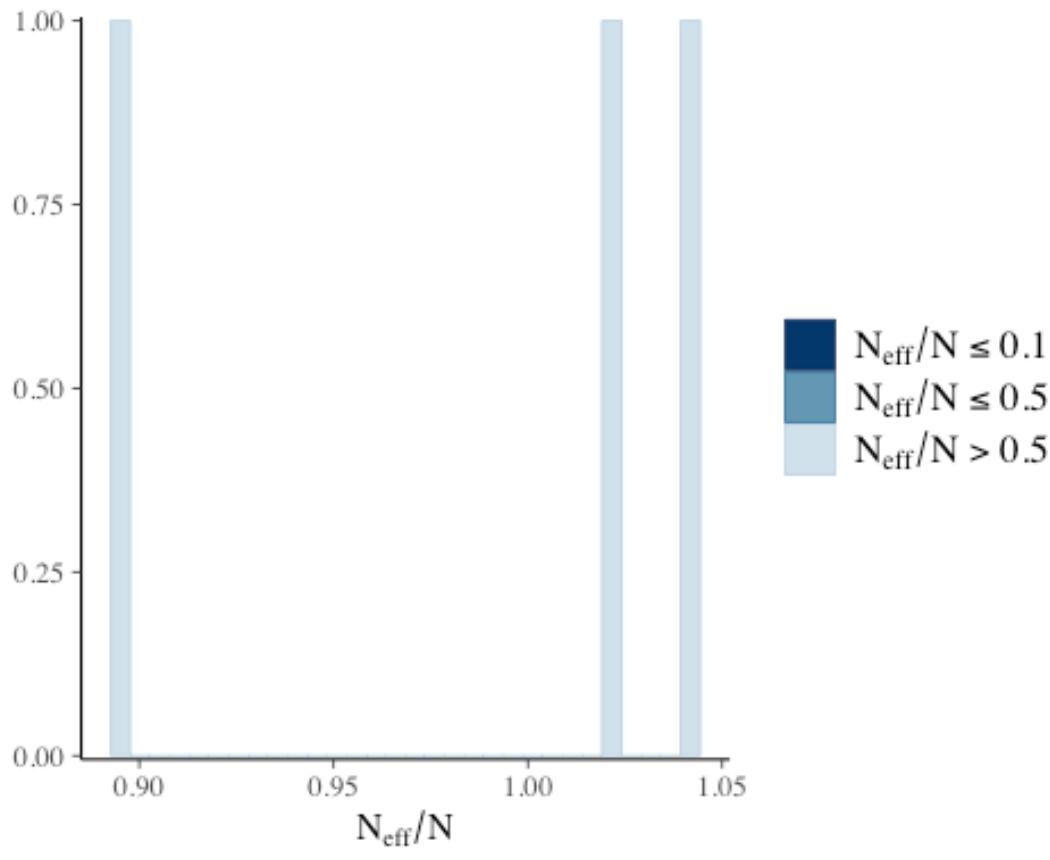
We want all values be <1.05 . This is the threshold above which they are not converged. This graph colors things in that are <1.05 , btw $1.05-1.1$, and >1.1 . But anything >1.05 = chains haven't converged.

If the chains haven't converged, we have to:

- increase the number of iterations (perhaps one sample focused on only one area)
- narrow the priors a bit more (as one might have been too wide and got stuck in a specific area) → This is usually not the case if you supply your own priors, but could happen if you use the default R priors.
- let the priors run for longer (might not be an option as this takes a LOONG time)

4. Number of effective samples

```
plot(fert.rstanarm3, "neff_hist")
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



You

want the values to be close to 1 (=100% effective). If you have a large number of effective samples, the number of samples that are rejected might be low, but if you got a low number of effective samples, the model might get lost sampling somewhere else.

If we have values far away from 1, the solution is to narrow the priors.

2. PPC plots - Posterior probability checks

These plots check if our model are close to reality.

(!) MURRAY SAID THAT WE COULD JUMP AHEAD TO DO THE DHARMA RESIDUALS, AND DON'T FOCUS ON THE PPC PLOTS (!)

We got the following plots available

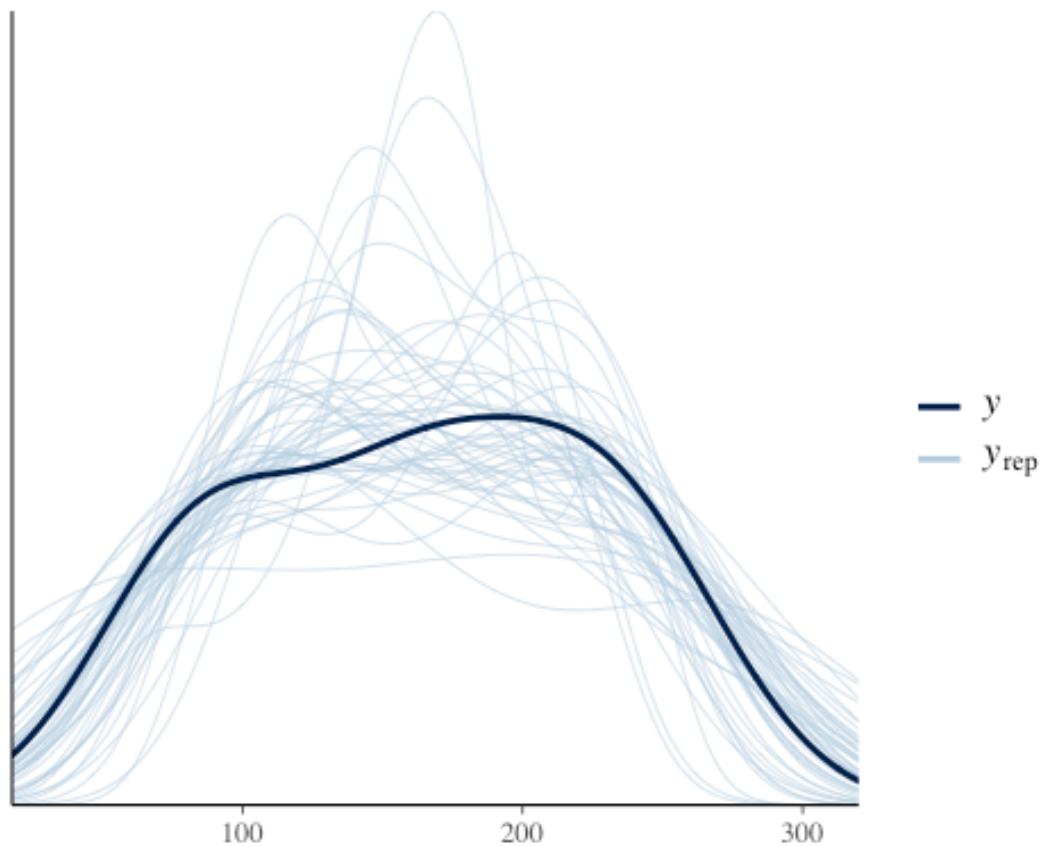
```
available_ppc()

## bayesplot PPC module:
##   ppc_bars
##   ppc_bars_grouped
##   ppc_boxplot
##   ppc_data
##   ppc_dens
##   ppc_dens_overlay
```

```
## ppc_ecdf_overlay
## ppc_error_binned
## ppc_error_hist
## ppc_error_hist_grouped
## ppc_error_scatter
## ppc_error_scatter_avg
## ppc_error_scatter_avg_vs_x
## ppc_freqpoly
## ppc_freqpoly_grouped
## ppc_hist
## ppc_intervals
## ppc_intervals_data
## ppc_intervals_grouped
## ppc_loo_intervals
## ppc_loo_pit
## ppc_loo_pit_overlay
## ppc_loo_pit_qq
## ppc_loo_ribbon
## ppc_ribbon
## ppc_ribbon_data
## ppc_ribbon_grouped
## ppc_rootogram
## ppc_scatter
## ppc_scatter_avg
## ppc_scatter_avg_grouped
## ppc_stat
## ppc_stat_2d
## ppc_stat_freqpoly_grouped
## ppc_stat_grouped
## ppc_violin_grouped
```

1. Density overlay plot

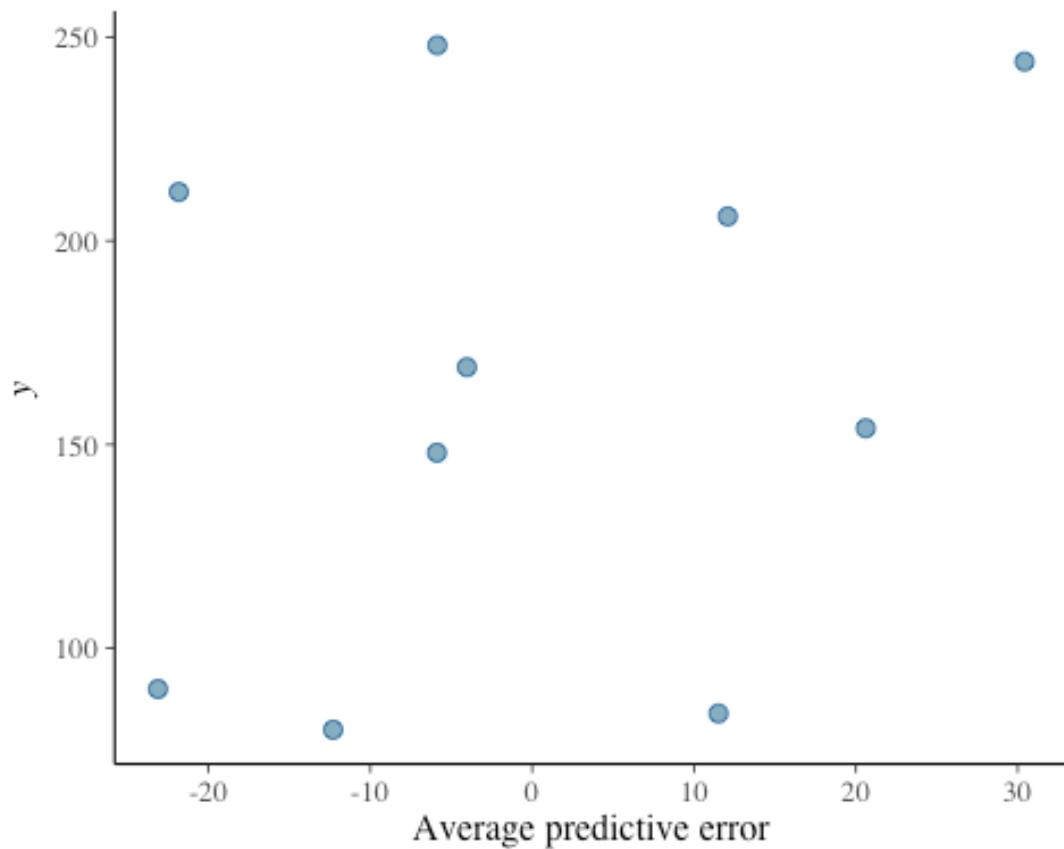
```
pp_check(fert.rstanarm3, plotfun="dens_overlay")
```



This plot plots the actual distribution of our response (in this case YIELD, dark blue line). And each of the lighter blue lines are realisations that have been drawn from the model; i.e. they are predictions from the model. You can see that they are fairly consistent, i.e. the model is in fairly big line with reality.

2. Error scatterplot

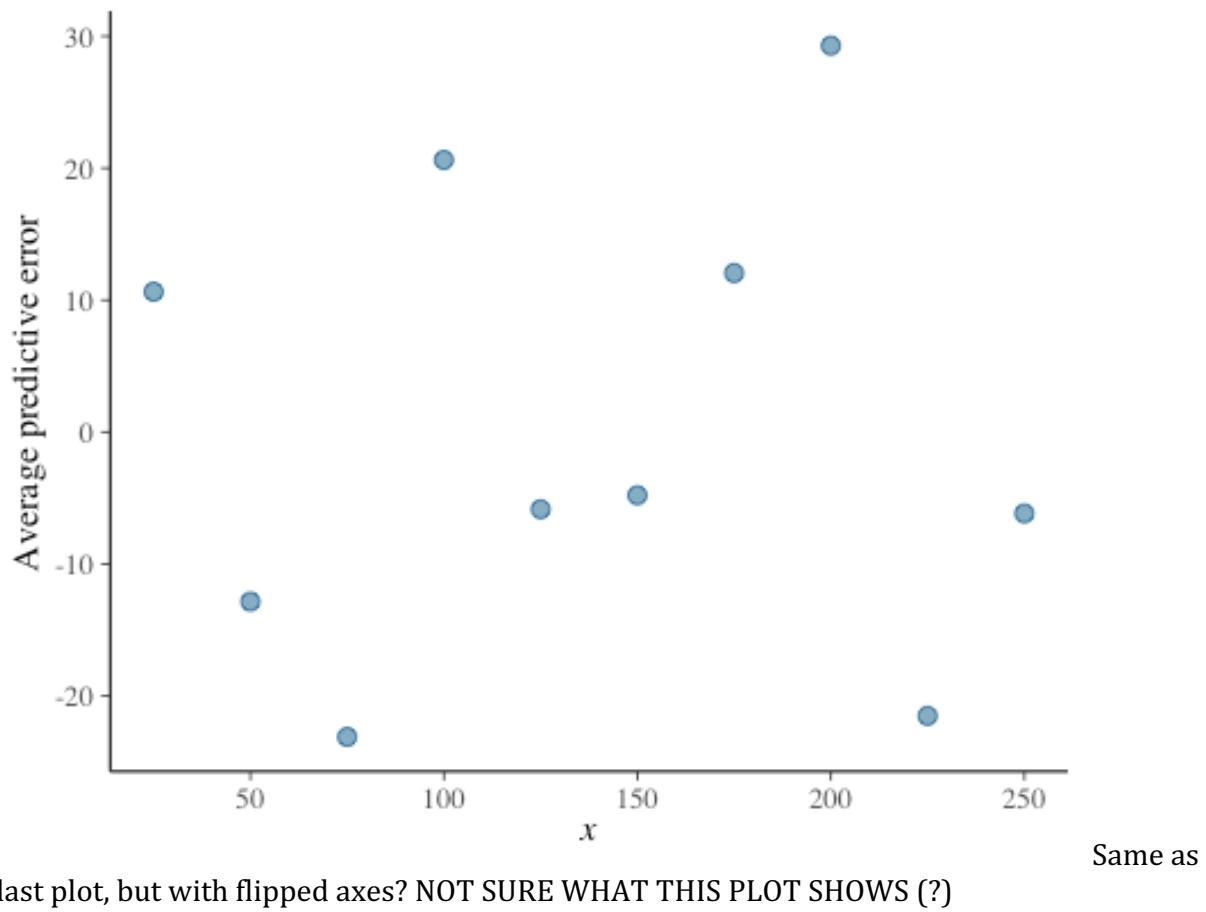
```
pp_check(fert.rstanarm3, plotfun="error_scatter_avg")
```



You don't want to see a plot where points are very far away from other points = That point would have a very big error. We would have been very bad at predicting that point. And possibly, that point was an outlier. This plot is analogous to a residual plot.

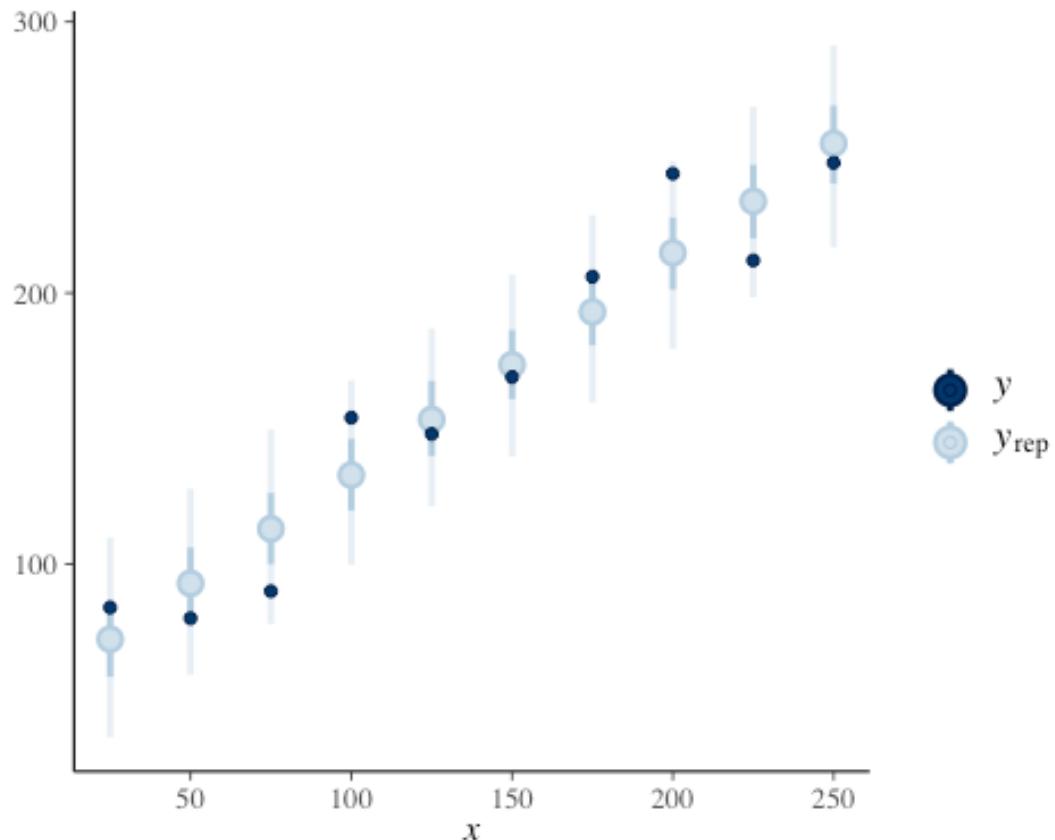
3. Predictive error plotted against each response

```
pp_check(fert.rstanarm3, x=fert$FERTILIZER, plotfun="error_scatter_avg_vs_x")
```



4. Raw data vs pred range of responses.

```
pp_check(fert.rstanarm3, x=fert$FERTILIZER, plotfun="intervals")
```



The

predictions are similar to the raw data, which is what we wanna see. It is an extension to the first plot of this series. We don't want to see black dots near the conf bands.

The predictions (y_{rep}) are about the same as the raw data (y).

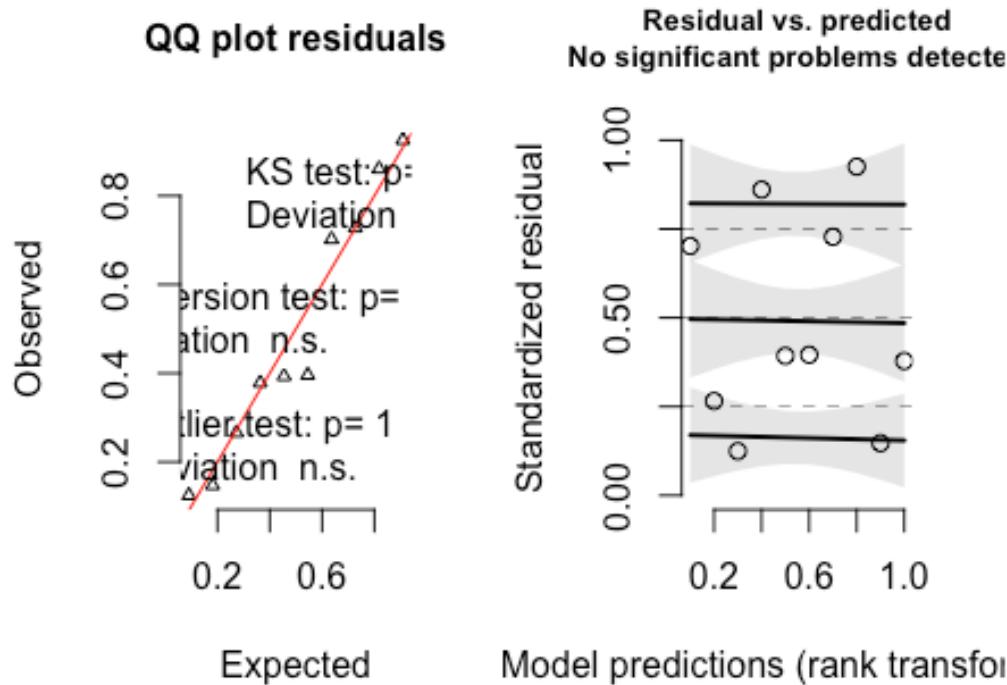
1. DHARMA residuals

```
preds<-posterior_predict(fert.rstanarm3, nsamples=250, summary=FALSE) # We tell
R to give us 250 predictions to do simulations (standard number)

fert.resids<-createDHARMA(simulatedResponse=t(preds), # predicted resp (best
fit values)
                           observedResponse=fert$YIELD, # this is our raw
response values, the YIELD in the example (this is the only thing we'd change
when using other datasets)
                           fittedPredictedResponse=apply(preds, 2, median),
#We tell R to take the predicted values, and calculate the median for each
integerResponse="gaussian")

plot(fert.resids)
```

DHARMA residual diagnostics



For creating the DHARMA residuals plot, we have to provide:

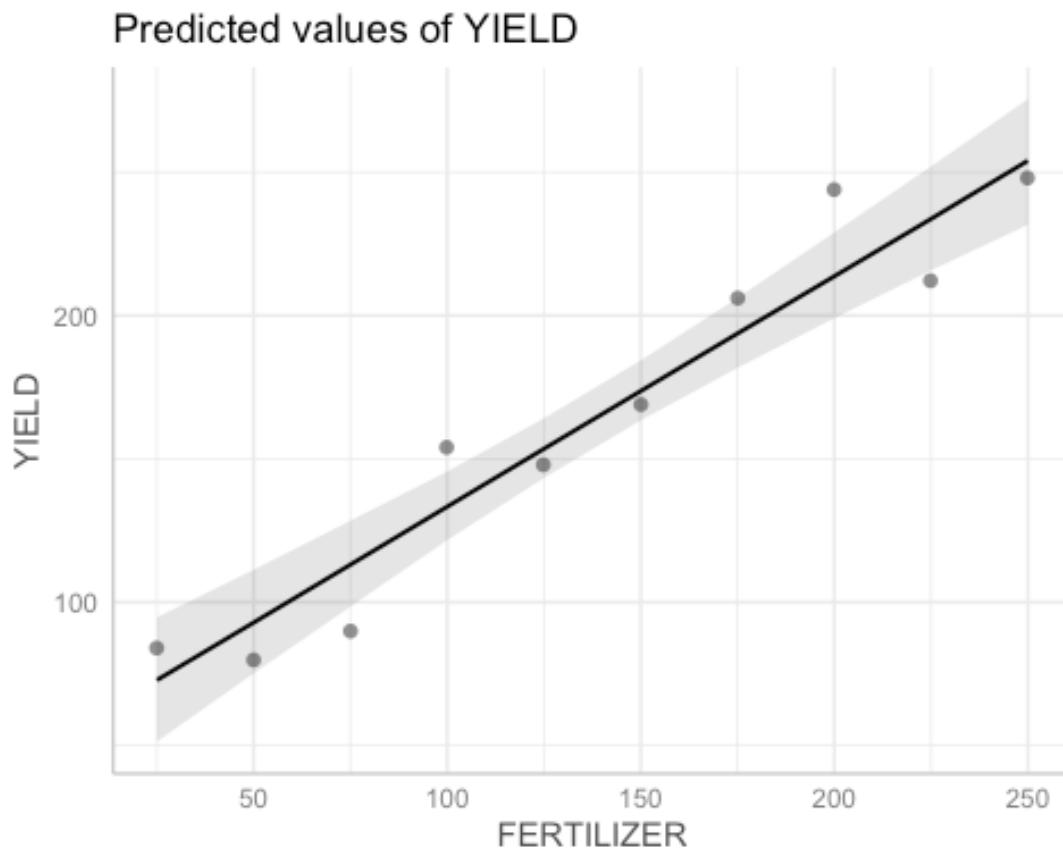
- simulated resp - observed values
- predicted resp (best fit values)
- obs resp (our raw response values, the YIELD in the example)
- the fitted pred results = the values along the line of best fit.

We tell R to take the predicted values, and calculate the median for each

To interpret these values, we'd do it the same way as for the other, frequentist models, and handle the problems (normality, etc) in the same way.

Partial effects plots

```
fert.rstanarm3 %>% ggpredict() %>% plot(add.data=TRUE)
## $FERTILIZER
```



tells us how the predicted values look like.

This plot

Model investigation

Sometimes, there are situations where niches open up, and we get massive rates of speciation. The same thing can happen with models. There was an opportunity to extract results, and EVERYONE came up with one. We got many different ways of achieving the same thing. Murray will give us a toolbox to start with, all of which are good.

```
fert.rstanarm3 %>% as.matrix %>% head

##          parameters
## iterations (Intercept) FERTILIZER      sigma
## [1,]    58.69204   0.7759527 14.81721
## [2,]    66.56636   0.7288457 29.29767
## [3,]    39.80162   0.9140677 18.10286
## [4,]    40.20463   0.8768917 18.20166
## [5,]    58.61159   0.7919139 24.79651
## [6,]    59.57668   0.7631511 15.91665
```

We have converted the Bayesian model into a matrix, and asked R to show us the first 6 rows (as there is gonna be heaps). Murray's output looks diff than mine, which is normal, as

each model creates different chains. We start with our likelihood, and multiply by our priors to get a posterior, which we sampled over and over again to get a distribution. This output ARE THOSE SAMPLES. Let's assume there were all the samples in this output => $5,000 - 1,000 = 4,000 / 5$ (thinning factor) = 800×3 (number of chains) = 2,400 estimates of the intercept, FERTILIZER, and the slope and sigma.

We can now take this output and make whatever stats we want (mean, median, best estimate of the slope, of sigma... Now, stats becomes just manipulating data really easily. It is as if we had the population in our hands. All of the hard bits is already done.

Let's do the following: ### Step 1 - We turn the output into a tibble

```
fert.mcmc<-fert.rstanarm3 %>% as.matrix %>% as_tibble

fert.mcmc

## # A tibble: 2,400 x 3
##   `(Intercept)` FERTILIZER sigma
##       <dbl>      <dbl>    <dbl>
## 1      58.7     0.776   14.8
## 2      66.6     0.729   29.3
## 3      39.8     0.914   18.1
## 4      40.2     0.877   18.2
## 5      58.6     0.792   24.8
## 6      59.6     0.763   15.9
## 7      53.2     0.802   16.0
## 8      57.5     0.817   18.9
## 9      15.8     1.04    25.1
## 10     25.5     0.918   21.4
## # ... with 2,390 more rows
```

We can now return: - the estimated intercept (mean)

```
fert.mcmc %>% summarise(Intercept=mean(`(Intercept)`)) # DO NOT REGULAR
QUOTATION MARKS, BUT BACK TICKS (=SPECIAL CHARACTERS FOR NAMES)
```

```
## # A tibble: 1 x 1
##   Intercept
##       <dbl>
## 1      52.9
```

- the estimated intercept (median)

```
fert.mcmc %>% summarise(Intercept=median(`(Intercept)`)) # DO NOT REGULAR
QUOTATION MARKS, BUT BACK TICKS (=SPECIAL CHARACTERS FOR NAMES)
```

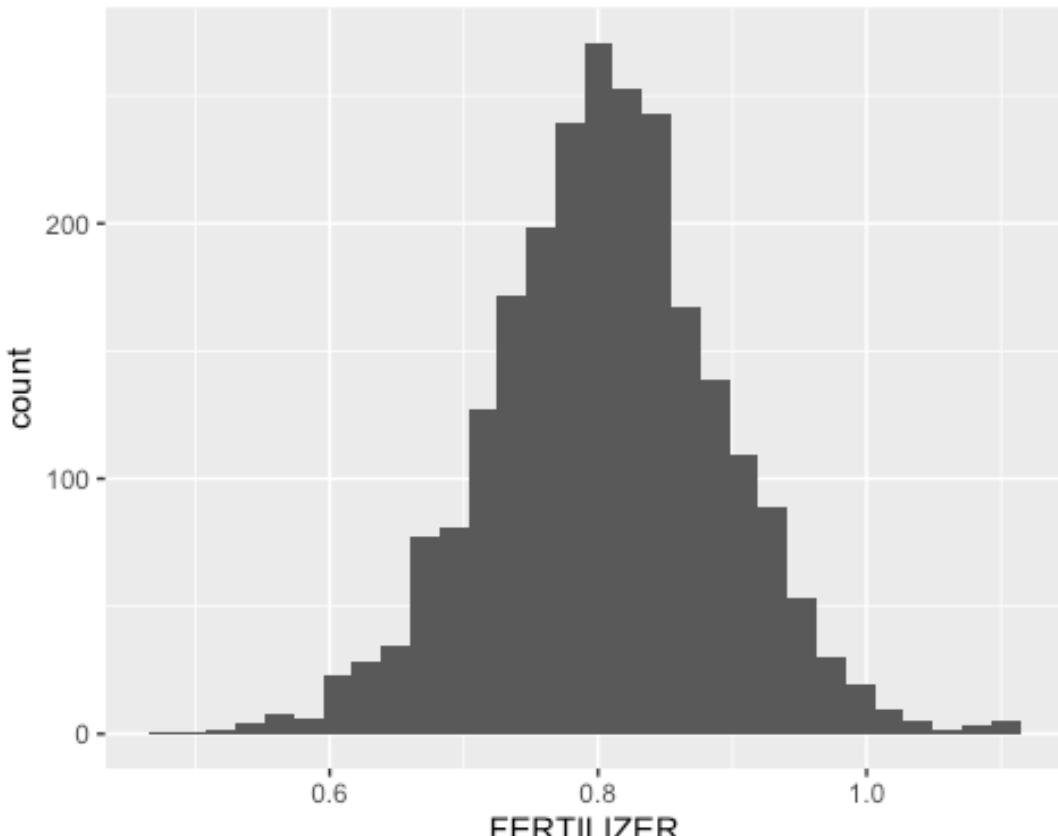
```
## # A tibble: 1 x 1
##   Intercept
##       <dbl>
## 1      52.6
```

- the estimated median

- the estimated slope, etc

Step 2 - quick plot

```
ggplot(fert.mcmc, aes(x=FERTILIZER)) + geom_histogram()  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



###

Step 3 - probabilities summary

```
fert.mcmc  
## # A tibble: 2,400 x 3  
##   `(Intercept)` FERTILIZER sigma  
##       <dbl>      <dbl> <dbl>  
## 1       58.7     0.776  14.8  
## 2       66.6     0.729  29.3  
## 3       39.8     0.914  18.1  
## 4       40.2     0.877  18.2  
## 5       58.6     0.792  24.8  
## 6       59.6     0.763  15.9  
## 7       53.2     0.802  16.0  
## 8       57.5     0.817  18.9  
## 9       15.8     1.04   25.1
```

```

## 10      25.5    0.918 21.4
## # ... with 2,390 more rows

sum(fert.mcmc$FERTILIZER>0)/2400
## [1] 1

```

We have 100% prob of having fertilizer >0

```

sum(fert.mcmc$FERTILIZER>0.5)/2400
## [1] 0.9995833

```

We have 99.9% prob of having fertilizer >0.5

Step 4 - Summary of the model (!)

```

summary(fert.rstanarm3)

##
## Model Info:
##   function:     stan_glm
##   family:      gaussian [identity]
##   formula:     YIELD ~ FERTILIZER
##   algorithm:   sampling
##   sample:      2400 (posterior sample size)
##   priors:      see help('prior_summary')
##   observations: 10
##   predictors:  2
##
## Estimates:
##             mean     sd    10%   50%   90%
## (Intercept) 52.9   12.8  37.0  52.6  69.1
## FERTILIZER   0.8    0.1   0.7   0.8   0.9
## sigma       19.2    5.0  14.1  18.3  25.6
##
## Fit Diagnostics:
##             mean     sd    10%   50%   90%
## mean_PPD 163.5   8.1 153.4 163.4 173.3
##
## The mean_ppd is the sample average posterior predictive distribution of
## the outcome variable (for details see help('summary.stanreg')).
##
## MCMC diagnostics
##             mcse Rhat n_eff
## (Intercept) 0.3  1.0  2450
## FERTILIZER  0.0  1.0  2505
## sigma       0.1  1.0  2153
## mean_PPD    0.2  1.0  2360
## log-posterior 0.0  1.0  1792
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude

```

measure of effective sample size, and Rhat is the potential scale reduction factor on split chains (at convergence Rhat=1).

It calculated the means along with the medians and 90% conf intervals for each of our parameters. These are rounded, but the data exists for the full set of digits.

sample: 2400 (posterior sample size) -> THIS NUMBER IS IMPORTANT, AS IT DETERMINES THE POSTERIOR SAMPLE SIZE, WHICH WE WILL USE FOR OUR CALCULATIONS AFTERWARDS

Our credibility intervals (prob that there is an effect) are the following: Estimates: mean sd 10% 50% 90% (Intercept) 52.7 12.9 36.8 53.0 68.7 FERTILIZER 0.8 0.1 0.7 0.8 0.9 sigma 19.0 4.9 13.8 18.1 25.3

This summary function also provides the MCMC diagnostics, which we will look into afterwards.

Step 5 - tidy output

This option provides a more compact output

```
tidyMCMC(fert.rstanarm3$stanfit,
           estimate.method="median",
           conf.int=TRUE, # we want conf int
           conf.method="HPDinterval", # way of calculating CI from a
           distribution. Quantiles are the simplest ways (95% quantiles). This is fine,
           but not very robust if the distribution is not symmetrical / normal. Highest
           probability density intervals are more robust to a non-even shape, so Murray
           recommends using these. They will almost always gonna be better than
           quantiles. They won't be symm around the medial, and will adapt to the shape
           of the distribution, providing more realistic conf intervals.
           rhat=TRUE, # diagnostics for convergence
           ess=TRUE) # diagnostics for efficiency

## # A tibble: 5 x 7
##   term      estimate std.error conf.low conf.high rhat    ess
##   <chr>     <dbl>     <dbl>     <dbl>     <dbl> <dbl> <int>
## 1 (Intercept) 52.6     12.8     28.4     79.0   1.00  2450
## 2 FERTILIZER  0.806    0.0855    0.637    0.979   1.00  2505
## 3 sigma       18.3     4.96     11.4     29.0   1.00  2153
## 4 mean_PPD    163.     8.08     147.     179.   1.00  2360
## 5 log-posterior -50.9    1.34    -53.9    -49.6   1.00  1792
```

This table is what we can report in our publication. The last two rows are not important. They are additional parameters which do not need to be reported.

Step 6 - R sq value

```
fert.rstanarm3 %>% bayes_R2() %>% median_hdci

##          y      ymin      ymax .width .point .interval
## 1 0.917637 0.7921361 0.9714954  0.95 median      hdci
```

This code calculates the highest density interval, the median. This value is our median R sq value. It gives us the upper and lower 95% hdci intervals.

Why do we specify “median” in the code? IF we had a bimodal distribution, the HDCI would return TWO rows as it would recognise the distribution as two models. This is not useful, so we tell HDCI to only give us the median value.

```
fert.rstanarm3 %>% bayes_R2() %>% head  
## [1] 0.9401625 0.7800097 0.9359252 0.9300564 0.8538740 0.9294319
```

We can now assess the prob of getting specific R sq

```
a=fert.rstanarm3 %>% bayes_R2()  
  
sum(a>0.9)/2400 # 2400 predictions  
## [1] 0.65375
```

The prob of having an Rsq of 0.9 is 0.66

We could heck this with any other parameter, not only with R sq.

Predictions

Step 1 - We select the values we are interested in

Example: We want to know, with a fertiliser of 110, what is the yield?

```
newdata=data.frame(FERTILIZER=110)
```

Step 2 - Predictions of a specific value

We now run 2400 predictions for what the yield would be if we used fertilizer of 110

```
fert.rstanarm3 %>% fitted_draws(newdata=newdata) %>%  
  median_hdci  
  
## Instead of posterior_linpred(..., transform=TRUE) please call  
posterior_epred(), which provides equivalent functionality.  
  
## # A tibble: 1 x 8  
## # Groups:   FERTILIZER [1]  
##   FERTILIZER .row .value .lower .upper .width .point .interval  
##       <dbl> <int> <dbl> <dbl> <dbl> <dbl> <chr>  <chr>  
## 1      110     1   141.   130.   152.    0.95 median hdci
```

In this case, R tells us:

“For our fertilizer value 110, we expect... - to get a YIELD concentration of 141.4 - that our upper and lower CI are 130 and 152”

Step 3 - Predicted values

```
fert.rstanarm3 %>% emmeans(~FERTILIZER, at=newdata)

## FERTILIZER emmean lower.HPD upper.HPD
##      110     141     130     152
##
## Point estimate displayed: median
## HPD interval probability: 0.95
```

We could ask R what the prob is that the fertiliser would be >142 if we used a fertilizer dose of 110: In this case, it would be 50%.

Let's check how much YIELD we get if we increase our fertilizer dose from 100 to 200:

```
newdata=list(FERTILIZER=c(200,100))

fert.rstanarm3 %>% emmeans(pairwise~FERTILIZER, at=newdata)

## $emmeans
## FERTILIZER emmean lower.HPD upper.HPD
##      200     214     199     229
##      100     133     122     146
##
## Point estimate displayed: median
## HPD interval probability: 0.95
##
## $contrasts
## contrast estimate lower.HPD upper.HPD
## 200 - 100    80.6     63.7    97.9
##
## Point estimate displayed: median
## HPD interval probability: 0.95

# this is another way of writing it, but with summarised output:
# fert.rstanarm3 %>% emmeans(~FERTILIZER, at=newdata) %>% pairs()
```

Our avg effect, when we move from a concentration of 100 up to 200, we increase the yield by on average 80. We can also see the conf int here.

Alternative way

```
fert.mcmc<-fert.rstanarm3 %>%
  emmeans(~FERTILIZER, at=newdata) %>%
  tidy_draws() # Instead of doing summaries, R is gonna give us all 2400
  predictions for 100 and 200 fertiliser concentrations

fert.mcmc

## # A tibble: 2,400 x 5
##   .chain .iteration .draw `FERTILIZER 200` `FERTILIZER 100`
##   <int>     <int> <int>          <dbl>          <dbl>
```

```

## 1 1 1 1 214. 136.
## 2 1 2 2 212. 139.
## 3 1 3 3 223. 131.
## 4 1 4 4 216. 128.
## 5 1 5 5 217. 138.
## 6 1 6 6 212. 136.
## 7 1 7 7 213. 133.
## 8 1 8 8 221. 139.
## 9 1 9 9 224. 120.
## 10 1 10 10 209. 117.
## # ... with 2,390 more rows

```

And the following command would give us a summary of the above (the summary is not really needed though)

```

fert.mcmc_summary<-fert.rstanarm3 %>%
  emmeans(~FERTILIZER, at=newdata) %>%
  pairs() %>%
  as.data.frame()

fert.mcmc_summary

##   contrast estimate lower.HPD upper.HPD
## 1 200 - 100 80.58509 63.71909 97.94689

```

We now specify the differences btw fertiliser concentrations

```

fert.mcmc<-fert.rstanarm3 %>%
  emmeans(~FERTILIZER, at=newdata) %>%
  tidy_draws() %>%
  rename_with(~str_replace(., "FERTILIZER ", "p")) %>% # We replace all rows
with columns saying FERTILIZER for "p" (make sure you include the space after
FERTILIZER, as the original column name was FERTILIZER...number)
  mutate(Eff=p200-p100, # calculates the diff btw fertilizer (p) at 200 conc
and 100 conc. This will give us the absolute diff btw the 2 predictions
  PEff=100*Eff/p100) # gives us the % change in YIELD

fert.mcmc %>% head

## # A tibble: 6 x 7
##   .chain .iteration .draw  p200  p100    Eff  PEff
##     <int>      <int> <int> <dbl> <dbl> <dbl> <dbl>
## 1     1          1     1  214.  136.  77.6  56.9
## 2     1          2     2  212.  139.  72.9  52.3
## 3     1          3     3  223.  131.  91.4  69.7
## 4     1          4     4  216.  128.  87.7  68.6
## 5     1          5     5  217.  138.  79.2  57.5
## 6     1          6     6  212.  136.  76.3  56.2

```

We got now: - EFF: the diff btw fertiliser at 100 conc and at 200 conc (renamed as p100 and p200). - PEff: The % change

We display now the estimates, SD and conf int

```
fert.mcmc %>% tidyMCMC(estimate.method="median",
                           conf.int=TRUE,
                           conf.method="HPDinterval")

## # A tibble: 7 x 5
##   term      estimate std.error conf.low conf.high
##   <chr>     <dbl>     <dbl>     <dbl>     <dbl>
## 1 .chain      2       0.817      1        3
## 2 .iteration  400.    231.       1        761
## 3 .draw       1200.   693.       1       2281
## 4 p200       214.    7.45       199.     229.
## 5 p100       133.    6.08       122.     146.
## 6 Eff        80.6    8.55       63.7     97.9
## 7 PEff       60.4    8.24       44.4     76.7
```

We can now ask the following questions:

What is the prob that yield will increase?

```
fert.mcmc %>% summarise(P=sum(Eff>0)/n()) # Eff>0 = increase in YIELD

## # A tibble: 1 x 1
##       P
##   <dbl>
## 1     1
```

Chances that we'll increase YIELD are 100%.

What is the prob that the yield will increase >50%?

```
fert.mcmc %>% summarise(p=sum(PEff>50)/n())

## # A tibble: 1 x 1
##       p
##   <dbl>
## 1 0.903
```

The prob that yield will increase >50% is 91%.

Hypothesis testing

IGNORE THE FOLLOWING

Summary figures

rstanarm

brms

References

15_Bayesian_LinearRegr

Sara Kophamel

15/12/2020

Preparations

Load the necessary libraries

```
library(rstanarm)    #for fitting models in STAN

## Loading required package: Rcpp

## This is rstanarm version 2.21.1

## - See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!

## - Default priors may change, so it's safest to specify priors, even if equivalent to the defaults.

## - For execution on a local, multicore CPU with excess RAM we recommend calling

##   options(mc.cores = parallel::detectCores())

library(brms)        #for fitting models in STAN

## Loading 'brms' package (version 2.14.4). Useful instructions
## can be found by typing help('brms'). A more detailed introduction
## to the package is available through vignette('brms_overview').

##
## Attaching package: 'brms'

## The following objects are masked from 'package:rstanarm':
## 
##     dirichlet, exponential, get_y, lasso, ngrps

## The following object is masked from 'package:stats':
## 
##     ar

library(coda)         #for diagnostics
library(bayesplot)   #for diagnostics

## This is bayesplot version 1.7.2

## - Online documentation and vignettes at mc-stan.org/bayesplot
```

```

## - bayesplot theme set to bayesplot::theme_default()
##   * Does _not_ affect other ggplot2 plots
##   * See ?bayesplot_theme_set for details on theme setting

library(ggmcmc)      #for MCMC diagnostics

## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
## 
##   filter, lag

## The following objects are masked from 'package:base':
## 
##   intersect, setdiff, setequal, union

## Loading required package: tidyr

## Loading required package: ggplot2

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

library(DHARMa)      #for residual diagnostics

## This is DHARMa 0.3.3.0. For overview type '?DHARMa'. For recent changes,
## type news(package = 'DHARMa') Note: Syntax of plotResiduals has changed in
## 0.3.0, see ?plotResiduals for details

library(rstan)        #for interfacing with STAN

## Loading required package: StanHeaders

## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend
## calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)

##
## Attaching package: 'rstan'

## The following object is masked from 'package:tidyR':
## 
##   extract

```

```

## The following object is masked from 'package:coda':
##
##     traceplot

library(emmeans)      #for marginal means etc
library(broom)         #for tidying outputs
library(broom.mixed)  #for tidying outputs

## Registered S3 method overwritten by 'broom.mixed':
##   method      from
##   tidy.gamlss broom

library(tidybayes)  #for more tidying outputs

##
## Attaching package: 'tidybayes'

## The following objects are masked from 'package:brms':
##
##     dstudent_t, pstudent_t, qstudent_t, rstudent_t

library(ggeffects)  #for partial plots
library(tidyverse)   #for data wrangling etc

## — Attaching packages

```

```

                                         tidyverse 1.3.0 —

## ✓ tibble  3.0.3      ✓ stringr 1.4.0
## ✓ readr   1.3.1      ✓forcats 0.5.0
## ✓ purrr   0.3.4

## — Conflicts

```

```

                                         tidyverse_conflicts() —

## x rstan:::extract() masks tidyrr:::extract()
## x dplyr:::filter()  masks stats:::filter()
## x dplyr:::lag()     masks stats:::lag()

library(broom.mixed) #for summarising models
library(ggeffects)  #for partial effects plots
theme_set(theme_grey()) #put the default ggplot theme back

```

Scenario

@Polis-1998-490 were interested in modelling the presence/absence of lizards (*Uta* sp.) against the perimeter to area ratio of 19 islands in the Gulf of California.

Uta lizard

Uta lizard

Format of polis.csv data file

ISLAND	RATIO	PA
Bota	15.41	1
Cabeza	5.63	1
Cerraja	25.92	1
Coronadito	15.17	0
..
ISLAND	Categorical listing of the name of the 19 islands used - variable not used in analysis.	
RATIO	Ratio of perimeter to area of the island.	
PA	Presence (1) or absence (0) of <i>Uta</i> lizards on island.	

The aim of the analysis is to investigate the relationship between island parameter to area ratio and the presence/absence of *Uta* lizards.

Read in the data

```
polis = read_csv('data/polis.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   ISLAND = col_character(),
##   RATIO = col_double(),
##   PA = col_double()
## )

glimpse(polis)

## # Rows: 19
## # Columns: 3
## # $ ISLAND <chr> "Bota", "Cabeza", "Cerraja", "Coronadito", "Flecha",
## "Gemelose..." ...
## # $ RATIO <dbl> 15.41, 5.63, 25.92, 15.17, 13.04, 18.85, 30.95, 22.87,
## 12.01, ...
## # $ PA <dbl> 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1

head(polis)

## # A tibble: 6 x 3
##   ISLAND      RATIO     PA
##   <chr>      <dbl> <dbl>
## 1 Bota       15.4     1
## 2 Cabeza    5.63     1
## 3 Cerraja   25.9     1
## 4 Coronadito 15.2     0
```

```

## 5 Flecha    13.0      1
## 6 Gemelose   18.8      0

str(polis)

## # tibble [19 x 3] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
## # $ ISLAND: chr [1:19] "Bota" "Cabeza" "Cerraja" "Coronadito" ...
## # $ RATIO : num [1:19] 15.41 5.63 25.92 15.17 13.04 ...
## # $ PA     : num [1:19] 1 1 1 0 1 0 0 0 0 1 ...
## # - attr(*, "spec")=
## #   .. cols(
## #     .. ISLAND = col_character(),
## #     .. RATIO = col_double(),
## #     .. PA = col_double()
## #   )

```

We got binomial data (presence or absence). The only possible distribution is Binomial. We are therefore gonna fit a Logistic Regression, which uses the binomial.

Exploratory data analysis

We'll skip this, but we'd do it in the same way we learned it for frequentist analysis.

Model formula:

$$\begin{aligned}
 y_i &\sim \text{Bin}(n, p_i) \\
 \ln\left(\frac{p_i}{1-p_i}\right) &= \beta_0 + \beta_1 x_i \\
 \beta_0 &\sim \mathcal{N}(0, 10) \\
 \beta_1 &\sim \mathcal{N}(0, 1)
 \end{aligned}$$

$N=1$ (1 trial per islands), so we don't specify it. There is also no sigma.

We have to consider these values on the link scale = We are not just working on the raw data, but on a link scale.

Fit the model

Prior selection

For stan models (NUTS models), 5000 iterations should be enough to settle itself

```

polis.rstanarm=stan_glm(PA~RATIO,
                        data=polis,
                        family=binomial(),
                        iter=5000,
                        warmup=1000,
                        chains=3,

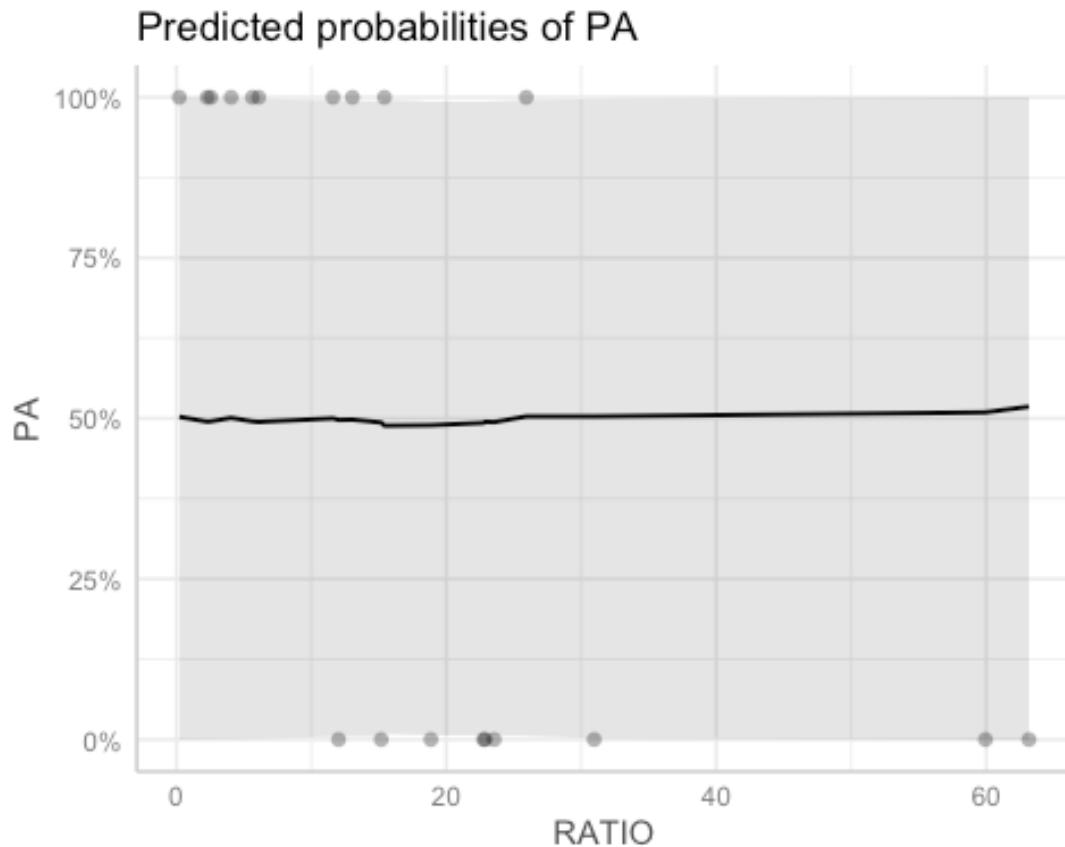
```

```
    thin=5,  
    refresh=0)  
  
prior_summary(polis.rstanarm)  
  
## Priors for model 'polis.rstanarm'  
## -----  
## Intercept (after predictors centered)  
## ~ normal(location = 0, scale = 2.5)  
##  
## Coefficients  
##   Specified prior:  
##     ~ normal(location = 0, scale = 2.5)  
##   Adjusted prior:  
##     ~ normal(location = 0, scale = 0.14)  
## -----  
## See help('prior_summary.stanreg') for more details
```

R has decided... - for the Intercept: standard 0 and 2.5 priors . it also calculated an adjusted Intercept, but because it was <2.5, it got removed and does not appear here - for the Coefficients: - the Adjusted prior indicates the slope (0 and 0.14)

Prior check

```
polis.rstanarm1<-update(polis.rstanarm,prior_PD=TRUE)  
  
ggemmeans(polis.rstanarm1,~RATIO) %>% plot(add.data=TRUE)
```



Our prior is not influencing our trends at all, and is VERY vague. We might wanna narrow it down.

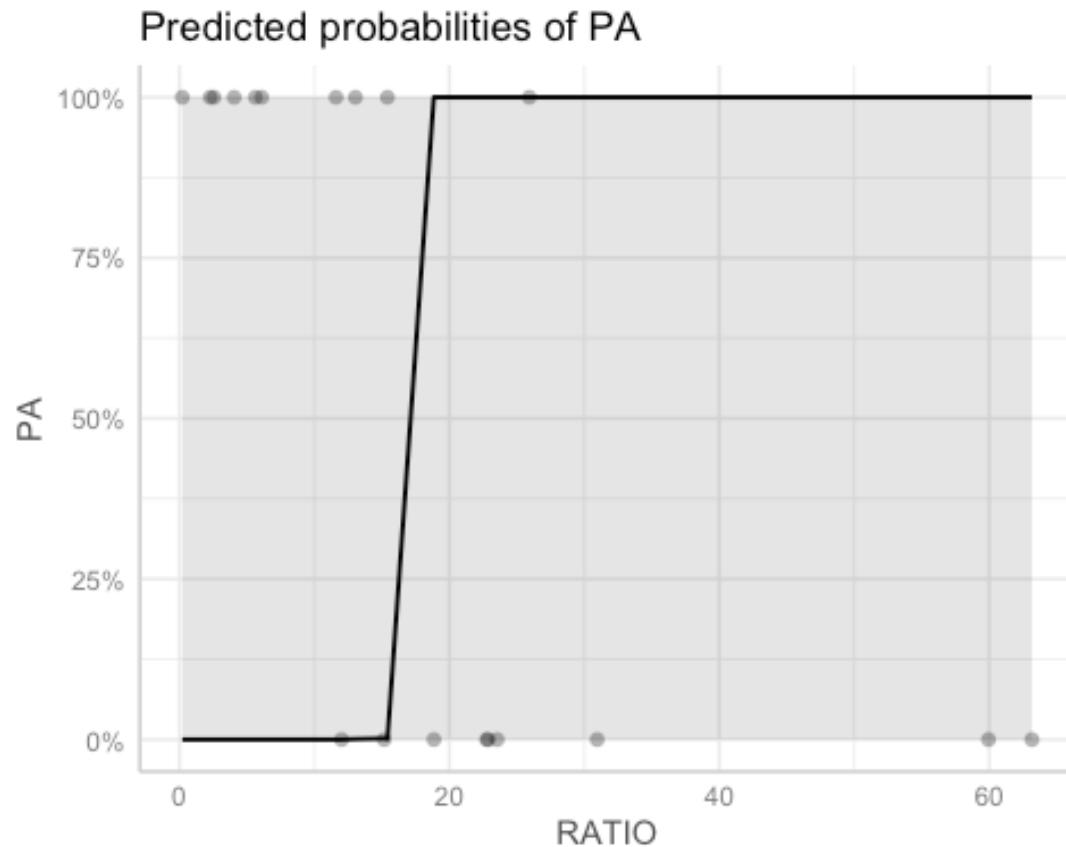
Narrowing down the prior, but still allowing it to be pretty vague

For simple models, Murray likes to make them quite wide. If you make them narrow, you have to be able to justify why. But if they are really wide, noone is gonna question you.

Wide option (made up by me - IGNORE)

```
polis.rstanarm_wide=stan_glm(PA~RATIO,
                               data=polis,
                               family=binomial(),
                               prior_intercept=normal(10,100,autoscale=FALSE),
                               prior=normal(1,100,autoscale=FALSE),
                               prior_PD=TRUE,
                               iter=5000,
                               warmup=1000,
                               chains=3,
                               thin=5,
                               refresh=0)

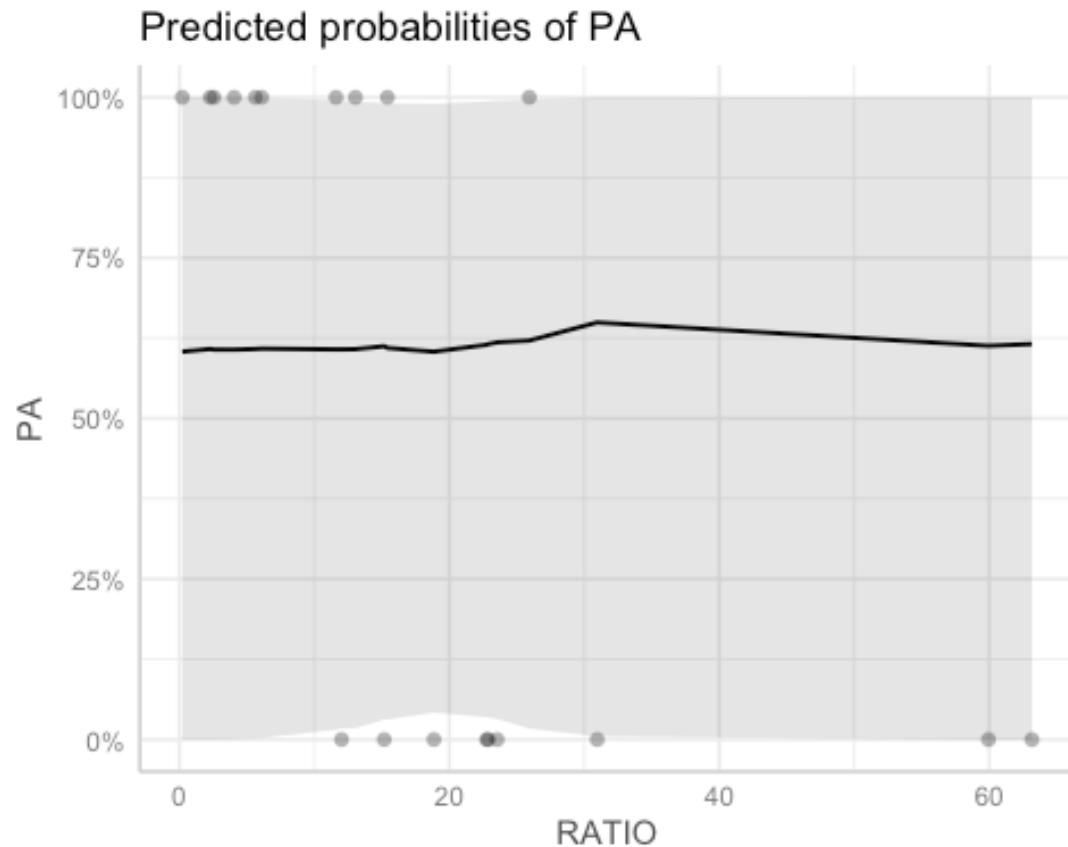
ggemmeans(polis.rstanarm_wide,~RATIO) %>%
  plot(add.data=TRUE)
```



Narrow option (we asked Murray to do a narrow one)

```
polis.rstanarm2=stan_glm(PA~RATIO,
                           data=polis,
                           family=binomial(),
                           prior_intercept=normal(0.5,2,autoscale=FALSE),
                           prior=normal(0,0.2,autoscale=FALSE), #
                           autoscale=FALSE as we already considered the priors we want; we don't have to
                           scale them to our data
                           prior_PD=TRUE, # only the priors are involved in this
                           model, as we wanna see whether our priors look vaguely sensible (is the
                           scheme not too wide and not too narrow?)
                           iter=5000,
                           warmup=1000,
                           chains=3,
                           thin=5,
                           refresh=0)

ggemmeans(polis.rstanarm2,~RATIO) %>%
  plot(add.data=TRUE)
```



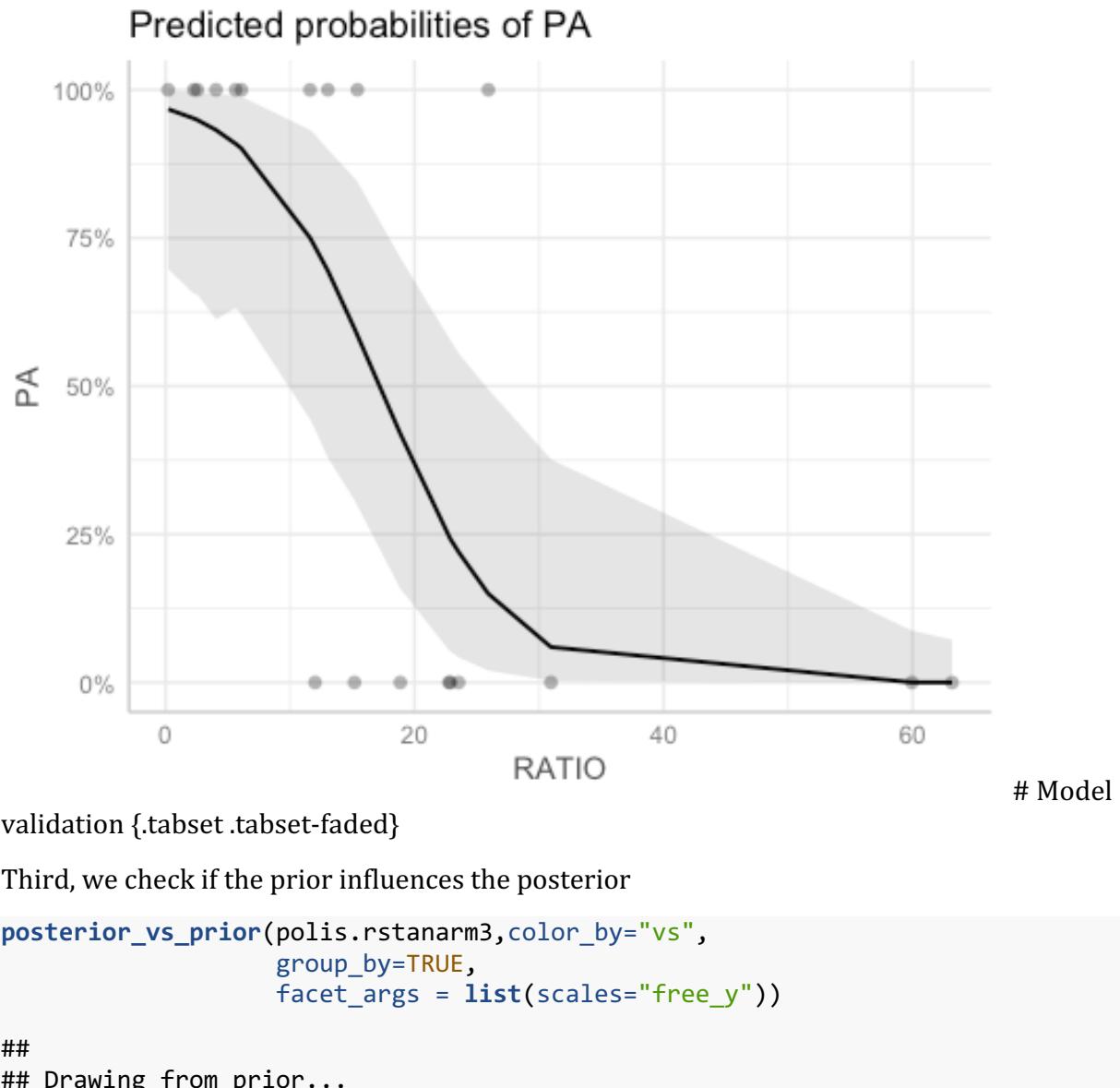
Plot the predicted values with the narrow prior

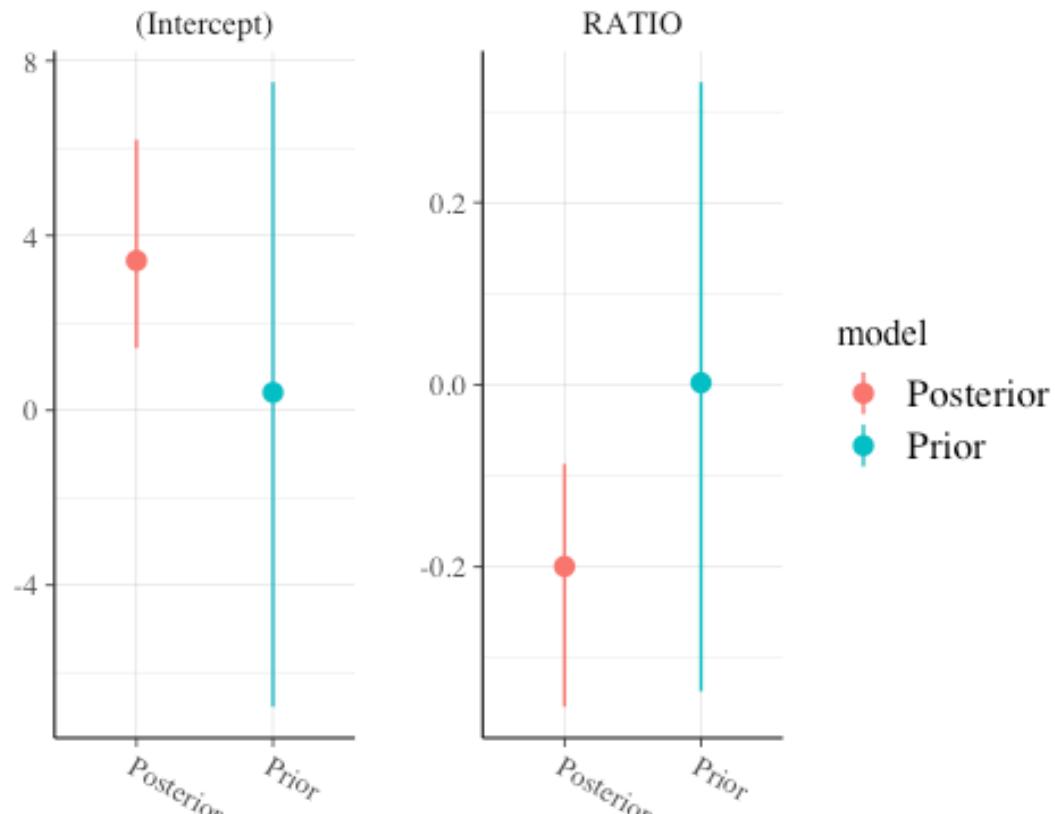
First, we say we want to create a model with more than only the priors

```
polis.rstanarm3=update(polis.rstanarm2,prior_PD=FALSE)
```

Second, we plot the prediction using the defined prior

```
ggemmeans(polis.rstanarm3,~RATIO) %>%  
  plot(add.data=TRUE)
```





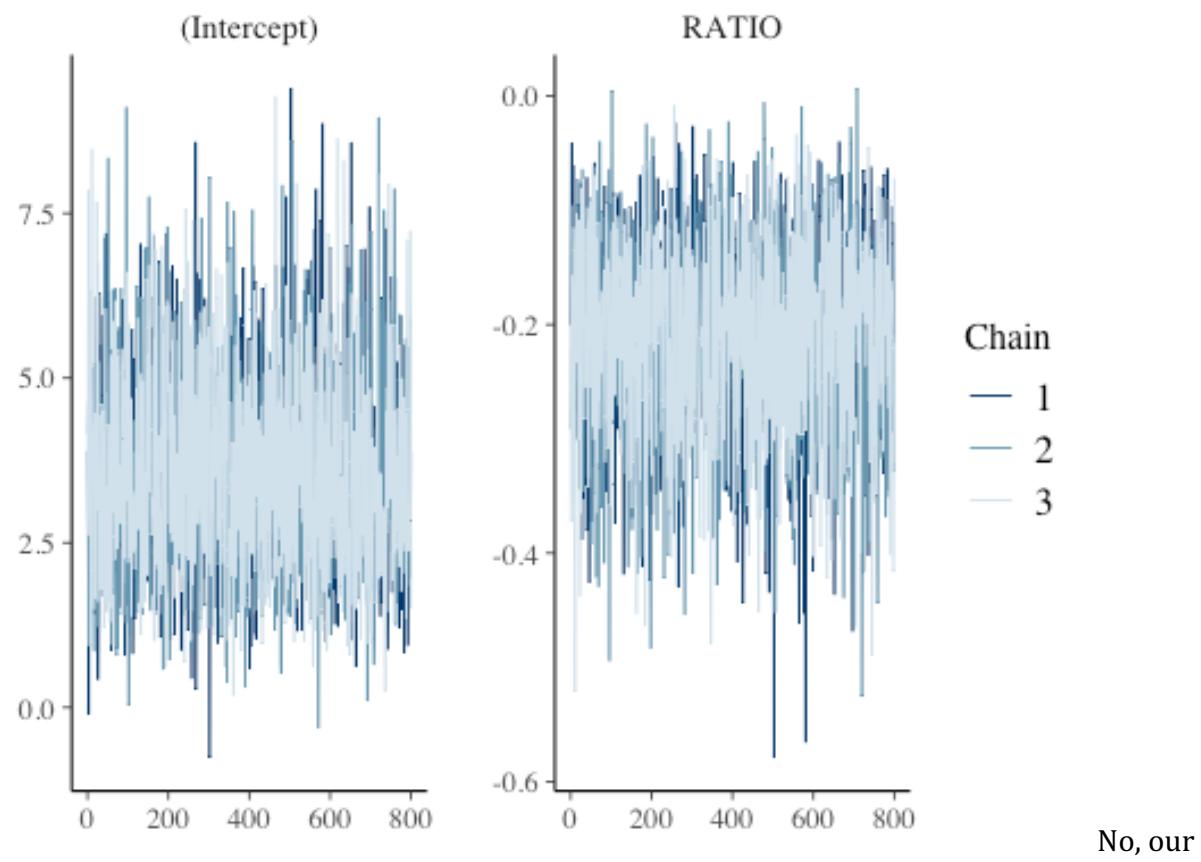
prior does not seem to influence our posterior

Our

Has the sample converged on a sample posterior?

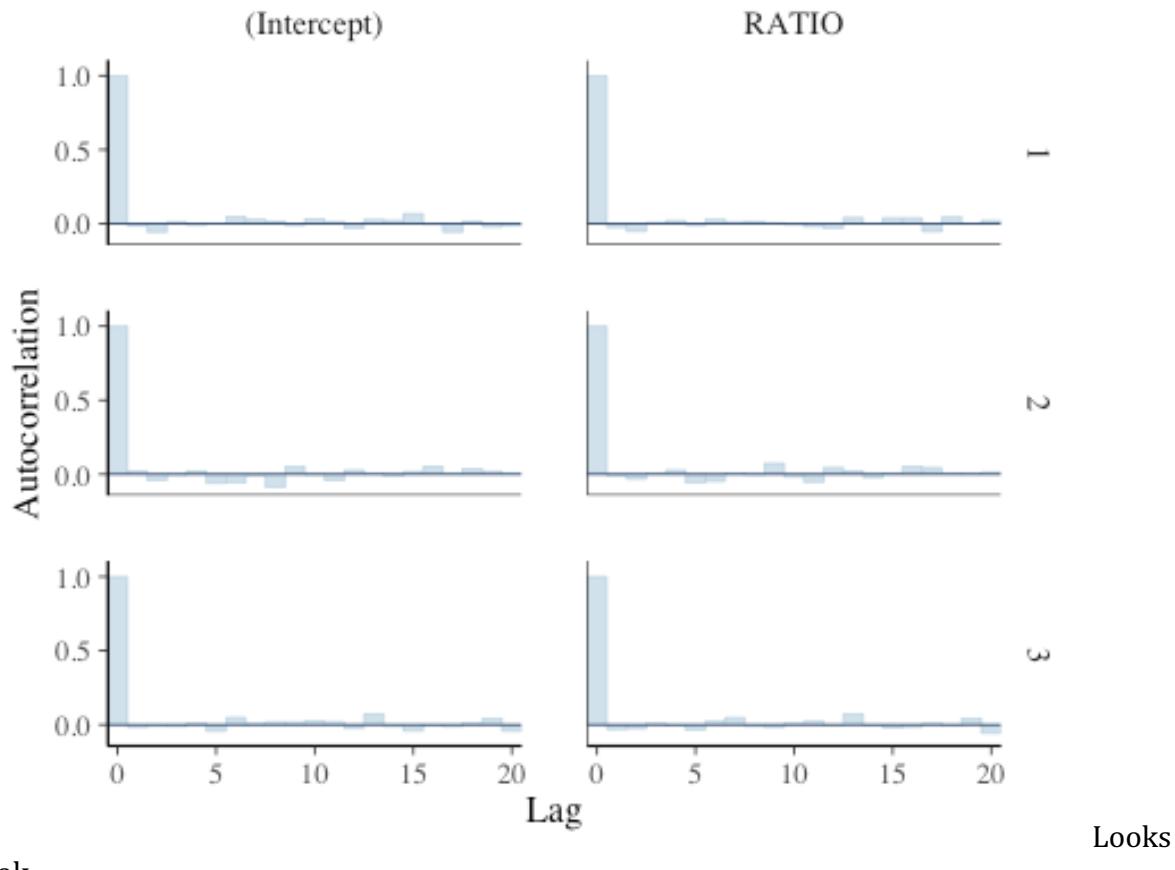
1. Trace plots

```
plot(polis.rstanarm3, plotfun="mcmc_trace")
```



2. Autocorrelation function for thinning

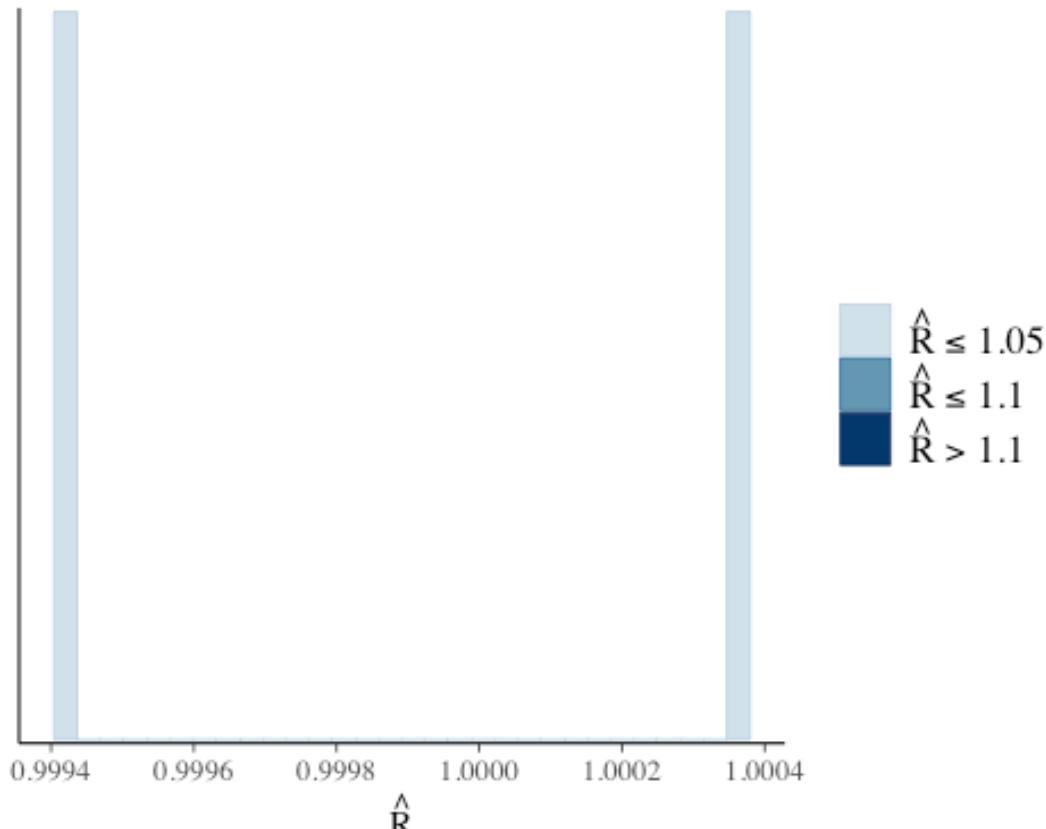
```
plot(polis.rstanarm3, "acf_bar")
```



ok

3. Measure of chain convergence - R hat

```
plot(polis.rstanarm3, "rhat_hist")  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

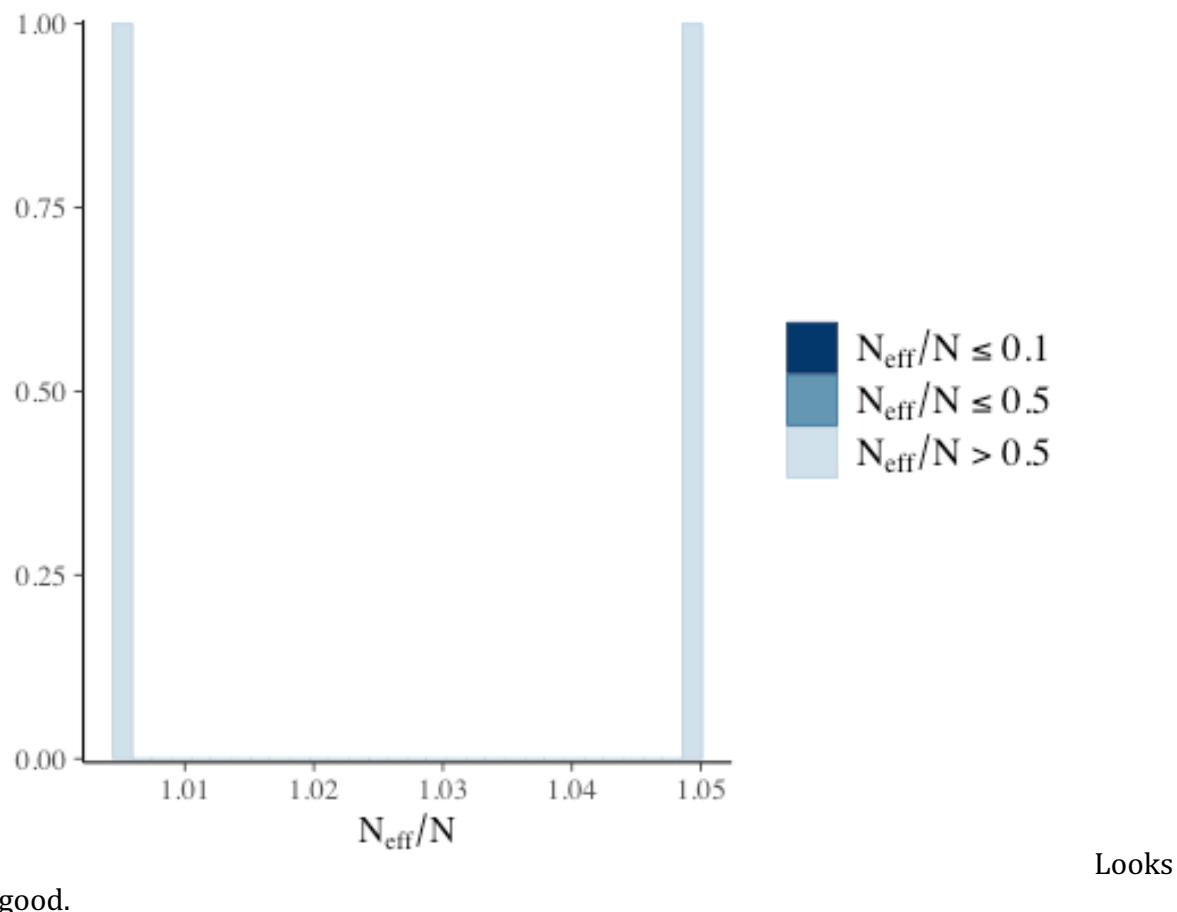


all < 1.05 . so that's good

They are

4. Number of effective samples

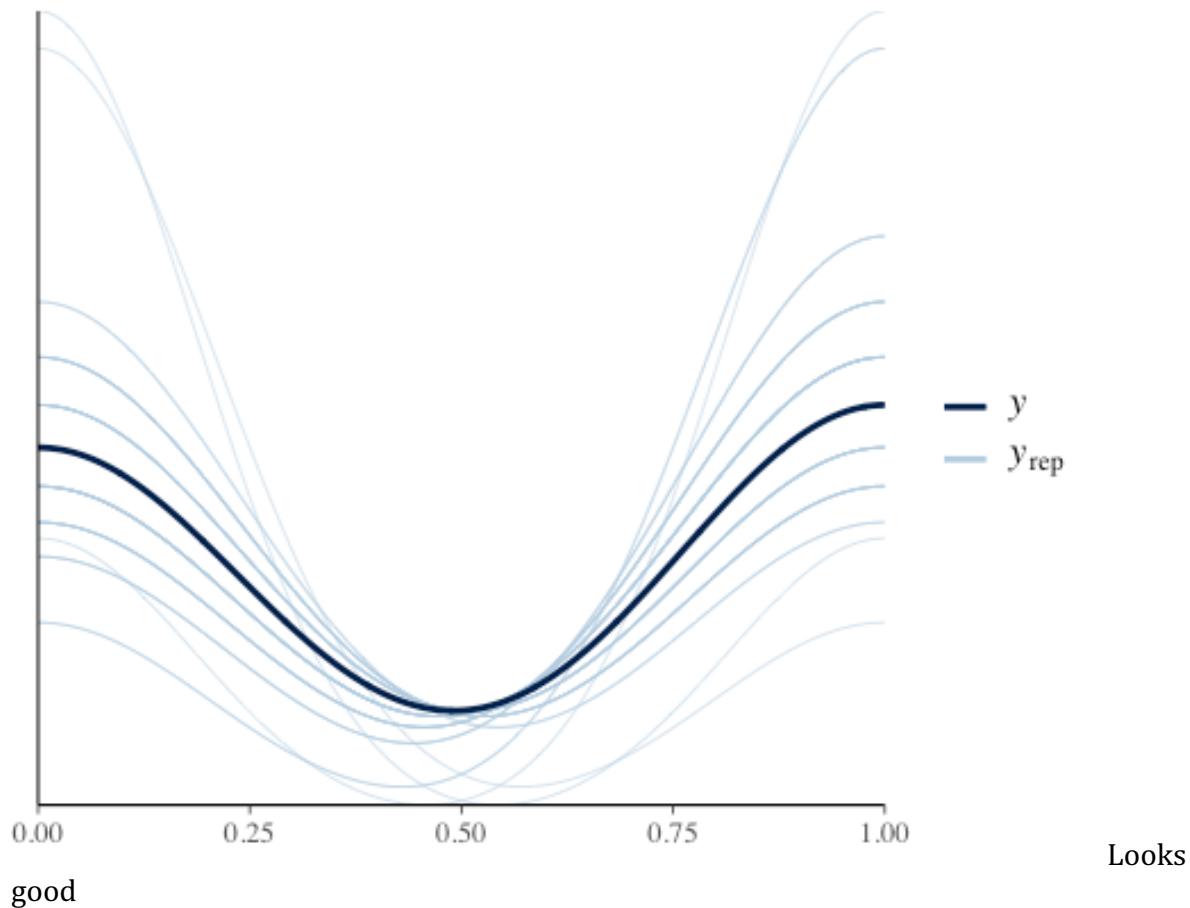
```
plot(polis.rstanarm3, "neff_hist")  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Posterior checks - Is the model predicting the data well?

