Statistical Analysis with R using RxP Dataset

Table of contents

1 Basics

1.1 Loading Data

Since the data is copied from internet, I used a python script to process the data to make it readable using read.csv():

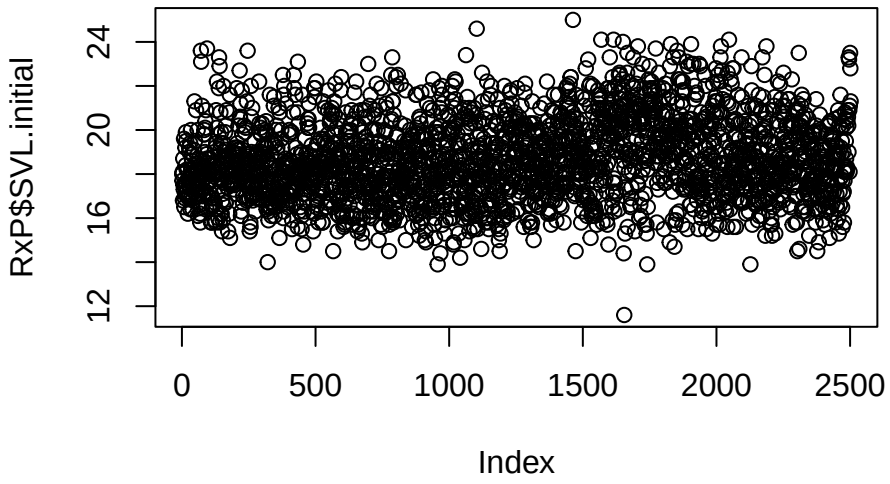
```
RxP<-read.csv("./RxP.csv", sep=",",header=TRUE, stringsAsFactors=T)
knitr::kable(head(RxP[, c(1:4,7,10,13)]), "pipe")
```

Ind	Block	Tank	Tank.Unique	Res	SVL.initial	Mass.final
1	5	7	55	Hi	18.0	0.38
2	5	4	52	Hi	17.7	0.35
3	5	4	52	Hi	18.1	0.41
4	5	7	55	Hi	16.8	0.30
5	5	10	58	Hi	18.7	0.46
6	5	4	52	Hi	17.5	0.30

1.2 Data Exploration

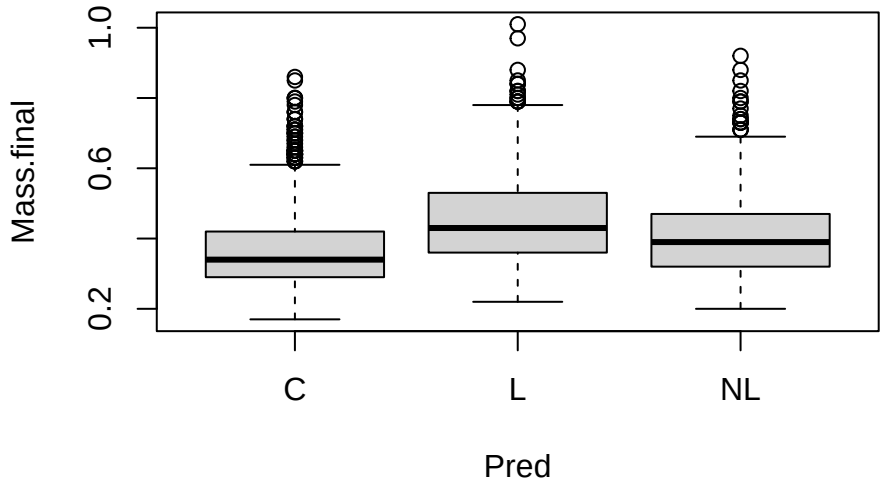
The plot of `SVL.initial` looks like the following:

```
# | echo: true
plot(RxP$SVL.initial)
```

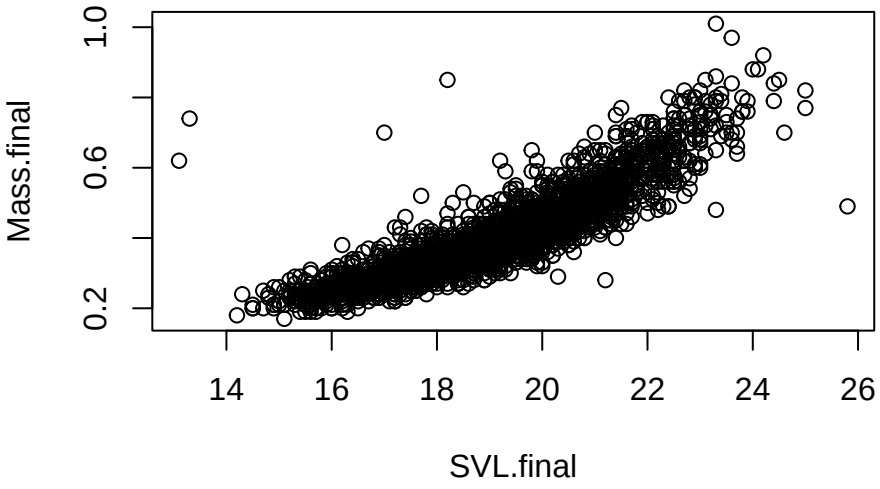


The default outcome against level data of a `plot()` function is *box plot*:

```
plot(Mass.final~Pred, data=RxP)
```

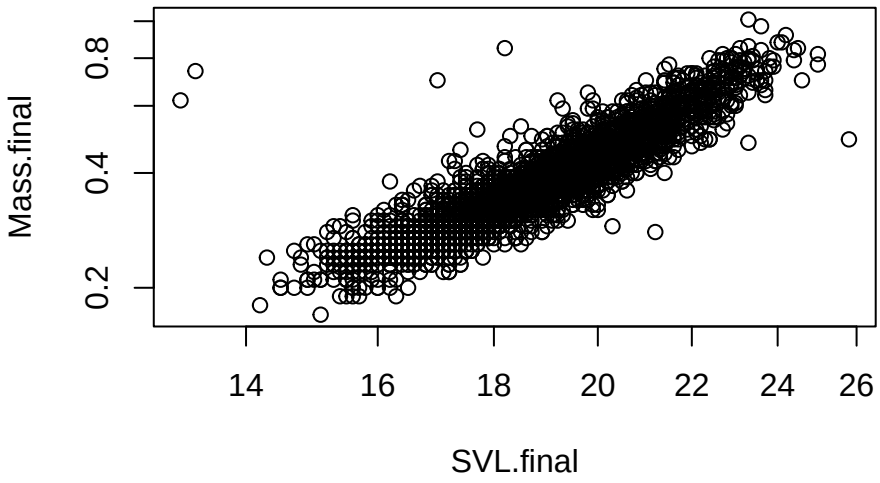


For continuous function, the default plot is a graph:



The plots are used to inspect data outliers. Besides, as can be seen from the The curviture of the graph indicates that the relationship between `Mass.final` and `SVL.final` is not linear, yet after log transformation, the graph appears to be linear.

```
plot(Mass.final~SVL.final, data=RxP, log="xy")
```

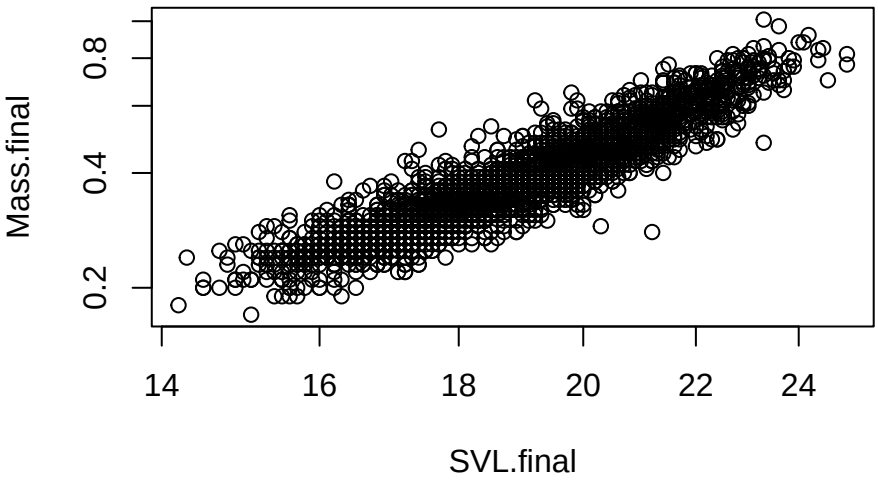


1.3 Fixing data

1.3.1 `library(dplyr): filter()`

Remove the outlier from the data using `filter()` from the `dplyr` library, the number of outliers are reduced.

```
library(dplyr)

#Remove the offending data
temp<-filter(RxP, Ind!=734 &
                  Ind!=1024 &
                  Ind!=1078 &
                  Ind!=1127 &
                  Ind!=1284)
temp <- temp[temp$SVL.initial>12,]
temp<-filter(temp, Tail.initial<=15)

plot(Mass.final~SVL.final, data=temp, log="xy")
```



1.4 Summerizing data

1.4.1 `aggregate()` from base R

- The `summarize()` function from the dplyr package.
- The `aggregate()` function from base R.

Using `aggregate` to get summary statistics across combination of different groups:

```
aggregate(SVL.initial~Pred*Res, FUN=mean, data=RxP.clean)
```

Pred	Res	SVL.initial
C	Hi	18.37231
L	Hi	19.64221
NL	Hi	19.39076
C	Lo	17.87029
L	Lo	19.14696
NL	Lo	18.41102

1.4.2 pip command %>%, `summarize()` from `dplyr` package

The pip command, %>%, is a quick and convenient way to pass the output of one line of R code to the input of another line. It is a part of the magrittr package and was designed to improve the readability of code by allowing users to avoid nesting functions. Instead of using nested functions (e.g. `res <- func1(func2(func3(x)))`, the pipe command can be used to pass the

output of one function to another (e.g. `res <- x %>% func3 %>% func2 %>% func1`). This can make code more concise and easier to read.

It's included in the `dplyr` package.

To achieve same effect as above using `summarize()` and pip command from the `dplyr` package:

```
RP.means<-RxP.clean %>% #First, identify the data frame
    group_by(Res, Pred) %>% #Next, establish grouping variables
    summarize(SVL.mean = mean(SVL.initial)) #Last, calc. means

str(RP.means)
```

```
gropd_df [6 x 3] (S3: grouped_df/tbl_df/tbl/data.frame)
 $ Res     : Factor w/ 2 levels "Hi","Lo": 1 1 1 2 2 2
 $ Pred    : Factor w/ 3 levels "C","L","NL": 1 2 3 1 2 3
 $ SVL.mean: num [1:6] 18.4 19.6 19.4 17.9 19.1 ...
 - attr(*, "groups")= tibble [2 x 2] (S3: tbl_df/tbl/data.frame)
  ..$ Res  : Factor w/ 2 levels "Hi","Lo": 1 2
  ..$ .rows: list<int> [1:2]
  .. ..$ : int [1:3] 1 2 3
  .. ..$ : int [1:3] 4 5 6
  .. ..@ ptype: int(0)
  ..- attr(*, ".drop")= logi TRUE
```

```
RP.means<-as.data.frame(RP.means)
str(RP.means)
```

```
'data.frame':   6 obs. of  3 variables:
 $ Res     : Factor w/ 2 levels "Hi","Lo": 1 1 1 2 2 2
 $ Pred    : Factor w/ 3 levels "C","L","NL": 1 2 3 1 2 3
 $ SVL.mean: num  18.4 19.6 19.4 17.9 19.1 ...
```

1.4.3 leveling data: use `factor()` or `mutate(X=factor(...,levels=...))`

Either way, it's only changes how the factor is ordered, not the actual data itself.

```
#This is one way to reorder a factor
RxP.clean$Pred<-factor(RxP.clean$Pred, levels=c('C','NL','L'))
```

```r
#This is another way to reorder a factor, using dplyr
RxP.clean<-RxP.clean %>%
    mutate(Pred = factor(Pred, levels=c('C','NL','L')))
```

1.4.4 Calculating means and stadard error based on group

```r
RP.means<-RxP.clean %>% #Define what dataset we are using
    group_by(Res, Pred) %>% #Set the groups
    summarize(SVL.mean = mean(SVL.initial), #Calculate the mean
              SVL.sd = sd(SVL.initial), #SD
              SVL.n = length(SVL.initial)) %>% #and N of SVL
    mutate(SVL.se = SVL.sd/sqrt(SVL.n)) #Calculate the SE

RP.means
```

Res	Pred	SVL.mean	SVL.sd	SVL.n	SVL.se
Hi	C	18.37231	1.804411	679	0.0692469
Hi	NL	19.39076	1.719233	249	0.1089519
Hi	L	19.64221	1.618992	398	0.0811527
Lo	C	17.87029	1.744590	626	0.0697279
Lo	NL	18.41102	1.678895	245	0.1072607
Lo	L	19.14696	1.735509	296	0.1008744

1.5 Exercises

1.5.1 %>% with `filter()`

```r
before30 <- filter(RxP.clean, Age.FromEmergence <= 30)
after30 <- RxP.clean %>% filter(Age.FromEmergence > 30)
```

1.5.2 Create new binary column based on condition: `ifelse()`

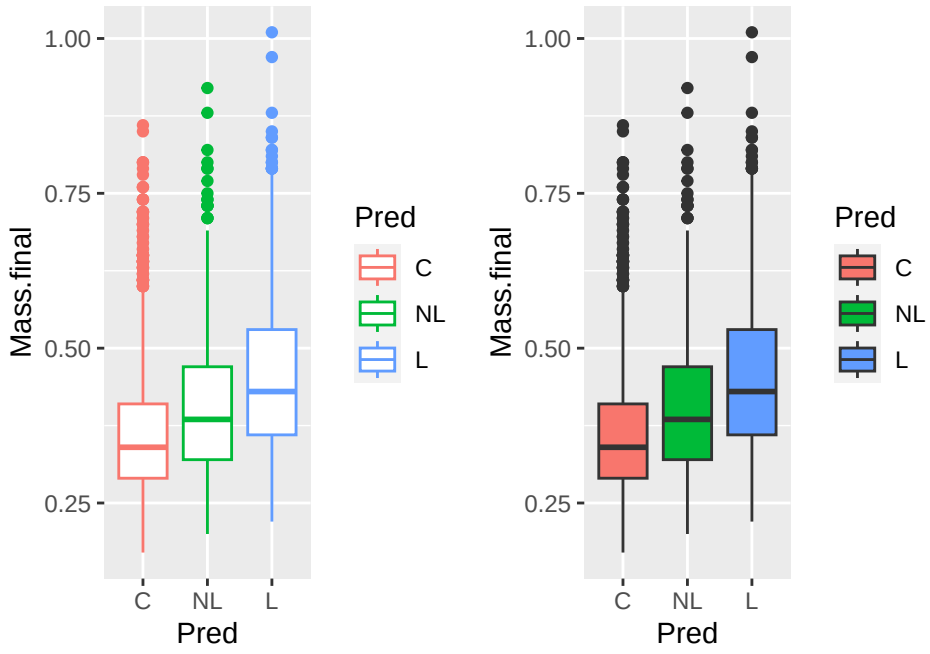
```
RxP.clean$before30 <- ifelse(RxP.clean$Age.FromEmergence<=30,
↪  "YES","NO")
```

2 Plotting

- Use `library(ggplot2)` to plot nice graphs (`qplot()` is a quick plotting function comes with ggplot2 package).
- Use `library(cowplot)` , which provides `plot_grid()` to plot multiple figures together.

2.1 Use `plot_grid()` from `library(cowplot)` to plot multipul graphs together

```
#Load the necessary package
library(ggplot2)
library(cowplot)

#Change the outline color of the boxes and whiskers
a<-qplot(data=RxP.clean,
        y=Mass.final,
        x=Pred,
        geom='boxplot',
        col=Pred)

#Change the color that fills each of the boxes
b<-qplot(data=RxP.clean,
        y=Mass.final,
        x=Pred,
        geom='boxplot',
        fill=Pred)

plot_grid(a,b, ncol=2)#specify to put the plots in two columns
```



2.2 Use `qplot()` from `library(ggplot2)`

2.2.1 Use `facets=rows~columns` arguments to split the graphs

> **Note**
>
> The facets argument only takes two variables together, if only takes one, it won't work, if only want one variables, keep the placeholder of another variable with a `.`.

