An introduction to R markdown

Chris B. Wall – chris.wall@hawaii.edu

10/15/2019

This is an R Markdown!

R markdown is great, and when I say great, **I mean REALLY great.**

When exporting your markdown you can...
(1) embed figures
(2) view tables
(3) export models
(4) archive your data pipeline

You can write code and execute functions in a similar way to normal R script files. However, there are a few distinct differences between R scripts and R markdown scripts. Some of these differences relate to how code is executed. Other differences can be seen in the added benefit of integrating scripts with outputs and other embedded properties.

The benefit of using markdown with colleagues is they can actually see the script, the figures, the path of model selection, even the assumptions of ANOVA (QQ plots, etc) that normally are hidden behind the curtain. In short, **R markdown helps make your science repeatable, sharable, and data analysis coherent!**

Here are some common commands and examples of what Markdown can do for you. I reccommend visiting the [R Bookdown for a complete and exhaustive guide](https://bookdown.org/yihui/rmarkdown/ or the R markdown cheat sheet

If you want to test yourself, the R Markdown tutorial is another great guide.

Markdown basics

First: Git with R Projects

My first advise to you would be to set up a GitHub account. Having a GitHub allows you to have all your code, markdown, figures, data, etc archived online for safety and version control. You can make these repositories private or collaborative. It is easy! There are a lot of resources out there on why/how to use GitHub, and you'll see why it is important as you advance your carpentry–you can read about how painless this is.

Whether you choose to GIT with it, or stick with running from your device, you should start by making an *R project* in R-Studio. You can read all about how to do this at happygitwithR.com.

In R Studio, clicking the option to make an R project with version control will set you on the path to linking with GitHub (if you go this route, see happygitwithR.com first, as you must set up a github account, then make a repository, then use the repository URL to link to your R project).

If you decide to instead without version control, then set up the project wherever you wish on your computer. When you set your "R Project" directory, R Studio will generate a folder that houses your R Project. From here you can create your Rmd file and folder directory as you wish. For best practices, I recomment setting up a directory for each project (*see directory for this project below*). From here, everytime you open want to work on this project in R studio, click on the Project. This will allow you to load data in a simple way that anyone can understand (i.e., upload file "symbiont.csv", from folder "data")–and it is easy to share this structure with collaborators.

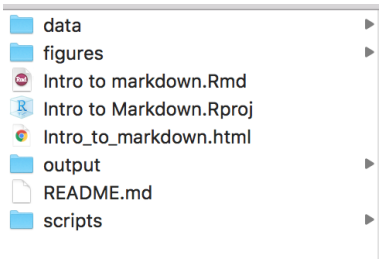


Figure 1: **Figure 1. Data Directory example**

Yet another reason to use R Projects: it will eliminate the pesky and sometimes overly personal directories we all have on our desktop aka: "~/Desktop/PhD/WTFproject/NeverGoingToGetPublished/data.csv"

Common Commands

Notice the script at the very top of your markdown (this is not standard, but is customized)

---

output:
html_document:
code_folding: hide
toc: yes
toc_depth: 3
toc_float: yes ***

Let's go through this line by line...

- `html_document`: an option you originally set when you open the markdown. This is how your file will be exported.
- `code_folding: hide` will allow you to *show* or *hide* all code (option at top of file)
- `toc`: for table of contents on the left
- `toc_depth`: how many header levels you will have in table of contents
- `toc_float`: lets your contents move as you advance in your document.

Play around with these options, and see how they affect your code

There are other options too, such as `number_sections: true` to add numbering to your sections.

The setup chunk

- First, if you want to make a new chunk (where code is written), use the shortcut! `Control + Option + I"` will generate a new chunk for you
- `r setup`: This is your set up code and is a useful way to get around the fact that knitr will look for your .Rmd file and all files in the this directory. By setting the root.dir you can force knitr to look for files in the directory you specify and the folders within.