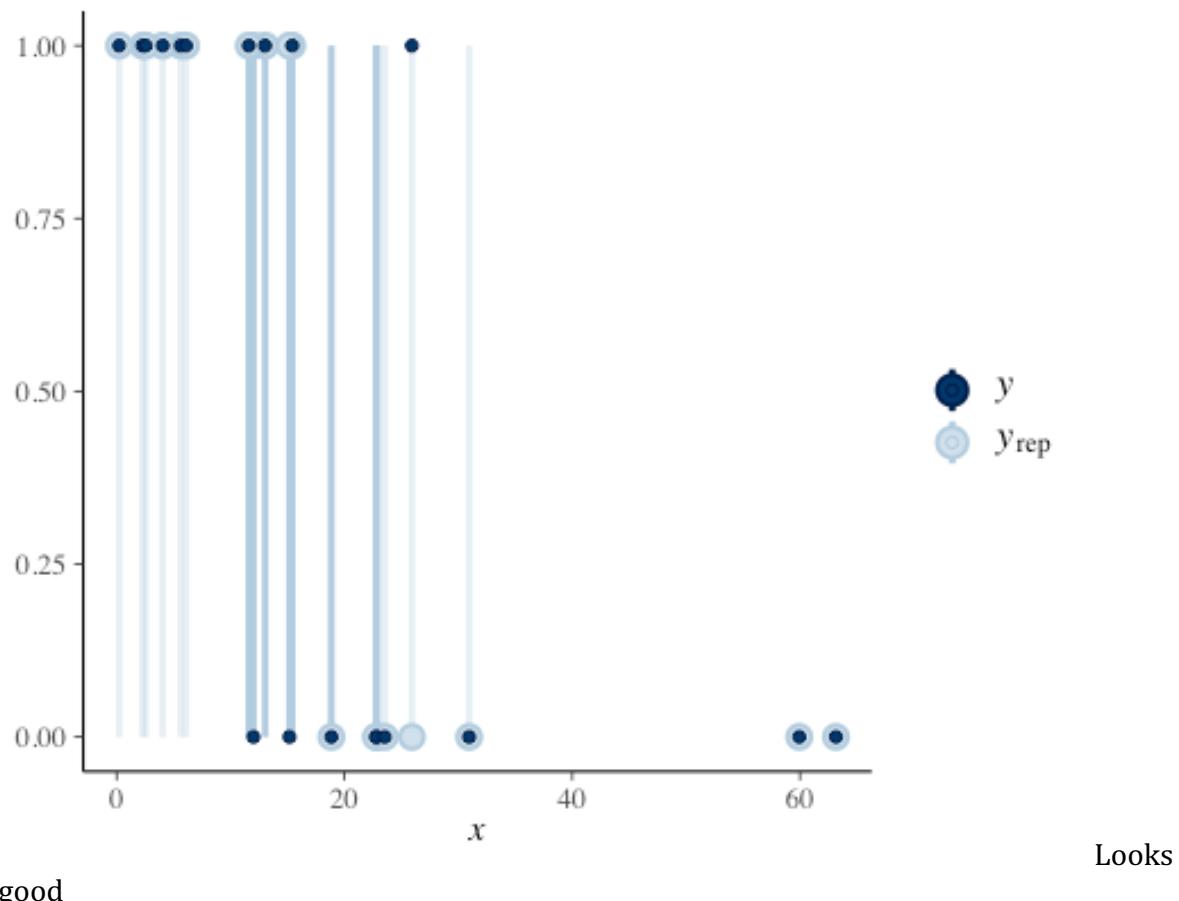
1. Observed data and the scope of our predictions

```
pp_check(polis.rstanarm3, plotfun="dens_overlay")
```

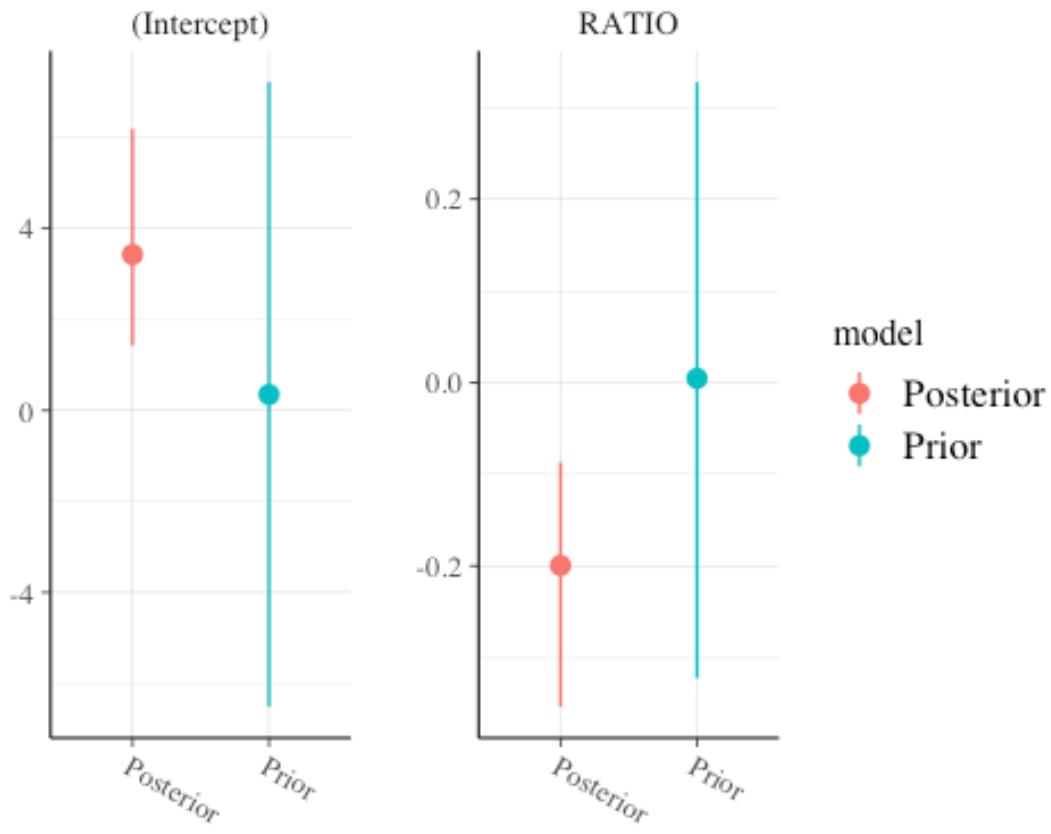


4. Number of effective samples

```
pp_check(polis.rstanarm3,x=polis$RATIO, plotfun="intervals")
```



```
posterior_vs_prior(polis.rstanarm3,color_by="vs",group_by=TRUE,  
                   facet_args=list(scales="free_y"))  
  
##  
## Drawing from prior...
```



Looks

good, as the posterior is smaller than the prior.

If they were equally large, our prior would have impact on the posterior, which we don't want to happen.

A bad graph would be: I I I pos prior This would indicate that the predictions are driven by our prior.

A good graph is: I I I pos prior Which indicates that the posteriors are driving the predictions.

DHARMA residuals

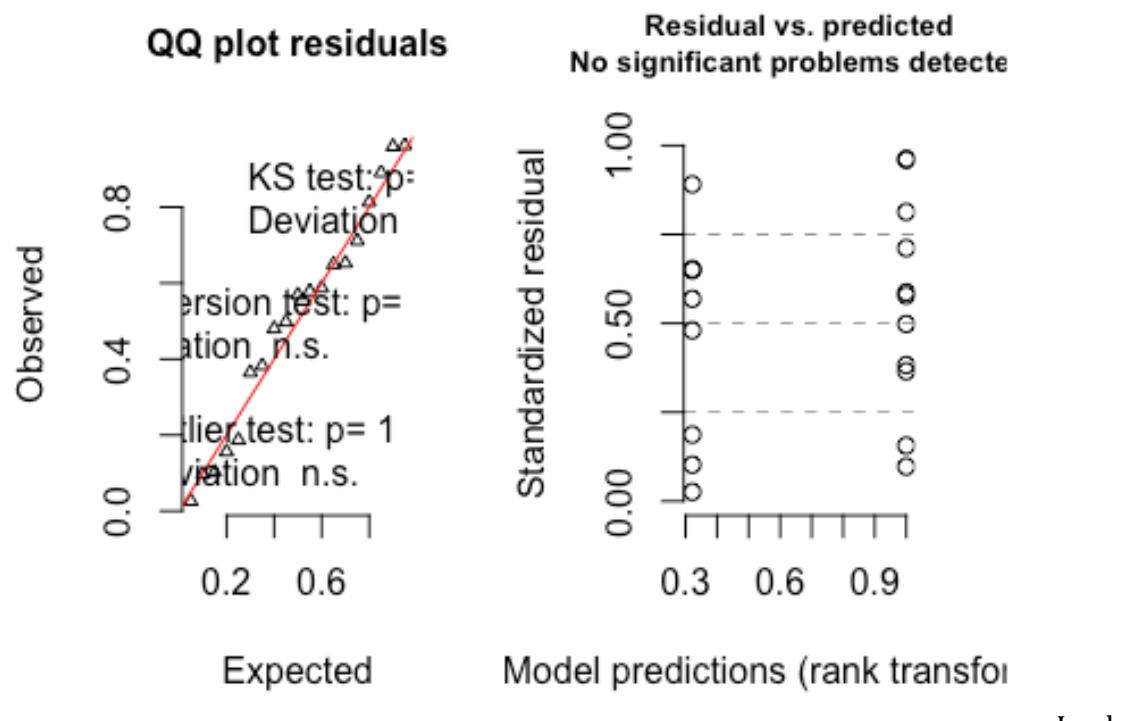
```
preds<- posterior_predict(polis.rstanarm3, nsamples=250, summary=FALSE)

polis.resids<-createDHARMA(simulatedResponse = t(preds),
                             observedResponse = polis$PA,
                             fittedPredictedResponse = apply(preds, 2, median),
                             integerResponse = TRUE) # Only set it as TRUE if
there is count data. If there were counts, R would treat our data as if it
was a Poisson distribution, and would sample accordingly.

plot(polis.resids)
```

```
## Warning in smooth.construct.tp.smooth.spec(object, dk$data, dk$knots):  
basis dimension, k, increased to minimum possible  
  
## Unable to calculate quantile regression for quantile 0.25. Possibly to few  
(unique) data points / predictions. Will be ommited in plots and significance  
calculations.  
  
## Warning in smooth.construct.tp.smooth.spec(object, dk$data, dk$knots):  
basis dimension, k, increased to minimum possible  
  
## Unable to calculate quantile regression for quantile 0.5. Possibly to few  
(unique) data points / predictions. Will be ommited in plots and significance  
calculations.  
  
## Warning in smooth.construct.tp.smooth.spec(object, dk$data, dk$knots):  
basis dimension, k, increased to minimum possible  
  
## Unable to calculate quantile regression for quantile 0.75. Possibly to few  
(unique) data points / predictions. Will be ommited in plots and significance  
calculations.
```

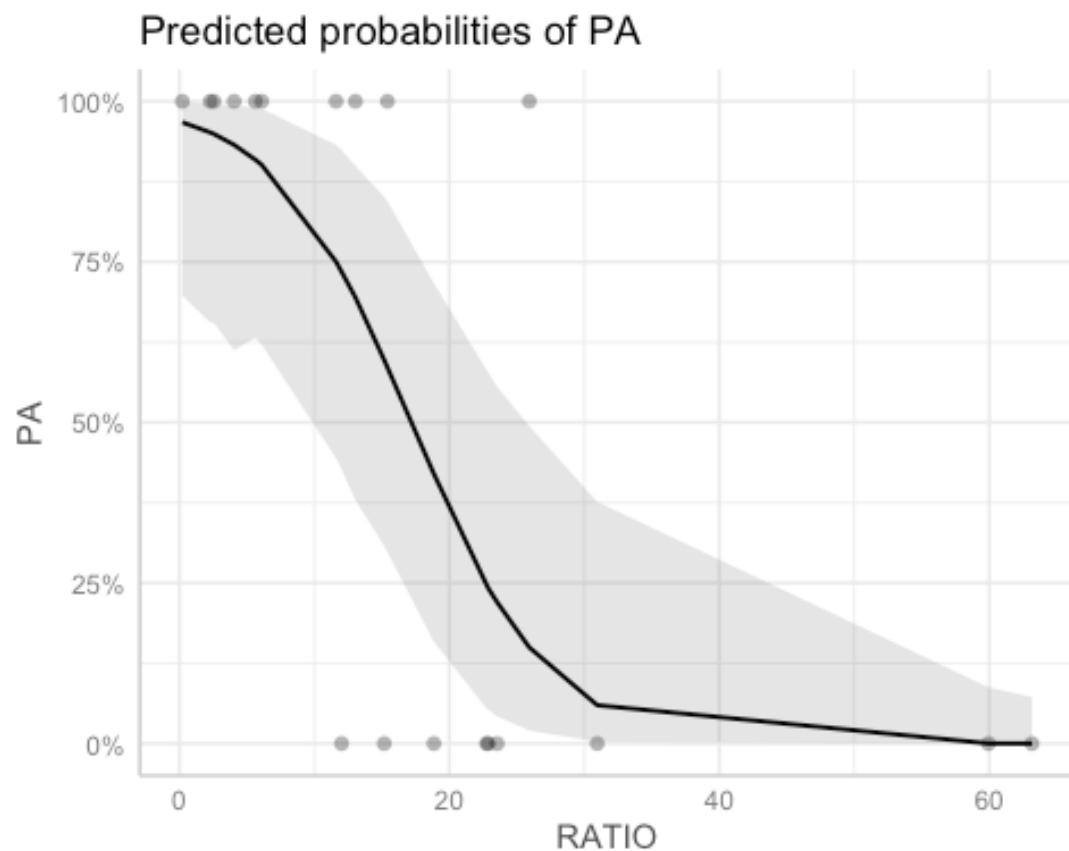
DHARMA residual diagnostics



good.

Partial plot

```
polis.rstanarm3 %>%  
  ggmeans(~RATIO) %>%  
  plot(add.data=TRUE)
```



We will

use this plot to interpret our summary statistics

Summary stats

```
summary(polis.rstanarm3)  
  
##  
## Model Info:  
##   function:    stan_glm  
##   family:      binomial [logit]  
##   formula:     PA ~ RATIO  
##   algorithm:   sampling  
##   sample:      2400 (posterior sample size)  
##   priors:      see help('prior_summary')  
##   observations: 19  
##   predictors:  2  
##  
## Estimates:
```

```

##               mean   sd   10%   50%   90%
## (Intercept) 3.6   1.5   1.8   3.4   5.6
## RATIO       -0.2   0.1  -0.3  -0.2  -0.1
##
## Fit Diagnostics:
##               mean   sd   10%   50%   90%
## mean_PPD 0.5   0.1   0.4   0.5   0.7
##
## The mean_ppd is the sample average posterior predictive distribution of
## the outcome variable (for details see help('summary.stanreg')).
##
## MCMC diagnostics
##               mcse Rhat n_eff
## (Intercept) 0.0   1.0  2413
## RATIO       0.0   1.0  2519
## mean_PPD   0.0   1.0  2321
## log-posterior 0.0   1.0  2268
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude
## measure of effective sample size, and Rhat is the potential scale reduction
## factor on split chains (at convergence Rhat=1).

```

We can recheck the MCMC diagnostics: mean_PPD 0.0 1.0 2283 → These are the number of effective samples (we got 2400 samples in total).

Estimates: mean sd 10% 50% 90% (Intercept) 3.5 1.5 1.8 3.4 5.5 RATIO -0.2 0.1 -0.3 -0.2 -0.1

Now let's check the prob of lizard being present:

```

exp(3.5) # this is the estimated mean
## [1] 33.11545

```

→ When the perimeter to area ratio is = 0, we are 33.1x more likely to find the lizard being present than not present

```

plogis(3.6)
## [1] 0.973403

```

We got a 97% chance of the lizard being present if the area was = 0

Let's look at these numbers from the output: 10% 90% (Intercept) 1.8 5.5 RATIO -0.3 -0.1

```

exp(1.8)
## [1] 6.049647
exp(5.5)
## [1] 244.6919

```

```
plogis(1.8)
## [1] 0.8581489
plogis(5.5)
## [1] 0.9959299
```

= When the perim area ratio is 0, we have a 90% chance that we are 86-99% likely to have these lizards.

```
exp(-0.1)
## [1] 0.9048374
```

We got a 10% decline (1-0.9) when we go from perim area = 0 to perim area = ???

```
33.1*0.9
## [1] 29.79
```

Not sure what this meant, its the previous code chunk (0.9) and the first code chunk (33.1) of this section

MCMC sampling diagnostics

tidy MCMC

```
broom.mixed:::tidyMCMC(polis.rstanarm3$stanfit, estimate.method="median",
conf.int=TRUE, conf.method="HPDinterval", rhat=TRUE, ess=TRUE)

## # A tibble: 4 x 7
##   term      estimate std.error conf.low conf.high rhat    ess
##   <chr>        <dbl>     <dbl>     <dbl>     <dbl> <dbl> <int>
## 1 (Intercept)  3.43     1.49     0.852     6.38  1.00  2413
## 2 RATIO       -0.199    0.0826   -0.370    -0.0568 0.999  2519
## 3 mean_PPD     0.526    0.113     0.316     0.737  1.00  2321
## 4 log-posterior -10.9    1.04    -13.3    -10.2   1.00  2268
```

R sq

```
polis.rstanarm3 %>% bayes_R2() %>% median_hdci

##           y      ymin      ymax .width .point .interval
## 1 0.5021736 0.198257 0.7202087   0.95 median      hdci
```

LD 50

```
polis.rstanarm3 %>% tidy_draws() %>%
  mutate(LD50=-1*(Intercept)`/RATIO) %>%
  pull(LD50) %>% # extracts a single vector from a dataframe
  median_hdci # works on vectors, so we must have a vector to give it to the
function
```

```
##          y      ymin      ymax .width .point .interval
## 1 17.18689 10.44855 25.02777   0.95 median      hdci
```

Predicted data

```
polis.grid=with(polis, list(RATIO=seq(min(RATIO), max(RATIO), len=100)))

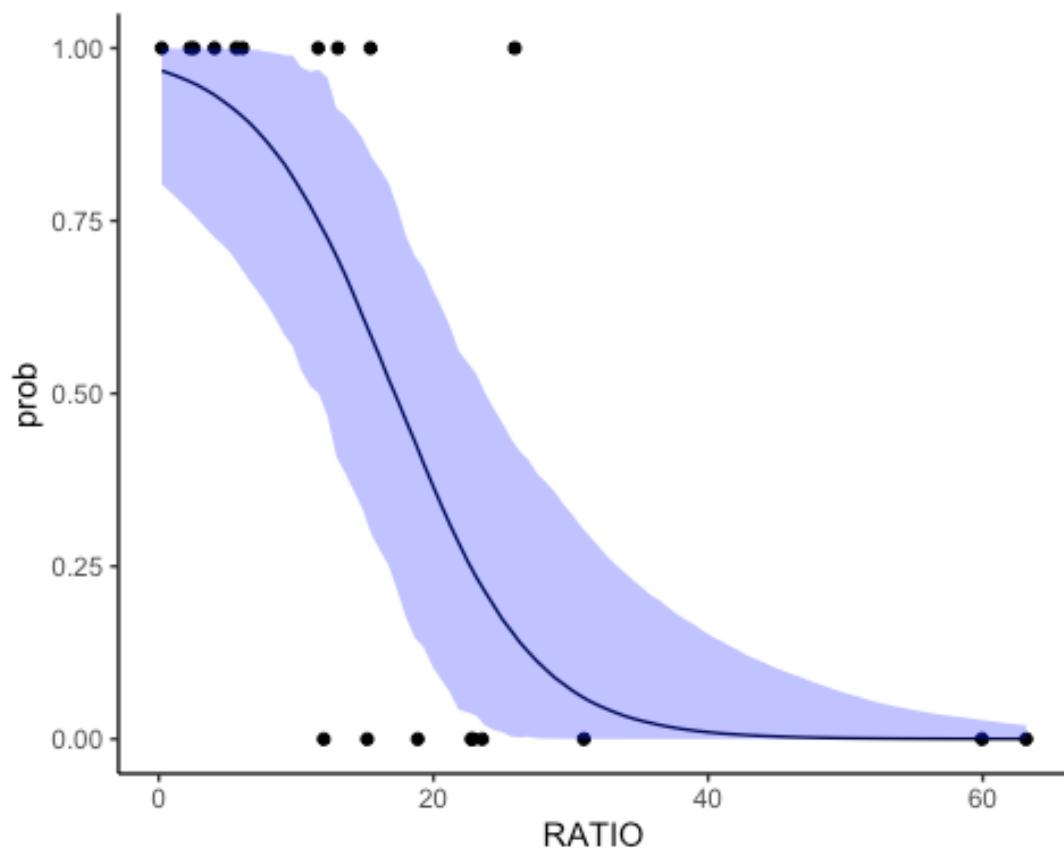
newdata=emmeans(polis.rstanarm3, ~RATIO, at=polis.grid, type="response") %>%
  as.data.frame

head(newdata)

##          RATIO      prob lower.HPD upper.HPD
## 1 0.2100000 0.9673298 0.8028625 0.9999063
## 2 0.8458586 0.9629458 0.7914466 0.9998647
## 3 1.4817172 0.9581276 0.7794115 0.9998159
## 4 2.1175758 0.9528329 0.7668872 0.9997546
## 5 2.7534343 0.9470783 0.7541782 0.9996729
## 6 3.3892929 0.9401467 0.7401484 0.9994102
```

Plot with predicted data

```
ggplot(newdata, aes(y=prob, x=RATIO))+
  geom_point(data=polis, aes(y=PA))+
  geom_line()+
  geom_ribbon(aes(ymin=lower.HPD, ymax=upper.HPD), fill="blue", alpha=0.3)+
  theme_classic()
```



This plot shows the probability of having lizards present, depending on the perimeter area ratio of the island.

IGNORE THE FOLLOWING (MURRAY'S TITLES, DIDNT FOLLOW THEM)

- Partial effects plots {.tabset .tabset-faded}
- Model investigation {.tabset .tabset-faded}
- Further analyses {.tabset .tabset-faded}
- Summary figure {.tabset .tabset-faded}
- References

16_Bayesian_Gaussian_Log

Sara Kophamel

Open: bglm_example4 15/12/2020

Preparations

Load the necessary libraries

```
library(car)

## Loading required package: carData

library(rstanarm)  #for fitting models in STAN

## Loading required package: Rcpp

## Registered S3 methods overwritten by 'lme4':
##   method           from
##   cooks.distance.influence.merMod car
##   influence.merMod      car
##   dfbeta.influence.merMod    car
##   dfbetas.influence.merMod   car

## This is rstanarm version 2.21.1

## - See https://mc-stan.org/rstanarm/articles/priors for changes to default
## priors!

## - Default priors may change, so it's safest to specify priors, even if
## equivalent to the defaults.

## - For execution on a local, multicore CPU with excess RAM we recommend
## calling

##   options(mc.cores = parallel::detectCores())

## 
## Attaching package: 'rstanarm'

## The following object is masked from 'package:car':
## 
##   logit

library(brms)      #for fitting models in STAN
```

```

## Loading 'brms' package (version 2.14.4). Useful instructions
## can be found by typing help('brms'). A more detailed introduction
## to the package is available through vignette('brms_overview').

##
## Attaching package: 'brms'

## The following objects are masked from 'package:rstanarm':
##
##     dirichlet, exponential, get_y, lasso, ngrps

## The following object is masked from 'package:stats':
##
##     ar

library(coda)      #for diagnostics
library(bayesplot) #for diagnostics

## This is bayesplot version 1.7.2

## - Online documentation and vignettes at mc-stan.org/bayesplot
## - bayesplot theme set to bayesplot::theme_default()
##   * Does _not_ affect other ggplot2 plots
##   * See ?bayesplot_theme_set for details on theme setting

library(ggmcmc)    #for MCMC diagnostics

## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:car':
##
##     recode

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: tidyverse

## Loading required package: ggplot2

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

```

```

library(rstan)      #for interfacing with STAN

## Loading required package: StanHeaders

## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend
## calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)

##
## Attaching package: 'rstan'

## The following object is masked from 'package:tidyverse':
## extract

## The following object is masked from 'package:coda':
## traceplot

library(emmeans)    #for marginal means etc
library(DHARMa)     #for residual diagnostics

## This is DHARMa 0.3.3.0. For overview type '?DHARMa'. For recent changes,
## type news(package = 'DHARMa') Note: Syntax of plotResiduals has changed in
## 0.3.0, see ?plotResiduals for details

library(broom)       #for tidying outputs
library(tidybayes)  #for more tidying outputs

##
## Attaching package: 'tidybayes'

## The following objects are masked from 'package:brms':
## dstudent_t, pstudent_t, qstudent_t, rstudent_t

library(ggeffects)  #for partial plots
library(broom.mixed)#for summarising models

## Registered S3 method overwritten by 'broom.mixed':
##   method      from
##   tidy.gamlss broom

library(ggeffects)  #for partial effects plots
library(tidyverse)  #for data wrangling etc

## — Attaching packages

```

tidyverse 1.3.0 —

```

## ✓ tibble  3.0.3      ✓ stringr 1.4.0
## ✓ readr   1.3.1      ✓forcats 0.5.0
## ✓ purrr   0.3.4

## — Conflicts
----- tidyverse_conflicts() -----
## x rstan::extract() masks tidyrr::extract()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x dplyr::recode()  masks car::recode()
## x purrr::some()    masks car::some()

```

Scenario

@Loyn-1987-1987 modeled the abundance of forest birds with six predictor variables (patch area, distance to nearest patch, distance to nearest larger patch, grazing intensity, altitude and years since the patch had been isolated).



Regent honeyeater

Format of loyn.csv data file

ABUND	DIST	LDIST	AREA	GRAZE	ALT	YR.ISOL
..
ABUND	Abundance of forest birds in patch-response variable					
DIST	Distance to nearest patch - predictor variable					
LDIST	Distance to nearest larger patch - predictor variable					
AREA	Size of the patch - predictor variable					

GRAZE Grazing intensity (1 to 5, representing light to heavy) - predictor variable

ALT Altitude - predictor variable

YR.ISOL Number of years since the patch was isolated - predictor variable

The aim of the analysis is to investigate the effects of a range of predictors on the abundance of forest birds.

THE ABUNDANCE OF BIRDS SHOULD HAVE BEEN A POISSON DISTRIBUTION ("HOW MANY BIRDS ARE THERE?"), BUT INSTEAD THE AUTHORS PROVIDED AVG COUNTS PER AREAS (NOT AN INTEGER). WE ARE THEREFORE USING A LOG-NORMAL LIKE MODEL (a Gaussian, but with a log link to prevent values <0>)

Read in the data

```
loyn = read_csv('data/loyn.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   ABUND = col_double(),
##   AREA = col_double(),
##   YR.ISOL = col_double(),
##   DIST = col_double(),
##   LDIST = col_double(),
##   GRAZE = col_double(),
##   ALT = col_double()
## )
```

Exploratory data analysis

Model formula:

$$\text{y}_i \sim \mathcal{N}(\mu_i, \sigma^2) \quad \log(\mu_i) = \boldsymbol{\beta} \cdot \mathbf{X}_i$$

where β is a vector of effects parameters and \mathbf{X} is a model matrix representing the additive effects of the scaled versions of distance (ln), distance to the nearest large patch (ln), patch area (ln), grazing intensity, year of isolation and altitude on the abundance of forest birds.

```
glimpse(loyn)

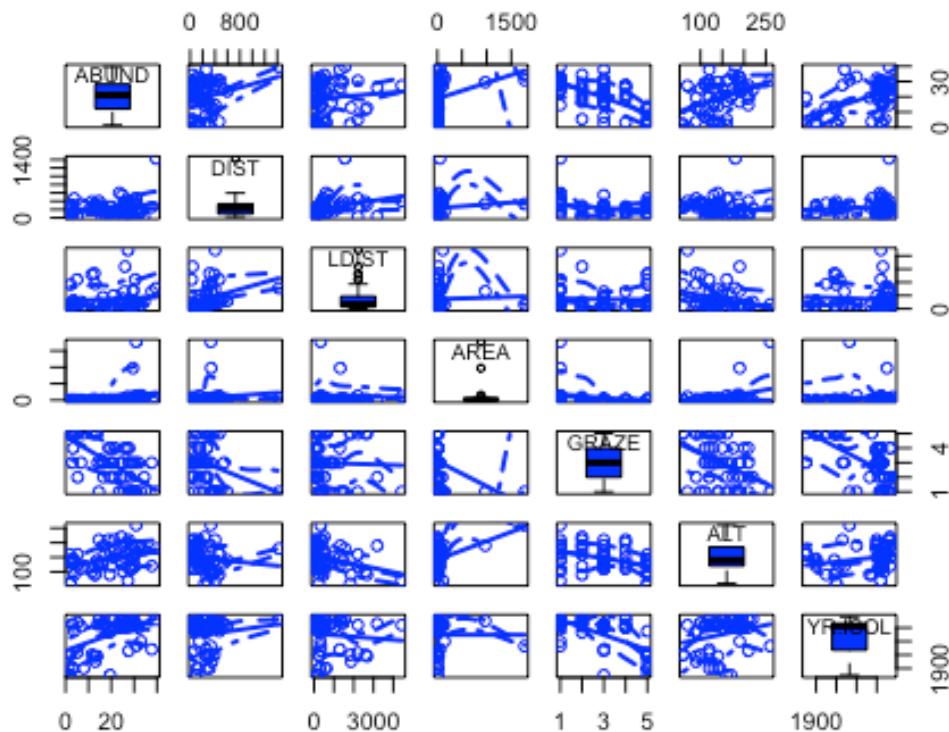
## Rows: 56
## Columns: 7
## $ ABUND    <dbl> 5.3, 2.0, 1.5, 17.1, 13.8, 14.1, 3.8, 2.2, 3.3, 3.0, 27.6,
## ...
## $ AREA     <dbl> 0.1, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0,
```

```
2.0, 2...
## $ YR.ISOL <dbl> 1968, 1920, 1900, 1966, 1918, 1965, 1955, 1920, 1965,
1900, 1...
## $ DIST     <dbl> 39, 234, 104, 66, 246, 234, 467, 284, 156, 311, 66, 93,
39, 4...
## $ LDIST    <dbl> 39, 234, 311, 66, 246, 285, 467, 1829, 156, 571, 332, 93,
39, ...
## $ GRAZE   <dbl> 2, 5, 5, 3, 5, 3, 5, 5, 4, 5, 3, 5, 2, 1, 5, 5, 3, 3, 3,
2, 2...
## $ ALT      <dbl> 160, 60, 140, 160, 140, 130, 90, 60, 130, 130, 210, 160,
210, ...
```

Transforming characters into factors

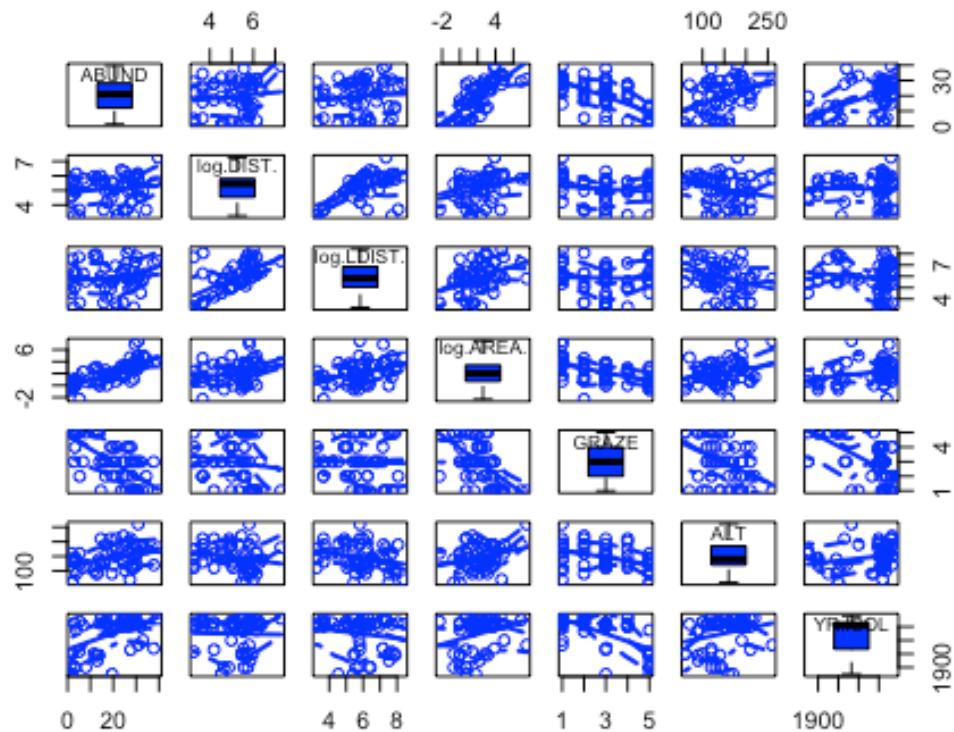
```
loyn=loyn %>% mutate(fGRAZE=factor(GRAZE))

scatterplotMatrix(~ABUND+DIST+LDIST+AREA+GRAZE+ALT+YR.ISOL,data=loyn,
diagonal=list(method="boxplot"))
```



Distance, LDist and Area need to be transformed

```
scatterplotMatrix(~ABUND+log(DIST)+log(LDIST)+log(AREA)+GRAZE+ALT+YR.ISOL,data=loyn,
diagonal=list(method="boxplot"))
```



This is better.

Murray says we should centre our predictors to be able to compare this example to the frequentist analysis we did. In theory we should not have to center our data (as Bayesian does it automatically). We are here only doing it for comparison purposes

```
loyn.rstanarm=stan_glm(ABUND~scale(log(DIST), scale=FALSE)+  
scale(log(DIST), scale=FALSE)+  
scale(log(AREA), scale=FALSE)+  
fGRAZE+  
scale(ALT, scale=FALSE)+  
scale(YR.ISOL, scale=FALSE),  
data=loyn,  
family=gaussian(link="log"),  
iter=5000,  
warmup=1000,  
chains=3,  
thin=5,  
refresh=0)  
  
## Warning: There were 711 divergent transitions after warmup. See  
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup  
## to find out why this is a problem and how to eliminate them.
```

```

## Warning: There were 766 transitions after warmup that exceeded the maximum
treedepth. Increase max_treedepth above 15. See
## http://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded

## Warning: There were 3 chains where the estimated Bayesian Fraction of
Missing Information was low. See
## http://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: The largest R-hat is 4.57, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating
posterior means and medians may be unreliable.
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating
posterior variances and tail quantiles may be unreliable.
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess

## Warning: Markov chains did not converge! Do not analyze results!

```

Do NOT ignore the warning messages. What do they mean? - We got 956 divergent transitions, but we only had 2400 samples all up (after warmup). What is a divergent transition? When the sampler hits a sharp feature and falls off / when it samples in an area where there is nothing, we got a divergent transition. If we get 5-6, even 10 divergent transitions, don't worry about it. But such a large number IS a concern. - The chains are not well mixed, and might be in trouble - The sample size is too low

Perhaps... - Our model is insane to fit (not the case here though) - Changing the adaptive delta: We have to tell R how many samples to learn from. It is set by default to 0.8, which is very low. We could tell R to use more of its warmup samples to work out how to better find the prob distribution. The model will just run slower, but it might be more effective. - Doing more warmups: We can increase the amount of warmups - Reduce the priors: They are usually too wide, so narrowing them down might help -> Let's try this, but first we have to know what they are - Reduce the thinning won't help (!)

Checking the priors

```
prior_summary(loyn.rstanarm)
```

```

## Priors for model 'loyn.rstanarm'
## -----
## Intercept (after predictors centered)
##   Specified prior:
##     ~ normal(location = 0, scale = 2.5)
##   Adjusted prior:
##     ~ normal(location = 0, scale = 27)

```

```
##  
## Coefficients  
##   Specified prior:  
##     ~ normal(location = [0,0,0,...], scale = [2.5,2.5,2.5,...])  
##   Adjusted prior:  
##     ~ normal(location = [0,0,0,...], scale = [28.17,14.35,76.00,...])  
##  
## Auxiliary (sigma)  
##   Specified prior:  
##     ~ exponential(rate = 1)  
##   Adjusted prior:  
##     ~ exponential(rate = 0.093)  
## -----  
## See help('prior_summary.stanreg') for more details
```

Our slopes (betas) seem to have different scales: Coefficients Specified prior: $\sim \text{normal}(\text{location} = [0,0,0,\dots], \text{scale} = [2.5,2.5,2.5,\dots])$ Adjusted prior: $\sim \text{normal}(\text{location} = [0,0,0,\dots], \text{scale} = [28.17,14.35,76.00,\dots])$ —> Each predictor is on their own scale. We might give them all the same prior. This vectorises them, and makes them run faster.

Intercept (after predictors centered) Specified prior: $\sim \text{normal}(\text{location} = 0, \text{scale} = 2.5)$
 Adjusted prior: $\sim \text{normal}(\text{location} = 0, \text{scale} = 27)$ —> We have a VERY WIDE adjusted prior (27).

—> A log normal distribution is not very common, so by deciding what adjustments and priors to use, R picked those priors without accounting for the log link (that is Murray's guess). Those numbers are WAY TOO BIG for a log scale.

So let us create our own priors instead:

Not sure why Murray checked the following:

```
exp(3)  
## [1] 20.08554
```

20 is the avg number of birds across the exp

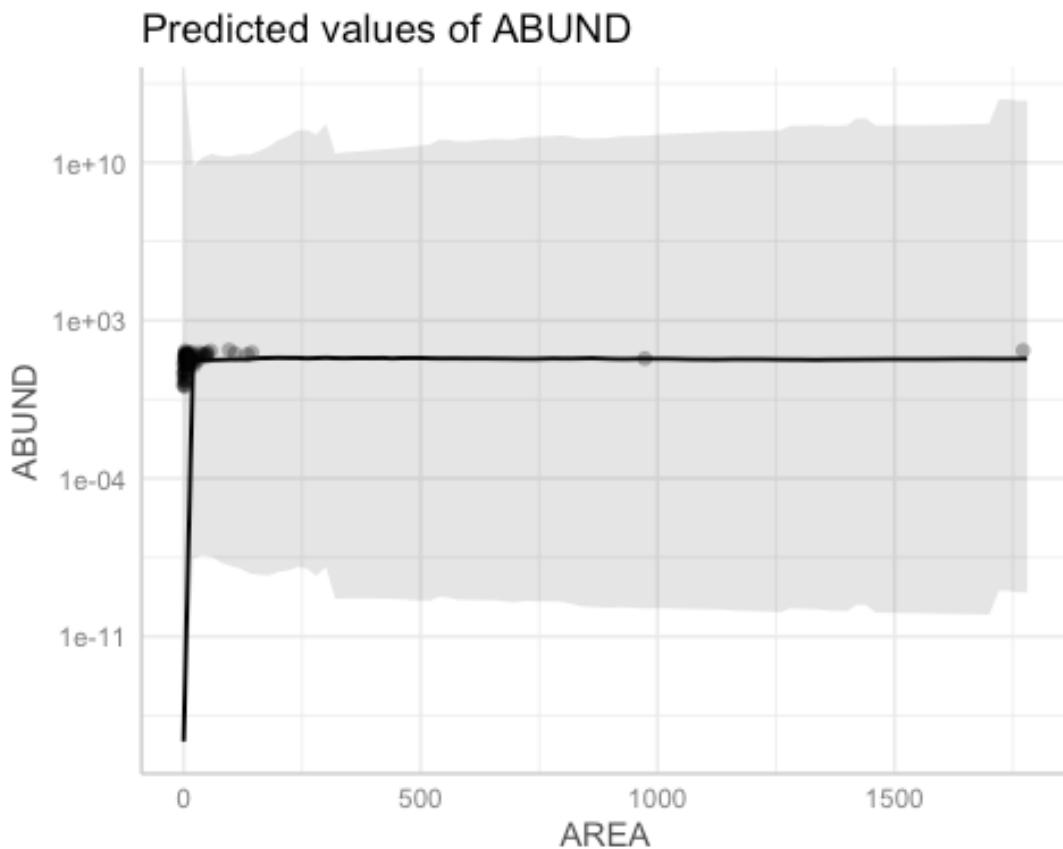
We will go for a Caushy centered distribution:

```
loyn.rstanarm2<-stan_glm(ABUND~scale(log(DIST))+ # In this case, do not set  
scale=FALSE, as we did not define the priors for scale=FALSE  
  scale(log(LDIST))+  
  scale(log(AREA))+  
  fGRAZE+  
  scale(ALT)+  
  scale(YR.ISOL), data=loyn,  
  family=gaussian(link="log"),  
  prior_intercept=normal(3,10,autoscale=FALSE),  
  prior=normal(0,2.5,autoscale=FALSE),  
  prior_aux=cauchy(0,5),
```

```
prior_PD=TRUE,  
iter=5000,thin=5, chains=3, warmup=2000,  
refresh=0)
```

And we plot this

```
ggemmeans(loyn.rstanarm2,~AREA) %>% # Normally, you'd go through all  
# variables (not only AREA), as the primer could affect some differently.  
# Murray chose area bec AREA (and GRAZE) was the important one  
plot(add.data=TRUE)+  
scale_y_log10()  
  
## Scale for 'y' is already present. Adding another scale for 'y', which will  
## replace the existing scale.
```



It does

not look like we will modify the outcome, so that is good.

Fit the final model

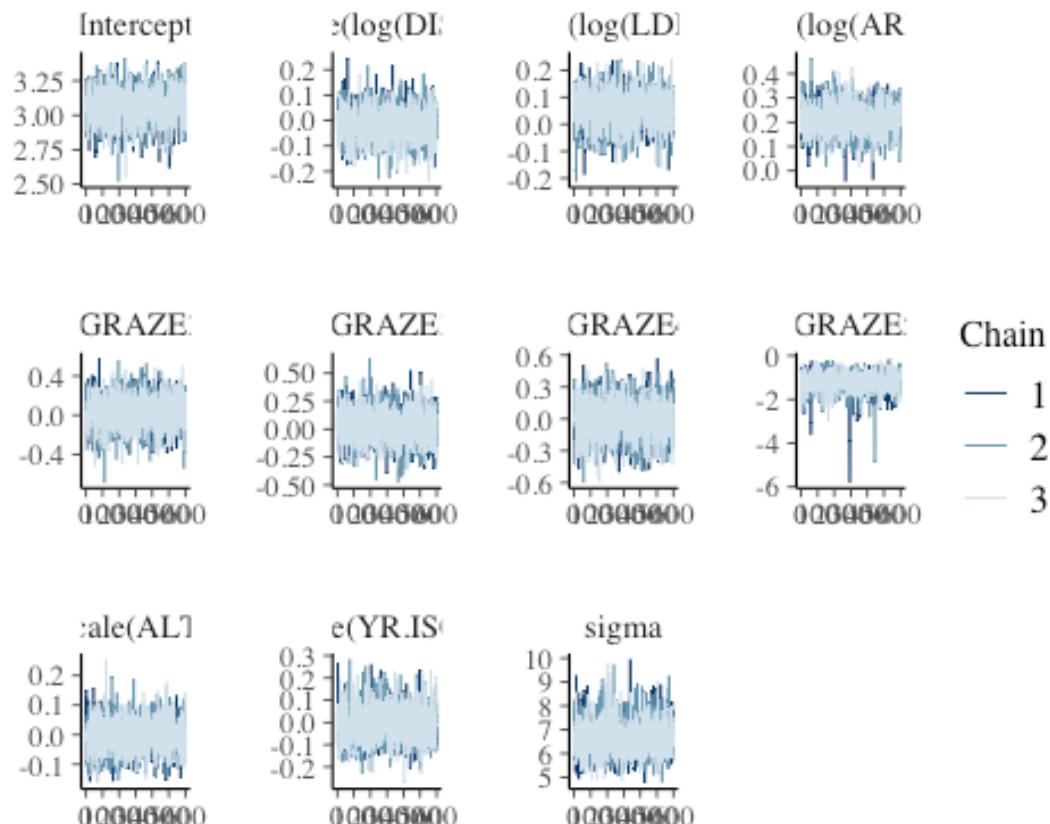
We have now to allow the data defining the priors, and not the other way round:

```
loyn.rstanarm3=update(loyn.rstanarm2,prior_PD=FALSE)
```

This is our final model.

MCMC sampling diagnostics

```
plot(lyn.rstanarm3, plotfun="mcmc_trace")
```

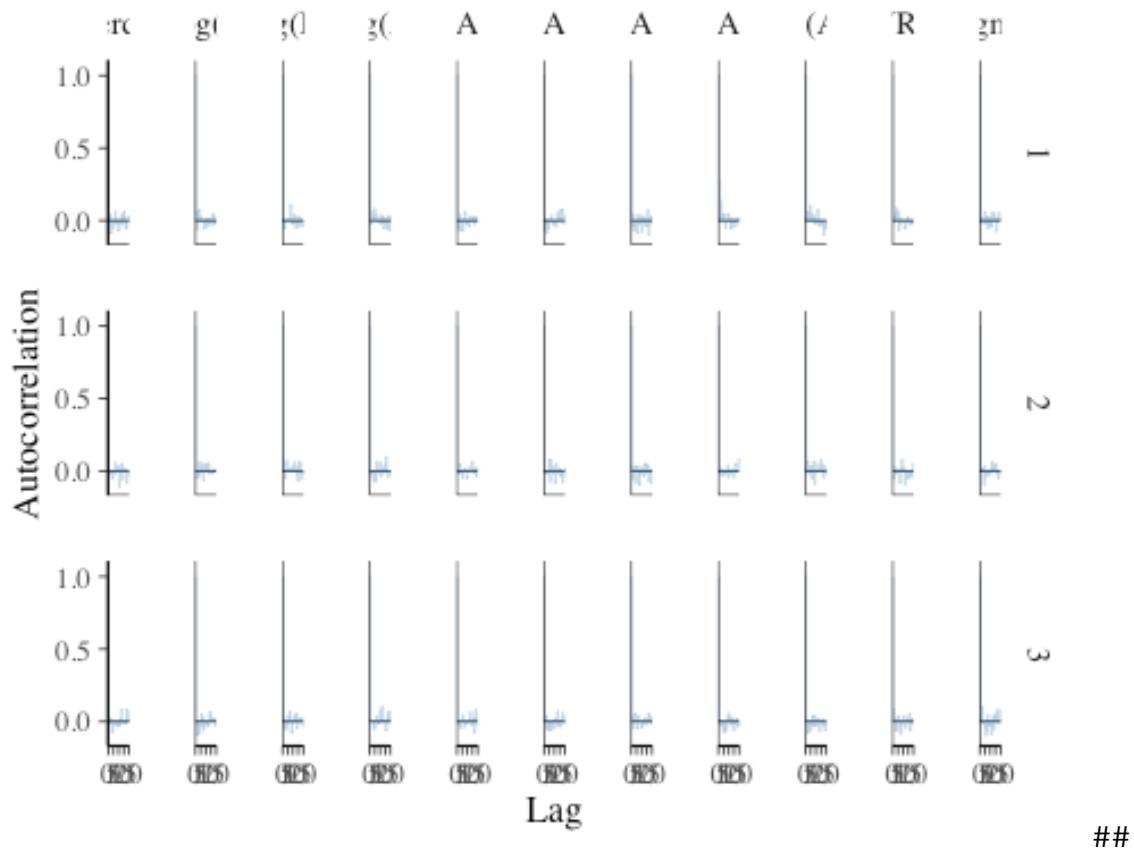


looks great.

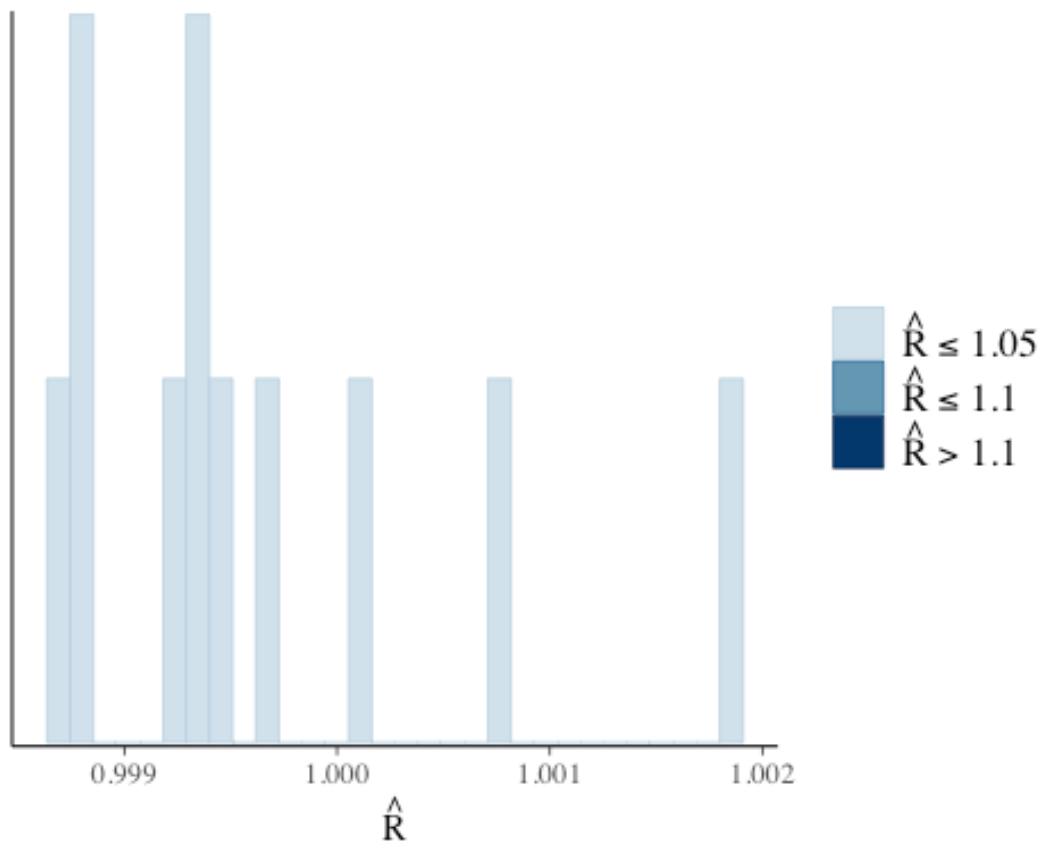
This

Autocorrelation

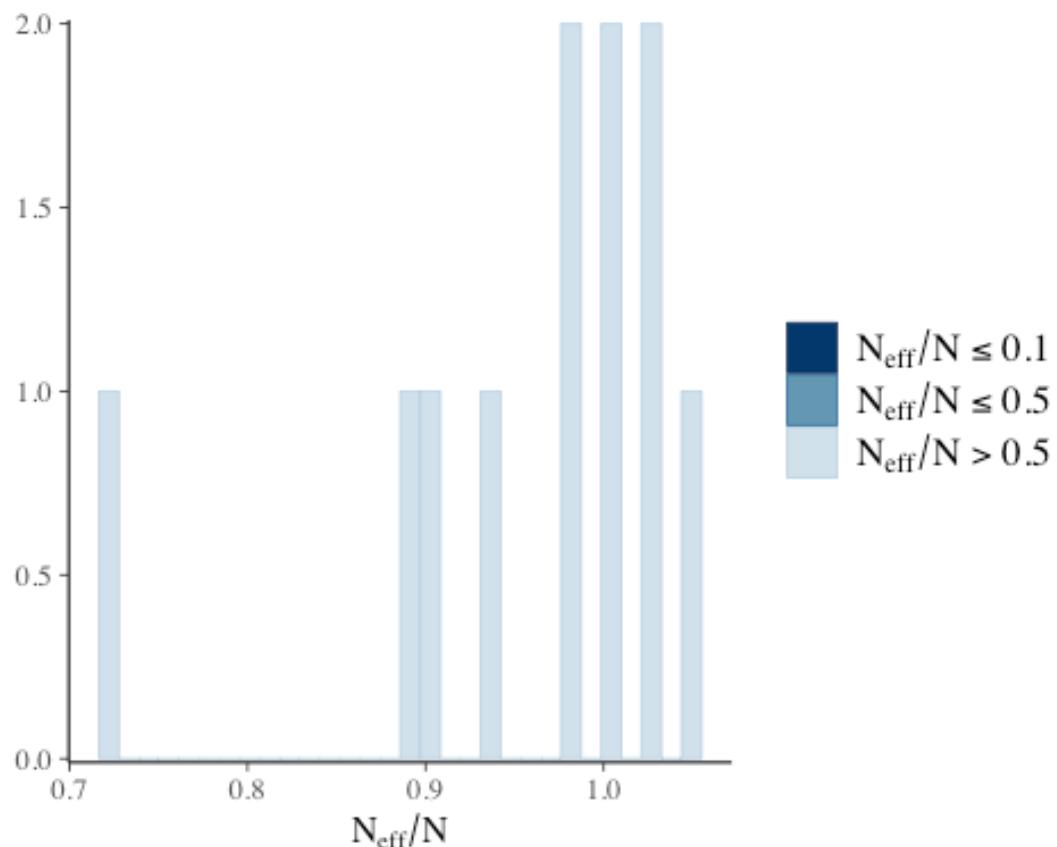
```
plot(lyn.rstanarm3, "acf_bar")
```



```
plot(loyn.rstanarm3, "rhat_hist")
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



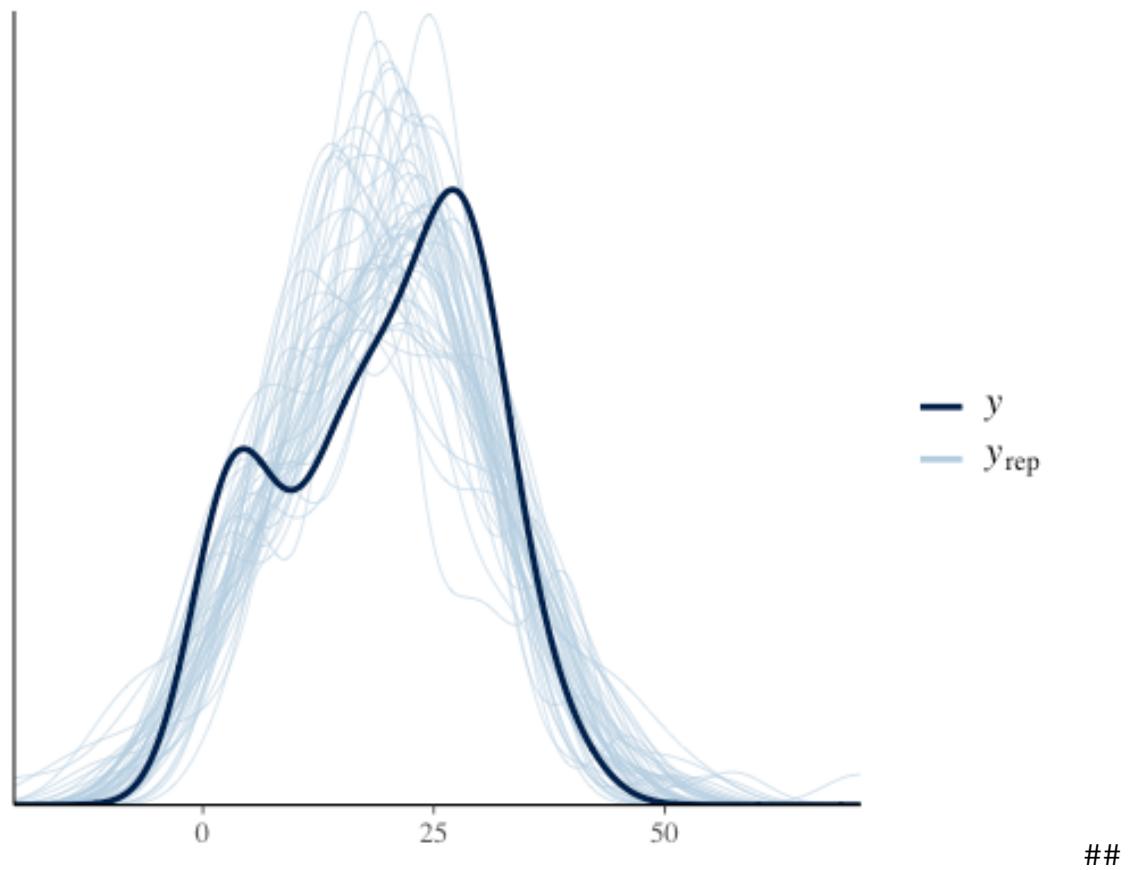
```
##  
plot(loyn.rstanarm3,"neff_hist")  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

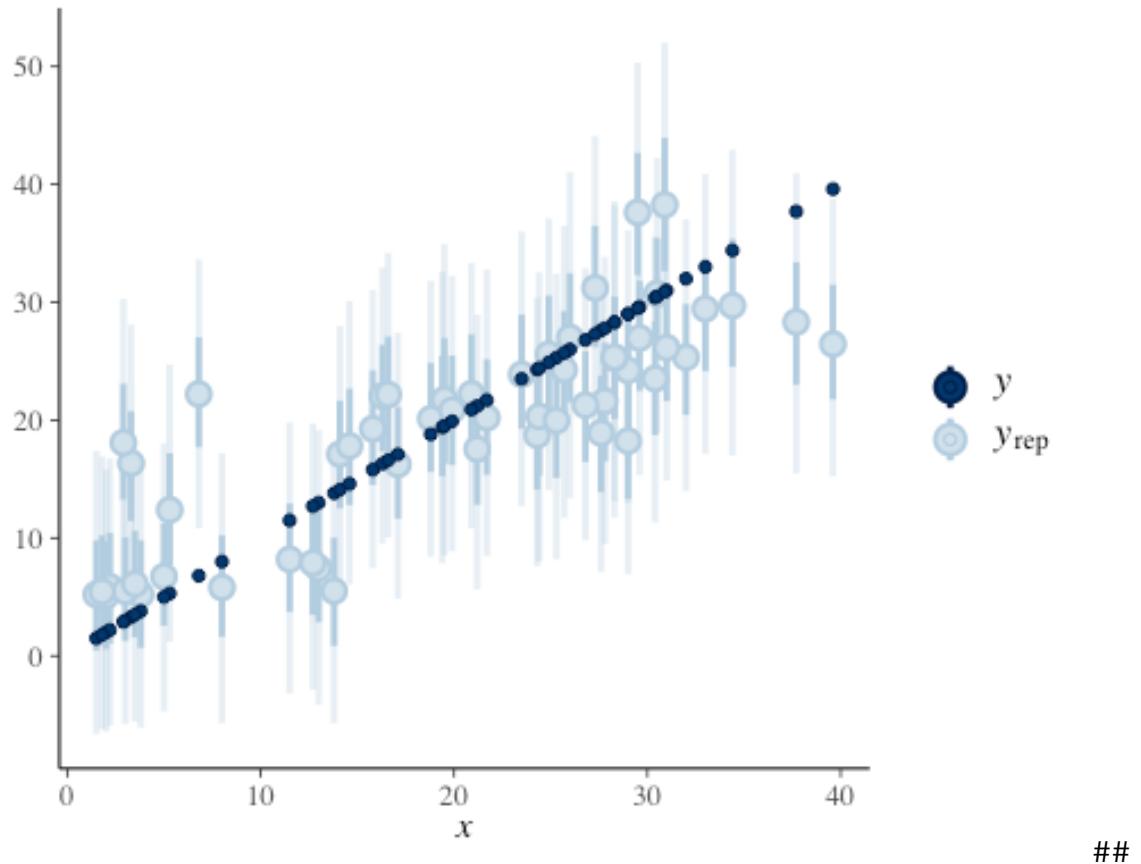


Posterior checks - Is the model predicting the data well?

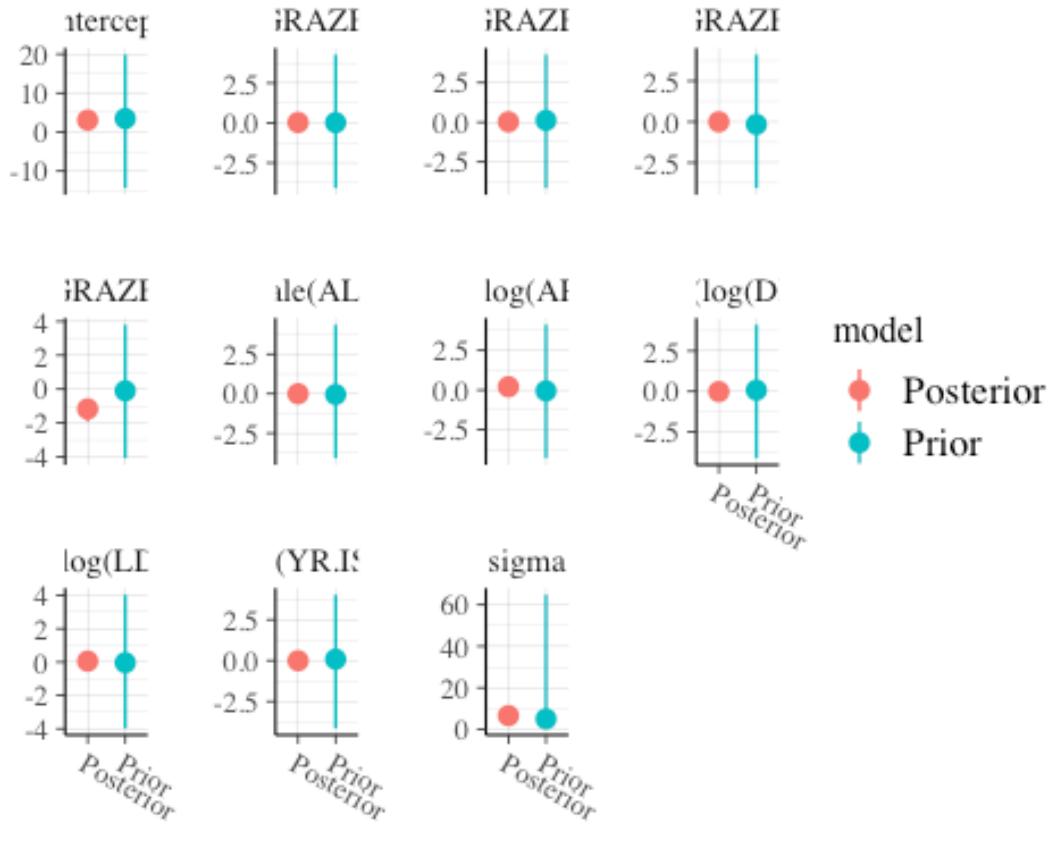
1. Observed data and the scope of our predictions

```
pp_check(loyn.rstanarm3, plotfun="dens_overlay")
```





```
posterior_vs_prior(loyn.rstanarm3,color_by="vs",group_by=TRUE,  
                   facet_args=list(scales="free_y"))  
  
##  
## Drawing from prior...
```



DHARMA residuals

```

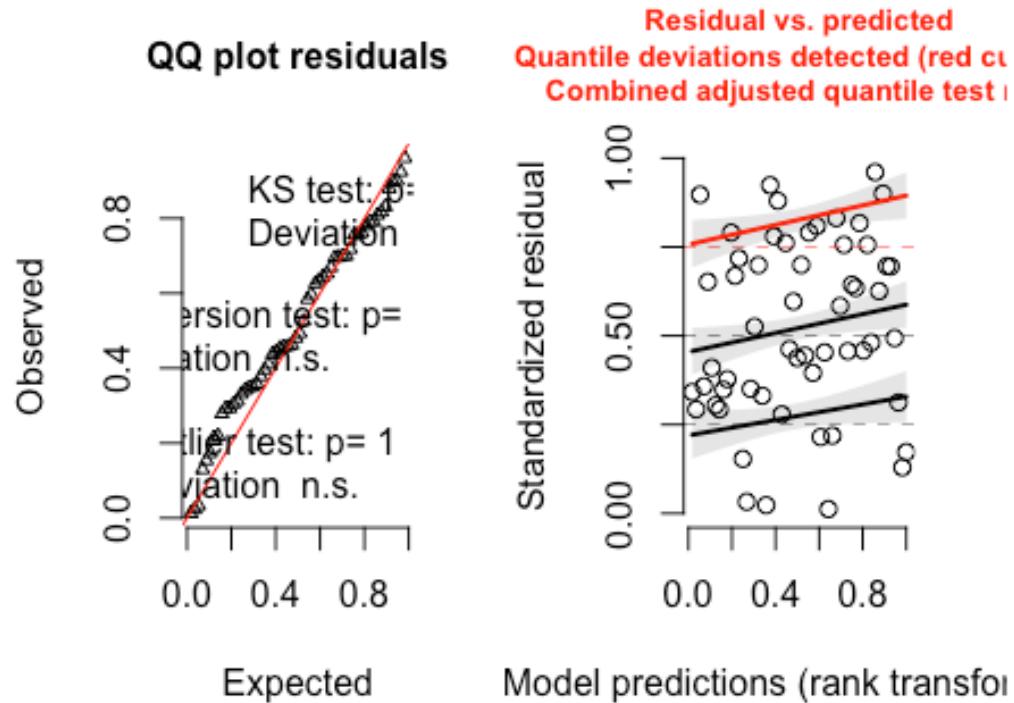
preds<- posterior_predict(loyn.rstanarm3, nsamples=250, summary=FALSE)

loyn.resids<-createDHARMA(simulatedResponse = t(preds),
                           observedResponse = loyn$ABUND,# This has to be the RESPONSE (abundance), and can't change. DHARMA residuals is for the response, not for other variables.
                           fittedPredictedResponse = apply(preds, 2, median),
                           integerResponse = FALSE) # we have to set integerResponse to FALSE whenever there are no counts. If there were counts, R would treat our data as if it was a Poisson distribution, and would sample accordingly.

plot(loyn.resids)

```

DHARMA residual diagnostics

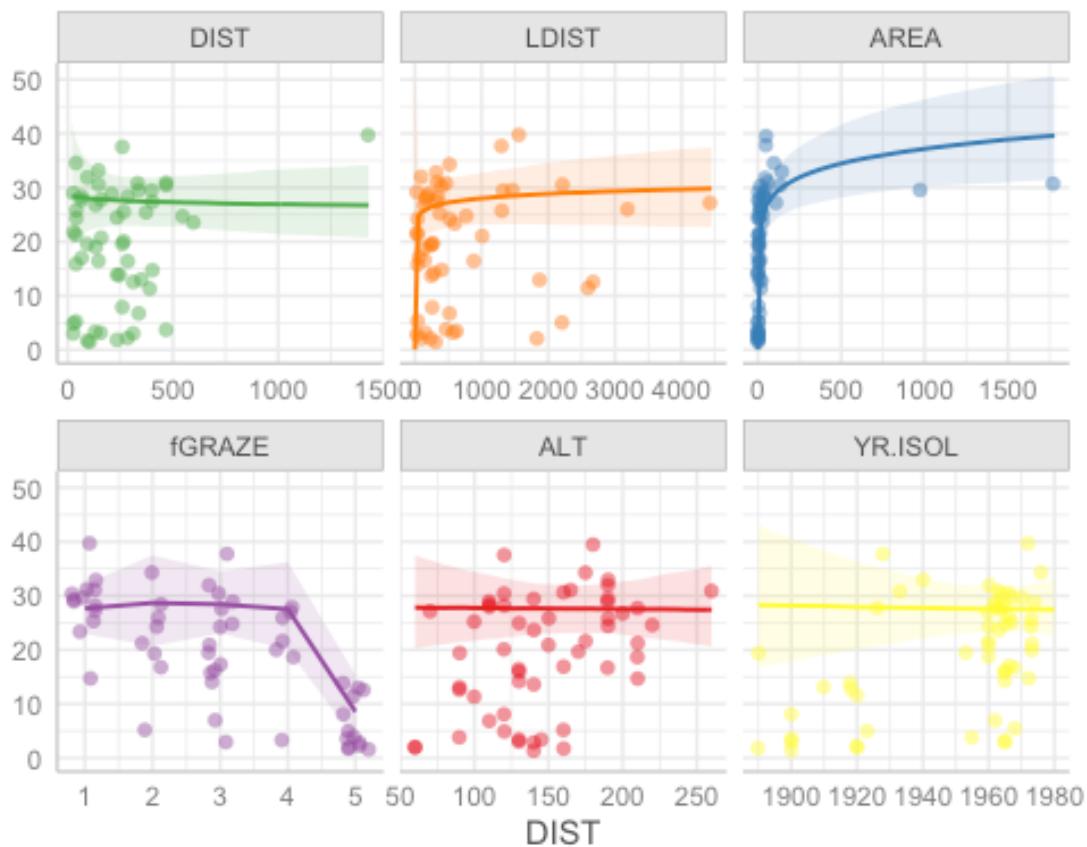


Partial effects plots

They help us interpret our results

Plot for all variables

```
loyn.rstanarm3 %>%  
  ggpredict() %>%  
  plot(add.data=TRUE, facet=TRUE)
```



*Never

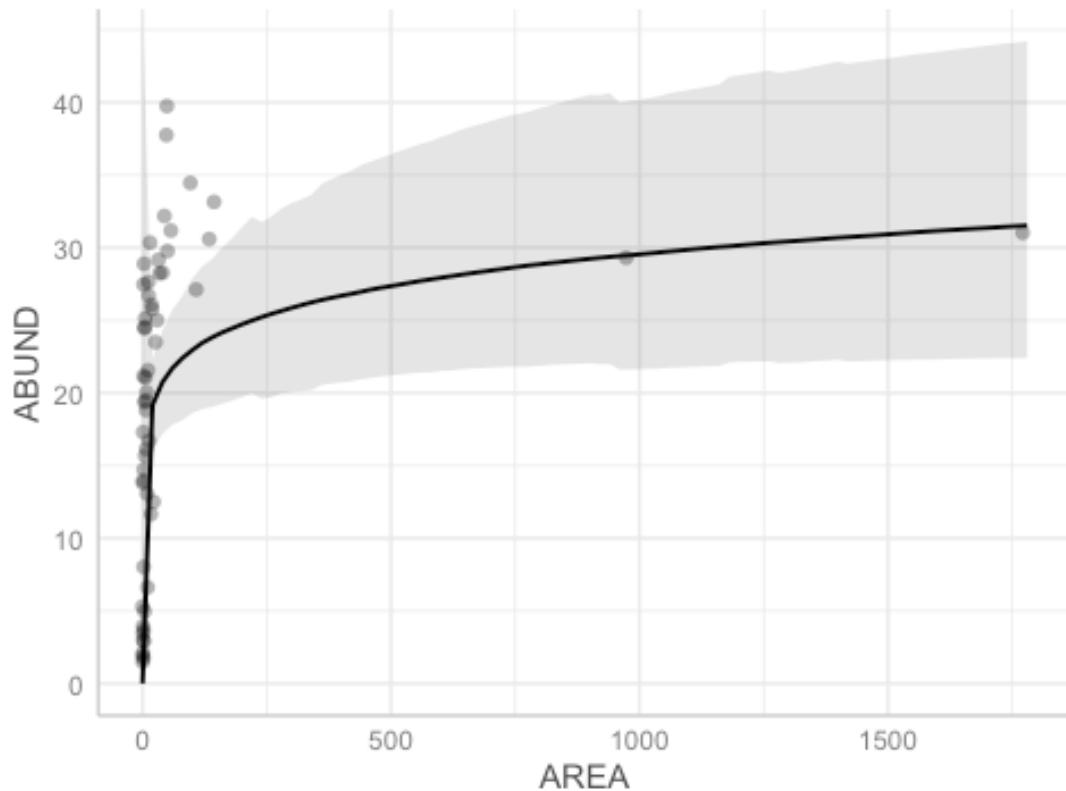
add partial residuals to a plot. - GRAZE seems to have a negative effect on the abundance of birds. - AREA also has a dramatic increase effect. All the changes in birds occur at areas <100. In summary, there is potentially an effect of AREA and GRAZE

(!) WE ARE USING A LOG SCALE, SO WE HAVE TO BE CAREFUL WHEN EXTRAPOLATING
(!)