- The case where `facets = var1 ~ var2`

```
#Plot the mass data by all three categorical
#predictors by using facets
qplot(data=RxP.clean,
        y=Mass.final,
        x=Pred,
        geom='boxplot',
        fill=Pred,
        facets=Hatch~Res)
```

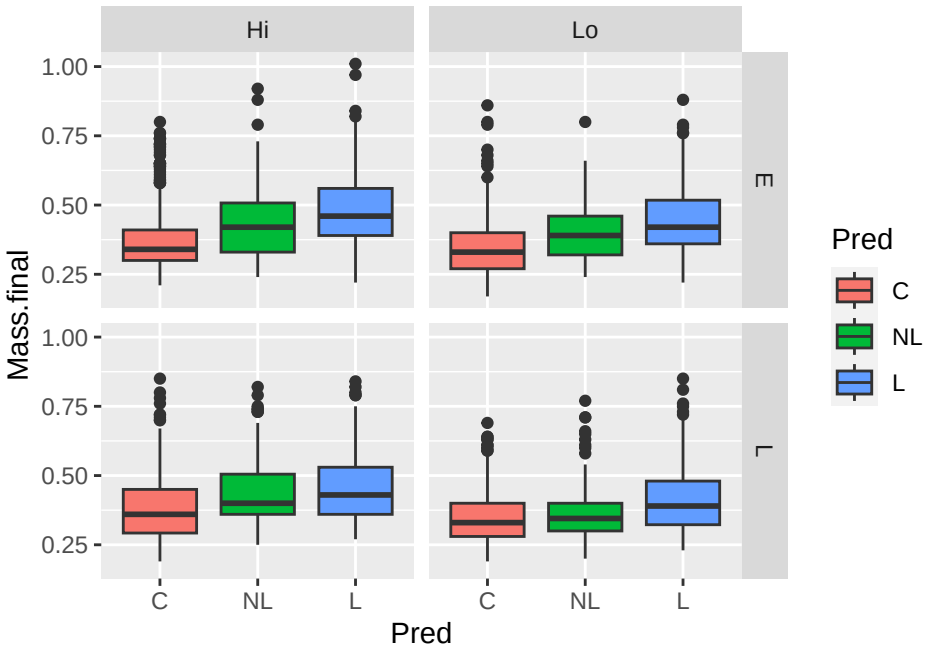
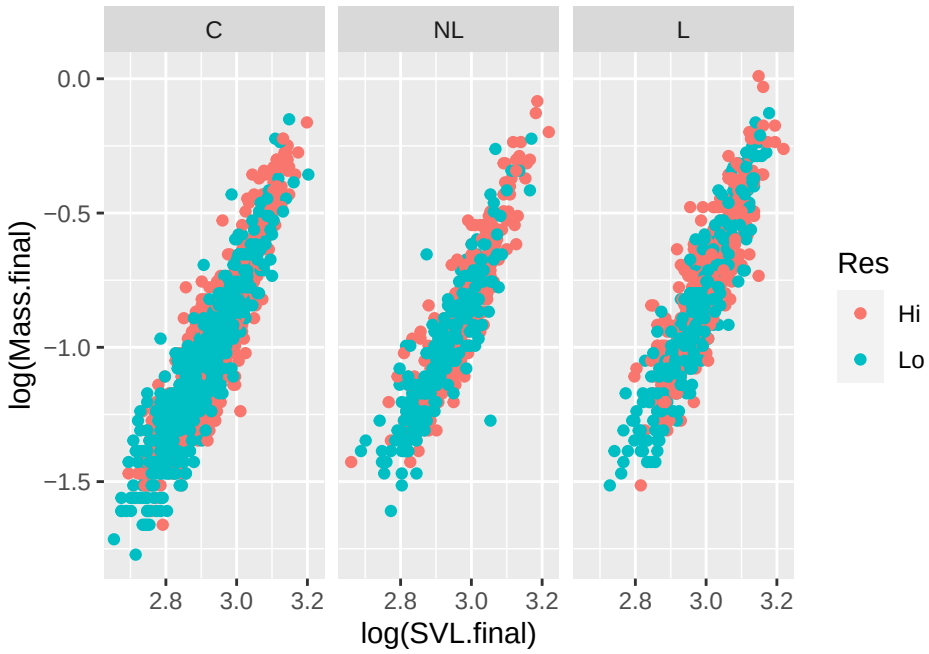


Figure 1: facets = var1 ~ var2

- The case where `facets = . ~ var1` (a continuous example)

```
qplot(data=RxP.clean,
        x=log(SVL.final),
        y=log(Mass.final),
        col=Res,
        facets=.~Pred)
```



2.2.2 Use `geom=` to determine the kind of graph to plot

- density, histogram, violin, boxplot etc.
- `hist()` is used to create histogram when using `gplots()` library

2.2.3 Use `fill=` and `col=` to determine the color of the graph

- Use `color()` to get the all the colors.

2.2.4 Use `alpha=` to determine the transparancy of the graph

2.2.5 Use `shape=` to determine the shape of the graph

2.3 Use `barplot2()` from `library(gplots)` for bar plot

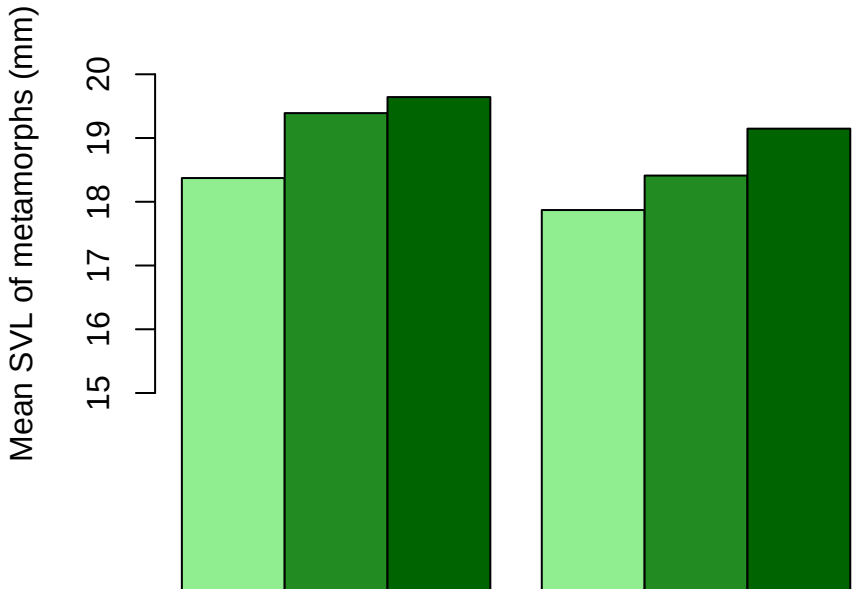
`gplots` is one plot that's used to generate bar plot. Note that bar plot is one of the most complicated basic graph to generate.

The plot is based on aggregate data `RP.means`:

Res	Pred	SVL.mean	SVL.sd	SVL.n	SVL.se
Hi	C	18.37231	1.804411	679	0.0692469
Hi	NL	19.39076	1.719233	249	0.1089519
Hi	L	19.64221	1.618992	398	0.0811527
Lo	C	17.87029	1.744590	626	0.0697279
Lo	NL	18.41102	1.678895	245	0.1072607
Lo	L	19.14696	1.735509	296	0.1008744

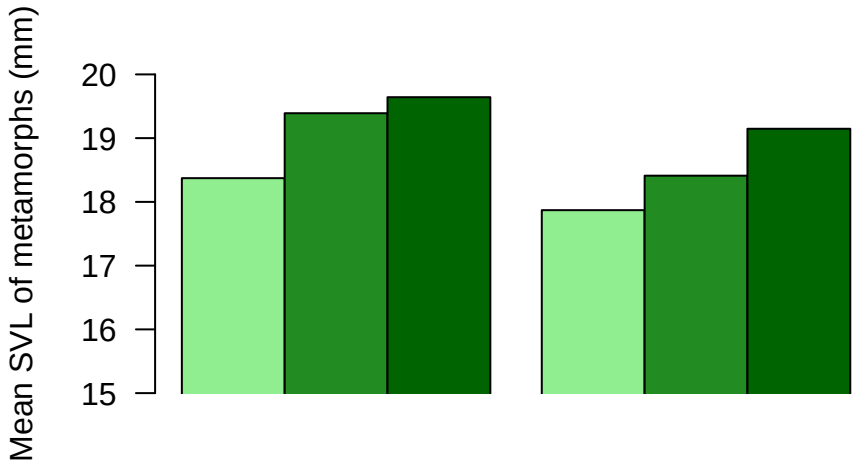
2.3.1 Use `space=c()` to add space between bars

```
library(gplots)
barplot2(RP.means$SVL.mean, ylim=c(15,20),
            ylab='Mean SVL of metamorphs (mm)',
            space=c(0,0,0,0.5,0,0),
            col=c('light green',
                    'forest green',
                    'dark green'))
```



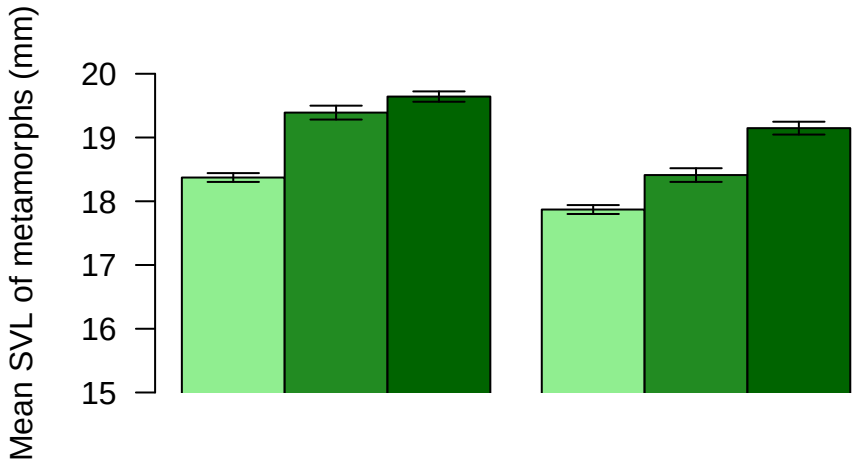
2.3.2 Use `xpd=F` to trim the plot, and `las=1` to turn the numbers

```
barplot2(RP.means$SVL.mean, ylim=c(15,20),
            ylab='Mean SVL of metamorphs (mm)',
            space=c(0,0,0,0.5,0,0),
            col=c('light green',
                    'forest green',
                    'dark green'),
            xpd=F,
            las=1) #turn the y-axis numbers 90 degrees)
```



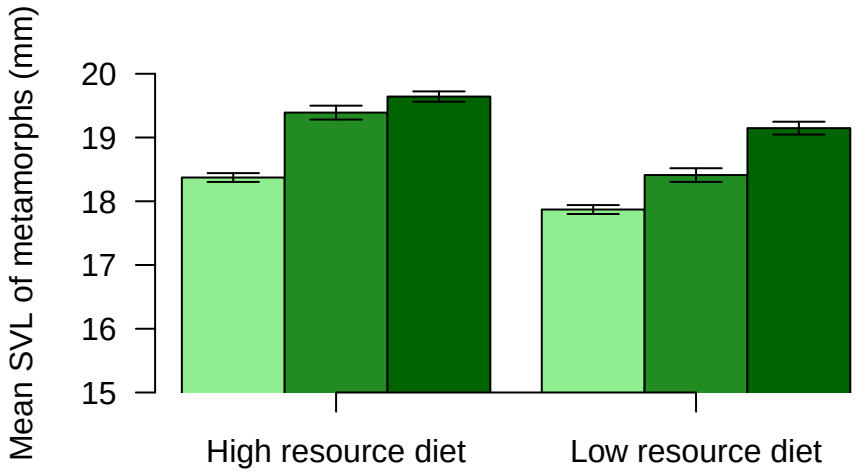
2.3.3 Use `plot.ci=T`, `ci.u` and `ci.l` to plot confidence interval

```
barplot2(RP.means$SVL.mean, ylim=c(15,20),
        ylab="Mean SVL of metamorphs (mm)",
        space=c(0,0,0,0.5,0,0),
        col=c("light green",
                "forest green",
                "dark green"),
        xpd=F,
        las=1,
        plot.ci=T, #tell barplot2 to plot error bars
        ci.u=RP.means$SVL.mean+RP.means$SVL.se, #the upper limit
        ci.l=RP.means$SVL.mean-RP.means$SVL.se) #the lower limit
```



2.3.4 Use `axis()` to add additional information along the axises

```
barplot2(RP.means$SVL.mean, ylim=c(15,20),
        ylab="Mean SVL of metamorphs (mm)",
        space=c(0,0,0,0.5,0,0),
        col=c("light green",
                "forest green",
                "dark green"),
        xpd=F,
        las=1,
        plot.ci=T, #tell barplot2 to plot error bars
        ci.u=RP.means$SVL.mean+RP.means$SVL.se, #the upper limit
        ci.l=RP.means$SVL.mean-RP.means$SVL.se) #the lower limit

axis(side=1, #set the side
        at=c(1.5, 5), # set the location for x-axis labels
        labels=c("High resource diet", "Low resource diet")) #set labels
```

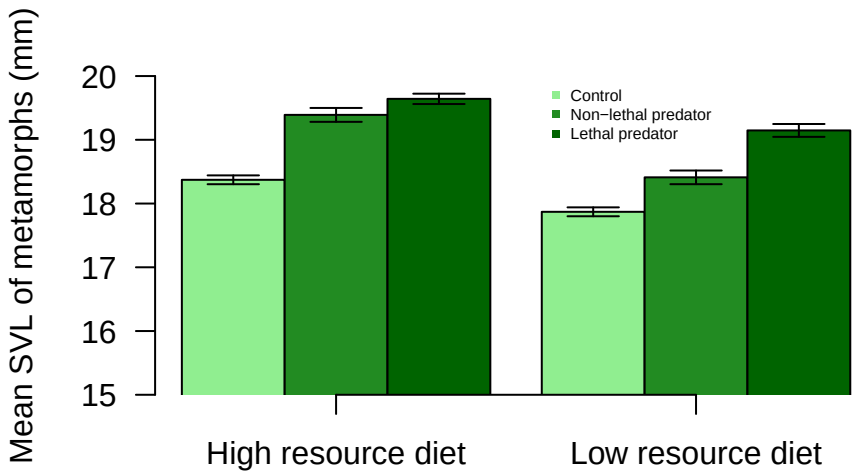


2.3.5 Use `legend()` to define the legend

- The placement of `legend()` function can also be words such as `topright` or `bottomleft`.

```
barplot2(RP.means$SVL.mean, ylim=c(15,20),
        ylab="Mean SVL of metamorphs (mm)",
        space=c(0,0,0,0.5,0,0),
        col=c("light green",
                "forest green",
                "dark green"),
        xpd=F,
        las=1,
        plot.ci=T, #tell barplot2 to plot error bars
        ci.u=RP.means$SVL.mean+RP.means$SVL.se, #the upper limit
        ci.l=RP.means$SVL.mean-RP.means$SVL.se) #the lower limit

axis(side=1, #set the side
        at=c(1.5, 5), # set the location for x-axis labels
        labels=c("High resource diet", "Low resource diet")) #set labels

legend(x=3.5, y=20, #provide x and y coordinates for legend
        legend=c("Control",
                "Non-lethal predator",
                "Lethal predator"), #legend text
        col=c("light green",
                "forest green",
                "dark green"), #legend colors
```

```
        bty="n", #Do you want a box around the legend? Nope.
        pch=15, #the plot character (the square) to put next to the text
        cex=0.5) #the scaling of the legend, relative to 1
```



2.4 Use `ggplot()` from `library(gplots)` for bar plot

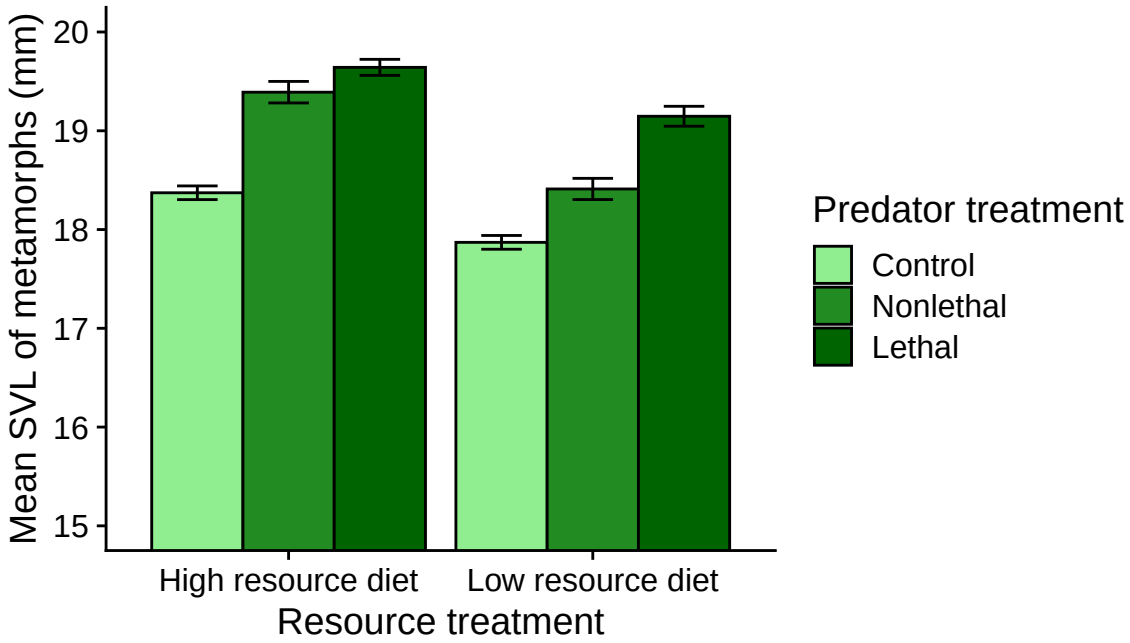
The `ggplot()` function is used to make a bar plot, which is more visually appealing compared to `barplot2()`. To plot with `ggplot2`, you must define the data source (`data=rp.means`) and aesthetics (`aes()`) within the `ggplot()` function, and add a `geom` function. It could be strange at first, but it works well once you get used to it.