In the setup chunk (the first code chunk) you can set 'global' options, such as message/warning exclusion, or hiding results or outputs.
`knitr::opts_chunk$set(warning=FALSE, message=FALSE)`: This command here is requiring the package `knitr` in the code chunk ('set' for set up) and is saying to "make all warning and message FALSE" i.e., hide them from output.

- `include=FALSE`: this command makes your code absent from final html. It is executed but the code is hidden.
- `eval=FALSE`: this command allows you to show the code in your script within R studio, but not evaluate (or run) the code. It is effectively silent in your analysis and output html.
- `echo=FALSE`: this will show the output but not the code chunk... notice the difference.
- `results='hide'`: this will hide all results in a code chunk, or any returned results from your commands.

Text formatting

It is useful to understand how to modify text and format headings. You can do this in a variety of ways.

- `line break`: this is executed by two spaces at the end of previous line, and a return
- `headings`: use `#` for headings, `#`.... `####` and so on. # is largest and #### smallest heading

`Italics` comes from two calls *italics* or *italics*; `bold` is done the same way **bold** and **bold**

- `Superscript` superscript^2 or `strikethroughs` ~~strikethrough~~ can also be useful text modifiers.
- `add links` to urls like this R studio link
- `endash` : –
- `emdash`: —
- `ellipsis`: ...
- `inline equation`: $A = \pi * r^2$ or $y = a * x + b$
- (a line can inserted with `***`)
- using the `>` before and after a line can give you a quote/emphasis, such as...

  "Its Van Halen, not Van Hagar!"
  - Garth Algar

Code chunks

Code chunks are where your code is executed. If you do not set the working directory in *setup* each code chunk will revert to its original directory, or where your .Rmd file lives. This is one more reason why you should run everything out of your R Project.

```
## love your data and it will love you back
getwd() #where are your files? See how they should all be running from your R project in the directory
```

```
## [1] "/Users/chriswall/Desktop/Research and Teaching/github/Intro to Markdown"
```

Below: This is a code chunk, and this is how you enter your data into R markdown. Note the `code chunks` always start with a `` ```{r...} `` and ends with a `{...}`. You can add these chunks with the shortcut `Control + Option + I`".

The code chunk below is an example of attaching the data (.csv), familiar to you R-heads. Since the data file is in the directory specified by `knitr::opts_knit$set(root.dir =...` above, you can reference the .csv easily.

```
################################################
# import data, observe structure
################################################
```

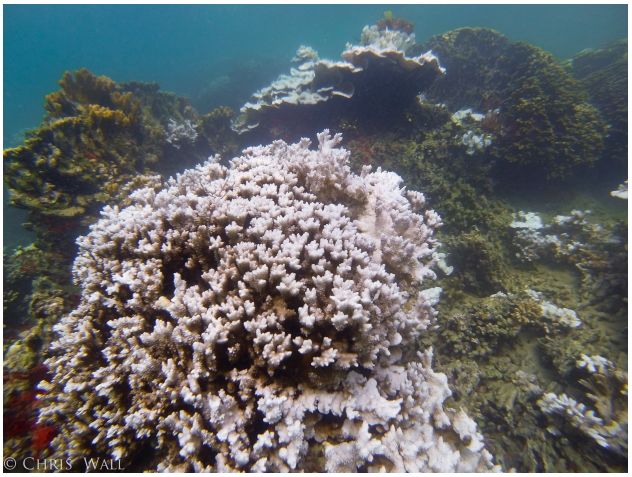


Figure 2: **Figure 2. This is a figure of a bleached *Montipora capitata* coral during the 2014 bleaching event in Kāne'ohe Bay, O'ahu, Hawai'i. (PC: CB Wall)**

Figure 3: **Figure 3. Pa'āni is *beautiful* (and sassy). Beware!**

```
# data file is in the folder 'data', within main working directory
data<-read.csv("data/coral_data.csv")
names(data)
```

```
##  [1] "Time.point" "Period"     "Site"       "Species"    "Status"
##  [6] "Sample.ID"  "Pair"       "Depth.ft"   "biomass"    "chla"
```

Including figures

You can include figures from script output as results, or figures from files in your directory. First, let's see how you can add an image from a file to your markdown.