Plot for a specific variable (not needed here)

```
loyn.rstanarm3 %>%
  ggmeans(~AREA) %>%
  plot(add.data=TRUE)
```

Predicted values of ABUND



Summary

```
summary(loyn.rstanarm3)

##
## Model Info:
##   function:     stan_glm
##   family:      gaussian [log]
##   formula:     ABUND ~ scale(log(DIST)) + scale(log(LDIST)) +
##                 scale(log(AREA)) +
##                 fGRAZE + scale(ALT) + scale(YR.ISOL)
##   algorithm:   sampling
##   sample:      1800 (posterior sample size)
##   priors:      see help('prior_summary')
##   observations: 56
##   predictors:  10
##
## Estimates:
##               mean    sd   10%   50%   90%
## (Intercept) 3.1 0.1 2.9 3.1 3.2
## scale(log(DIST)) 0.0 0.1 -0.1 0.0 0.1
## scale(log(LDIST)) 0.1 0.1 0.0 0.1 0.1
## scale(log(AREA)) 0.2 0.1 0.1 0.2 0.3
```

```

## fGRAZE2          0.0   0.2 -0.2   0.0   0.2
## fGRAZE3          0.0   0.1 -0.2   0.0   0.2
## fGRAZE4          0.0   0.2 -0.2   0.0   0.2
## fGRAZE5         -1.2   0.4 -1.7  -1.2  -0.7
## scale(ALT)        0.0   0.1 -0.1   0.0   0.1
## scale(YR.ISOL)    0.0   0.1 -0.1   0.0   0.1
## sigma            6.6   0.7  5.8   6.6   7.6
##
## Fit Diagnostics:
##               mean   sd  10%  50%  90%
## mean_PPD 19.2   1.3 17.5 19.2 20.8
##
## The mean_ppd is the sample average posterior predictive distribution of
## the outcome variable (for details see help('summary.stanreg')).
##
## MCMC diagnostics
##               mcse Rhat n_eff
## (Intercept)  0.0  1.0 1856
## scale(log(DIST)) 0.0  1.0 1804
## scale(log(LDIST)) 0.0  1.0 1682
## scale(log(AREA))  0.0  1.0 1805
## fGRAZE2       0.0  1.0 1847
## fGRAZE3       0.0  1.0 1897
## fGRAZE4       0.0  1.0 1768
## fGRAZE5       0.0  1.0 1308
## scale(ALT)     0.0  1.0 1772
## scale(YR.ISOL) 0.0  1.0 1604
## sigma          0.0  1.0 1621
## mean_PPD      0.0  1.0 1692
## log-posterior  0.1  1.0 1681
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude
## measure of effective sample size, and Rhat is the potential scale reduction
## factor on split chains (at convergence Rhat=1).

```

Here, we see the avg number of birds at the avg distance, ag altitude, avg yrs of isolation, etc. in grazing intensity 1. Given that all variables were centered, we are seeing the avg number of birds when all other variables are at 0. We had already checked how many birds there are on avg (around 22) in a previous code chunk.

Let's ahve a look at the Estimates: mean sd 10% 50% 90%
(Intercept) 3.1 0.1 2.9 3.1 3.2
scale(log(DIST)) 0.0 0.1 -0.1 0.0 0.1 scale(log(LDIST)) 0.1 0.1 0.0 0.1 0.1 scale(log(AREA)) 0.2 0.1 0.1 0.2 0.3
-> looks like AREA has an effect on abundance
fGRAZE2 0.0 0.2 -0.2 0.0 0.2 fGRAZE3 0.0 0.1 -0.2 0.0 0.2 fGRAZE4 0.0 0.2 -0.2 0.0 0.2 fGRAZE5 -1.2 0.4 -1.8 -1.2 -0.7
-> looks like fGRAZE5 has an effect on abundance
scale(ALT) 0.0 0.1 -0.1 0.0 0.1
scale(YR.ISOL) 0.0 0.1 -0.1 0.0 0.1 sigma 6.7 0.7 5.8 6.6 7.6

tidy summary

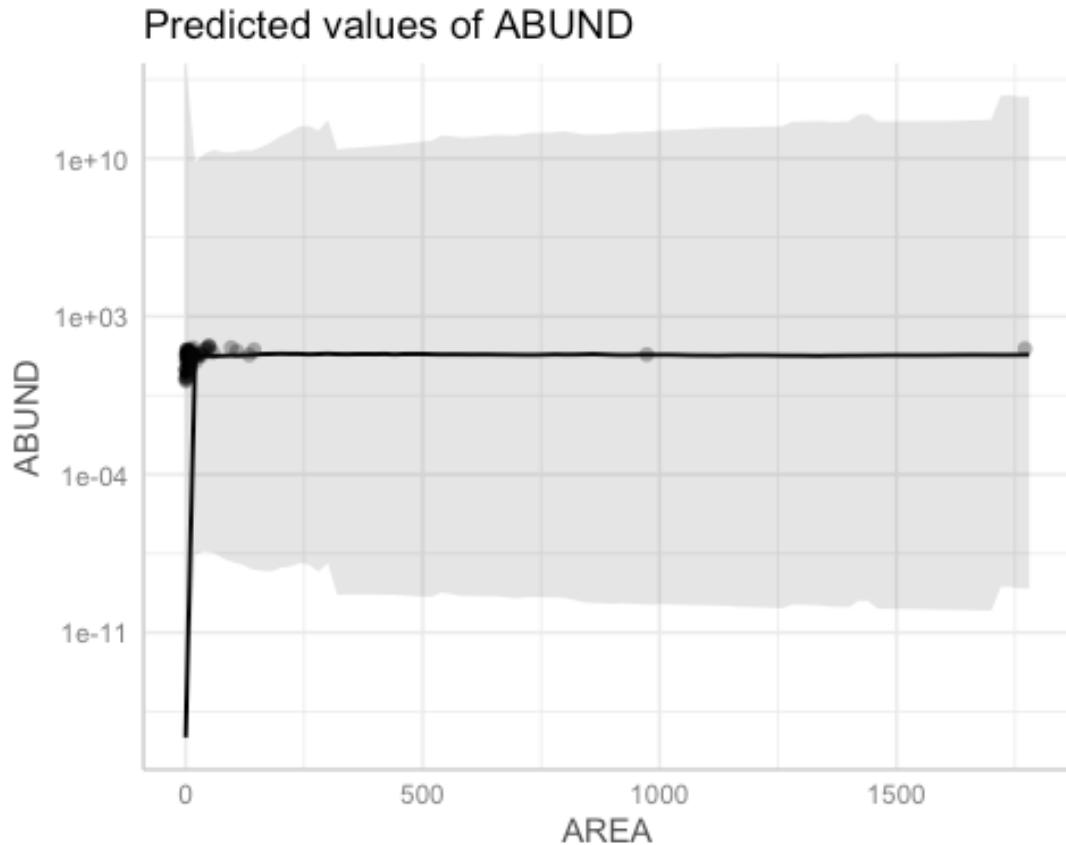
```
tidyMCMC(loyn.rstanarm3$stanfit, estimate.method="median", conf.int=TRUE,
          conf.method="HPDinterval", rhat=TRUE, ess=TRUE)

## # A tibble: 13 x 7
##   term      estimate std.error conf.low conf.high rhat    ess
##   <chr>     <dbl>     <dbl>     <dbl>     <dbl> <dbl> <int>
## 1 (Intercept) 3.06      0.118     2.82      3.28  0.999 1856
## 2 scale(log(DIST)) -0.0136    0.0630    -0.138     0.107 1.00  1804
## 3 scale(log(LDIST)) 0.0550    0.0640    -0.0783    0.171 1.00  1682
## 4 scale(log(AREA))  0.207     0.0612     0.0950    0.332 0.999 1805
## 5 fGRAZE2       0.0277    0.164     -0.283     0.367 0.999 1847
## 6 fGRAZE3       0.0266    0.143     -0.248     0.301 0.999 1897
## 7 fGRAZE4       -0.000951  0.173     -0.363     0.326 0.999 1768
## 8 fGRAZE5       -1.17      0.440     -2.04     -0.418 1.00  1308
## 9 scale(ALT)     -0.00358  0.0540    -0.105     0.104 0.999 1772
## 10 scale(YR.ISOL) -0.00896  0.0796    -0.175     0.136 0.999 1604
## 11 sigma         6.57      0.733      5.38      8.17  1.00  1621
## 12 mean_PPD      19.2      1.30      16.7      21.9  1.00  1692
## 13 log-posterior -198.      2.76     -203.     -193.  1.00  1681
```

We could now look and see the conf int to make conclusions which ones overlap with 0 or not. But that's not the point. The Bayesian way would be to calculate a prob for each of these. What is the prob that the value is 0? What is the PROBABILITY that birds have declined in fGRAZE5? If the intercept does not overlap with conf int that encompass 0, the prob must at least be 95%. But if the intercept overlaps with conf int that encompass 0, we have another prob, which we can only determine with Bayesian statistics.

```
ggemmeans(loyn.rstanarm2, ~AREA) %>% # Normally, you'd go through all
# variables
  plot(add.data=TRUE) +
  scale_y_log10()

## Scale for 'y' is already present. Adding another scale for 'y', which will
## replace the existing scale.
```



It does not look like we will modify the outcome, so that is good.

Fit the final model

We have now to allow the data defining the priors, and not the other way round:

```
loyn.rstanarm3=update(loyn.rstanarm2,prior_PD=FALSE)

tidyMCMC(loyn.rstanarm3$stanfit,estimate.method="median",conf.int=TRUE,
conf.method="HPDinterval", rhat=TRUE,ess=TRUE)

## # A tibble: 13 x 7
##   term          estimate std.error conf.low conf.high rhat    ess
##   <chr>        <dbl>     <dbl>    <dbl>    <dbl> <dbl> <int>
## 1 (Intercept)  3.07      0.119     2.82     3.29  1.00  1803
## 2 scale(log(DIST)) -0.0166   0.0605   -0.132    0.107 0.999 1909
## 3 scale(log(LDIST))  0.0546   0.0642   -0.0826   0.175 0.999 1822
## 4 scale(log(AREA))   0.210    0.0626    0.0769   0.325 0.999 1918
## 5 fGRAZE2       0.0276    0.165    -0.307    0.350 1.00  1883
## 6 fGRAZE3       0.0223    0.147    -0.286    0.283 1.00  1802
## 7 fGRAZE4      -0.00597   0.169    -0.339    0.314 1.00  1569
## 8 fGRAZE5       -1.17     0.425    -2.06    -0.427 1.00  1720
## 9 scale(ALT)    -0.000999  0.0524   -0.0975   0.106 0.999 1891
```

```
## 10 scale(YR.ISOL)      -0.00633    0.0809   -0.163     0.149  1.00   1816
## 11 sigma                 6.59       0.698    5.26      7.99   1.00   1727
## 12 mean_PPD                19.2      1.30     16.7     21.7   0.999  1859
## 13 log-posterior      -198.        2.79    -204.     -193.   1.00   1645
```

We could now look and see the conf int to make conclusions which ones overlap with 0 or not. But that's not the point. The Bayesian way would be to calculate a prob for each of these. What is the prob that the value is 0? What is the PROBABILITY that birds have declined in fGRAZE5? If the intercept does not overlap with conf int that encompass 0, the prob must at least be 95%. But if the intercept overlaps with conf int that encompass 0, we have another prob, which we can only determine with Bayesian statistics.

What happens now?

We could do Dredging = fit ev combi of model with AREA and fGRAZE5, and determine the best predictor. Murray did this, and there were 64 models, and they picked the best AIC. BUT that does not mean that it is an ecologically sensible model. Murray then focused models that focused on diff aspects, and picked the one that was more sensible.

However, often people only fit one model and don't bother about fitting others, which is very old school (before computing).

An alternative option is to fit HORSESHOE PRIORS. They effectively shrink effects which aren't important. So if it isn't important, it shrinks the effect down as if it was nothing. Whether you include that predictor in the model or not is almost irrelevant. It is not having any impact. rstanarm doe snot support these priors, but it is an option in Bayesian Statistics. -> We'd have to use the package brms, or write it ourselves. In this class, we won't continue with this option though.

We will therefore go with the dredging option, creating ecologically relevant models:

A three-way interaction is the most complex we should go (model 4c, see below).

Fit the final models

We have to specify scale(variable) for every continuous variable (#factor variable with different categories). scale() centers the data.

Distance/Connectivity model

```
loyn.rstanarm4a<-update(loyn.rstanarm3, .~scale(log(DIST))*scale(log(LDIST)),
                           diagnostic_file=file.path(tempdir(), "dfa.csv"))
```

We are taking an existing model (rstanarm3), and we are changing the predictors, but not the response. A dot means “whatever it used to be” to the left, and on the right handside we replace whatever was there with the part on the right. We are now including the effect of distance, of log dist and the interaction btw them.

We are scaling our variables, but we'd want to do that for Bayesian anyway.

rstanarm unfortunately does not capture all the info needed to support a Bayes factor. We therefore have to specify the following:

```
diagnostic_file=file.path(tempdir(), "dfa.csv"))
```

We are telling rstanarm to dump the info into a temporary directory, which will disappear in 30 days (or whatever the temp clearing schedule is for our computer). rstanarm will therefore be able to use that information that the rstan object needed, but we had to hack it separately.

Area and Graze model

```
loyn.rstanarm4b<-update(loyn.rstanarm3,.~scale(log(AREA))*fGRAZE,
                           diagnostic_file=file.path(tempdir(), "dfa.csv"))
```

Although we created these temporary files, we don't need to keep track of them at all.

Area, Grazing and Years of Isolation

```
loyn.rstanarm4c<-
  update(loyn.rstanarm3,.~scale(log(AREA))*fGRAZE*scale(YR.ISOL),
         diagnostic_file=file.path(tempdir(), "dfa.csv"))
```

Altitude

```
loyn.rstanarm4d<-update(loyn.rstanarm3,.~scale(ALT),
                           diagnostic_file=file.path(tempdir(), "dfa.csv"))
```

Null model to compare to (!)

```
loyn.rstanarm4e<-update(loyn.rstanarm3,.~1,
                           diagnostic_file=file.path(tempdir(), "dfa.csv"))
```

Model validation

First of all, in Bayesian analysis, we don't use AIC. AIC can only be calculated from the whole chain, it can't be calculated each sample at a time = We wouldn't get a distribution for it.

Other, analogous AICs for Bayesian are (examples below): ## waic

```
waic(loyn.rstanarm4a)

## Warning:
## 2 (3.6%) p_waic estimates greater than 0.4. We recommend trying loo
instead.

##
## Computed from 1800 by 56 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic    -216.1 3.5
```

```
## p_waic      4.5 0.7
## waic       432.2 6.9
##
## 2 (3.6%) p_waic estimates greater than 0.4. We recommend trying loo
instead.
```

- waic: The waic estimate indicates the Widely applicable information criteria. It is like the AIC, but indicates a distribution for it, indicating a SE. As with the AIC, it does not say anything by its own, but has to be compared against other models.
- p_waic: Indicates the complexity of the model, i.e. how many parameters are used in the model. They are often not integers, so can't be compared to df. However, if several models have similar waic, you'd pick the one with lower p_waic.
- elpd_waic: I think we can ignore this one, Murray did not say anything about it.

loo (TAKE THIS ONE)

Specify the package (brms)

```
brms:::loo(loyn.rstanarm4a,k_threshold=0.9)

## All pareto_k estimates below user-specified threshold of 0.9.
## Returning loo object.

##
## Computed from 1800 by 56 log-likelihood matrix
##
##           Estimate   SE
## elpd_loo    -216.2 3.5
## p_loo        4.6  0.8
## looic       432.4 7.0
## -----
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                               Count Pct.  Min. n_eff
## (-Inf, 0.5]   (good)    55  98.2%  543
## (0.5, 0.7]   (ok)      1   1.8%  704
## (0.7, 1]     (bad)     0   0.0% <NA>
## (1, Inf)     (very bad) 0   0.0% <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

loo = leave one out (package brms)

Runs the Bayesian analysis, leaves one observation (a row) out, reruns the model, leaves one observation (a row) out, reruns... It simulates leaving one out and calculating information criterions.

If one information was particularly important, it is not gonna work.

The second part gives you a measure of the influence of the observation:

96% of our observations are considered very good. One is considered ok, the other one very bad. If an observation is bad, loo will rerun the analysis. The default cut off (k threshold) is of 0.7, which is quite low, so some people prefer to increase it to 0.9 (better for ecological studies, that is what we have done here).

If all of your observations are fine, loo does not even bother showing you the outcome.

To check what loo does:

Model investigation

How do we compare now two models with loo?

Unfortunately, we can't compare all models together. Murray compares the highest model to the next highest, so it would compare them all, and not doing it sequentially.

We therefore have to manually compare each model to the NULL model (4e).

Connectivity model

```
loo_compare(brms::loo(loyn.rstanarm4a),
            brms::loo(loyn.rstanarm4e))

##          elpd_diff se_diff
## loyn.rstanarm4e  0.0      0.0
## loyn.rstanarm4a -2.6     1.6
```

The diff is -2.5. It means that the number it is next to has a better AIC. But 2.5 is not a big difference, This diff must be somewhere btw 1.1 and 3.1 (1 SD from the diff). If we used 2 SD, which is more common, then it is -2.5 ± 3 . So it might as well be no diff btw the two models.

In other words, our distance/connectivity model does not provide much additional information from the null model.

Habitat model (Area and Graze)

Let's have a go at the habitat model

```
loo_compare(brms::loo(loyn.rstanarm4b),
            brms::loo(loyn.rstanarm4e))

## Warning: Found 2 observation(s) with a pareto_k > 0.7. We recommend
## calling 'loo' again with argument 'k_threshold = 0.7' in order to calculate
## the ELPD without the assumption that these observations are negligible. This
## will refit the model 2 times to compute the ELPDs for the problematic
## observations directly.
```

```
##           elpd_diff se_diff
## loyn.rstanarm4b   0.0      0.0
## loyn.rstanarm4e -30.0     7.3
```

Ignore the warning. In this case, model b has more support than model e (null). The null model e is much worse in this case. This indicates that Habitat has an effect on abundance.

Grazing model (Area, Grazing and Years of Isolation)

```
loo_compare(brms::loo(loyn.rstanarm4c),
            brms::loo(loyn.rstanarm4e))

## Warning: Found 6 observation(s) with a pareto_k > 0.7. We recommend
## calling 'loo' again with argument 'k_threshold = 0.7' in order to calculate
## the ELPD without the assumption that these observations are negligible. This
## will refit the model 6 times to compute the ELPDs for the problematic
## observations directly.

##           elpd_diff se_diff
## loyn.rstanarm4c   0.0      0.0
## loyn.rstanarm4e -19.4     6.6
```

Model c is better than model e (null). Grazing does have a significant effect on abundance.

Altitude model

```
loo_compare(brms::loo(loyn.rstanarm4d),
            brms::loo(loyn.rstanarm4e))

##           elpd_diff se_diff
## loyn.rstanarm4d   0.0      0.0
## loyn.rstanarm4e -3.5     2.3
```

Not a big difference btw model d and e (null)

Although model 4d is a bit better than model 4e, when you take into acc the uncertainty in the 2.2 SE, then this sort of comes out to be 0. Altitude does not have an effect on abundance.

Ratio of support (bayes_factor)

This is a bit outdated, but it shows similar results than loo_compare. It shows what impact our different models have (eg. impact of Area and Grazing on abundance)

We now check the ratio of support from one model to the other model.

Impact of Distance

There aren't many iterations for the Connectivity model (4a), which shows that the model is not that different from the null model (4e).

Impact of Area and Grazing

There are MANY iterations for the Area and Graze model (4b), which shows that the model IS VERY different from the null model (4e).

Plot 1

1- Prepare the AREA axes (like a grid)

```
loyn.list=with(loyn, list(AREA=seq(min(AREA), max(AREA), len=100)))
```

2- Create the predicted data

```
newdata=emmeans(loyn.rstanarm3, ~AREA|fGRAZE, at=loyn.list, type="response") %>%
# our response is ABUNDANCE
  as.data.frame

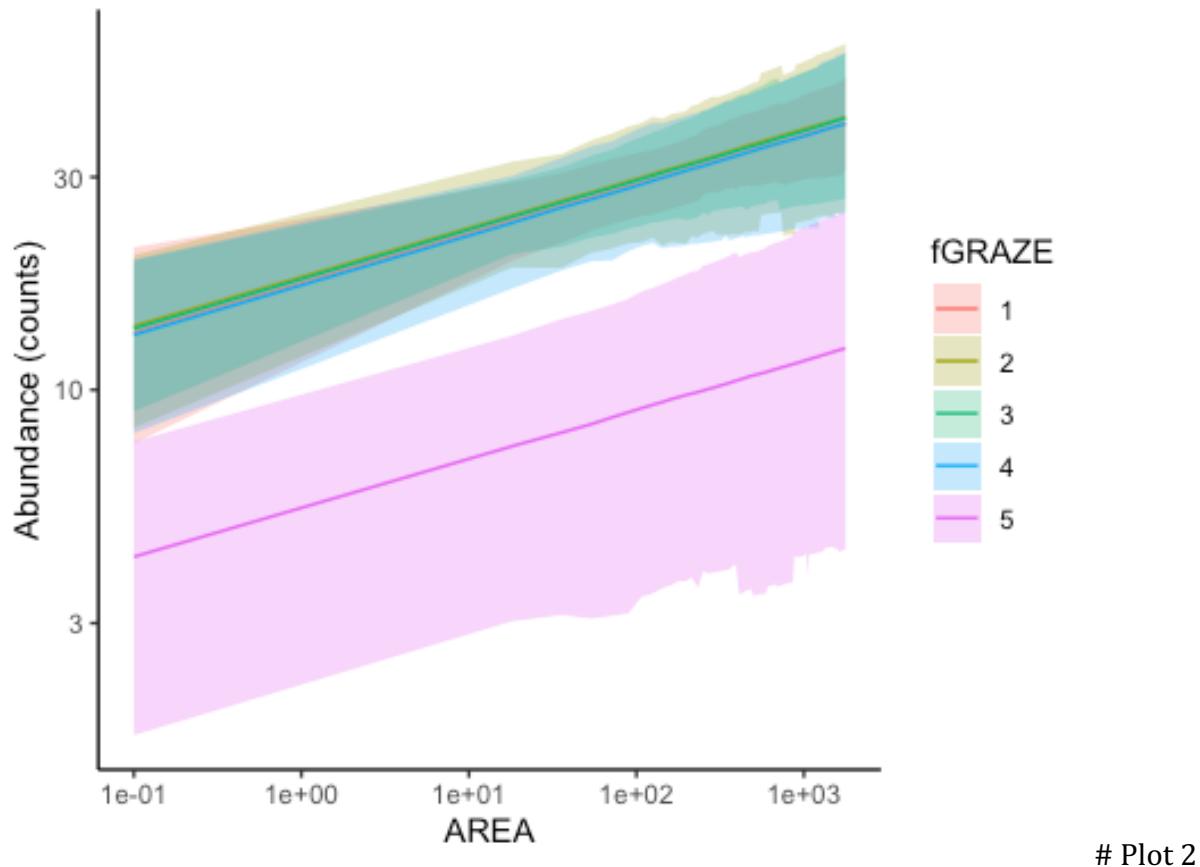
head(newdata)

##          AREA fGRAZE response lower.HPD upper.HPD
## 1  0.10000      1 13.39466  7.592471 20.85713
## 2 17.98788      1 23.85765 19.450914 29.10443
## 3 35.87576      1 25.78495 21.426507 30.58368
## 4 53.76364      1 26.94377 22.771751 31.96501
## 5 71.65152      1 27.78398 23.565462 32.87312
## 6 89.53939      1 28.45249 24.116189 33.56598
```

The area is for the x axis, the response is for the y axis. The lower and upper HPD are for the ribbons.

3- Plot

```
ggplot(newdata, aes(y=response, x=AREA))+
  #geom_point(data=Loyn,aes(y=ABUND))+ 
  geom_ribbon(aes(ymin=lower.HPD,ymax=upper.HPD, fill=fGRAZE), alpha=0.3)+
  geom_line(aes(color=fGRAZE))+
  theme_classic()+
  scale_x_log10()+
  scale_y_log10()+
  ylab("Abundance (counts)")
```



- Spaghetti plot {.tabset .tabset-faded} We could also plot lots of realisations of what abundance would look like in Grazing1, Grazing2, Grazing3, etc. We will do a SPAGHETTI PLOT instead.

1- Prepare the AREA axes (like a grid)

```
loyn.list=with(loyn, list(AREA=seq(min(AREA), max(AREA), len=100)))
```

2- Create the predicted data

```
spaghetti=emmeans(loyn.rstanarm3, ~AREA|fGRAZE, at=loyn.list, type="response")
%>% # our response is ABUNDANCE
  gather_emmeans_draws() %>%
    mutate(Fit=exp(.value)) # we perform the back transform ourselves, so in
theory we don't need the code type="response" above
  head(spaghetti)

## # A tibble: 6 x 7
## # Groups:   AREA, fGRAZE [1]
##   AREA fGRAZE .chain .iteration .draw .value   Fit
##   <dbl> <fct>   <int>      <int> <int> <dbl> <dbl>
## 1 0.1  1        NA        NA     1  2.47 11.8
## 2 0.1  1        NA        NA     2  2.82 16.8
## 3 0.1  1        NA        NA     3  2.65 14.1
## 4 0.1  1        NA        NA     4  2.37 10.7
```

```
## 5 0.1 1 NA NA 5 2.53 12.6
## 6 0.1 1 NA NA 6 2.20 9.04
```

3- Summarising the rows

We have got 1800 lines (rows) that we could plot: The line of best fit, the CI, all grazing parameters and iterations... That means that we could draw 1800 lines, which might look too messy. We can therefore define how many lines/rows to use. Let's tell R to use 100 rows.

```
wch=sample(1:max(spaghetti$.draw), 100, replace=FALSE) # spaghetti$.draw indicates that we need to keep all of its parameters in order to reconstruct the line. Otherwise, R would merge the conf intervals and plot a ribbon for all points (?)
```

We are telling R to label 100 random rows.

```
wch
## [1] 65 537 1253 504 992 434 119 1116 411 1555 753 457 240
645 451
## [16] 1067 746 649 980 1558 1777 1663 342 1650 993 358 747 754
977 379
## [31] 82 1251 234 525 1138 1070 1479 653 1418 493 1472 1740 1046
1381 145
## [46] 3 937 1743 255 1301 1410 1416 783 1573 259 1585 536 94
1024 320
## [61] 1523 821 1356 632 1725 972 1636 1218 293 1039 660 623 85
840 568
## [76] 870 1604 1759 340 693 403 1718 578 352 634 1631 22 822
1368 1145
## [91] 1271 1127 1212 1178 461 1525 1705 715 880 511
```

These are the 100 samples/draws we are gonna keep to plot 100 lines. Now we tell R to filter our spaghetti dataset to keep those 100 rows

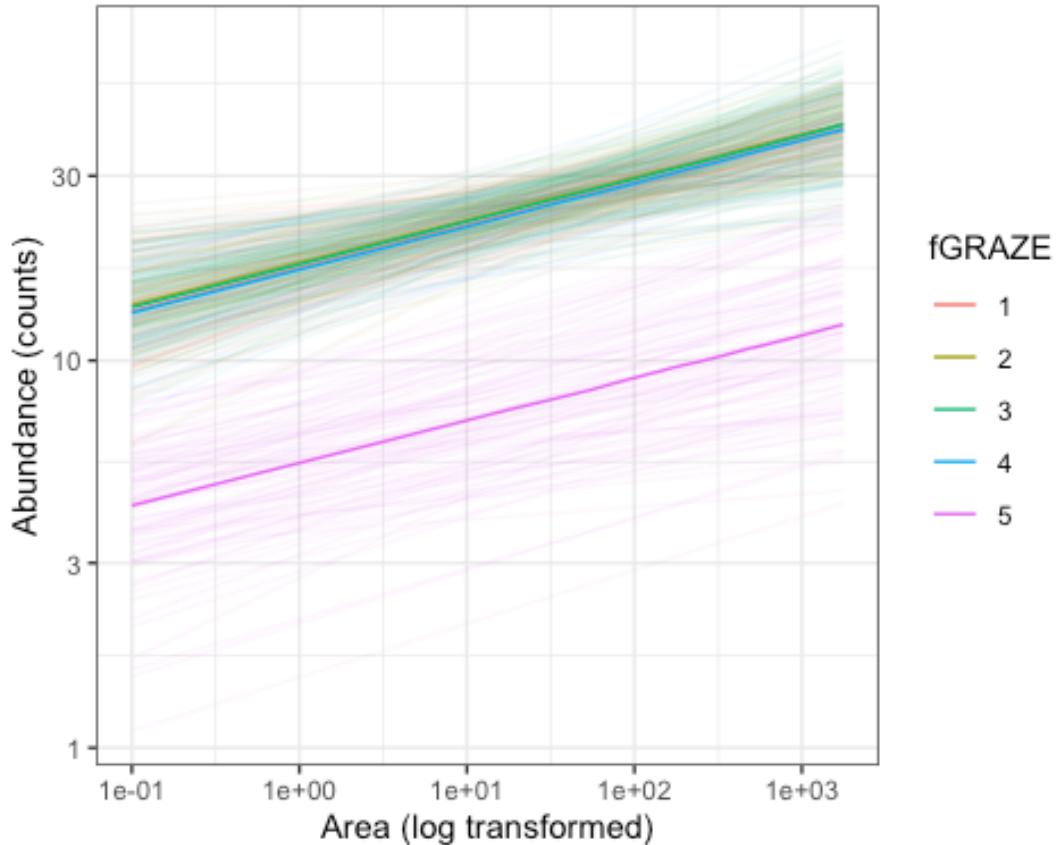
```
spaghetti=spaghetti %>%
  filter(.draw %in% wch)
```

4- Plot

And finally, we can draw them

```
ggplot(newdata, aes(y=response,x=AREA))+
  geom_line(data=spaghetti, aes(y=Fit, x=AREA, color=fGRAZE,
                                group=interaction(fGRAZE,.draw)),alpha=0.05)+
#Set alpha quite Low, so that you can see the individual line for each Graze parameter
  geom_line(data=newdata, aes(y=response,x=AREA, color=fGRAZE))+
  theme_bw()+
  scale_x_log10()+scale_y_log10()+
```

```
ylab("Abundance (counts)")+
xlab("Area (log transformed)")
```



We have created lots and lots of lines, that all blend to create the impression of a ribbon, but without an actual ribbon.

TO MAKE YOUR ANALYSIS REPRODUCIBLE IN BAYESIAN

For example for sharing data in the Supp Mat

- (a) Not sure what this one does or where to write it: `set.seed(123)`
- (b) Cache the model

Within any given script though, you don't need to cache every chunk. Cache only the chunks running Bayesian models. That way, our script will be reproducible (the Bayesian results won't change if running the model again and again)

`cache=TRUE` -> Write this in the curly brackets of each code chunk:

```
# The chunk is called fitModel (this is just an example name)
# cache=TRUE caches the output of this code. It will ONLY rerun and re-cache
# if it detects that the code inside has changed
```

```
# dependson='readData' --> This chunk will rerun if the chunk reading in the  
Data has changed
```

(!) DON'T EVER CACHE A CHUNK WITH A CALL TO A LIBRARY. It will cache ok, but that library won't be available to any other chunk, as R won't rerun the library. You could also say cache=FALSE.

(!) Don't ever save the R Environment.

GOOD CODE MANAGEMENT PRACTICE

You need the following folders: - data -> split it into “primary” (raw data) and “processed” (modified data). I might then have a script that reads the processed data and does exploratory data analysis, without having to go back to the primary. We can also have one that does the modelling, so when we do summary figures, we don't have to go back to the very start. Murray's point is, that if we keep our code organised, we can more effectively and efficiently run our analysis. Once the data is processed, we don't have to do that stage anymore. - scripts - results - figures

To save extra steps in running code chunks, Murray recommends defining the following commands:

code_management1

code_management1

The code chunks define what needs to be done, but you had already defined the commands at the beginning of the script. It's a good idea to name the scripts:
“NameProject_AnalysisType”

code_management2

code_management2

IF USING GIT, DON'T FORGET TO COMMIT YOUR CHANGES

If some of Murray's files (.htmls especially) take too long to commit, ignore them.

17_Bayesian_Poisson_counts

Sara Kophamel

16/12/2020

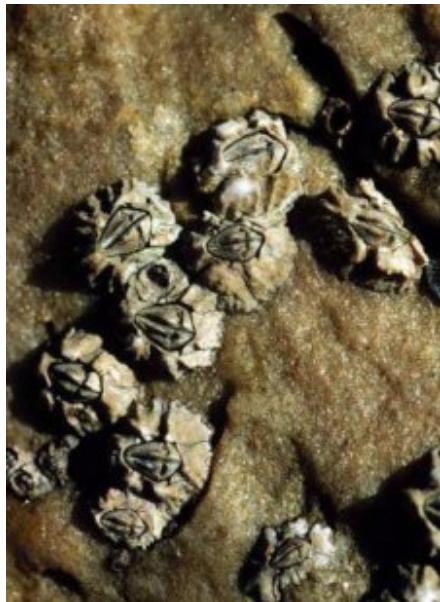
Preparations

Load the necessary libraries

```
library(rstanarm)    #for fitting models in STAN
library(brms)         #for fitting models in STAN
library(coda)          #for diagnostics
library(ggmcmc)        #for MCMC diagnostics
library(bayesplot)      #for diagnostics
library(rstan)          #for interfacing with STAN
library(DHARMa)        #for residual diagnostics
library(emmeans)        #for marginal means etc
library(broom)           #for tidying outputs
library(broom.mixed)
library(tidybayes)      #for more tidying outputs
library(ggeffects)       #for partial plots
library(tidyverse)        #for data wrangling etc
library(patchwork)
```

Scenario

Here is a modified example from @Quinn-2002-2002. Day and Quinn (1989) described an experiment that examined how rock surface type affected the recruitment of barnacles to a rocky shore. The experiment had a single factor, surface type, with 4 treatments or levels: algal species 1 (ALG1), algal species 2 (ALG2), naturally bare surfaces (NB) and artificially scraped bare surfaces (S). There were 5 replicate plots for each surface type and the response (dependent) variable was the number of newly recruited barnacles on each plot after 4 weeks.



Six-plated barnacle

Format of day.csv data files

TREAT	BARNACLE
ALG1	27
..	..
ALG2	24
..	..
NB	9
..	..
S	12
..	..
TREAT	Categorical listing of surface types. ALG1 = algal species 1, ALG2 = algal species 2, NB = naturally bare surface, S = scraped bare surface.
BARNACLE	The number of newly recruited barnacles on each plot after 4 weeks.

Read in the data

```
day = read_csv('data/day.csv', trim_ws=TRUE)
glimpse(day)

## Rows: 20
## Columns: 2
## $ TREAT    <chr> "ALG1", "ALG1", "ALG1", "ALG1", "ALG1", "ALG2", "ALG2",
## "ALG...
```

```
## $ BARNACLE <dbl> 27, 19, 18, 23, 25, 24, 33, 27, 26, 32, 9, 13, 17, 14,
22, 1...
```

This is count data (barnacles) => We are aiming for a POISSON distribution

Start by declaring the categorical variables as factor.

```
day = day %>% mutate(TREAT=factor(TREAT))
```

Model formula:

$$\begin{aligned} y_i \sim \text{Pois}(\lambda_i) \ln(\mu_i) &= \\ \boldsymbol{\beta} \cdot \mathbf{X}_i + \beta_0 &\sim N(0, 10) \cdot \beta_{1,2,3} \\ &\sim N(0, 1) \end{aligned}$$

where β is a vector of effects parameters and \mathbf{X} is a model matrix representing the intercept and treatment contrasts for the effects of Treatment on barnacle recruitment.

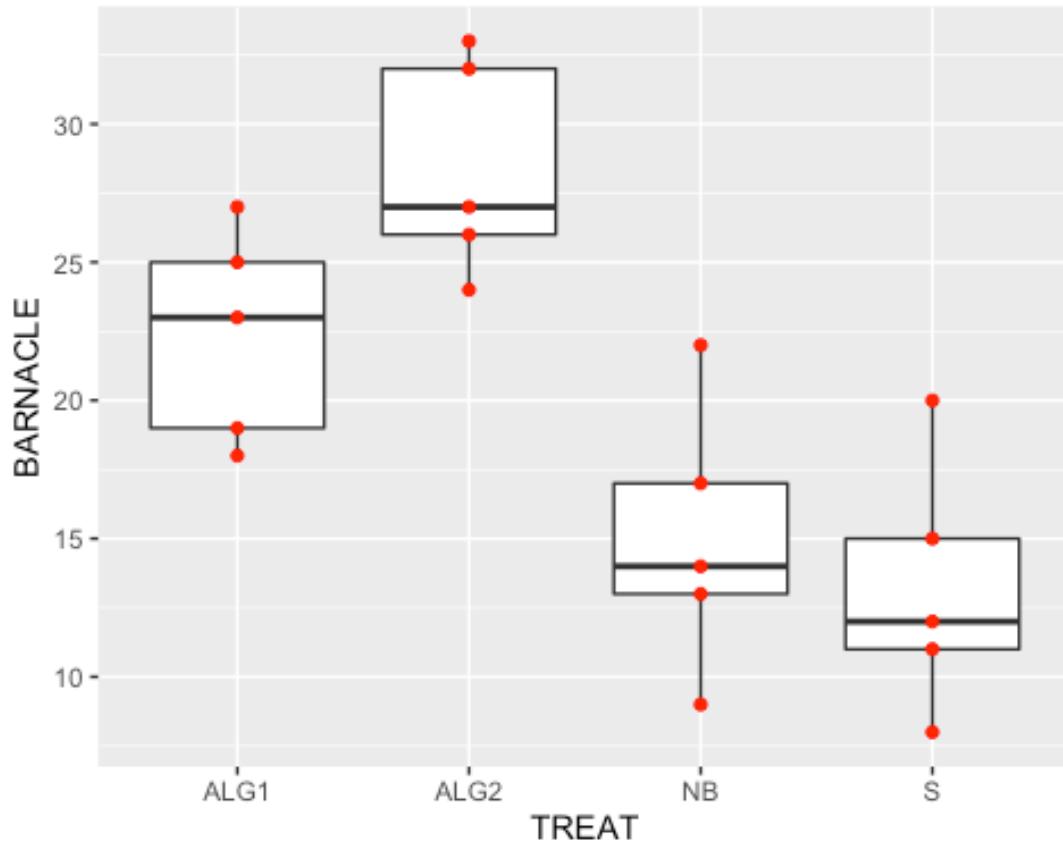
I did not fully understand the following, but Murray explained the equation: By log transforming our data, we got the following equation: $\log(\lambda_i) = \beta_0 + \beta_1 X_2 + \beta_2 X_3 + \beta_3 X_4 \rightarrow x_2$ is the diff btw groups x1-x2?

Whenever we are doing analysis of variance, our dummy var get coded as 0 and 1 numbers:
y TREAT x1 x2 x3 x4 . ALG1 1 0 0 0 . ALG1 1 0 0 0 . ALG2 1 1 0 0 . ALG2 1 1 0 0 . NB 1 0 1 0 .
NB 1 0 1 0 . S 1 0 0 1 . S 1 0 0 1

Therefore, we need to modify the equation: $\beta_0 \sim N(3, 5)$ $\beta_1 - \beta_3 \sim N(0, 2)$ # 0=zero centered;
2=diff after centering the data Because this is a Poisson distrib, we don't have sigma.

Exploratory data analysis

```
ggplot(day, aes(y=BARNACLE, x=TREAT)) +
  geom_boxplot() +
  geom_point(color="red")
```



We can see that: - we don't have zeros (useful to know for count data) - ALG1 and ALG2 seem to have an effect on barnacle count, compared to NB and S

Fit the model

Decide on the parameters for the model

We have to decide: - the priors (let's not care about the default ones) - number of iterations: let's pick 5000 as usual - warmup: let's pick 1000 as usual (we could also pick 2000, does not really matter, as it usually settles even before 1000) - chains: let's pick 3 as usual (3-4, does not matter) - thinner: let's pick 5 as usual (good starting point always)

Initial model - only priors

If we let R define the default priors, we'd call the model "day.rstanarm". But because we are skipping this step, we go for day.rstanarm2.

```
day.rstanarm2=stan_glm(BARNACLE~TREAT, data=day,
                       family=poisson(link="log"),
                       prior_intercept=normal(3,5,autoscale=FALSE), # HOW DO
WE SELECT OUR PRIORS
                       prior=normal(0,2,autoscale=FALSE),# autoscale=FALSE as
```

we already considered the priors we want; we don't have to scale them to our data

prior_PD=TRUE, # only the priors are involved, as we wanna see whether our priors look vaguely sensible (is the scheme not too wide and not too narrow?). Because we log transformwd our data, we will have to remember to backtransform the priors later!

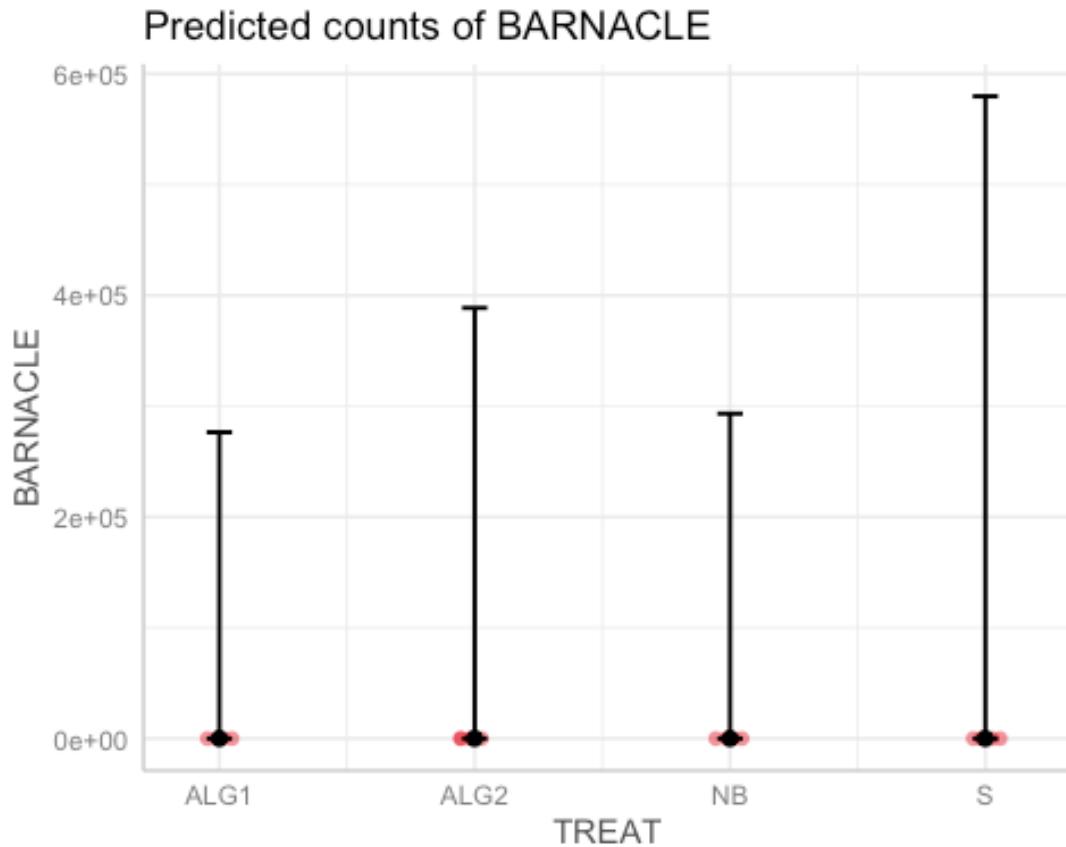
```
iter=5000,
warmup=1000,
chains=3,
thin=5,
refresh=0)
```

Let's have a quick look at our model

```
ggpredict(day.rstanarm2) %>%
```

```
  plot(add.data=TRUE)
```

```
## $TREAT
```



We see that our priors are REALLY wide. If we had a problem with our model, we would narrow down our priors. This involves the risk of the sampler having trouble sampling in the right location (it might get lost sampling in the wrong location). However, we better start off with wide priors, and then narrow them down; otherwise we would be creating a model which may not reflect the data.

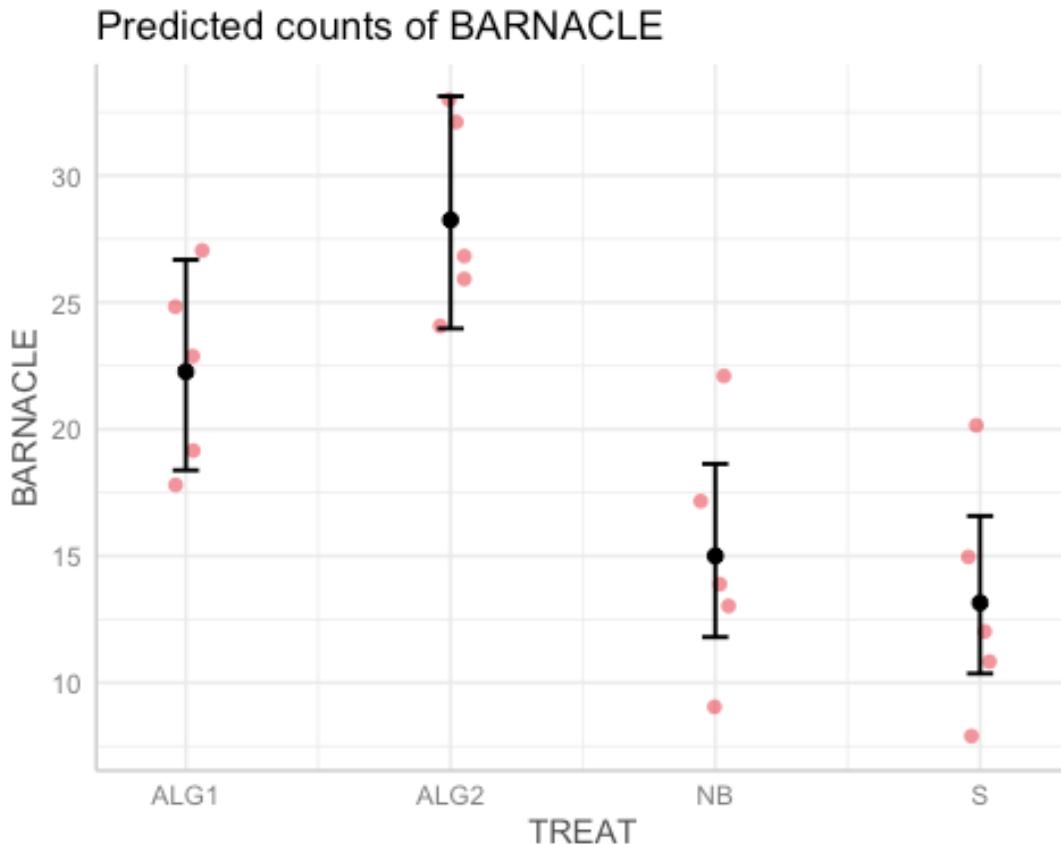
Model to form the posterior priors

The likelihood / the data now informs about the posteriors. In the previous model, we had set the priors = posteriors.

```
day.rstanarm3=update(day.rstanarm2, prior_PD=FALSE)

ggpredict(day.rstanarm3) %>% plot(add.data=TRUE)

## $TREAT
```

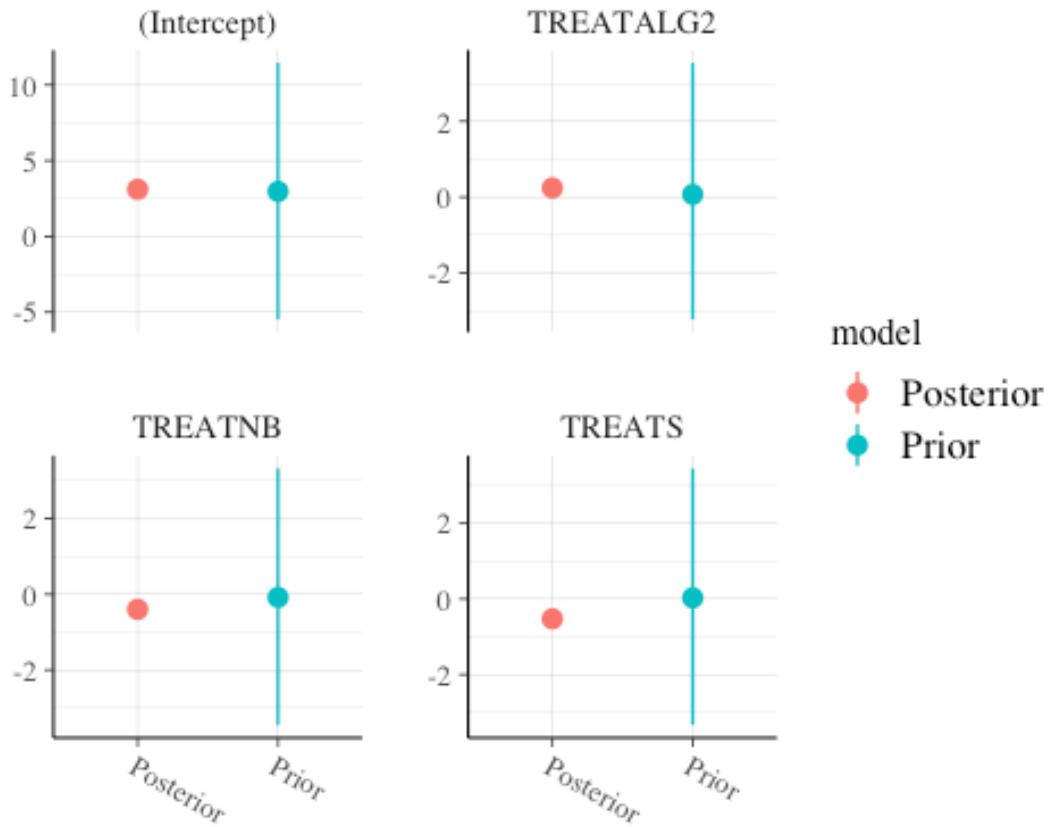


In the absence of taking into consideration our responses, the predictions were too wide. But now that we implemented the actual data to define the posteriors, the predictions are actually ok. In other words, we have let the DATA define the prediction, not the priors.

The idea of the priors is either "I do have some info I want to include" or "I have no idea". You want to set the priors to prior_PD=TRUE if you had informative priors; if you knew how the priors define the data. But if you had a really small sample size, and had noisy data, you wanna set very vague priors, and let the data/posteriors decide the predictions -> This is usually the case.

Posterior vs prior

```
posterior_vs_prior(day.rstanarm3, color_by="vs", group_by=TRUE,
                    facet_args=list(scales="free_y"))
```



The graph is good, as the priors are not equally large. If they were equally large, our prior would have impact on the posterior, which we don't want to happen.

A bad graph would be: I I I pos prior This would indicate that the predictions are driven by our prior. This happens a lot in medical research, where the priors might be recycled across experiments.

A good graph is: I I I pos prior Which indicates that the posteriors are driving the predictions.

Another option would be to use a UNIFORM PRIOR, which just says: "The values must be in the range x and y = My slope must have a specific range. As long as the data is within that range, all is good". The boundaries of these priors are informative, as they are blocking the result. This is useful for negative estimates. Unfortunately, these priors often don't work so well when defining the model. So we won't use them.

Summary of the model

```
summary(day.rstanarm3)
```

```

## 
## Model Info:
##   function: stan_glm
##   family: poisson [log]
##   formula: BARNACLE ~ TREAT
##   algorithm: sampling
##   sample: 2400 (posterior sample size)
##   priors: see help('prior_summary')
##   observations: 20
##   predictors: 4
##
## Estimates:
##           mean    sd   10%   50%   90%
## (Intercept) 3.1  0.1  3.0  3.1  3.2
## TREATALG2   0.2  0.1  0.1  0.2  0.4
## TREATNB    -0.4  0.2 -0.6 -0.4 -0.2
## TREATS     -0.5  0.2 -0.7 -0.5 -0.3
##
## Fit Diagnostics:
##           mean    sd   10%   50%   90%
## mean_PPD 19.8  1.4 18.0 19.7 21.6
##
## The mean_ppd is the sample average posterior predictive distribution of
## the outcome variable (for details see help('summary.stanreg')).
##
## MCMC diagnostics
##           mcse Rhat n_eff
## (Intercept) 0.0  1.0 2153
## TREATALG2   0.0  1.0 2119
## TREATNB    0.0  1.0 2398
## TREATS     0.0  1.0 2003
## mean_PPD   0.0  1.0 2390
## log-posterior 0.0  1.0 2221
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude
## measure of effective sample size, and Rhat is the potential scale reduction
## factor on split chains (at convergence Rhat=1).

```

What can we see here, is that ALG2 is diff to ALG1: Estimates: mean sd 10% 50% 90%
 (Intercept) 3.1 0.1 3.0 3.1 3.2 TREATALG2 0.2 0.1 0.1 0.2 0.4 -> THE 90% CREDIBILITY
 INTERVAL DOES NOT INCLUDE 0, THEREFORE THERE IS EVIDENCE THAT THE EFFECT
 BTW ALG2 AND ALG1 IS NON-ZERO TREATNB -0.4 0.1 -0.6 -0.4 -0.2 TREATS -0.5 0.2 -0.7 -
 0.5 -0.3

The prob that we get more recruitment on ALG2 than ALG1. If we were to look at this column, we'd had four columns: (Intercept=ALG1) ALG2 NB S

ALG2 = effects of ALG2 vs ALG1

Could we not take the ALG2 column and count how many are above zero? In this case, the prob is counting ALG2 all up, and dividing by 2000.

We could back transform them to get the prob:

```
exp(3.1) # this is the LOG mean of the intercept  
## [1] 22.19795
```

This tells us that the mean of the Intercept is 22.2

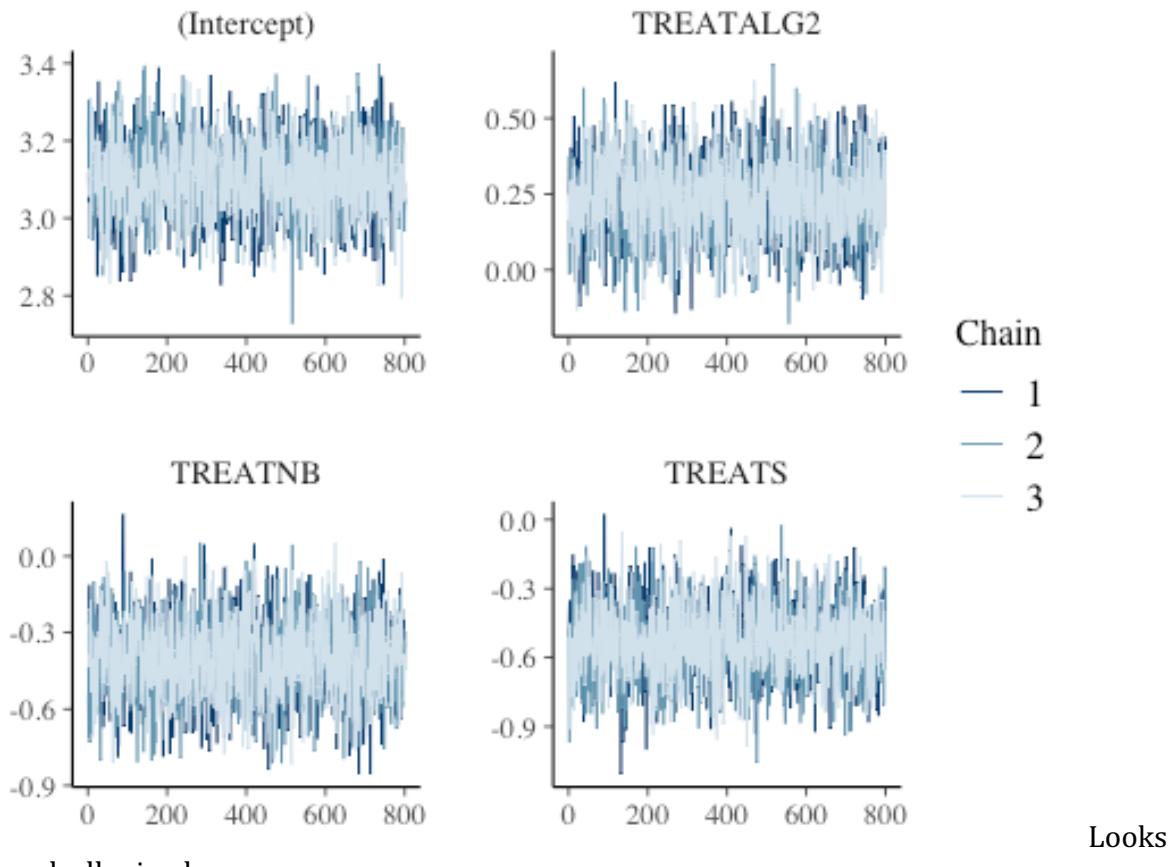
```
exp(0.2) # this is the LOG mean of ALG2  
## [1] 1.221403
```

This tells us that there has been a 1.22x increase from ALG2 to ALG1 = There has been a 22% increase from ALG2 in comparison to ALG1.

MCMC sampling diagnostics

Trace plot

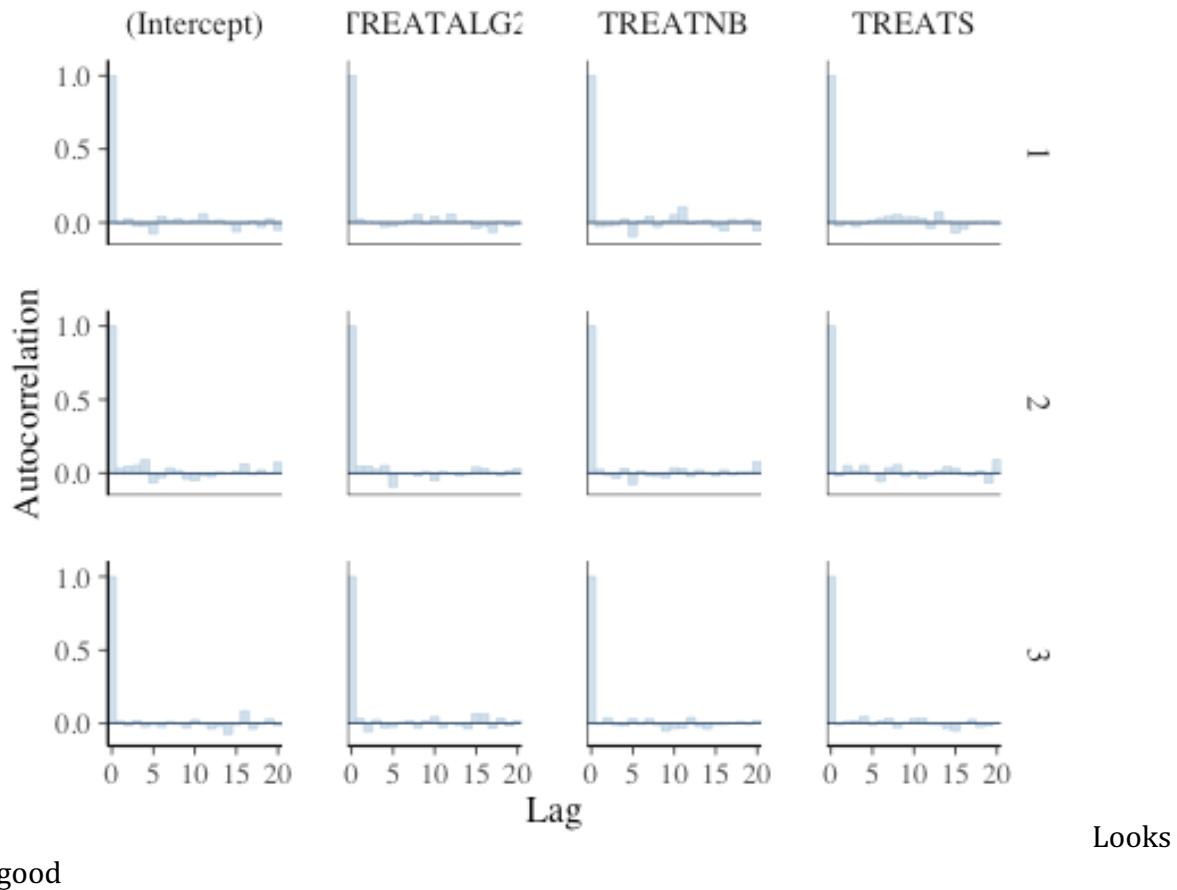
```
plot(day.rstanarm3,plotfun="mcmc_trace")
```



good, all mixed up

Autocorrelation function for thinning

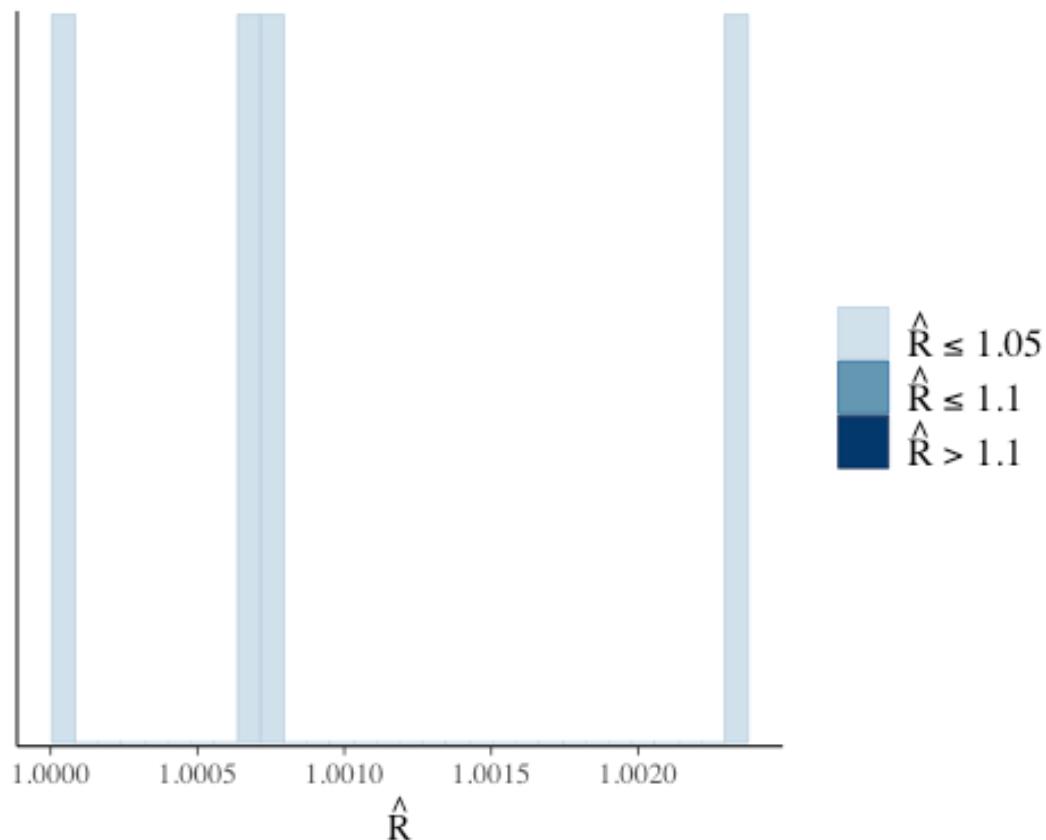
```
plot(day.rstanarm3, "acf_bar")
```



good

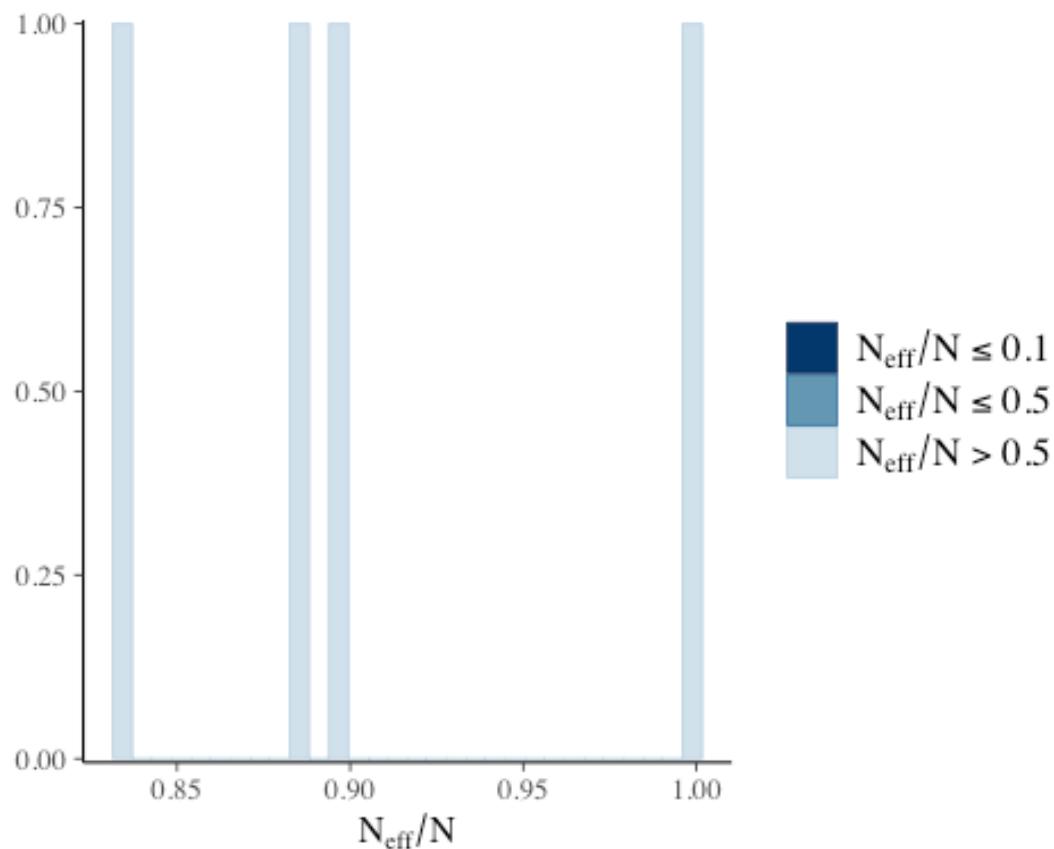
Measure of chain convergence - R hat

```
plot(day.rstanarm3, "rhat_hist")
```



Number of effective samples

```
plot(day.rstanarm3, "neff_hist")
```



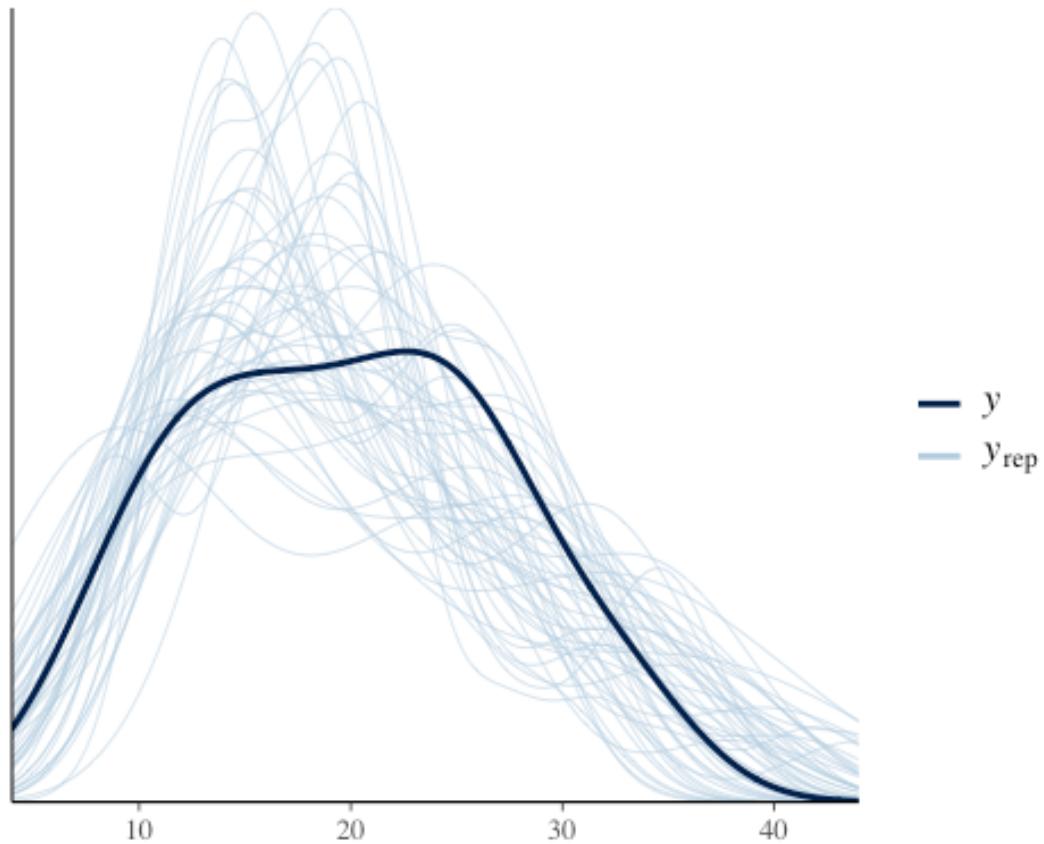
All the MCMC sampling diagnostics look good. BUT even if one was not perfect, we could still continue using our priors.

PPC plots - Posterior probability checks

These plots check if our model are close to reality.

1. Density overlay plot

```
pp_check(day.rstanarm3, plotfun="dens_overlay")
```

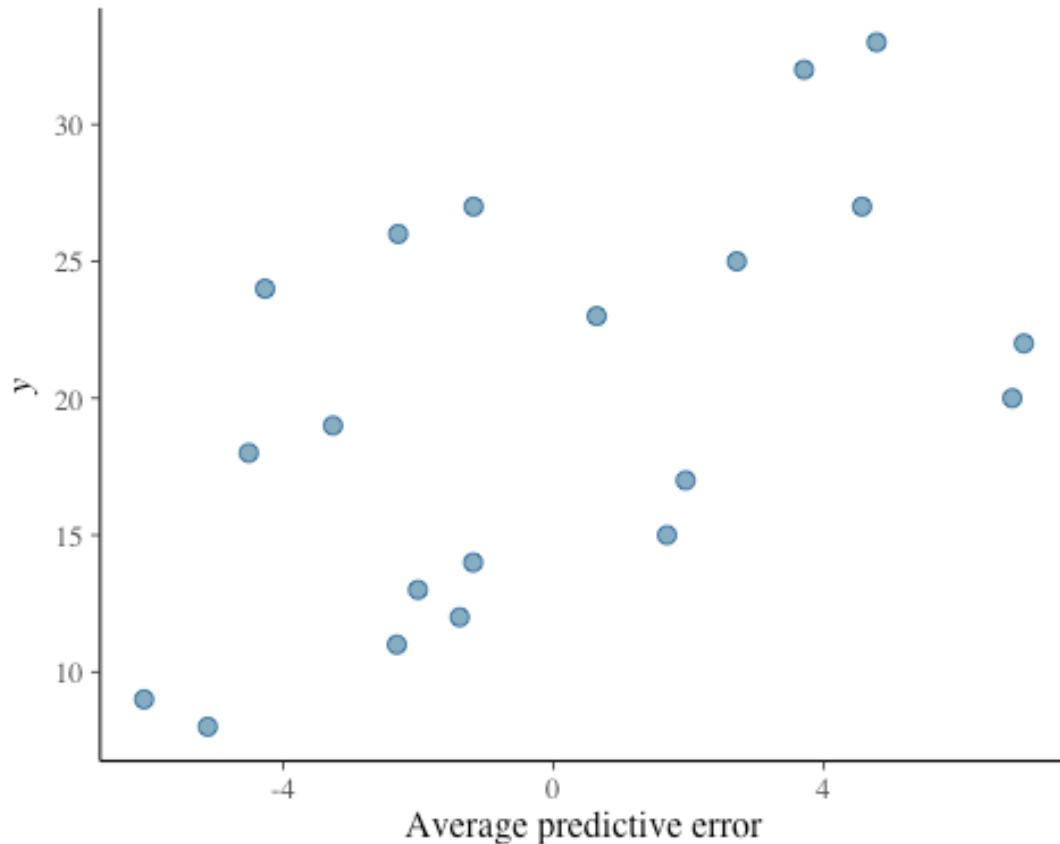


This plot

plots the actual distribution of our response (in this case YIELD, dark blue line). And each of the lighter blue lines are realisations that have been drawn from the model; i.e. they are predictions from the model. You can see that they are fairly consistent, i.e. the model is in fairly big line with reality.

2. Error scatterplot

```
pp_check(day.rstanarm3, plotfun="error_scatter_avg")
```

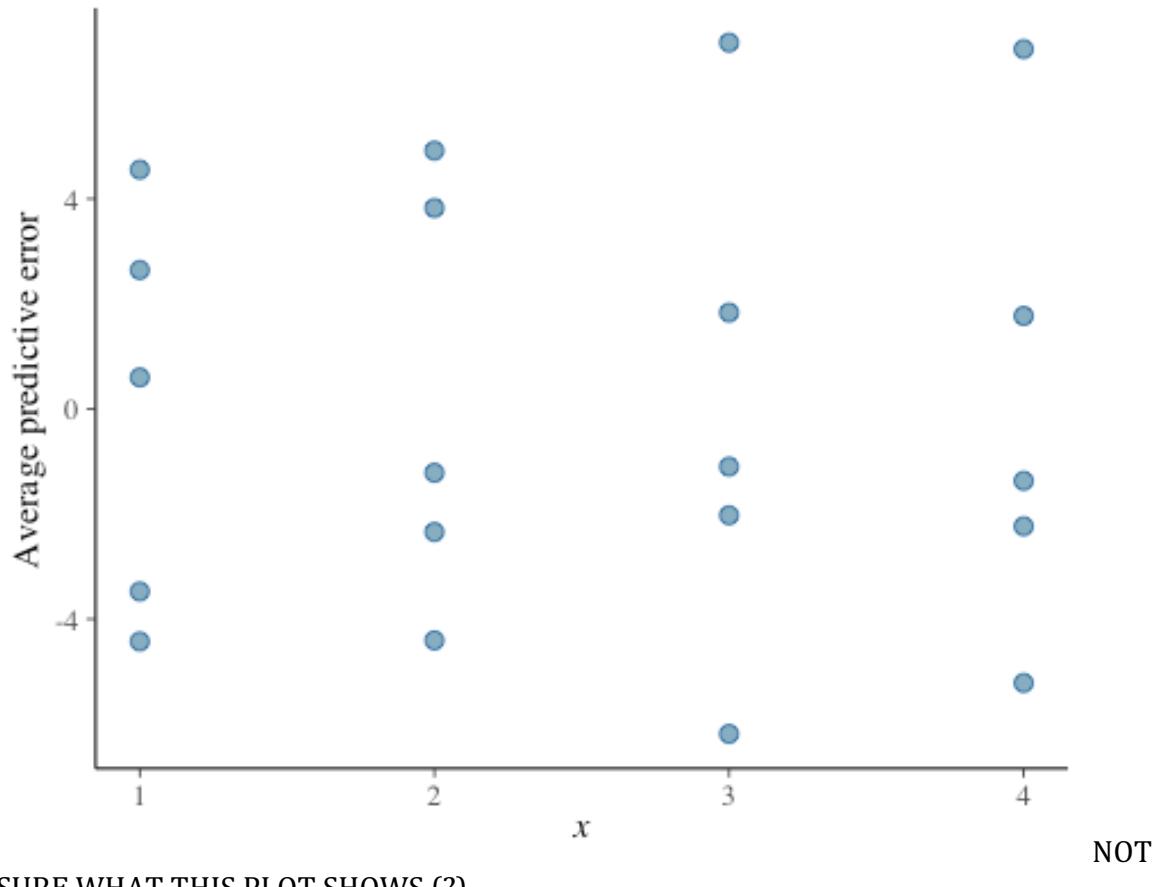


You don't want to see a plot where points are very far away from other points = That point would have a very big error. We would have been very bad at predicting that point. And possibly, that point was an outlier. This plot is analogous to a residual plot.

This output is not ideal, as the pred error seems to be lower for the low values as for the higher ones. It should be more horizontal. The pred error does not take into acc as well as it should that we got a log transformation. This happens in Poisson distrib. Murray reckons it has to do with our log transformation.

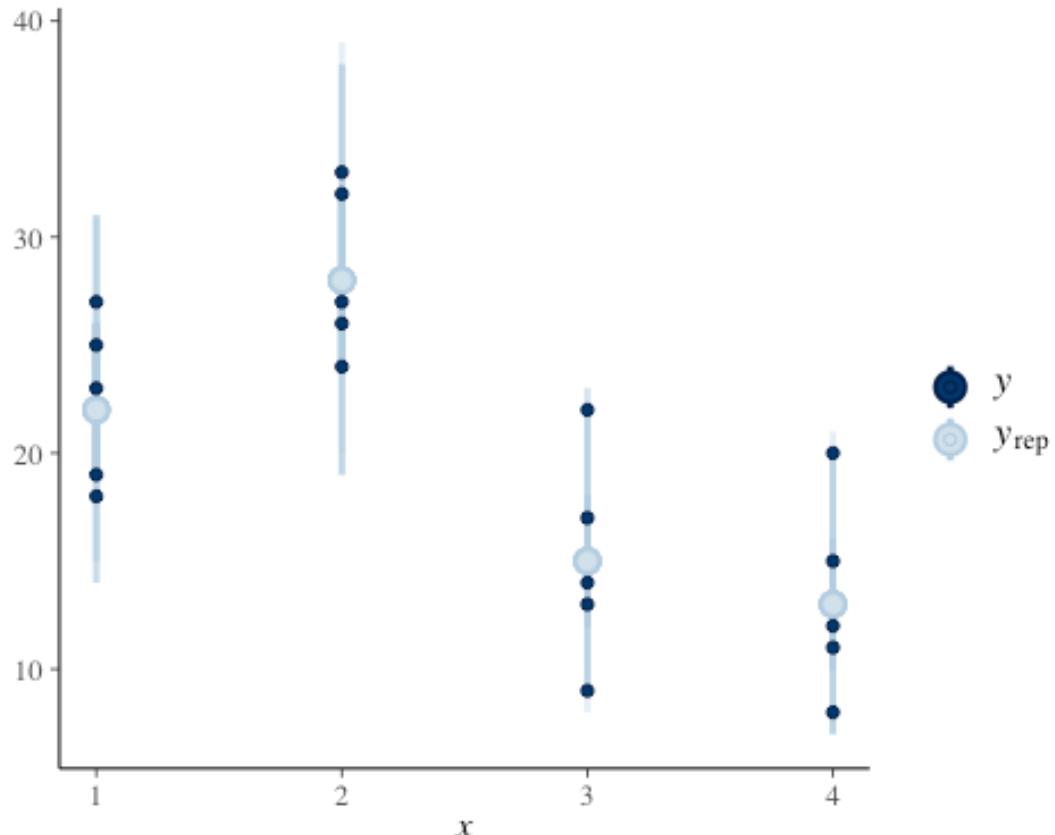
3. Predictive error plotted against each response

```
pp_check(day.rstanarm3, x=as.numeric(day$TREAT),
plotfun="error_scatter_avg_vs_x")
```



4. Raw data vs pred range of responses.