- Use `geom_col()` to specify position of the bar and specifying boarder color
  - `position = dodge` to place to bar plot side by side, the default is stacking them on top of each other
  - `col = black` set the border of the bars to black
- Use `scale_fill_manual()` to fill the column with customized color, also add legend.
  - The `value=` parameter specifies the color
  - The `label=` parameter specifies the legend
- Use `geom_errorbar()` to add confidence interval
  - The `position=` determines whether the bar will be stack together or aligned side by side; the number determines how far apart are these bars.
- Use `labs()` to customize the y-axis
- Use `facet_grid(facets = )` to create facets
- Use `facet_wrap(facets= , scale =, nrow=)`

  - if specifying `scale='free'`, the scales can vary between different panel, which might be crucial in many cases.

- Use `scale_x_discrete()` to customize the x-axis
  - The `name=` adds name to the axis
  - The `label=` adds customized label to x-axis
- Use `theme_cowplot()` : This theme makes larger than default fonts for the axis labels

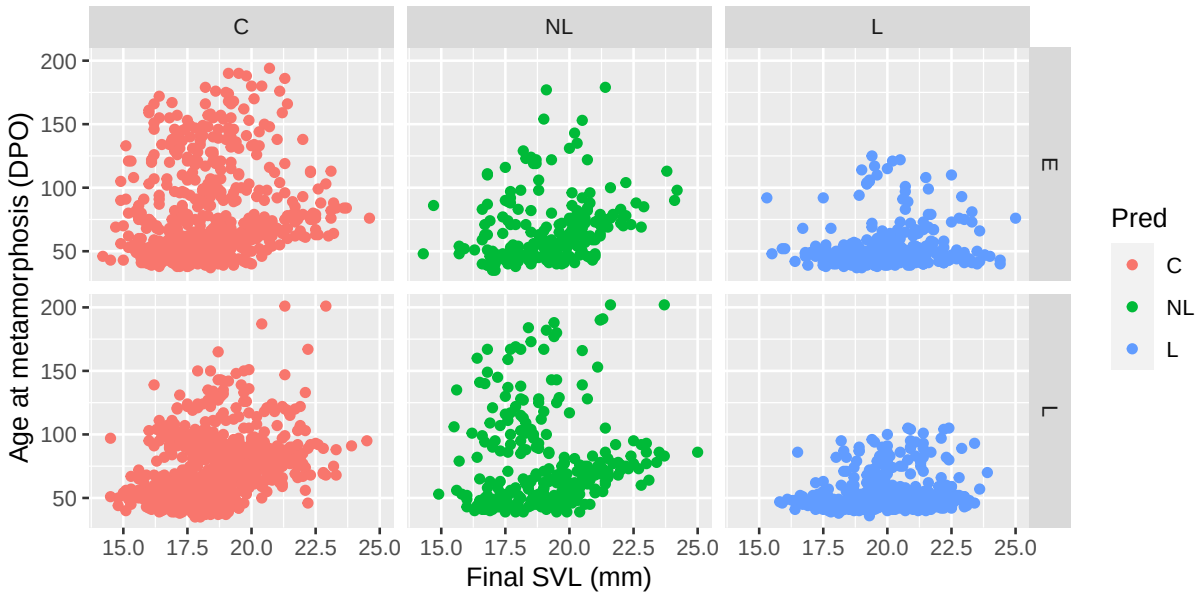
```
ggplot(data=RP.means, aes(x=Res, y=SVL.mean, fill=Pred))+
       geom_col(position="dodge", col="black")+
       coord_cartesian(ylim=c(15,20))+
       scale_fill_manual(values = c("light green","forest green","dark
       ↪  green"),
                         name="Predator treatment",
                         labels=c("Control","Nonlethal","Lethal"))+ #
                         ↪  Custom legend!
        geom_errorbar(aes(ymin=SVL.mean-SVL.se, ymax=SVL.mean+SVL.se),
                      position=position_dodge(0.9), width=0.4)+
        labs(y="Mean SVL of metamorphs (mm)")+ # Custom y-axis label
        scale_x_discrete(name="Resource treatment",
                         labels=c("High resource diet","Low resource
                         ↪  diet"))+ # Custom x-axis
        theme_cowplot() # Make it look great with cowplot!
```



2.5 Exercises

2.5.1 Add `xlab` and `ylab` to `qplot()`

```
qplot(data = RxP.clean,
      y = Age.DPO,
      x= SVL.final,
      col = Pred,
      facets = Hatch~Pred,
      xlab = "Final SVL (mm)",
      ylab = "Age at metamorphosis (DPO)")
```

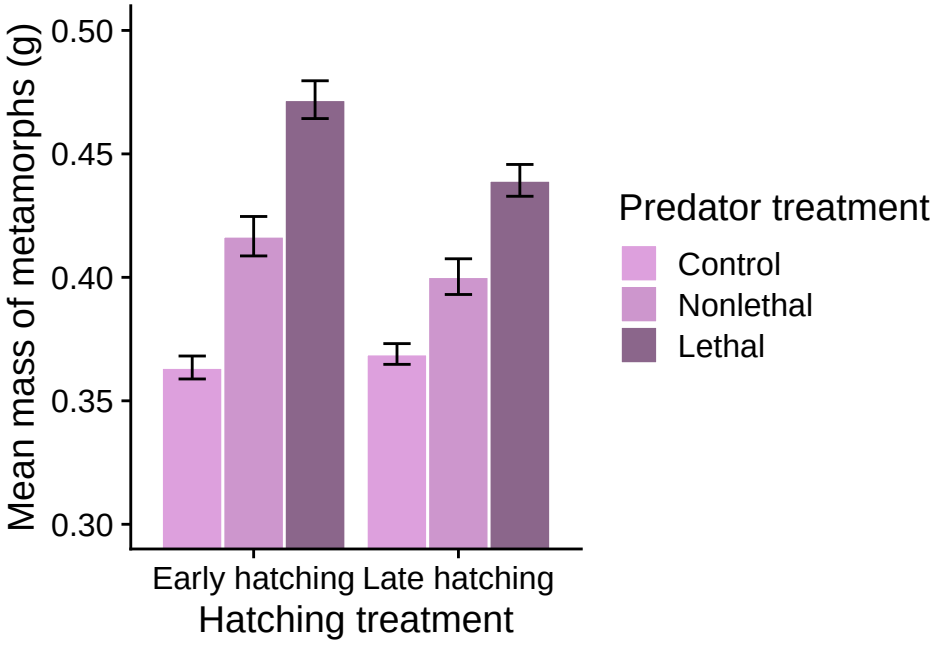


2.5.2 Bar plot grouped by `Pred` and `Hatch` using `ggplot()`

```
RP.means2 <- RxP.clean %>%
  group_by(Hatch,Pred) %>%
  summarize(Mass.mean = mean(Mass.final),
            Mass.sd = sd(Mass.final),
            Mass.n = length(Mass.final)) %>%
  mutate(Mass.se = Mass.sd/sqrt(Mass.n))
```

```
`summarise()` has grouped output by 'Hatch'. You can override using the
`.groups` argument.
```

```
ggplot(data = RP.means2, aes(x = Hatch,
                             y = Mass.mean,
                             fill = Pred))+
  geom_col(position="dodge", col="white")+
  coord_cartesian(ylim = c(0.3,0.5))+
  scale_fill_manual(values = c("plum","plum3","plum4"),
                    name = "Predator treatment",
                    label = c("Control", "Nonlethal", "Lethal"))+
  geom_errorbar(aes(ymin=Mass.mean-Mass.se, ymax = Mass.mean + Mass.se),
                width = 0.4, position = position_dodge(0.9))+
  labs(y = "Mean mass of metamorphs (g)")+
  scale_x_discrete(name = "Hatching treatment",
                   labels = c("Early hatching", "Late hatching"))+
  theme_cowplot()
```



3 Basics of Statistical Analysis

3.1 Grouping data to avoid pseudoreplication, `group_by()`

When observations are closely related to each other (in this particular setting, when metamorphs are raised from the same environment), by treating each individuals as independent, we risk committing to pseudo-replication. In order to eliminate the risk, one way is to take the average of the individuals by group.

```
RxP.byTank<-RxP.clean %>%
            group_by(Block,Tank.Unique,Pred,Hatch,Res) %>%
            summarize(Age.DPO = mean(Age.DPO),
                      Age.FromEmergence = mean(Age.FromEmergence),
                      SVL.initial = mean(SVL.initial),
                      Tail.initial = mean(Tail.initial),
                      SVL.final = mean(SVL.final),
                      Mass.final = mean(Mass.final),
                      Resorb.days = mean(Resorb.days))

RxP.byTank<-as.data.frame(RxP.byTank)
knitr::kable(head(RxP.byTank[,c(1:5,7,10)]),digits = 2)
```

Block	Tank.Unique	Pred	Hatch	Res	Age.FromEmergence	SVL.final
1	1	NL	L	Hi	13.19	19.66
1	2	C	E	Hi	11.38	19.01
1	3	C	L	Hi	19.82	19.12
1	4	L	L	Lo	22.92	19.12
1	5	NL	E	Hi	30.75	20.11
1	6	L	E	Hi	10.00	21.98

3.2 Use `shapiro.test()` for normality test

The Shapiro Test is a normality test used to compare a dataset to a normal distribution with a similar mean and variance and assign a p-value of the probability that the data is normal. It is executed in R using the `shapiro.test()` function. This function does not accept the `data=` argument, so the vector of data needs to be specified. A small p-value does not necessarily mean the data isn't normal, but rather that it is unlikely that the data comes from a larger pool of normally distributed data.

```
shapiro.test(log(RxP.byTank$SVL.final))
```

```
    Shapiro-Wilk normality test

data:  log(RxP.byTank$SVL.final)
W = 0.95421, p-value = 0.006943
```

p<0.05, despite log transformation, reject the null that the variable is normally distributed.

3.3 Use `fitdistr(),AIC()` to test variables against normal distribution

- `fitdistr()` and `AIC()` are from `library(MASS)`, which stands for "Mordern Applied Statistics with S". It's a library for statistical modeling and hypothesis testing. The package includes some other well-known functions for statistical analysis:
  - `lm()` for fitting linear model, `predic.lm()` for making predictions, `anova.lm()` for performing hypothesis tests;
  - `glm()` for fitting logistic model, and `predic.glm()` for doing prediction;
  - `kmeans()` for k-means clustering and `hclust()` for hierarchical clustering;
  - `lda()` , `qda()` for discriminant analysis and quadratic discriminant analysis, which are functions for classification.
- `fitdistr()` uses maximum likelihood to assess the fit of the data against a predefined error distribution, `AIC()` stands for Akaike information criterion score. A lower AIC score is preferred, but mind that scores are only comparable when it's computed based on the EXACT same response variable.

```
library(MASS)
fit1<-fitdistr(RxP.byTank$SVL.final, "normal")
fit2<-fitdistr(RxP.byTank$SVL.final, "lognormal")
print(AIC(fit1,fit2))
```

```
     df      AIC
fit1  2 280.7438
fit2  2 276.4907
```

```
fit1<-fitdistr(round(RxP.byTank$Age.FromEmergence), "normal")
fit2<-fitdistr(round(RxP.byTank$Age.FromEmergence), "lognormal")
```

```
AIC(fit1,fit2)
```

	df	AIC
fit1	2	718.7014
fit2	2	665.9925

The lower score for lognormal distribution indicates the lognorm of the variables better describes the distribution of the variables. So it's better transform them before preceed to next step of analysis.

```
RxP.byTank$log.SVL.final<-log(RxP.byTank$SVL.final)
RxP.byTank$log.Age.FromEmergence<-log(RxP.byTank$Age.FromEmergence)
RxP.byTank$log.Age.DPO<-log(RxP.byTank$Age.DPO)
```

3.4 Use `wilcox.test()`, `kruskal.test()` for nonparametric test

Nonparametric data is a type of data that does not assume any particular distribution and can be used to analyze any kind of data. Nonparametric statistics do not analyze the actual values of the data but rather the ranked order of the data which are assigned ranks. Nonparametric tests are known to have low power to distinguish between two groups that are not different significantly, however they are considered to be conservative and less likely to give a false-positive result.

3.4.1 Use `wilcox.test()` for Mann-Whitney U test

The Mann–Whitney U test is a nonparametric statistical hypothesis test for assessing whether two independent samples were drawn from the same population. It can be used to test whether the medians of two groups are different. It is also known as Wilcoxon–Mann–Whitney test, Wilcoxon rank-sum test, or Mann–Whitney–Wilcoxon (MWW) test.

The Mann–Whitney U test is an alternative to the t-test when the data is not normally distributed or if the samples are of different sizes. It is also used when the variances of the two samples are not equal.

The test statistic is the U statistic, which is the sum of ranks in one sample minus the sum of ranks in the other sample, divided by the total number of samples. The null hypothesis is that the distributions of the two samples are equal. If the U statistic is greater than a critical value, the null hypothesis is rejected and it is concluded that the two samples are not equal.

```r
res<-wilcox.test(SVL.final~Res, data=RxP.byTank)
res
```

```
    Wilcoxon rank sum exact test

data:  SVL.final by Res
W = 964, p-value = 0.04204
alternative hypothesis: true location shift is not equal to 0
```

The p-value $= 0.042 < 0.05$, indicating significant differences.

> **Note**
>
> If you want to do actual `Wilcoxon signed-rank test`, includes argument `paired=T`

3.4.2 Use `kruskal.test()` for Kruskal-Wallis test

If the test for differences against treatment has more than two levels, run the Kruskal-Wallis test. It's like a non-parametric version of One-way ANOVA test.

```r
kruskal.test(SVL.final~Pred, data=RxP.byTank)
```

```
    Kruskal-Wallis rank sum test

data:  SVL.final by Pred
Kruskal-Wallis chi-squared = 32.505, df = 2, p-value = 8.744e-08
```

3.5 Use `t.test()` for parametric Studemt's test

```r
t.test(log.Age.FromEmergence~Res, data=RxP.byTank, var.equal=T)
```

```
    Two Sample t-test

data:  log.Age.FromEmergence by Res
t = -4.6246, df = 76, p-value = 1.511e-05
alternative hypothesis: true difference in means between group Hi and group Lo is not equal t
```

```
95 percent confidence interval:
 -0.9493877 -0.3778055
sample estimates:
mean in group Hi mean in group Lo
        2.835664         3.499261
```

- Use `data=` to specify the data frame
- `var.equal = T` since the defualt assumes unequal variance
- Be cautious to use t-test because when $n \longrightarrow \infty$, t-statistics becomes numerically identical to normal distribution.

4 One-way Analysis of Variance

`lm()` is one of the most flexible models in all statistics, and you can extend it to do generalized linear model too. After fitting the model, use `summary()` to generate the essential information from model, which includes adjusted R-sqaured, treatment means, F-statistics and p-value of the model. Bisdes, using `plot()` to get the diagnostic plots of the residuals of the model.

4.1 Use `Anova()` from `library(car)` for ANOVA summary

Another way to generate the summary statistics given that the explanatory variable is a discrete variable is by using `anova()` or `Anova()` from the `library(car)`.

`anova()` provides breif summary of the overall models but lacks the individual levels within factors. The statistical significance of each predictor is calculated in a stepwise manner, adding each facto one by one.