Adding figures this way can help you present your results for GIS maps that may have been rendered outside of R, or imagery that you want to include of your study organisms, etc.

Note in the coding above I used the html codes for centering the image and the caption. You must leave a line between the first html call `<center>`, the caption and image code, and the end html code line of `</center>`.

Figure formatting

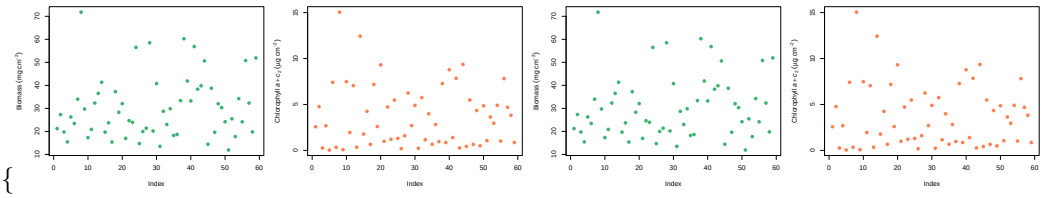
- `fig.show=FALSE`: this will hide the figures you generate. This may be useful for some of those assumptions of ANOVA– you want to see it, but do you want all of them in your final archived datafile? Maybe not.
- `fig.width=5, fig.height=3`: The (graphical device) size of R plots in inches. This records your output figure as a graphical device using *knitr* and will write this to files. There are other ways to edit figure size from this intial dimension (*see below*). You can also specify the two (width and height using) `fig.dim`, as in the previous examle `fig.dim = c(5, 3)`
- `out.width` and `out.height`: These set the output of a figure in output document. This is best to scale the image realtive to document dimensions, such as `out.width= "50%"` would be 50% of page width.
- `fig.align`: gives you figure alignment as either: right, center, left.
- `dev`: graphical devices, typically `png` (html) or `pdf` for LaTex.
- `fig.cap`: caption your figure
- `fig.show`: options here control your figures. Setting to `'asis'` will show plots in the location they were generated (similar to R terminal–this is the default). If you set to `'hide'` then the figures are generated but not shown in your output file. Another option below is my favorite.
- `fig.show='hold'`: this is a really cool option. This option holds you plots and doesn't display until the end of the code chunk. You can use it to place multiple figures side-by-side as long as the `out.width` is called. For example, two plots side-by-side at `out.widh="50%"`

Now let's use some of these options to see how this code could be executed.

```
data<-read.csv("data/coral_data.csv")
par(mar = c(4, 6, 0.2, 0.1))
plot(data$biomass, pch=16, cex=1.2, col="mediumseagreen",
     ylab=expression(paste("Biomass", ~(mg~cm^-2), sep="")))
plot(data$chla, pch=16, cex=1.2, col="coral",
      ylab=expression(paste("Chlorophyll", ~italic("a"+"c"[2]), ~(mu*g~cm^-2), sep="")))
plot(data$biomass, pch=16, cex=1.2, col="mediumseagreen",
     ylab=expression(paste("Biomass", ~(mg~cm^-2), sep="")))
plot(data$chla, pch=16, cex=1.2, col="coral",
      ylab=expression(paste("Chlorophyll", ~italic("a"+"c"[2]), ~(mu*g~cm^-2), sep="")))
```