```
pp_check(day.rstanarm3, x=as.numeric(day$TREAT), plotfun="intervals")
```



The

predictions are similar to the raw data, which is what we wanna see. It is an extension to the first plot of this series. We don't want to see black dots near the conf bands.

The predictions (y_{rep}) are about the same as the raw data (y).

1. DHARMA residuals

```

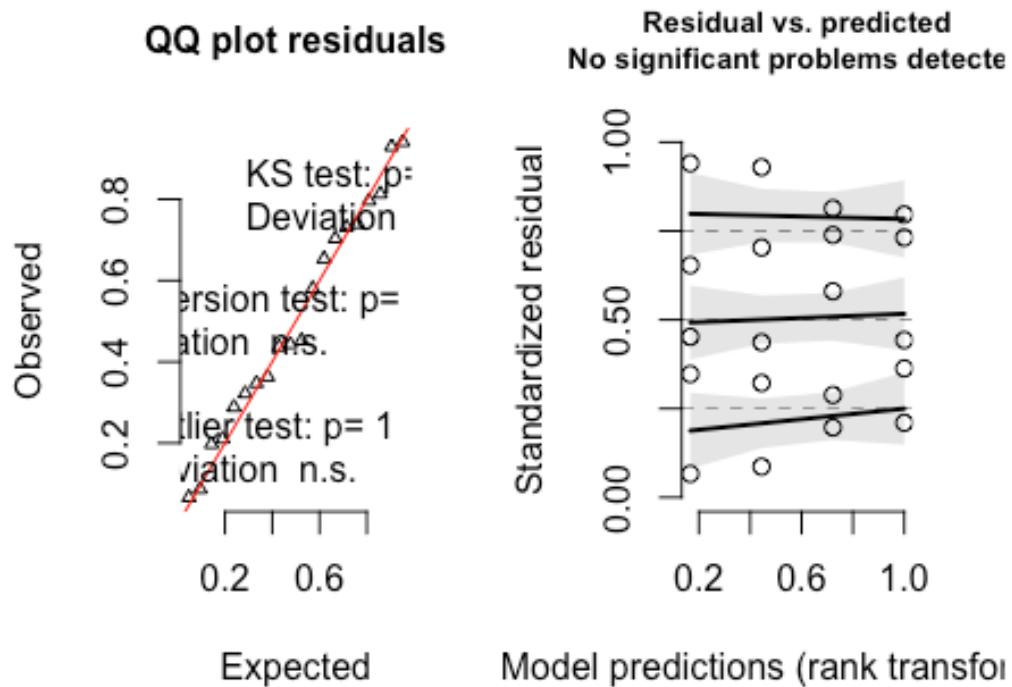
preds<-posterior_predict(day.rstanarm3,nsamples=250,summary=FALSE) # We tell
R to give us 250 predictions to do simulations (standard number)

day.resids<-createDHARMA(simulatedResponse=t(preds), # predicted resp (best
fit values)
                           observedResponse=day$BARNACLE, # this is our raw
response values, the YIELD in the example (this is the only thing we'd change
when using other datasets)
                           fittedPredictedResponse=apply(preds, 2, median),
#We tell R to take the predicted values, and calculate the median for each
                           integerResponse=TRUE) # We tell R we are working
with COUNT data

plot(day.resids)

```

DHARMA residual diagnostics



DHARMA residuals look good.

We would be worried if we'd see Overdispersion (overdispersed Poisson models are not good)

Predictions

1- Display all predictions as tibble

We first have to create the predictions (2400 per treatment) and save them as tibble

```
day.mcmc<-as.matrix(day.rstanarm3) %>%
  as_tibble()

day.mcmc

## # A tibble: 2,400 x 4
##   `"(Intercept)"` TREATALG2 TREATNB TREATS
##       <dbl>      <dbl>     <dbl>    <dbl>
## 1        3.12     0.249    -0.266   -0.511
## 2        3.06     0.374    -0.255   -0.414
## 3        3.04     0.330    -0.235   -0.353
## 4        3.05     0.351    -0.114   -0.362
```

```

## 5      3.13   0.137 -0.505 -0.486
## 6      2.95   0.404 -0.418 -0.378
## 7      3.19   0.115 -0.476 -0.820
## 8      3.25   0.163 -0.306 -0.674
## 9      3.02   0.209 -0.351 -0.370
## 10     3.10   0.354 -0.468 -0.518
## # ... with 2,390 more rows

```

2- Exponentiate the log variables

```

day.mcmc %>%
  mutate(ALG2.resp=exp(TREATALG2))

## # A tibble: 2,400 x 5
##   `(Intercept)` TREATALG2 TREATNB TREATS ALG2.resp
##   <dbl>       <dbl>    <dbl>    <dbl>    <dbl>
## 1 3.12        0.249   -0.266  -0.511    1.28
## 2 3.06        0.374   -0.255  -0.414    1.45
## 3 3.04        0.330   -0.235  -0.353    1.39
## 4 3.05        0.351   -0.114  -0.362    1.42
## 5 3.13        0.137   -0.505  -0.486    1.15
## 6 2.95        0.404   -0.418  -0.378    1.50
## 7 3.19        0.115   -0.476  -0.820    1.12
## 8 3.25        0.163   -0.306  -0.674    1.18
## 9 3.02        0.209   -0.351  -0.370    1.23
## 10     3.10   0.354   -0.468  -0.518    1.42
## # ... with 2,390 more rows

```

The ALG2.resp indicate the possible changes in percentage (there is 24000 predictions in total): 1.22 = ALG2 has 22% more recruitment than ALG1 1.20 = ALG2 has 20% more recruitment than ALG1 1.16 = ALG2 has 16% more recruitment than ALG1 etc

3- Pick a column to compare with

What is the prob that there will be more recruitment in ALG2 in comparison to ALG1?

```

day.mcmc %>% mutate(ALG2.resp=exp(TREATALG2)) %>%
  summarise(p=sum(ALG2.resp>1)/n()) # 1="more"

## # A tibble: 1 x 1
##       p
##   <dbl>
## 1 0.969

```

The prob that there will be more recruitment on the ALG2 surface in comparison to the ALG1 surface is 97.5%

What is the prob that there will be more than 10% more recruitment in ALG2 in comparison to ALG1?

```

day.mcmc %>% mutate(ALG2.resp=exp(TREATALG2)) %>%
  summarise(p=sum(ALG2.resp>1.1)/n()) # 1.1="more than 10%"

```

```
## # A tibble: 1 × 1
##      p
##   <dbl>
## 1 0.871
```

The prob that there will be more than 10% more recruitment on the ALG2 surface in comparison to the ALG1 surface is 88%.

Report: "There is strong evidence that..."

To change the column we are comparing the intercept (ALG1) with, just change the column name in the code:

4- Multiple comparisons - Post-hoc test (Tukey's)

If you want to compare across all columns, do a multiple comparisons post hoc test

```
day.rstanarm3 %>% emmeans(pairwise~TREAT, type="response") # type=response
backtransforms the log transf data

## $emmeans
##   TREAT rate lower.HPD upper.HPD
##   ALG1  22.3    18.3    26.5
##   ALG2  28.3    23.9    33.0
##   NB    15.0    11.8    18.5
##   S     13.2    10.3    16.5
##
## Point estimate displayed: median
## Results are back-transformed from the log scale
## HPD interval probability: 0.95
##
## $contrasts
##   contrast    ratio lower.HPD upper.HPD
##   ALG1 / ALG2 0.787    0.607    0.999
##   ALG1 / NB   1.487    1.055    1.933
##   ALG1 / S    1.690    1.202    2.251
##   ALG2 / NB   1.883    1.407    2.449
##   ALG2 / S    2.131    1.609    2.846
##   NB / S     1.143    0.803    1.559
##
## Point estimate displayed: median
## Results are back-transformed from the log scale
## HPD interval probability: 0.95
```

"contrasts" controls the order in which this gets compared: \$contrasts contrast ratio
lower.HPD upper.HPD ALG1 / ALG2 0.788 0.622 0.999 -> Tells us that ALG1 is 1-0.788 = 0.2 = 20% lower than ALG1
ALG1 / NB 1.490 1.073 1.927 -> Tells us that ALG1 is 49% bigger than NB
ALG1 / S 1.687 1.235 2.249 -> Tells us that ALG1 is 69% bigger than S
ALG2 / NB 1.889 1.381 2.422 -> Tells us that ALG2 is 89% bigger than NB
ALG2 / S 2.138 1.581 2.784 -> Tells us that ALG2 is 113% bigger than S
NB / S 1.132 0.806 1.546 -> Tells

us that THERE IS NO CHANGE BTW NB AND S, as the lower and upper HPD encompass ONE (because our data, now backtransformed, is centered at 1; the threshold would have been zero if we hadn't back transformed our data) (!)

Every comparison is at their maximum power. There is no adjustments there.

Probabilities within groups

Instead of spitting out the table, we are taking the contrasts part, gathering the emmeans draws, and doing the backtransform:

```
day.em=emmeans(day.rstanarm3,pairwise~TREAT, type="link")$contrasts %>%
  gather_emmeans_draws() %>%
  mutate(Fit=exp(.value))

day.em

## # A tibble: 14,400 x 6
## # Groups:   contrast [6]
##   contrast   .chain .iteration .draw .value   Fit
##   <fct>      <int>     <int> <int> <dbl> <dbl>
## 1 ALG1 - ALG2     NA       NA    1 -0.249 0.780
## 2 ALG1 - ALG2     NA       NA    2 -0.374 0.688
## 3 ALG1 - ALG2     NA       NA    3 -0.330 0.719
## 4 ALG1 - ALG2     NA       NA    4 -0.351 0.704
## 5 ALG1 - ALG2     NA       NA    5 -0.137 0.872
## 6 ALG1 - ALG2     NA       NA    6 -0.404 0.667
## 7 ALG1 - ALG2     NA       NA    7 -0.115 0.891
## 8 ALG1 - ALG2     NA       NA    8 -0.163 0.850
## 9 ALG1 - ALG2     NA       NA    9 -0.209 0.811
## 10 ALG1 - ALG2    NA       NA   10 -0.354 0.702
## # ... with 14,390 more rows
```

How many rows should there be? 2400 rows per contrast, and we got 6 contrasts. We should therefore have 2,400 per comparison we wanna make $\rightarrow 2,400 * 6 = 14,400$ (this is the case)

Using the median HDI, we will now group the probabilities by contrast

```
day.em %>% group_by(contrast) %>%
  median_hdi()

## # A tibble: 6 x 10
##   contrast .value .value.lower .value.upper   Fit Fit.lower Fit.upper
##   <fct>     <dbl>       <dbl>       <dbl> <dbl>      <dbl>      <dbl>
## 1 ALG1 - ... -0.240      -0.496      0.000976 0.787      0.607      0.999
## 2 ALG1 - ...  0.397       0.101       0.697      1.49       1.06      1.93
```

```

0.95
## 3 ALG1 - S 0.525      0.224    0.838    1.69     1.20     2.25
0.95
## 4 ALG2 - ... 0.633      0.372    0.919    1.88     1.41     2.45
0.95
## 5 ALG2 - S 0.757      0.476    1.05     2.13     1.61     2.85
0.95
## 6 NB - S   0.134      -0.212   0.448    1.14     0.803    1.56
0.95
## # ... with 2 more variables: .point <chr>, .interval <chr>

```

For each contrast (each of the unique categories in the contrasts variable), we tell R to calculate the medians of the probabilities.

The Fit column should be the same as what the summary() told us, the probabilities.

Let's now get only the probabilities of which categories are greater in general.

```

day.em %>% group_by(contrast) %>%
  summarize(P=sum(Fit>1)/n()) # >1 = greater in general

## # A tibble: 6 x 2
##   contrast          P
##   <fct>        <dbl>
## 1 ALG1 - ALG2  0.0312
## 2 ALG1 - NB   0.997
## 3 ALG1 - S    1.00
## 4 ALG2 - NB   1
## 5 ALG2 - S    1
## 6 NB - S     0.776

```

For each contrast, sum up the number of instances of fit that are greater than 1, divided by the total number of contrast.

- The prob of NB having more recruitment than S is 77%, which is not too much.
- The prob of ALG1 having more recruitment than NB, ALG1 being greater than S, and ALG2 being greater than NB is almost 100%
- The prob of ALG1 having more recruitment than ALG2 is 2.4%, which is nothing/very unlikely. BUT ALG2 has a prob of $100-2.4=97.6\%$ of having more recruitment than ALG1, which is a lot (!) Don't forget to check it from the other way round too (!)

Let's now get only the probabilities of which categories are greater than 10%

```

day.em %>% group_by(contrast) %>%
  summarize(P=sum(Fit>1.1)/n()) # >1.1 = greater than 10%

## # A tibble: 6 x 2
##   contrast          P
##   <fct>        <dbl>
## 1 ALG1 - ALG2  0.00458
## 2 ALG1 - NB   0.978

```

```
## 3 ALG1 - S    0.997
## 4 ALG2 - NB   1
## 5 ALG2 - S    1
## 6 NB - S     0.583
```

In this case: - ALG1 has a 0.5% prob of being greater than 10% in comparison to ALG2. So I guess we can say that ALG2 has $100-0.5=99.5$ prob of being 10% greater?

We can make any number of comparisons we like in Bayesian. It does not make the Bayesian analysis lose Power, which was the problem in frequentist.

Other combinations

We will now check the following combinations: ALG2vsALG1 NBvsS
 ALGcombivsBare(Bare=NB+S) AlgcombivsNB ALG1 -1 0 0.5 0.5 ALG2 1 0 0.5 0.5 S 0 1 -0.5 -1 NB 0 -1 -0.5 0

```
cmat<-cbind("Alg2_Alg1"=c(-1,1,0,0),
             "NB_S"=c(0,0,1,-1),
             "Alg_Bare"=c(0.5,0.5,-0.5,-0.5), # Alg=Alg1+Alg2; Bare=NB+S
             "Alg_NB"=c(0.5,0.5,-1,0)) # Alg=Alg1+Alg2

cmat

##          Alg2_Alg1 NB_S Alg_Bare Alg_NB
## [1,]      -1     0     0.5     0.5
## [2,]       1     0     0.5     0.5
## [3,]       0     1    -0.5    -1.0
## [4,]       0    -1    -0.5     0.0
```

Pairwise comparisons

```
emmeans(day.rstanarm3, ~TREAT, contr=list(TREAT=cmat), type="response")

## $emmeans
##  TREAT rate lower.HPD upper.HPD
##  ALG1  22.3     18.3     26.5
##  ALG2  28.3     23.9     33.0
##  NB    15.0     11.8     18.5
##  S     13.2     10.3     16.5
##
## Point estimate displayed: median
## Results are back-transformed from the log scale
## HPD interval probability: 0.95
##
## $contrasts
##  contrast      ratio lower.HPD upper.HPD
##  TREAT.Alg2_Alg1  1.27    0.981    1.61
##  TREAT.NB_S       1.14    0.803    1.56
##  TREAT.Alg_Bare   1.78    1.465    2.20
```

```
##   TREAT.Alg_NB      1.68      1.288      2.14
## 
## Point estimate displayed: median
## Results are back-transformed from the log scale
## HPD interval probability: 0.95
```

Let's check the \$contrasts: contrast ratio lower.HPD upper.HPD TREAT.Alg2_Alg1 1.27
 0.989 1.59 TREAT.NB_S 1.13 0.806 1.55 TREAT.Alg_Bare 1.79 1.438 2.16 TREAT.Alg_NB
 1.68 1.306 2.17 - Recruit is 27% higher on ALG2 than on the ALG1 surface. - Recruit is 13%
 higher on NB than on the S surface. - Recruit is 79% higher on algal than on the Bare
 surfaces. - Recruit is 68% higher on the algal surfaces than on the naturally bare.

Probabilities of the new combinations

We can now also check the probabilities of which groups are higher than what other groups

```
day.em=emmeans(day.rstanarm3, ~TREAT,
contr=list(TREAT=cmat), type="link")$contrasts %>%
  gather_emmeans_draws() %>% mutate(Fit=exp(.value))

day.em %>% group_by(contrast) %>%
  summarise(P=sum(Fit>1)/n())

## # A tibble: 4 x 2
##   contrast          P
##   <fct>        <dbl>
## 1 TREAT.Alg_Bare  1
## 2 TREAT.Alg_NB   1
## 3 TREAT.Alg2_Alg1 0.969
## 4 TREAT.NB_S     0.776
```

This shows us that... - Alg combi has a 100% prob of having more barnacle recruitment than Bare substrate (=NB+S) - Alg combi has a 100% prob of having more barnacle recruitment than NB substrate - Alg2 has a 98% prob of having more barnacle recruitment than Alg1 - NB has a 77% prob of having more barnacle recruitment than S

Reporting probabilities

Murray would say - "very strong": >95% - "strong": 80-95%

Plot

Plot 1- Caterpillar plot for our effects

We wanna plot the following:

```
day.em %>% group_by(contrast) %>% mean_hdi()
```

```
## # A tibble: 4 x 10
##   contrast .value .value.lower .value.upper   Fit Fit.lower Fit.upper
##   <fct>     <dbl>      <dbl>       <dbl> <dbl>    <dbl>      <dbl>
## 1 TREAT.A... 0.580      0.382       0.788  1.80    1.47      2.20
## 2 TREAT.A... 0.515      0.262       0.770  1.69    1.29      2.14
## 3 TREAT.A... 0.237     -0.000976    0.496  1.28    0.981     1.61
## 4 TREAT.N... 0.130     -0.212       0.448  1.16    0.803     1.56
## # ... with 2 more variables: .point <chr>, .interval <chr>
```

So let's store this table:

```
day.sum<-day.em %>%
  group_by(contrast) %>%
  mean_hdi(.width=c(0.8,0.95)) # here, we define the width the mean hdi
  (=hpd) (for plotting purposes, see next code chunk)
# Murray would rather pick MEDIAN than MEAN, so we'd should usually go for
# the MEDIAN (it is more robust)

day.sum

## # A tibble: 8 x 10
##   contrast .value .value.lower .value.upper   Fit Fit.lower Fit.upper
##   <fct>     <dbl>      <dbl>       <dbl> <dbl>    <dbl>      <dbl>
## 1 TREAT.A... 0.580      0.441       0.709  1.80    1.54      2.02
## 2 TREAT.A... 0.515      0.348       0.683  1.69    1.39      1.95
## 3 TREAT.A... 0.237      0.0725     0.394  1.28    1.06      1.46
## 4 TREAT.N... 0.130     -0.0733     0.371  1.16    0.855     1.36
## 5 TREAT.A... 0.580      0.382       0.788  1.80    1.47      2.20
## 6 TREAT.A... 0.515      0.262       0.770  1.69    1.29      2.14
## 7 TREAT.A... 0.237     -0.000976    0.496  1.28    0.981     1.61
## 8 TREAT.N... 0.130     -0.212       0.448  1.16    0.803     1.56
## # ... with 2 more variables: .point <chr>, .interval <chr>
```

Let's create the plot

```

g1<-ggplot(day.sum)+  

  geom_hline(yintercept=1,linetype="dashed") +  

  geom_pointrange(aes(x=contrast,y=Fit,ymin=Fit.lower,ymax=Fit.upper,  

    size=factor(.width)), # here, we defined the width (see  

  previous code chunk)  

    show.legend=FALSE)+  

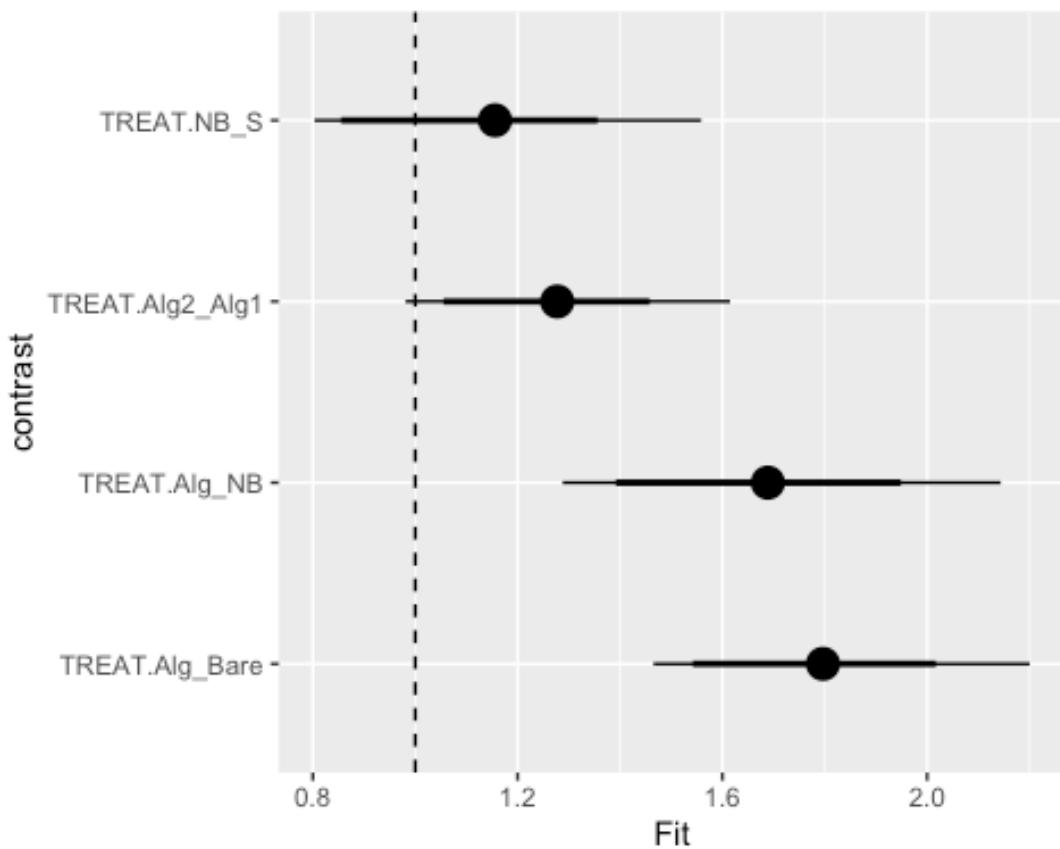
  scale_size_manual(values=c(1,0.5)) # defines the size of the points and  

  the conf int  

  coord_flip()

```

g1



```
g1<-ggplot
```

Plot 2 - ...

First, the predicted values

```

newdata=emmeans(day.rstanarm3, ~TREAT, type="response") %>% # type response  

back_transforms our data  

as.data.frame

```

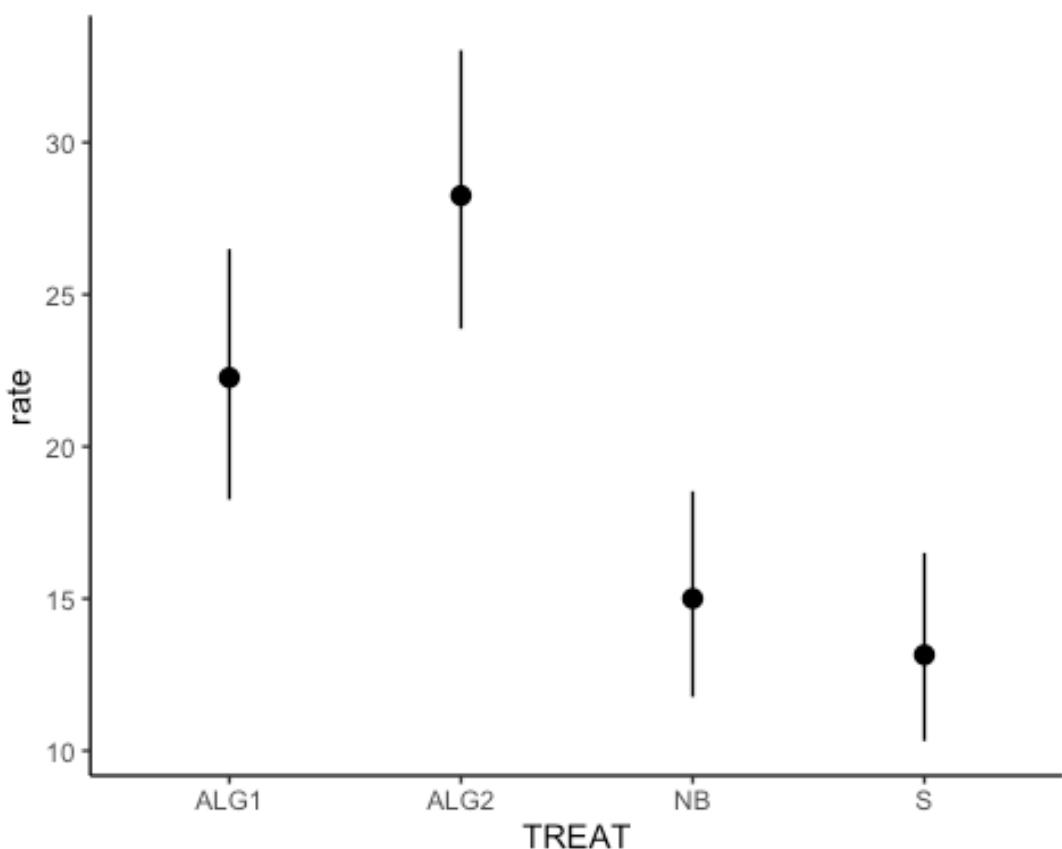
```
newdata
```

```
##   TREAT      rate lower.HPD upper.HPD
## 1 ALG1 22.26887 18.25505 26.49577
## 2 ALG2 28.25195 23.87335 33.03517
## 3   NB 15.00332 11.76615 18.53115
## 4     S 13.15268 10.31672 16.50679
```

Now, the plot

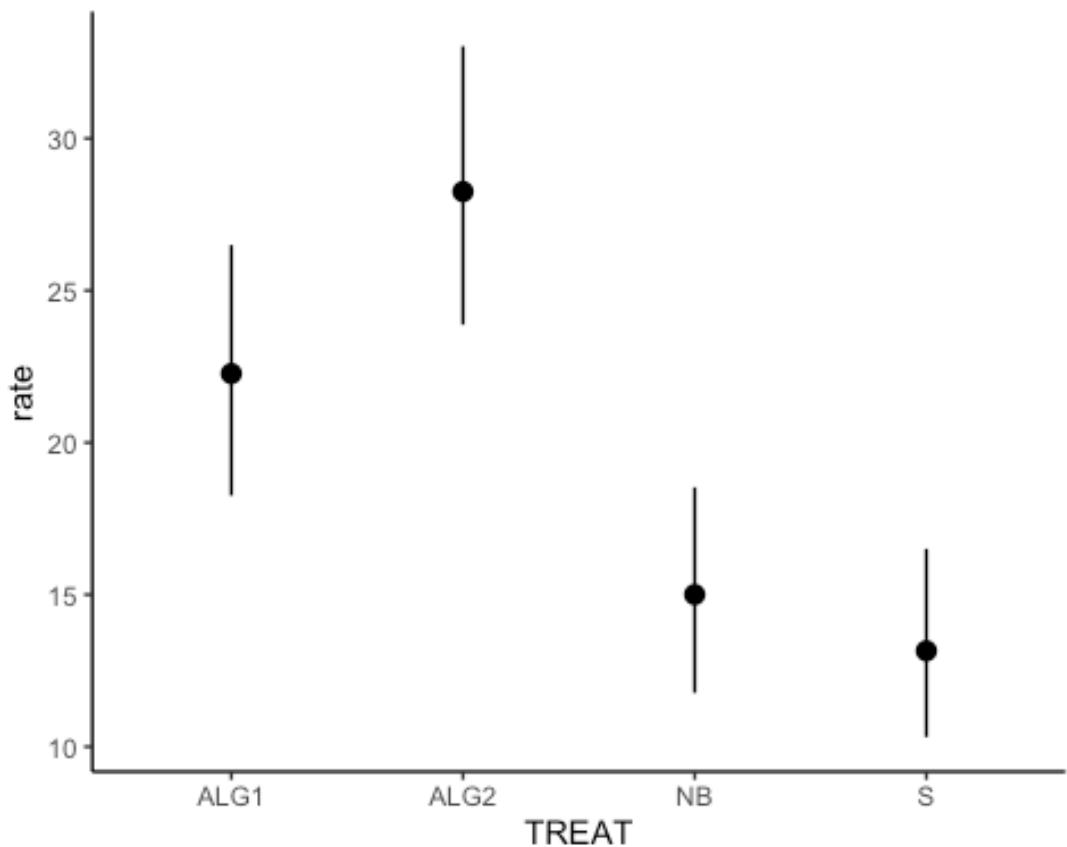
```
g2<-ggplot(newdata, aes(y=rate, x=TREAT))+
  geom_pointrange(aes(ymin=lower.HPD,ymax=upper.HPD))+  
  theme_classic()
```

g2



Displaying both plots

```
g2+(g1+theme_classic())
```



IGNORE BELOW

Partial effects plots

Model validation

Model investigation

Further investigations

Planned contrasts

Define your own

Compare:

- a) ALG1 vs ALG2
- b) NB vs S
- c) average of ALG1+ALG2 vs NB+S

Summary Figure

```

day.form <- bf(BARNACLE ~ TREAT, family=poisson(link='log'))
get_prior(day.form, data=day)
day.priors <- c(
  prior(normal(0, 10), class='Intercept'),
  prior(normal(0, 2.5), class='b')
)
day.brms <- brm(day.form, data=day,
                  prior=day.priors,
                  chains=3, iter=5000, warmup=2000, thin=5,
                  refresh=0)

plot(day.brms)
mcmc_plot(day.brms, type='acf_bar')
mcmc_plot(day.brms, type='rhat_hist')
mcmc_plot(day.brms, type='neff_hist')

preds <- posterior_predict(day.brms, nsamples=250, summary=FALSE)
day.resids <- createDHARMA(simulatedResponse = t(preds),
                            observedResponse = day$BARNACLE,
                            fittedPredictedResponse = apply(preds, 2, median),
                            integerResponse = TRUE)
plot(day.resids)

#pp_check(day.brmsP,
x=as.numeric(day$TREAT),'intervals')

ggpredict(day.brms, term='TREAT') %>% plot
ggpredict(day.brms, ~TREAT) %>% plot
ggemmeans(day.brms, ~TREAT) %>% plot

summary(day.brms)

tidyMCMC(day.brms$fit, conf.int=TRUE,
          conf.method='HPDinterval', rhat=TRUE, ess=TRUE)

# Pairwise comparisons
library(emmeans)
## factor statements
emmeans(day.brms, pairwise~TREAT, type='response')
## what about probabilities
day.em = emmeans(day.brms, pairwise~TREAT, type='link')$contrasts %>%

```

```

gather_emmeans_draws() %>%
  mutate(Fit=exp(.value))
day.em %>% head
day.em %>% group_by(contrast) %>%
  ggplot(aes(x=Fit)) +
  geom_histogram() +
  geom_vline(xintercept=1, color='red') +
  facet_wrap(~contrast, scales='free')
day.em %>% group_by(contrast) %>% median_hdi(.width=c(0.8, 0.95))

day.sum <- day.em %>%
  group_by(contrast) %>%
  median_hdci(.width=c(0.8, 0.95))
day.sum
ggplot(day.sum) +
  geom_hline(yintercept=1, linetype='dashed') +
  geom_pointrange(aes(x=contrast, y=Fit, ymin=Fit.lower, ymax=Fit.upper,
size=factor(.width)),
                  show.legend = FALSE) +
  scale_size_manual(values=c(1, 0.5)) +
  coord_flip()

g1 <- ggplot(day.sum) +
  geom_hline(yintercept=1) +
  geom_pointrange(aes(x=contrast, y=Fit, ymin=Fit.lower, ymax=Fit.upper,
size=factor(.width)), show.legend = FALSE) +
  scale_size_manual(values=c(1, 0.5)) +
  scale_y_continuous(trans=scales::log2_trans(), breaks=c(0.5, 1, 2, 4)) +
  coord_flip()
g1
# Probability of effect
day.em %>% group_by(contrast) %>% summarize(P=sum(.value>0)/n())
day.em %>% group_by(contrast) %>% summarize(P=sum(Fit>1)/n())
##Probability of effect greater than 10%
day.em %>% group_by(contrast) %>% summarize(P=sum(Fit>1.1)/n())

##Planned contrasts
cmat<-cbind('Alg2_Alg1'=c(-1,1,0,0),
             'NB_S'=c(0,0,1,-1),
             'Alg_Bare'=c(0.5,0.5,-0.5,-0.5),
             'Alg_NB'=c(0.5,0.5,-1,0))
#crossprod(cmat)
emmeans(day.brms, ~TREAT, contr=list(TREAT=cmat), type='link')
emmeans(day.brms, ~TREAT, contr=list(TREAT=cmat), type='response')
day.em = emmeans(day.brms, ~TREAT, contr=list(TREAT=cmat),
type='link')$contrasts %>%
  gather_emmeans_draws() %>%
  mutate(Fit=exp(.value))
day.em %>% group_by(contrast) %>% median_hdci()
# Probability of effect

```

```
day.em %>% group_by(contrast) %>% summarize(P=sum(Fit>1)/n())
##Probability of effect greater than 10%
day.em %>% group_by(contrast) %>% summarize(P=sum(Fit>1.1)/n())

hist(bayes_R2(day.brms, summary=FALSE))

bayes_R2(day.brms, summary=FALSE) %>% median_hdi

## Summary plot
day.grid = with(day, list(TREAT=levels(TREAT)))
newdata = emmeans(day.brms, ~TREAT, type='response') %>% as.data.frame
head(newdata)
g2 <- ggplot(newdata, aes(y=rate, x=TREAT)) +
  geom_pointrange(aes(ymin=lower.HPD, ymax=upper.HPD))

library(patchwork)
g1 + g2
```

References

18_Bayesian_Poisson

Sara Kophamel

16/12/2020

Preparations

Load the necessary libraries

```
library(rstanarm)    #for fitting models in STAN
library(brms)        #for fitting models in STAN
library(coda)         #for diagnostics
library(bayesplot)   #for diagnostics
library(rstan)        #for interfacing with STAN
library(DHARMA)       #for residual diagnostics
library(emmeans)      #for marginal means etc
library(broom)         #for tidying outputs
library(broom.mixed)  #for tidying outputs
library(tidybayes)    #for more tidying outputs
library(ggeffects)    #for partial plots
library(tidyverse)     #for data wrangling etc
```

Scenario

An ecologist studying a rocky shore at Phillip Island, in southeastern Australia, was interested in how clumps of intertidal mussels are maintained [@Quinn-1988-137]. In particular, he wanted to know how densities of adult mussels affected recruitment of young individuals from the plankton. As with most marine invertebrates, recruitment is highly patchy in time, so he expected to find seasonal variation, and the interaction between season and density - whether effects of adult mussel density vary across seasons - was the aspect of most interest.

The data were collected from FOUR SEASONS, and with TWO DENSITIES of adult mussels (high and low). The experiment consisted of clumps of adult mussels attached to the rocks. These clumps were then brought back to the laboratory, and the number of baby mussels recorded. There were 3-6 replicate clumps for each density and season combination.

We want to check at diff according to SEASONS, and DENSITIES.

Format of quinn.csv data files

SEASON	DENSITY	RECRUITS	SQRTRECRUITS	GROUP
Spring	Low	15	3.87	SpringLow

..
Spring	High	11	3.32	SpringHigh
..
Summer	Low	21	4.58	SummerLow
..
Summer	High	34	5.83	SummerHigh
..
Autumn	Low	14	3.74	AutumnLow
..
SEASON	Categorical listing of Season in which mussel clumps were collected independent variable			
DENSITY	Categorical listing of the density of mussels within mussel clump independent variable			
RECRUITS	The number of mussel recruits response variable			
SQRTRECRUITS	Square root transformation of RECRUITS - needed to meet the test assumptions			
GROUPS	Categorical listing of Season/Density combinations - used for checking ANOVA assumptions			



Mussel

Read in the data

```
quinn = read_csv('data/quinn.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   SEASON = col_character(),
##   DENSITY = col_character(),
##   RECRUITS = col_double(),
##   SQRTRECRUITS = col_double(),
##   GROUP = col_character()
## )

glimpse(quinn)

## Rows: 42
## Columns: 5
## $ SEASON      <chr> "Spring", "Spring", "Spring", "Spring", "Spring",
## "Sprin...
## $ DENSITY     <chr> "Low", "Low", "Low", "Low", "Low", "High", "High",
## "High...
## $ RECRUITS    <dbl> 15, 10, 13, 13, 5, 11, 10, 15, 10, 13, 1, 21, 31, 21,
## 18...
## $ SQRTRECRUITS <dbl> 3.872983, 3.162278, 3.605551, 3.605551, 2.236068,
## 3.3166...
## $ GROUP       <chr> "SpringLow", "SpringLow", "SpringLow", "SpringLow",
## "Spr...

summary(quinn)
```

```

##      SEASON          DENSITY        RECRUITS      SQRTRECRUITS
##  Length:42          Length:42       Min.   : 0.00    Min.   :0.000
##  Class  :character  Class  :character  1st Qu.: 9.25    1st Qu.:3.041
##  Mode   :character  Mode   :character  Median  :13.50    Median  :3.674
##                                         Mean   :18.33    Mean   :3.871
##                                         3rd Qu.:21.75    3rd Qu.:4.663
##                                         Max.   :69.00    Max.   :8.307
##      GROUP
##  Length:42
##  Class  :character
##  Mode   :character
## 
## 
## 
```

We are working with COUNT data, so POISSON might be a good choice.

Let's define SEASON (chr) and DENSITY (chr) as factors

```
quinn = quinn %>% mutate(SEASON = factor(SEASON),
                           DENSITY = factor(DENSITY))
```

Ordering seasons

```
quinn = quinn %>% mutate(SEASON = factor(SEASON, levels=c('Spring', 'Summer',
                           'Autumn', 'Winter')),
                           DENSITY = factor(DENSITY))
```

Exploratory data analysis

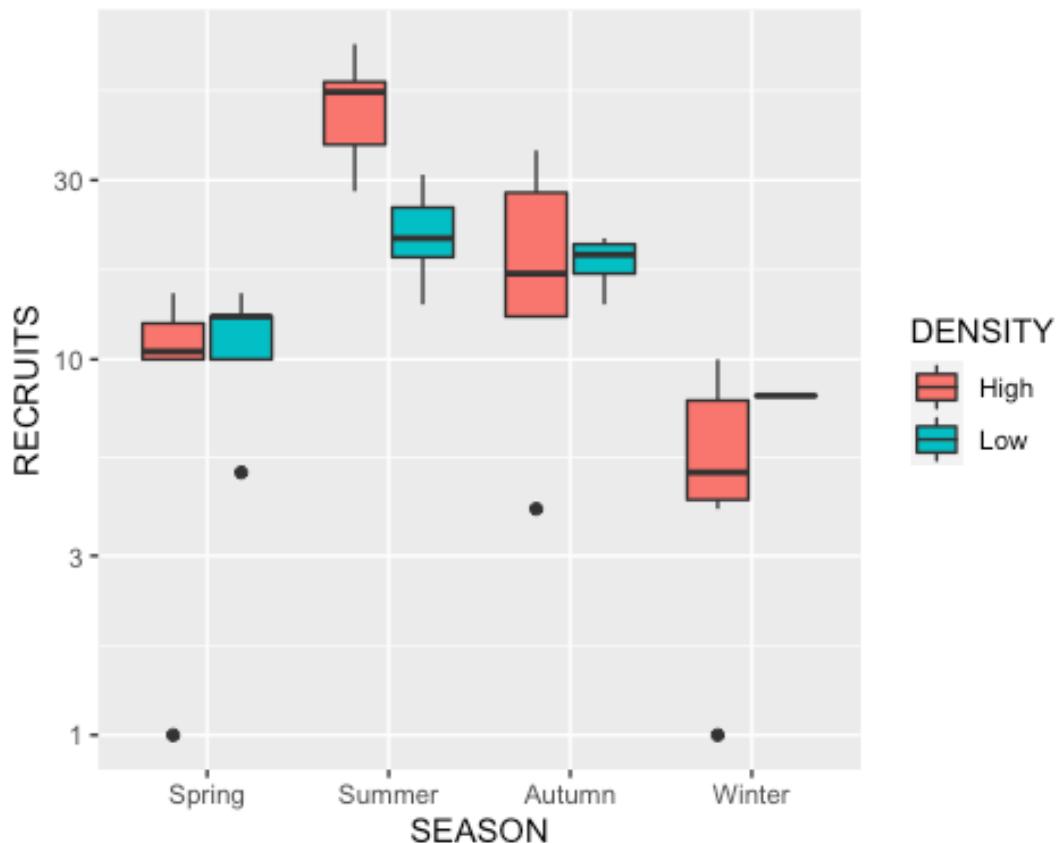
Model formula:

$$\begin{aligned} \text{y}_i &\sim \mathcal{N}(\lambda_i, \theta) \\ \boldsymbol{\beta} \cdot \mathbf{X}_i + \beta_0 &\sim N(0, 10) \\ \beta_{1,2,3} &\sim N(0, 2.5) \end{aligned}$$

where β is a vector of effects parameters and \mathbf{X} is a model matrix representing the intercept and effects of season, density and their interaction on mussel recruitment.

```
ggplot(quinn, aes(y=RECRUITS, x=SEASON, fill=DENSITY)) +
  geom_boxplot() +
  scale_y_log10()

## Warning: Transformation introduced infinite values in continuous y-axis
## Warning: Removed 2 rows containing non-finite values (stat_boxplot).
```



Conclusions:

- there is clear evidence of non-homogeneity of variance
- specifically, there is evidence that the variance is related to the mean in that boxplots that are lower on the y-axis (low mean) also have lower variance (shorter boxplots)
- this might be expected for count data and we might consider that a Poisson distribution (which assumes that mean and variance are equal - and thus related in a very specific way).

We will use a log scaled response (y axis)

Initial model - only priors

Default priors

1- Prior selection

For stan models (NUTS models), 5000 iterations should be enough to settle itself

```
quinn.rstanarm=stan_glm(RECRUITS~SEASON*DENSITY,data=quinn,
                         family=poisson(link="log"),
                         iter=5000,
                         warmup=1000,
```

```

            chains=3,
            thin=5, # we are only gonna keep the 5th value
            refresh=0)

quinn.rstanarm

## stan_glm
## family:      poisson [log]
## formula:     RECRUITS ~ SEASON * DENSITY
## observations: 42
## predictors:  8
## -----
##                               Median MAD_SD
## (Intercept)                2.3   0.1
## SEASONSummer               1.6   0.1
## SEASONAutumn                0.7   0.2
## SEASONWinter                -0.6  0.2
## DENSITYLow                  0.1   0.2
## SEASONSummer:DENSITYLow   -0.9  0.2
## SEASONAutumn:DENSITYLow   -0.2  0.2
## SEASONWinter:DENSITYLow   -0.9  0.4
##
## -----
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg

```

2- Check the default priors that were applied (they used the Adjusted prior within Intercept)

`prior_summary(quinn.rstanarm)`

```

## Priors for model 'quinn.rstanarm'
## -----
## Intercept (after predictors centered)
## ~ normal(location = 0, scale = 2.5)
##
## Coefficients
##   Specified prior:
##     ~ normal(location = [0,0,0,...], scale = [2.5,2.5,2.5,...])
##   Adjusted prior:
##     ~ normal(location = [0,0,0,...], scale = [5.47,5.80,6.02,...])
## -----
## See help('prior_summary.stanreg') for more details

```

The priors chosen were the following: Intercept (after predictors centered) ~ `normal(location = 0, scale = 2.5)`

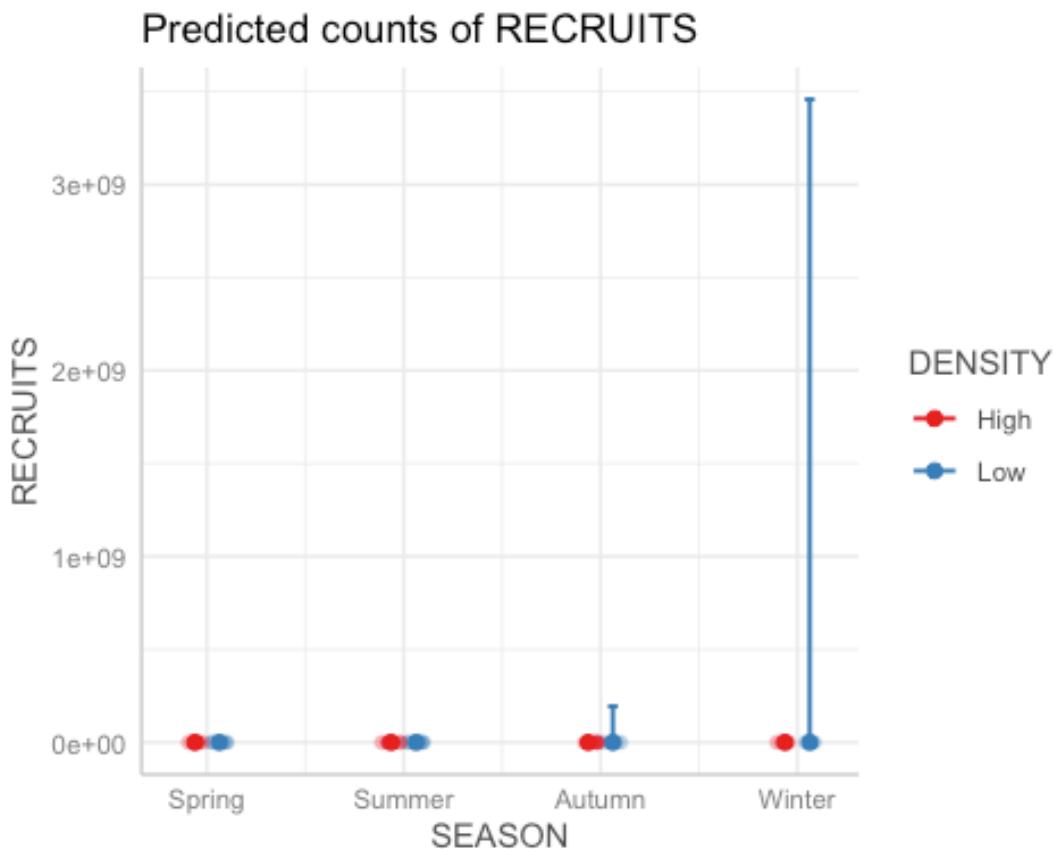
Coefficients Specified prior: `~ normal(location = [0,0,0,...], scale = [2.5,2.5,2.5,...])` Adjusted prior: `~ normal(location = [0,0,0,...], scale = [5.62,5.47,6.02,...])`

3- Run the model again, only including the prior

We are trying to make sure that the priors we are setting are sensible to the posteriors resulting from the data. You want the priors to be vague, but not too vague.

We tell R to show us only the model, WITHOUT the y axis (it includes x)

```
quinn.rstanarm1<-update(quinn.rstanarm,prior_PD=TRUE)  
ggpredict(quinn.rstanarm1, ~SEASON+DENSITY) %>% plot(add.data=TRUE) # We tell  
R to display SEASON AND DENSITY together (if we did not specify this, we'd  
get two separate predictive plots)
```



We got 8

interaction terms

(springH,springL,summerH,summerL,autumnH,autumnL,winterH,winterL)

The default priors are HORRIBLE... So...

Fit the model

Decide on the parameters for the model

We have to decide: - the priors (let's not care about the default ones) - number of iterations: let's pick 5000 as usual - warmup: let's pick 1000 as usual (we could also pick 2000, does not really matter, as it usually settles even before 1000) - chains: let's pick 3 as usual (3-4, does not matter) - thinner: let's pick 5 as usual (good starting point always)

4- Fit our own priors

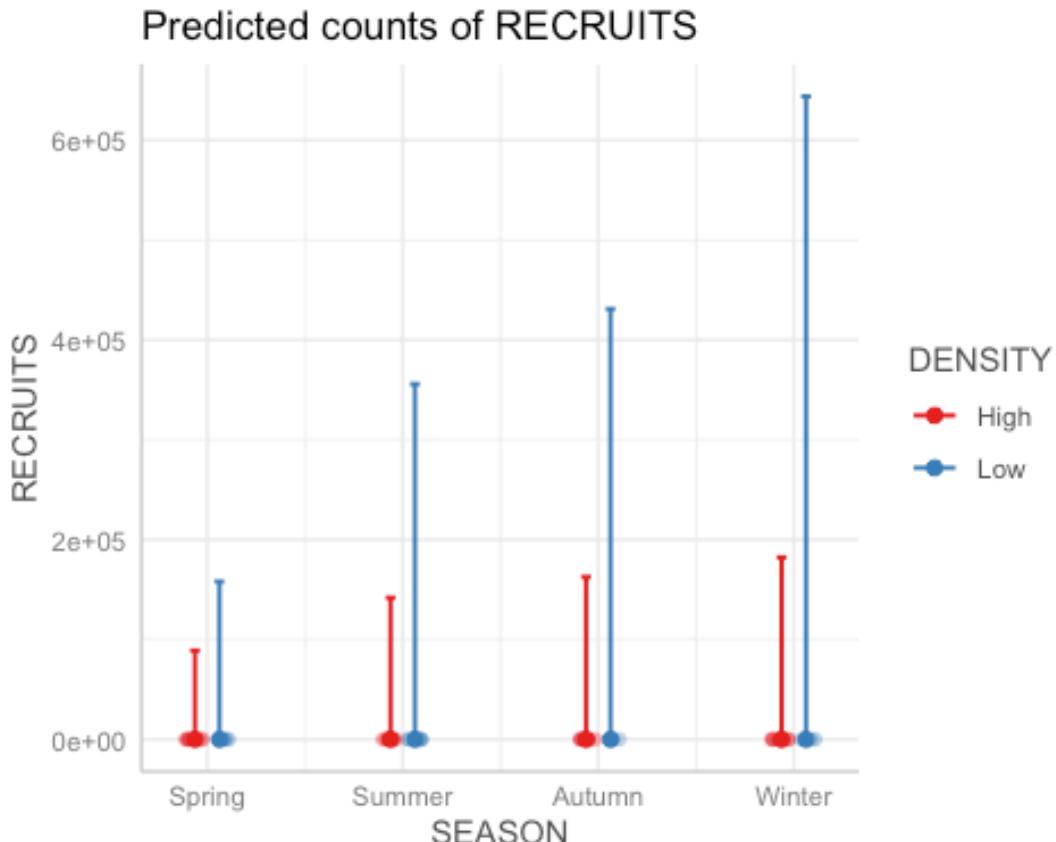
We can come up with them any way we like. Murray will make them a bit narrower than the ones we chose. We could also define them as just pos values if we got no - values in our data, for example.

We first create the model as if we were only using the priors, not the actual data (see step 5. for final model)

```
quinn.rstanarm2=stan_glm(RECRUITS~SEASON*DENSITY, data=quinn, # Here, I have
defined an interaction btw SEASON and DENSITY (as evidenced by the plot).
However, we could also have tried an ADDITIVE model
  family=poisson(link="log"),
  prior_intercept=normal(2.3,4.6,autoscale=FALSE),
  # 2.3 = log(10) --> this is the approx value of the
first group on the x axis (see initial plot)
  # 4.6 for variance --> Approximate Log(orders of
magnitude) --> Let's say we got a max value of ~70, so ~log(70)=4.6. It would
also be ok if we changed this value, cause it's just a rough idea of what
could be the max value
  prior=normal(0,2,autoscale=FALSE), # default values
  # autoscale=FALSE as we already considered the priors
we want; we don't have to scale them to our data
  # 0 = We don't wanna define a pos or neg interaction
(we don't know how the response is gonna change), so we set this value to
zero
  # 2 = we give the priors enough space to develop
  prior_PD=TRUE, # only the priors are involved, as we
wanna see whether our priors look vaguely sensible (is the scheme not too
wide and not too narrow?). Because we Log transformed our data, we will have
to remember to backtransform the priors later!
  iter=5000,
  warmup=1000,
  chains=3,
  thin=5,
  refresh=0)
```

Let's have a quick look at our model

```
ggpredict(quinn.rstanarm2, ~SEASON+DENSITY) %>% # We tell R to plot the
predictions together
plot(add.data=TRUE)
```



We see that our priors are REALLY wide. If we had a problem with our model, we would narrow down our priors. This involves the risk of the sampler having trouble sampling in the right location (it might get lost sampling in the wrong location). However, we better start off with wide priors, and then narrow them down; otherwise we would be creating a model which may not reflect the data.

Model to form the posterior priors - Fit the ACTUAL model

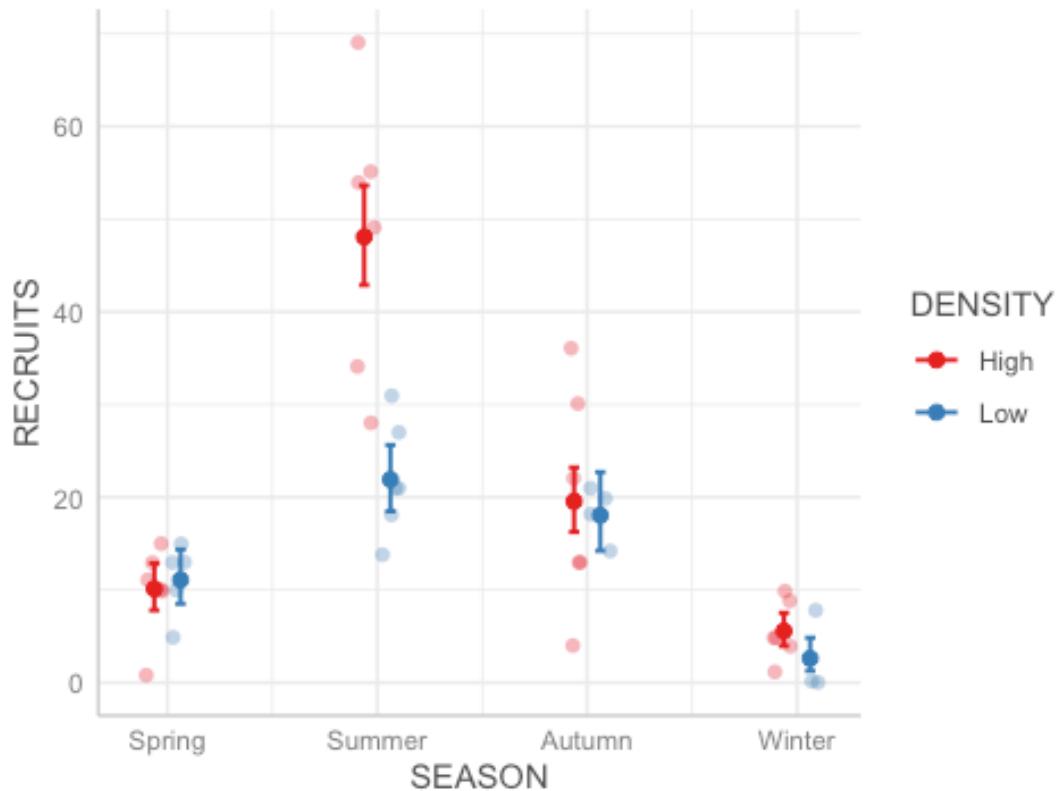
The likelihood / the data now informs about the posteriors. In the previous model, we had set the priors = posteriors.

Now that we have checked the priors of rstanarm2, and are happy with priors selected, we tell R that we want to include the data as well (and not only the prior). We therefore update the model, and define prior_PD=FALSE

```
quinn.rstanarm3=update(quinn.rstanarm2, prior_PD=FALSE)

ggpredict(quinn.rstanarm3, ~SEASON+DENSITY) %>% plot(add.data=TRUE)
```

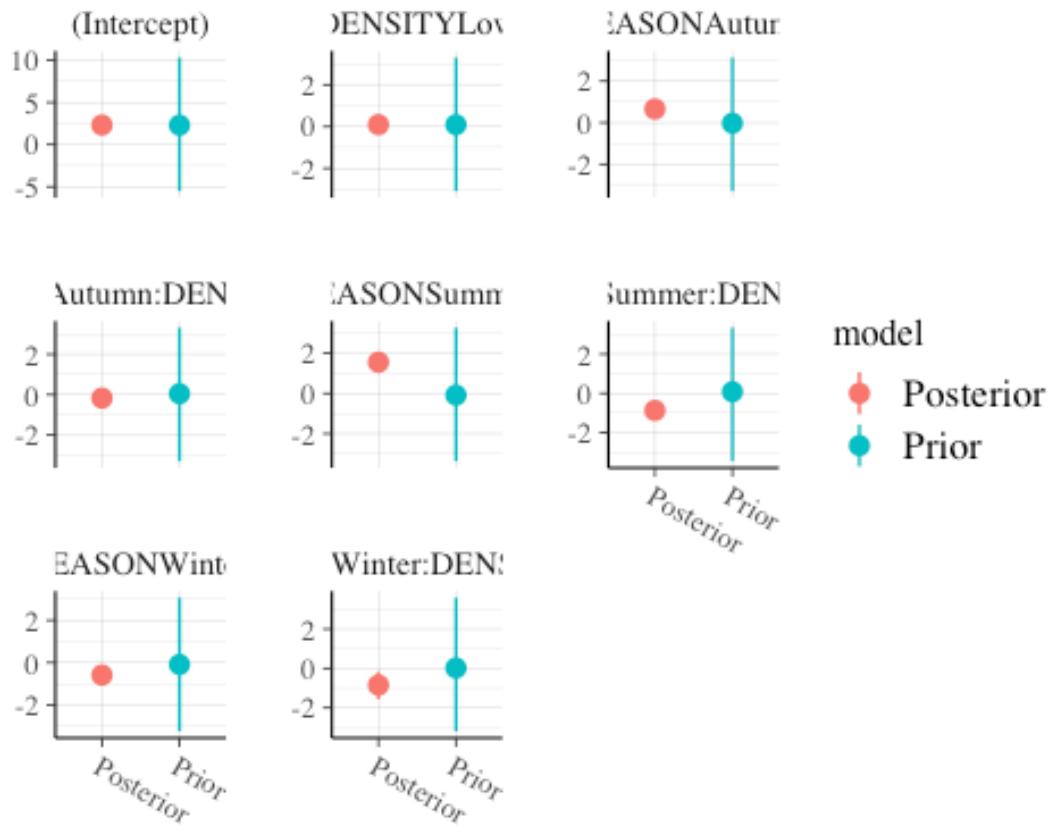
Predicted counts of RECRUITS



###

Prior vs posterior check - Checking how well our priors fit the actual data

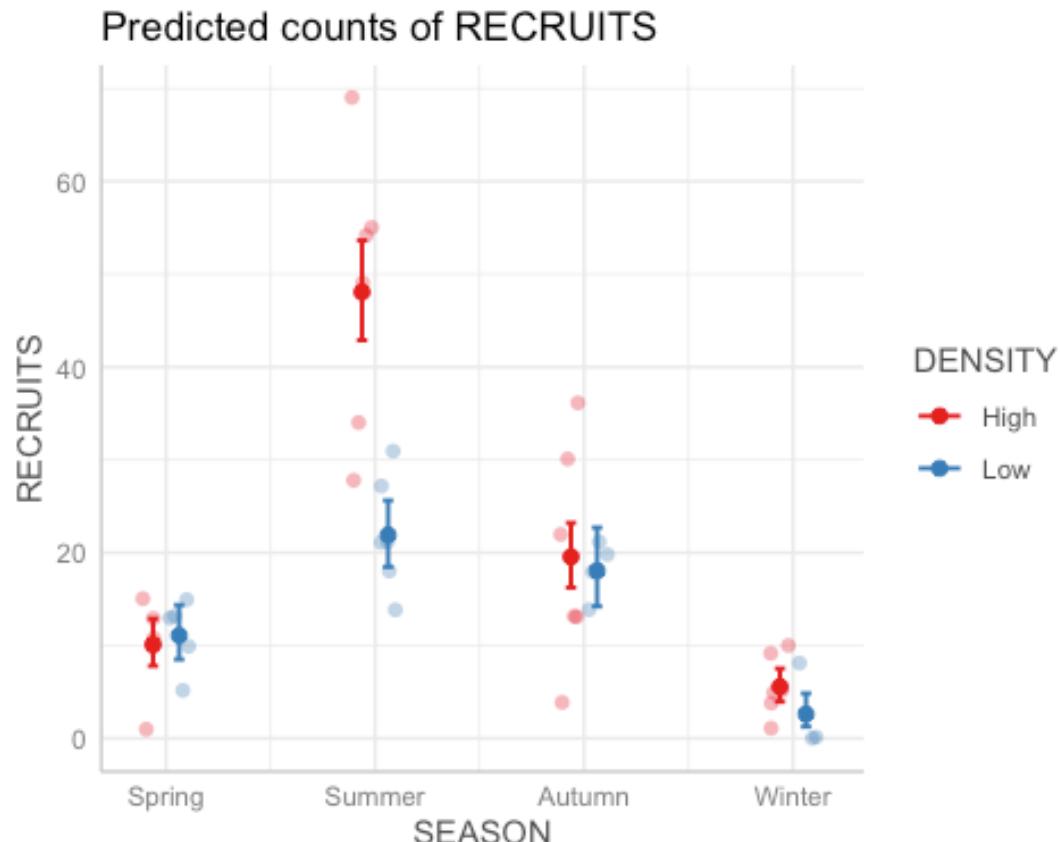
```
posterior_vs_prior(quinn.rstanarm3,color_by="vs",group_by=TRUE,  
                    facet_args=list(scales="free_y"))  
  
##  
## Drawing from prior...
```



Posterior and prior are quite diff from one another, so that's good

6. Predict the data

```
ggpredict(quinn.rstanarm3, ~SEASON+DENSITY) %>% plot(add.data=TRUE)
```



estimates look reasonable. Winter has some zeros

This looks pretty much like the real data, so it seems like our priors were great.

The

8. Model validation

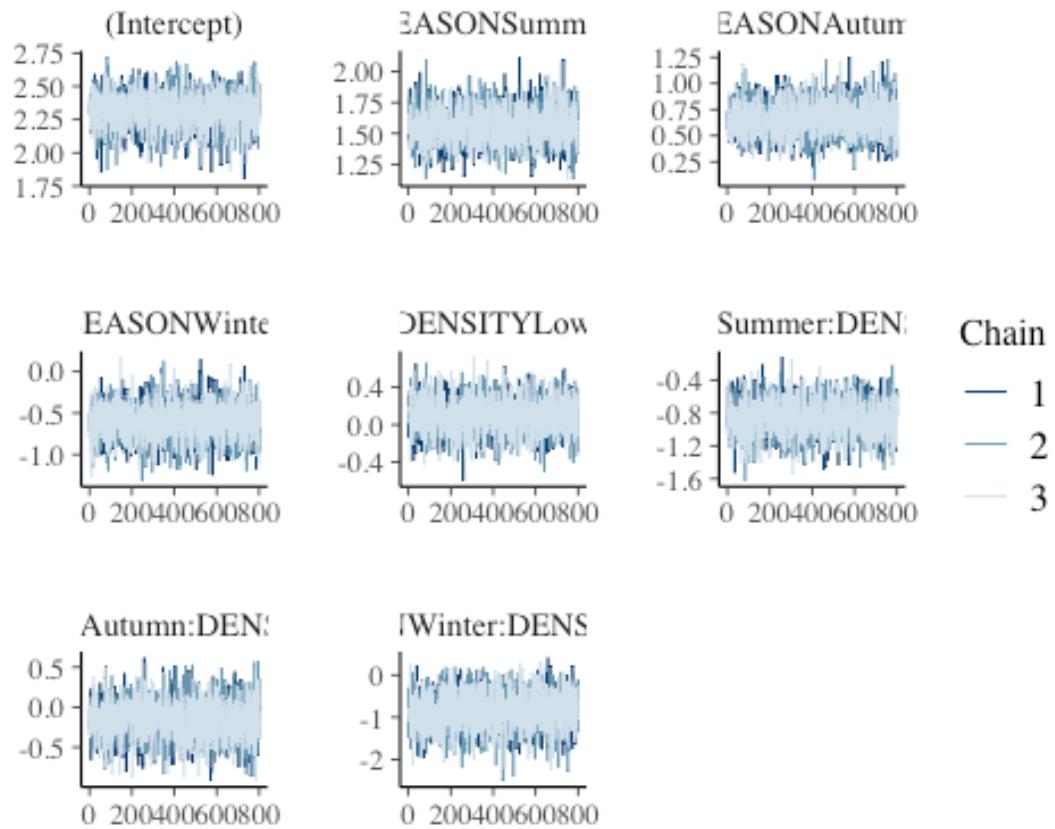
WE NOW CONDUCT SAMPLING DIAGNOSTICS:

1. MCMC sampling diagnostics

We want to know: - is our thinning ok? - have the chains converged after 3x iteration? etc

1. Trace plots

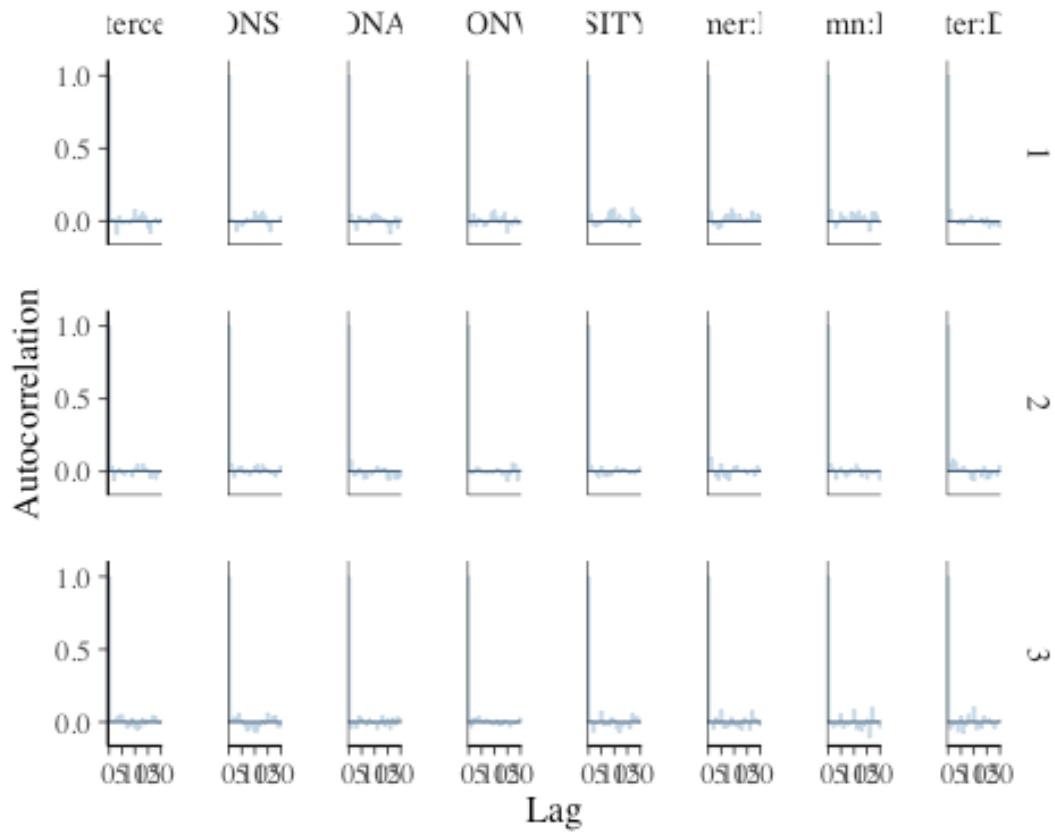
```
plot(quinn.rstanarm3, plotfun="mcmc_trace")
```



2.

Autocorrelation function for thinning

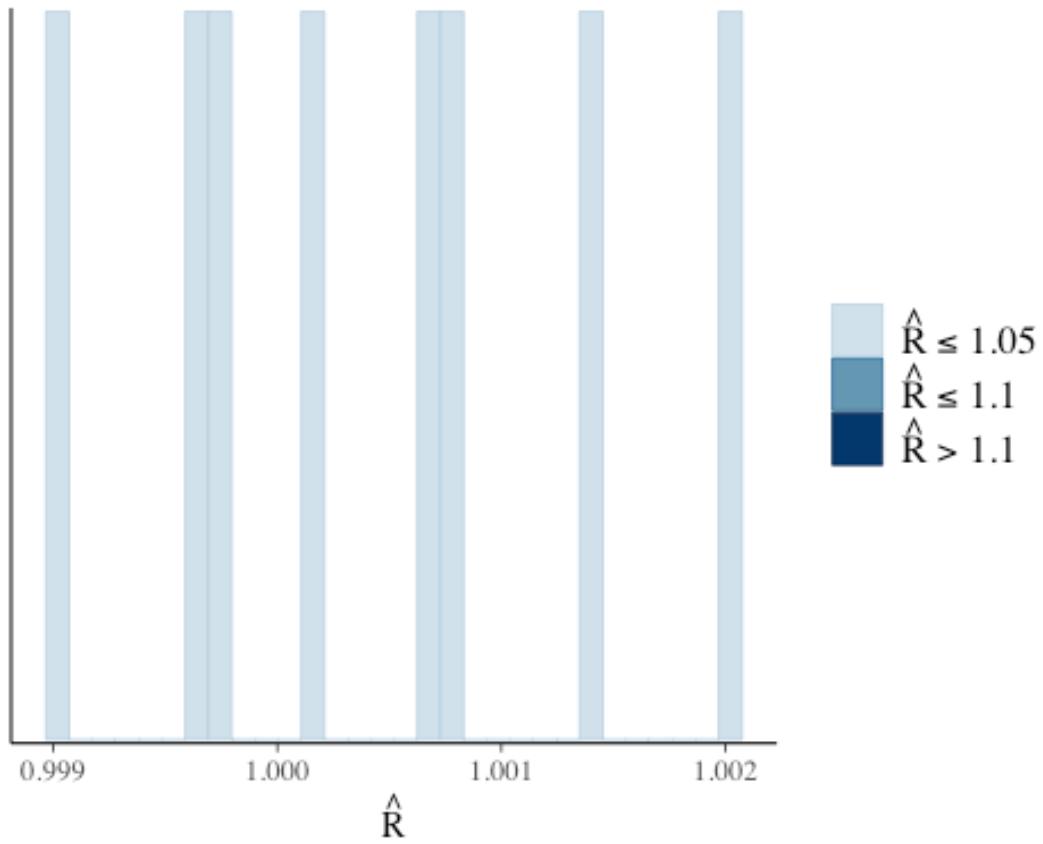
```
plot(quinn.rstanarm3, plotfun="acf_bar")
```



We don't wanna see big bars after the first one. In this case, we don't want to thin any further. 5 is adequate (this number is usually enough for all models).

3. Measure of chain convergence - R hat

```
plot(quinn.rstanarm3, "rhat_hist")
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We

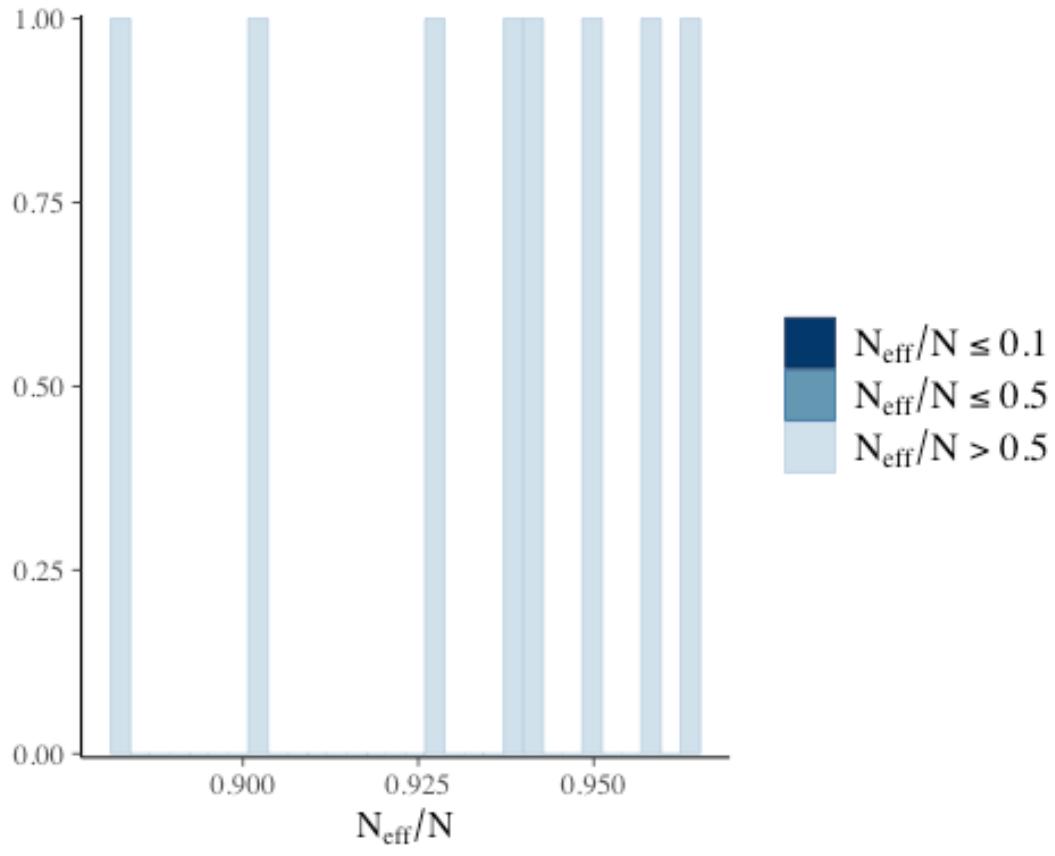
want all values be <1.05 . This is the threshold above which they are not converged. This graph colors things in that are <1.05 , btw $1.05-1.1$, and >1.1 . But anything >1.05 = chains haven't converged.

If the chains haven't converged, we have to:

- increase the number of iterations (perhaps one sample focused on only one area)
- narrow the priors a bit more (as one might have been too wide and got stuck in a specific area) \rightarrow This is usually not the case if you supply your own priors, but could happen if you use the default R priors.
- let the priors run for longer (might not be an option as this takes a LOONG time)

4. Number of effective samples

```
plot(quinn.rstanarm3, "neff_hist")
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



You

want the values to be close to 1 (=100% effective). If you have a large number of effective samples, the number of samples that are rejected might be low, but if you got a low number of effective samples, the model might get lost sampling somewhere else.

If we have values far away from 1, the solution is to narrow the priors.

2. IGNORE THIS SECTION (Murray did not discuss these plots) - PPC plots - Posterior probability checks

These plots check if our model are close to reality.

1. Density overlay plot

```
# pp_check(quinn.rstanarm3, plotfun="dens_overLay")
```

2. Error scatterplot

```
# pp_check(quinn.rstanarm3, plotfun="error_scatter_avg")
```

You don't want to see a plot where points are very far away from other points = That point would have a very big error. We would have been very bad at predicting that point. And possibly, that point was an outlier. This plot is analogous to a residual plot.

3. Predictive error plotted against each response

```
# pp_check(quinn.rstanarm3, x=quinn$GROUP, plotfun="error_scatter_avg_vs_x") #  
# not sure if this is right
```

4. Raw data vs pred range of responses.