`Anova()` generate statistical significance assuming all other factors are in the model, hence providing correct and more conservative results.

```
lm1<-lm(log.Age.FromEmergence~Pred, data=RxP.byTank)
summary(lm1)
```

```
Call:
lm(formula = log.Age.FromEmergence ~ Pred, data = RxP.byTank)

Residuals:
     Min       1Q   Median       3Q      Max
-1.85101 -0.34088 -0.07498  0.37065  1.40721
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.5325     0.1107  31.920  < 2e-16 ***
PredNL       -0.2677     0.2006  -1.335    0.186
PredL        -0.7725     0.1565  -4.936 4.68e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.626 on 75 degrees of freedom
Multiple R-squared:  0.2483,    Adjusted R-squared:  0.2283
F-statistic: 12.39 on 2 and 75 DF,  p-value: 2.244e-05
```

```
library(car)
Anova(lm1)
```

	Sum Sq	Df	F value	Pr(>F)
Pred	9.710023	2	12.38871	2.24e-05
Residuals	29.391750	75	NA	NA

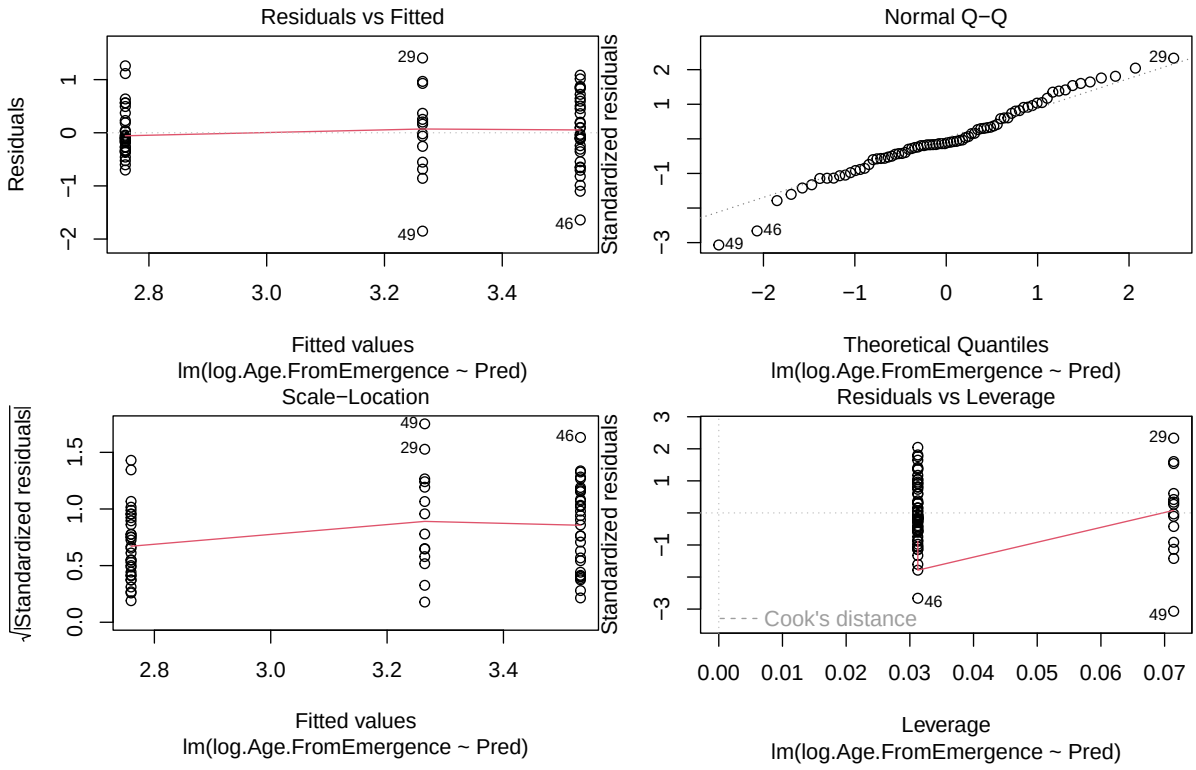
Note

It's better to ignore the test statistics produced by `summary()` function because the test statistics is obtained through *pairwise t-test* that doesn't adjust for other tests. In this case, `C` is compared with `NL` and `L` respectively, and the comparison between `NL` and `L` is ignored. These two facts almost render the comparison useless.

4.2 Use `plot()` for diagnostic plots

1. Residuals were plotted against the fitted values
2. Q-Q plot: plots standardized residuals against the theoretical quantiles from the a normal distribution (hence a close 1:1 line will be a good indication). 3.the standardized residuals plot against the fitted values to check consistency
3. cook's distance: measure the influence of each observation on the parameter estimates

```
plot(lm1)
```



> **Note**
>
> R will highlight 3 points with most extreme residual values. The number corresponds to the row number in the data frame.

4.3 Posthoc comparisons: adjusting for multilevel categorical variables

Post-hoc comparisons are statistical tests used to compare the means of two or more groups after a experiment has been conducted. It can be used to test for differences between means of the same group at different points in time or between the means of two or more groups. Post-hoc tests are typically used after a main effect has been found through an omnibus test such as ANOVA or a t-test.

4.3.1 Use `glht()` from `library(multcomp)`

The function helps compare the different levels within a single categorical variable. In this particular example, the test we are conducting is Tukey's honestly significant difference (HSD) test, which compares all pairs of groups within a categorical variable.

There's another test called Dunnett's test which compares the experimental treatment groups against *just* the control.

One important feature of these test is that they adjust the p-value for the fact that we are conducting multiple comparisons in our hypothesis testing. If we are not adjusting for the multiple comparisons, more statistical comparisons increases the possibility of finding a "significant" p-value below 0.05, even when the null hypothesis may be true.

To conduct tukey's test, one way is to use `dlht()` from `library(multcomp)` :

```
library(multcomp)
#Run a post-hoc test using glht()
ph1<-glht(lm1, linfct=mcp(Pred="Tukey"))
summary(ph1)
```

```
     Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = log.Age.FromEmergence ~ Pred, data = RxP.byTank)

Linear Hypotheses:
            Estimate Std. Error t value Pr(>|t|)
NL - C == 0  -0.2677     0.2006  -1.335   0.3777
L - C == 0   -0.7725     0.1565  -4.936   <0.001 ***
L - NL == 0  -0.5048     0.2006  -2.516   0.0363 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

4.3.2 Use `emmeans()`, `pairs()` or `cld()` from `library(emmeans)`

The `emmeans()` from `library(emmeans)` calculate the estimated marginal means from the model and pairwise differences between them. The marginal means are estimated means for the treatment groups adjusted for other potential covariates or random effects in the model. Hence, if the model is more complicated than having only one covariates, we can still use `emmeans` to obtain what we want.

Both `pairs()` and `cld()` can be used to obtain the test results based on tukey's test. The `cld()` result `.group` means, the treatments sharing the same number don't have statistical significance in their treatment differences.

```r
library(emmeans)
#Run a post-hoc test using emmeans()
ph1<-emmeans(lm1, specs="Pred")
summary(ph1)
```

Pred	emmean	SE	df	lower.CL	upper.CL
C	3.532453	0.1106642	75	3.311998	3.752907
NL	3.264705	0.1673085	75	2.931409	3.598000
L	2.759929	0.1106642	75	2.539475	2.980384

```r
#Use the pairs function to compute the Tukey test
pairs(ph1)
```

```
 contrast estimate    SE df t.ratio p.value
 C - NL      0.268 0.201 75   1.335  0.3806
 C - L       0.773 0.157 75   4.936  <.0001
 NL - L      0.505 0.201 75   2.516  0.0368

P value adjustment: tukey method for comparing a family of 3 estimates
```

```r
#You can also use the cld function with emmeans
cld(ph1)
```

	Pred	emmean	SE	df	lower.CL	upper.CL	.group
3	L	2.759929	0.1106642	75	2.539475	2.980384	1
2	NL	3.264705	0.1673085	75	2.931409	3.598000	2
1	C	3.532453	0.1106642	75	3.311998	3.752907	2

4.4 Exercises

4.4.1 Conduct a complete ANOVA analysis on `Mass.final`

```
# Check normality of mass.final

res <- shapiro.test(RxP.byTank$Mass.final)
res
```

```
    Shapiro-Wilk normality test

data:  RxP.byTank$Mass.final
W = 0.84713, p-value = 1.717e-07
```

```
logres <- shapiro.test(log(RxP.byTank$Mass.final))
logres
```

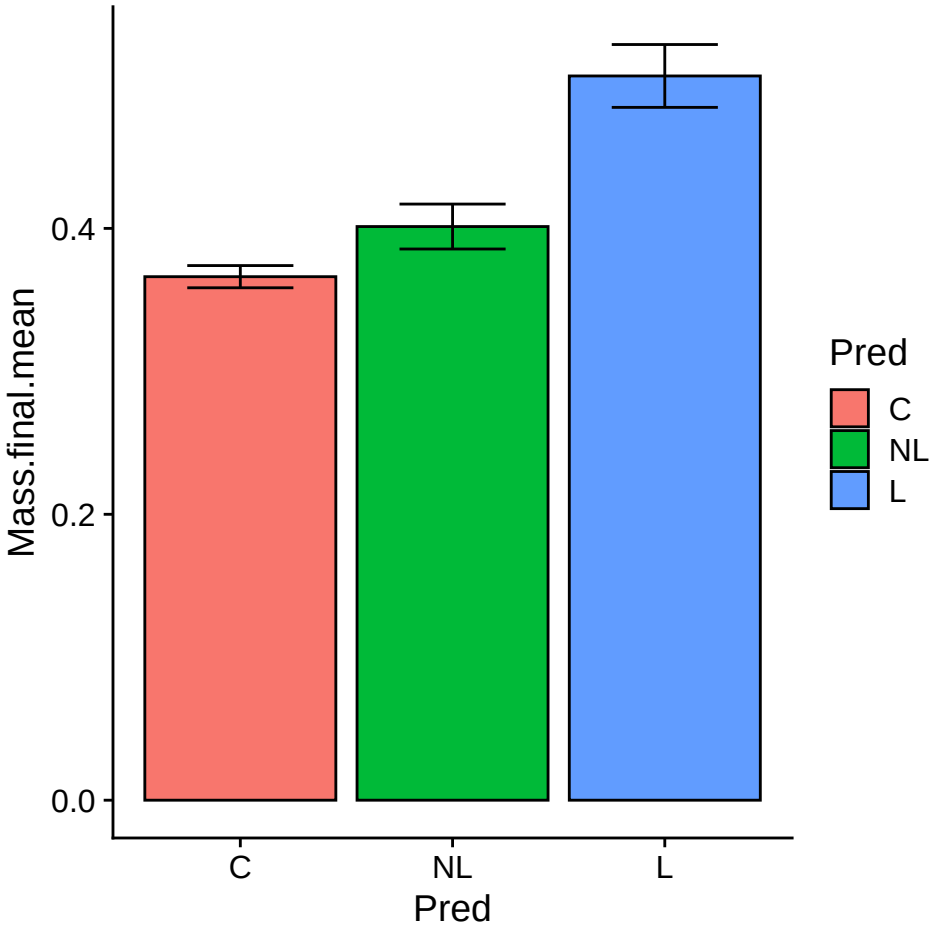
```
    Shapiro-Wilk normality test

data:  log(RxP.byTank$Mass.final)
W = 0.92494, p-value = 0.0001989
```

Despite slight improvement, neither `mass.final` nor its log transformation results in a normal distribution. However, it's still desirable to work on the transformed result.

A visal inspection using `ggplot` yet suggests significant differences across different treatment group.

```
RxP.byTank %>%
  group_by(Pred)%>%
  summarize(Mass.final.mean = mean(Mass.final),
            Mass.final.sd = sd(Mass.final),
            Mass.final.N = length(Mass.final))%>%
  mutate(Mass.final.se = Mass.final.sd/sqrt(Mass.final.N))%>%
  ggplot(data=., aes(x=Pred, y=Mass.final.mean, fill=Pred))+
  geom_col(col="black")+
  geom_errorbar(aes(ymin=Mass.final.mean-Mass.final.se,
                    ymax=Mass.final.mean+Mass.final.se),
```

```
                width=0.5)+
  theme_cowplot()
```

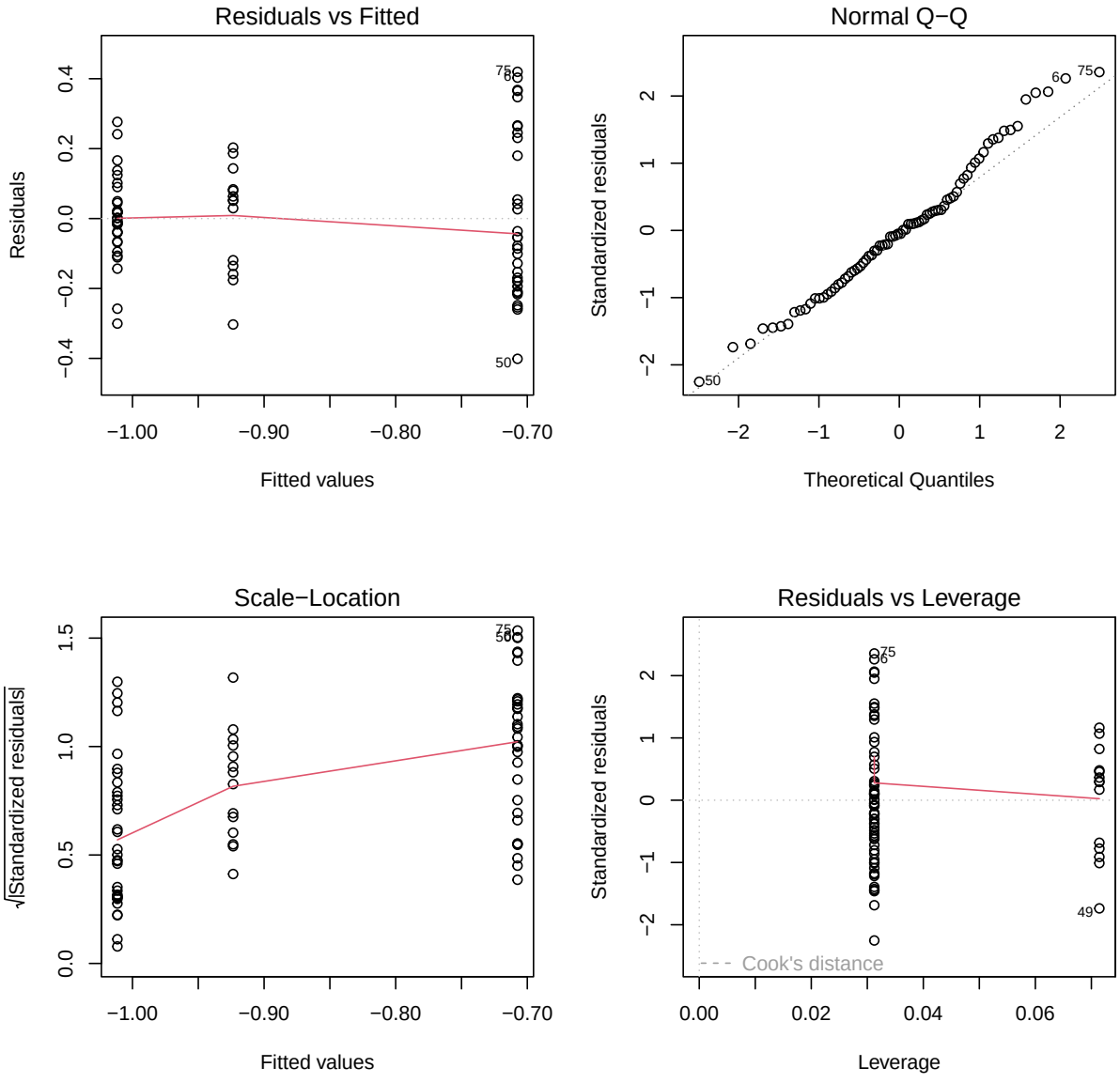


In the diagnostic results, nothing out of ordinary is detected, even though it's not a vary good fit at the end of the quantile. The adjusted results from the linear model indicates that `L` and `C`, `L` and `NL` are significantly different from each other.

```
library(MASS)
library(car)

RxP.byTank$log.mass.final <- log(RxP.byTank$Mass.final)
fit <- lm(log.mass.final~Pred, data = RxP.byTank)
Anova(fit)
```

	Sum Sq	Df	F value	Pr(>F)
Pred	1.525938	2	23.28965	0
Residuals	2.456999	75	NA	NA

```
par(mfrow=c(2,2)) # This will put the 4 plots in a 2x2 grid
plot(fit)
```



```
library(multcomp)
ph1 <- glht(fit, linfct = mcp(Pred="Tukey"))
summary(ph1)
```

```
     Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = log.mass.final ~ Pred, data = RxP.byTank)

Linear Hypotheses:
            Estimate Std. Error t value Pr(>|t|)
NL - C == 0  0.08794    0.05800   1.516  0.28656
L - C == 0   0.30401    0.04525   6.719  < 0.001 ***
L - NL == 0  0.21607    0.05800   3.726  0.00116 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

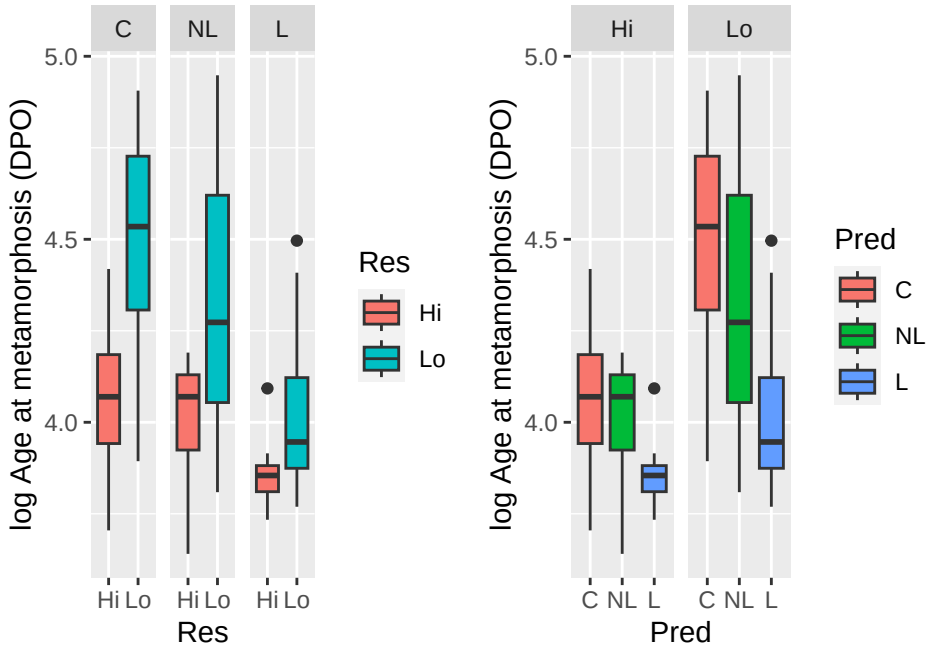
5 Multi-way Analysis of Variance

Suppose we want to investigate the effects of predator on age at metamorphosis, mass at metamorphosis or other responsive variables, but the effects might differ under different resource condition. The linear model requires an interactive term. In R, the interaction is represented as `X:Y`, but there's a short handed version to express the full model: `X*Y` is the same as writing `X+Y+X:Y`.