\begin{figure}

{

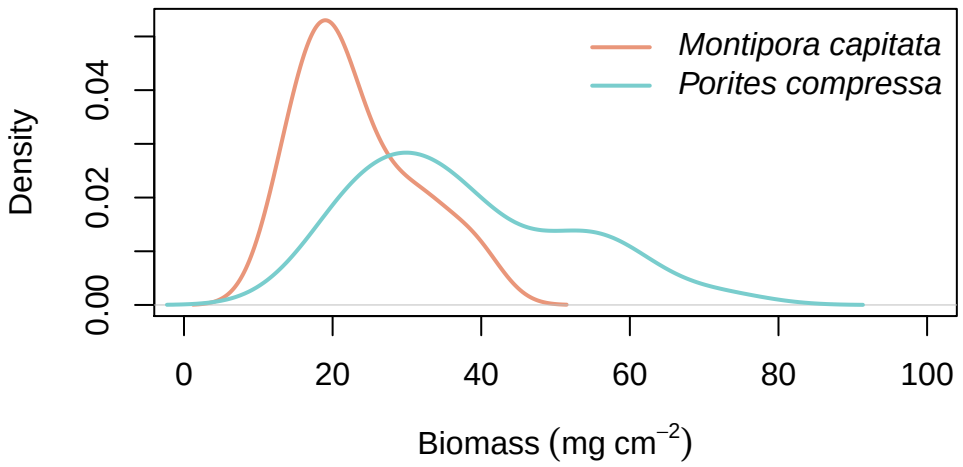


}

\caption{**Figure 4. Example of side-by-side plots at 50%, with a 'fig hold' option**} \end{figure}

But something to consider... R scripts in code chunks operate a bit differently than classic R scripts. If you want to run your plot commands line-by-line you may need to include calls, such as `with()` or `par(new=T)` to specify that you do NOT want a new plot device to be made, but instead to keep working on the device you have opened. So to execute the whole chunk hit the green arrow in the right of the chunk, or highlight *all* the lines you want to run and execute them this way.

Figure example. Tissue biomass in two coral species




```
# use results="hide" here to suppress messages associate with "dev.off" and dev.print to save figure

MC.df<-data[(data$Species=="MC"),] ## Montipora capitata alone
PC.df<-data[(data$Species=="PC"),] ## Porites compress alone

par(mar = c(4, 4, 3, 0.1))

plot(density(MC.df$biomass), lwd=2, col="darksalmon", xlim=c(0,100), main="Figure example. Tissue biomas
lines(density(PC.df$biomass), lwd=2, col="darkslategray3", yaxt="n", ylab="", xaxt="n", xlab="")
legend("topright", legend=c(expression(italic("Montipora capitata")), expression(italic("Porites compres
```

```
# how to export the figure you made--first print it, then save the device
# many ways to do this, but this one works well in Markdown
# notice you are specifying the folder ('figures'), which is where the figure will be exported to

dev.print(pdf, "figures/biomass in two species.pdf", encod="MacRoman", height=4, width=6) # export in d
dev.off()
```

Now let's play!

In this chunk we will make use of some data of photopigments (Chlorophyll *a* concentrations cm^{-2}) in bleached and pigmented corals. Some of the syntax here should be familiar...

```
# message = FALSE to hide any messages with exporting the figure with dev.off(), or anything else that

par(mfrow=c(1,1), pty="sq")
Status.label<-c("Bleached", "Pigmented") # for x labels

plot(chla~Status, data=data, xaxt="n", col="salmon",
     ylab=expression(paste("Chlorophyll", ~italic("a"), ~(mu*g~cm^-2), sep="")),
     xlab="Coral Physiological State",
     main="Example Figure")
axis(1, at=1:2, labels=Status.label) # this plots new labels on the x axis (axis = 1)
```

```
##### save the figure and export to directory? ####
# in this case, there is a file in my directory names 'figures' where I want plots to go
```