```
# pp_check(quinn.rstanarm3, x=quinn$GROUP, plotfun="intervals") # not sure  
# about this one
```

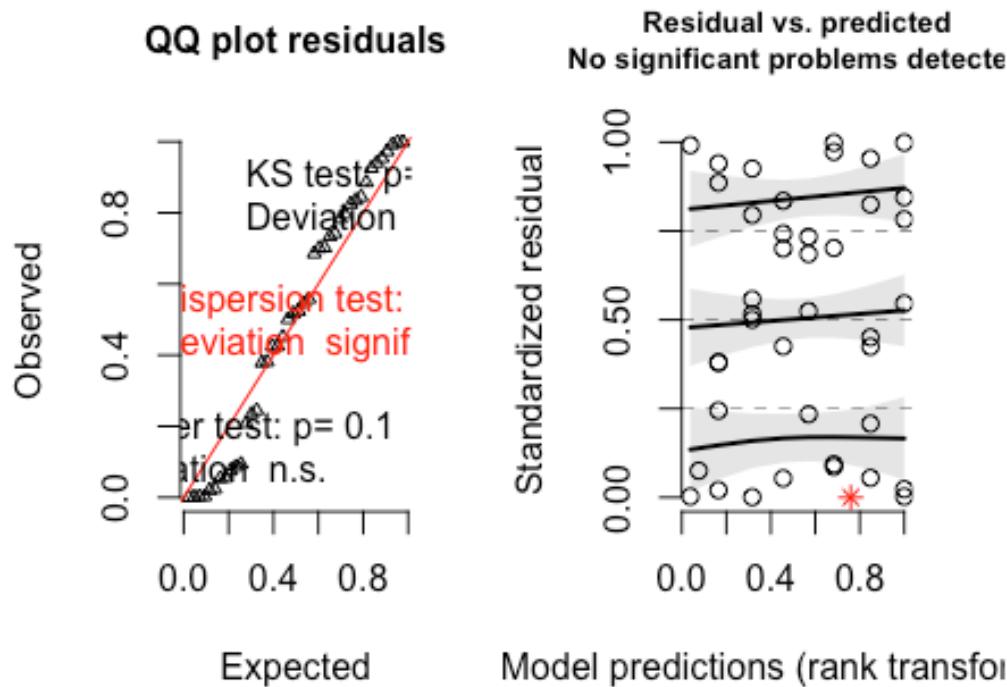
The predictions are similar to the raw data, which is what we wanna see. It is an extension to the first plot of this series. We don't want to see black dots near the conf bands.

The predictions (*yrep*) are about the same as the raw data (*y*).

3. DHARMA residuals

```
preds<-posterior_predict(quinn.rstanarm3, nsamples=250, summary=FALSE) # We  
# tell R to give us 250 predictions to do simulations (standard number)  
  
quinn.resids<-createDHARMA(simulatedResponse=t(preds), # predicted resp (best  
# fit values)  
# observedResponse=quinn$RECRUITS, # this is our raw  
# response values, the YIELD in the example (this is the only thing we'd change  
# when using other datasets)  
# fittedPredictedResponse=apply(preds, 2, median),  
# We tell R to take the predicted values, and calculate the median for each  
# integerResponse="gaussian")  
  
plot(quinn.resids)
```

DHARMA residual diagnostics



DHARMA residuals are bad.

We will have to change our model to a NEG BINOMIAL \rightarrow Dispersion becomes a parameter in a neg binomial; it is estimated. Instead of assuming dispersion=1, we will actually test if dispersion=1. Dispersion CAN'T be <0. So "normal" is NOT a useful prior for dispersion. A sensible prior would be an EXPONENTIAL Another option would be a GAMMA prior.

Interestingly enough, for sigma (variance) and for dispersion, gamma are the canonical priors, supposed to work best. BUT this is NOT the case. Even if in theory they should. That is why you usually don't see gamma priors. WE WILL THEREFORE USE AN EXPONENTIAL PRIOR.

First, let's check what package uses exponential. Type in the console "exponential". We get following code:

```
exponential function (link = "log") { slink <- substitute(link) .brmsfamily("exponential", link = link, slink = slink) }
<bytecode: 0x7fabd05b4400> <environment: namespace:brms>
```

`brmsfamily("exponential", link = link, slink = slink)` \rightarrow This tells us its brms package which defines the exponential function. We want rstanarm to define it, so we have to define this in the model:

Create the new binomial model (!)

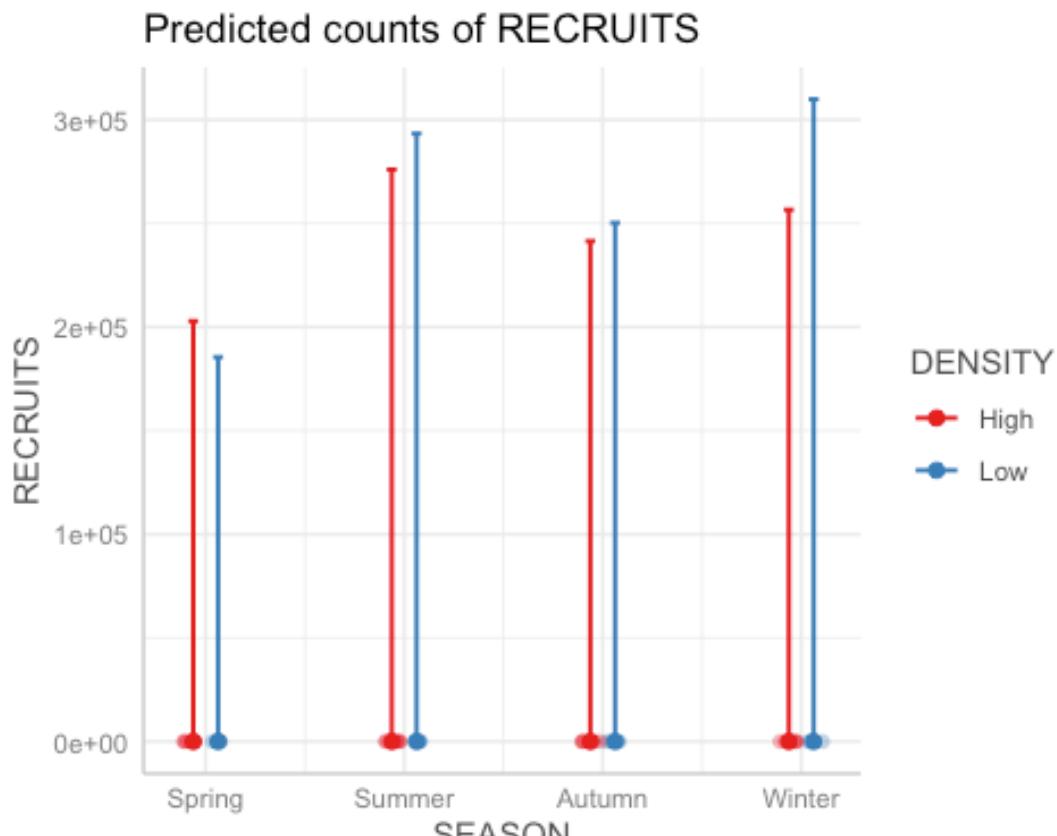
```
quinn.rstanarmNB<-stan_glm(RECRUITS~SEASON*DENSITY, data=quinn,
                               family=neg_binomial_2(link="log"), refresh=0,
                               prior_intercept=normal(2.3,5,autoscale=FALSE),
                               prior=normal(0,1,autoscale=FALSE), # don't forget
to set this line, even though we are using an exponential model
# 2.3 = Log of mean value of first group in
boxplots above
# 5 = Log of max value (could be anything, really;
in this case Murray has taken a value of 150 --> log(150)=5; and exp(5)=150)

prior_aux=rstanarm::exponential(1,autoscale=FALSE),#exponential prior
                                prior_PD=TRUE,
                                chains=3,iter=5000,thin=5,warmup=2000) # default
(warmup 1000 would also have been ok)
```

- neg binomial 2 is the second way to parametrizing a binomial, still with the log link
- The axillary prior is called like that cause it might be sth else depending on the family used (sigma, dispersion or shape)
- A neg binomial model should NOT be overdispersed, which would indicate our priors were too wide.

Predictions based on PRIOR

```
ggpredict(quinn.rstanarmNB,~SEASON+DENSITY) %>%
  plot(add.data=TRUE)
```



wide enough

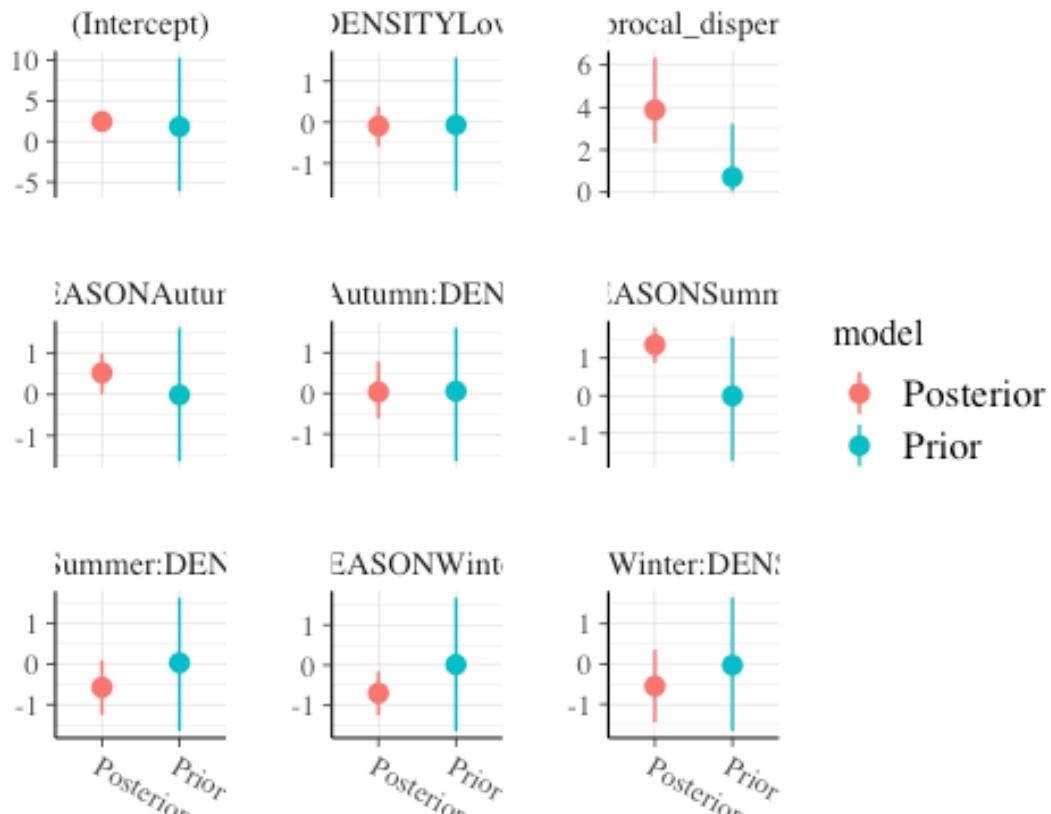
Looks

Now let's create the model with prior_PD=FALSE

```
quinn.rstanarmNB=update(quinn.rstanarmNB,prior_PD=FALSE)
```

Check the posterior vs prior

```
posterior_vs_prior(quinn.rstanarmNB,color_by="vs",group_by=TRUE,  
facet_args=list(scales="free_y"))  
  
##  
## Drawing from prior...
```



good

Looks

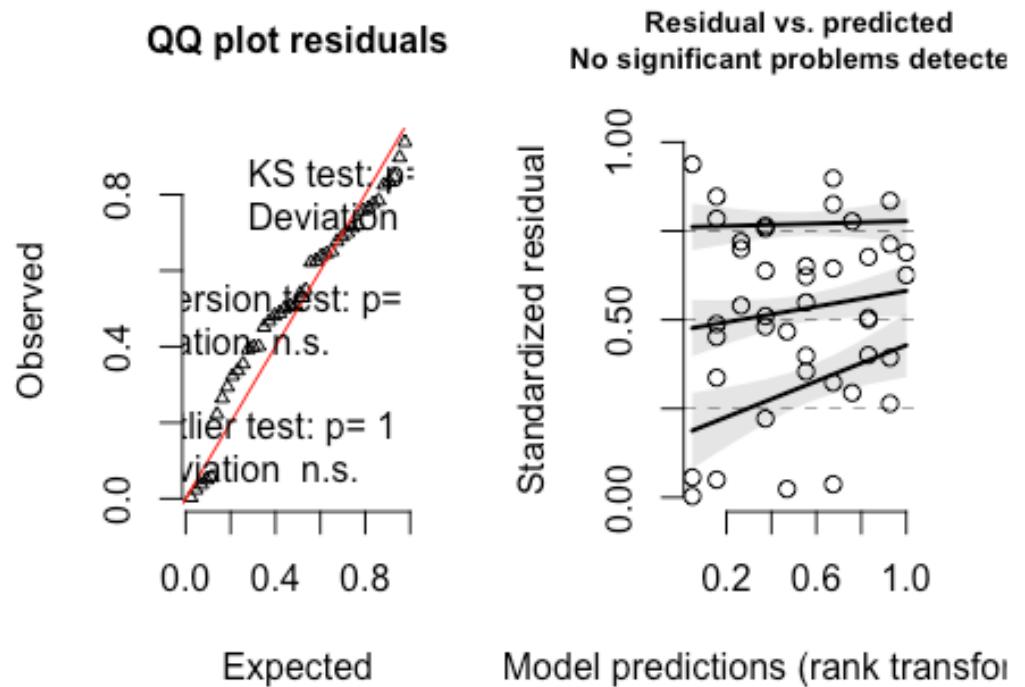
Model validation - DHARMA (let's skip the others)

```
preds<-posterior_predict(quinn.rstanarmNB, nsamples=250, summary=FALSE)

quinn.resids<-createDHARMA(simulatedResponse=t(preds),
                             observedResponse=quinn$RECRUITS,
                             fittedPredictedResponse=apply(preds, 2, median),
                             integerResponse=TRUE)

plot(quinn.resids)
```

DHARMA residual diagnostics

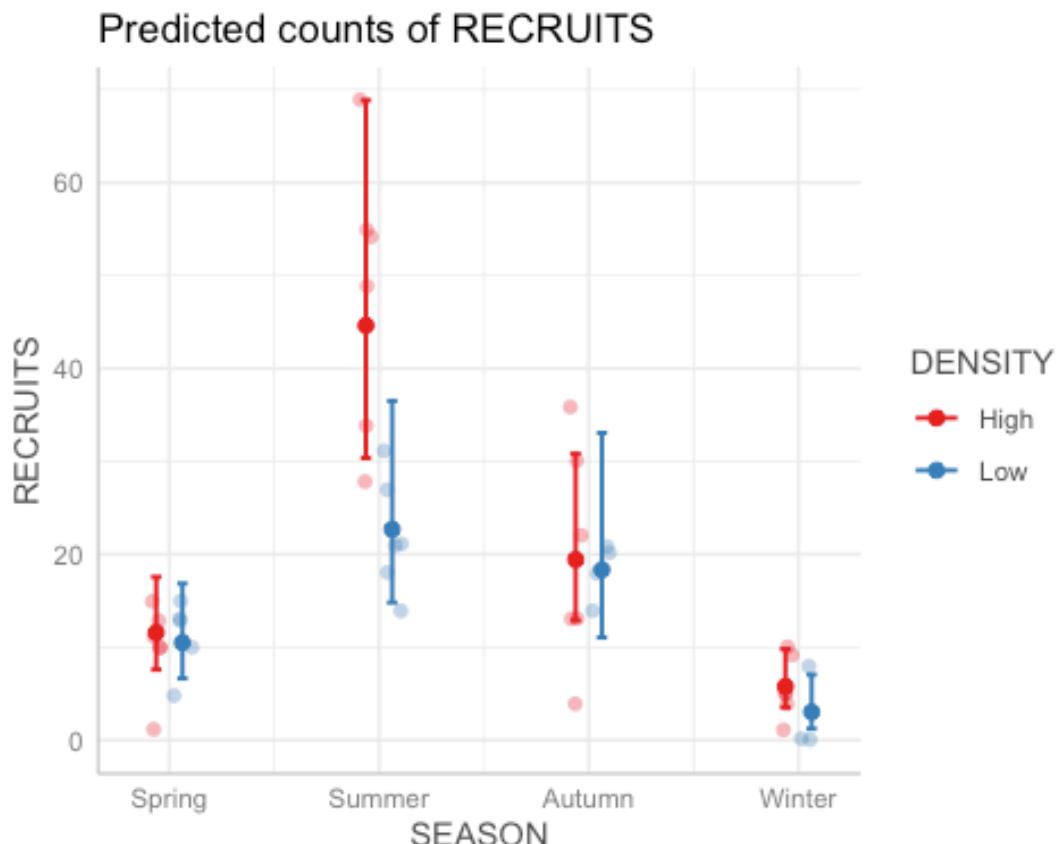


GOOD.

WE GOT OUR MODEL, AND IT LOOKS GOOD !! - A neg binomial model should NOT be overdispersed, which would indicate our priors were too wide.

Predicttions based on POSTERIOR

```
ggpredict(quinn.rstanarmNB, ~SEASON+DENSITY) %>%  
  plot(add.data=TRUE)
```



Our posterior looks pretty much like the raw data, so that was a good prior selection.

Table of the predicted data

```
broom.mixed:::tidyMCMC(quinn.rstanarmNB$stanfit, # tells R where the parameter
estimate (mcmc samples) are
  estimate.method="median", # we tell R what estimate we want CI from
  (mean or median)
  conf.int=TRUE,conf.method="HPDinterval",# we want them by HPD
  intervals, instead of quantiles
  rhat=TRUE, ess=TRUE) %>% # we want the effect size
knitr:::kable() # If we were to pipe all of this to the package kable, it
makes a nice neat table
```

term	estimate	std.error		conf.low	conf.high	rhat	ess
		r	std.error				
(Intercept)	2.449476	0.21727	2.009895	2.836430	1.00064	169	
	2	89	4	5	35	8	
SEASONSummer	1.354889	0.29021	0.780528	1.916730	0.99943	163	
	2	59	5	8	35	1	
SEASONAutumn	0.526515	0.29471	-	1.035205	0.99937	171	
	1	08	0.091759	0	64	1	

					8		
SEASONWinter	-	0.33381	-	-	0.99960	167	
	0.694966	91	1.331001	0.010452	34	3	
	3		0	9			
DENSITYLow	-	0.29646	-	0.441780	1.00017	164	
	0.092693	94	0.736699	3	89	8	
	8		3				
SEASONSUMMER:DENSITYLow	-	0.40385	-	0.225502	0.99901	168	
	0.574880	70	1.362674	9	69	2	
	4		6				
SEASONAutumn:DENSITYLow	0.033158	0.42316	-	0.942440	0.99903	167	
	2	16	0.728086	7	96	7	
			3				
SEASONWinter:DENSITYLow	-	0.53229	-	0.515787	0.99902	163	
	0.550147	95	1.596652	1	21	8	
	6		1				
reciprocal_dispersion	3.880490	1.24511	1.849838	6.441487	1.00325	184	
	2	53	5	1	56	4	
mean_PPD	18.38095	2.79461	13.83333	24.54761	1.00007	169	
	24	56	33	90	48	3	
log-posterior	-	2.26431	-	-	0.99944	162	
	157.5621	87	162.4390	154.2151	86	4	
	412		028	507			

By now, we know:

- The intercept is the avg number of whatever the response is (eg. amount of mussels in Spring at high density)
- These are on log scale (cause we used the log link for the neg binomial model) -> we'd have to check the `exp()` of these numbers to make interpretations about the data (e.g. $\exp(2.4)=11$ values of response)
- SEASONSUMMER = Diff bt Summer and Spring at high density. We'd expect here to be a large effect (see the graph). How much diff in % terms is the Summer recruitment and the Spring recruitment? On a log scale, it's 1.37 units on a log scale -> $\exp(1.37) = 3.935351 \rightarrow 3.94x$ more recruitment in Summer = 294% more recruitment in Summer at high density
- DENSITYLow = Diff btw high and low density ONLY in Spring. The diff btw low and high is not much -> $\exp(-0.08)=0.92 \rightarrow 1-0.92=0.08x$ less recruitment = 8% less recruitment; and indeed, the conf int overlap at 0 (=overlap-boundary in log scale; in normal scale = overlap at 1) - SEASONAutumn:DENSITYLow -> At the 95% it is non significant, but it might be at 92%

- There is no diff bte Low and High dens in SPring
- Big diff btw high dens in Spring and Summer
- We can also check the diff btw seasons
- R has NOT told us what diff we got btw high-low dens in Summer-Autumn-Winter -> We will do a pairwise comparisons test

Pairwise comparisons - emmeans

```
emmeans(quinn.rstanarmNB,pairwise~DENSITY|SEASON,type="response") #
type="response" back transforms our log data

## $emmeans
## SEASON = Spring:
## DENSITY prob lower.HPD upper.HPD
## High    11.58     7.463    17.05
## Low     10.51     6.403    16.32
##
## SEASON = Summer:
## DENSITY prob lower.HPD upper.HPD
## High    44.62    29.641    66.69
## Low     22.69    14.406    35.62
##
## SEASON = Autumn:
## DENSITY prob lower.HPD upper.HPD
## High    19.43    12.323    29.04
## Low     18.34    9.786    29.86
##
## SEASON = Winter:
## DENSITY prob lower.HPD upper.HPD
## High    5.78     3.347    9.25
## Low     3.04     0.926    6.05
##
## Point estimate displayed: median
## Results are back-transformed from the log scale
## HPD interval probability: 0.95
##
## $contrasts
## SEASON = Spring:
## contrast ratio lower.HPD upper.HPD
## High / Low 1.10     0.507    1.83
##
## SEASON = Summer:
## contrast ratio lower.HPD upper.HPD
## High / Low 1.94     1.007    3.30
##
## SEASON = Autumn:
## contrast ratio lower.HPD upper.HPD
## High / Low 1.05     0.440    1.86
##
## SEASON = Winter:
## contrast ratio lower.HPD upper.HPD
## High / Low 1.92     0.561    4.32
##
## Point estimate displayed: median
## Results are back-transformed from the log scale
## HPD interval probability: 0.95
```

emmeans gives us - The means itself (interesting for getting the ABSOLUTE values, see last code chunks of this Markdown file), the predicted values of each of the dots on the graph - The differences btw the groups (!) = “contrasts”

In this case, we see the following output: \$contrasts SEASON = Spring: -> Effect of density in Spring (what the previous table told us) contrast ratio lower.HPD upper.HPD High / Low 1.09 0.540 1.84 -> No effect, as the conf int overlap with 1 (would be 0 if on log scale) AND the high density only has 9% more recruitment than the low density in Spring

SEASON = Summer: -> Recruitment in high density is nearly twice as much than in the low density (99% more). In this case, even though the conf int overlap with 1, we see that the increase is massive (massive=over 90% Murray would say). If we had let's say 80% more, and the conf int overlap with 0, we'd still say that the response might not be that difference. So instead of using the conf int, we could calculate the PROBABILITIES, which tell us even more (see next code chunk) contrast ratio lower.HPD upper.HPD High / Low 1.99 0.950 3.32

SEASON = Autumn: contrast ratio lower.HPD upper.HPD High / Low 1.03 0.479 1.88

SEASON = Winter: contrast ratio lower.HPD upper.HPD High / Low 1.96 0.495 4.33

=> Conf int: Conf int can be useful for the threshold (0 for log, 1 for normal), but you always have to check the proportions too! If overlap with 1 on normal scale, but >90% diff btw groups, Murray would “ignore” the conf int and still say there is a strong effect.

Probabilities

```
quinn.em=emmeans(quinn.rstanarmNB,
                  pairwise~DENSITY|SEASON,
                  type="link")$contrast %>% # this package IGNORES the back
transformation "type=response", so we better define that we are on a Log Link
scale (Murray reckons it's a bug)
gather_emmeans_draws() %>%
mutate(Fit=exp(.value)) # we manually turn the Logged values to normal
```

This code should be a dataframe or tibble, with all MCMC samples (1800 x 4 seasons = 7200) and the columns will represent the contrasts/pairwise comparisons.

When we ran the first emmeans code (the pairwise comparisons, previous section), we were checking the pairwise comparisons. And it already does all the summary for us. But if we wanted to manipulate the data ourselves, it is better to do it this way, as we can get PROBABILITIES, CONF INT, and everything you need for PLOTTING.

```
quinn.em

## # A tibble: 7,200 x 7
## # Groups:   contrast, SEASON [4]
##   contrast   SEASON .chain .iteration .draw   .value   Fit
##   <fct>     <fct>    <int>      <int> <int>   <dbl> <dbl>
## 1 High - Low Spring     NA        NA     1  0.179  1.20
```

```

## 2 High - Low Spring      NA      NA    2  0.0851 1.09
## 3 High - Low Spring      NA      NA    3 -0.562  0.570
## 4 High - Low Spring      NA      NA    4 -0.244  0.784
## 5 High - Low Spring      NA      NA    5 -0.396  0.673
## 6 High - Low Spring      NA      NA    6  0.486  1.63
## 7 High - Low Spring      NA      NA    7  0.418  1.52
## 8 High - Low Spring      NA      NA    8  0.384  1.47
## 9 High - Low Spring      NA      NA    9  0.0396 1.04
## 10 High - Low Spring     NA      NA   10  0.380  1.46
## # ... with 7,190 more rows

```

Let's display the same table with the median and grouped by season

```

quinn.em %>% group_by(contrast,SEASON) %>%
  median_hdci(Fit)

## # A tibble: 4 x 8
## # Groups:   contrast [1]
##   contrast   SEASON   Fit .lower .upper .width .point .interval
##   <fct>     <fct>   <dbl> <dbl>  <dbl> <dbl> <chr>  <chr>
## 1 High - Low Spring  1.10  0.507  1.83   0.95 median hdci
## 2 High - Low Summer  1.94  1.01   3.30   0.95 median hdci
## 3 High - Low Autumn  1.05  0.440  1.86   0.95 median hdci
## 4 High - Low Winter  1.92  0.561  4.32   0.95 median hdci

```

Comparisons

Which groups have a chance to be GREATER in general?

```

quinn.em %>% group_by(contrast,SEASON) %>%
  summarize(P=sum(Fit>1)/n())

## `summarise()` regrouping output by 'contrast' (override with `groups` argument)

## # A tibble: 4 x 3
## # Groups:   contrast [1]
##   contrast   SEASON      P
##   <fct>     <fct>   <dbl>
## 1 High - Low Spring  0.63
## 2 High - Low Summer  0.991
## 3 High - Low Autumn  0.56
## 4 High - Low Winter  0.908

```

Let's have a look at this output: - 62 % chance that there is a diff btw High and Low in Spring, not much - 98 % chance that there is a diff btw High and Low in Summer, which is quite strong - 54% chance that there is a diff btw High and Low in Autumn, not much - 91% chance that there is a diff btw High and Low in Winter, which is strong Because of this output, we favour the probabilities rather than P values and conf intervals (!) Everyone can understand this output.

Which groups have a chance to be greater than 10%?

```
quinn.em %>% group_by(contrast,SEASON) %>%
  summarize(P=sum(Fit>1.1)/n())

## `summarise()` regrouping output by 'contrast' (override with ` .groups` argument)

## # A tibble: 4 x 3
##   contrast     SEASON      P
##   <fct>       <fct>    <dbl>
## 1 High        - Low      Spring 0.493
## 2 High        - Low      Summer 0.972
## 3 High        - Low      Autumn 0.454
## 4 High        - Low      Winter 0.875
```

Same explanation as above

Absolute change

What if we did not want % change, but absolute change? What if we wanted to say HOW MANY NEW RECRUITS we got? In frequentist analysis, you can get the point estimate, but not the uncertainty in it. When you work on a log link (log scale), or when back transformed on a ratio scale, you can't get the absolute response in frequentist analysis. BUT in Bayesian, YOU CAN!

The table we previously created, quinn.em, can take the actual MEANS and make the subtraction/addition btw groups. We had the following output: \$emmeans SEASON = Spring: DENSITY prob lower.HPD upper.HPD High 11.56 7.11 17.5 Low 10.63 6.53 16.9

SEASON = Summer: DENSITY prob lower.HPD upper.HPD High 45.17 29.65 65.0 Low 22.79 14.15 34.9

SEASON = Autumn: DENSITY prob lower.HPD upper.HPD High 19.25 11.80 28.6 Low 18.37 10.77 31.5

--> Rather than SUMMARISING the comparisons and putting them on the response scale, let's mutate in order to back transform.

Summary table (FYI)

```
emmeans(quinn.rstanarmNB, ~DENSITY | SEASON, type="response")

## SEASON = Spring:
##   DENSITY prob lower.HPD upper.HPD
##   High     11.58     7.463     17.05
##   Low      10.51     6.403     16.32
##
## SEASON = Summer:
##   DENSITY prob lower.HPD upper.HPD
```

```

##  High    44.62    29.641    66.69
##  Low     22.69    14.406    35.62
##
## SEASON = Autumn:
## DENSITY prob lower.HPD upper.HPD
##  High    19.43    12.323    29.04
##  Low     18.34    9.786    29.86
##
## SEASON = Winter:
## DENSITY prob lower.HPD upper.HPD
##  High     5.78     3.347     9.25
##  Low      3.04     0.926     6.05
##
## Point estimate displayed: median
## Results are back-transformed from the log scale
## HPD interval probability: 0.95

```

Example: SEASON = Spring: DENSITY prob lower.HPD upper.HPD High 11.56 7.11 17.5 =>
= Summary of the fit column of the table below Low 10.63 6.53 16.9

Absolute values table

```

emmeans(quinn.rstanarmNB, ~DENSITY | SEASON, type = "link") %>%
  gather_emmeans_draws() %>%
  mutate(Fit = exp(.value)) # we back transform the values

## # A tibble: 14,400 x 7
## # Groups:   DENSITY, SEASON [8]
##   DENSITY SEASON .chain .iteration .draw .value   Fit
##   <fct>   <fct>   <int>     <int> <int> <dbl> <dbl>
## 1 High     Spring     NA       NA     1    2.07  7.93
## 2 High     Spring     NA       NA     2    2.67 14.4
## 3 High     Spring     NA       NA     3    2.35 10.5
## 4 High     Spring     NA       NA     4    2.13  8.44
## 5 High     Spring     NA       NA     5    2.23  9.32
## 6 High     Spring     NA       NA     6    2.26  9.57
## 7 High     Spring     NA       NA     7    2.70 14.9
## 8 High     Spring     NA       NA     8    2.54 12.7
## 9 High     Spring     NA       NA     9    2.12  8.34
## 10 High    Spring     NA       NA    10    2.38 10.8
## # ... with 14,390 more rows

```

These rows have now become our population. The absolute values (=number of recruits) are displayed in the Fit column. We can do whatever we want with those, putting them on a spreadsheet and subtract the high from the low ones, therefore calculating the absolute diff btw high and low in Spring.

We could express diff in % terms and absolute terms. Once you got those numbers, you can do whatever with them.

**WE WOULD IN THEORY CALCULATE THE DIFF IN ABSOLUTE VALUES
AND PLOT THIS**

IGNORE BELOW

Model validation

Model investigation / hypothesis testing

Predictions

Summary figures

References

19_Bayesian_GLMM

Sara Kophamel

17/12/2020

Open: bglmm_example1.Rmd

THEORY - MIXED EFFECTS MODEL (MEM)

In frequentist, R tries to find the peak via various algorithms to find that peak, which is a difficult task for mixed effects models. When we MEM, the stats to use are VERY difficult. But when it comes to Bayesian, it is actually not that much different or difficult than regular parameters. It won't find a peak, as it is trying to recreate the shape. Finding the peak is what's difficult. What IS slightly more difficult in Bayesian is the following:

Model formula:

$$y_i \sim N(\mu_i, \sigma^2) \quad \mu_i = \boldsymbol{\beta} \bf{X}_i + \boldsymbol{\gamma} \bf{Z}_i$$

where β and γ are vectors of the fixed and random effects parameters respectively and \bf{X} is the model matrix representing the overall intercept and effects of the treatment on the number of lesions. \bf{Z} represents a cell means model matrix for the random intercepts associated with leaves.

This equation represents how the intercepts vary between blocks (groups): population mean = $\beta_0 + \beta_1 * x_1 + \gamma_2$ (γ looks like a Y with a lazo at the bottom, is made up of many intercepts)

In other words, there is no slope associated to the groups, just the variance. We assume that all the intercepts are drawn from a normal distribution with equal variance:

$$y \sim \text{Normal}(\beta_0 + \beta_1 * x_1 + \gamma_2, \sigma^2)$$

Although most of the times Bayesian is fine for MEM, the distribution of MEM data tends to have a funnel shape, with some data being very narrow. Bayesian has troubles finding this narrower area.

=> That is why we will rely on the DEFAULT PRIORS drawn from the CAUCHY distribution, which is what statistitians use for MEM. If we defined our own priors, we would very likely miss the real distribution.

Preparations

Load the necessary libraries

```
library(rstanarm)    #for fitting models in STAN
library(brms)         #for fitting models in STAN
library(coda)          #for diagnostics
library(bayesplot)    #for diagnostics
library(rstan)         #for interfacing with STAN
library(emmeans)       #for marginal means etc
library(broom)          #for tidying outputs
library(broom.mixed)   #for tidying outputs
library(DHARMa)        #for residual diagnostics
library(tidybayes)     #for more tidying outputs
library(ggeffects)     #for partial plots
library(tidyverse)      #for data wrangling etc
```

Scenario

A plant pathologist wanted to examine the effects of two different strengths of tobacco virus on the number of lesions on tobacco leaves. She knew from pilot studies that leaves were inherently very variable in response to the virus. In an attempt to account for this leaf to leaf variability, both treatments were applied to each leaf. Eight individual leaves were divided in half, with half of each leaf inoculated with weak strength virus and the other half inoculated with strong virus. So the leaves were blocks and each treatment was represented once in each block. A completely randomised design would have had 16 leaves, with 8 whole leaves randomly allocated to each treatment.



Tobacco plant

Format of tobacco.csv data files

LEAF	TREAT	NUMBER
1	Strong	35.898
1	Weak	25.02
2	Strong	34.118
2	Weak	23.167
3	Strong	35.702
3	Weak	24.122
...
LEAF	The blocking factor - Factor B	
TREAT	Categorical representation of the strength of the tobacco virus - main factor of interest Factor A	
NUMBER	Number of lesions on that part of the tobacco leaf - response variable	

=> Each leaf receives both treatments, so it acts as control for itself. It is important to randomise the order to not introduce more systematic bias. What we are interested in here, is in how varied the leaves were (we don't care about the values, the intercepts). Whereas, in the treatment, we are interested in the diff btw weak and strong per leaf.

Read in the data

```
tobacco = read_csv('data/tobacco.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   LEAF = col_character(),
##   TREATMENT = col_character(),
##   NUMBER = col_double()
## )

glimpse(tobacco)

## Rows: 16
## Columns: 3
## $ LEAF      <chr> "L1", "L1", "L2", "L2", "L3", "L3", "L4", "L4", "L5",
## "L5",...
## $ TREATMENT <chr> "Strong", "Weak", "Strong", "Weak", "Strong", "Weak",
## "Stro...
## $ NUMBER    <dbl> 35.89776, 25.01984, 34.11786, 23.16740, 35.70215,
## 24.12191,...
```

NUMBER = avg number of lesions, which is a pity. We'd benefit more from the actual counts of lesions. It is hard to match a distribution to an avg count. Counts we'd fit with Poisson or Neg Binomial. For avg counts, you can use a Gaussian or normal, but data WILL be skewed

with small numbers, as data won't be <0. However, because we got heaps of lesions, Murray hopes we'll be ok and have no skewed data. But we need to keep in mind that small sample sizes are usually right skewed whenever the data is positive.

Declare the chr variables as factors

We don't really need to order TREATMENT, as we only got two groups (weak and strong)

```
tobacco=tobacco %>%
  mutate(LEAF=factor(LEAF),
        TREATMENT=factor(TREATMENT))

glimpse(tobacco)

## # Rows: 16
## # Columns: 3
## $ LEAF      <fct> L1, L1, L2, L2, L3, L3, L4, L4, L5, L5, L6, L6, L7, L7,
## $ TREATMENT <fct> Strong, Weak, Strong, Weak, Strong, Weak, Strong, Weak,
## Str...
## $ NUMBER    <dbl> 35.89776, 25.01984, 34.11786, 23.16740, 35.70215,
## 24.12191,...
```

That looks better

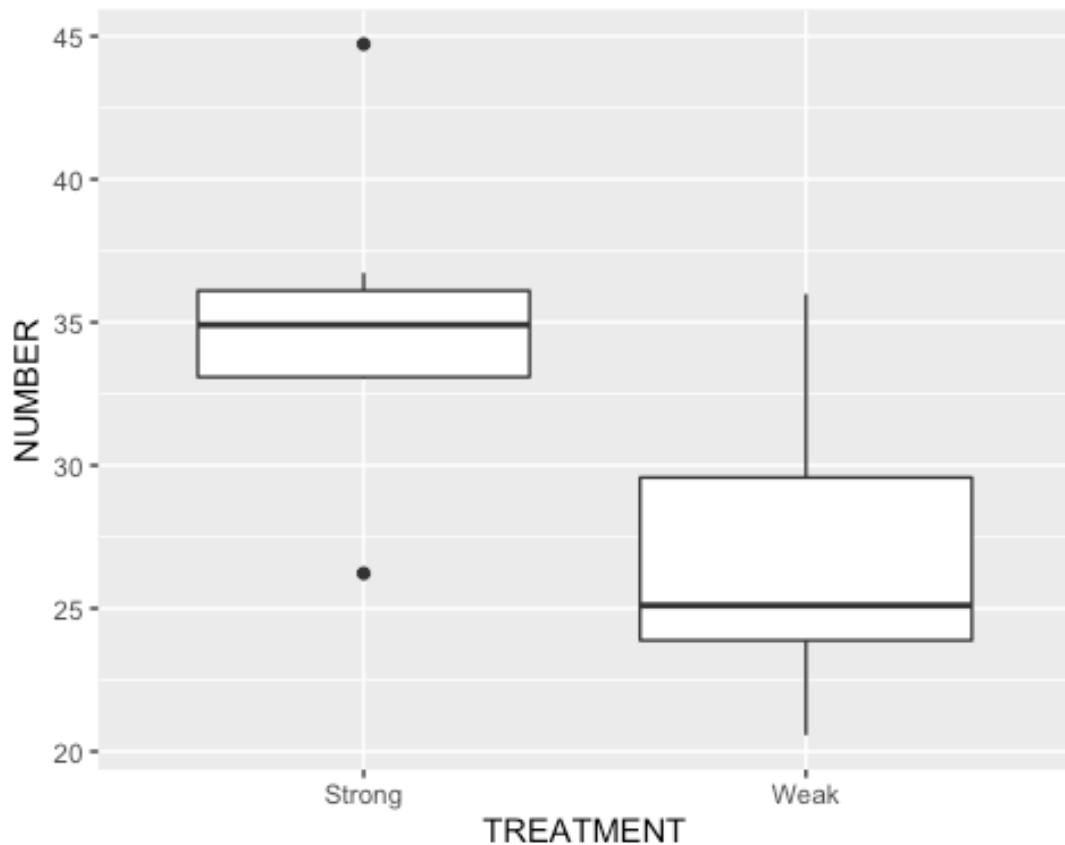
Exploratory data analysis

Plots

Boxplot

We got a categorical predictor, so we'd go for a boxplot. We are trying to assess normality and equal variance

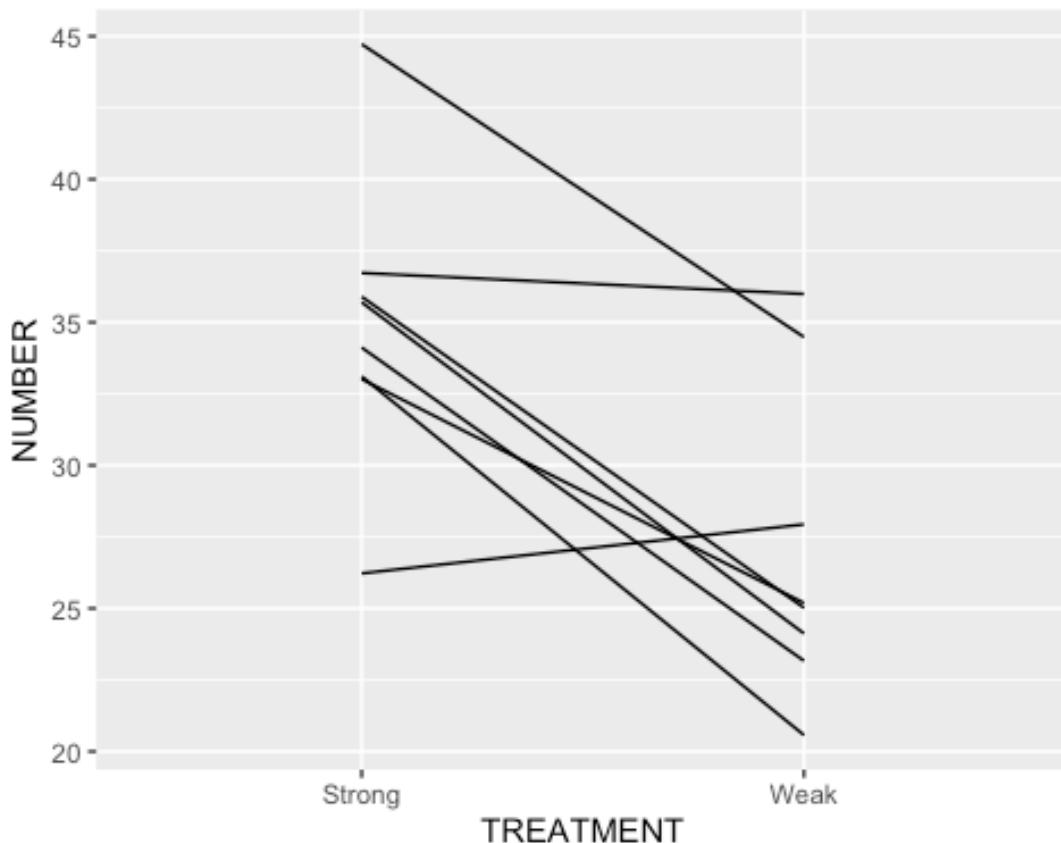
```
ggplot(data=tobacco, aes(x=TREATMENT,y=NUMBER))+
  geom_boxplot()
```



The data looks: - normally distributed - good reg homogeneity of variance => We'll use GAUSSIAN distribution

Line plots

```
ggplot(tobacco,aes(y=NUMBER,x=TREATMENT, group=LEAF))+
  geom_line()
```



This plot

is useful cause it describes the random intercept nature of the data.

The intercept varies per leaf btw strong and weak treatments. Therefore, displaying intercepts and random slopes (and analysing them further) might be useful. In this case, two leaves differ from the other six. When we try to run this analysis in frequentist, we identified it would be nice to fit a random slope model, but it did not have enough df to fit this sort of complex model. But we don't have this problem in Bayesian. As long as we got sensible priors, it will still run. That means that Bayesian works with small datasets, although the results might be very noisy (=we'd get an estimate with very high uncertainty).

Another important thing about random effects is that, if you have <5 blocks/levels of a random effect, you could get rubbish. Frequentist shrinks a LOT of effect when you got <5 blocks (eg. 5 leaves, 5 sites...). Don't try and fit sth with let's say 3 levels and expect it to run well.

Fit the models

Let's start with a simple model and build up. Afterwards, we'll complicate it.

1- Random intercept model

Model for priors

Let's use the default priors, maybe with 2000 warmups as it's a Gaussian.

```
tobacco.rstanarm<-stan_glmer(NUMBER~(1|LEAF)+ # we nominate a random
intercept that depends/is conditional on each Leaf
TREATMENT, data=tobacco,
family=gaussian, # you could define the family
with "", (), whatever. If we leave it without specification, R would use the
default link, the IDENTITY link.
refresh=0,
prior_PD=TRUE, # we define the model entirely
based on the (default) prior
chains=3,
iter=5000,
warmup=2000,
thin=5)
```

Let's check the Priors

```
prior_summary(tobacco.rstanarm) # this is run with the package rstanarm (if
we see that brms is running this command, run it with rstanarm::)

## Priors for model 'tobacco.rstanarm'
## -----
## Intercept (after predictors centered)
## Specified prior:
##     ~ normal(location = 31, scale = 2.5)
## Adjusted prior:
##     ~ normal(location = 31, scale = 16)
##
## Coefficients
## Specified prior:
##     ~ normal(location = 0, scale = 2.5)
## Adjusted prior:
##     ~ normal(location = 0, scale = 32)
##
## Auxiliary (sigma)
## Specified prior:
##     ~ exponential(rate = 1)
## Adjusted prior:
##     ~ exponential(rate = 0.15)
##
## Covariance
## ~ decov(reg. = 1, conc. = 1, shape = 1, scale = 1)
## -----
## See help('prior_summary.stanreg') for more details
```

We need - the intercept - the slope - sigma (variance) - the covariance (we won't touch this one, as R has worked out how to be sensible about it)

=> If we wanted to find our own priors, we'll use this: Covariance $\sim \text{decov}(\text{reg.} = 1, \text{conc.} = 1, \text{shape} = 1, \text{scale} = 1)$

=> Let's have a look at the Intercept: Intercept (after predictors centered) Specified prior: $\sim \text{normal}(\text{location} = 31, \text{scale} = 2.5)$ Adjusted prior: $\sim \text{normal}(\text{location} = 31, \text{scale} = 16) \Rightarrow 31$ is the mean of the strong treatment => The scale is the SD of the strength of the predictors. We can define this based on the plot, so we could come up with anything btw 0-100. 2.5 is the default, and it multiplies by the scale of our data to get the SD=16 in the Adjusted prior. We'll use 16 as the SD. It looks like the highest data (see line plot) goes from somewhere around mid 30s to mid 20s. A scale of 10 (instead of 16) would therefore probably have also been ok for our prior.

Coefficients Specified prior: $\sim \text{normal}(\text{location} = 0, \text{scale} = 2.5)$ Adjusted prior: $\sim \text{normal}(\text{location} = 0, \text{scale} = 32) \Rightarrow$ Coefficients always have a mean of zero, cause you don't know if the response is gonna increase or decrease from 0 => A scale of 32 is quite high, but we'll see how the model does in the next steps

=> Cauchy always uses an exponential. => Our values are on a NATURAL (not log) scale, so the priors look good

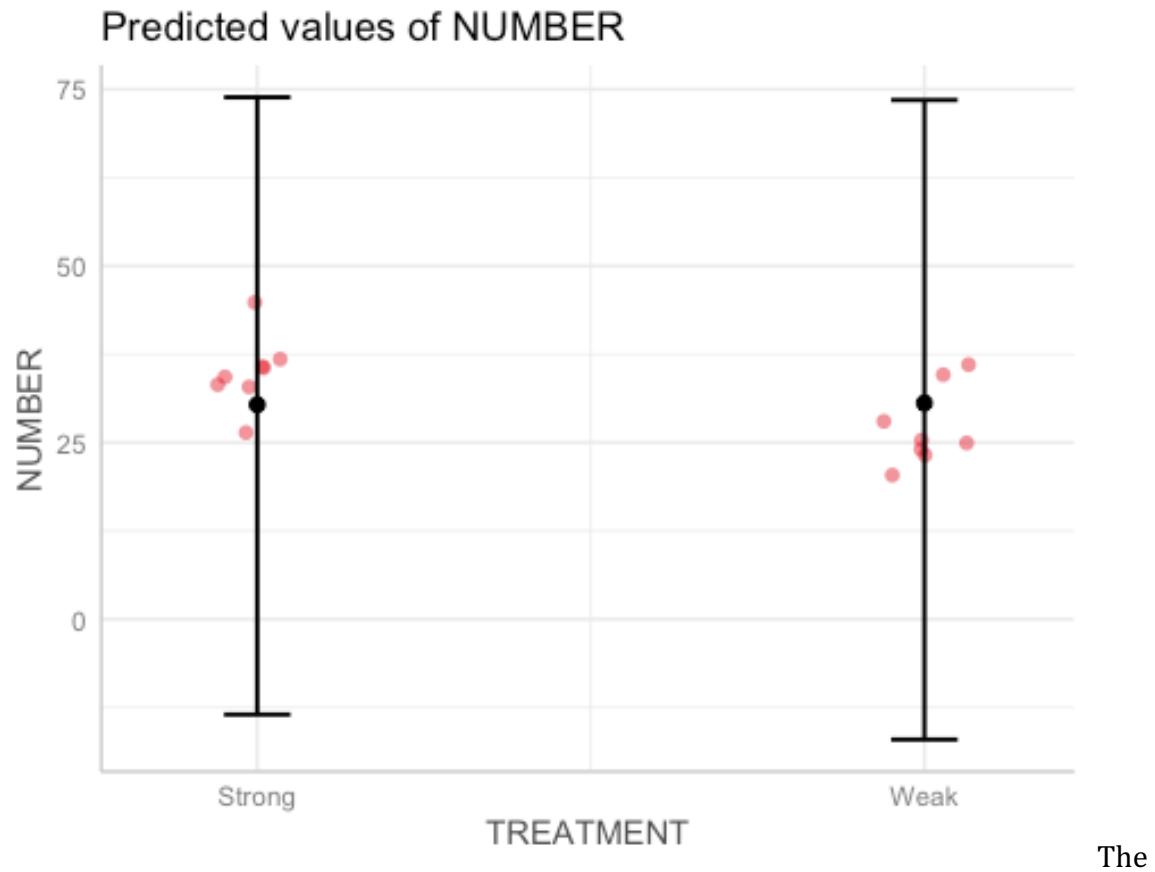
=> We could now either fit our own priors, or go with the DEFAULT ones. Let's go first with the default priors.

Prior predictions

Prior predictions

What is the range of values that we could predict in the absence of our actual data?

```
ggpredict(tobacco.rstanarm) %>%  
  plot(add.data=TRUE)  
  
## Note: uncertainty of error terms are not taken into account. You may want  
## to use `rstantools::posterior_predict()`.  
  
## $TREATMENT
```



prior predictions are ok.

Now we have to update the model so that the actual data gets included, and check the predictions again

Model for posteriors

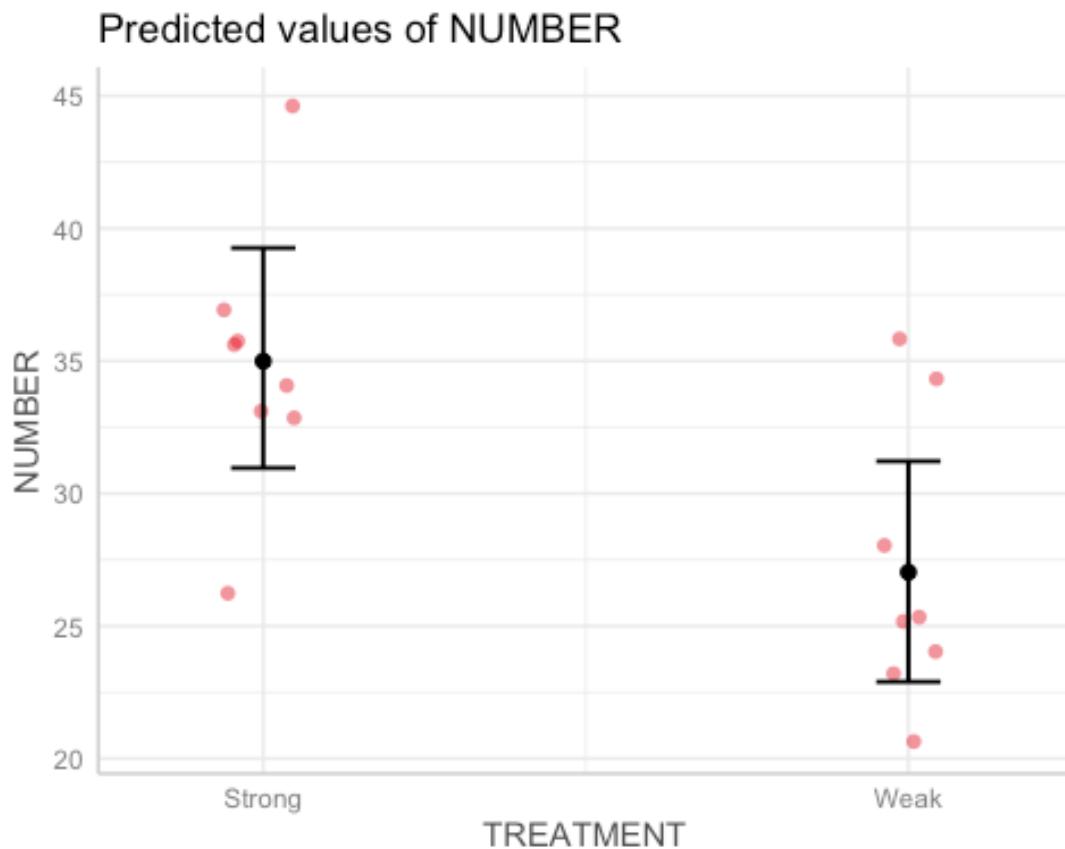
```
tobacco.rstanarm<-update(tobacco.rstanarm,prior_PD=FALSE)
```

Prediction for posteriors

```
ggpredict(tobacco.rstanarm) %>% plot(add.data=TRUE)
```

```
## Note: uncertainty of error terms are not taken into account. You may want  
to use `rstantools::posterior_predict()`.
```

```
## $TREATMENT
```



posterior predictions (equivalent to our data) look reasonable.

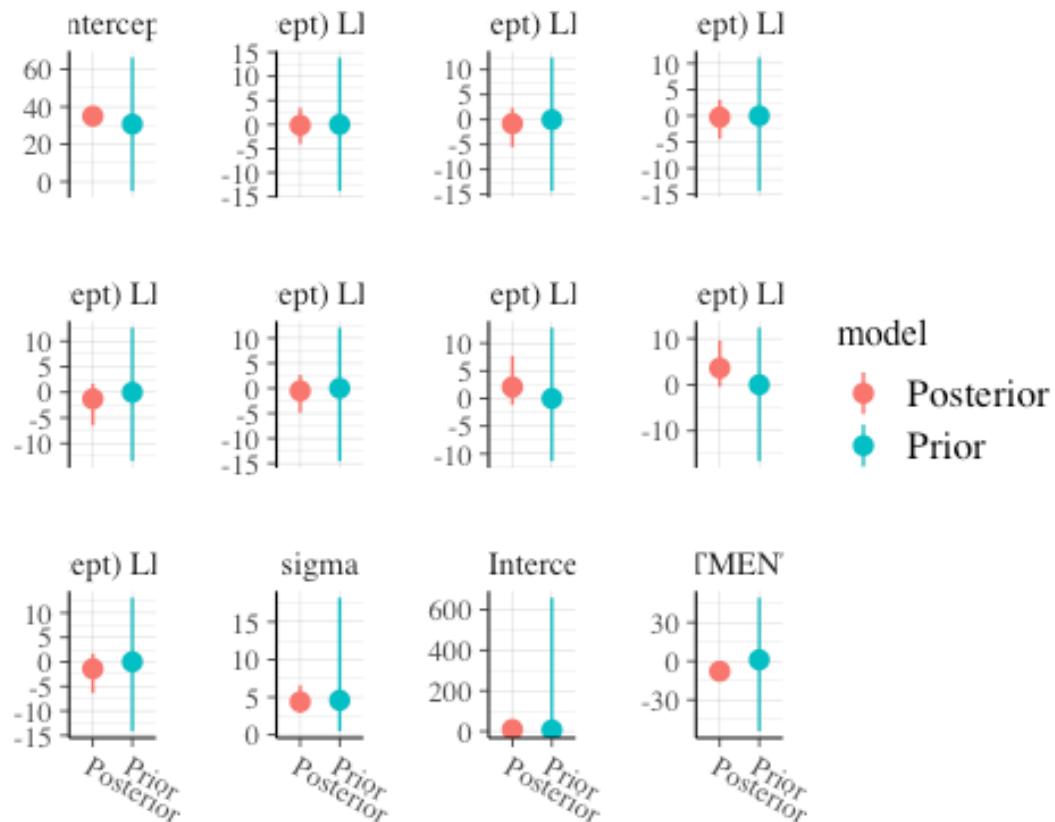
It might still be worth plotting the prior and the posterior to confirm that the posteriors are not being driven by the priors alone

The

Prior vs posterior

```
posterior_vs_prior(tobacco.rstanarm, color_by="vs",
                    group_by=TRUE,
                    facet_args=list(scales="free_y"))

##  
## Drawing from prior...
```



This looks great. - Here, we can see all components of the equation we discussed earlier on (with beta 1, beta 2, Y...) - The last plot shows the Treatment. - The priors are substantially larger than the posteriors, so they are unlikely influencing the posteriors. It is the DATA which is influencing the posteriors.

WE COULD NOW CONTINUE WITH OUR MODEL VALIDATION, OR FIT A MORE COMPLEX MODEL (NOT REALLY NEEDED IN BAYESIAN, BUT LET'S DO IT FOR COMPARISON PURPOSES)

2- Random intercept random slope model

This model takes into acc that slopes might be different (as identified in the line plot). We therefore assume that a random SLOPE model will better fit the data

Model for priors

Let's use the default priors, maybe with 2000 warmups as it's a Gaussian.

```
tobacco.rstanarm1<-stan_glmer(NUMBER~(TREATMENT|LEAF)+ # we nominate a random
intercept. It tells R that the intercept varies by Leaf and the slope varies
by Leaf. It is the same as if we had written: (1+TREATMENT|1+LEAF)
TREATMENT, data=tobacco,
```

```

family=gaussian, # you could define the family
with "", (), whatever. If we leave it without specification, R would use the
default Link, the IDENTITY link.
refresh=0,
prior_PD=TRUE, # we define the model entirely
based on the (default) prior
chains=3,
iter=5000,
warmup=2000,
thin=5)

```

Let's check the Priors

```

prior_summary(tobacco.rstanarm1) # this is run with the package rstanarm (if
we see that brms is running this command, run it with rstanarm::)

## Priors for model 'tobacco.rstanarm1'
## -----
## Intercept (after predictors centered)
##   Specified prior:
##     ~ normal(location = 31, scale = 2.5)
##   Adjusted prior:
##     ~ normal(location = 31, scale = 16)
##
## Coefficients
##   Specified prior:
##     ~ normal(location = 0, scale = 2.5)
##   Adjusted prior:
##     ~ normal(location = 0, scale = 32)
##
## Auxiliary (sigma)
##   Specified prior:
##     ~ exponential(rate = 1)
##   Adjusted prior:
##     ~ exponential(rate = 0.15)
##
## Covariance
##   ~ decov(reg. = 1, conc. = 1, shape = 1, scale = 1)
## -----
## See help('prior_summary.stanreg') for more details

```

We need - the intercept - the slope - sigma (variance) - the covariance (we won't touch this one, as R has worked out how to be sensible about it)

=> If we wanted to find our own priors, we'll use this: Covariance $\sim \text{decov}(\text{reg.} = 1, \text{conc.} = 1, \text{shape} = 1, \text{scale} = 1)$

=> Let's have a look at the Intercept: Intercept (after predictors centered) Specified prior: $\sim \text{normal}(\text{location} = 31, \text{scale} = 2.5)$ Adjusted prior: $\sim \text{normal}(\text{location} = 31, \text{scale} = 16)$ => 31 is the mean of the strong treatment => The scale is the SD of the strength of the predictors. We can define this based on the plot, so we could come up with anything btw 0-100. 2.5 is

the default, and it multiplies by the scale of our data to get the SD=16 in the Adjusted prior. We'll use 16 as the SD. It looks like the highest data (see line plot) goes from somewhere around mid 30s to mid 20s. A scale of 10 (instead of 16) would therefore probably have also been ok for our prior.

Coefficients Specified prior: $\sim \text{normal}(\text{location} = 0, \text{scale} = 2.5)$ Adjusted prior: $\sim \text{normal}(\text{location} = 0, \text{scale} = 32)$ => Coefficients always have a mean of zero, cause you don't know if the response is gonna increase or decrease from 0 => A scale of 32 is quite high, but we'll see how the model does in the next steps

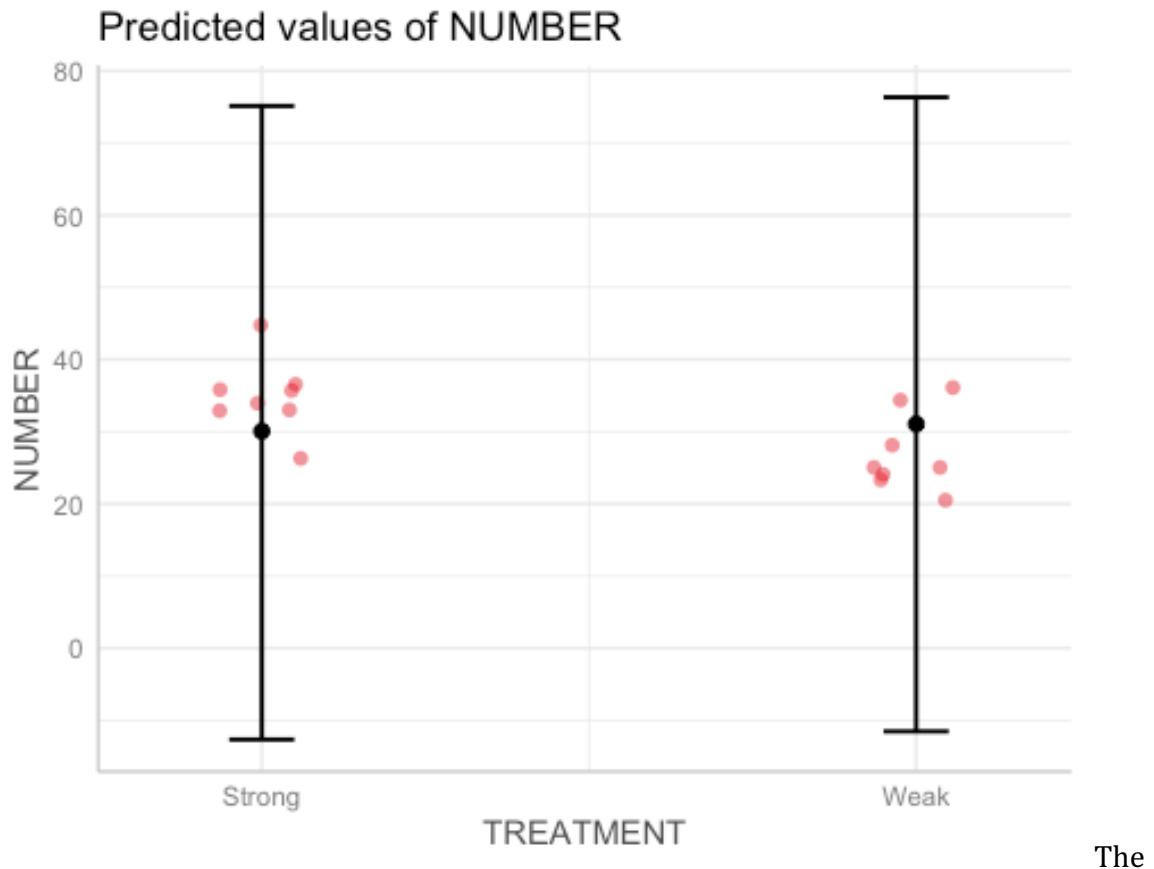
=> Cauchy always uses an exponential. => Our values are on a NATURAL (not log) scale, so the priors look good

=> We could now either fit our own priors, or go with the DEFAULT ones. Let's go first with the default priors.

Prior predictions

What is the range of values that we could predict in the absence of our actual data?

```
ggpredict(tobacco.rstanarm1) %>%  
  plot(add.data=TRUE)  
  
## Note: uncertainty of error terms are not taken into account. You may want  
to use `rstantools::posterior_predict()`.  
  
## $TREATMENT
```



prior predictions are ok.

Now we have to update the model so that the actual data gets included, and check the predictions again

Model for posteriors

```
tobacco.rstanarm1<-update(tobacco.rstanarm1, prior_PD=FALSE)
```

Ignore the warning message if the divergent transitions are not too high (<10 is ok). Murray had 57 (not sure why), so he'd be slightly concerned. Imagine again that our data is funnel-shaped. This is because: - Somewhere in our model, the sampler hit a very sharp/narrow feature and was not getting the distribution right (in Murrays case, this happened 57 times) = R is having a lot of samples which are not effective/ efficient. If we had <10 divergent transitions, we would not really care, cause it's only very few of the samplings which went the wrong way.

-> We could set our adaptive delta = Tell R what size of steps the sampler has to take. The recommendation is to take a sample space of 0.9, but we'll increase this to 0.99 to try and get the sampler to take bigger steps and possibly sample the areas where there is actually data. In other words, the model will take longer to run, but it will get the distribution right.

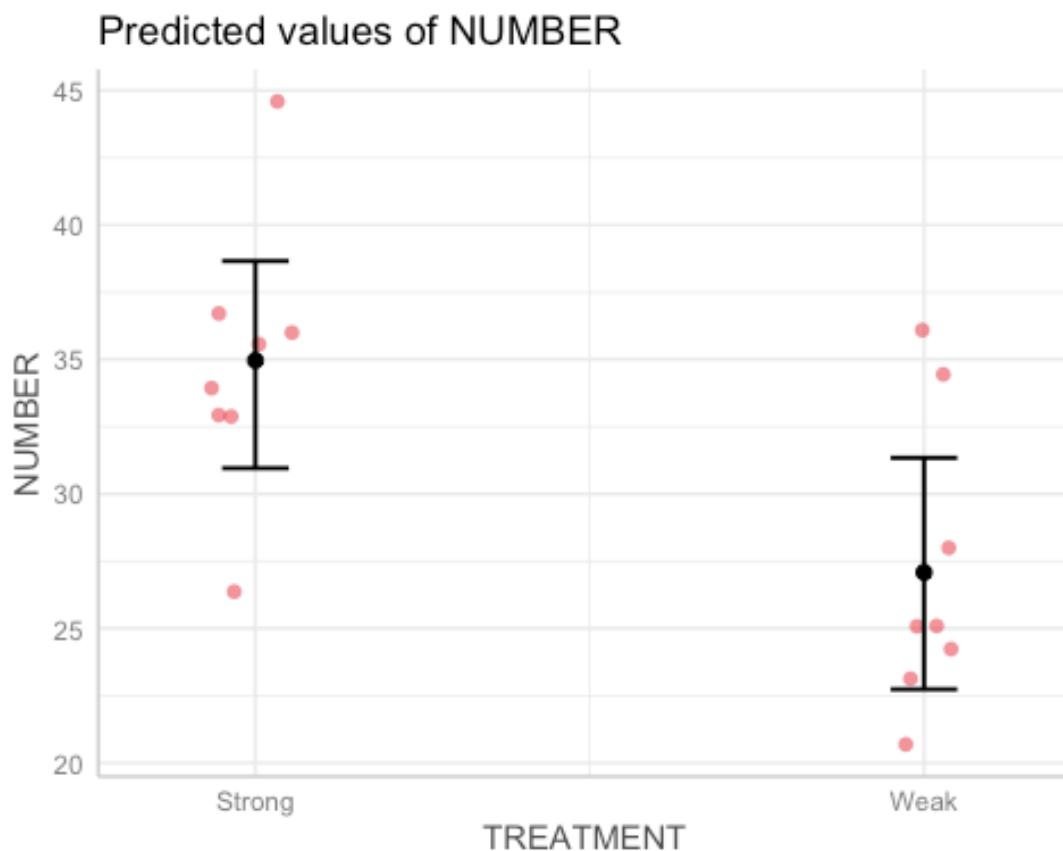
Correcting the steps

This would be done with the following code:

```
tobacco.rstanarm1<-update(tobacco.rstanarm1,prior_PD=FALSE,adapt_delta=0.99)
```

Prediction for posteriors

```
ggpredict(tobacco.rstanarm1) %>% plot(add.data=TRUE)  
  
## Note: uncertainty of error terms are not taken into account. You may want  
to use `rstantools::posterior_predict()`.  
  
## $TREATMENT
```



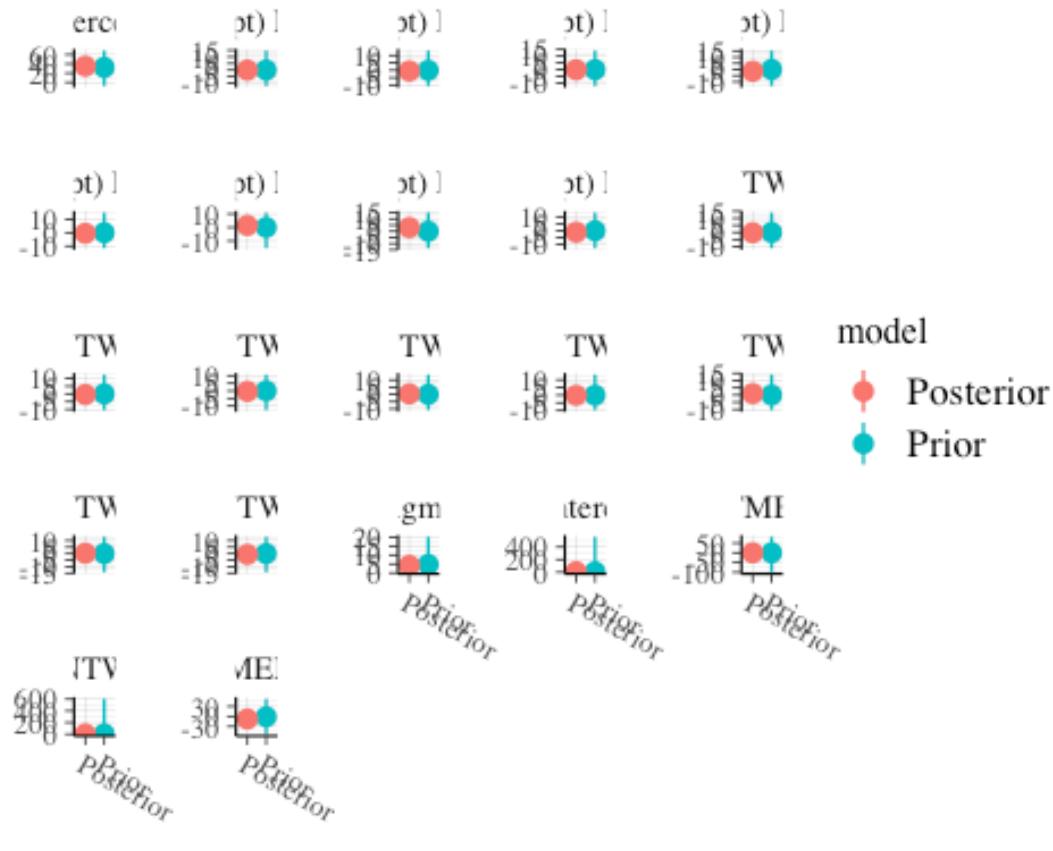
The posterior predictions (equivalent to our data) look reasonable based on the raw data. (FYI: The dots are jittered along the x axis)

It might still be worth plotting the prior and the posterior to confirm that the posteriors are not being driven by the priors alone

Prior vs posterior

```
posterior_vs_prior(tobacco.rstanarm1,color_by="vs",  
                    group_by=TRUE,  
                    facet_args=list(scales="free_y"))
```

```
##  
## Drawing from prior...
```



good, with priors substantially larger than posteriors = Posteriors are mainly driven by the DATA, not the priors.

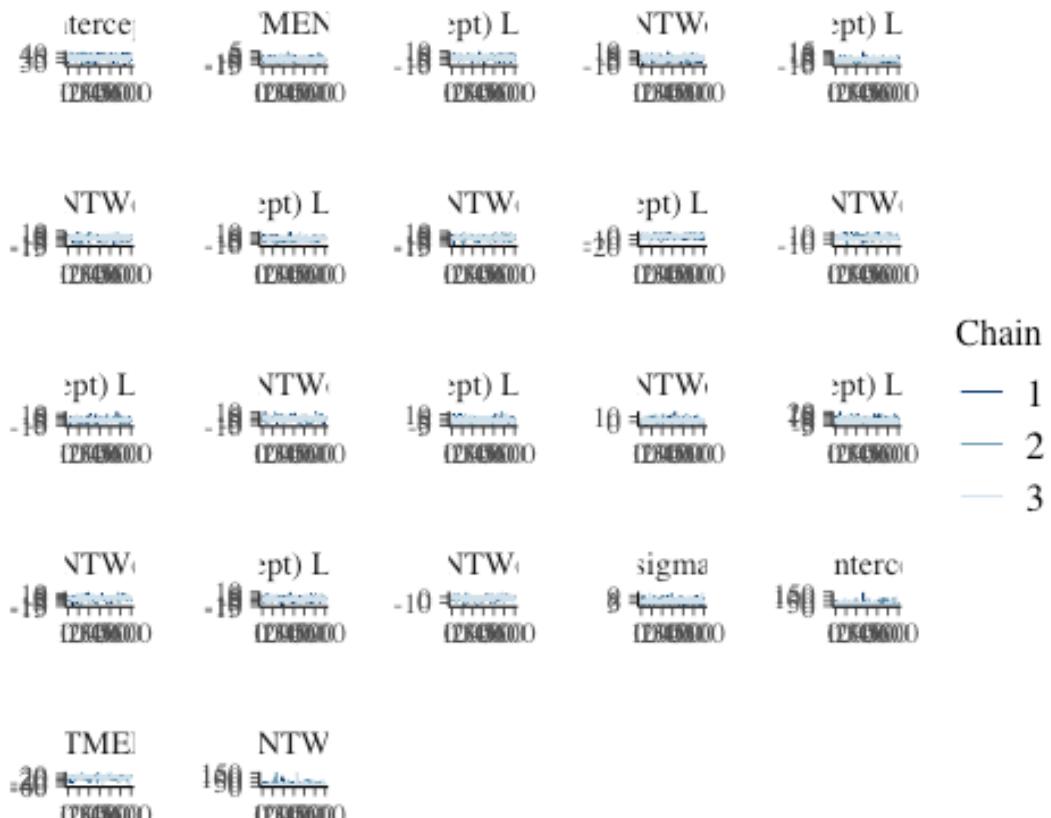
=> In summary, both the random intercept and the random slope model seem to be suitable for our data.

Model validation

We decide to go for the most complicated model.

Trace plots

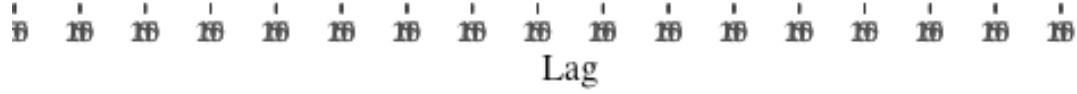
```
plot(tobacco.rstanarm1, "mcmc_trace")
```



Intercept, Treatment and the both sigmas (sigma, and F:(Intercept)). Sigmas will be truncated at 0. The plot sigma is quite unusual, as the residuals and SD are quite high.

Autocorrelation

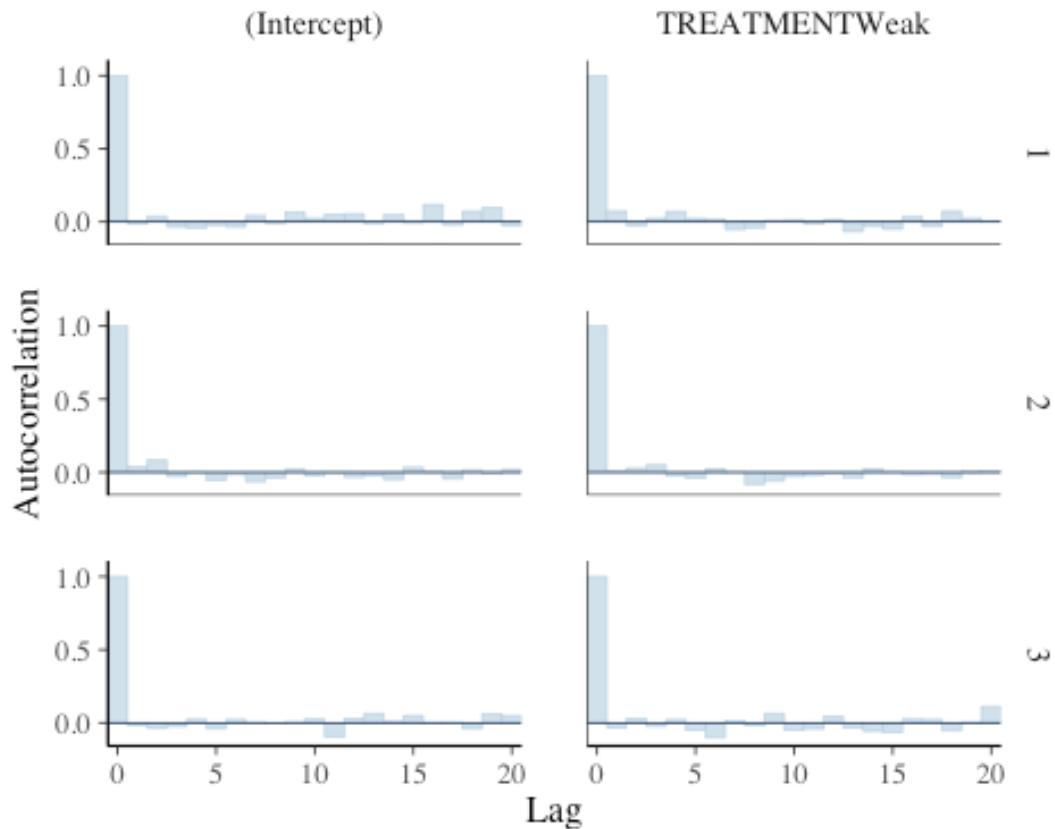
```
plot(tobacco.rstanarm1, "mcmc_acf_bar")
```



You'd have to open the autocorrelation plot additionally and zoom in here to see sth, but it's ok.

Another solution is to define THE parameters that we need the diagnostic test for

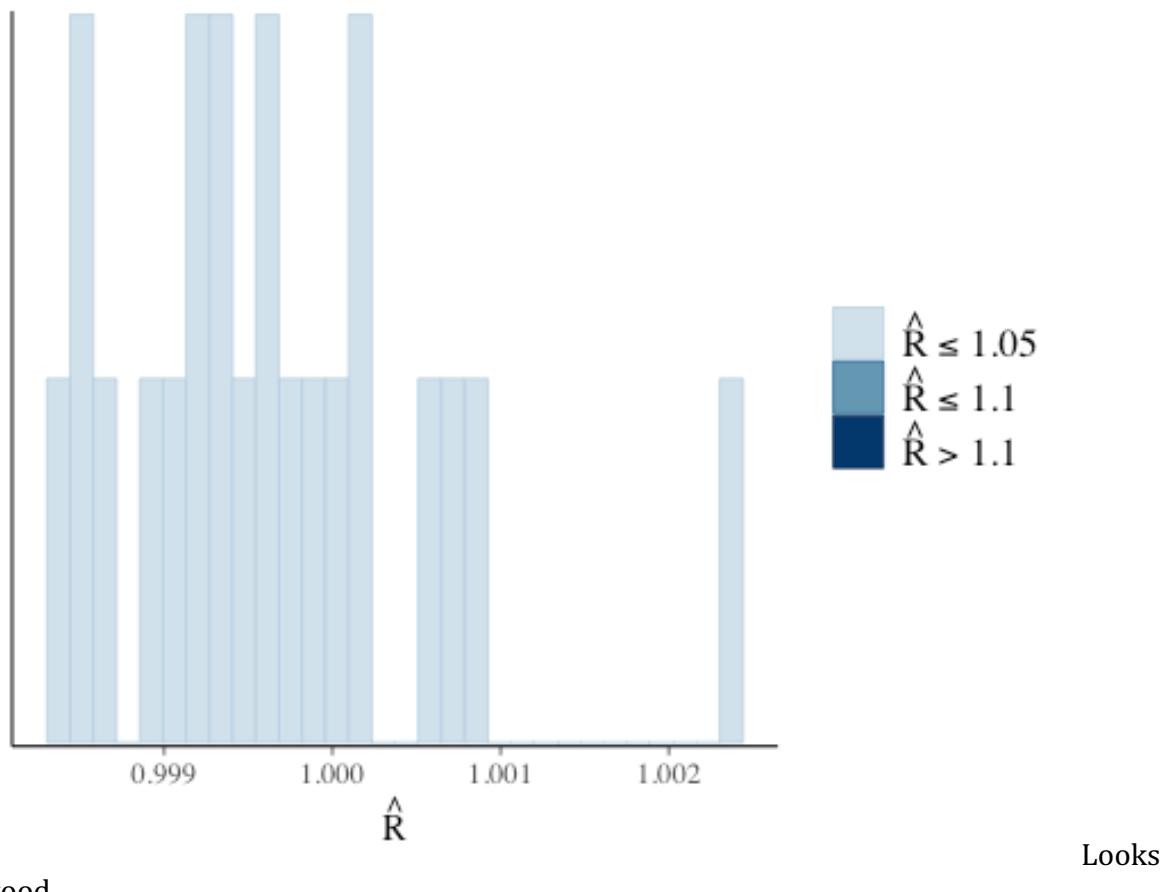
```
plot(tobacco.rstanarm1, "mcmc_acf_bar", pars=c('(Intercept)', 'TREATMENTWeak'))
```



Here, we are only displaying the parameters we are interested in (Intercept and Treatment weak). We want the first bar to be the highest, which is the case.

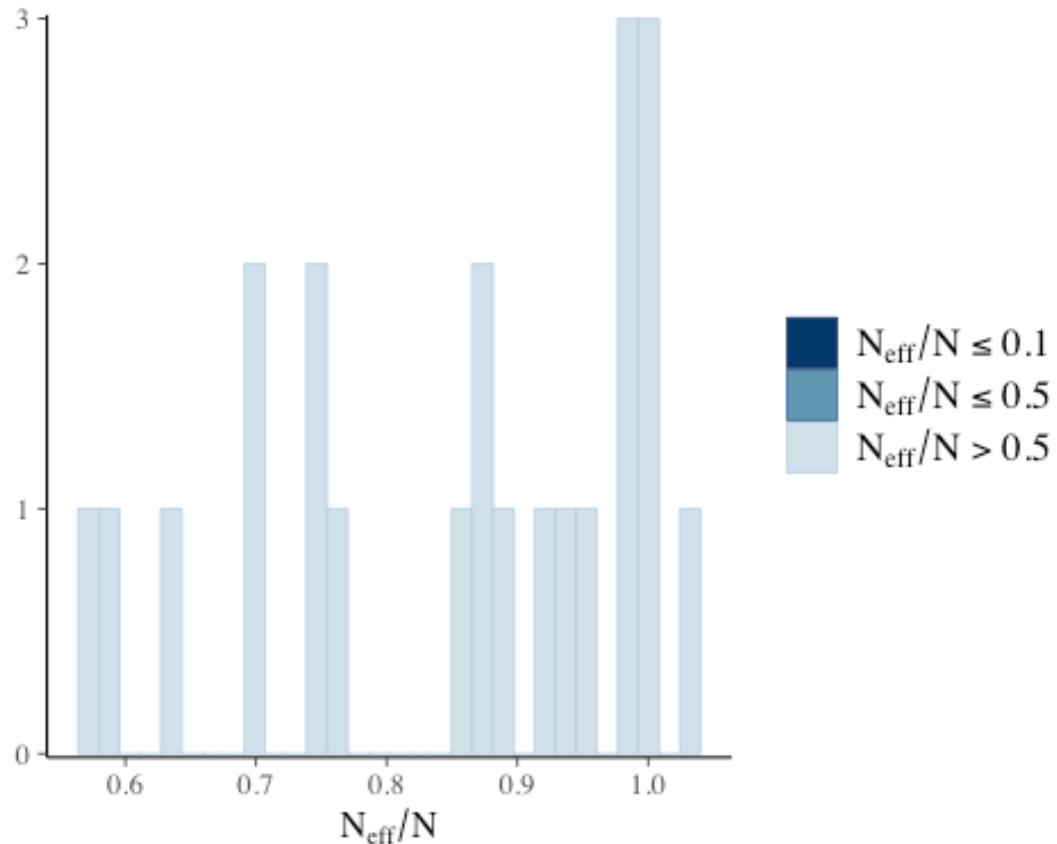
Measure of chain convergence - R hat