Plotting for the first step investigation:

```
a<- qplot(
  data = RxP.byTank,
  y = log.Age.DPO,
  ylab = "log Age at metamorphosis (DPO)",
  x = Res,
  fill = Res,
  geom = "boxplot",
  facets = .~Pred)
```

```
b <- qplot(
  data = RxP.byTank,
  y = log.Age.DPO,
  ylab = "log Age at metamorphosis (DPO)",
  x = Pred,
  fill = Pred,
  geom = "boxplot",
  facets = .~Res)

plot_grid(a,b,ncol=2)
```



The plot seems indicating interaction between Resources and the type of predators. In addition to plot, you can also use `emmeans()` to check if the effects of predator treatment differs across the resources condition, or vice versa.

5.1 Use `by=` in `emmeans()` to get primilimary results on interactions

```
lm2<-lm(log.Age.DPO~Res*Pred, data=RxP.byTank)

ph2<-emmeans(lm2, specs="Pred", by="Res")
```

```
print(pairs(ph2))
```

```
Res = Hi:
 contrast estimate     SE df t.ratio p.value
 C - NL     0.0531 0.1032 72   0.515  0.8644
 C - L      0.2022 0.0805 72   2.512  0.0374
 NL - L     0.1491 0.1032 72   1.445  0.3234

Res = Lo:
 contrast estimate     SE df t.ratio p.value
 C - NL     0.1548 0.1032 72   1.500  0.2968
 C - L      0.4854 0.0805 72   6.030  <.0001
 NL - L     0.3306 0.1032 72   3.204  0.0057

P value adjustment: tukey method for comparing a family of 3 estimates
```

```
ph2<-emmeans(lm2, specs="Res", by="Pred")
pairs(ph2)
```

```
Pred = C:
 contrast estimate     SE df t.ratio p.value
 Hi - Lo    -0.440 0.0805 72  -5.470  <.0001

Pred = NL:
 contrast estimate     SE df t.ratio p.value
 Hi - Lo    -0.339 0.1217 72  -2.782  0.0069

Pred = L:
 contrast estimate     SE df t.ratio p.value
 Hi - Lo    -0.157 0.0805 72  -1.951  0.0549
```

5.2 Use `paste()` or `unite()` from `library(tidyr)` to create combined variables

5.2.1 Use `paste()` to combined two factors to form one factor

```
RxP.byTank$ResPred<-as.factor(paste(RxP.byTank$Res, RxP.byTank$Pred,
↪  sep="-"))
```

5.2.2 Use `unite()` to achieve the same result

The function can be found in `library(tidyr)`.

```
library(tidyr)
RxP.byTank <- RxP.byTank %>% unite(col="ResPred", Pred, Res, sep="-",
↪  remove=F)
```

It doesn't make too much sense to run pairwise individual level comparisons so I am skipping the code and results here.

5.3 Use `[-c()]` or `slice()` to subset: excluding outliers

When the explanatory variable is continuous, use `lm()` to run the linear model still, and use `Anova()` to print the F-statistics.

The results from the diagnostic plot suggest the data point at 75, 26, and 49 are outliers. So to improve the model fit, we can subset the dataset to exclude those points.

```
lm4<-lm(log.SVL.final~log.Age.DPO, data=RxP.byTank)
summary(lm4)
```

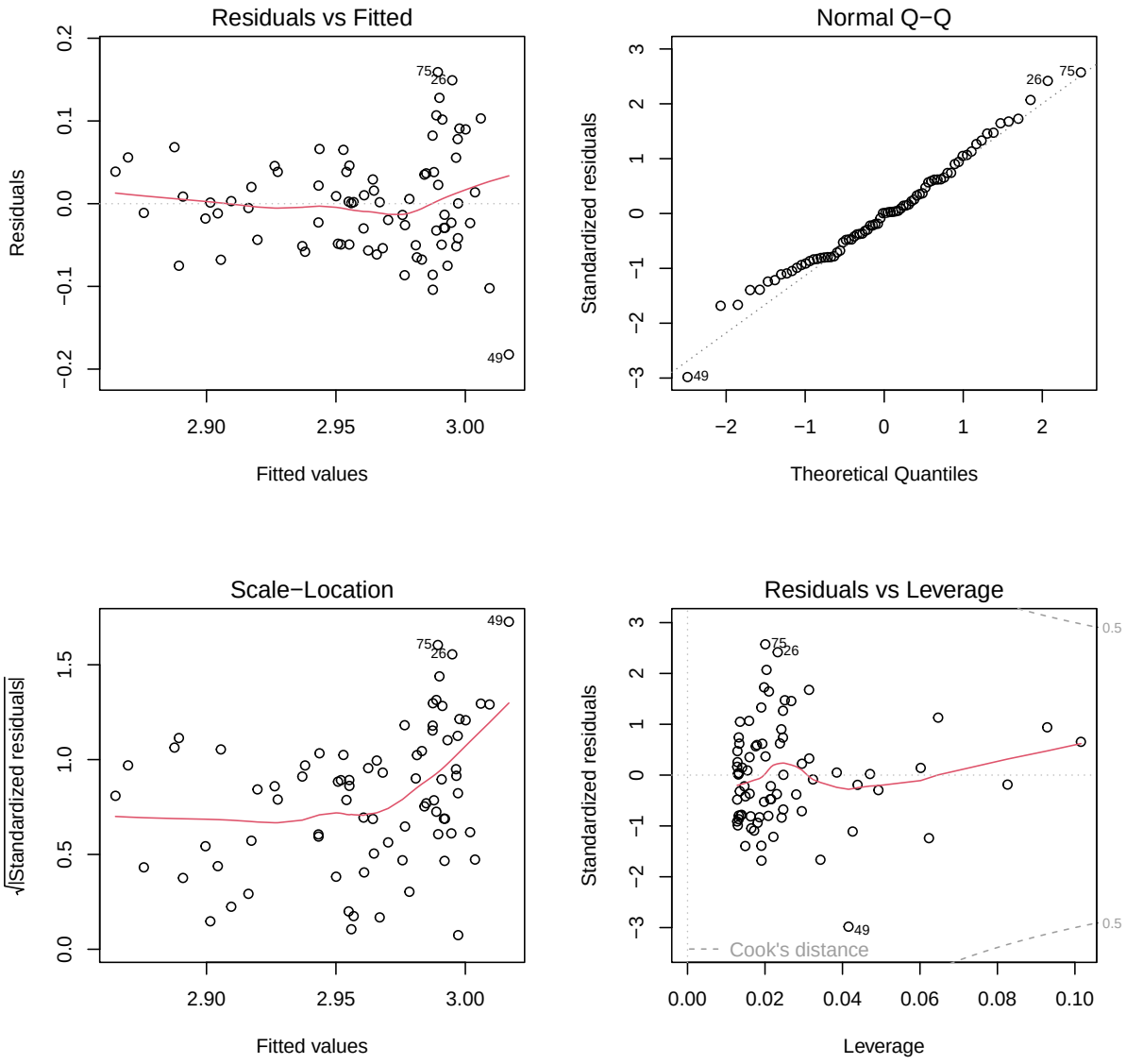
```
Call:
lm(formula = log.SVL.final ~ log.Age.DPO, data = RxP.byTank)

Residuals:
      Min        1Q    Median        3Q       Max
-0.182298 -0.049070  0.000516  0.038342  0.159057

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)  3.44001    0.09211  37.345  < 2e-16 ***
log.Age.DPO -0.11624    0.02232  -5.208 1.58e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.06244 on 76 degrees of freedom
Multiple R-squared:  0.263, Adjusted R-squared:  0.2533
F-statistic: 27.12 on 1 and 76 DF,  p-value: 1.581e-06
```

```r
Anova(lm4)
```

	Sum Sq	Df	F value	Pr(>F)
log.Age.DPO	0.1057572	1	27.12328	1.6e-06
Residuals	0.2963340	76	NA	NA

```r
par(mfrow=c(2,2))
plot(lm4)
```



5.3.1 Use `[-c()]` for subsetting

There are two ways to achieve that, one is to use `[]` to subset the variable:

```
lm4.1<-lm(log.SVL.final~log.Age.DPO, data=RxP.byTank[-c(26,49,75),])
```

5.3.2 Use `slice()` for subsetting

```
temp <- RxP.byTank %>% slice(-c(26,49,75))
lm4.2 <- lm(log.SVL.final ~ log.Age.DPO, data=temp)
```

Either way the result should stay the same:

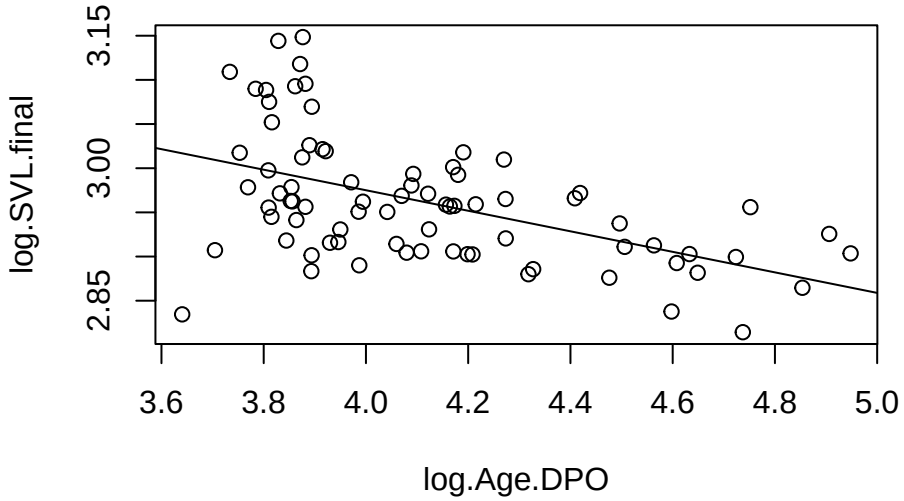
```
knitr::kable(Anova(lm4.1)[c(1,2,3)],digits = 2)
knitr::kable(Anova(lm4.2)[c(1,2,3)],digits = 2)
```

	Sum Sq	Df	F value
log.Age.DPO	0.10	1	34.46
Residuals	0.22	73	NA

	Sum Sq	Df	F value
log.Age.DPO	0.10	1	34.46
Residuals	0.22	73	NA

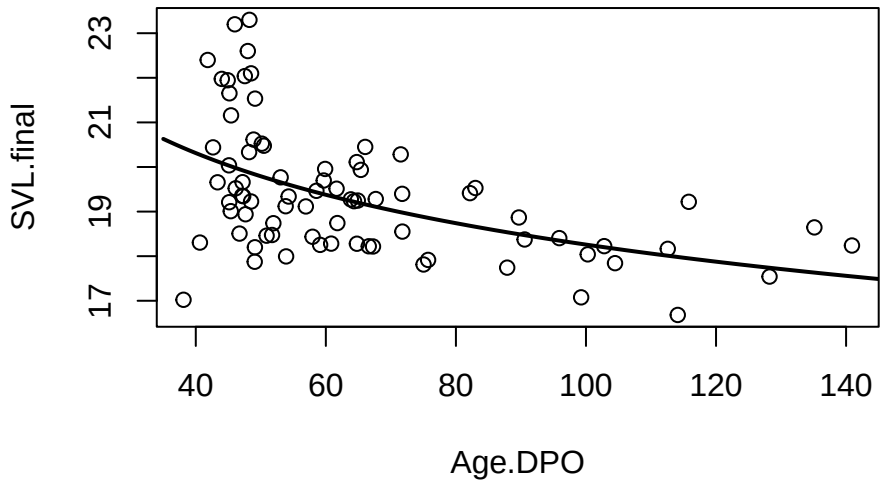
5.4 Use `abline()` to draw a line in scattered plot

```
plot(log.SVL.final~log.Age.DPO, data=RxP.byTank)
abline(3.44001,-0.11624) # the two numbers are the intercept estimate
↪  and the slope estimate
```



5.5 Use `lines()` to draw the line manually

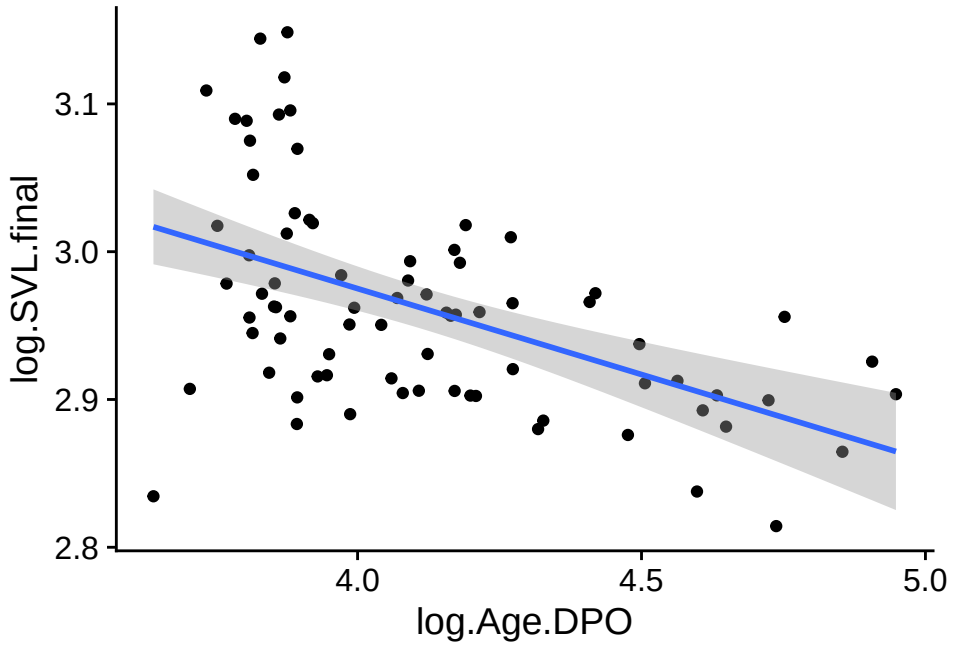
```
#plot raw values
plot(SVL.final~Age.DPO, data=RxP.byTank)
#make a vector of the predicted SVL's
SVL.line<-exp(3.44001-0.11624*log(35:145))
#Use the predicted values to plot a line
lines(x=35:145, y=SVL.line, lwd=2)
```



5.5.1 Use `qplot()` to achieve the goal

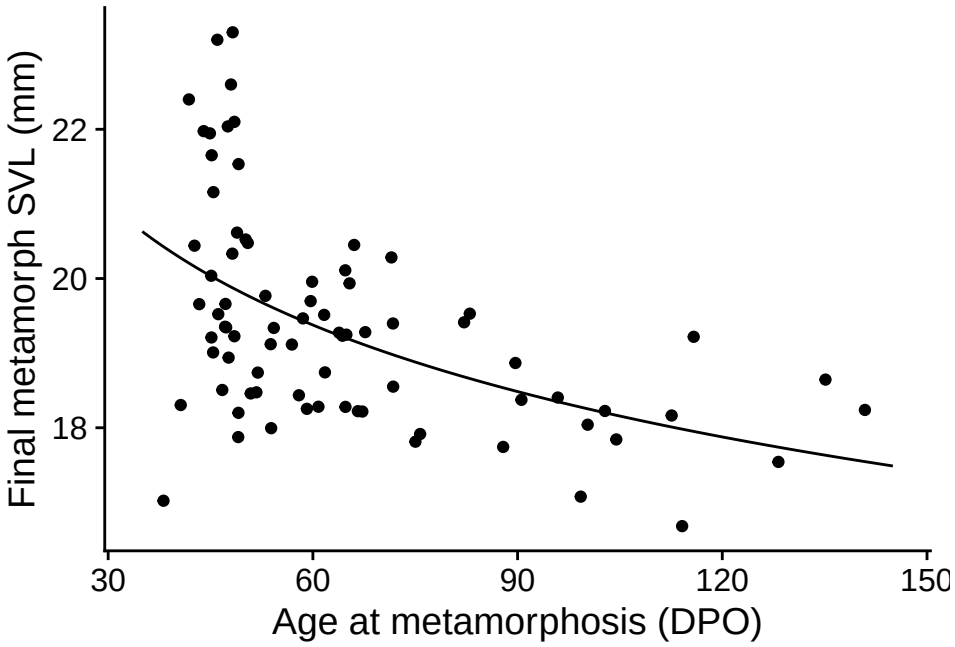
```
qplot(data=RxP.byTank,
      x=log.Age.DPO,
      y=log.SVL.final,
      geom='point')+
    geom_smooth(method='lm')+
    theme_cowplot()
```