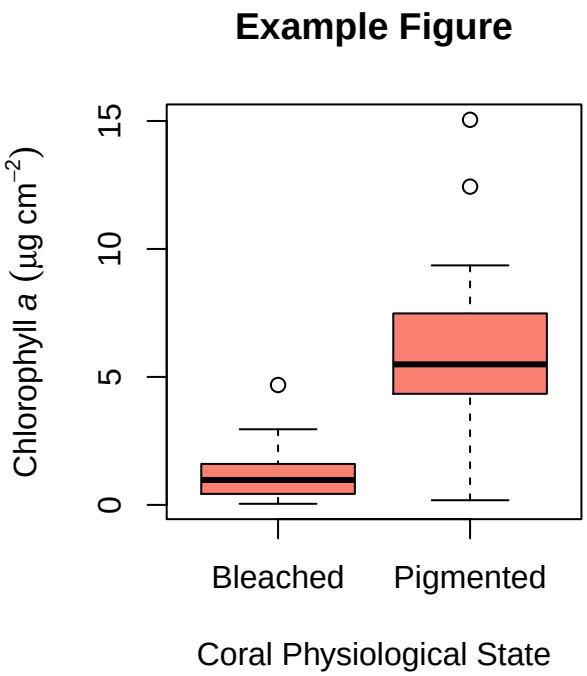


Figure 4: **Figure 5. Chlorophyll a density**. *Note: This caption was set in code chunk setting*

```
dev.print(pdf, "figures/Chlorophyll.figure.pdf", encod="MacRoman", height=5, width=5)
# height and width set here for the output figure
dev.off() # close the object
```

I'm using `dev.copy(pdf, "figures/Chlorophyll.figure.pdf", encod="MacRoman", height=5, width=5)` to print the devise, export it as a pdf, save in the folder names "figures", I'm naming the figure as "Chlorophyll.figure.pdf", and exporting at 5x5 dimensions. Note the code chunk option is specifying to print the figure in the Markdown at a smaller scale (4x4).

Examples

- execute the code and see the returned results

```
chla<-(data$chla)
mean(chla)
```

```
## [1] 3.659065
```

- see a code chunk in my markdown file, but it isn't executed if `eval=FALSE` is set

```
mean(data$chla)
```

- execute the code and hide results `results='hide'`

```
chla<-(data$chla)
mean(chla)
```

- execute and show code and SEE the figure I generate, but hide results (returned data) `results='hide', fig.height=4, fig.width=4, fig.align='center'`

```
par(mfrow=c(1,1))
hist(chla, col="gray85", xlab=expression(paste("Chlorophyll", ~italic("a"), ~(mu*g~cm^-2), sep="")), ce
```

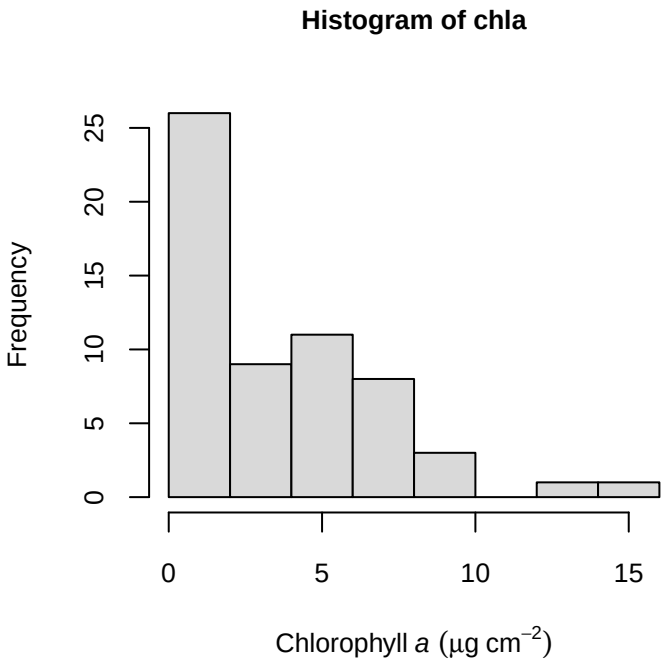


Figure 5: **Figure 6. Boxplot of chlorophyll *a* data**

- make a two boxplots with "bleached" and "non-bleached" corals separated

```
par(mfrow=c(1,2))
# make figure 4 x 4 and center align
hist(data$chla[data$Status=="B"], col="darksalmon", xlab=expression(paste("Chlorophyll", ~italic("a"),
hist(data$chla[data$Status=="NB"], col="darkslategray3", xlab=expression(paste("Chlorophyll", ~italic("
```