```
plot(tobacco.rstanarm1,"rhat_hist")  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Number of effective samples

```
plot(tobacco.rstanarm1, "neff_hist")  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



DHARMA residuals (most important part of the model validation)

```

preds<-posterior_predict(tobacco.rstanarm1, nsamples=250, summary=FALSE)

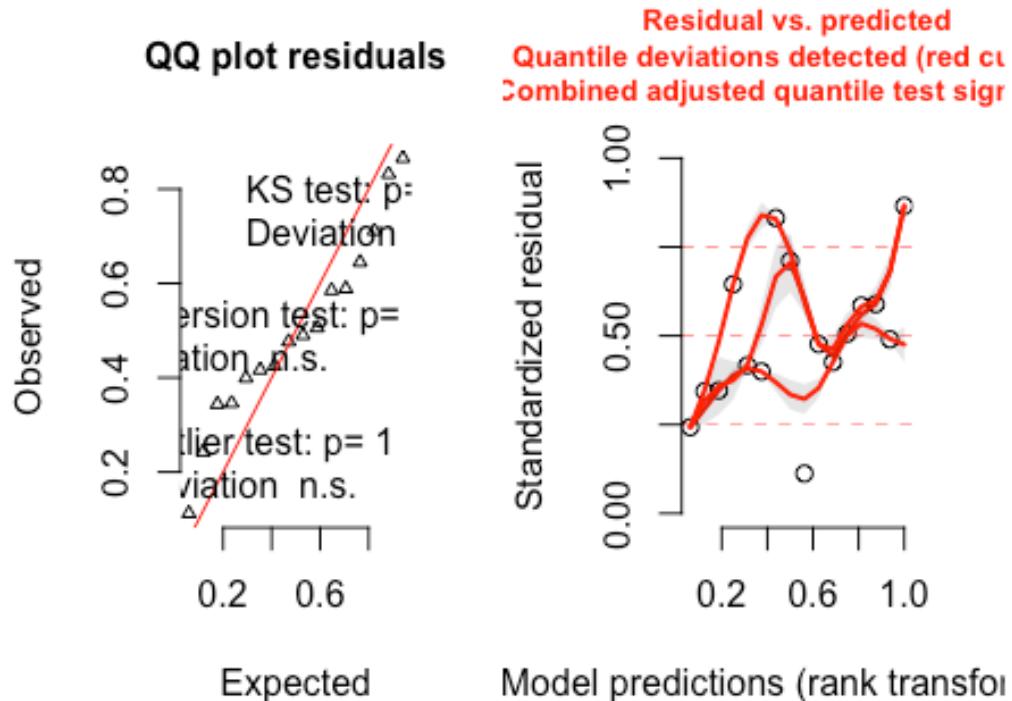
tobacco.resids<-createDHARMA(simulatedResponse = t(preds),
                                observedResponse = tobacco$NUMBER,
                                fittedPredictedResponse = apply(preds, 2, median),
                                # 2 = we tell R to take preds and apply it to
the rows (1) or to the columns (2)
                                integerResponse = FALSE)

plot(tobacco.resids)

## qu = 0.5, log(sigma) = -2.544224 : outer Newton did not converge fully.
## qu = 0.5, log(sigma) = -3.175997 : outer Newton did not converge fully.

```

DHARMA residual diagnostics



If R does the Quantile regressions, we might get the Residual vs Predicted plot all in red. Murray says we shouldn't pay attention to it, and interpret it as if it was a residual plot (see if the residuals are homogeneously distributed on both sides of the line). The quantiles are partitioning the data into 3 bins, so we got very few data to try and do the DHARMA residuals. What I am probably finding is that we got data clustered close to the upper and lower quantiles (see x axis ~0.8, y~0.5). The DHARMA will try to fit the residuals around those clustered data points, and give you an error. In other words, this error is just a reflection of our small sample size, but otherwise the residuals look ok.

(!) COMPARING MODELS - NOT NEEDED

WE COULD NOW ASK FOR THE MOST PARSIMONIOUS APPROACH, FITTING MORE MODELS AND COMPARING THEM (SAME AS IN FREQUENTIST). BUT THERE IS NOT REALLY THE NEED TO DO THIS IN BAYESIAN IF OUR MODEL WAS OK IN THE MODEL VALIDATION

FYI, though, Murray shows us what a comparison btw models would tell us:

loo (not needed, really)

Model 2 (random intercept random slopes model)

```
(1.1<-loo:::loo(tobacco.rstanarm1))
```

```
## Warning: Found 3 observation(s) with a pareto_k > 0.7. We recommend
calling 'loo' again with argument 'k_threshold = 0.7' in order to calculate
the ELPD without the assumption that these observations are negligible. This
will refit the model 3 times to compute the ELPDs for the problematic
observations directly.
```

```
##
## Computed from 1800 by 16 log-likelihood matrix
##
##           Estimate   SE
## elpd_loo     -50.2 2.9
## p_loo        7.7 1.8
## looic       100.4 5.8
## -----
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                               Count Pct.  Min. n_eff
## (-Inf, 0.5]    (good)      3  18.8%  212
## (0.5, 0.7]    (ok)       10  62.5%  143
## (0.7, 1]      (bad)       2  12.5%  398
## (1, Inf)     (very bad)  1  6.2%   13
## See help('pareto-k-diagnostic') for details.
```

1.1

```
##
## Computed from 1800 by 16 log-likelihood matrix
##
##           Estimate   SE
## elpd_loo     -50.2 2.9
## p_loo        7.7 1.8
## looic       100.4 5.8
## -----
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                               Count Pct.  Min. n_eff
## (-Inf, 0.5]    (good)      3  18.8%  212
## (0.5, 0.7]    (ok)       10  62.5%  143
## (0.7, 1]      (bad)       2  12.5%  398
## (1, Inf)     (very bad)  1  6.2%   13
## See help('pareto-k-diagnostic') for details.
```

This warning message indicates: Loo identified that we have a LOT of bad samples. loo is mimicking one sample out each time, so if we have too many of those, we might have a warning message. loo might then decide that it wants to rerun from some of those.

Info: Putting brackets around the command prints the output directly, it's a shortcut:

```
(l.1<-loo::loo(tobacco.rstanarm1))
```

This is the same as writing l.1

We'd have to set a k threshold to ~0.9 if that happens:

```
(1.1<-loo::loo(tobacco.rstanarm1, k_threshold=0.9))

## 1 problematic observation(s) found.
## Model will be refit 1 times.

##
## Fitting model 1 out of 1 (leaving out observation 7)

##
## Computed from 1800 by 16 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo    -50.3 3.0
## p_loo       7.8 1.9
## looic     100.5 5.9
## -----
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                               Count Pct.  Min. n_eff
## (-Inf, 0.5]    (good)      3  20.0%  212
## (0.5, 0.7]    (ok)        10 66.7%  143
## (0.7, 1]      (bad)       2 13.3%  398
## (1, Inf)     (very bad)  0  0.0% <NA>
## See help('pareto-k-diagnostic') for details.
```

Although loo still identifies the samples are bad, it will get going

```
1.2<-loo::loo(tobacco.rstanarm1)

## Warning: Found 3 observation(s) with a pareto_k > 0.7. We recommend
## calling 'loo' again with argument 'k_threshold = 0.7' in order to calculate
## the ELPD without the assumption that these observations are negligible. This
## will refit the model 3 times to compute the ELPDs for the problematic
## observations directly.
```

```
1.2
```

```

## 
## Computed from 1800 by 16 log-likelihood matrix
## 
##           Estimate SE
## elpd_loo    -50.2 2.9
## p_loo       7.7 1.8
## looic     100.4 5.8
## -----
## Monte Carlo SE of elpd_loo is NA.
## 
## Pareto k diagnostic values:
##                               Count Pct.   Min. n_eff
## (-Inf, 0.5]   (good)      3   18.8%   212
## (0.5, 0.7]   (ok)        10  62.5%   143
## (0.7, 1]     (bad)       2   12.5%   398
## (1, Inf)     (very bad) 1   6.2%    13
## See help('pareto-k-diagnostic') for details.

```

Model 1 (initial model, random intercept model)

```

1.3<-loo::loo(tobacco.rstanarm)

## Warning: Found 1 observation(s) with a pareto_k > 0.7. We recommend
## calling 'loo' again with argument 'k_threshold = 0.7' in order to calculate
## the ELPD without the assumption that these observations are negligible. This
## will refit the model 1 times to compute the ELPDs for the problematic
## observations directly.

1.3

## 
## Computed from 1800 by 16 log-likelihood matrix
## 
##           Estimate SE
## elpd_loo    -50.5 2.7
## p_loo       6.3 1.5
## looic     101.0 5.4
## -----
## Monte Carlo SE of elpd_loo is NA.
## 
## Pareto k diagnostic values:
##                               Count Pct.   Min. n_eff
## (-Inf, 0.5]   (good)      5   31.2%   336
## (0.5, 0.7]   (ok)        10  62.5%   207
## (0.7, 1]     (bad)       1   6.2%    72
## (1, Inf)     (very bad) 0   0.0%   <NA>
## See help('pareto-k-diagnostic') for details.

```

The looic estimates are the same in model 1 and 2. As you can see, it does not matter whether we fit a random intercept model (Model 1) or a random intercept random slope model (Model 2).

=> WE WILL THEREFORE SELECT THE SIMPLEST MODEL, AND PROCEED WITH THAT ONE. HOWEVER, WE COULD ALSO HAVE JUST GOEN WITH THE MORE COMPLEX MODEL AS INITIALLY PLANNED (NOT AN ISSUE IN BAYESIAN)

Model investigation / hypothesis testing

```
broom.mixed::tidyMCMC(tobacco.rstanarm$stanfit, conf.int=TRUE, conf.method="HPD
interval",
                       rhat=TRUE, ess=TRUE, estimate.method="median") %>%
  data.frame

##                                     term   estimate std.error
conf.low
## 1                         (Intercept) 34.99120441 2.103135
3.098363e+01
## 2                         TREATMENTWeak -7.85765245 2.325699 -
1.223065e+01
## 3                         b[(Intercept) LEAF:L1] -0.15476144 2.297339 -
4.701309e+00
## 4                         b[(Intercept) LEAF:L2] -0.80506064 2.377459 -
5.947224e+00
## 5                         b[(Intercept) LEAF:L3] -0.28159789 2.256539 -
5.210922e+00
## 6                         b[(Intercept) LEAF:L4] -1.30123253 2.576196 -
7.571036e+00
## 7                         b[(Intercept) LEAF:L5] -0.52655225 2.378366 -
5.834781e+00
## 8                         b[(Intercept) LEAF:L6] 2.06162560 2.795544 -
1.795366e+00
## 9                         b[(Intercept) LEAF:L7] 3.59203725 3.290399 -
1.288193e+00
## 10                        b[(Intercept) LEAF:L8] -1.40141642 2.581801 -
7.496531e+00
## 11                         b[(Intercept) LEAF:_NEW_LEAF] -0.04314517 3.655672 -
8.192313e+00
## 12                               sigma 4.38349852 1.208937
2.502161e+00
## 13 Sigma[LEAF:(Intercept),(Intercept)] 8.87726992 14.599263 1.857344e-
06
## 14                               mean_PPD 31.02945038 1.666450
2.760429e+01
## 15                           log-posterior -65.90250543 3.886274 -
7.340199e+01
##     conf.high      rhat    ess
## 1 39.282895 1.0009945 1832
## 2 -3.017688 0.9994372 1848
## 3 4.902786 1.0002604 1592
## 4 3.989200 0.9986506 1692
## 5 3.967644 0.9998308 1775
```

```
## 6   2.384128 1.0000404 1833
## 7   3.730850 1.0015883 1844
## 8   8.658034 1.0004076 1467
## 9   10.092517 0.9996898 1571
## 10  2.545908 1.0009218 1703
## 11  7.335327 0.9992970 1803
## 12  6.774364 0.9995997 1387
## 13  39.017563 0.9996945 1505
## 14  34.207460 1.0001059 1885
## 15 -59.010853 0.9998157 1288
```

- TREATMENTWeak => The conf int does not intercept with zero = there are sign more lesions in the strong inoculation than in the weak. On avg, there are 7.7 more lesions (these are absolute values, as we did not log the data) in the STRONG Treatment (Intercept) than in the weak Treatment
- Is the variability stronger between or within the leaves? We have to check the following rows sigma row = within leaves variability -> has less variability (4.3 estimate) sigma[LEAF:(Intercept),(Intercept)] row = between leaves variability -> has more variability (9.6 estimate)

R square: marginal and conditional

Marginal R sq

Examines the differences WITHIN leaves (cause each leave has got one side with strong treatment and one side with weak treatment)

```
bayes_R2(tobacco.rstanarm,re.form=NA) %>%
  median_hdci

##           y      ymin      ymax .width .point .interval
## 1 0.4703604 0.143053 0.7811332    0.95 median      hdci
```

In frequentist, we get a marginal and a conditional Rsq. In this case, in the random effects formula, when we say that re=NA we are defining the marginal Rs, the R sq associated with the treatments.

The treatment effects alone explain 47% of the variability

Conditional R sq

Examines the differences BETWEEN leaves (=the variability of the whole experiments, essentially)

```
bayes_R2(tobacco.rstanarm,re.form=~(1|LEAF)) %>%
  median_hdci

##           y      ymin      ymax .width .point .interval
## 1 0.5794129 0.2081239 0.855811    0.95 median      hdci
```

When you account for the leaves, we can in total explain 59%. There is a diff btw leaves within their lesions, and btw treatments. In our case, both treatment (marginal Rsq) and leaves (conditional Rsq) have a massive effect.

If we had a conditional R sq of let's say 0.8, and a marginal R sq of 0.2, it would tell us that treatments are not that important, and don't account for much. The variability would essentially be explained by the leaves (subjects)

Predictions

Creating the predicted data from our model

This is what the predicted data would look like without spreading it for TREATMENT

```
newdata=emmeans(tobacco.rstanarm, ~TREATMENT) %>%
  gather_emmeans_draws()

head(newdata)

## # A tibble: 6 x 5
## # Groups:   TREATMENT [1]
##   TREATMENT .chain .iteration .draw .value
##   <fct>     <int>     <int> <int>  <dbl>
## 1 Strong      NA        NA     1    31.4
## 2 Strong      NA        NA     2    35.9
## 3 Strong      NA        NA     3    36.2
## 4 Strong      NA        NA     4    33.0
## 5 Strong      NA        NA     5    35.6
## 6 Strong      NA        NA     6    35.9
```

And this is spreading it for TREATMENT (more informative option)

```
newdata=emmeans(tobacco.rstanarm, ~TREATMENT) %>%
  gather_emmeans_draws() %>%
  spread(key=TREATMENT,value=.value)

head(newdata)

## # A tibble: 6 x 5
##   .chain .iteration .draw Strong Weak
##   <int>     <int> <int> <dbl> <dbl>
## 1 NA        NA     1    31.4  26.7
## 2 NA        NA     2    35.9  28.9
## 3 NA        NA     3    36.2  28.4
## 4 NA        NA     4    33.0  24.4
## 5 NA        NA     5    35.6  27.8
## 6 NA        NA     6    35.9  27.5
```

Absolute and percentage differences

Lets add a column called Eff for effect, which indicates the absolute and percentage differences btw Strong and Weak

```
newdata=newdata %>%
  mutate(Eff=Strong-Weak, # absolute difference
        PEff=100*(Strong-Weak)/Weak) # percentage difference

head(newdata)

## # A tibble: 6 x 7
##   .chain .iteration .draw Strong  Weak    Eff  PEff
##   <int>      <int> <int>  <dbl> <dbl> <dbl> <dbl>
## 1     NA        NA     1   31.4  26.7  4.74 17.8
## 2     NA        NA     2   35.9  28.9  7.02 24.3
## 3     NA        NA     3   36.2  28.4  7.78 27.3
## 4     NA        NA     4   33.0  24.4  8.65 35.5
## 5     NA        NA     5   35.6  27.8  7.87 28.3
## 6     NA        NA     6   35.9  27.5  8.48 30.9
```

Now, we can see the diff btw treatments for each leaf (ID=.draw)

Avg absolute eff and avg perc effect

We could just calculate the median of the two columns, and that would tell us on avg how many more lesions to expect or how many percentage more lesions to expect:

```
newdata %>% dplyr::select(Eff,PEff) %>%
  median_hdci

## # A tibble: 1 x 9
##   Eff  Eff.lower  Eff.upper  PEff PEff.lower PEff.upper .width .point
##   <dbl>    <dbl>     <dbl> <dbl>    <dbl>     <dbl>    <dbl> <chr>
## 1  7.86     3.02     12.2  29.1     9.97     49.7    0.95 median hdci
```

What is the prob that there are more lesions in the strong treatment than the weak treatment?

We just have to identify the rows of Eff (absolute values) or PEff (percentage values) that are >0.

PEff

```
newdata %>% summarize(Prob=sum(PEff>0)/n()) # remember that PEff is already
in percentage, so we'd specify PEff>percentage value (0-100)

## # A tibble: 1 x 1
##   Prob
```

```
## <dbl>
## 1 0.999
```

What is the prob that there are more than 10% of lesions in the strong treatment than the weak treatment?

PEff

```
newdata %>% summarize(Prob=sum(PEff>10)/n()) # remember that PEff is already
in percentage, so we'd specify PEff>percentage value (0-100)

## # A tibble: 1 × 1
##       Prob
##   <dbl>
## 1 0.986
```

Eff

IF we were working with ODD's RATIO (percent change), on a LOG scale, ONLY THEN we could define the following:

```
newdata %>% summarize(Prob=sum(Eff>1.1)/n()) # we'd specify Eff>absolute
value (1.1 = >10% greater)

## # A tibble: 1 × 1
##       Prob
##   <dbl>
## 1 0.999

# THIS IS NOT CORRECT FOR OUR MODEL THOUGH, AS WE ARE WORKING IN ABSOLUTE
VALUES (NOT percentage, or odds ratio).
```

BUT BECAUSE WE ARE WORKING WITH ABSOLUTE VALUES (Eff), this code above essentially tells R: " What is the prob that I have 1.1 more lesions in the strong treatment, compared to the weak treatment? "

IN SUMMARY: When defining percent change in BAYESIAN (not using Odds ratio and log scale), we CAN ONLY calculate percentage change from the column which ALREADY incorporated percentages. In this case, that is PEff.

Summary figures

WE'LL CONTINUE AS WE DID WITH THE OTHER EXAMPLES

References

20_Bayesian_Repeated_Measures

Sara Kophamel

17/12/2020

Preparations

Load the necessary libraries

```
library(car)      #for regression diagnostics
library(broom)    #for tidy output
library(ggfortify) #for model diagnostics
library(sjPlot)   #for outputs
library(knitr)    #for kable
library(effects)  #for partial effects plots
library(ggeffects) #for partial effects plots
library(emmeans)  #for estimating marginal means
library(MASS)     #for glm.nb
library(MuMIn)    #for AICc
library(broom.mixed) # for tidying outputs
library(rstanarm) #for fitting models in STAN
library(tidybayes) #for more tidying outputs
library(tidyverse) #for data wrangling
library(nlme)
library(lme4)
library(glmmTMB)
library(broom.mixed)
library(glmmTMB)  #for glmmTMB
library(DHARMa)   #for residuals and diagnostics
library(performance) #for diagnostic plots
library(see)       #for diagnostic plots
```

Scenario

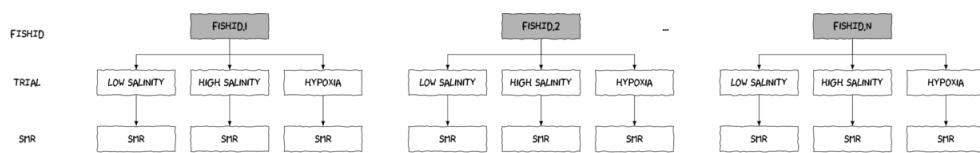
To investigate differential metabolic plasticity in barramundi (*Lates calcarifer*), @Norin-2016-369 exposed juvenile barramundi to various environmental changes (increased temperature, decreased salinity and increased hypoxia) as well as control conditions. Metabolic plasticity was calculated as the percentage difference in standard metabolic rate between the various treatment conditions and the standard metabolic rate under control conditions. They were interested in whether there was a relationship between metabolic plasticity and typical (control) metabolism and how the different treatment conditions impact on this relationship. They also measured MASS, as this could potentially influence metabolism.

A total of 60 barramundi juveniles were subject to EACH of the three conditions (high temperature, low salinity and hypoxia) in addition to control conditions. Fish mass was also recorded as a covariate as this is known to influence metabolic parameters. => We got a BLOCKING design, because each treatment is applied to each block (=fish).

In other words, we'd be interested in the three trials and the interaction btw the three of them (example below) $y=\text{change} \sim T1\ T2\ T3\ | \dots | \dots | \dots | \dots$ > x=metabolism (SMR)

Barramundi

Barramundi



Sampling design

Format of norin.csv data files

FISHID	MASS	TRIAL	SMR_contr	CHANGE
1	35.69	LowSalinity	5.85	-31.92
2	33.84	LowSalinity	6.53	2.52
3	37.78	LowSalinity	5.66	-6.28
..
1	36.80	HighTemperature	5.85	18.32
2	34.98	HighTemperature	6.53	19.06
3	38.38	HighTemperature	5.66	19.03
..
1	45.06	Hypoxia	5.85	-18.61
2	43.51	Hypoxia	6.53	-5.37
3	45.11	Hypoxia	5.66	-13.95

WE CAN HAVE NEGATIVES, SO WE WILL GO FOR A GAUSSIAN (NORMAL) DISTRIBUTION.

- FISHID** Categorical listing of the individual fish that are repeatedly sampled
- MASS** Mass (g) of barramundi. Covariate in analysis
- TRIAL** Categorical listing of the trial (LowSalinity: 10ppt salinity; HighTemperature: 35 degrees; Hypoxia: 45% air-sat. oxygen.)
- SMR_contr** Standard metabolic rate (mg/h/39.4 g of fish) under control trial conditions (35 ppt salinity, 29 degrees, normoxia)
- CHANGE** Percentage difference in Standard metabolic rate (mg/h/39.4 g of fish)

between Trial conditions and control adjusted for 'regression to the mean'.

Read in the data

```
norin = read_csv('data/norin.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   FISHID = col_double(),
##   MASS = col_double(),
##   TRIAL = col_character(),
##   SMR_contr = col_double(),
##   CHANGE = col_double()
## )

glimpse(norin)

## Rows: 180
## Columns: 5
## $ FISHID      <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
## 17, ...
## $ MASS        <dbl> 35.69, 33.84, 37.78, 26.58, 37.62, 37.68, 30.62, 50.37,
## 24...
## $ TRIAL       <chr> "LowSalinity", "LowSalinity", "LowSalinity",
## "LowSalinity",...
## $ SMR_contr  <dbl> 5.847466, 6.530707, 5.659556, 6.278200, 4.407336,
## 4.818589, ...
## $ CHANGE      <dbl> -31.919389, 2.520929, -6.284968, -4.346675, -3.071329, -
## 15...
```

Change chr to factor variables

If we had >2 treatments, we could also order the variables (as we did in the example with the seasons, 18_Bayesian)

```
norin=norin %>%
  mutate(FISHID=factor(FISHID),
        TRIAL=factor(TRIAL))

glimpse(norin)

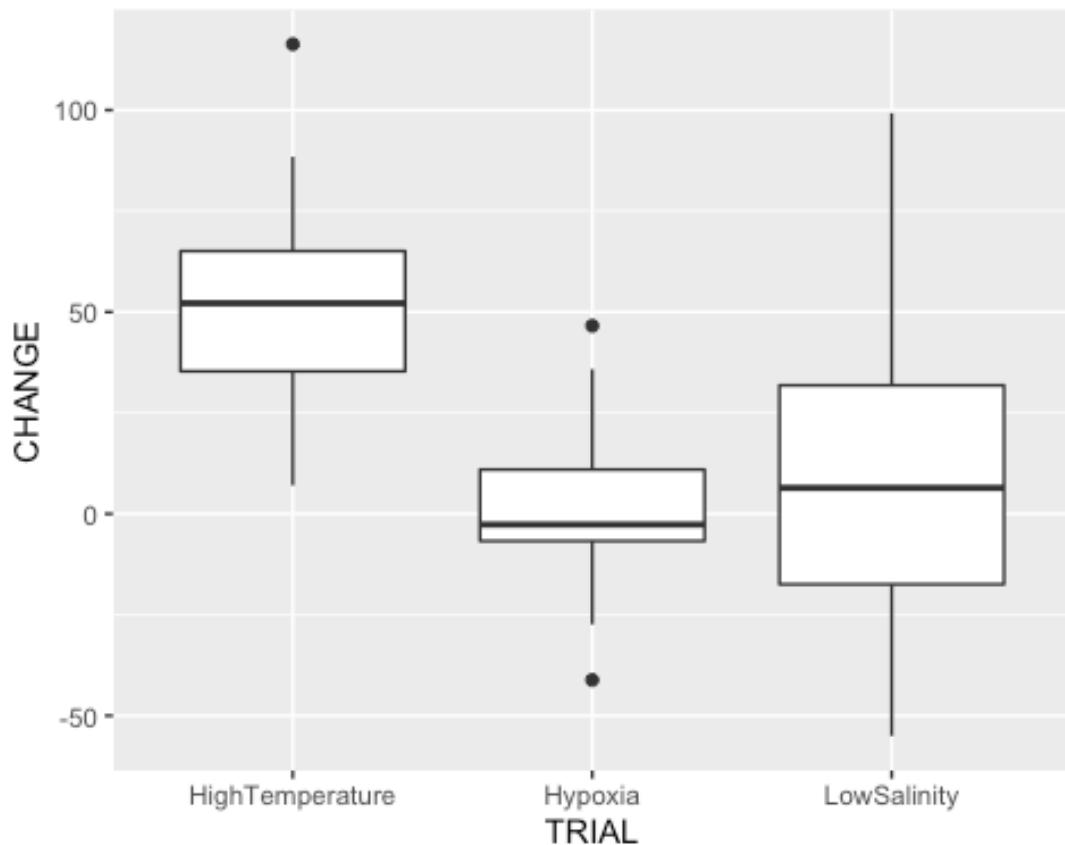
## Rows: 180
## Columns: 5
## $ FISHID      <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
## 17, ...
## $ MASS        <dbl> 35.69, 33.84, 37.78, 26.58, 37.62, 37.68, 30.62, 50.37,
## 24...
## $ TRIAL       <fct> LowSalinity, LowSalinity, LowSalinity, LowSalinity,
## LowSal...
## $ SMR_contr  <dbl> 5.847466, 6.530707, 5.659556, 6.278200, 4.407336,
```

```
4.818589,...  
## $ CHANGE      <dbl> -31.919389, 2.520929, -6.284968, -4.346675, -3.071329, -  
15....
```

Exploratory data analysis

Boxplots for differences in CHANGE across Treatment

```
ggplot(norin, aes(x=TRIAL,y=CHANGE))+  
  geom_boxplot()
```



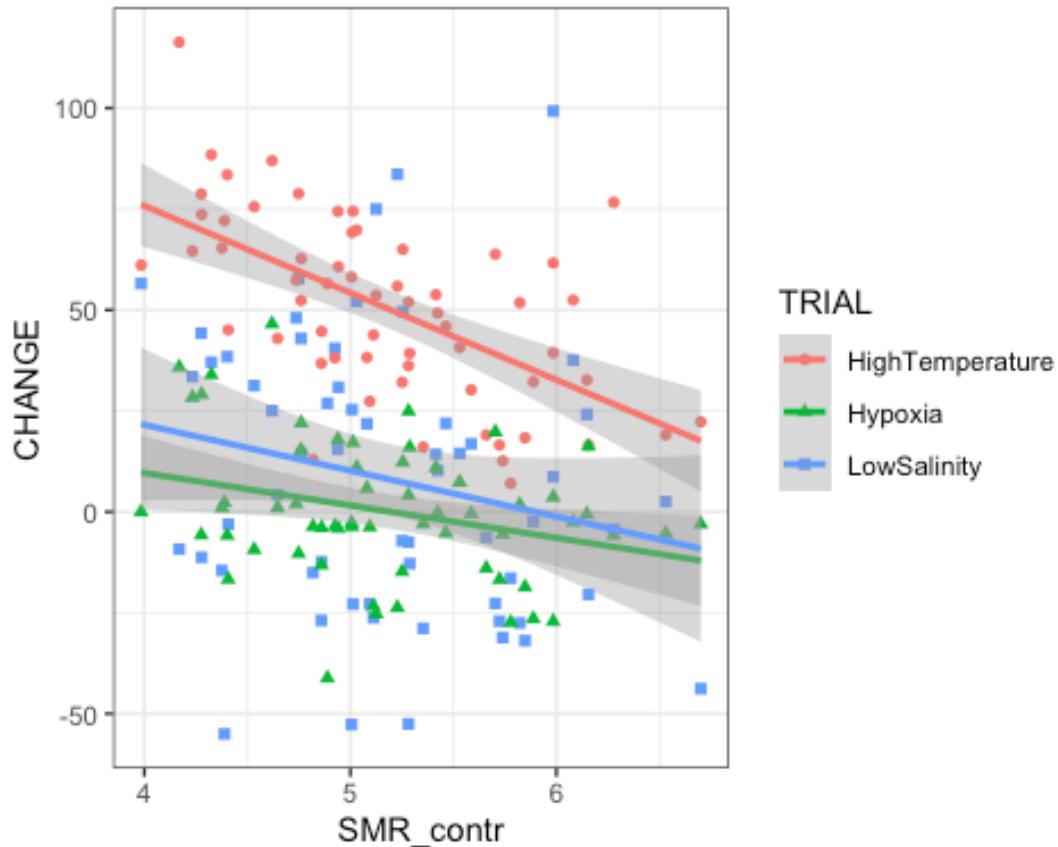
Normality looks ok - Homogeneity of var looks ok. You don't want to see a relationship btw mean and variance = smaller boxplots low down and bigger boxplots higher up (!)

AS WE GOT NEG AND POS VALUES, WE CAN PRETTY MUCH ONLY USE A GAUSSIAN DISTRIBUTION.

Scatterplot for differences in Metabolic rate

```
ggplot(norin, aes(y=CHANGE,x=SMR_contr, shape=TRIAL, color=TRIAL))+  
  geom_point() +  
  #geom_line() +  
  geom_smooth(method="lm") +  
  theme_bw()
```

```
## `geom_smooth()` using formula 'y ~ x'
```



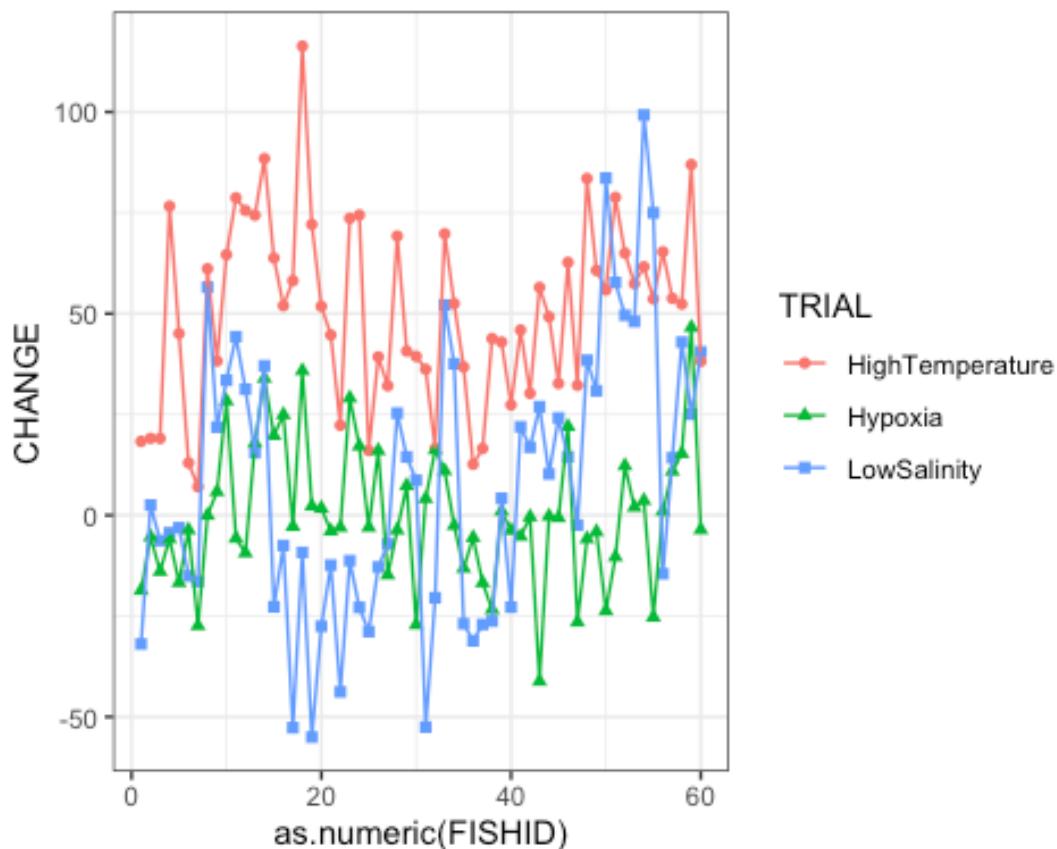
- Our data seems to have a linear trend. - Homogeneity of variance (spread of points around the lines) looks uniform, which is good.

What if our lines are not parallel (which seems to be the case)? You don't have to worry about data converging (=what seems to happen if you extend the x axis), as you can ONLY make conclusions about the x axis values you actually assessed. If you inflated the axes, you'd have a statistical artifact. - In frequentist: If we created an additive model, without interaction, we'd give the remaining, more complex models, less power, as we'd be adding df. However, in Bayesian we don't have to worry about creating more complex models from the very beginning. So let's assume an INTERACTION.

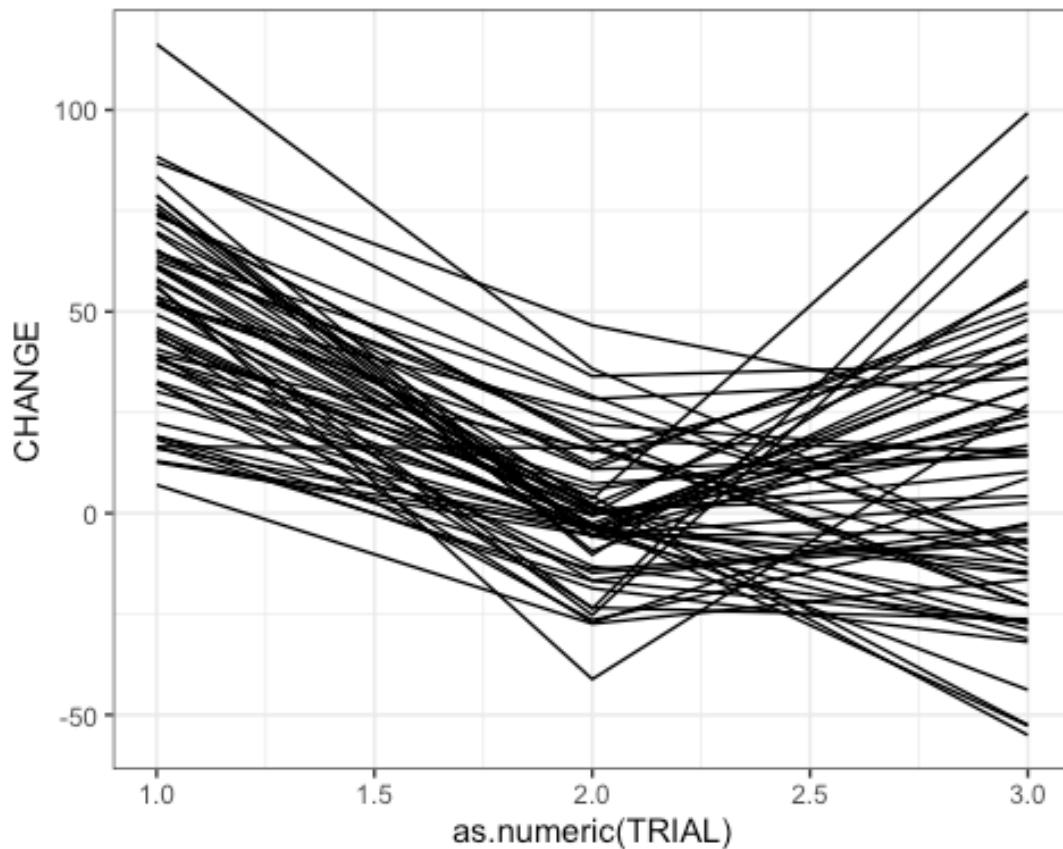
Other plots that we could check

Line plot for differences in Fish ID

```
ggplot(norin, aes(y=CHANGE,x=as.numeric(FISHID), shape=TRIAL, color=TRIAL))+
  geom_point()+
  geom_line()+
  #geom_smooth(method="Lm")+
  theme_bw()
```



```
ggplot(norin, aes(y=CHANGE,x=as.numeric(TRIAL)))+
  #geom_point()+
  geom_line(aes(group=FISHID))+
  #geom_smooth(method="Lm")+
  theme_bw()
```



Although the x axis is in numeric, I can't see that value 1 is low salinity, 2 is high temp, and 3 is hypoxia. We SEE that the intercepts DO VARY by fish. We already thought they might. But, importantly, we might want to know whether the effects also differ per fish = "Do they all respond in a similar way to the diff treatments?" And what we can see is that, by large, they sort of do, but there is a bit of variation btw the lines as well. The change value might be less for the middle group (high temp), and then they might all go up for hypoxia. In that case I might not need a random intercept random slope model. BUT it might be useful. And it does not hurt, as Bayesian does not penalise the model for its complexity. We'd be sure we'd need a random int random slope model if we had the following trends:

`^ Fish1 | . / / Fish 3 | / / | / . | . Fish2 | _____ > x=TRIAL` If we see that fishes behave in very diff ways, we'd go for sure with a rand int rand slope model

Model formula:

$$\$y_i \sim \mathcal{N}(\mu_i, \sigma^2) \quad \mu_i = \boldsymbol{\beta} \bf{X}_i + \boldsymbol{\gamma} \bf{Z}_i$$

where β and γ are vectors of the fixed and random effects parameters respectively and \bf{X} is the model matrix representing the overall intercept and effects of temperature and (centered) mean fish size on SDA peak. \bf{Z} represents a cell means model matrix for the random intercepts associated with individual fish.

Fit the model

1- Random intercept model (FYI)

- Check the scatterplot with the lines again: We need to define the prior intercept: Intercept is the value of y where x=0. In Bayesian models, the predictors are CENTERED. In this case, the priors are on the PARAMETERS, not just on the data. SO our center point is ~ 50 x values. So we'd select the intercept prior value for ~50

For each of the following modelling alternatives, we will:

- establish whether MASS is a useful covariate (based on AICc). This step involves evaluating alternative fixed effects structures. When doing so, we must use Maximum Likelihood (ML) rather than Residual Maximum Likelihood (REML).
- establish whether a random intercept/slope model is useful (based on AICc). This step involves evaluating alternative random effects structures. When doing so, we should use Restricted Maximum Likelihood (REML)

Model for priors

```
norin.rstanarm=rstanarm::stan_glmer(CHANGE~(1|FISHID)+TRIAL*SMR_contr+MASS, #  
MASS is our offset. It soaks up any additional noise (we know it influences  
metabolism), that is why we are not defining that it has an interaction with  
anything
```

```
data=norin,  
prior_PD=TRUE, # we first only tell the model to be  
defined by the priors  
family=gaussian, # we define we got a Gaussian  
family  
iter=5000,  
warmup=2000, # for Gaussian models  
chains=3,  
thin=5,  
refresh=0)
```

- Why did we include MASS? If we have sth we are not really interested in, but that we know is gonna change our response, we need to account for it, and we'd put it as an Offset (as in the frequentist style), with a "+". An offset is accounted for, but is not interacting with the other variables
- If you have individuals, because they will react in different ways, you also have to include it as random intercept. Fish is considered as RANDOM intercept cause it's the same fish in all trials.

To see what the model is defining:

```
args(rstanarm::stan_glmer)  
  
## function (formula, data = NULL, family = gaussian, subset, weights,  
## na.action = getOption("na.action", "na.omit"), offset, contrasts =
```

```

NULL,
##     ..., prior = default_prior_coef(family), prior_intercept =
default_prior_intercept(family),
##     prior_aux = exponential(autoscale = TRUE), prior_covariance = decov(),
##     prior_PD = FALSE, algorithm = c("sampling", "meanfield",
##         "fullrank"), adapt_delta = NULL, QR = FALSE, sparse = FALSE)
## NULL

```

To see the different levels of TRIAL

```

levels(norin$TRIAL)

## [1] "HighTemperature" "Hypoxia"           "LowSalinity"

```

Prior summary

```

prior_summary(norin.rstanarm)

## Priors for model 'norin.rstanarm'
## -----
## Intercept (after predictors centered)
##   Specified prior:
##     ~ normal(location = 20, scale = 2.5)
##   Adjusted prior:
##     ~ normal(location = 20, scale = 85)
##
## Coefficients
##   Specified prior:
##     ~ normal(location = [0,0,0,...], scale = [2.5,2.5,2.5,...])
##   Adjusted prior:
##     ~ normal(location = [0,0,0,...], scale = [178.99,178.99,135.15,...])
##
## Auxiliary (sigma)
##   Specified prior:
##     ~ exponential(rate = 1)
##   Adjusted prior:
##     ~ exponential(rate = 0.03)
##
## Covariance
##   ~ decov(reg. = 1, conc. = 1, shape = 1, scale = 1)
## -----
## See help('prior_summary.stanreg') for more details

```

Let's examine the priors for the model 'norin.rstanarm':

Intercept (after predictors centered) Specified prior: $\sim \text{normal}(\text{location} = 20, \text{scale} = 2.5)$
 Adjusted prior: $\sim \text{normal}(\text{location} = 20, \text{scale} = 85)$ -> The model is using 20 as the prior intercept value. 50 would have been more appropriate (see explanation under Section "Fit the model")

Coefficients Specified prior: $\sim \text{normal}(\text{location} = [0,0,0,...], \text{scale} = [2.5,2.5,2.5,...])$ Adjusted prior: $\sim \text{normal}(\text{location} = [0,0,0,...], \text{scale} = [178.99,178.99,135.15,...])$ ->

[178.99, 178.99, 135.15,...] are the scales R has given the priors. It essentially tells us that the priors are diff. This will slow the model down. It would be faster if these brackets had a single value. Given that they are similar, it looks ok.

Auxiliary (sigma) Specified prior: $\sim \text{exponential}(\text{rate} = 1)$ Adjusted prior: $\sim \text{exponential}(\text{rate} = 0.03)$

Covariance

`decov(reg. = 1, conc. = 1, shape = 1, scale = 1)` -> Don't mess with this one. Keep these default values (!)

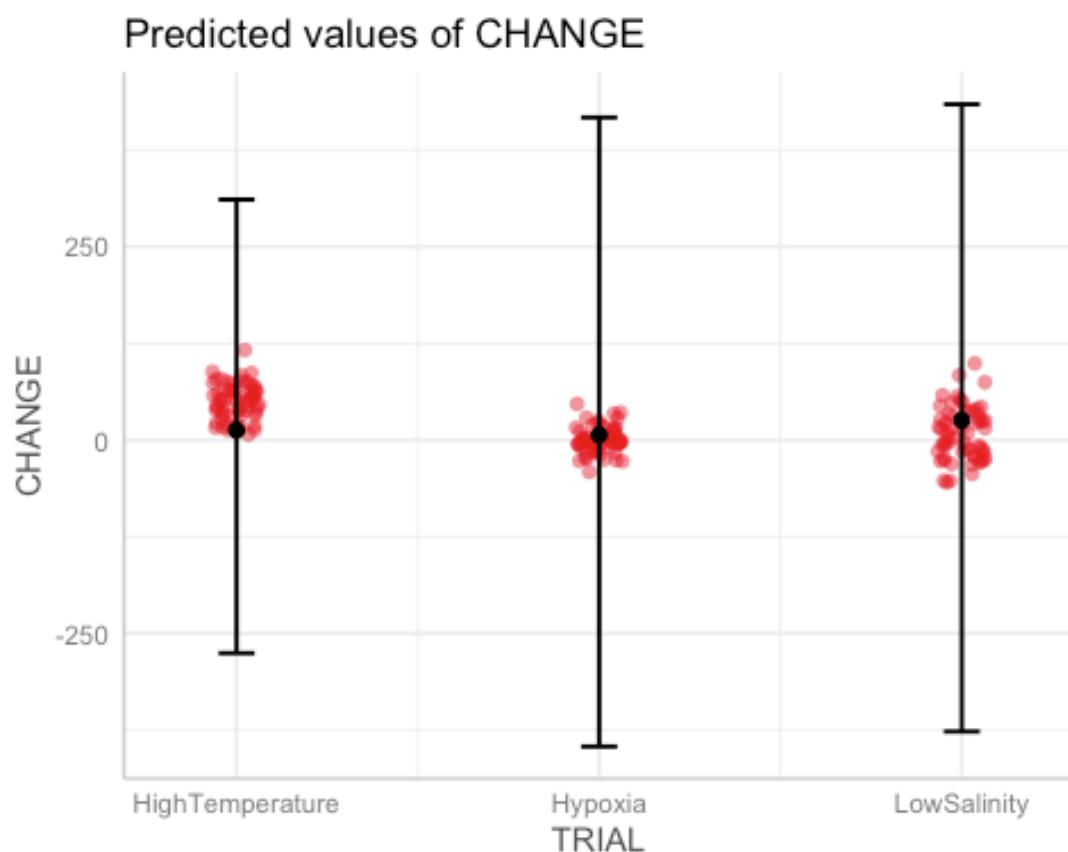
Prior predictions

What is the range of values that we could predict in the absence of our actual data?

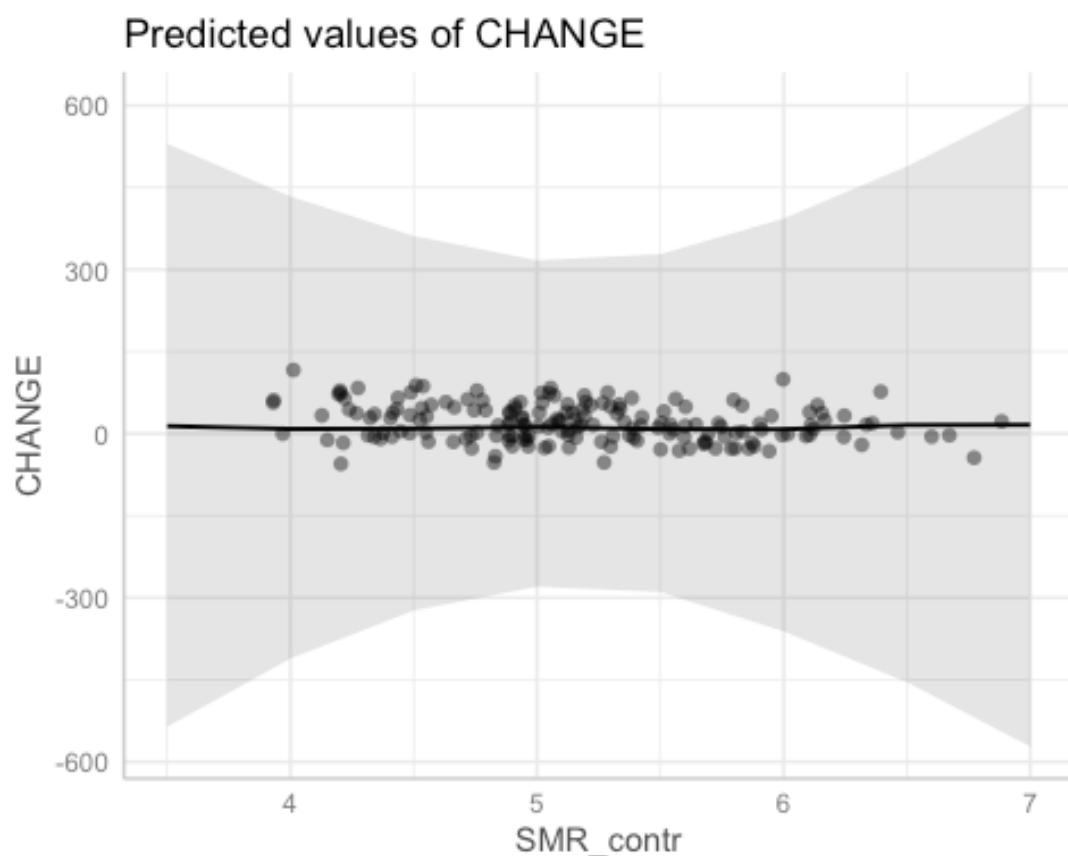
```
ggpredict(norin.rstanarm) %>%
  plot(add.data=TRUE)

## Note: uncertainty of error terms are not taken into account. You may want
## to use `rstantools::posterior_predict()` .
## Note: uncertainty of error terms are not taken into account. You may want
## to use `rstantools::posterior_predict()` .
## Note: uncertainty of error terms are not taken into account. You may want
## to use `rstantools::posterior_predict()` .

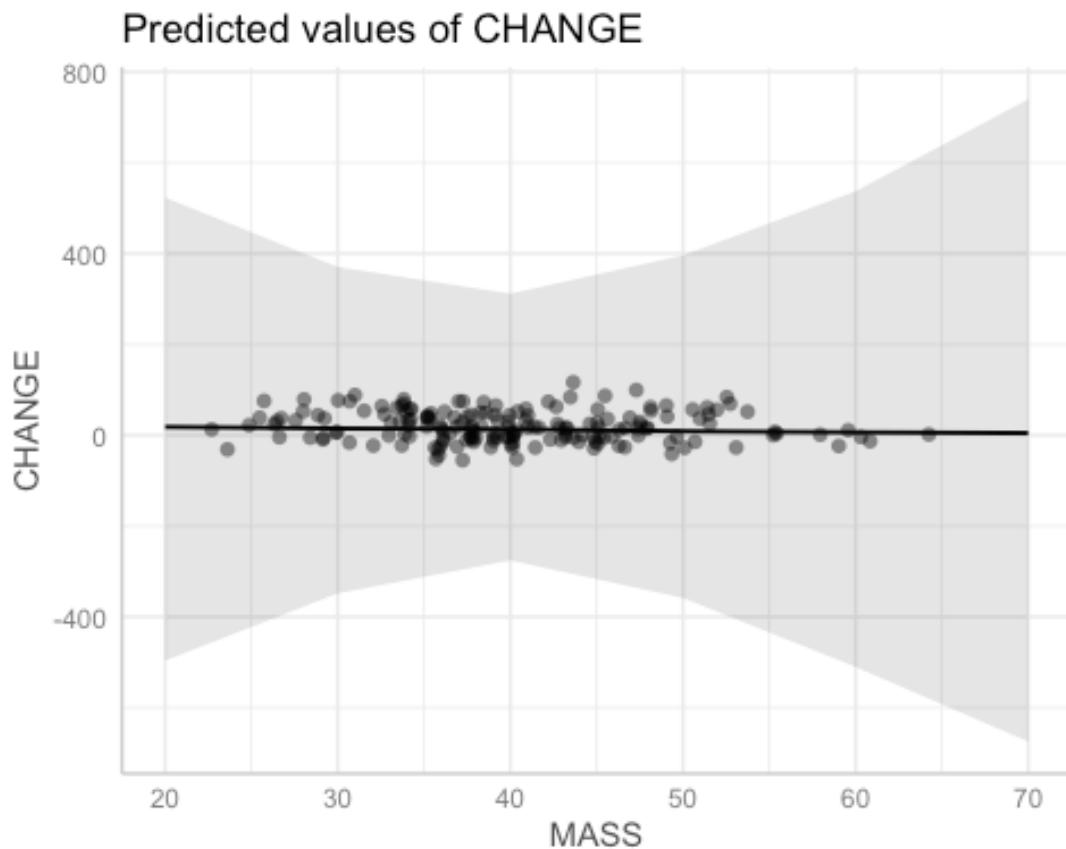
## $TRIAL
```



```
##  
## $SMR_contr
```



```
##  
## MASS
```



WE
COULD CONTINUE WITH THIS MODEL AND CHECK THE POSTERIORS AND PREDICTIONS,
BUT MURRAY PREFERS TO PICK A MORE COMPLEX MODEL TO SHOW US HOW IT WOULD
LOOK LIKE (WE WOULD IN THEORY HAVE PICKED THE MORE COMPLEX MODEL FROM
THE VERY BEGGINING, MODEL2)

2- Random intercept random slope model

- Check the scatterplot with the lines again: We need to define the prior intercept: Intercept is the value of y where x=0. In Bayesian models, the predictors are CENTERED. In this case, the priors are on the PARAMETERS, not just on the data. SO our center point is ~ 50 x values. So we'd select the intercept prior value for ~50

For each of the following modelling alternatives, we will:

- establish whether MASS is a useful covariate (based on AICc). This step involves evaluating alternative fixed effects structures. When doing so, we must use Maximum Likelihood (ML) rather than Residual Maximum Likelihood (REML).
- establish whether a random intercept/slope model is useful (based on AICc). This step involves evaluating alternative random effects structures. When doing so, we should use Restricted Maximum Likelihood (REML)

Model for priors

`norin.rstanarm1=rstanarm::stan_glmer(CHANGE~(SMR_contr|FISHID)+TRIAL*SMR_cont+r+MASS,` # This model looks at the corr btw slopes. We could force it to have an interaction of zero. In this case, bec we have a random int random slope model, we have to include another parameter that might change the slopes of FishID, such as SMR_contr, which effectively seems to alter the slopes (see the scatter plot we created at the very beginning) (!) Once we include a continuous variable (SMR_contr) in the slope, the model can work easier than if it was a categorical variable.

`data=norin,`
`prior_PD=TRUE, # we first only tell the model to be defined by the priors`
`family=gaussian, # we define we got a Gaussian family`
`iter=5000,`
`warmup=2000, # for Gaussian models`
`chains=3, # chains=3 tells R to run one chain, then finish, then run another`
`thin=5,`
`refresh=0,`
`cores=3) # cores=3 tells R to run all chains together (using all three cores of the computer), instead of doing the default chains=3 function`

- Why did we include MASS? If we have sth we are not really interested in, but that we know is gonna change our response, we need to account for it, and we'd put it as an Offset (as in the frequentist style), with a “+”. An offset is accounted for, but is not interacting with the other variables
- If you have individuals, because they will react in different ways, you also have to include it as random intercept. Fish is considered as RAMNDOM intercept cause it's the same fish in all trials. In this case, bec we have a random int random slope model, we have to include another parameter that might change the slopes of FishID, such as SMR_contr, which effectively seems to alter the slopes (see the scatter plot we created at the very beginning). (!)

Prior summary

```
prior_summary(norin.rstanarm1)

## Priors for model 'norin.rstanarm1'
## -----
## Intercept (after predictors centered)
##   Specified prior:
##     ~ normal(location = 20, scale = 2.5)
##   Adjusted prior:
##     ~ normal(location = 20, scale = 85)
##
## Coefficients
##   Specified prior:
```

```

##      ~ normal(location = [0,0,0,...], scale = [2.5,2.5,2.5,...])
## Adjusted prior:
##      ~ normal(location = [0,0,0,...], scale = [178.99,178.99,135.15,...])
##
## Auxiliary (sigma)
## Specified prior:
##      ~ exponential(rate = 1)
## Adjusted prior:
##      ~ exponential(rate = 0.03)
##
## Covariance
## ~ decov(reg. = 1, conc. = 1, shape = 1, scale = 1)
## -----
## See help('prior_summary.stanreg') for more details

```

Let's examine the priors for the model 'norin.rstanarm1' => In this model, the SAME priors as with the previous model have been selected.

Intercept (after predictors centered) Specified prior: $\sim \text{normal}(\text{location} = 20, \text{scale} = 2.5)$
 Adjusted prior: $\sim \text{normal}(\text{location} = 20, \text{scale} = 85)$ -> The model is using 20 as the prior intercept value. 50 would have been more appropriate (see explanation under Section "Fit the model")

Coefficients Specified prior: $\sim \text{normal}(\text{location} = [0,0,0,...], \text{scale} = [2.5,2.5,2.5,...])$ Adjusted prior: $\sim \text{normal}(\text{location} = [0,0,0,...], \text{scale} = [178.99,178.99,135.15,...])$ -> [178.99,178.99,135.15,...] are the scales R has given the priors. It essentially tells us that the priors are diff. This will slow the model down. It would be faster if these brackets had a single value. Given that they are similar, it looks ok.

Auxiliary (sigma) Specified prior: $\sim \text{exponential}(\text{rate} = 1)$ Adjusted prior: $\sim \text{exponential}(\text{rate} = 0.03)$

Covariance

`decov(reg. = 1, conc. = 1, shape = 1, scale = 1)` -> Don't mess with this one. Keep these default values (!)

Prior predictions

What is the range of values that we could predict in the absence of our actual data?

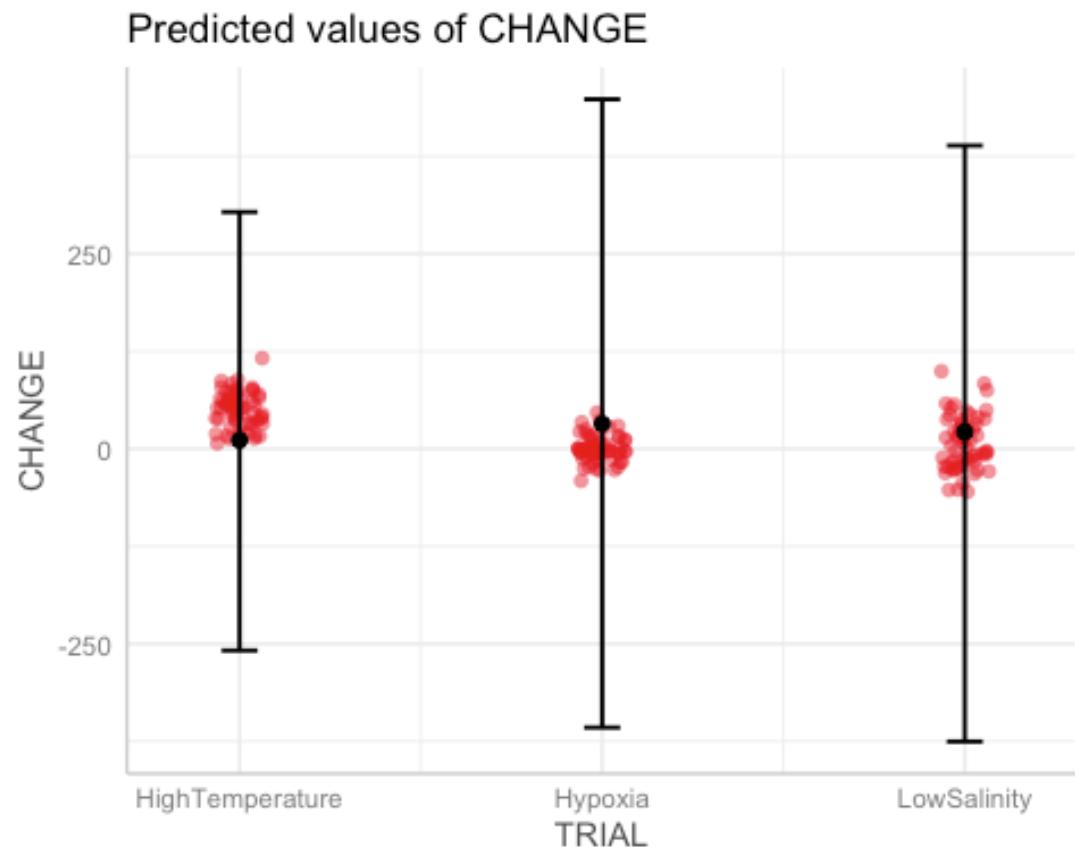
```

ggpredict(norin.rstanarm1) %>%
  plot(add.data=TRUE)

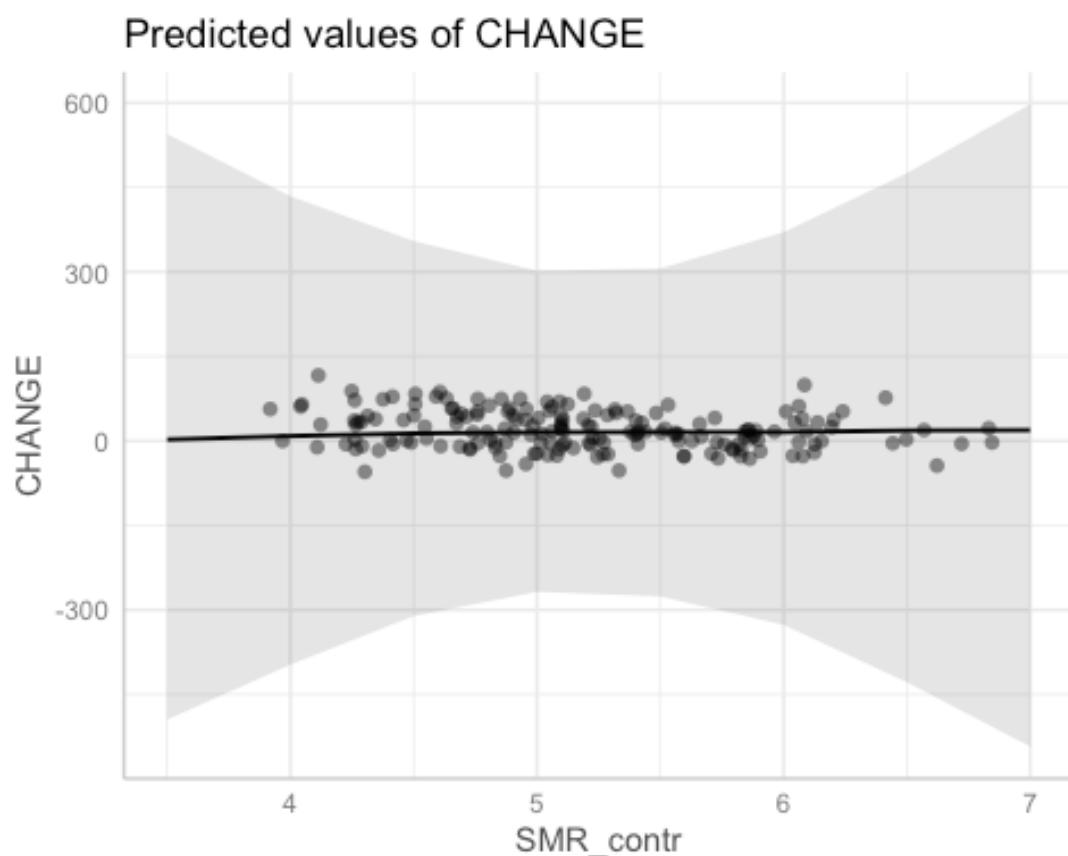
## Note: uncertainty of error terms are not taken into account. You may want
## to use `rstantools::posterior_predict()` .
## Note: uncertainty of error terms are not taken into account. You may want
## to use `rstantools::posterior_predict()` .
## Note: uncertainty of error terms are not taken into account. You may want
## to use `rstantools::posterior_predict()` .