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



5.5.2 Use `ggplot()` to achieve the goal

```
#First, make a data frame of the x- and y-coordinates
SVL.line<-data.frame(X = 35:145,
                     Y = exp(3.44001-0.11624*log(35:145)))

ggplot()+ #Leave out the data for now
  geom_point(data = RxP.byTank,
             aes(x = Age.DPO, y = SVL.final))+ #points
  geom_line(data = SVL.line,
            aes(x = X, y = Y))+ #the line
  ylab("Final metamorph SVL (mm)")+ #y-axis labels
  xlab("Age at metamorphosis (DPO)")+ #x-axis labels
  theme_cowplot() #make it look nice
```



5.6 Use `emtrends()` to test the differences in slope in ANCOVA

When doing analysis of covariance, `anova()` is misleading and one should use `Anova()`. This is because when calculating significance, the `anova()` calculate it in s a stepwise manner, hence when calculating the significance for the first variable, it doesn't take into account the other variables. This is not a problem for `Anova()`

```
#Now let's make an ANCOVA that looks at the
#effects of categorical and continuous data together
lm5<-lm(log.SVL.final~log.Age.DPO*Pred, data=RxP.byTank)
summary(lm5)
```

```
Call:
lm(formula = log.SVL.final ~ log.Age.DPO * Pred, data = RxP.byTank)

Residuals:
      Min        1Q    Median        3Q       Max
-0.122214 -0.029839  0.002357  0.036826  0.122903

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)        3.08467    0.12387  24.902   <2e-16 ***
```

```
log.Age.DPO         -0.03974    0.02890  -1.375   0.1733
PredNL              -0.05606    0.21020  -0.267   0.7905
PredL                0.67183    0.24458   2.747   0.0076 **
log.Age.DPO:PredNL   0.02000    0.04982   0.401   0.6893
log.Age.DPO:PredL   -0.14883    0.06090  -2.444   0.0170 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.05227 on 72 degrees of freedom
Multiple R-squared:  0.5107,    Adjusted R-squared:  0.4767
F-statistic: 15.03 on 5 and 72 DF,  p-value: 4.324e-10
```

```
knitr::kable(anova(lm5)[c(3,4)],digits = 2)
knitr::kable(Anova(lm5)[c(3,4)],digits = 2)
```

	Mean Sq	F value
log.Age.DPO	0.11	38.70
Pred	0.04	14.62
log.Age.DPO:Pred	0.01	3.61
Residuals	0.00	NA

	F value	Pr(>F)
log.Age.DPO	7.28	0.01
Pred	14.62	0.00
log.Age.DPO:Pred	3.61	0.03
Residuals	NA	NA

To conduct Tukey post hoc comparisons, use `emmeans()`, the results will average over the effect of `log.Age.DPO`, which probably won't be very useful because there might have interaction.

A more useful function is `emtrends()` which can compares the slopes of a variable(`var=`) between different levels (`specs=`):

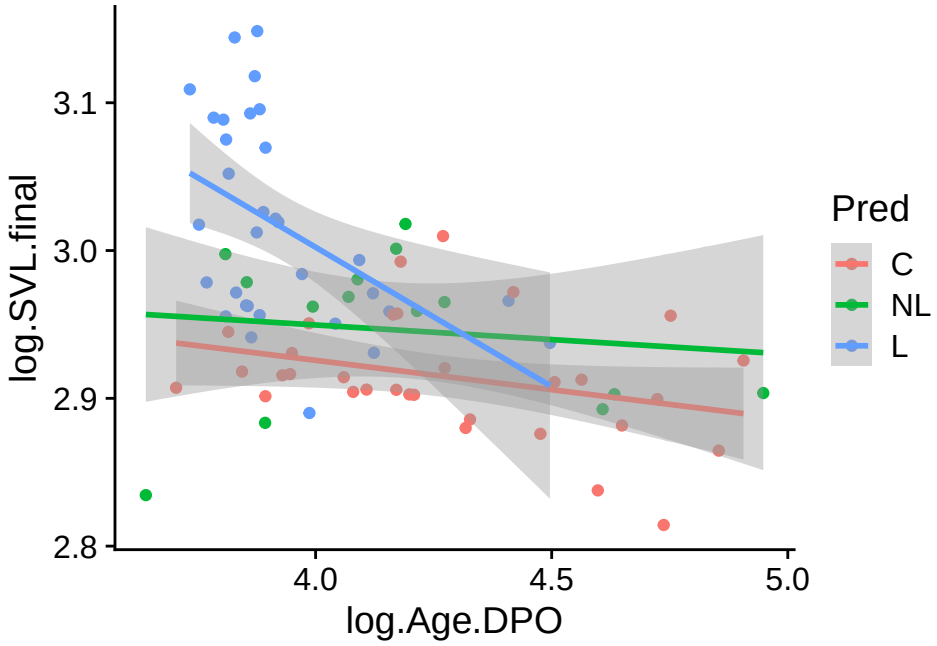
```
#Use emtrends() to test for differences in slopes
ph4<-emtrends(lm5, specs="Pred", var="log.Age.DPO")
pairs(ph4)
```

```
 contrast estimate     SE df t.ratio p.value
 C - NL     -0.020 0.0498 72  -0.401  0.9151
 C - L       0.149 0.0609 72   2.444  0.0443
 NL - L      0.169 0.0672 72   2.511  0.0376

P value adjustment: tukey method for comparing a family of 3 estimates
```

It's also clear from a `qplot()` that the slops are different across three different group (mind that the `se=T` creates a standard error band around the fitted line, turn it off by set it to `se=F`; the `fullrange = F` plots the line at the extend of the data; if it's set to true, the lines will be plotted in full range):

```
qplot(data=RxP.byTank,
      x=log.Age.DPO,
      y=log.SVL.final,
      geom="point",
      col=Pred)+
    geom_smooth(method="lm", se=T, fullrange = F)+
    theme_cowplot()
```

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



5.7 Use `predict()` to get predicted results from fitted model

The output from the `predit()` function is a list, which can holds multiple objects at different lengths.

```
range <- range(RxP.byTank$log.Age.DPO)
lm4.newdata<-data.frame("log.Age.DPO" = seq(from=range[1],
                                            to=range[2],
                                            by=0.01))
lm4.predicted<-predict(lm4,
                       newdata=lm4.newdata,
                       se.fit=T)
str(lm4.predicted)
```

```
List of 4
 $ fit           : Named num [1:131] 3.02 3.02 3.01 3.01 3.01 ...
  ..- attr(*, "names")= chr [1:131] "1" "2" "3" "4" ...
 $ se.fit        : Named num [1:131] 0.0127 0.0125 0.0124 0.0122 0.012 ...
  ..- attr(*, "names")= chr [1:131] "1" "2" "3" "4" ...
 $ df            : int 76
 $ residual.scale: num 0.0624
```

5.7.1 Use `expand.grid()` to create every possible conbinations of input

Instead of using `paste()` or `unite()` mentioned Section 5.2 to create combinaed single variable based on input, using `expland.grid()` is a neat and more suitable function in this senario.

```
#First create a data frame to provide values to predict()
lm5.newdata<-expand.grid("log.Age.DPO"=log(35:145),
                         Pred=c("C","NL","L"))
```

Now fit the data and create a data frame ready for grpah plot:

```
#Now we create a data frame that houses the predicted values
#and matches the variables in our data frame lm5
lm5.predicted<-predict(lm5, newdata=lm5.newdata, se.fit=T)
#First, exponentiate the fitted values
lm5.newdata$predict.SVL<-exp(lm5.predicted$fit)

#Next, calculate the upper and lower bounds of the CIs
lm5.newdata$predict.SVL.CIupper<-exp(lm5.predicted$fit+
                                       lm5.predicted$se.fit)

lm5.newdata$predict.SVL.CIlower<-exp(lm5.predicted$fit-
```

```
                                    lm5.predicted$se.fit)

#Lastly, exponentiate the age data so it is back
#on the original scale
lm5.newdata$Age.DPO<-exp(lm5.newdata$log.Age.DPO)

str(lm5.newdata)
```

```
'data.frame':   333 obs. of  6 variables:
 $ log.Age.DPO        : num  3.56 3.58 3.61 3.64 3.66 ...
 $ Pred               : Factor w/ 3 levels "C","NL","L": 1 1 1 1 1 1 1 1 1 1 ...
 $ predict.SVL        : num  19 19 18.9 18.9 18.9 ...
 $ predict.SVL.CIupper: num  19.4 19.4 19.3 19.3 19.3 ...
 $ predict.SVL.CIlower: num  18.6 18.5 18.5 18.5 18.5 ...
 $ Age.DPO            : num  35 36 37 38 39 40 41 42 43 44 ...
 - attr(*, "out.attrs")=List of 2
  ..$ dim     : Named int [1:2] 111 3
  .. ..- attr(*, "names")= chr [1:2] "log.Age.DPO" "Pred"
  ..$ dimnames:List of 2
  .. ..$ log.Age.DPO: chr [1:111] "log.Age.DPO=3.555348" "log.Age.DPO=3.583519" "log.Age.DPO=
  .. ..$ Pred       : chr [1:3] "Pred=C" "Pred=NL" "Pred=L"
```

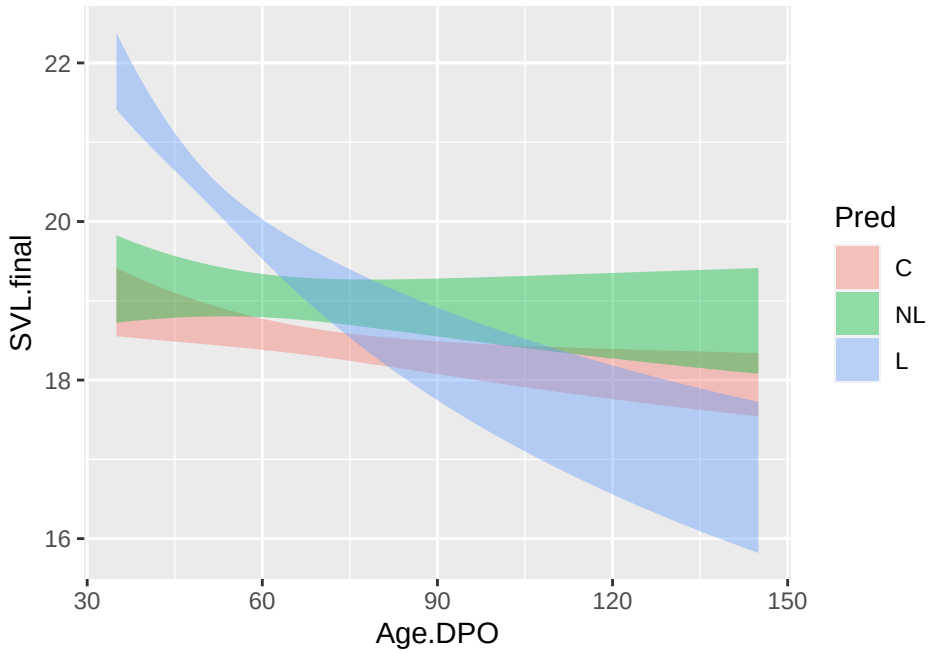
5.8 Use `ggplot()` to plot the predicted results

The main difference between `qplot()` and `ggplot()` functions is that in ggplot() you don't specify the geom in the first function, instead you set a separate geom function and add it to the original function. It is also important to note that the order of the objects matters, such as plotting the ribbon first, then the points, and then the regression line. Additionally, everything added to the original function with ggplot2 adjusts how the plot looks and the plot fits together. For example, adding a legend will change the entire plot to fit the legend without it overlapping any other elements.

First add the ribbon:

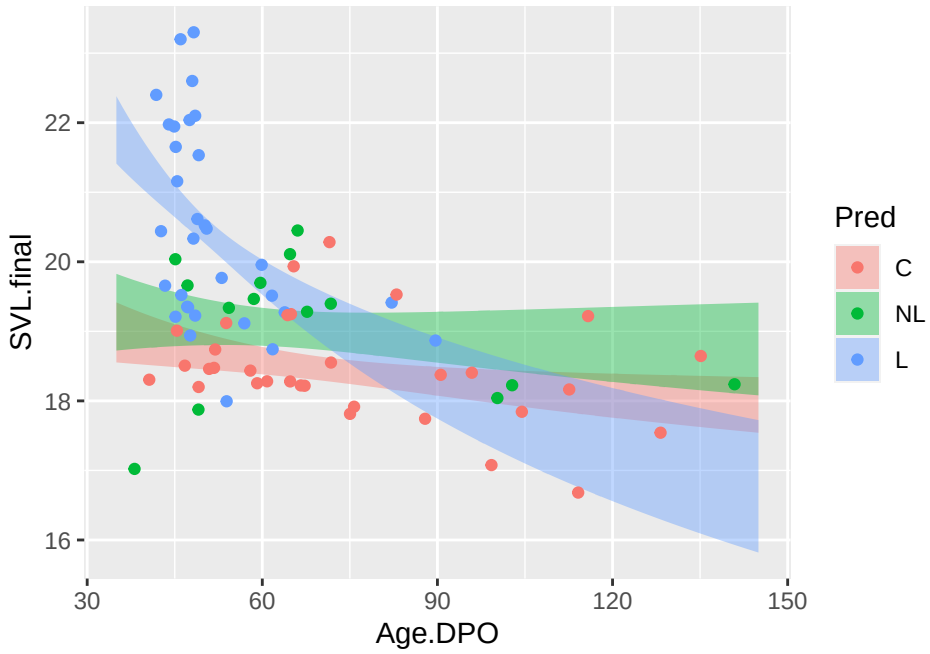
```
ggplot(data=RxP.byTank, aes(x=Age.DPO,
                            y=SVL.final,
                            col=Pred))+
    geom_ribbon(data=lm5.newdata,
                inherit.aes=F,
                aes(x=Age.DPO,
```

```
                    ymax=predict.SVL.CIupper,
                    ymin=predict.SVL.CIlower,
                    fill=Pred),
                alpha=0.4)
```



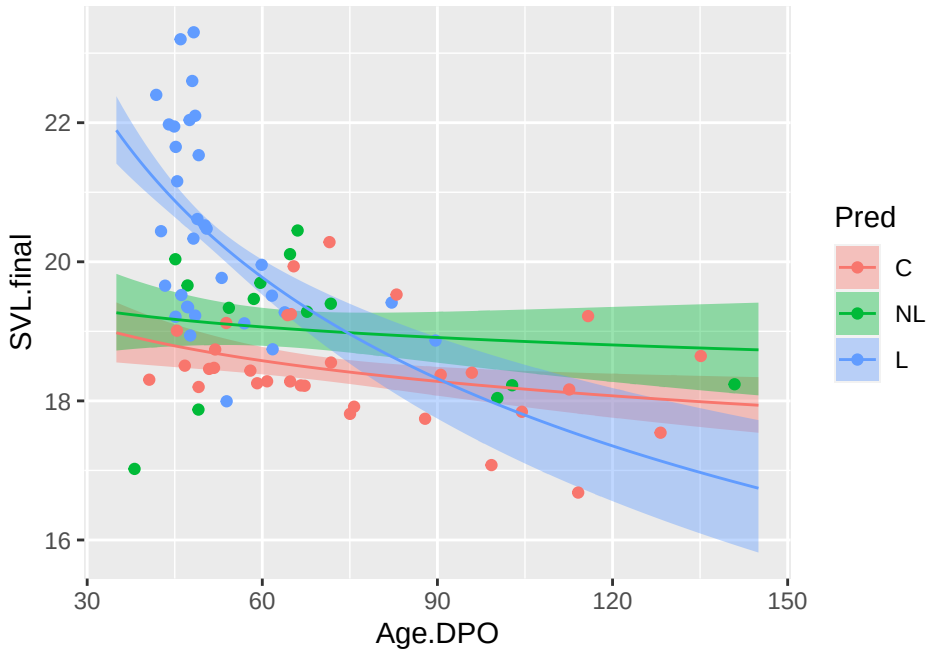
Second add the point:

```
ggplot(data=RxP.byTank, aes(x=Age.DPO,
                            y=SVL.final,
                            col=Pred))+
    geom_ribbon(data=lm5.newdata,
                inherit.aes=F,
                aes(x=Age.DPO,
                    ymax=predict.SVL.CIupper,
                    ymin=predict.SVL.CIlower,
                    fill=Pred),
                alpha=0.4)+
    geom_point()
```



Then add the line:

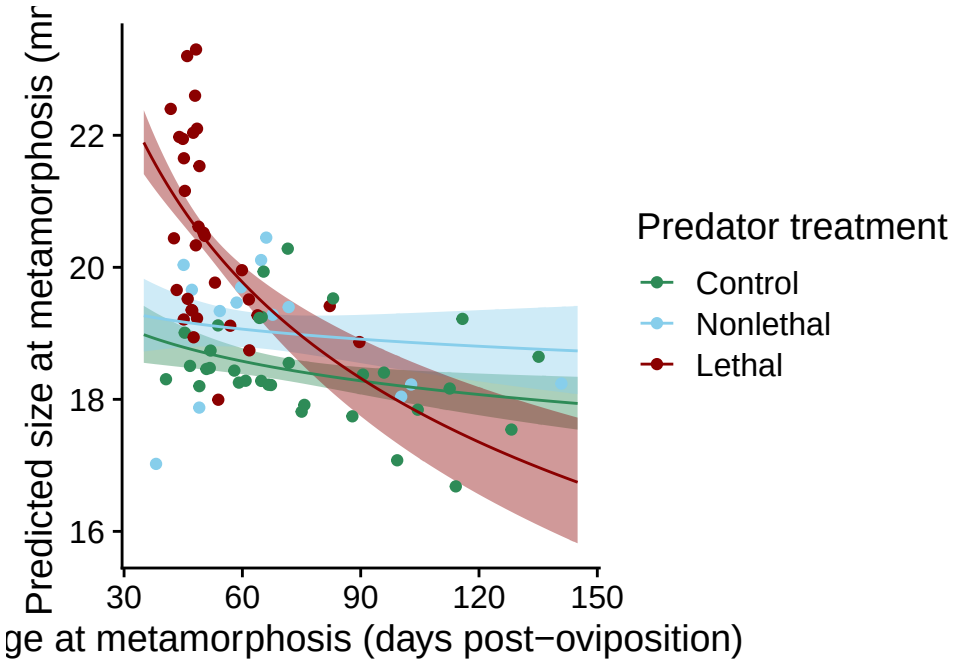
```
ggplot(data=RxP.byTank, aes(x=Age.DPO,
                            y=SVL.final,
                            col=Pred))+
    geom_ribbon(data=lm5.newdata,
                inherit.aes=F,
                aes(x=Age.DPO,
                    ymax=predict.SVL.CIupper,
                    ymin=predict.SVL.CIlower,
                    fill=Pred),
                alpha=0.4)+
    geom_point()+
    geom_line(data=lm5.newdata,
              inherit.aes=F,
              aes(x=Age.DPO,
                  y=predict.SVL,
                  col=Pred))
```



Finally add all the sugar:

```
ggplot(data=RxP.byTank, aes(x=Age.DPO,
                            y=SVL.final,
                            col=Pred))+
    geom_ribbon(data=lm5.newdata,
                inherit.aes=F,
                aes(x=Age.DPO,
                    ymax=predict.SVL.CIupper,
                    ymin=predict.SVL.CIlower,
                    fill=Pred),
                alpha=0.4)+
    geom_point()+
    geom_line(data=lm5.newdata,
              inherit.aes=F,
              aes(x=Age.DPO,
                  y=predict.SVL,
                  col=Pred))+
    scale_fill_manual(values=c("seagreen",
                               "skyblue",
                               "dark red"),
                      guide=F)+
```

```
    scale_color_manual(values=c("seagreen",
                                "skyblue",
                                "dark red"),
                       labels=c("Control",
                                "Nonlethal",
                                "Lethal"),
                       name="Predator treatment")+
    ylab("Predicted size at metamorphosis (mm)")+
    xlab("Age at metamorphosis (days post-oviposition)")+
    theme_cowplot()
```



6 Generalized Linear Model

GLM works by transforming data to normal scale through `link` function, the common link function are built in along with the error function:

- binomial(link = "logit")
- gaussian(link = "identity")
- poisson(link = "log")
- Gamma(link = "inverse")
- inverse.gaussian(link = "1/mu^2")

- quasi(link = "identity", variance = "constant")
- quasibinomial(link = "logit")
- quasipoisson(link = "log")

Going back to Section 1.3.1, we need to add additional information to the `RxP.byTank` data that we are working with:

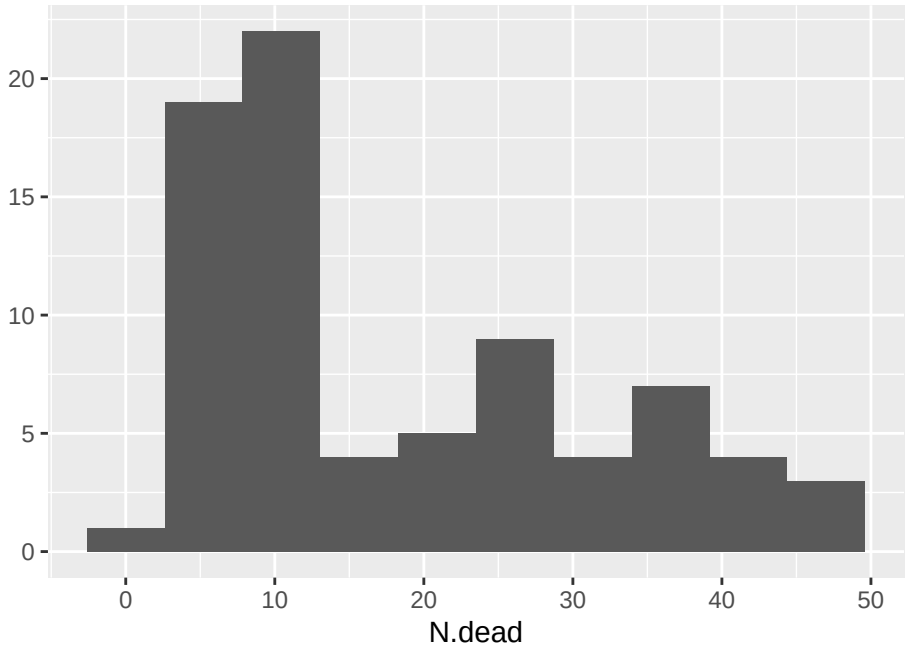
```
temp <- RxP.clean %>%
        group_by(Tank.Unique) %>%
        summarize(N.alive = length(Ind))
RxP.byTank$N.alive<-temp$N.alive
RxP.byTank$N.dead<-50-RxP.byTank$N.alive
glimpse(RxP.byTank)
```

```
Rows: 78
Columns: 19
$ Block                 <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, ~
$ Tank.Unique           <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1~
$ ResPred               <chr> "NL-Hi", "C-Hi", "C-Hi", "L-Lo", "NL-Hi", "L-Hi"~
$ Pred                  <fct> NL, C, C, L, NL, L, NL, C, L, C, NL, L, NL, NL, ~
$ Hatch                 <fct> L, E, L, L, E, E, L, E, L, L, E, E, L, L, E, L, ~
$ Res                   <fct> Hi, Hi, Hi, Lo, Hi, Hi, Lo, Lo, Hi, Lo, Lo, Lo, ~
$ Age.DPO               <dbl> 47.19149, 45.38095, 53.82222, 56.92308, 64.75000~
$ Age.FromEmergence     <dbl> 13.19149, 11.38095, 19.82222, 22.92308, 30.75000~
$ SVL.initial           <dbl> 19.42553, 18.40476, 18.92667, 18.82692, 19.71500~
$ Tail.initial          <dbl> 4.834043, 5.369048, 4.802222, 4.634615, 5.435000~
$ SVL.final             <dbl> 19.65957, 19.00952, 19.12000, 19.11538, 20.11000~
$ Mass.final            <dbl> 0.4178723, 0.3821429, 0.4117778, 0.3823077, 0.48~
$ Resorb.days           <dbl> 3.489362, 3.785714, 3.511111, 3.653846, 4.225000~
$ log.SVL.final         <dbl> 2.978564, 2.944940, 2.950735, 2.950493, 3.001217~
$ log.Age.FromEmergence <dbl> 2.579572, 2.431941, 2.986804, 3.132144, 3.425890~
$ log.Age.DPO           <dbl> 3.854214, 3.815092, 3.985686, 4.041701, 4.170534~
$ log.mass.final        <dbl> -0.8725793, -0.9619608, -0.8872714, -0.9615295, ~
$ N.alive               <int> 47, 42, 45, 26, 40, 8, 43, 39, 23, 44, 42, 22, 4~
$ N.dead                <dbl> 3, 8, 5, 24, 10, 42, 7, 11, 27, 6, 8, 28, 6, 6, ~
```

Plot histogram to see which distribution fit the data best:

```
qplot(data=RxP.byTank,
      x=N.dead,
      geom="histogram",
```

```
       bins=10)
```



Also do statistical testing using `fitdistr()`:

```
#Create 4 objects for evaluating the best
#error distribution for the N.dead variable
fit1<-fitdistr(RxP.byTank$N.dead, "normal")
fit2<-fitdistr(RxP.byTank$N.dead, "lognormal")
fit3<-fitdistr(RxP.byTank$N.dead, "Poisson")
fit4<-fitdistr(RxP.byTank$N.dead, "negative binomial")
#Use AIC() to compare the fit to each distribution
AIC(fit1,fit2,fit3,fit4)
```

	df	AIC
fit1	2	625.4562
fit2	2	598.1027
fit3	1	1060.4417
fit4	2	598.3479

6.1 Use `glm()` to fit the model

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

If modeling using poisson distribution, one need to worry about over dispersion. Ideally, the *Residual deviance* shouldn't be more than twice the degree of freedom. Which isn't the case of the model fitting:

```
summary(glm.p)
```

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

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

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

(Dispersion parameter for poisson family taken to be 1)

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

Number of Fisher Scoring iterations: 5
```

This is understandable, because the possion distribution assumes the mean and variance of the variable equals to each other, which isn't the case of the example data.

```
print(mean(RxP.byTank$N.dead))
```

```
[1] 18.03846
```

```
print(var(RxP.byTank$N.dead))
```

```
[1] 171.1284
```

6.1.1 Use `cbind()` to create balanced data for *binomial error family*

Binomial data typically consists of zeros and ones but R allows modeling balanced data with a binomial error family by coding the data as a two-column table of wins and losses or animals that lived or died. The two columns must add up to the total starting number of individuals. It is important to note that coding the data as individuals that died and the total starting number will result in incorrect results. The two columns can be combined using the cbind() function to form a two-column table.

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

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

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

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   1.72483    0.09868  17.480  < 2e-16 ***
ResLo        -0.44454    0.13070  -3.401 0.000671 ***
PredNL       -0.82250    0.15380  -5.348  8.9e-08 ***
```

```
PredL        -1.73483    0.12140 -14.291  < 2e-16 ***
ResLo:PredNL  0.38950    0.21120   1.844 0.065146 .
ResLo:PredL  -0.07768    0.16566  -0.469 0.639134
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

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

Number of Fisher Scoring iterations: 4
```

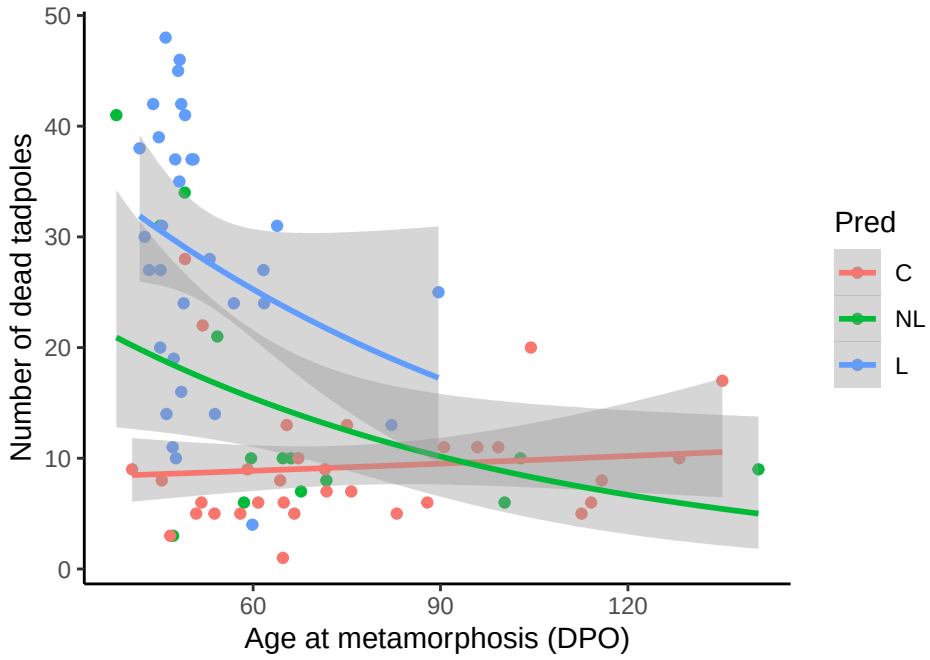
6.2 Model specification shortcut: *, -

- `Res\*Pred` is a shortcut for `Res+Pred+Res:Pred`.
- `Res*Pred-Res:Pred` indicates removing interaction between the two variables.

6.3 `predict()` the results from `glm()`

```
# Model fitting
glm.negb2 <- glm.nb(N.dead~Age.DPO*Pred, RxP.byTank)
# Create the table for predicted data
predicted.data<-expand.grid(Age.DPO=40:140,
                            Pred=c("C","NL","L"),
                            N.dead=NA)
# Predicting the number of death per tank
predicted.data$N.dead<-predict(glm.negb2,
                            newdata=data.frame(predicted.data),
                            type="response")
# Generate a ggplot based on the predicted result
qplot(x=Age.DPO, y=N.dead, data=RxP.byTank, col=Pred,
      ylab="Number of dead tadpoles",
      xlab="Age at metamorphosis (DPO)")+
  geom_smooth(method=glm.nb)+
  theme_classic()
```

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



7 Mixed effect model

Mixed-effects models are a way to control for variation in data by partitioning variance into two groups: fixed effects and random effects. Fixed effects affect the mean of the data, while random effects primarily affect the variance. Mixed models allow for the use of data from non-independent observations and provide a clearer picture of the effects of interest by reducing noise in the data.

They are a form of *hierarchical model* that builds regressions from each level of random effects instead of only calculating a single regrssion. The submodels are pooled together while variances between them are accounted for while preserving and califying the fixed effect at the same time.

There are several packges for fitting mixed effect model:

1. lme4 : `lmer()`

- This is the most widely used and supported function for conducting mixed-effects models. The coding is similar to the lm() and glm() functions, but with added components to account for random effects.

2. glmmadmb :

- This package uses different algorithms to estimate the model compared to the lme4 package and may be useful in cases when lme4 fails to run. However, it is slower than lme4.

3. glmmTMB :`glmmTMB()`

- This package has emerged as a robust alternative to lme4 and glmmadmb. It also allows us to run zero-inflation models, which are useful for handling lots of zeros in the dataset.

4. MCMCglmm :

- This package uses a Bayesian framework to estimate the parameters, which is beyond the scope of this course but is a useful resource for Bayesian methods.

7.1 Nested random intercepts

The nested random intecept mixted effect model is used when some effects fit entirly witghin another effect:

`...+(random_slope|random_intercept1/random_intercept2)...`

```
library(lme4)
RxP.clean$log.Age.DPO<-log(RxP.clean$Age.DPO)
#Code the lmm just like a regular lm,
#but with the random effect

lmm1<-lmer(log.Age.DPO~Res*Pred+(1|Block/Tank.Unique),
           data=RxP.clean)

summary(lmm1)
```

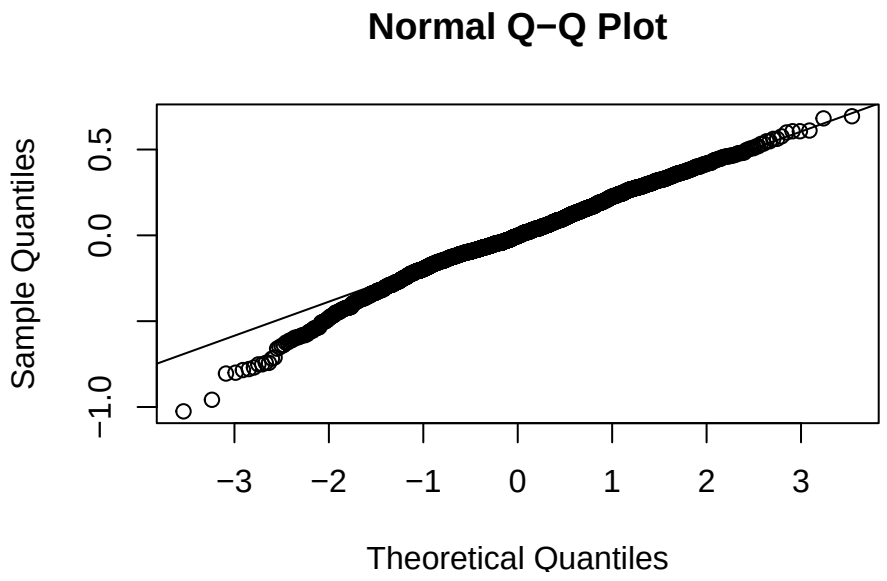
```
Linear mixed model fit by REML ['lmerMod']
Formula: log.Age.DPO ~ Res * Pred + (1 | Block/Tank.Unique)
   Data: RxP.clean

REML criterion at convergence: -152.3

Scaled residuals:
    Min      1Q  Median      3Q     Max
-4.6111 -0.5561 -0.0161  0.6464  3.1204

Random effects:
```

```
 Groups            Name        Variance Std.Dev.
 Tank.Unique:Block (Intercept) 0.038365 0.19587
 Block             (Intercept) 0.008768 0.09364
 Residual                      0.049453 0.22238
Number of obs: 2493, groups:  Tank.Unique:Block, 78; Block, 8

Fixed effects:
             Estimate Std. Error t value
(Intercept)   4.03758    0.05973  67.597
ResLo         0.42107    0.07036   5.984
PredNL       -0.03919    0.09311  -0.421
PredL        -0.19079    0.07095  -2.689
ResLo:PredNL -0.09945    0.12809  -0.776
ResLo:PredL  -0.26933    0.10096  -2.668

Correlation of Fixed Effects:
            (Intr) ResLo  PredNL PredL  RL:PNL
ResLo       -0.588
PredNL      -0.445  0.377
PredL       -0.583  0.495  0.374
ResLo:PrdNL  0.323 -0.549 -0.689 -0.272
ResLo:PredL  0.410 -0.697 -0.264 -0.703  0.383
```

There are two random effects in this model, `Block` and `Tank.Unique` nested within the block. The variance column in the random effects model tells how much random efects are accounted for by the block and tank within the block. Yet, there's no good metrics to determin what's a large or small sffect so usually the variances of the random effect are overlooked.