... or just print the figure with `echo=FALSE` sans the code

... or show NOTHING with `results='hide', fig.show=FALSE`

```
plot(chla~Site, data=data, col="darkseagreen1")
```

Make a table of summary data

plot raw data 'as is'

```
knitr::kable(data[c(1:5), c(1:10)], digits = c(0, 0, 0, 0, 0, 0, 0, 3, 3, 3))
```

Time.point	Period	Site	Species	Status	Sample.ID	Pair	Depth.ft	biomass	chla
2014 Oct	Bleaching	HIMB	MC	B	3	2	0.667	21.223	2.572
2014 Oct	Bleaching	HIMB	MC	NB	4	2	0.667	27.318	4.766
2014 Oct	Bleaching	HIMB	MC	B	5	3	1.000	19.746	0.259
2014 Oct	Bleaching	HIMB	MC	NB	6	3	1.000	15.449	2.679
2014 Oct	Bleaching	HIMB	PC	B	9	5	1.667	26.273	0.043

```
# the digits call here is specifying the # of decimals for each column
```

Or make summary data and include this table in your markdown

```
chl.mean<-aggregate(chla~Time.point + Site + Species + Status, data, mean)
biomass.mean<-aggregate(biomass~Time.point + Site + Species + Status, data, mean)
data.table<-cbind(chl.mean[, c(1:5)], biomass.mean[,5])
colnames(data.table)<-c("Time Point", "Site", "Species", "Status", "mean chla", "mean AFDW")
knitr::kable(data.table, digits = c(0,0,0,0, 3, 3))
```