## $TRIAL

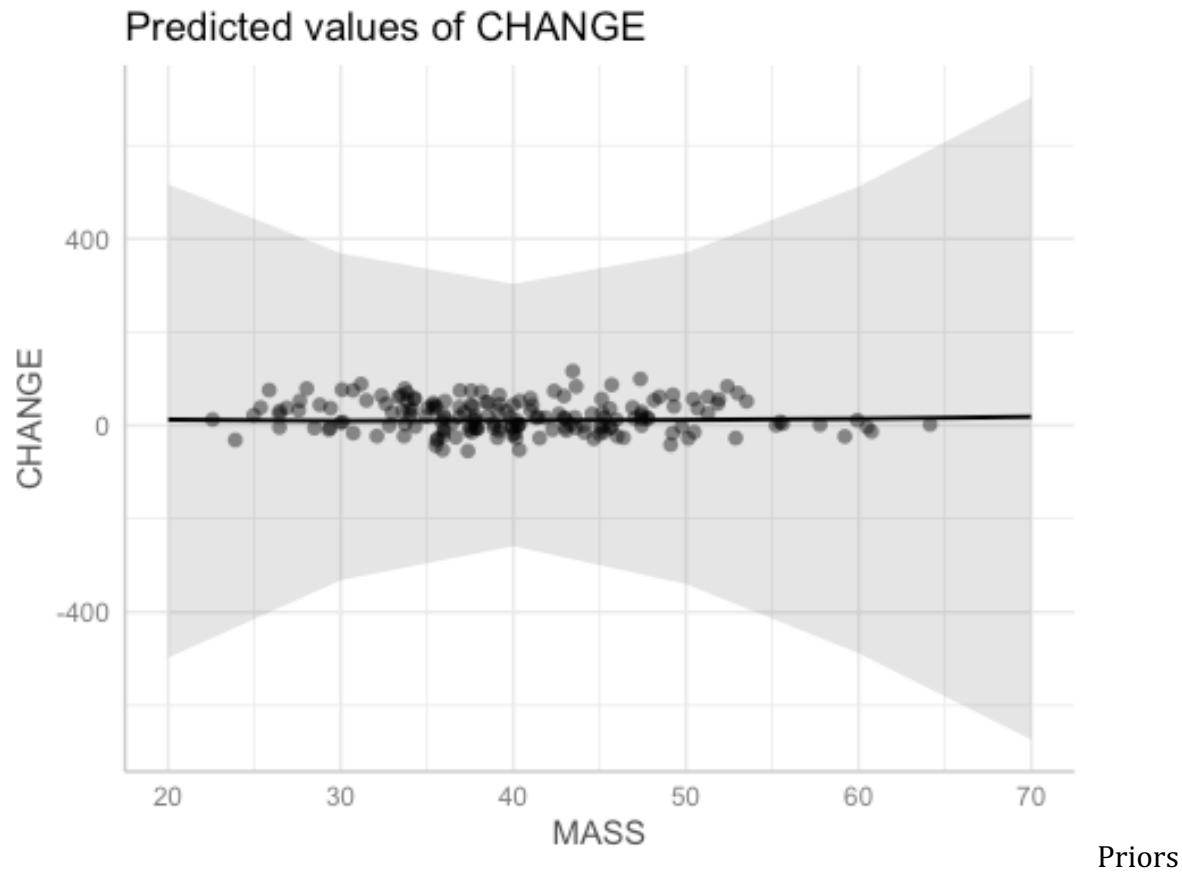
```



```
##  
## $SMR_contr
```



```
##  
## MASS
```



look ok and fairly wide, let's accept them

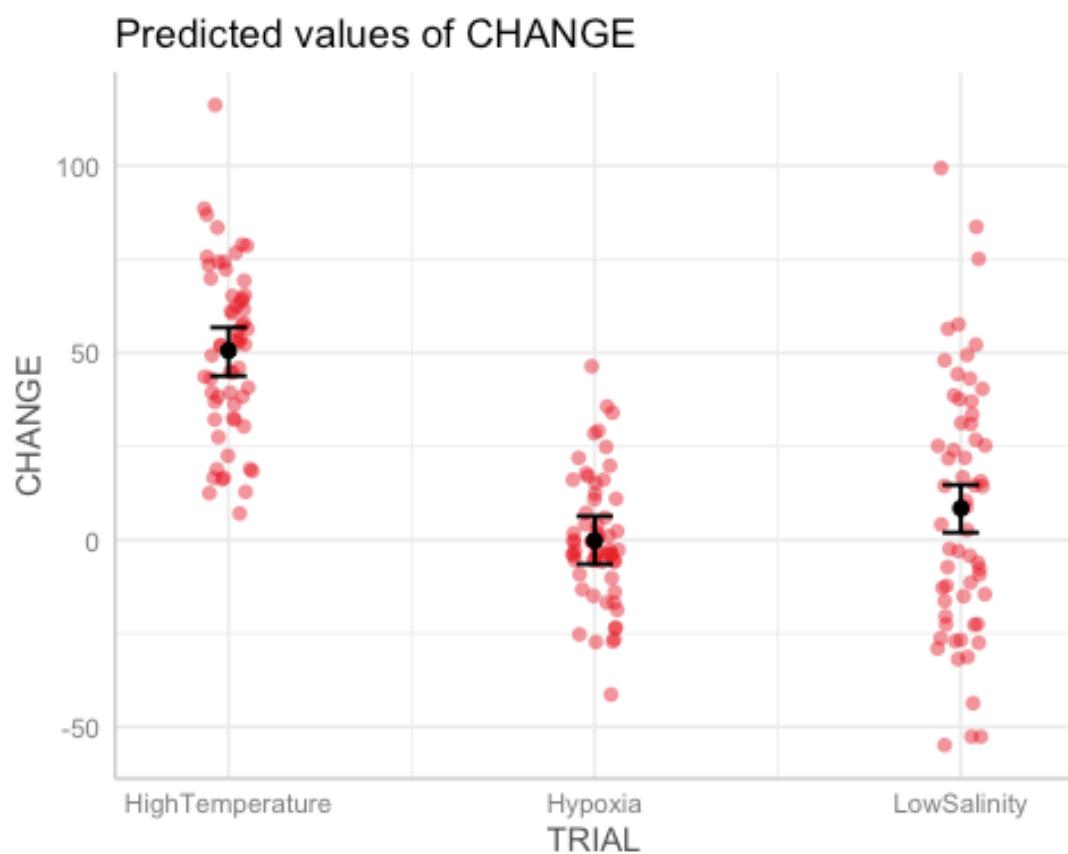
Model for posteriors (let's do it on model 1, the more complex one)

```
norin.rstanarm1<-update(norin.rstanarm1,prior_PD=FALSE)
```

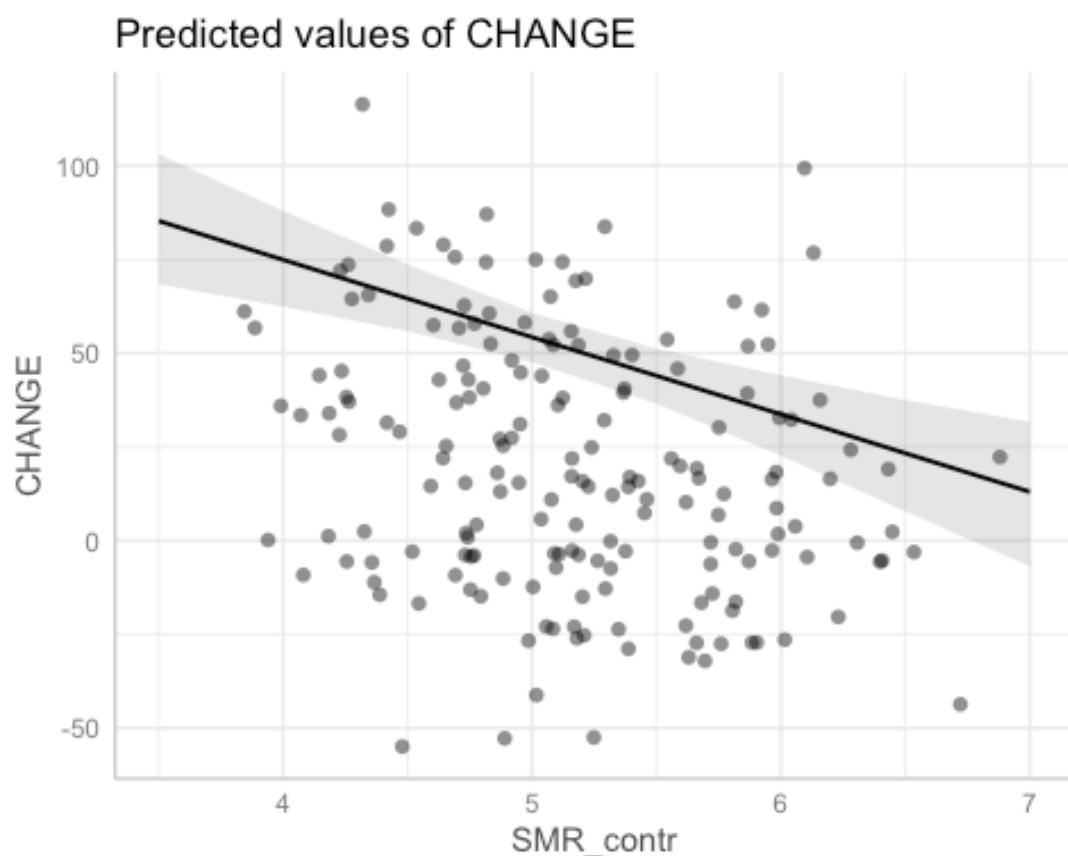
Prediction for posteriors

```
ggpredict(norin.rstanarm1) %>% plot(add.data=TRUE)
```

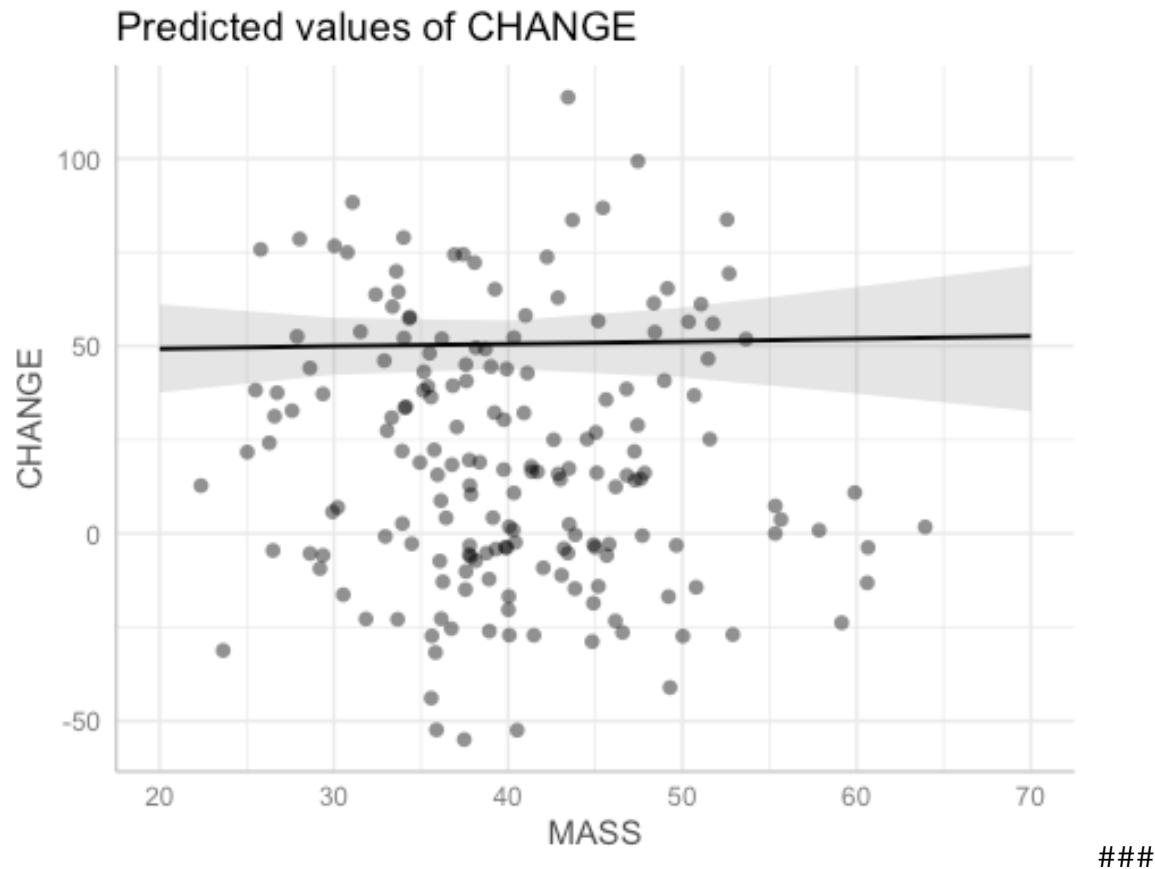
```
## Note: uncertainty of error terms are not taken into account. You may want
to use `rstantools::posterior_predict()` .
## Note: uncertainty of error terms are not taken into account. You may want
to use `rstantools::posterior_predict()` .
## Note: uncertainty of error terms are not taken into account. You may want
to use `rstantools::posterior_predict()` .
## $TRIAL
```



```
##  
## $SMR_contr
```



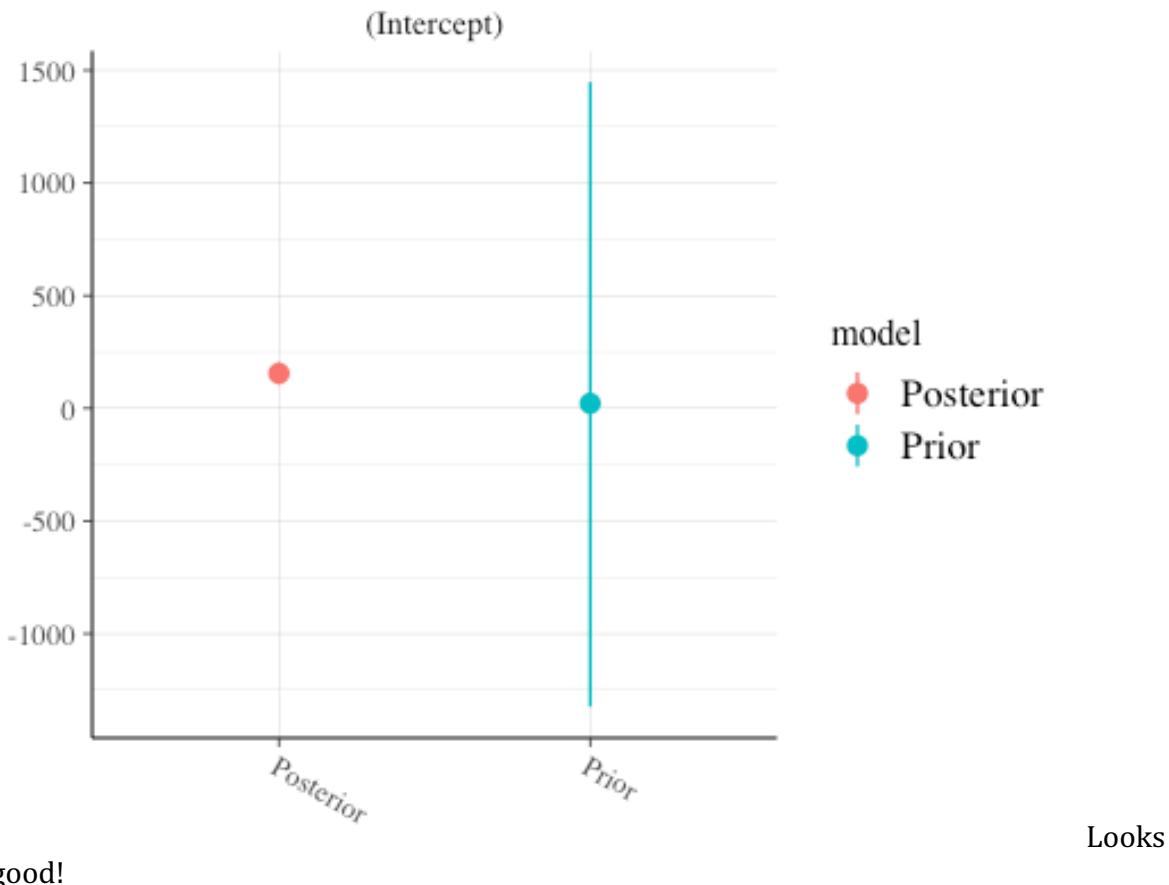
```
##  
## MASS
```



Prior vs posterior

```
posterior_vs_prior(norin.rstanarm1,color_by="vs",
                    group_by=TRUE,
                    facet_args=list(scales="free_y"),
                    pars=c('(Intercept)')) # shows only the variable we are
interested in

##
## Drawing from prior...
```



3- Model 3 - FAR more complicated random intercept random slope model - Just for fun

Let's fit a FAR more complicated random intercept random slope model to see what would happen

```
norin.rstanarm2=rstanarm::stan_glmer(CHANGE~(TRIAL*SMR_contr|FISHID)+TRIAL*SMR_contr+MASS, # This model Looks at the corr btw slopes. We could force it to have an interaction of zero. In this case, bec we have a random int random slope model, we have to include another parameter that might change the slopes of FishID, such as SMR_contr, which effectively seems to alter the slopes (see the scatter plot we created at the very beginning) (!) Once we include a continuous variable (SMR_contr) in the slope, the model can work easier than if it was a categorical variable.
```

```
data=norin,
prior_PD=TRUE, # we first only tell the model to be defined by the priors
family=gaussian, # we define we got a Gaussian family
iter=5000,
```

```

    warmup=2000, # for Gaussian models
    chains=3, # chains=3 tells R to run one chain, then
finish, then run another
    thin=5,
    refresh=0,
    cores=3) # cores=3 tells R to run all chains
together (using all three cores of the computer), instead of doing the
default chains=3 function

```

Prior summary

```

prior_summary(norin.rstanarm2)

## Priors for model 'norin.rstanarm2'
## -----
## Intercept (after predictors centered)
##   Specified prior:
##     ~ normal(location = 20, scale = 2.5)
##   Adjusted prior:
##     ~ normal(location = 20, scale = 85)
##
## Coefficients
##   Specified prior:
##     ~ normal(location = [0,0,0,...], scale = [2.5,2.5,2.5,...])
##   Adjusted prior:
##     ~ normal(location = [0,0,0,...], scale = [178.99,178.99,135.15,...])
##
## Auxiliary (sigma)
##   Specified prior:
##     ~ exponential(rate = 1)
##   Adjusted prior:
##     ~ exponential(rate = 0.03)
##
## Covariance
##   ~ decov(reg. = 1, conc. = 1, shape = 1, scale = 1)
## -----
## See help('prior_summary.stanreg') for more details

```

Let's examine the priors for the model 'norin.rstanarm2' => In this model, the SAME priors as with the previous model have been selected.

Intercept (after predictors centered) Specified prior: $\sim \text{normal}(\text{location} = 20, \text{scale} = 2.5)$
 Adjusted prior: $\sim \text{normal}(\text{location} = 20, \text{scale} = 85)$ -> The model is using 20 as the prior intercept value. 50 would have been more appropriate (see explanation under Section "Fit the model")

Coefficients Specified prior: $\sim \text{normal}(\text{location} = [0,0,0,...], \text{scale} = [2.5,2.5,2.5,...])$ Adjusted prior: $\sim \text{normal}(\text{location} = [0,0,0,...], \text{scale} = [178.99,178.99,135.15,...])$ -> $[178.99,178.99,135.15,...]$ are the scales R has given the priors. It essentially tells us that the priors are diff. This will slow the model down. It would be faster if these brackets had a single value. Given that they are similar, it looks ok.

Auxiliary (σ) Specified prior: $\sim \text{exponential}(\text{rate} = 1)$ Adjusted prior: $\sim \text{exponential}(\text{rate} = 0.03)$

Covariance

`decov(reg. = 1, conc. = 1, shape = 1, scale = 1)` \rightarrow Don't mess with this one. Keep these default values (!)

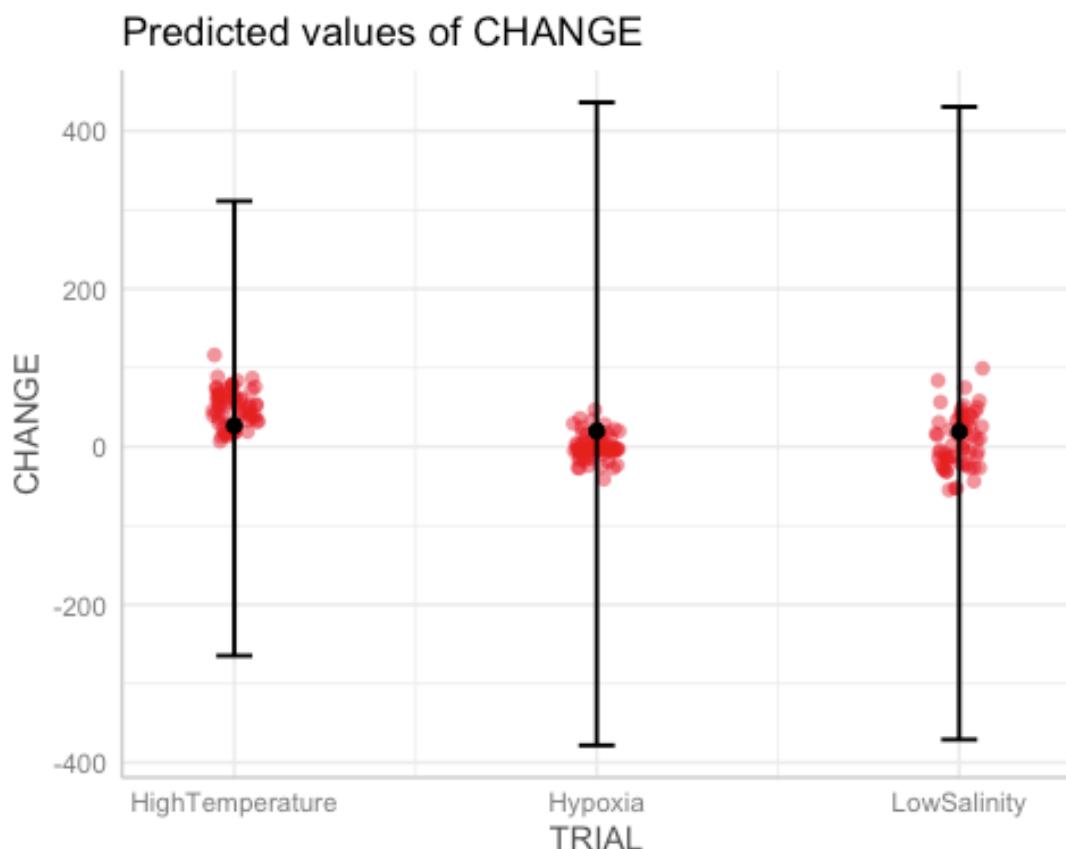
Prior predictions

What is the range of values that we could predict in the absence of our actual data?

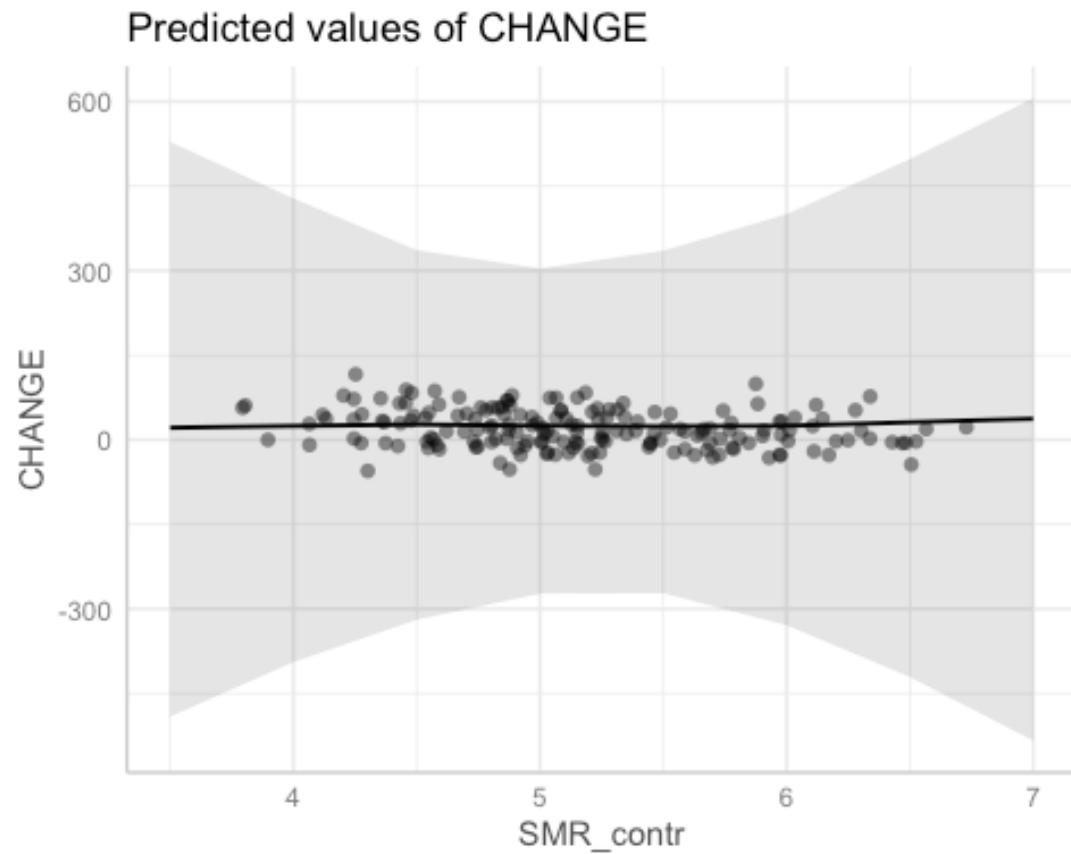
```
ggpredict(norin.rstanarm2) %>%
  plot(add.data=TRUE)

## Note: uncertainty of error terms are not taken into account. You may want
## to use `rstantools::posterior_predict()` .
## Note: uncertainty of error terms are not taken into account. You may want
## to use `rstantools::posterior_predict()` .
## Note: uncertainty of error terms are not taken into account. You may want
## to use `rstantools::posterior_predict()` .

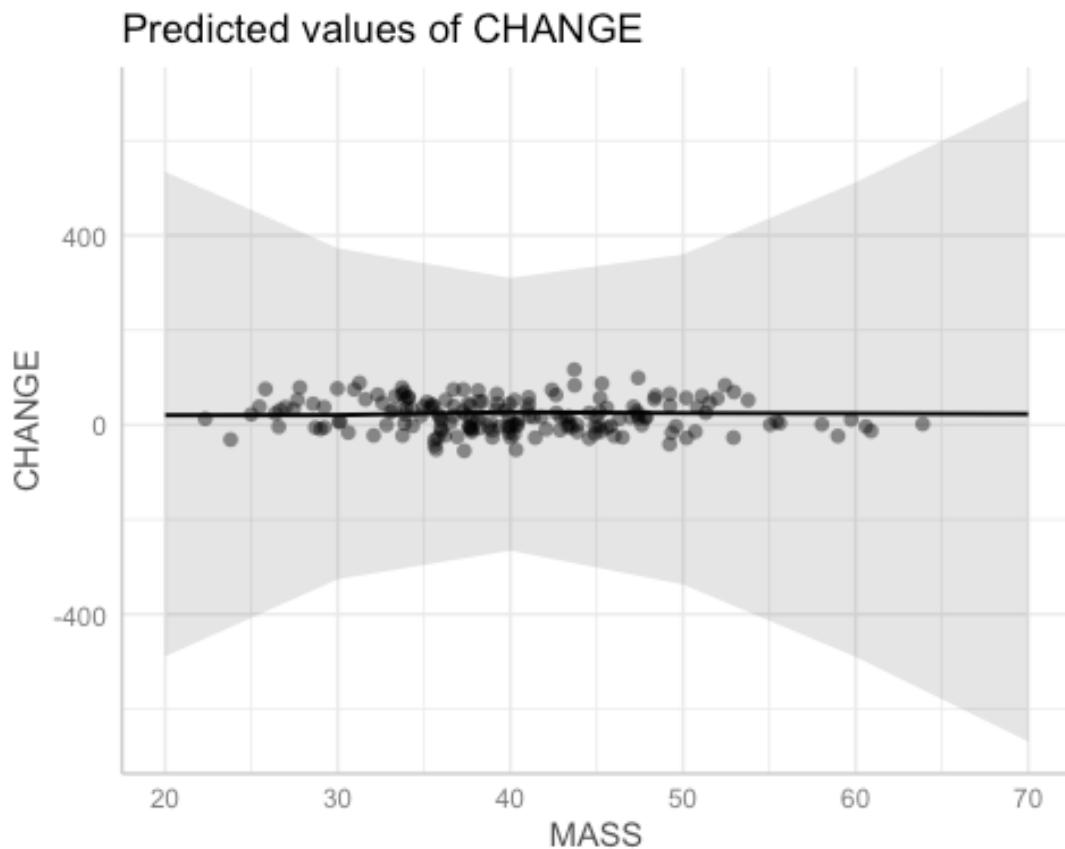
## $TRIAL
```



```
##  
## $SMR_contr
```



```
##  
## $MASS
```



Priors look ok and fairly wide, let's accept them

Model for posteriors (let's do it on model quinn.rstanarm1, the more complex one)

```
norin.rstanarm2<-update(norin.rstanarm2,prior_PD=FALSE)

## Warning: There were 2 divergent transitions after warmup. See
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: There were 2 chains where the estimated Bayesian Fraction of
Missing Information was low. See
## http://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating
posterior means and medians may be unreliable.
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess

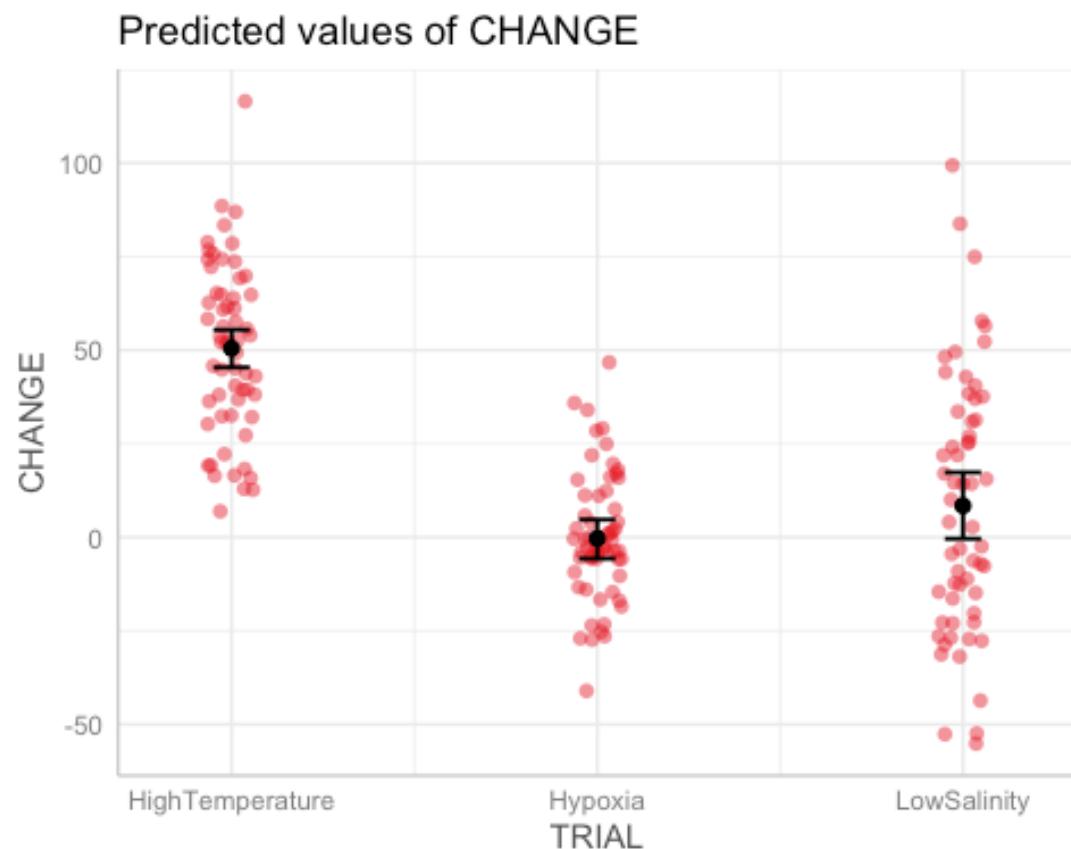
## Warning: Tail Effective Samples Size (ESS) is too low, indicating
posterior variances and tail quantiles may be unreliable.
```

```
## Running the chains for more iterations may help. See  
## http://mc-stan.org/misc/warnings.html#tail-ess
```

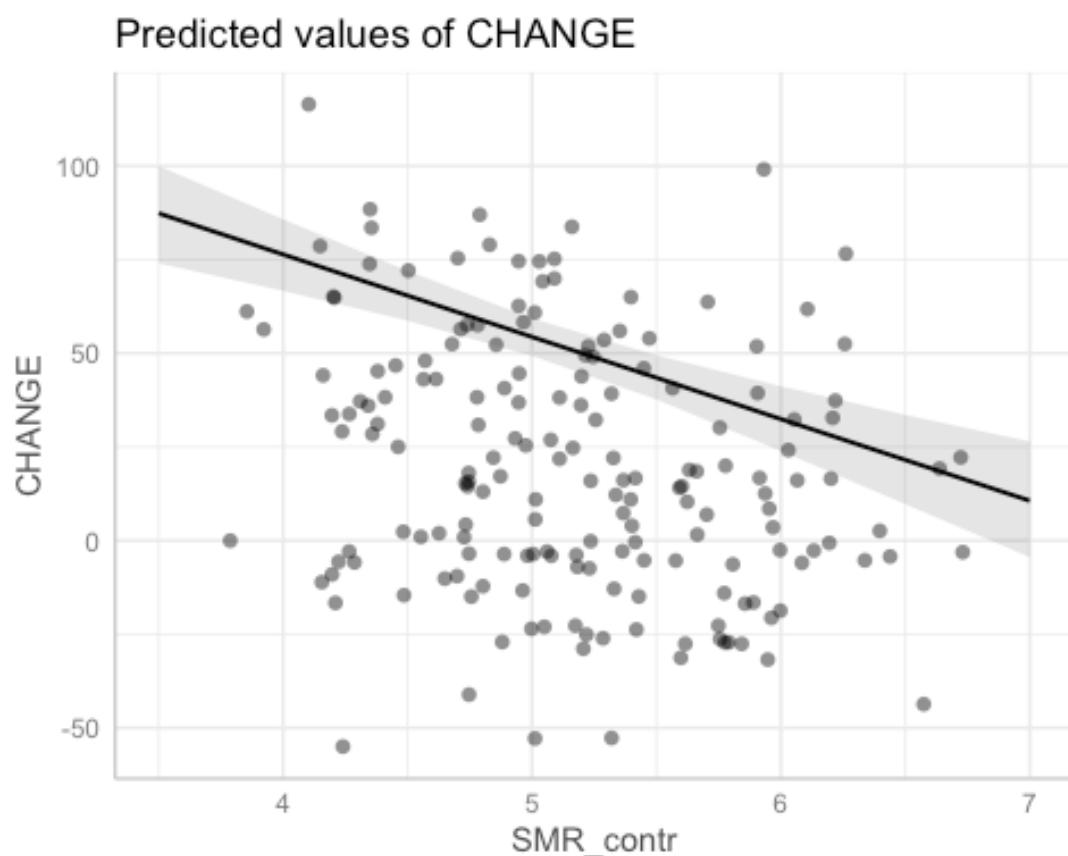
THIS MODEL TAKES AGES TO RUN BECAUSE IT IS SO COMPLEX. MURRAY WOULD RATHER GO FOR model quinn.rstanarm1.

Prediction for posteriors

```
ggpredict(norin.rstanarm2) %>% plot(add.data=TRUE)  
  
## Note: uncertainty of error terms are not taken into account. You may want  
to use `rstantools::posterior_predict()`.  
## Note: uncertainty of error terms are not taken into account. You may want  
to use `rstantools::posterior_predict()`.  
## Note: uncertainty of error terms are not taken into account. You may want  
to use `rstantools::posterior_predict()`.  
  
## $TRIAL
```



```
##  
## $SMR_contr
```



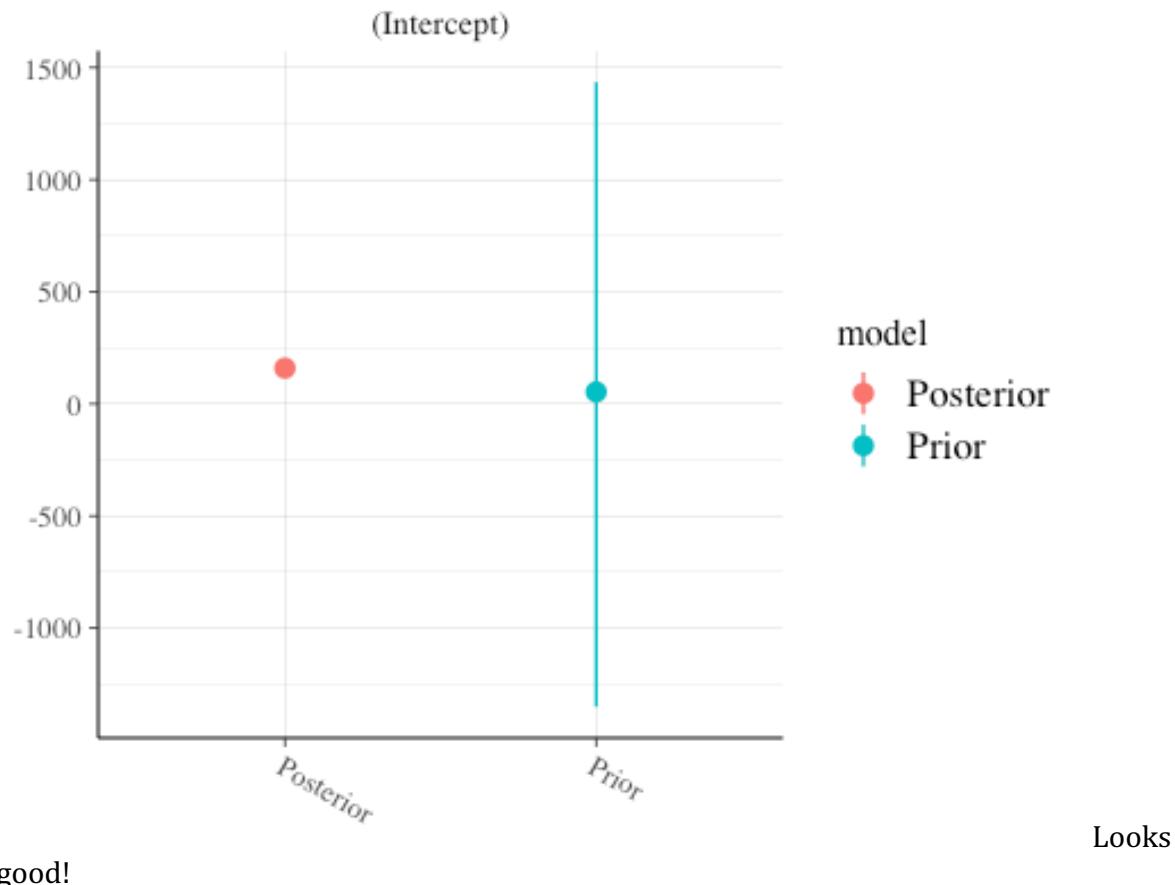
```
##  
## MASS
```



Prior vs posterior

```
posterior_vs_prior(norin.rstanarm2,color_by="vs",
                    group_by=TRUE,
                    facet_args=list(scales="free_y"),
                    pars=c('(Intercept)')) # shows only the variable we are
interested in

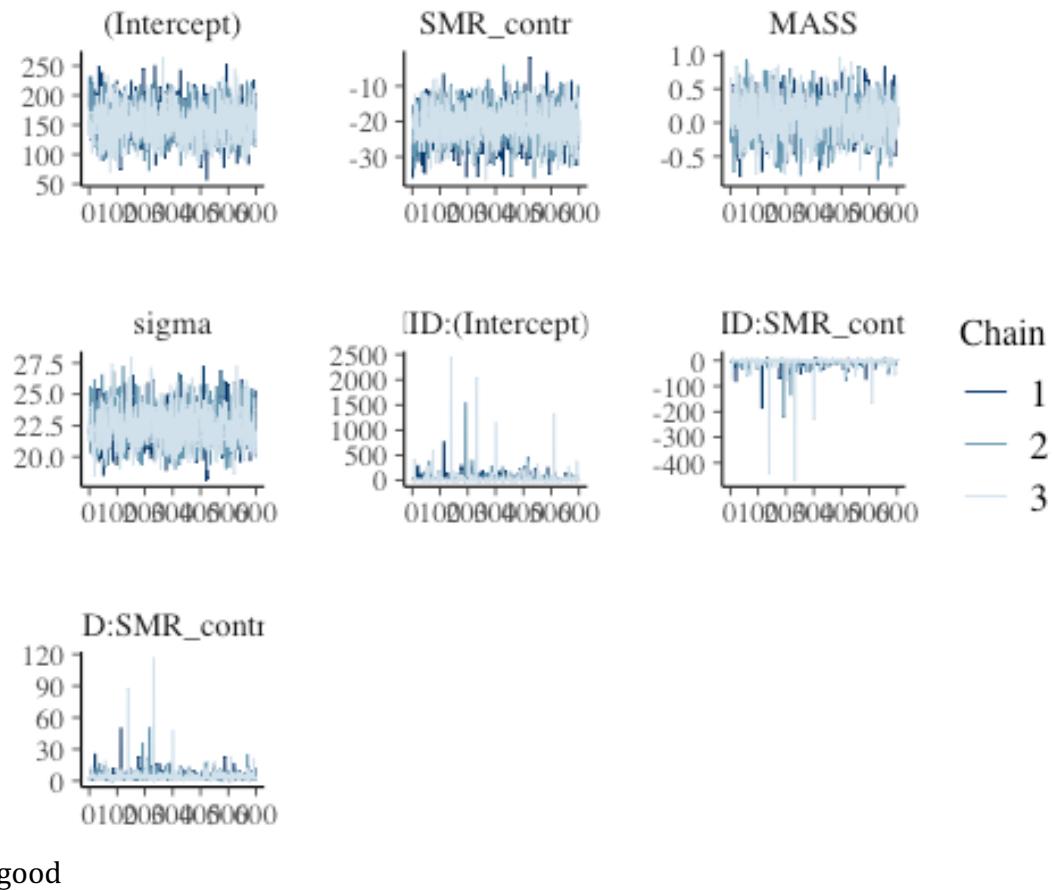
##
## Drawing from prior...
```



Model validation with model2 (norin.rstanarm1)

Trace plots

```
plot(norin.rstanarm1, "mcmc_trace", regex_pars='^.Intercept|^SMR|MASS|[sS]igma')
```

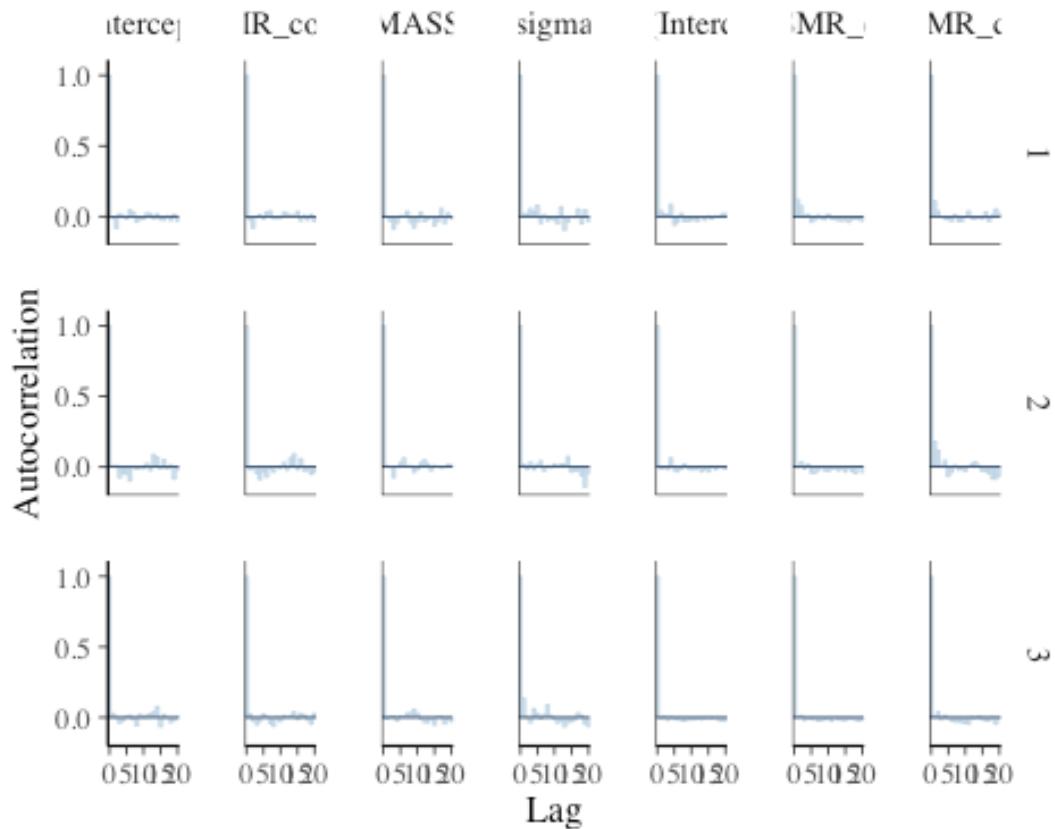


REGULAR EXPRESSIONS - EXPLANATION ABOUT THIS CODE:

Regular expressions help you to target diff patterns of text. . = any character ^ = must start with | = or (in this case, has to have the word TRIAL in it, or has to start with S, M or R, or had to contain MASS, or it had either have a lower case s or an uppercase S and igma (so it will get both sigma and Sigma))

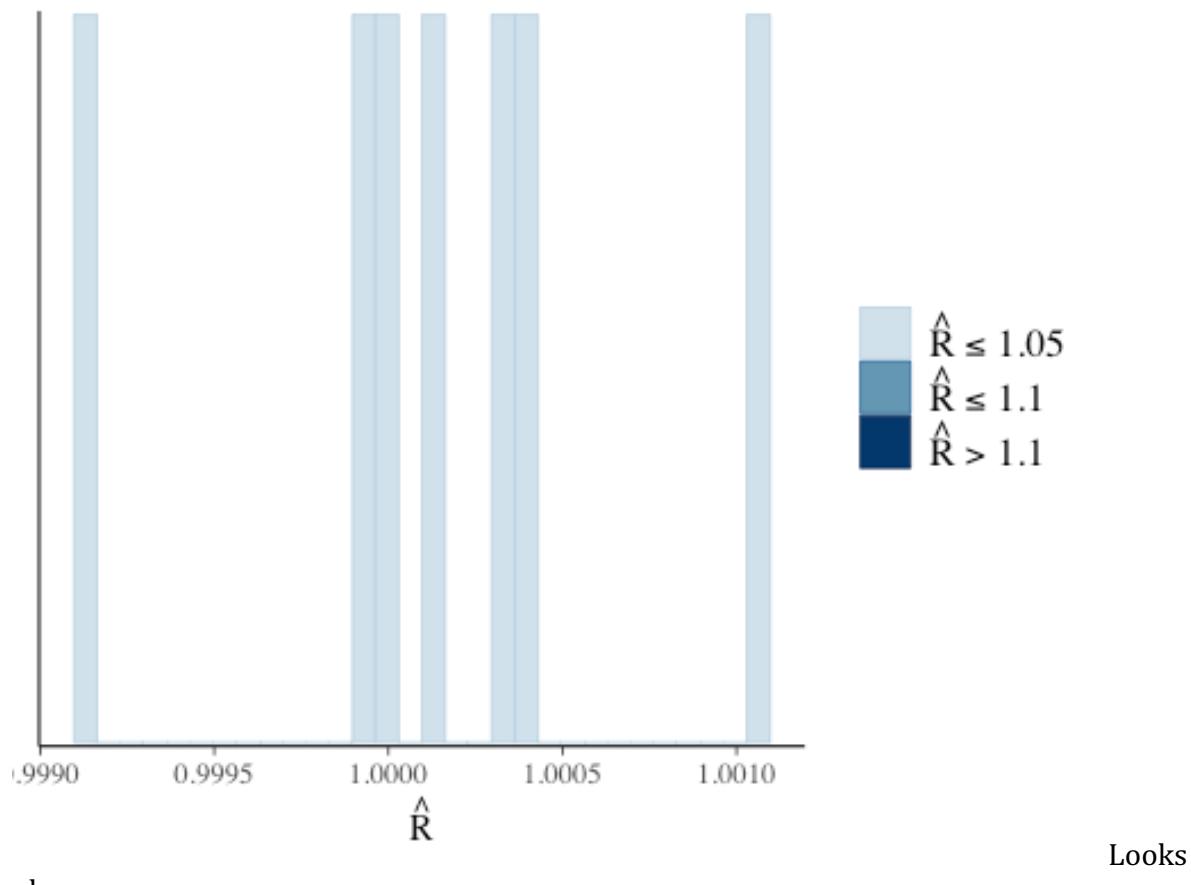
Autocorrelation

```
plot(norin.rstanarm1, "mcmc_acf_bar", regex_pars='^.Intercept|^SMR|MASS|[sS]igma')
```



Measure of chain convergence - R hat

```
plot(norin.rstanarm1, "rhat_hist", regex_pars='^.Intercept|^SMR|MASS|[sS]igma')
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

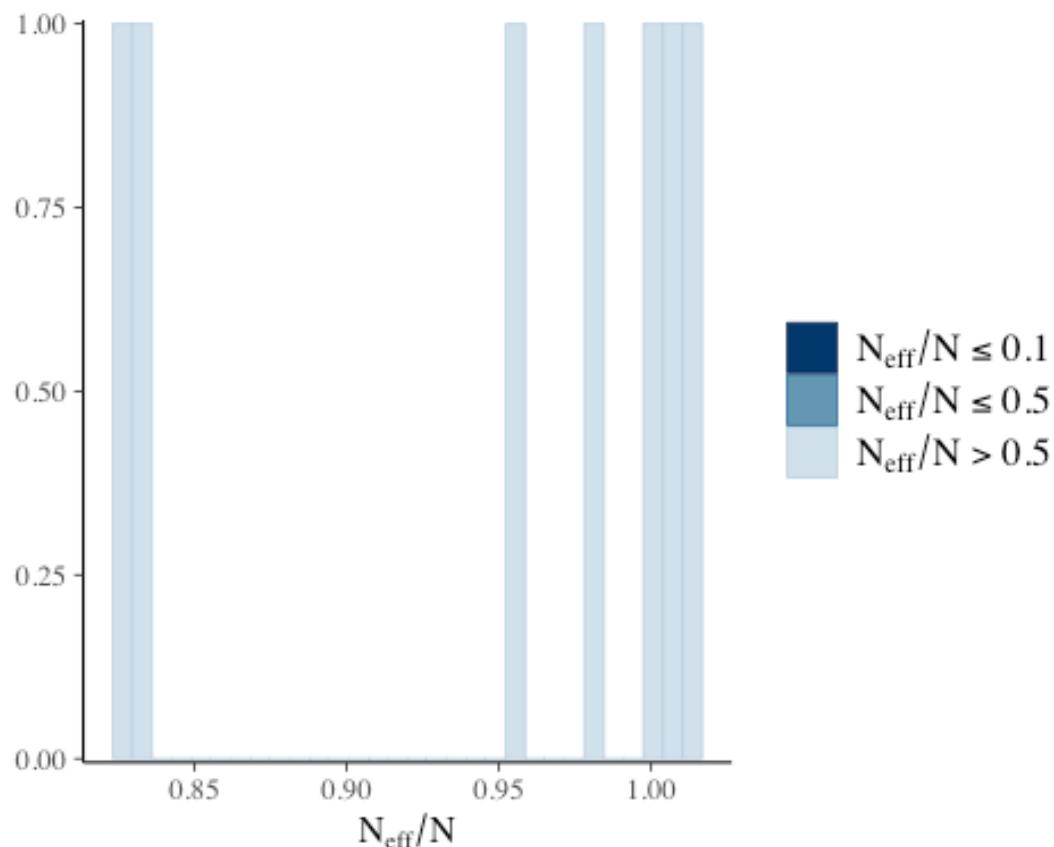


ok

Looks

Number of effective samples

```
plot(norin.rstanarm1, "neff_hist", regex_pars='^.Intercept|^SMR|MASS|[sS]igma')  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



DHARMA residuals (most important part of the model validation)

```

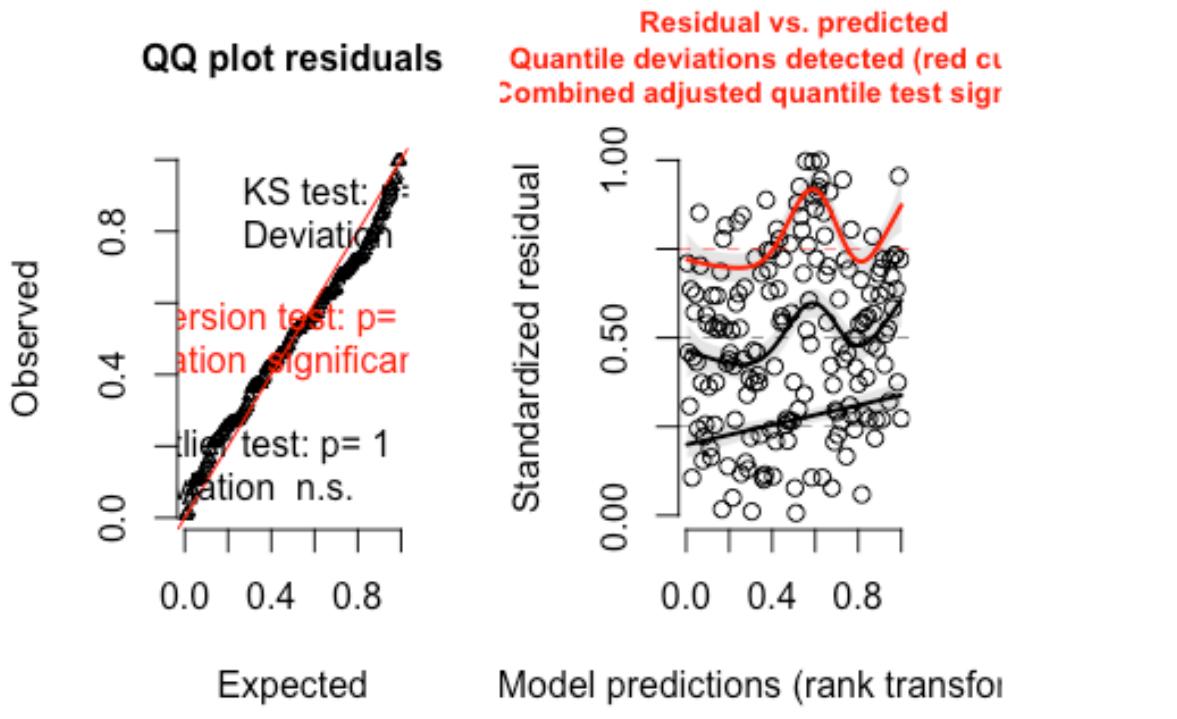
preds<-posterior_predict(norin.rstanarm1, nsamples=250, summary=FALSE)

norin.resids<-createDHARMA(simulatedResponse = t(preds),
                             observedResponse = norin$CHANGE,
                             fittedPredictedResponse = apply(preds, 2, median),
                             # 2 = we tell R to take preds and apply it to
                             the rows (1) or to the columns (2)
                             integerResponse = FALSE)

plot(norin.resids)

```

DHARMA residual diagnostics



Does not look too great. But I kind of know Murray would say it's still good enough to continue

Picking out the parameters that we want to run predictions on

The problem with our model is that we got HEAPS of variables (as identified in the model validation, where heaps of parameters got plotted and we had to define the important ones). Therefore, we do the same, but first checking:

WHICH parameters are actually important:

```
nms<-norin.rstanarm1 %>% get_variables()

head(nms)

## [1] "(Intercept)"                 "TRIALHypoxia"                "TRIALLowSalinity"
## [4] "SMR_contr"                  "MASS"                       "TRIALHypoxia:SMR_contr"
```

Alternative: We tell R to show us the rows of which parameters match the pattern that we established earlier.

```
wch<-grep('^.Intercept|TRIAL|^SMR|[sS]igma',nms)
head(wch)
## [1] 1 2 3 4 6 7
```

Predictions

```
broom.mixed:::tidyMCMC(norin.rstanarm1$stanfit,conf.int=TRUE,conf.method="HPDInterval",
                         rhat=TRUE,ess=TRUE,pars=nms[wch],estimate.method="median")

## # A tibble: 10 x 7
##   term                  estimate std.error conf.low conf.high rhat
##   <chr>                 <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept)           155.      29.8     9.49e+1   209.     1.00
## 2 TRIALHypoxia         -112.     32.6    -1.81e+2   -52.5     1.00
## 3 TRIALLowSalinity     -87.3     32.6    -1.50e+2   -24.4     1.00
## 4 SMR_contr            -20.6     5.10    -3.06e+1   -11.2     1.00
## 5 TRIALHypoxia:SMR_contr 12.0     6.31    3.39e-2    25.0     1.00
## 6 TRIALLowSalinity:SMR_contr 8.79     6.26    -3.13e+0    21.1     1.00
## 7 sigma                 22.2     1.48    1.94e+1    25.3     1.00
## 8 Sigma[FISHID:(Intercept),... 5.46     105.     1.15e-3   114.     1.00
## 9 Sigma[FISHID:SMR_contr,(I... -0.108    20.2    -1.66e+1    7.06     1.00
## 10 Sigma[FISHID:SMR_contr,SM...  3.30     4.99    4.18e-4    9.60     1.00
```

WE'D NOW CONTINUE AS WE DID WITH PREVIOUS EXAMPLES, PLOTTING THE PREDICTION VALUES

IGNORE BELOW

Partial plots model

Model investigation / hypothesis testing

Predictions / further analyses

Summary figures

References

lme (nlme)

fixed structures

```
##Compare models that estimate partial slope for MASS vs an offset for MASS
##must use ML to compare models that vary in fixed effects
norin.lme1 <- lme(CHANGE ~ TRIAL*scale(SMR_contr, scale=FALSE) +
offset(MASS), random=~1|FISHID, data=norin, method='ML')
## Unfortunately, update() does not remove offset().
## We will just have to write the other model out in full as well.
norin.lme2 <- lme(CHANGE ~ TRIAL*scale(SMR_contr, scale=FALSE) + MASS,
random=~1|FISHID, data=norin, method='ML')
## Now without MASS altogether
norin.lme3 <- lme(CHANGE ~ TRIAL*scale(SMR_contr, scale=FALSE),
random=~1|FISHID, data=norin, method='ML')

## Compare these models via AICc
AICc(norin.lme1,norin.lme2, norin.lme3)

##          df      AICc
## norin.lme1 8 1664.473
## norin.lme2  9 1666.596
## norin.lme3  8 1664.473

## Alternatively, we can use sequential Likelihood Ratio Tests (LRT)
anova(norin.lme1, norin.lme2, norin.lme3)
```

```
##          Model df     AIC     BIC   logLik   Test    L.Ratio p-value
## norin.lme1     1  8 1663.631 1689.175 -823.8155
## norin.lme2     2  9 1665.537 1694.274 -823.7688 1 vs 2 0.09355013  0.7597
## norin.lme3     3  8 1663.631 1689.175 -823.8155 2 vs 3 0.09355013  0.7597
```

Conclusions:

- on the basis of AICc, the model without MASS is considered the most parsimonious (lowest AICc)
- we will proceed with the fixed structure of model 3

random structures

```
norin.lme3a = update(norin.lme3, method='REML')
norin.lme3b = update(norin.lme3a, random=~TRIAL|FISHID)
AICc(norin.lme3a, norin.lme3b)

##          df      AICc
## norin.lme3a 8 1637.191
## norin.lme3b 13 1596.153

anova(norin.lme3a, norin.lme3b)

##          Model df     AIC     BIC   logLik   Test    L.Ratio p-value
## norin.lme3a     1  8 1636.349 1661.621 -810.1744
## norin.lme3b     2 13 1593.961 1635.028 -783.9803 1 vs 2 52.38814 <.0001
```

Conclusions:

- the AICc of the random intercept/slope model is lowest
- therefore we will proceed with a random intercept/slope model

lmer (lme4)

fixed structure

```
##Compare models that estimate partial slope for MASS vs an offset for MASS
##must use ML to compare models that vary in fixed effects
norin.lmer1 <- lmer(CHANGE ~ TRIAL*scale(SMR_contr, scale=FALSE) +
offset(MASS) + (1|FISHID), data=norin, REML=FALSE)
## Unfortunately, update() does not remove offset().
## We will just have to write the other model out in full as well.
norin.lmer2 <- lmer(CHANGE ~ TRIAL*scale(SMR_contr, scale=FALSE) + MASS +
(1|FISHID), data=norin, REML=FALSE)
## Now without MASS altogether
norin.lmer3 <- lmer(CHANGE ~ TRIAL*scale(SMR_contr, scale=FALSE)+ (1|FISHID),
data=norin, REML=FALSE)

## Compare these models via AICc
AICc(norin.lmer1,norin.lmer2, norin.lmer3)
```

```

##          df     AICc
## norin.lmer1 8 1673.943
## norin.lmer2 9 1666.596
## norin.lmer3 8 1664.473

## Alternatively, we can use sequential Likelihood Ratio Tests (LRT)
anova(norin.lmer1, norin.lmer2, norin.lmer3)

## Data: norin
## Models:
## norin.lmer1: CHANGE ~ TRIAL * scale(SMR_contr, scale = FALSE) +
## offset(MASS) +
## norin.lmer1: (1 | FISHID)
## norin.lmer3: CHANGE ~ TRIAL * scale(SMR_contr, scale = FALSE) + (1 |
## FISHID)
## norin.lmer2: CHANGE ~ TRIAL * scale(SMR_contr, scale = FALSE) + MASS + (1 |
## norin.lmer2: FISHID)
##           npar     AIC     BIC logLik deviance Chisq Df Pr(>Chisq)
## norin.lmer1     8 1673.1 1698.6 -828.55    1657.1
## norin.lmer3     8 1663.6 1689.2 -823.82    1647.6 9.4701   0
## norin.lmer2     9 1665.5 1694.3 -823.77    1647.5 0.0936   1     0.7597

```

Conclusions:

- on the basis of AICc, the model without MASS is considered the most parsimonious (lowest AICc)
- we will proceed with the fixed structure of model 3

random structure

```

norin.lmer3a = update(norin.lmer3, REML=TRUE)
## unfortunately, the following random intercept/slope model cannot be fit
## norin.lmer3b = update(norin.lmer3a, ~ . -(1/FISHID) +(TRIAL|FISHID))
## AICc(norin.lmer3a, norin.lmer3b)
## anova(norin.lmer3a, norin.lmer3b)

```

Conclusions:

- unfortunately, we are unable to fit a model with random intercept/slope
- therefore we will proceed with a random intercept model

glmmTMB

fixed structure

```

## Compare models that estimate partial slope for MASS vs an offset for MASS
norin.glmmTMB1 = glmmTMB(CHANGE ~ TRIAL*scale(SMR_contr, scale=FALSE) +
offset(MASS) + (1|FISHID), data=norin, REML=FALSE)
## Unfortunately, update() does not remove offset().
## We will just have to write the other model out in full as well.
norin.glmmTMB2 <- lmer(CHANGE ~ TRIAL*scale(SMR_contr, scale=FALSE) + MASS +

```

```
(1|FISHID), data=norin, REML=FALSE)
## Now without MASS altogether
norin.glmmTMB3 <- lmer(CHANGE ~ TRIAL*scale(SMR_contr, scale=FALSE) +
(1|FISHID), data=norin, REML=FALSE)

## Compare these models via AICc
AICc(norin.glmmTMB1, norin.glmmTMB2, norin.glmmTMB3)

##           df      AICc
## norin.glmmTMB1  8 1673.943
## norin.glmmTMB2  9 1666.596
## norin.glmmTMB3  8 1664.473
```

Conclusions:

- on the basis of AICc, the model without MASS is considered the most parsimonious (lowest AICc)
- we will proceed with the fixed structure of model 3

random structure

```
norin.glmmTMB3a <- update(norin.glmmTMB3, REML=TRUE)
## unfortunately, the following random intercept/slope model cannot be fit
## norin.glmmTMB3b <- update(norin.glmmTMB3a, ~ . - (1/FISHID) +
## (TRIAL/FISHID))
##AICc(norin.glmmTMB1a, norin.glmmTMB1b)
```

Conclusions:

- unfortunately, we are unable to fit a model with random intercept/slope
- therefore we will proceed with a random intercept model

21_RegressionTrees_1

Sara Kophamel

17/12/2020

MACHINE LEARNING - REGRESSION TREES

THEORY

Open: file:///Users/sara/Documents/PhD/HELP/STATISTICS%20and%20GRAPHS/-%20R%20and%20STATS%20COURSES/2020%20Murray%20Logan%20R%20Course/Logan_Course/docs/pres9.html#1

Not good at handling good predictors, but they are REALLY good for handling multidimensionality in the PREDICTIONS.

They are also very good at identifying the drivers that are more likely driving the response of the data.

- Frequentist, normal GLM are ok in handling 7-8 additive effects, but >10 our model would crash. Also, adding more than 3 interactions (*) is a bit crazy.

Similar to GAMs, we can overfit a Regression Tree (slide 21). The point is that Regression Trees LEARN PATTERNS. By learning every pattern in our dataset, they easily overfit (as they learn our data by heart).

How do Regression Trees work?

Regression trees work by...

1- Splitting (partitioning) data into major chunks.

Our regression tree is like a decision tree (imagine a taxonomic tree). It is making a decision (slide 11):

Example $y \wedge \dots | \dots | \dots | \dots | _1_2_3_4 \rightarrow x$

After identifying a pattern, it splits this distribution into 2

V

$y \wedge \dots | \dots | \dots | \dots | \dots | \dots | _1_2_3_4 \rightarrow x$

If we allowed it to continue splitting, it might split this distribution again into 2 on the y axis

V

y ^ ..| ...| ..| ..| _____ |. ...| .| ...| ..| |||_1_2_3_4_>x

The model could keep going, until we had isolated all data points. This would mean that the data had been overfitted

So how can we improve this splitting? By boosting (slide 20)

2- Boosting

We take what one model does not know, fit another tree, take what that tree does not know, and take another tree, etc. But you force it to only learn a small amount at a time, fitting thousands of trees. The model will start off by learning the most basic features of the data. As you get along, it will start learning individual observations, which is the point when we want to cut it from learning ("it learnt enough, otherwise it would overfit the model!"). The key part is to learn slowly, and to learn lots, then create the perfect fit in data it had not been learning on.

By doing this, the tree produces a perfect fit.

How do we decide it has learnt enough? It's the same as GAMS = finding the perfect balance btw fitting the model and overfitting. Regression trees can very accurately predict data it has not seen before (this would be great for betting!).

The diff btw a Boosted Tree and a Random Forest is that a boosted tree randomly chooses half the data, then chooses another half of the data. A random forest would fit the model with the data, then leaving out a predictor, then fitting the data, leaving out another predictor, fitting the data again, etc.

3- Minimizing square error loss

This can be done in three ways

i. **Test (validate) the data: 75% train, 25% test**

ii. **out of bag method**

In a boosted regression tree, half the data goes in "a bag" (gets selected) to be trained, half of it is left out for accuracy. Then we move on, choosing another random 50%, and a random 50% for assessing, and so on. If you do millions of trees, it sorts itself out.

iii. cross validation

Multiple regression trees are run btw 3-10x, and each one will leave out multiple observations. Based on that, you can calculate the bias. Cross-validation would only be predicting one observation at a time, but would do it many times (3,5...).

So what option to pick? Murray prefers the following order: Cross validation (best) -> Out of bag (second) -> The test of train (try this as last resource)

4- Variable importance (slide 25)

Regr trees can tell you which variables are more important in your data.

Regr trees count up which time the variables are involved in all of these decision splits. The more involvement, the more importance.

PRACTICE

Open: tree_example1.Rmd

Preparations

Load the necessary libraries

```
library(gbm) #for gradient boosted models. Apparently, xtoobost is better,  
but it is REALLY hard to use (so forget about it). gbm is much easier to use  
and is often use in ecology, so this is fine  
  
## Loaded gbm 2.1.8  
  
library(car)  
  
## Loading required package: carData  
  
library(dismo)  
  
## Loading required package: raster  
  
## Loading required package: sp  
  
library(pdp) # does nice dependency plots for trees  
library(ggfortify)  
  
## Loading required package: ggplot2  
  
library(randomForest) # creates the random forests  
  
## randomForest 4.6-14  
  
## Type rfNews() to see new features/changes/bug fixes.
```

```

## 
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
## 
##     margin

library(tidyverse)

## — Attaching packages


---


———— tidyverse 1.3.0 —————

## ✓ tibble  3.0.3      ✓ dplyr   1.0.2
## ✓ tidyr   1.1.2      ✓ stringr 1.4.0
## ✓ readr   1.3.1      ✓forcats 0.5.0
## ✓ purrr   0.3.4

## — Conflicts


---


———— tidyverse_conflicts() —————

## x dplyr::combine()      masks randomForest::combine()
## x tidyr::extract()      masks raster::extract()
## x dplyr::filter()       masks stats::filter()
## x dplyr::lag()          masks stats::lag()
## x randomForest::margin() masks ggplot2::margin()
## x purrr::partial()       masks pdp::partial()
## x dplyr::recode()        masks car::recode()
## x dplyr::select()        masks raster::select()
## x purrr::some()          masks car::some()

library(gridExtra)

## 
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
## 
##     combine

## The following object is masked from 'package:randomForest':
## 
##     combine

library(patchwork)

## 
## Attaching package: 'patchwork'

## The following object is masked from 'package:raster':
## 
##     area

```

Scenario

Abalone are an important aquaculture shell fish that are farmed for both their meat and their shells. Abalone can live up to 50 years, although their longevity is known to be influenced by a range of environmental factors. Traditionally, abalone are aged by counting their GROWTH RINGS, however, this method is very laborious and expensive. Hence a study was conducted in which abalone growth ring counts were matched up with a range of other more easily measured physical characteristics (such as shell dimensions and weights) in order to see if any of these OTHER PARAMETERS could be used as proxies for the NUMBER of GROWTH RINGS (or age).



Format of abalone.csv data file

Read in the data

```
abalone = read_csv('data/abalone.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   SEX = col_character(),
##   LENGTH = col_double(),
##   DIAMETER = col_double(),
##   HEIGHT = col_double(),
##   WHOLE_WEIGHT = col_double(),
##   MEAT_WEIGHT = col_double(),
##   GUT_WEIGHT = col_double(),
##   SHELL_WEIGHT = col_double(),
##   RINGS = col_double(),
##   AGE = col_double()
## )

glimpse(abalone)
```

```

## Rows: 4,177
## Columns: 10
## $ SEX      <chr> "M", "M", "F", "M", "I", "I", "F", "F", "M", "F",
## "F", ...
## $ LENGTH    <dbl> 0.455, 0.350, 0.530, 0.440, 0.330, 0.425, 0.530,
## 0.545, ...
## $ DIAMETER   <dbl> 0.365, 0.265, 0.420, 0.365, 0.255, 0.300, 0.415,
## 0.425, ...
## $ HEIGHT     <dbl> 0.095, 0.090, 0.135, 0.125, 0.080, 0.095, 0.150,
## 0.125, ...
## $ WHOLE_WEIGHT <dbl> 0.5140, 0.2255, 0.6770, 0.5160, 0.2050, 0.3515,
## 0.7775, ...
## $ MEAT_WEIGHT <dbl> 0.2245, 0.0995, 0.2565, 0.2155, 0.0895, 0.1410,
## 0.2370, ...
## $ GUT_WEIGHT   <dbl> 0.1010, 0.0485, 0.1415, 0.1140, 0.0395, 0.0775,
## 0.1415, ...
## $ SHELL_WEIGHT <dbl> 0.150, 0.070, 0.210, 0.155, 0.055, 0.120, 0.330,
## 0.260, ...
## $ RINGS       <dbl> 15, 7, 9, 10, 7, 8, 20, 16, 9, 19, 14, 10, 11, 10,
## 10, 1...
## $ AGE         <dbl> 16.5, 8.5, 10.5, 11.5, 8.5, 9.5, 21.5, 17.5, 10.5,
## 20.5, ...

```

In summary, we are interested in the number of rings. However, the researchers collected a whole bunch of variables to see which one actually predicts number of rings.

We are NOT gonna use Linear Models, as they are not that great at predicting with many variables. We'll try with Regression Trees!

Define categorical variables as factors

```
abalone = abalone %>% mutate(SEX=factor(SEX))
```

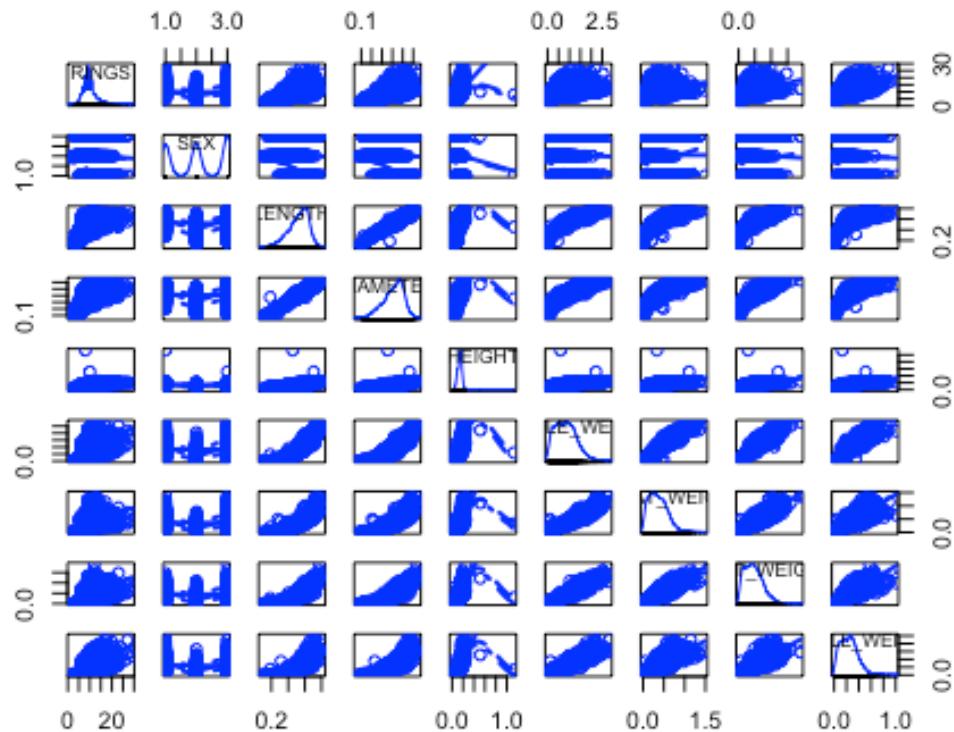
Exploratory data analysis

Scatterplot matrix

We don't have to check any assumptions, as there are NO assumptions. We just wanna get a sense of how the data look like. We wanna check what the patterns are, and which parameters might influence the predictor/response variable (number of rings).

To do a scatterplotMatrix, we define the response variable (RINGS) and list with (+) all variables that could possibly influence it.

```
scatterplotMatrix(~RINGS+SEX+LENGTH+DIAMETER+HEIGHT+WHOLE_WEIGHT+MEAT_WEIGHT+
GUT_WEIGHT+SHELL_WEIGHT,data=abalone)
```



```
# Figure was too Large, possibly run this in the console, or define the
length and width of the plot beforehand
```

SOME parameters seem to be related to the rings: - length - diameter - weight parameters

Fit the model

1- First, Define a random number of samples

Check how many rows we got

```
set.seed(123) # This is for everybody in the class to get the same thing
(easier to compare)
# DON't put a set.seed within your for Loop, or we will get the same results
for each Loop. Write it BEFORE the Loop.
```

```
nrow(abalone)
```

```
## [1] 4177
```

We have 4177 observations.

Tell R what rows we want

```
i=sample(1:nrow(abalone), 100, replace=FALSE) # we tell R we want 100 rows

abalone.train=abalone[-i,]

abalone.test=abalone[i,]
```

2- Fit the model with those 100 random rows

```
abalone.gbm=gbm(RINGS~SEX+LENGTH+DIAMETER+HEIGHT+WHOLE_WEIGHT+MEAT_WEIGHT+GUT
                 _WEIGHT+SHELL_WEIGHT,
                 data=abalone.train,
                 distribution="poisson", # we define the Loss function, the
                 distribution that our data seems to be more likely
                 var.monotone=c(0,1,1,1,1,1,1,1), # defining the polarity of
                 the expected trends. This is defined based on the first row of the
                 scatterplotMatrix that we plotted above. In this case, we got 8 variables
                 that might influence the numbers of rings. For each variable, we define in
                 order any of the following parameters: 0=no interaction, 1=+pos interaction, -
                 1=-neg interaction. In the case of CATEGORICAL variables though, we ALWAYS
                 write a 0
                 n.trees=10000, # we tell R to use 10,000 trees (default)
                 interaction.depth=5, # we define how big each tree needs to
                 be (5 is kind of default, but you'd increase it if you got even more
                 variables [8 in this case])
                 bag.fraction=0.5, # tells R to train on half of the bags
                 shrinkage=0.01, # this is the speed of Learning. A tree which
                 learns slower, learns better. You want values btw 0.01 and 0.001
                 train.fraction=1, # optimising factor. Always leave it at 1
                 (if we modify it, we'd tell R to train on a smaller fraction)
                 cv.folds=3) # cross-validation factor

#
```

What did we define in this code?

- distribution="poisson" ==> The distribution. In Regr trees, you don't have to worry about transformations of any predictors (eg. log), as regression trees do not assume anything about linearity, hom. of var, etc. = The shape is irrelevant. The regr trees automatically get the interaction. However, you have to nominate a LOSS FUNCTION, sth that looks like a family suitable for that model (in this case we got counts -> so we define that POISSON distribution is appropriate).
- var.monotone=c(0,1,1,1,1,1,1,1) -> The polarity of expected trends: We can force the trend to be monotonic (eg to go up). Example:

^ | .. | ... | .. | _____ >

we turn this distribution, which has got peaks and downs, into a monotonic distribution

V

```

^ | /
| /
| | / | / _____ >

```

This parameter is defined based on the FIRST row of the scatterplotMatrix that we plotted above. In this case, we got 8 variables that might influence the numbers of rings (so 8 numbers in total). For each variable, we define in the same order than in the scatterplotMatrix any of the following parameters: 0=no interaction, 1=pos interaction, -1=neg interaction. In the case of CATEGORICAL variables though, we ALWAYS write a 0 (that is why we have a 0 for the parameter SEX). In detail we defined the following influences: (1. variable) SEX -> categorical variable, so we define it as a 0 (2. variable) LENGTH -> seems to have a positive influence on number of rings = 1 (3. variable) DIAMETER -> seems to have a positive influence on number of rings = 1 (4. variable) HEIGHT -> seems to have a positive influence on number of rings = 1 (5. variable) WHOLE_WEIGHT -> seems to have a positive influence on number of rings = 1 (6. variable) MEAT_WEIGHT -> seems to have a positive influence on number of rings = 1 (7. variable) GUT_WEIGHT -> seems to have a positive influence on number of rings = 1 (8. variable) SHELL_WEIGHT -> seems to have a positive influence on number of rings = 1 And this written altogether would look like: var.monotone=c(0,1,1,1,1,1,1,1)

- number of trees
- size of trees = How complex深深 we think the interactions are. If we want a tree depth of 5 it is like saying "we got 5-way interactions". Some people like to have 30 observations per predictor. Nevertheless, the tree depth is a way of getting how many orders of interactions we expect. If we had 9 variables, we could go <9 interactions. In this case, we got 8 variables, so let's just go with 5 interactions
- speed of the learning = the lower you make it, the more trees you are gonna need to avoid overfitting (I think). Leave it btw 0.1-0.001
- optimising factor. Always leave it at 1 (if we modify it, we'd tell R to train on a smaller fraction)
- cross-validation factor = How many times does R repeat each tree? Somewhere btw 3-10 is good.

*Although R repeats each tree 3x, it is not sequential. We therefore cannot run on multiple cores; and we cannot define core=3

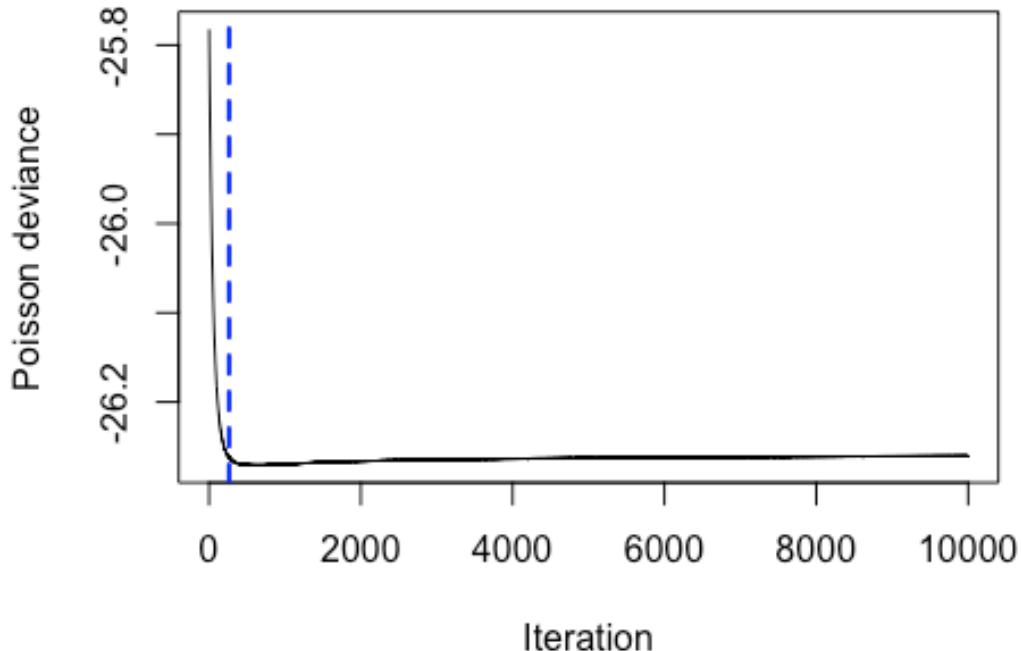
3- Finding the best iteration

This is equivalent to tossing trees out

Option 1- Bag method

```
(best.iter=gbm.perf(abalone.gbm,method='OOB'))
```

```
## OOB generally underestimates the optimal number of iterations although  
predictive performance is reasonably competitive. Using cv_folds>1 when  
calling gbm usually results in improved predictive performance.
```

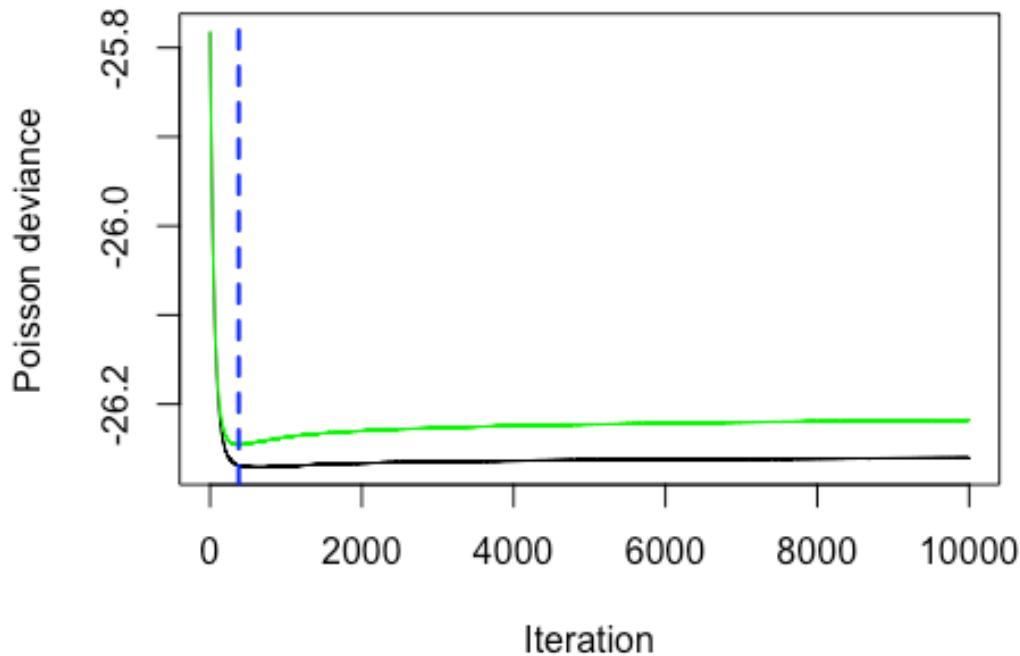


```
## [1] 264  
## attr(,"smoother")  
## Call:  
## loess(formula = object$oopbag.improve ~ x, enp.target = min(max(4,  
##           length(x)/10), 50))  
##  
## Number of Observations: 10000  
## Equivalent Number of Parameters: 39.99  
## Residual Standard Error: 8.359e-05
```

R tells us that it needs around 264 trees to learn the model. Usually, it is around 1000 trees. So we might need to tell R to learn slower.

Option 2- Cross-validation method (preferred method, although you can also check the bag method)

```
(best.iter=gbm.perf(abalone.gbm,method='cv'))
```



```
## [1] 378
```

Tells us that we need 377 trees, which is slightly better, as R is learning slower. We WANT TO AIM TO A BIT CLOSER TO 1000. The reason this is important is because, whether it is 264 trees, or 260 trees, will make a diff in the pattern that it has learnt. BUT if we tell it to use ~1000 trees, that difference won't really matter.

Murray will therefore tell R to learn slower. Let's change the model.

Re-fitting the model, telling it to learn slower and to use less trees

```
abalone.gbm=gbm(RINGS~SEX+LENGTH+DIAMETER+HEIGHT+WHOLE_WEIGHT+MEAT_WEIGHT+GUT_WEIGHT+SHELL_WEIGHT,
                  data=abalone.train,
                  distribution="poisson", # we define the Loss function, the
distribution that our data seems to be more likely
                  var.monotone=c(0,1,1,1,1,1,1,1), # defining the polarity of
the expected trends. This is defined based on the first row of the
scatterplotMatrix that we plotted above. In this case, we got 8 variables
that might influence the numbers of rings. For each variable, we define in
order any of the following parameters: 0=no interaction, 1=pos interaction, -
1=neg interaction. In the case of CATEGORICAL variables though, we ALWAYS
write a 0
                  n.trees=5000, # we tell R to use 5,000 trees instead
```

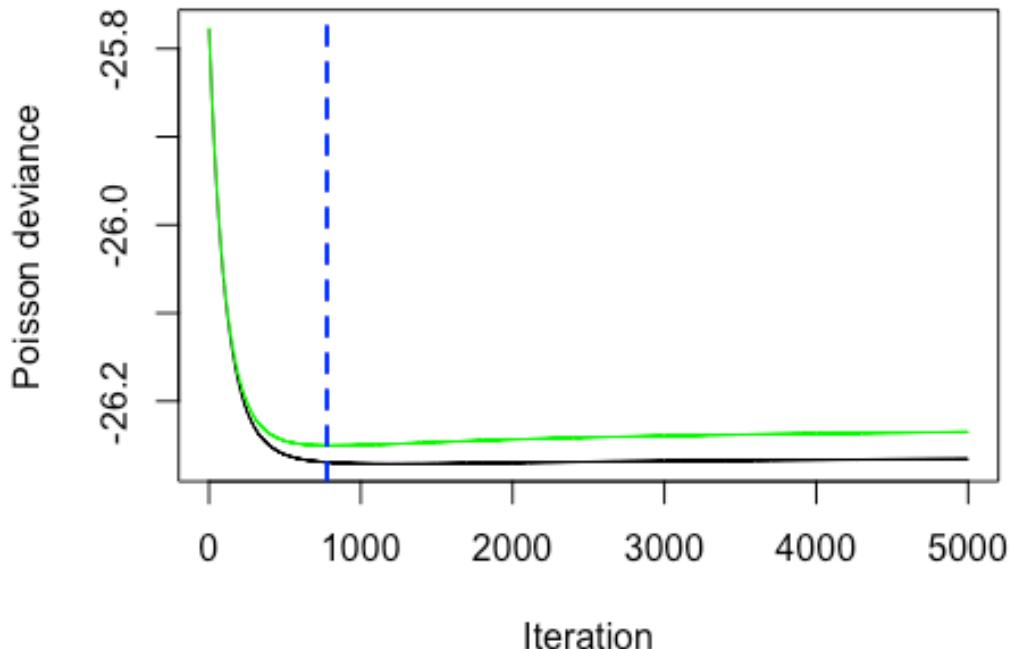
```

interaction.depth=5, # we define how big each tree needs to
be (5 is kind of default, but you'd increase it if you got even more
variables [8 in this case]). This variable depends on the complexity of the
model. The function gbm_step (I think that's the name) can give you a range
of values for shrinkage, tree depth, and will put in and put out your
variables to find out what works best. It therefore takes a while to run.
bag.fraction=0.5, # tells R to train on half of the bags
shrinkage=0.005, # this is the speed of Learning. A tree
which learns slower, learns better. You want values btw 0.01 and 0.001
train.fraction=1, # optimising factor. Always leave it at 1
(if we modify it, we'd tell R to train on a smaller fraction)
cv.folds=3) # cross-validation factor

```

Let's repeat Option 2- Cross-validation method

```
(best.iter=gbm.perf(abalone.gbm,method='cv'))
```



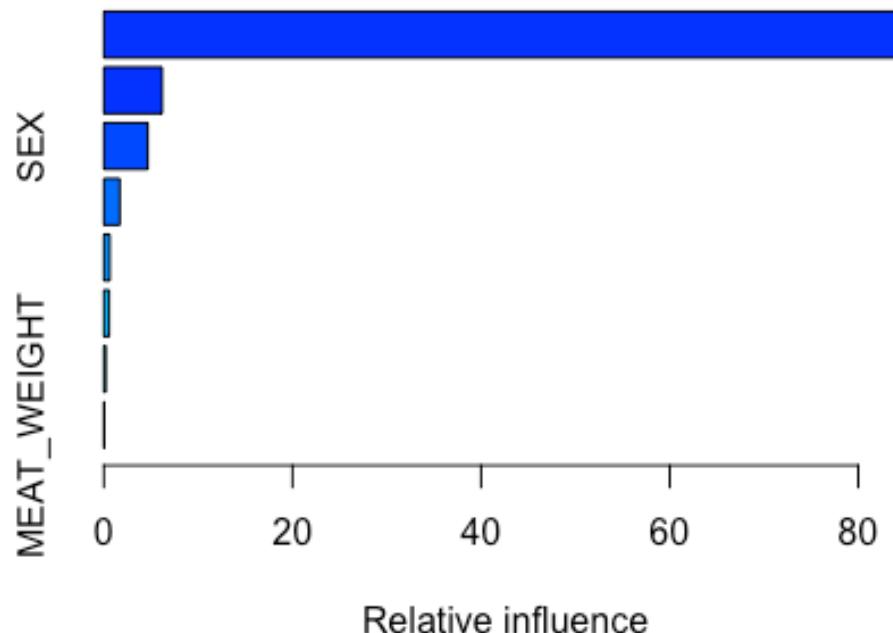
```
## [1] 777
```

We now get about 750 trees, which is close enough to 1000. If we had >1000 trees, that would also be fine. That means that, from now on, any code that we tell R to stop learning at ~750 trees

Explore relative influence of each of the predictors

The relative influence of each of the predictors is how many times did each predictor feature in various splits.