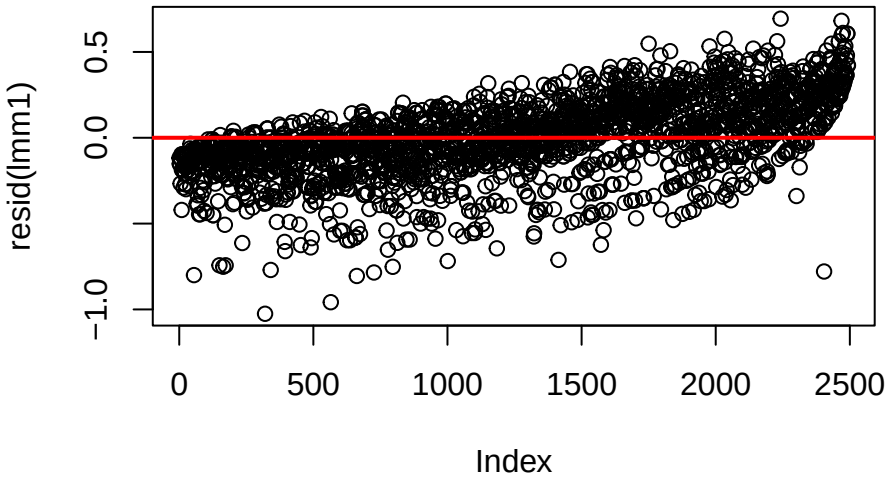
7.1.1 Building diagnostic plot ourselves using `qqnorm()` and `qqline()`

```
qqnorm(resid(lmm1));
qqline(resid(lmm1))
```



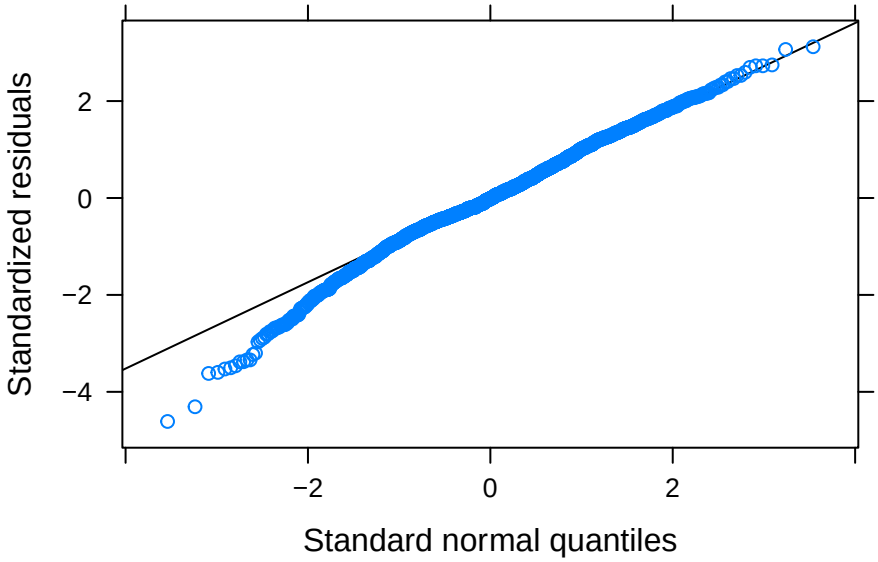
Build the residual:

```
plot(resid(lmm1))

abline(h=0, col='red', lwd=2)
```



7.1.2 Use `qqmath()` from `library(lattice)` to draw the Q-Q plot

```
library(lattice)
qqmath(lmm1)
```



7.2 Use `anove()` to compare between models and conduct likelihood ratio tests

```
lmm2<-lmer(log.Age.DPO~Res+Pred+(1|Block/Tank.Unique),
           data=RxP.clean)
anova(lmm1,lmm2)
```

```
refitting model(s) with ML (instead of REML)
```

	npar	AIC	BIC	logLik	deviance	Chisq	Df	Pr(>Chisq)
lmm2	7	-153.1077	-112.3590	83.55387	-167.1077	NA	NA	NA
lmm1	9	-156.4413	-104.0501	87.22064	-174.4413	7.333535	2	0.025559

Because the mixed effect model is hierarchical, it's impossible to accurately calculate the degree of freedom hence simply being ignored in the report.

8 Generalized Linear Mixed Model (GLMM)

8.1 Use `glmer.nb()` to fit the model

```
glmm1<-glmer.nb(N.dead~Pred*Res*Hatch+(1|Block), data=RxP.byTank)
```

Sometimes the model won't work because the model is not solved properly. Usually it's the parameters fail to converge to a single optimal solution.

Several solutions might help get around the issue:

1. ignore the warning
2. change the *optimizer* or the *number of iterations* the model runs. But in this case, neither will be helpful. Here is another example where the `optimizer` has changed.

```
RxP.clean$log.SVL.final <- log(RxP.clean$SVL.final)
lmm5<-lmer(log.SVL.final~log.Age.DPO*Pred+(1|Block/Tank.Unique),
            data=RxP.clean, control = lmerControl(optimizer
            ↪  ="Nelder_Mead"))
```

3. try a different function, such as `glmmTMB()`

There's a nother warning includes `?isSingular`, this means the estimated variance-covariance matrix for the random effect part is close to 0; usually it's not much of a big deal.

8.2 Use `glmmTMB()` from `library(glmmTMB)` to fit the model

The `glmmTMB` package is a relatively new and still developing package for fitting generalized linear mixed effects models. Unlike other packages such as `lmer`, the coding for fitting a mixed effects model in glmmTMB is slightly different. The glmmTMB() function requires that the random effects be specified as factors and also requires the specification of a family argument, which defines the error distribution. In this case, the family argument is set to `"nbinom2"` to indicate a negative binomial error distribution.

```
library(glmmTMB)
#First, convert Block to a factor
RxP.byTank$Block.factor<-as.factor(RxP.byTank$Block)
#Next, run the model. Note that the function and family
#specifications are different.
glmm7<-glmmTMB(N.dead~Pred+Res+Hatch+(1|Block.factor),
```

```
                family='nbinom2', data=RxP.byTank)
#Create a series of models that only each differ
#in one predictor from one another
glmm7.1<-glmmTMB(N.dead~Pred+Res+(1|Block.factor),
                family='nbinom2', data=RxP.byTank)
glmm7.2<-glmmTMB(N.dead~Pred+Hatch+(1|Block.factor),
                family='nbinom2', data=RxP.byTank)
glmm7.3<-glmmTMB(N.dead~Res+Hatch+(1|Block.factor),
                family='nbinom2', data=RxP.byTank)

#Run the likelihood ratio tests
anova(glmm7,glmm7.1)#sig of Hatch
```

	Df	AIC	BIC	logLik	deviance	Chisq	Chi Df	Pr(>Chisq)
glmm7.1	6	552.9603	567.1006	-270.4802	540.9603	NA	NA	NA
glmm7	7	547.6082	564.1052	-266.8041	533.6082	7.352075	1	0.0066985

```
anova(glmm7,glmm7.2) #sig of Res
```

	Df	AIC	BIC	logLik	deviance	Chisq	Chi Df	Pr(>Chisq)
glmm7.2	6	550.5529	564.6932	-269.2765	538.5529	NA	NA	NA
glmm7	7	547.6082	564.1052	-266.8041	533.6082	4.944689	1	0.0261709

```
anova(glmm7,glmm7.3) #sig of Res
```

	Df	AIC	BIC	logLik	deviance	Chisq	Chi Df	Pr(>Chisq)
glmm7.3	5	599.3997	611.1832	-294.6998	589.3997	NA	NA	NA
glmm7	7	547.6082	564.1052	-266.8041	533.6082	55.79145	2	0

The results are consistent with previous finding, where we can see all three factors plays important role in the survival of the metamorphs.

> **Note**
>
> Balancing random effects in a mixed-effects model is important because it helps ensure that the model runs smoothly and produces accurate results. When random effects are unbalanced, meaning that they only apply to one treatment or subset of the data, this can cause problems in the model and result in incorrect results. To mitigate these issues, it is best to design experiments so that the experimental units are evenly distributed across the levels of the random effects. This allows the model to effectively account for the effects of these variables on the outcome.

8.3 Set `REML=F` in the `lmer()` to make the mixed effect model comparable with `lm()` model fit

```
lm6<-lm(log.Age.DPO~Res*Pred, data=RxP.clean)
#Create a mixed model, but without using REML
lmm1<-lmer(log.Age.DPO~Res*Pred+(1|Block/Tank.Unique),
           data=RxP.clean, REML = F)
#Compare two models, one with random effects
#and one without
AIC(lmm1,lm6)
```

	df	AIC
lmm1	9	-156.4413
lm6	7	1113.8245

In mixed-effects models, restricted maximum likelihood (REML) is a technique used by the `lme4` package to fit mixed-effects models. The regular linear models and generalized linear models are fitted using a standard maximum likelihood approach. If a comparison is made between a linear model and a linear mixed-effects model, the mixed model should be fitted without using REML by specifying the `REML = F` argument in the model.

8.4 Estimating marginal means with mixed models

Mixed-effects models remove variation in the data due to the specified random effects. This can result in substantial differences between the model estimates and raw data, especially if the random effects have a strong influence on the data. To accurately represent the results of the data analysis, it's essential to plot the estimated marginal means obtained after controlling for the random effects. This can be done using the emmeans package in R, which calculates the means of the final metamorph SVL across different treatments.

8.4.1 Use `fixef()` to access the coefficeint of the fixed effect model

```
#First, calculate the means from the raw data
RxP.clean %>%
    group_by(Pred) %>%
    summarize(mean.SVL.final = mean(SVL.final))
```

Pred	mean.SVL.final
C	18.46475
NL	19.20223
L	19.83501

```
#Next, run a mixed model analyzing log.SVL.final
lmm1<-lmer(log.SVL.final ~ Pred+(1|Block/Tank.Unique),
           data=RxP.clean)

#We can view just the parameters with the fixef() function
#Calculate Control treatment size
exp(fixef(lmm1)[1])
```

```
(Intercept)
   18.37956
```

```
#Calculate Nonlethal treatment size
exp(fixef(lmm1)[1]+fixef(lmm1)[2])
```

```
(Intercept)
   18.77424
```

```
#Calculate Lethal treatment size
exp(fixef(lmm1)[1]+fixef(lmm1)[3])
```

```
(Intercept)
   20.20379
```

8.4.2 Use `emmeans()` to predict the means and their s.e.

```
ph1<-emmeans(lmm1, specs='Pred')
summary(ph1)
```

Pred	emmean	SE	df	lower.CL	upper.CL
C	2.911239	0.0124075	11.15607	2.883977	2.938501
NL	2.932486	0.0163869	25.65647	2.898780	2.966192
L	3.005870	0.0127442	12.38998	2.978199	3.033541

9 Advanced Data Manipulation

9.1 The `tidyverse` package

All three packages belong to the tidyverse:

- ggplot2 : visualization
- dplyr : data manipulation
- tidyr : data clean

9.2 Nemaespace resolution using `package::function()` syntax

When two packages have the same function name, the second package won't be used by default, leading to a conflict. To solve this, you can specify the package name followed by two colons before the function name to use the desired version of the function.

Example, both `MASS` and `dplyr` has function `select()`, since `MASS` is first called, the `select()` function by default is from the `MASS` package, in order to use the `select()` from the `dplyr` package instead, use

```
dplyr::select()
```

9.3 Use `group_by()` and `summarize()` to summerize data by group

See Section 3.1 for use case of `group_by()` and `summarize()`.

9.4 Use `mutate()` to calculate new variables

See Section 1.4.3 for use case of `mutate()`

9.5 Use `filter()` to remove sertain values

See Section 1.3.1 for use case of `filter()`

9.6 Use `select()` to include or exclude certain variable

```
RxP.clean %>%
    dplyr::select(-Ind, -Block, -Tank, -Tank.Unique, -Hatch, -Res) %>%
    group_by(Pred) %>%
    summarize_all(mean) #summarize everything at once
```

9.7 Use `full_join()` to join datasets together

The `by=` parameter in the `full_join()` function is used to specify the common column between the two data frames to be joined, which serves as the key to match the data objects together. The value for by= should be the name of the common column in both data frames.

```
#These do the exact same thing.
#Declare your x and y data inside the join
full_join(x = RxP.no4.summary,
            y = RxP.summary,
            by=c('Pred','Res','Hatch'))
```

Another way to achieve the same result:

```
#Declare your x data, then pipe it to the join
RxP.no4.summary %>%
    full_join(RxP.summary,
        by=c('Pred','Res','Hatch'))
```

9.7.1 Different type of joins in R

- Full join: Keeps all columns from both data frames and fills in NA for missing values
- Left join: Keeps all columns from both data frames, but only the rows with a match in x from y
- Right join: Keeps all columns from both data frames, but only the rows with a match in y from x (same as left join but with x and y switched)
- Inner join: Keeps all columns, but only the rows with a match in both data frames

9.8 Use `gather()` to change the data from wide format and converts it into long formate

```
RxP.clean.long <- RxP.clean %>%
    gather(key = Measurement, value = Value,
           -Ind, -Block, -Tank, -Tank.Unique, -Hatch, -Pred, -Res,
           ↪  -before30)
```

Then use the `group_by()` and `summarize()` to get the mean value of all the measurement by groups.

```
print_mean <- RxP.clean.long %>%
    group_by(Measurement, Pred, Res, Hatch) %>%
    summarize(Mean = mean(Value))

head(print_mean)
```

Measurement	Pred	Res	Hatch	Mean
Age.DPO	C	Hi	E	55.34462
Age.DPO	C	Hi	L	61.80226
Age.DPO	C	Lo	E	99.24834
Age.DPO	C	Lo	L	87.85802
Age.DPO	NL	Hi	E	58.42373
Age.DPO	NL	Hi	L	56.76336

9.9 Use `spread()` to transform the long format data into wide format

```
table <- RxP.clean.long %>%
    group_by(Measurement, Pred, Res, Hatch) %>%
    summarize(Mean = mean(Value)) %>%
    spread(key=Measurement, value=Mean)

knitr::kable(table[,c(1:3,5,7,8)],digits = 2)
```

Pred	Res	Hatch	Age.FromEmergence	log.SVL.final	Mass.final
C	Hi	E	21.34	2.92	0.38
C	Hi	L	27.80	2.93	0.39
C	Lo	E	65.25	2.89	0.35
C	Lo	L	53.86	2.90	0.35
NL	Hi	E	24.42	2.97	0.44
NL	Hi	L	22.76	2.98	0.44
NL	Lo	E	45.03	2.94	0.39
NL	Lo	L	62.42	2.91	0.36
L	Hi	E	14.87	3.00	0.49
L	Hi	L	13.55	2.99	0.46
L	Lo	E	25.51	2.98	0.45
L	Lo	L	27.40	2.96	0.42

9.10 Use `do()` to do statistical analysis by group

> **Note**
>
> data frame tidier are deprecated, gather is about to be deprecated and `do` is seldomly used these days. Use `nest()`, and `pivot_longer/wider()` instead.

```
library(broom)

#Whoa, this does some amazing stuff
RxP.models <- RxP.byTank %>%
    gather(key = Measurement, value = Value,
           -Block, -Tank.Unique, -Hatch, -Pred, -Res, -ResPred,
           ↪  -Block.factor) %>%
    group_by(Measurement) %>%
```

```
    do(mod.fit = Anova(lm(Block~Pred*Res*Hatch, data=.,
                          na.action=na.exclude)) %>% tidy(mod.fit) %>%
                          ↪  filter(p.value < 0.05))
```

10 How to write function

10.1 Use `replicate()` to replicate to run function as many time as you want

Index