Time Point	Site	Species	Status	mean chla	mean AFDW

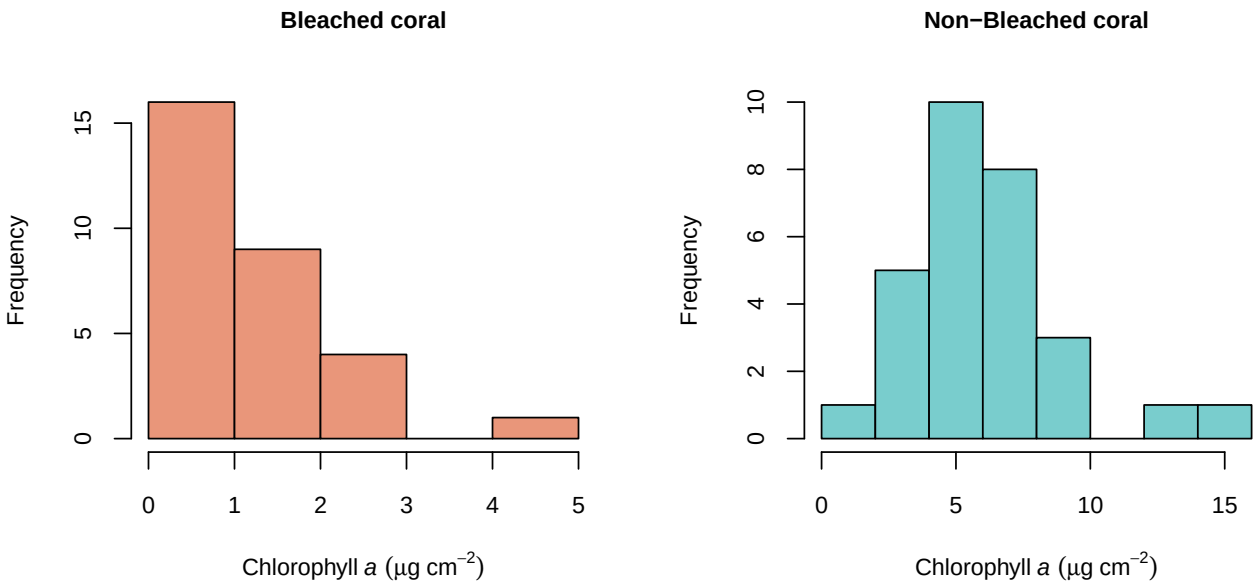


Figure 6: **Figure 7. Boxplot of chlorophyll *a* data by physiological state**

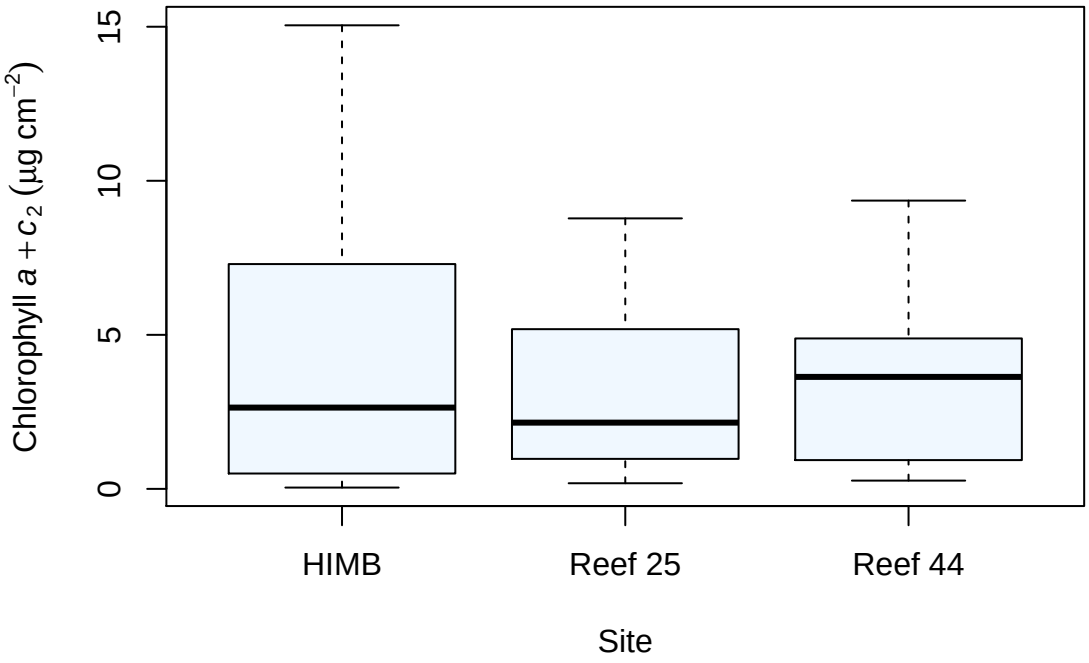


Figure 7: **Figure 8. Separate boxplots of chlorophyll *a* among three reef sites**

```
  summarise(mean.chla=mean(chla), mean.AFDW= mean(biomass),
            SE.chla=std.error(chla), SE.AFDW=std.error(biomass))

# knitr::kable(data.table2, digits = c(0,0,0, 3, 3, 3, 3), col.names = c("Time", "Site", "Species", "ch
```

Running models

Markdown makes running models easy. You can leave all candidate models in the script or you can only report final models. In either case, it is an easy way to keep your data from analysis easy to understand and to QA/QC before publication

LME chlorophyll model

Load packages and look at structure of dataframe: notice I am hiding these results with the `results="hide"` option in the code chunk

```
str(data)
data$Sample.ID<-as.factor(data$Sample.ID) # change 'Sample.ID' to a factor
```