```
summary(abalone.gbm, n.trees=best.iter)
```



```
##           var      rel.inf
## SHELL_WEIGHT SHELL_WEIGHT 86.05818166
## HEIGHT       HEIGHT     6.18036572
## SEX          SEX      4.65818916
## DIAMETER     DIAMETER   1.68283044
## GUT_WEIGHT   GUT_WEIGHT 0.63193331
## LENGTH       LENGTH    0.55823865
## WHOLE_WEIGHT WHOLE_WEIGHT 0.20309927
## MEAT_WEIGHT  MEAT_WEIGHT 0.02716179
```

Our most influential variable is SHELL WEIGHT -> 88% of the splits would use that variable ALONE !

Correlated predictors don't really matter, BUT...they somehow DO. If we put corr pred in a linea rmodel, the estimate for both of those predictors are WRONG. One will be way over-inflated, and the other one will be way under-inflated. This is a problem in LM, where we

are predicting effect sizes. In this case, we wanna know which variables are most important. IF 2 variables are correlated, one will come around as most influential, and the other one (the less related to the response), will drop out, bot being estimated correctly.

So in this case, we might have SHELL WEIGHT, and a correlated variable to it. However, shell weight takes all the pride, and shadows its correlated variables out. The strongest parameter related to the response is the one more likely causing the response, and the other one is just correlating.

FOR PREDICTIONS AND DET THE MOST IMPORT VARIABLES THIS DOES NOT MATTER THOUGH.

With RANDOM FORESTS, we might get two variables showing that they are BOTH influencing the response, but they will SHARE that importance (we'd have a 50-50%, or 40-60% importance). In other words, a random forest would REDUCE the importance of the variables that actually drive the responses. Murray therefore prefers to test the above method, which actually gives a variable the importance it deserves.

How do we decide on which variableS to keep?

If we were to fit a linear model, which variables would we keep, and which ones would we toss out? By chance, if each are equally important, we would have $100/8=12.5\%$ of importance per variable (8=number of variables). Anything that is ABOVE that threshold, we'd say it is an important variable. Anything below, is NOT. So essentiallym in this case, we could toss out everything but SHELL WEIGHT.

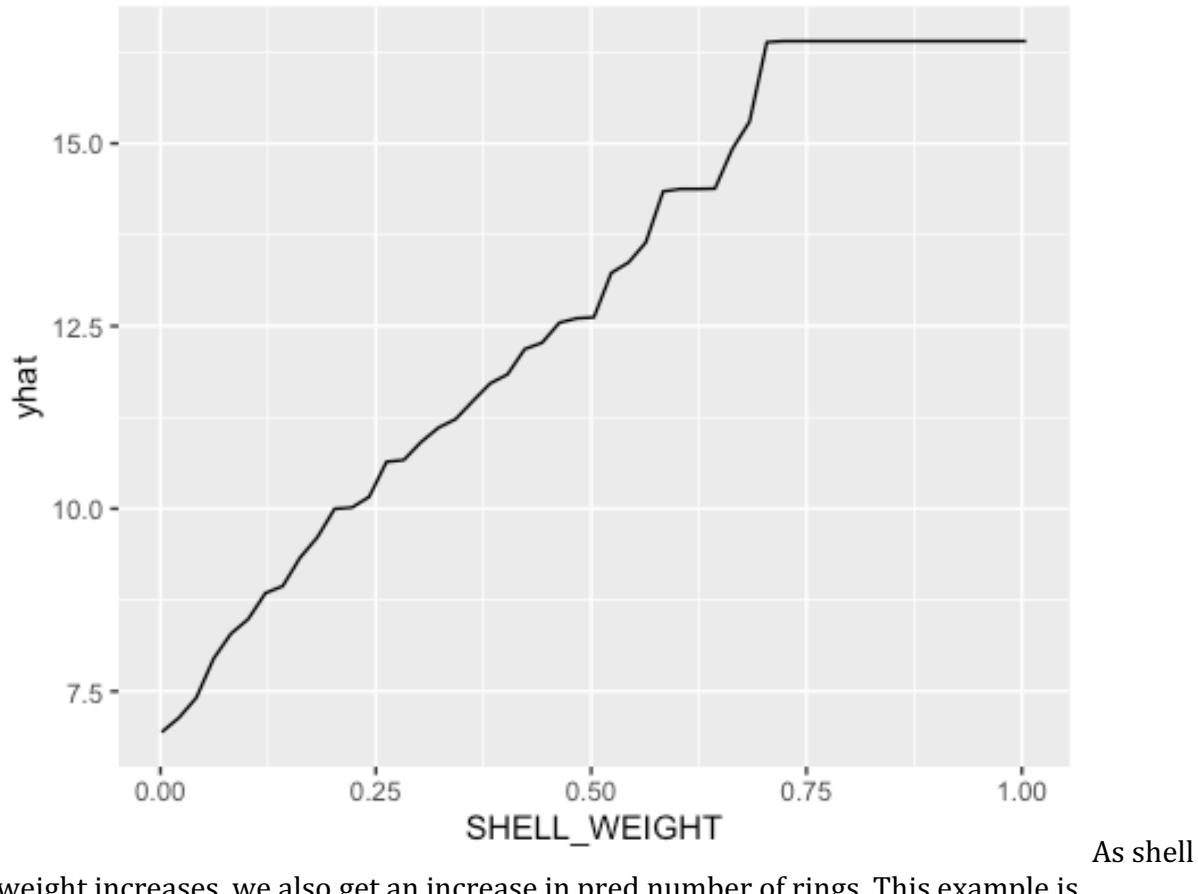
Explore partial effects

We nominate the variables that are driving the response, and predict how they influence it

```
abalone.gbm %>%
  pdp::partial(pred.var=c("SHELL_WEIGHT"), # we nominate the effect we wanna
               Look for (you can also write it without brackets)
               n.trees=best.iter,recursive=FALSE,inv.link=exp) %>% # we
               nominate the amount of trees and tell R to back transform
  autoplot()

## Warning: Use of `object[[1L]]` is discouraged. Use `$.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]]` is discouraged. Use `$.data[["yhat"]]]`#
## instead.
```



weight increases, we also get an increase in pred number of rings. This example is straightforward, but often you don't get such a straight line.

Explore accuracy

We got a dataset that we withheld, with 100 rows. Let's use it for the purpose of predicting the number of shells, using all variables

```
abalone.acc<-abalone.test %>%
  bind_cols(Pred=predict(abalone.gbm,newdata=abalone.test,n.tree=best.iter,type="response"))
# we bind the columns together using a predict function similar to how we used them in LM, to predict and put it all together

head(abalone.acc)

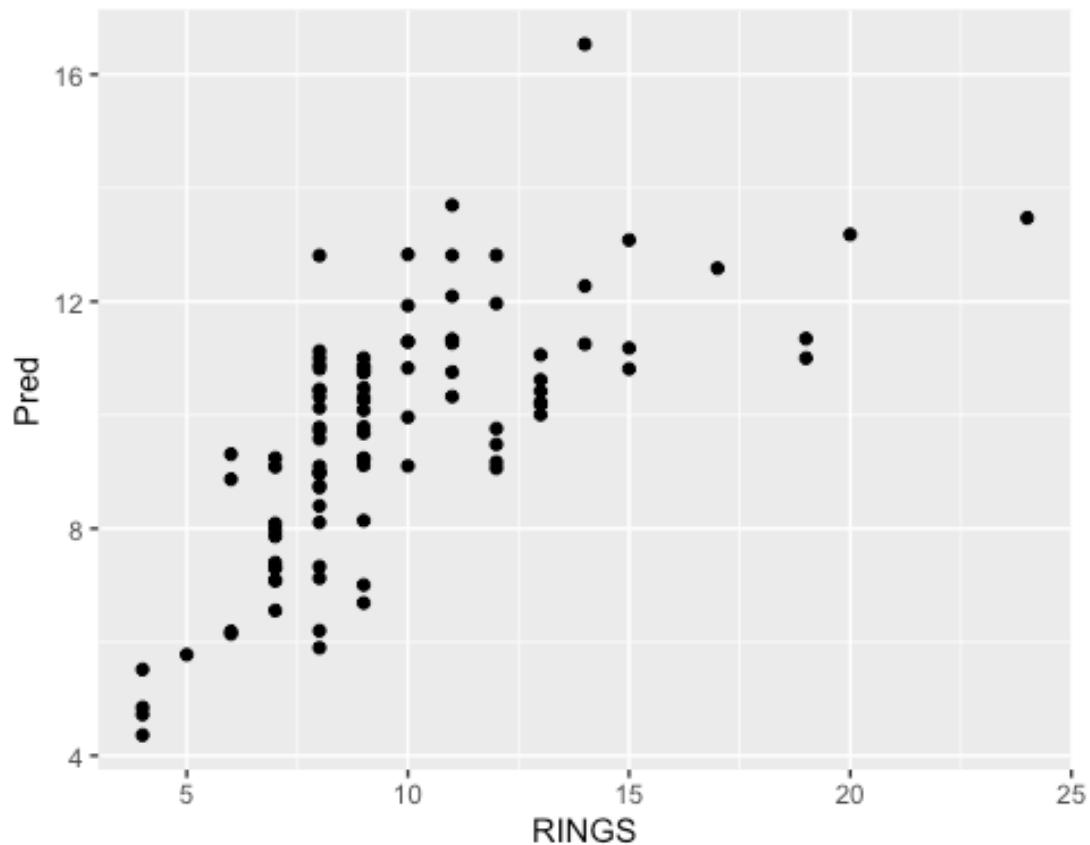
## # A tibble: 6 x 11
##   SEX    LENGTH DIAMETER HEIGHT WHOLE_WEIGHT MEAT_WEIGHT GUT_WEIGHT
##   <fct>   <dbl>     <dbl>   <dbl>       <dbl>        <dbl>        <dbl>
## 1 M      0.0437    0.0622  0.100      0.151      0.0345      0.0115
## 2 M      0.0446    0.0622  0.100      0.151      0.0345      0.0115
## 3 M      0.0455    0.0622  0.100      0.151      0.0345      0.0115
## 4 M      0.0464    0.0622  0.100      0.151      0.0345      0.0115
## 5 M      0.0473    0.0622  0.100      0.151      0.0345      0.0115
## 6 M      0.0482    0.0622  0.100      0.151      0.0345      0.0115
```

```
<dbl>
## 1 M      0.44     0.325   0.08       0.413    0.144    0.102
0.13
## 2 I      0.42     0.32     0.1        0.34     0.174    0.05
0.0945
## 3 I      0.355    0.28     0.11      0.224    0.0815   0.0525
0.08
## 4 M      0.175    0.125   0.04       0.024    0.0095   0.006
0.005
## 5 I      0.535    0.4      0.135     0.775    0.368    0.208
0.206
## 6 I      0.435    0.335   0.1        0.324    0.135    0.0785
0.098
## # ... with 3 more variables: RINGS <dbl>, AGE <dbl>, Pred <dbl>
with(abalone.acc,cor(RINGS,Pred))
## [1] 0.6757684
```

The accuracy of our model to predict the number of rings is around 69%

Plot

```
abalone.acc %>%
  ggplot() +
  geom_point(aes(y=Pred,x=RINGS))
```



There is a relationship, not massively tight, but possibly enough to what the researchers were after.

In this case, researchers use rings as a proxy of age, and wanted to see what variables drive the number of rings. In theory, this is a bit silly, as we could have just tested what factors just drive AGE (and not number of rings).

Tuning - Murray did not go through any of this

Explore interactions - Murray did not go through any of this

The function gbm.step can give you a range of values for shrinkage, tree depth, and will put in and put out your variables to find out what works best. It therefore takes a while to run.
→ See code below

Random Forest

This sets up the ability to calculate IMPORTANCE of the variables driving the response.

```
library(randomForest)
abalone.rf = randomForest(RINGS ~ SEX + LENGTH + DIAMETER + HEIGHT +
```

```

WHOLE_WEIGHT + MEAT_WEIGHT + GUT_WEIGHT + SHELL_WEIGHT,
data=abalone, importance=TRUE,
ntree=1000)

abalone.imp = randomForest:::importance(abalone.rf) # add randomForest::: in
front of importance if it does not work
# This function does the following (Murray did not explain this):
## Rank by either:
## *MSE (mean decrease in accuracy)
## For each tree, calculate OOB prediction error.
## This also done after permuting predictors.
## Then average diff of prediction errors for each tree
## *NodePurity (mean decrease in node impurity)
## Measure of the total decline of impurity due to each
## predictor averaged over trees

100*abalone.imp/sum(abalone.imp) # express it as relative importance (turns
it into percentages)

## %IncMSE IncNodePurity
## SEX 0.13006014 2.951724
## LENGTH 0.05845515 8.383552
## DIAMETER 0.07785303 9.791761
## HEIGHT 0.07647123 13.537655
## WHOLE_WEIGHT 0.06623944 14.605638
## MEAT_WEIGHT 0.13756292 13.807198
## GUT_WEIGHT 0.08830085 12.260254
## SHELL_WEIGHT 0.11481460 23.912459

```

Output: %IncMSE IncNodePurity SEX 0.12909785 2.982593 LENGTH 0.05550787
 8.581556 DIAMETER 0.07398878 9.870201 HEIGHT 0.08060450 13.150743
 WHOLE_WEIGHT 0.06422403 14.696833 MEAT_WEIGHT 0.14388773 13.718813
 GUT_WEIGHT 0.07259862 11.933380 SHELL_WEIGHT 0.11182072 24.334149

This function provides two different metrics of influence. And we have diff interpretations for them: - %IncMSE = Change in mean sq error (MSE) that we attribute to the particular predictor. MSE is the avg, the mean sq of the residuals. The larger the MSE, the more noisy the data is, and the less you explained. Cause an error is the diff btw observed and expected = how far away our observed values are from the expected. It is how much the MSE is DECREASED when we add Shell Weight (the variable that defines most of the response). In this case, the MSE is reduced by 0.11 -> This is an absolute decline, but the numbers don't mean anything. You understand these values ONLY when comparing them to the others = The higher the number, the better (=the more that variable modifies the response variable). In this case, there is THREE variables in total which substantially alter number of Rings = SEX, MEAT_WEIGHT and SHELL_WEIGHT.

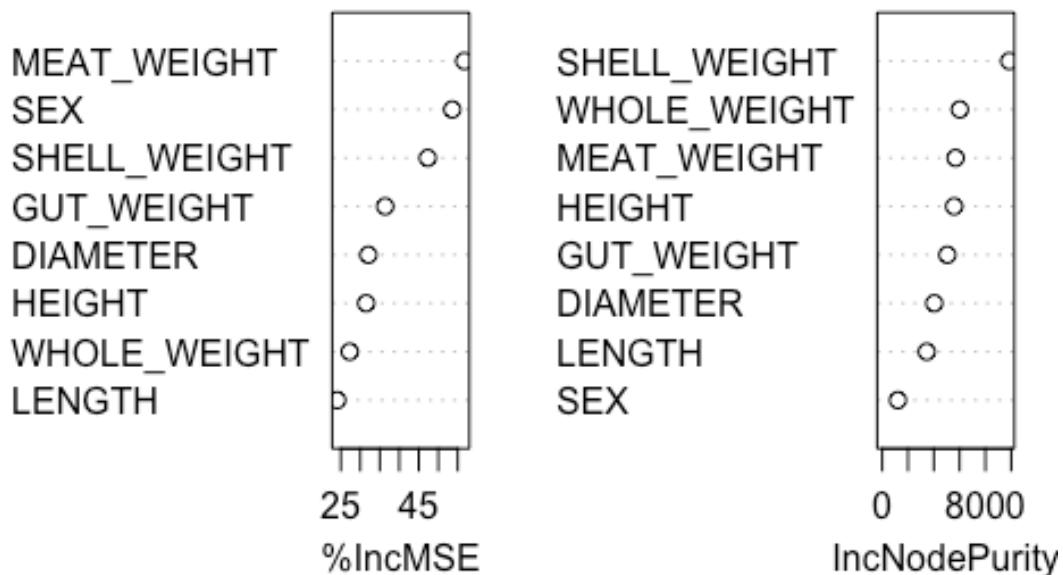
- IncNodePurity = This is a measure of Impurity, of how well each variable produces the results. If we were to make buckets of our predictions, and we predict that a specific variable should be in bucket A, and the other variable in bucket B, R compares the

buckets to what it should have been, and tells you the number of Impurities (~errors). In this case, we are looking at which values REDUCE impurities the most = The higher the number, the better (=the more that variable modifies the response variable). In this case, SHELL_WEIGHT reduces the numbers of impurities by 23%.

Partial plot

```
varImpPlot(abalone.rf)
```

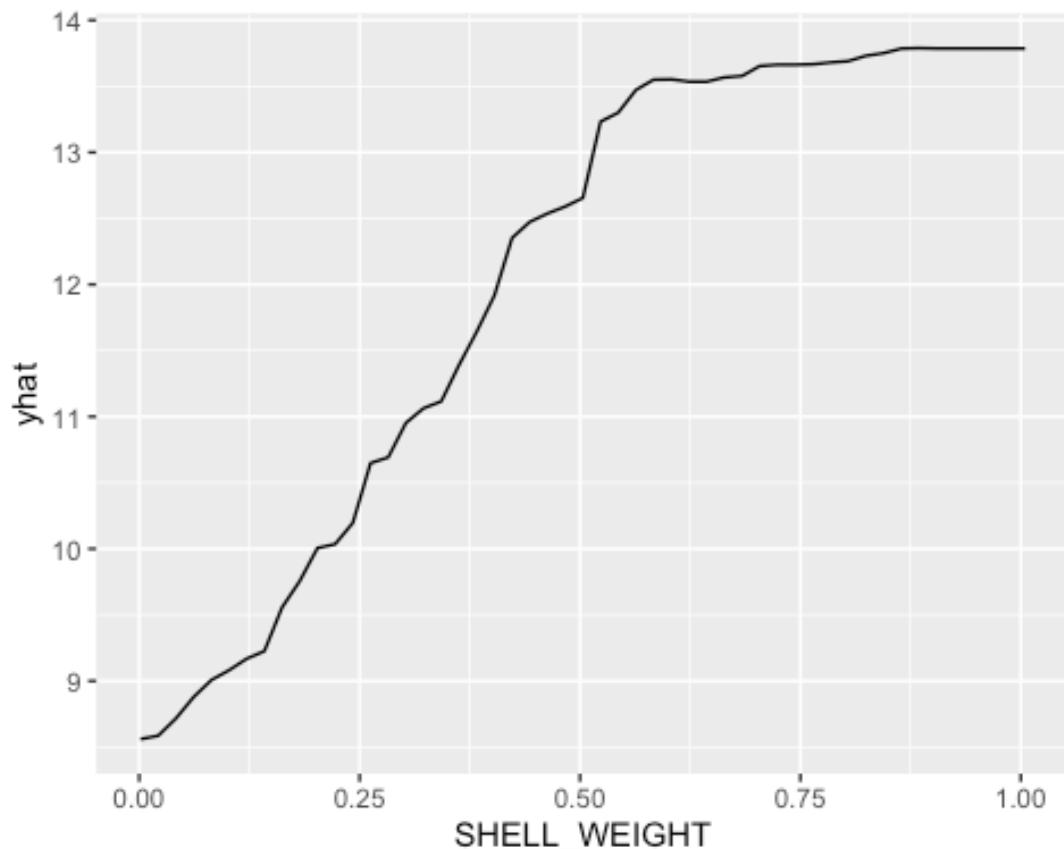
abalone.rf



```
## use brute force
abalone.rf %>% pdp::partial('SHELL_WEIGHT') %>% autoplot

## Warning: Use of `object[[1L]]` is discouraged. Use `$.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]]` is discouraged. Use `$.data[["yhat"]]]` instead.
```



Based

on how the variables compare in terms of influencing the response, you choose the most sensible variable, and plot it. BUT WHERE THE HECK ARE OUR CONF INT? There aren't any! Regression trees are where stats meets machine learning, which does only give conf int if you did bootstrap.

Bootstrapping:

Bootstrapping is a very powerful technique that can be applied to anything, even just a simple regression. What you do in bootstrapping is that you run the analysis (like we have done now), you take the data, and take a subset of it, eg. 1000 rows. We will do this with replacement, creating a new dataset with the same amount of rows than our original dataset, however some rows will be duplicate and others will be missing. We then do this again, taking another set of 1000 rows, and rerun the model. Then, we do it again, and again, and again... We might do this 100-1000 times. Our estimate of the effect would get more accurate each time. You can then look at the quantiles and conf int of this analysis, and plot those. For time constraint reasons, we haven't done it.

How would you do bootstrapping, creating a simple loop?

The steps are the following, however if we ran the whole analysis, it would take hours and hours... 1- Create a bootstrap dataset = You'd use the function sample to determine which

rows we want (like we did at the begining of this Markdown file) 2- You want to run the gbm (like we did at the begining of this Markdown file) 3- Use predict for the trend, or rather for partial effects 4- Calculate the relative importance You first write the code as if you wanted R to do it once. Then, you write the loop so that R does it again and again. To prevent using loops, you could use MAP (but it is quite complicated to work with, and you can't debug it). But it is quite useful when working on HPCs.

And in more detail... 1- Create an empty container waiting to be filled, a list (vector) of 10 empty things

```
rf.list<-vector("list",10)
```

2- Define how many times this function has to be performed

```
for(i in 1:10){ # perform this function 10x
rf.list[[i]]<-rnorm(1,0,1)} # We ask it to loop 10x, and each time do sth (eg
"fit a regression tree"). In this case, we are telling R to fit a random
number from a regression.
```

```
rf.list

## [[1]]
## [1] 0.4604984
##
## [[2]]
## [1] 0.3447661
##
## [[3]]
## [1] 0.8369627
##
## [[4]]
## [1] -0.641719
##
## [[5]]
## [1] 0.4462364
##
## [[6]]
## [1] -0.5688412
##
## [[7]]
## [1] -1.262425
##
## [[8]]
## [1] 0.616707
##
## [[9]]
## [1] -0.6878006
##
## [[10]]
## [1] -0.8087712
```

These are the random numbers that were allocated to each of our empty cells in the vector. This could be a whole dataframe of our predictions. The LOOPING puts them all together. Each loop will make a new dataset, repeating the calculations.

To save some time, each loop can also be run in PARALLEL (1-5, 6-10, etc). But this is next level.

22_RegressionTrees2

Sara Kophamel

18/12/2020

Open: tree_example2.Rmd

Preparations

Load the necessary libraries

```
library(gbm)          #for gradient boosted models  
## Loaded gbm 2.1.8  
  
library(car)  
  
## Loading required package: carData  
  
library(dismo)  
  
## Loading required package: raster  
  
## Loading required package: sp  
  
library(pdp)  
library(ggfortify)  
  
## Loading required package: ggplot2  
  
library(randomForest)  
  
## randomForest 4.6-14  
  
## Type rfNews() to see new features/changes/bug fixes.  
  
##  
## Attaching package: 'randomForest'  
  
## The following object is masked from 'package:ggplot2':  
##  
##     margin  
  
library(tidyverse)  
  
## — Attaching packages
```

tidyverse 1.3.0 —

```

## ✓ tibble  3.0.3    ✓ dplyr   1.0.2
## ✓ tidyr   1.1.2    ✓ stringr 1.4.0
## ✓ readr   1.3.1    ✓forcats 0.5.0
## ✓ purrr   0.3.4

## — Conflicts
----- tidyverse_conflicts() -----
## x dplyr::combine()      masks randomForest::combine()
## x tidyr::extract()      masks raster::extract()
## x dplyr::filter()       masks stats::filter()
## x dplyr::lag()          masks stats::lag()
## x randomForest::margin() masks ggplot2::margin()
## x purrr::partial()      masks pdp::partial()
## x dplyr::recode()       masks car::recode()
## x dplyr::select()       masks raster::select()
## x purrr::some()         masks car::some()

library(patchwork)

##
## Attaching package: 'patchwork'

## The following object is masked from 'package:raster':
##
##     area

library(stars)

## Loading required package: abind

## Loading required package: sf

## Linking to GEOS 3.7.2, GDAL 2.4.2, PROJ 5.2.0

```

Scenario

@Leathwick-2008-1481 compiled capture data of short-finned eels (*Anguilla australis*) within New Zealand freshwater streams to explore the distribution of the eels for conservation purposes. The goal was to be able to model the presence/absence of the eels against a range of environmental characteristics so as to predict their more broader occurrences and identify which predictors are the most important in the predictions.



eel

Format of leathwick.csv data file

Site	Angaus	SegSumT	SegTSeas	SegLowFlow	...
1	0	16.0	-0.10	1.036	...
2	1	18.7	1.51	1.003	...
3	0	18.3	0.37	1.001	...
4	0	16.7	-3.80	1.000	...
5	1	17.2	0.33	1.005	...
6	0	15.1	1.83	1.015	...
..
Site	Unique label for each site.				
Angaus	Presence (1) or absence (0) of <i>Anguilla australis</i> eels				
SegSumT	Summer air temperature (degrees celcius) at the river segment scale				
SegTSeas	Winter temperature normalised to January temperature at the river segment scale				
SegLowFlow	Forth root transformed low flow rate at the river segment scale				
DSDist	Distance to coast (km) (a downstream predictor)				
DSDam	Presence of known downstream obstructions (a downstream				

	predictor)
DSMaxSlope	Maximum downstream slope (a downstream predictor)
USAvgT	Upstream average temperture (normalised for the river segment)
USRainDays	Number of rainy days recorded in the upstream catchment
USSlope	Slope of the river upstream
USNative	Percentage of the upstream riparian vegetation that is native
Method	Method used to capture the eels (categorical predictor)
LocSed	Weighted average of the proportional cover of bed sediment (1=mud, 2=sand, 3=fine gravel, 4=course gravel, 5=cobble, 6=boulder, 7=bedrock)

Read in the data

```

leathwick = read_csv('data/leathwick.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   Site = col_double(),
##   Angaus = col_double(),
##   SegSumT = col_double(),
##   SegTSeas = col_double(),
##   SegLowFlow = col_double(),
##   DSDist = col_double(),
##   DSMaxSlope = col_double(),
##   USAvgT = col_double(),
##   USRainDays = col_double(),
##   USSlope = col_double(),
##   USNative = col_double(),
##   DSDam = col_double(),
##   Method = col_character(),
##   LocSed = col_double()
## )
glimpse(leathwick)

## Rows: 1,000
## Columns: 14

```

```

## $ Site      <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, ...
## $ Angaus    <dbl> 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1,
0, 0...
## $ SegSumT   <dbl> 16.0, 18.7, 18.3, 16.7, 17.2, 15.1, 12.7, 18.1, 18.9,
18.2...
## $ SegTSeas   <dbl> -0.10, 1.51, 0.37, -3.80, 0.33, 1.83, 2.17, 1.00, 1.59,
0...
## $ SegLowFlow <dbl> 1.036, 1.003, 1.001, 1.000, 1.005, 1.015, 1.001, 1.002,
1...
## $ DSDist     <dbl> 50.2000, 132.5300, 107.4400, 166.8200, 3.9500, 11.1700,
42...
## $ DSMaxSlope <dbl> 0.57, 1.15, 0.57, 1.72, 1.15, 1.72, 2.86, 2.29, 0.40,
3.43...
## $ USAvgT     <dbl> 0.09, 0.20, 0.49, 0.90, -1.20, -0.20, 1.45, 0.47, 0.25,
0...
## $ USRainDays <dbl> 2.470, 1.153, 0.847, 0.210, 1.980, 3.300, 0.430, 1.153,
0...
## $ USSlope    <dbl> 9.8, 8.3, 0.4, 0.4, 21.9, 25.7, 9.6, 4.9, 9.8, 20.5,
3.9, ...
## $ USNative   <dbl> 0.81, 0.34, 0.00, 0.22, 0.96, 1.00, 0.09, 0.02, 0.74,
0.92...
## $ DSDam      <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
0, 0...
## $ Method     <chr> "electric", "electric", "spo", "electric",
"electric",
"el...
## $ LocSed     <dbl> 4.8, 2.0, 1.0, 4.0, 4.7, 4.5, 4.3, NA, NA, 3.6, 3.7,
1.0, ...

```

Declare categorical variables as factors

```

leathwick = leathwick %>% mutate(Method=factor(Method),
                                    LocSed=factor(LocSed)) %>%
  as.data.frame

summary(leathwick)

##      Site          Angaus        SegSumT        SegTSeas
## Min.   : 1.0   Min.   :0.000   Min.   :10.10   Min.   :-4.2000
## 1st Qu.:250.8 1st Qu.:0.000   1st Qu.:15.20   1st Qu.:-0.3450
## Median :500.5 Median :0.000   Median :16.50   Median : 0.5400
## Mean   :500.5 Mean   :0.202   Mean   :16.42   Mean   : 0.3123
## 3rd Qu.:750.2 3rd Qu.:0.000   3rd Qu.:18.10   3rd Qu.: 1.3000
## Max.   :1000.0 Max.   :1.000   Max.   :19.60   Max.   : 4.0900
##
##      SegLowFlow       DSDist       DSMaxSlope      USAvgT
## Min.   :1.000   Min.   : 0.07   Min.   : 0.000   Min.   :-6.0000
## 1st Qu.:1.002   1st Qu.: 10.27  1st Qu.: 0.570   1st Qu.:-1.0000
## Median :1.008   Median : 43.52  Median : 1.720   Median :-0.1000
## Mean   :1.086   Mean   : 74.50  Mean   : 3.022   Mean   :-0.4055
## 3rd Qu.:1.042   3rd Qu.:114.68 3rd Qu.: 4.570   3rd Qu.: 0.3900

```

```

## Max.   :3.818   Max.   :411.27  Max.   :20.300  Max.   : 1.8900
##
##      USRainDays      USSlope      USNative      DSDam
##  Min.   :0.2100  Min.   : 0.00  Min.   :0.0000  Min.   :0.000
##  1st Qu.:0.6175  1st Qu.: 8.50  1st Qu.:0.1700  1st Qu.:0.000
##  Median :1.0335  Median :13.90  Median :0.6600  Median :0.000
##  Mean   :1.2231  Mean   :14.49  Mean   :0.5712  Mean   :0.169
##  3rd Qu.:1.5140  3rd Qu.:20.52  3rd Qu.:0.9600  3rd Qu.:0.000
##  Max.   :3.3000  Max.   :33.50  Max.   :1.0000  Max.   :1.000
##
##      Method       LocSed
##  electric:768    1     : 49
##  mixture  : 59    4     : 42
##  net      : 69    4.8   : 42
##  spo      : 46    4.2   : 41
##  trap     : 58    4.5   : 38
##                  (Other):612
##                  NA's   :176

```

The converted sediment variable (LocSed) looks horrendous, with too many categories (Other:612). We'd have to re-run the data and remove the statement for turning LocSed into a factor. Let's turn it into numeric again.

```

leathwick = leathwick %>% mutate(Method=factor(Method),
                                    LocSed=as.numeric(LocSed)) %>%
  as.data.frame

summary(leathwick)

##      Site        Angaus      SegSumT      SegTSeas
##  Min.   : 1.0   Min.   :0.000  Min.   :10.10  Min.   :-4.2000
##  1st Qu.:250.8 1st Qu.:0.000  1st Qu.:15.20  1st Qu.:-0.3450
##  Median :500.5  Median :0.000  Median :16.50  Median : 0.5400
##  Mean   :500.5  Mean   :0.202  Mean   :16.42  Mean   : 0.3123
##  3rd Qu.:750.2 3rd Qu.:0.000  3rd Qu.:18.10  3rd Qu.: 1.3000
##  Max.   :1000.0  Max.   :1.000  Max.   :19.60  Max.   : 4.0900
##
##      SegLowFlow      DSDist      DSMaxSlope      USAvgT
##  Min.   :1.000  Min.   : 0.07  Min.   : 0.000  Min.   :-6.0000
##  1st Qu.:1.002  1st Qu.: 10.27 1st Qu.: 0.570  1st Qu.:-1.0000
##  Median :1.008  Median : 43.52  Median : 1.720  Median :-0.1000
##  Mean   :1.086  Mean   : 74.50  Mean   : 3.022  Mean   :-0.4055
##  3rd Qu.:1.042  3rd Qu.:114.68 3rd Qu.: 4.570  3rd Qu.: 0.3900
##  Max.   :3.818  Max.   :411.27  Max.   :20.300  Max.   : 1.8900
##
##      USRainDays      USSlope      USNative      DSDam
##  Min.   :0.2100  Min.   : 0.00  Min.   :0.0000  Min.   :0.000
##  1st Qu.:0.6175  1st Qu.: 8.50  1st Qu.:0.1700  1st Qu.:0.000
##  Median :1.0335  Median :13.90  Median :0.6600  Median :0.000
##  Mean   :1.2231  Mean   :14.49  Mean   :0.5712  Mean   :0.169

```

```
## 3rd Qu.:1.5140   3rd Qu.:20.52   3rd Qu.:0.9600   3rd Qu.:0.000
## Max.     :3.3000   Max.      :33.50   Max.     :1.0000   Max.     :1.000
##
##          Method          LocSed
## electric:768    Min.    : 1.00
## mixture  : 59   1st Qu.:21.00
## net      : 69   Median   :31.00
## spo      : 46   Mean    :28.88
## trap     : 58   3rd Qu.:39.00
##                     Max.    :60.00
##                     NA's    :176
```

That looks better, however we still got 176 missing values in LocSed. In boosted regression trees though., missing values don't matter! They do "imputation" instead. Regression trees would infer what those values would be, based on the other predictors. Most linear models would just remove that row altogether, whereas boosted regression trees do data imputation. A boosted tree says: "I don't have a value for this, so I'll just estimate it based on the other values".

Create a subset dataset

We now create a test dataset (essentially a copy of half of the previous dataset), to do our modelling on top of this one.

Read the test dataset as well

```
leathwick_test = read_csv('data/leathwick_test.csv', trim_ws=TRUE)

## Parsed with column specification:
## cols(
##   Angaus_obs = col_double(),
##   SegSumT = col_double(),
##   SegTSeas = col_double(),
##   SegLowFlow = col_double(),
##   DSDist = col_double(),
##   DSMaxSlope = col_double(),
##   USAvgT = col_double(),
##   USRainDays = col_double(),
##   USSlope = col_double(),
##   USNative = col_double(),
##   DSDam = col_double(),
##   Method = col_logical(),
##   LocSed = col_double()
## )

glimpse(leathwick_test)

## #> #> #> #> #> #> #> #> #> #> #> #> #> #> #> #> #> #> #> #>
```

```

0, 0...
## $ SegSumT      <dbl> 16.6, 16.8, 16.3, 15.6, 14.6, 16.7, 18.3, 16.4, 17.2,
15.7...
## $ SegTSeas     <dbl> 1.01, -0.51, 0.76, 1.56, -0.20, 0.80, 0.67, 1.44, -
0.67, ...
## $ SegLowFlow   <dbl> 1.017, 1.002, 1.023, 1.003, 1.023, 1.003, 1.005, 1.031,
1...
## $ DSDist       <dbl> 5.2300, 2.2400, 162.2800, 4.0500, 127.0300, 2.4800,
94.077...
## $ DSMaxSlope   <dbl> 0.29, 0.00, 5.14, 0.57, 1.72, 14.57, 0.57, 1.72, 1.72,
2.8...
## $ USAvgT        <dbl> -1.40, 0.27, -0.60, 1.14, -1.90, -1.50, 0.54, 0.75, -
1.80, ...
## $ USRainDays   <dbl> 1.980, 0.460, 0.806, 3.300, 1.940, 1.980, 0.847, 2.249,
0...
## $ USSlope       <dbl> 10.0, 0.7, 21.4, 0.9, 28.9, 22.0, 1.0, 4.8, 26.4, 26.1,
23...
## $ USNative      <dbl> 1.00, 0.00, 0.66, 0.75, 0.97, 0.99, 0.01, 0.02, 0.74,
0.99...
## $ DSDam         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0...
## $ Method        <lgl> NA, NA,
NA...
## $ LocSed        <dbl> 4.9, 2.3, 4.3, 1.0, NA, 2.8, NA, NA, 4.2, 4.1, 4.3,
1.2, N...
leathwick_test = leathwick_test %>% mutate(Method=factor(Method),
                                             LocSed=factor(LocSed)) %>%
  as.data.frame

```

Exploratory data analysis

```

scatterplotMatrix(~Angaus+SegSumT+SegTSeas+SegLowFlow+DSDist+DSMaxSlope+DSDam
+
  USAvgT+USRainDays+USSlope+USNative+Method+LocSed,  data=leathwick,
  diagonal=list(method='boxplot'))

## Warning in smoother(x[subs], y[subs], col = smoother.args$col[i], log.x =
## FALSE, : could not fit smooth

## Warning in smoother(x[subs], y[subs], col = smoother.args$col[i], log.x =
## FALSE, : could not fit smooth

## Warning in smoother(x[subs], y[subs], col = smoother.args$col[i], log.x =
## FALSE, : could not fit smooth

## Warning in smoother(x[subs], y[subs], col = smoother.args$col[i], log.x =
## FALSE, : could not fit smooth

## Warning in smoother(x[subs], y[subs], col = smoother.args$col[i], log.x =
## FALSE, : could not fit smooth

```

```
## FALSE, : could not fit smooth

## Warning in smoother(x[subs], y[subs], col = smoother.args$col[i], log.x =
## FALSE, : could not fit smooth

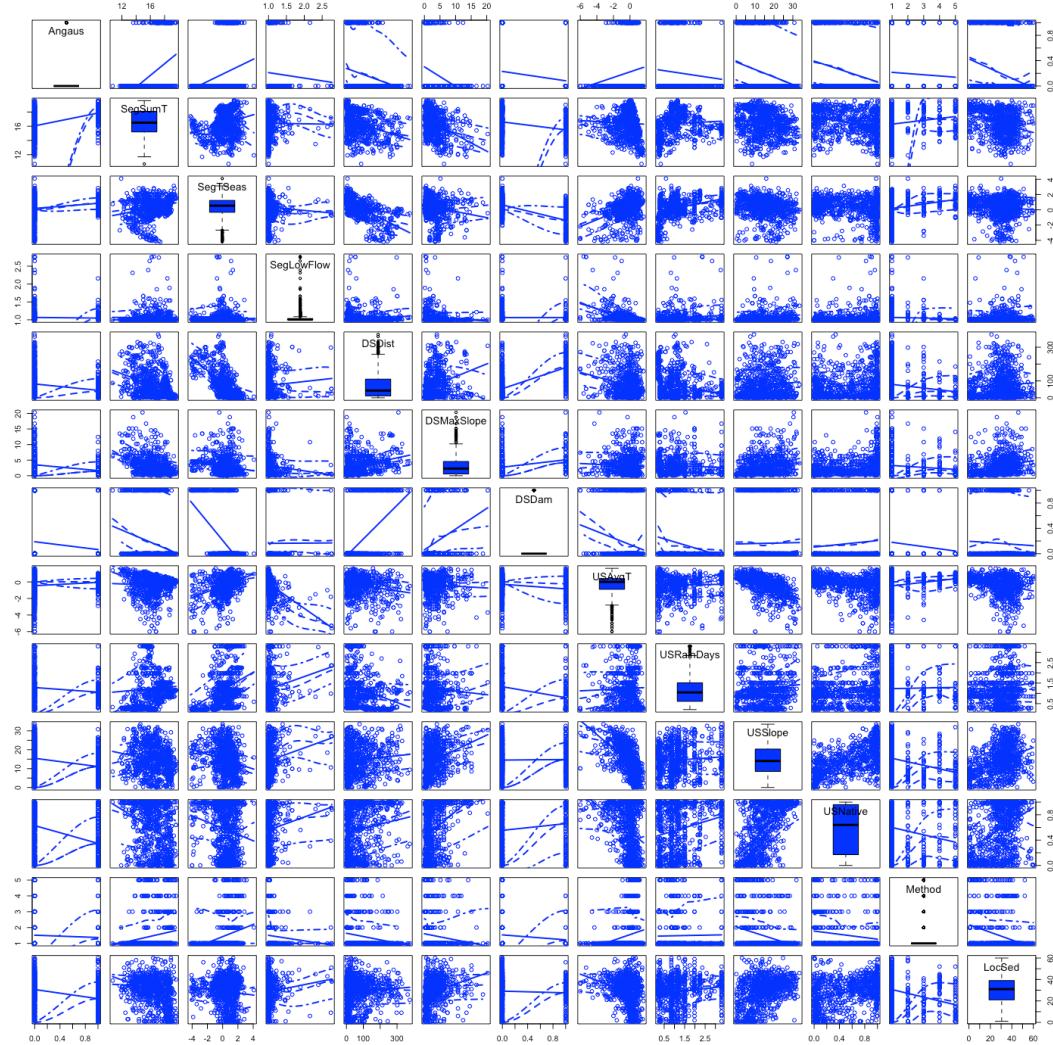
## Warning in smoother(x[subs], y[subs], col = smoother.args$col[i], log.x =
## FALSE, : could not fit smooth

## Warning in smoother(x[subs], y[subs], col = smoother.args$col[i], log.x =
## FALSE, : could not fit smooth

## Warning in smoother(x[subs], y[subs], col = smoother.args$col[i], log.x =
## FALSE, : could not fit smooth

## Warning in smoother(x[subs], y[subs], col = smoother.args$col[i], log.x =
## FALSE, : could not fit smooth

## Warning in smoother(x[subs], y[subs], col = smoother.args$col[i], log.x =
## FALSE, : could not fit smooth
```



This is

logistic regression, so there is only gonna be 1s and 0s for our response. To create the model, we have to force these variables to be monotonic (go up or down) -> Same thing happened in 21_RegressionTrees1.

If we keep the variables in this order, we will look at the first row and say:
 - 1st trend is pos
 - 2nd trend is pos - 3rd trend is slightly neg - 4th trend is neg etc

Our nominated datatype is PRESENCE ABSENCE. So the distribution is a Bernoulli.

```
set.seed(123) # This is for everybody in the class to get the same thing
               (easier to compare)
# DON't put a sed.seed within your for Loop, or we will get the same results
for each Loop. Write it BEFORE the Loop.
```

Fit the regression tree

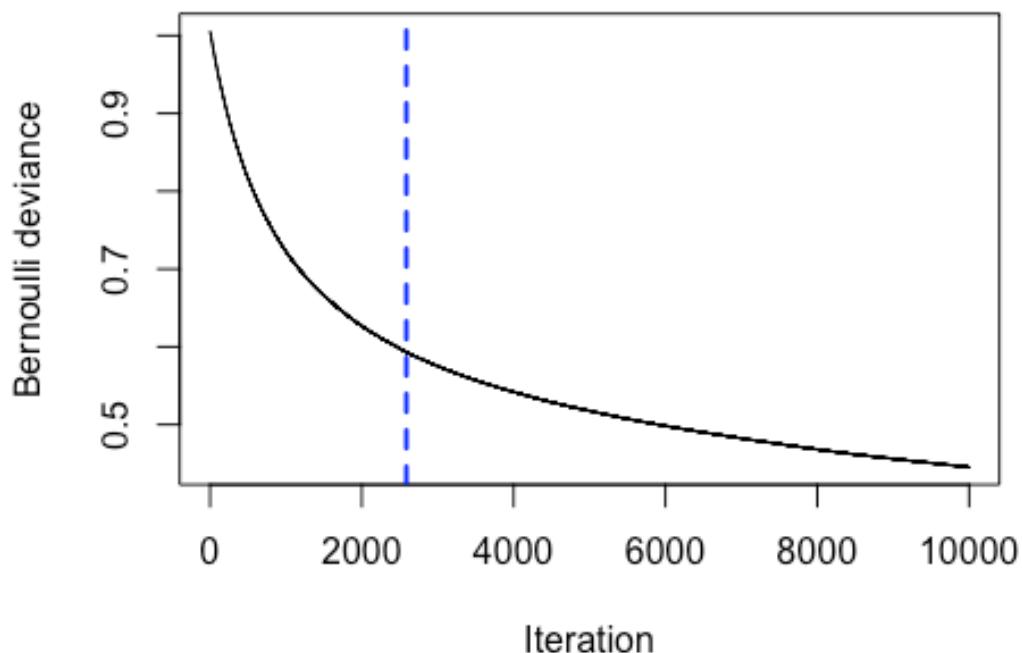
```
leathwick.gbm=gbm(Angaus~SegSumT+SegTSeas+SegLowFlow+DSDist+DSMaxSlope+DSDam+
USAvgT+USRainDays+USSlope+USNative+Method+LocSed,
  data=leathwick,
  distribution="bernoulli",
  var.monotone=c(1,1,0,-1,-1,0,1,-1,-1,-1,0,-1),
  n.trees=10000,
  interaction.depth=5,
  bag.fraction=0.5,
  shrinkage=0.001,
  train.fraction=1,
  cv.folds=3)
```

We now gotta work where to cut off the model (we enough trees, 10000, to run it)

Out of the bag method (use the next one, this one is only for comparison purposes)

```
(best.iter=gbm.perf(leathwick.gbm,method="OOB"))

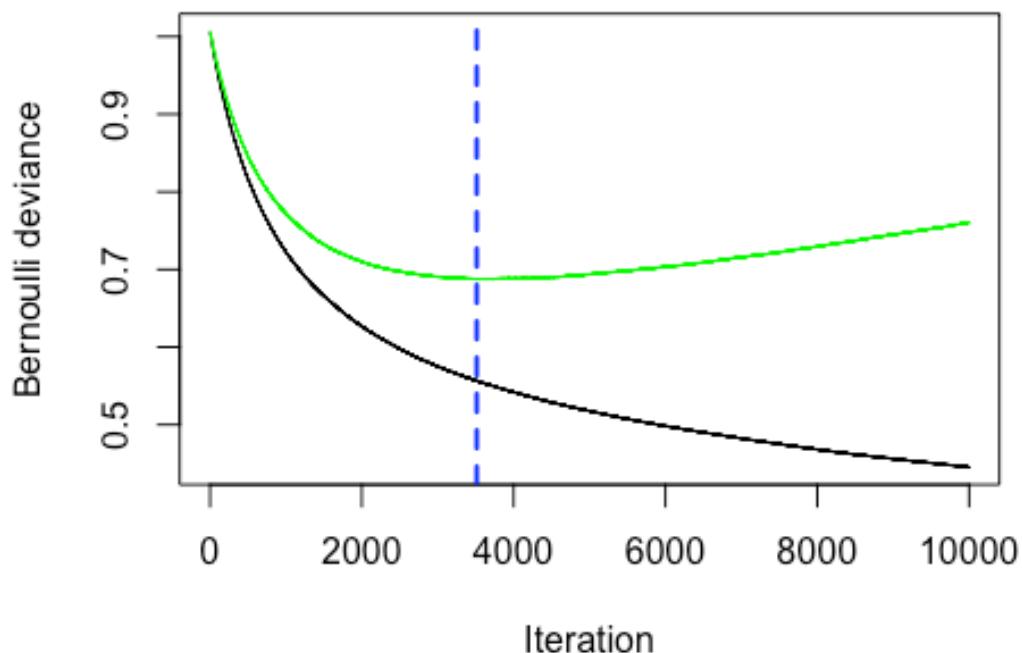
## OOB generally underestimates the optimal number of iterations although
predictive performance is reasonably competitive. Using cv_folds>1 when
calling gbm usually results in improved predictive performance.
```



```
## [1] 2587
## attr(),"smoother")
## Call:
## loess(formula = object$oobag.improve ~ x, enp.target = min(max(4,
##           length(x)/10), 50))
##
## Number of Observations: 10000
## Equivalent Number of Parameters: 39.99
## Residual Standard Error: 9.255e-06
```

Cross validation method (choose this one)

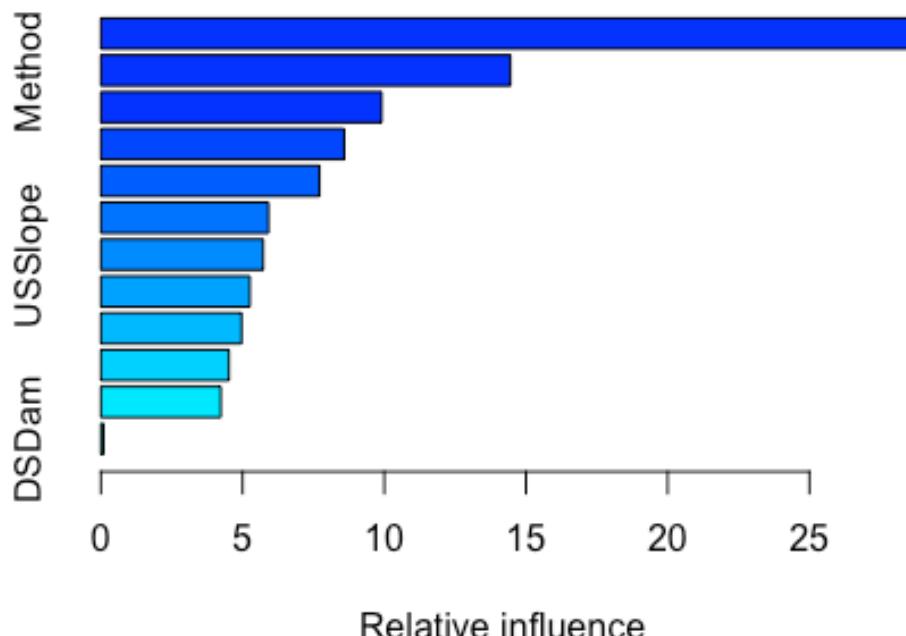
```
(best.iter=gbm.perf(leathwick.gbm,method="cv"))
```



```
## [1] 3514
```

Where the green line hits the dotted line = optimal number of trees; where the prediction starts to increase again. It is also the output in the console, under [1] = 2954 trees are needed.

```
nrow(leathwick)
## [1] 1000
summary(leathwick.gbm, n.trees=best.iter)
```



```
##           var      rel.inf
## SegSumT     SegSumT 28.63501344
## Method      Method 14.45082700
## USNative    USNative  9.90564910
## DSMaxSlope  DSMaxSlope 8.58537891
## USRainDays  USRainDays 7.71407726
## LocSed      LocSed  5.91390350
## USSlope     USSlope  5.72777322
## SegLowFlow  SegLowFlow 5.25915076
## USAvgT      USAvgT  4.96361713
## SegTSeas    SegTSeas  4.52091596
## DSDist      DSDist  4.23000236
## DSDam       DSDam   0.09369137
```

This output tells us that SegSumT, Method, USNative and DSMaxSlope have influence on abundance. We got 12 variables, so for a variable to be considered to have an effect, it should have at least $100/12=8.3\%$ of influence.

Partial plots

```
leathwick.gbm %>%
pdp::partial(pred.var="SegSumT", n.trees=best.iter, recursive=FALSE,
inv.link=plogis, # plogis turns the data into probability (exp would turn it
```

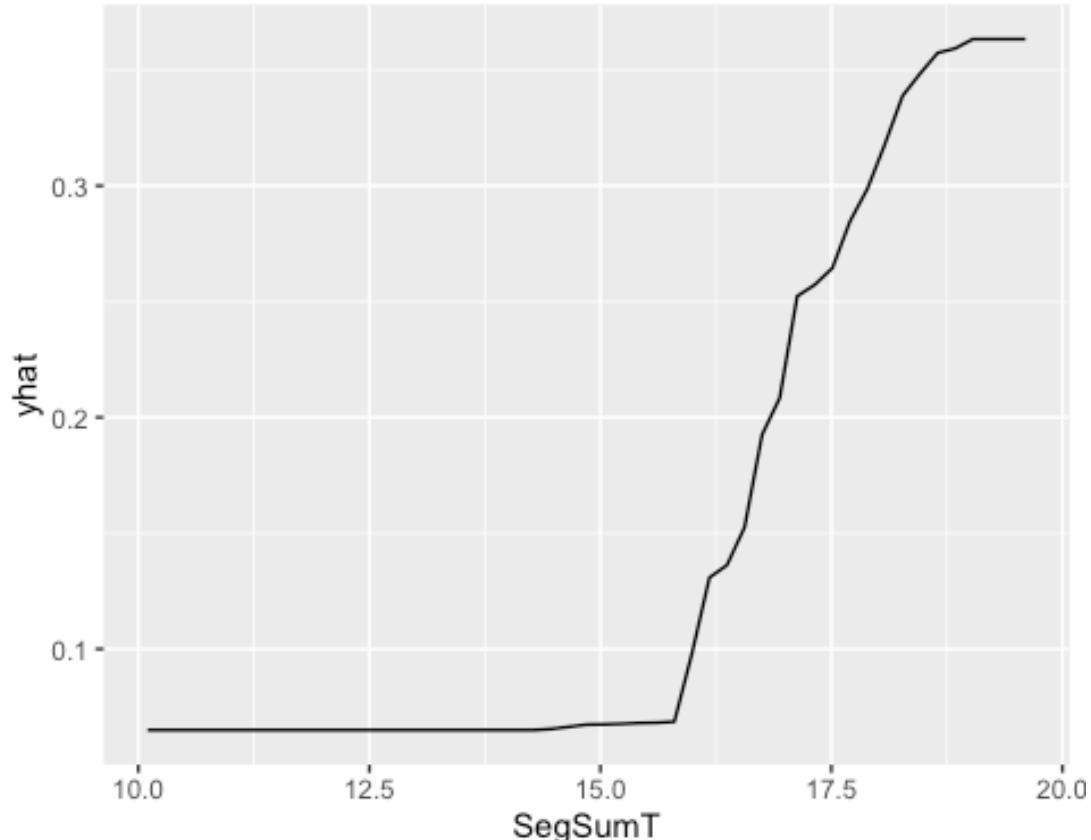
```

into Odds ratio)
  type="regression") %>%
  autoplot()

## Warning: Use of `object[[1L]]` is discouraged. Use `^.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]]` is discouraged. Use `^.data[["yhat"]]]` instead.

```



This plot

shows how SegSumT influences the response. This graph could be used to establish a threshold temperature, as it increases from a specific point.

We therefore have to use LOOPS, cause we will make a panel that makes estimates for the predictors. We gotta supply the name of the variable. We just need a list of all the variables, so we tell R to make a loop through that list and substitute each variable after each loop.

```

nms<-colnames(leathwick)

nms

## [1] "Site"        "Angaus"       "SegSumT"      "SegTSeas"     "SegLowFlow"
## [6] "DSDist"      "DSMaxSlope"   "USAvgT"       "USRainDays"   "USSlope"
## [11] "USNative"    "DSDam"        "Method"       "LocSed"

```

We don't want the first two variables, as they are NOT predictors (Site is the marker for the site, and Angaus is the abundance of eels, the response). In total, we want to check on 12 variables.

```
p<-vector("list",12) # Create an empty vector for our 12 variables

names(p)<-nms[3:14] # we tell R to only take variables 3-12

for(nm in nms[3:14]){ # the first loop it runs will be called SegSumT (nm is set to this)
  print(nm) # The reason we put this at the start is that, if something fails, we can diagnose what the problem is, checking the variable with T
  p[[nm]]<-leathwick.gbm %>% pdp::partial(pred.var=nm,
                                               n.trees=best.iter,
                                               inv.link=plogis,
                                               recursive=FALSE,
                                               type="regression") %>%
  autoplot() + ylim(0, 1) # write two spaces
}

## [1] "SegSumT"
## [1] "SegTSeas"
## [1] "SegLowFlow"
## [1] "DSDist"
## [1] "DSMaxSlope"
## [1] "USAvgT"
## [1] "USRainDays"
## [1] "USSlope"
## [1] "USNative"
## [1] "DSDam"
## [1] "Method"
## [1] "LocSed"
```

It should loop through now

```
do.call("grid.arrange", p)

## Warning: Use of `object[[1L]]` is discouraged. Use `$.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]]` is discouraged. Use `$.data[["yhat"]]]` instead.

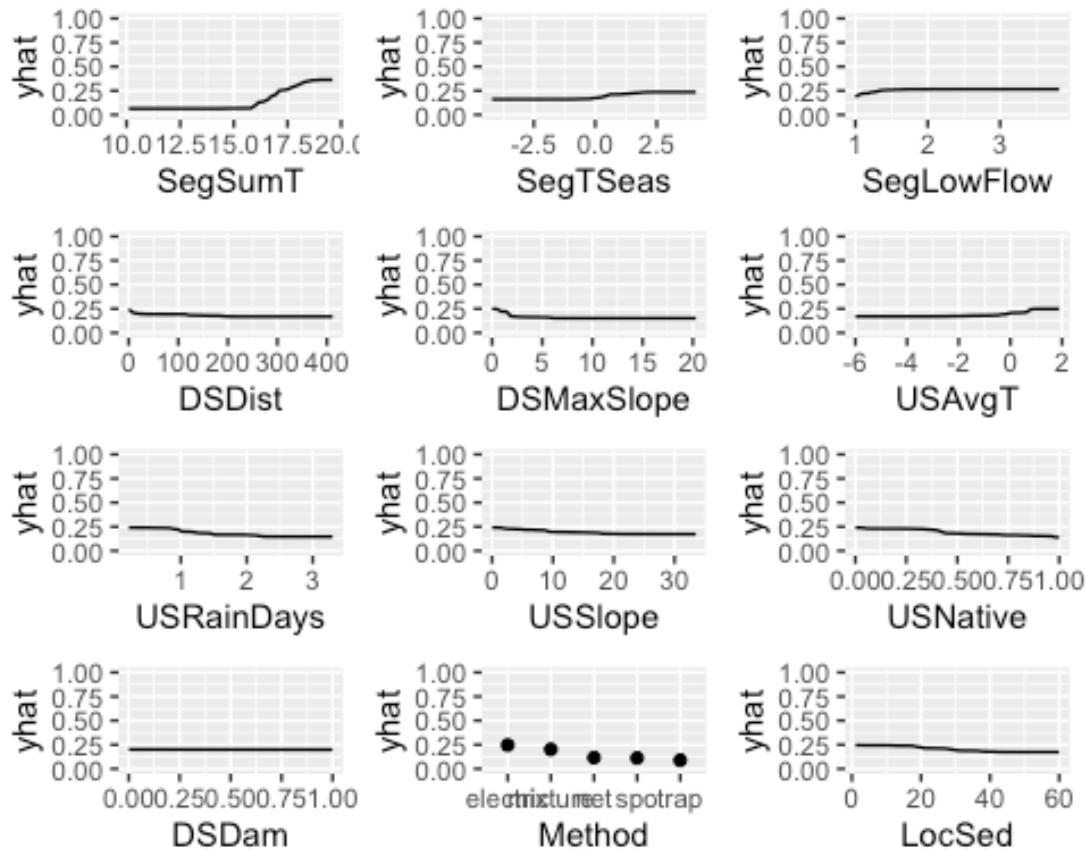
## Warning: Use of `object[[1L]]` is discouraged. Use `$.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]]` is discouraged. Use `$.data[["yhat"]]]` instead.

## Warning: Use of `object[[1L]]` is discouraged. Use `$.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]]` is discouraged. Use `$.data[["yhat"]]]` instead.
```

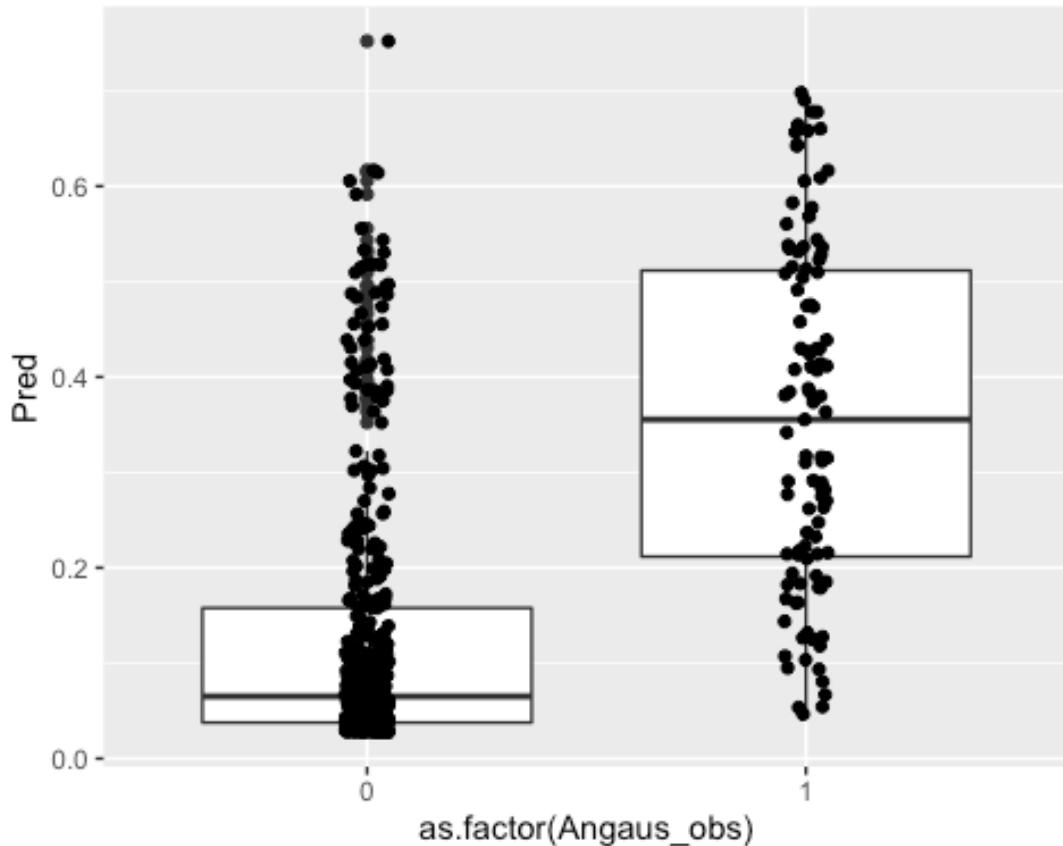
```
## Warning: Use of `object[[1L]]` is discouraged. Use `^.data[[1L]]` instead.  
## Warning: Use of `object[["yhat"]]]` is discouraged. Use `^.data[["yhat"]]]`  
## instead.  
## Warning: Use of `object[[1L]]` is discouraged. Use `^.data[[1L]]` instead.  
## Warning: Use of `object[["yhat"]]]` is discouraged. Use `^.data[["yhat"]]]`  
## instead.  
## Warning: Use of `object[[1L]]` is discouraged. Use `^.data[[1L]]` instead.  
## Warning: Use of `object[["yhat"]]]` is discouraged. Use `^.data[["yhat"]]]`  
## instead.  
## Warning: Use of `object[[1L]]` is discouraged. Use `^.data[[1L]]` instead.  
## Warning: Use of `object[["yhat"]]]` is discouraged. Use `^.data[["yhat"]]]`  
## instead.  
## Warning: Use of `object[[1L]]` is discouraged. Use `^.data[[1L]]` instead.  
## Warning: Use of `object[["yhat"]]]` is discouraged. Use `^.data[["yhat"]]]`  
## instead.  
## Warning: Use of `object[[1L]]` is discouraged. Use `^.data[[1L]]` instead.  
## Warning: Use of `object[["yhat"]]]` is discouraged. Use `^.data[["yhat"]]]`  
## instead.  
## Warning: Use of `object[[1L]]` is discouraged. Use `^.data[[1L]]` instead.  
## Warning: Use of `object[["yhat"]]]` is discouraged. Use `^.data[["yhat"]]]`  
## instead.  
## Warning: Use of `object[[1L]]` is discouraged. Use `^.data[[1L]]` instead.  
## Warning: Use of `object[["yhat"]]]` is discouraged. Use `^.data[["yhat"]]]`  
## instead.  
## Warning: Use of `object[[1L]]` is discouraged. Use `^.data[[1L]]` instead.  
## Warning: Use of `object[["yhat"]]]` is discouraged. Use `^.data[["yhat"]]]`  
## instead.
```



Let's

now take the predictors and check each of them separately. We take the test data and apply together with predictions, based on our tree.

```
leathwick_test %>%
  bind_cols(Pred=predict(leathwick.gbm,newdata=leathwick_test,
  n.tree=best.iter, type="response")) %>%
  ggplot()+
  geom_boxplot(aes(y=Pred, x=as.factor(Angaus_obs)))+
  geom_point(aes(y=Pred,
  x=as.factor(Angaus_obs)),position=position_jitter(width=0.05))
```



This

graph helps for measuring sensitivity and sensibility (ratio of false pos to false neg, or ratio of false neg to false pos AND false neg): If we were to turn these prob into 0s (not present) or 1s (present), we need a threshold. For example, if you wanted to work on where you are making errors (ie. false pos, false neg). So, if we draw a cut off somewhere, we'd end up with some incorrect ones. There is gonna be some false pos in here.

Let's create two sets: One for the 1s (pres=genuinely present), and one for the 0s (abs=genuinely absent) That allows us to calculate our false neg and false pos.

```

preds<-leathwick_test %>%
  bind_cols(Pred=predict(leathwick.gbm,newdata=leathwick_test,
                         n.tree=best.iter, type="response"))

pres<-preds %>% filter(Angaus_obs==1) %>% pull(Pred)

abs<-preds %>% filter(Angaus_obs==0) %>% pull(Pred)

e<-dismo::evaluate(p=pres,a=abs)

e

## class          : ModelEvaluation
## n presences   : 107
  
```

```
## n absences      : 393
## AUC            : 0.8629521
## cor            : 0.5276568
## max TPR+TNR at : 0.1246359
```

This is literally what the true pos and true neg rates would be for any given threshold. From this, you can calculate the area under the curve. And the area under the curve AUC is a measure of accuracy. In this case, the AUC is 0.86, which is our measure of ACCURACY.

We could also just do pure correlation (cor), which is 0.52. However, for this function, AUC is the one which gives you the accuracy.

The best threshold is 0.14 (=max TPR+TNR at). That means that that number CALIBRATES the model. Once we get a prob of 0.14, and once our response variable has been turned to 1s or 0s, any prediction above that value is a 1, and any value below that is 0. That is your accuracy. This function also take into account interactions. You could also plot two variables and get a surface.

For info, check the image AUC under: AUC

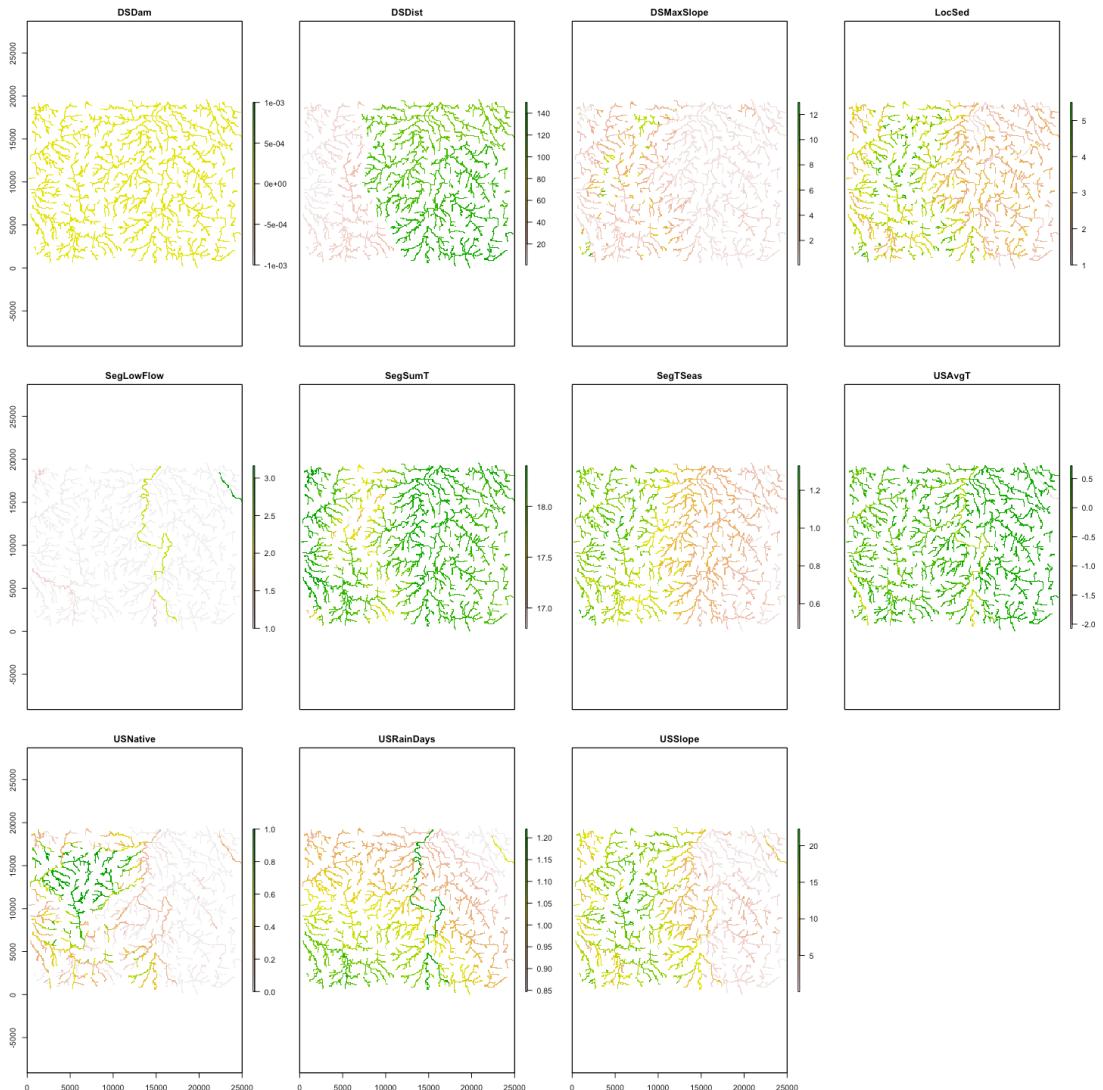
Assessing accuracy

Plot spatial distribution of eels

```
data(Anguilla_grids)
leathwick.grid = Anguilla_grids
glimpse(leathwick.grid)

## Formal class 'RasterBrick' [package "raster"] with 12 slots
##   ..@ file    :Formal class '.RasterFile' [package "raster"] with 13 slots
## Warning: Not a validObject(): no slot of name "NAchanged" for this object
##   ..@ data    :Formal class '.MultipleRasterData' [package "raster"] with
##     14 slots
##   ..@ legend  :Formal class '.RasterLegend' [package "raster"] with 5
##     slots
##   ..@ title   : chr(0)
##   ..@ extent  :Formal class 'Extent' [package "raster"] with 4 slots
##   ..@ rotated : logi FALSE
##   ..@ rotation:Formal class '.Rotation' [package "raster"] with 2 slots
##   ..@ ncols   : int 250
##   ..@ nrows   : int 196
##   ..@ crs     :Formal class 'CRS' [package "sp"] with 1 slot
##   ..@ history : list()
##   ..@ z       : list()

plot(leathwick.grid)
```



```

Method <- factor('electric', levels=levels(leathwick$Method))
Method = as.data.frame(Method)

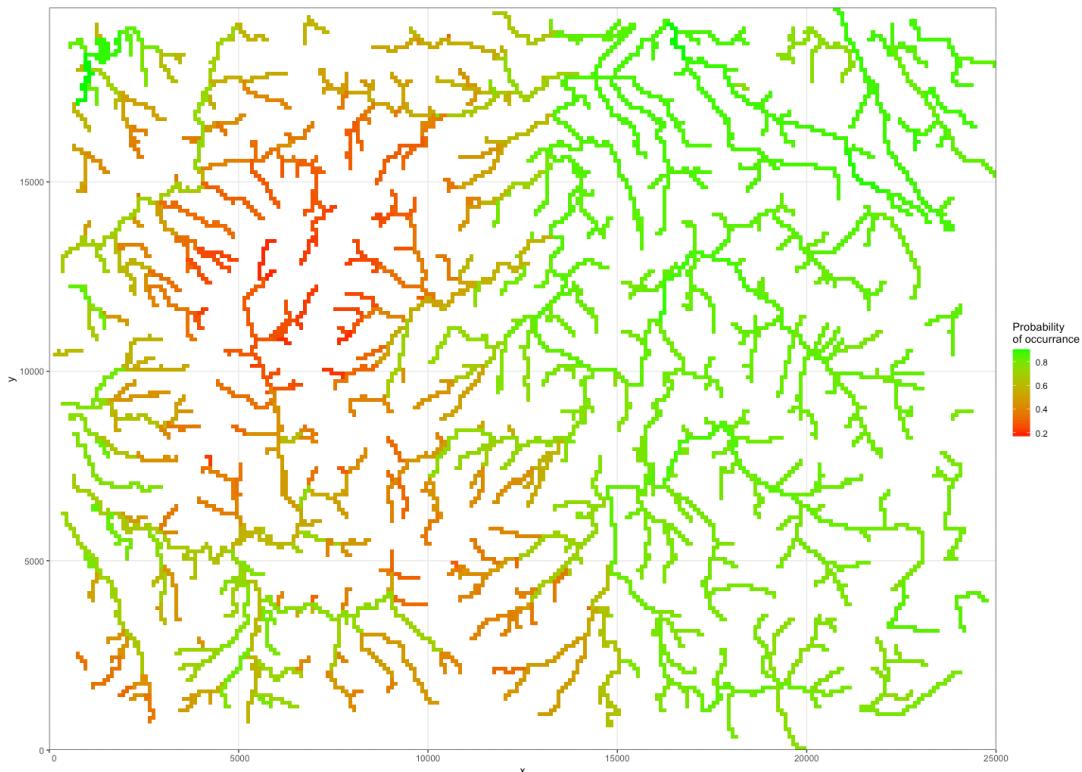
fit <- predict(leathwick.grid, leathwick.gbm, const=Method,
               n.trees=best.iter, type='response')
#fit <- mask(fit, raster(Leathwick.grid, 1))

fit= stars::st_as_stars(fit)

ggplot() +
  geom_stars(data=fit) +
  scale_fill_gradient(low='red', high='green', 'Probability\nof occurrence',
na.value=NA) +
  coord_sf(expand=FALSE) +
  theme_bw()

```

```
## Warning: Removed 40942 rows containing missing values (geom_raster).
```



References