Fixed and random effects

Run a linear mixed effect model, see the anova output, random effects, and plot effects

```
mod<-lmer(chla~Species+Site*Status+(1|Pair), data=data)
anova(mod, type=2) # fixed effects
```

```
## Type II Analysis of Variance Table with Satterthwaite's method
##             Sum Sq Mean Sq NumDF DenDF F value    Pr(>F)
## Species      30.28   30.28     1    52  7.4830  0.008499 **
## Site         17.64    8.82     2    52  2.1798  0.123291
## Status      371.25  371.25     1    52 91.7299 4.516e-13 ***
## Site:Status  22.38   11.19     2    52  2.7652  0.072238 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
ranova(mod) # random effects
```

```
## ANOVA-like table for random-effects: Single term deletions
##
## Model:
## chla ~ Species + Site + Status + (1 | Pair) + Site:Status
##            npar  logLik    AIC LRT Df Pr(>Chisq)
## <none>        9 -118.33 254.67
## (1 | Pair)    8 -118.33 252.67   0  1          1
```

```
plot(allEffects(mod), ylab="total chlorophyll/cm2", par.strip.text=list(cex=0.6))
```

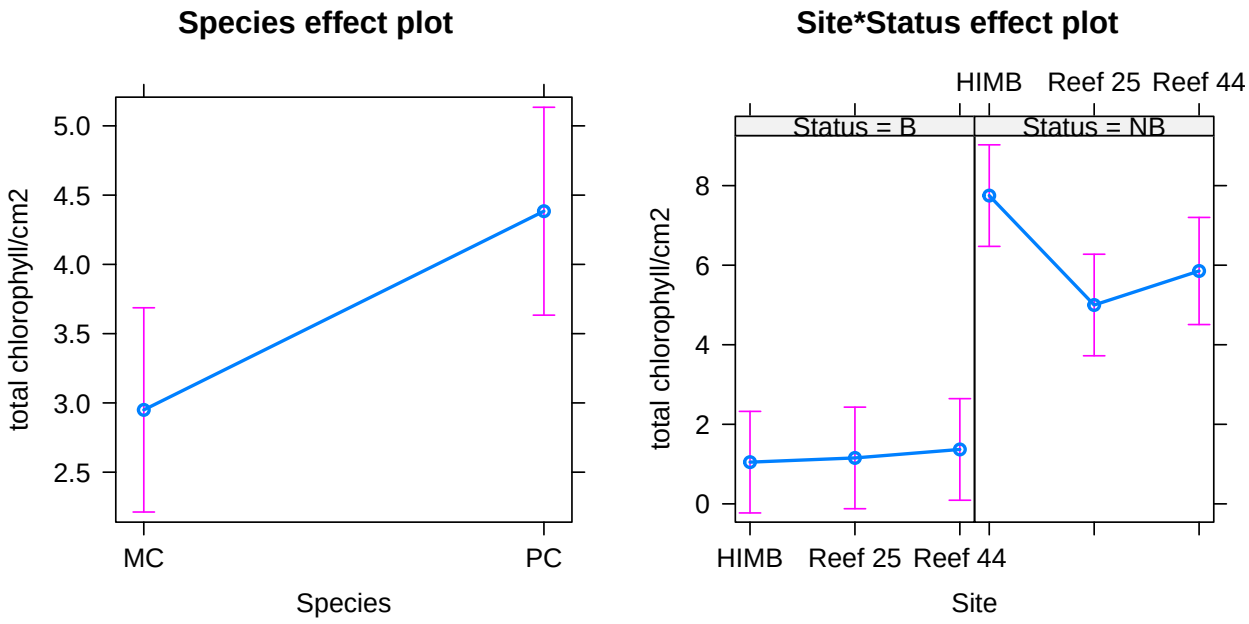


Figure 8: **Figure 9. Model output fixed effect plots**