3. Data structures and Computing in R

Z620: Quantitative Biodiversity, Indiana University

OVERVIEW

This handout builds from the pre-class introduction to R and RStudio, which highlighted operators, vectors, and simple commands that will be useful to you during the course and beyond. Now, we introduce some new data types, specifically **matrices** and **dataframes**. After working with these data types, you will learn about contributed **packages**, which will aid in the visualization of data and with performing descriptive and hypothesis-testing statistical procedures, including **linear regression** and **analysis of variance (ANOVA)**.

1) SETTING YOUR WORKING DIRECTORY

A good first step when you sit down to work in R is to clear your working directory of any variables from your workspace:

```
rm(list = ls())
```

Now we want to set the working directory. This is where your R script and output will be saved. It is also a logical place to put data files that you plan to import into R. The following command will return the exact location of where RStudio is currently working from, which is called your current working directory. This is important because RStudio imports and analyzes data based on its current location (i.e., the working directory) at the time any command is submitted.

```
getwd()
```

If you are working from Posit.cloud, the output from `getwd()` might look this this:

`"/cloud/project/QB-2025/Week1-RStudio"`

This output is known as a **filepath**. Each directory (folder) or file is separated by slash marks (/) and can be read left to right, indicating that the current file or folder is found within the previous entry. For example, `"/cloud/project/QB-2025/Week1-Rstudio/"`, tells you that the folder `Week1` is inside the folder `QB-2025`, which is inside the folder `project`, which is inside the folder `cloud`. The first folder in a file path is often called the **root directory**, as it is the starting point, of the path. In this example, "cloud" would be the root.

When working in R, you will need to change the working directory, for example when switching between project. To do so, you can use the function `setwd()` When you set your working directory, you have the ability to change the **root** directory. In other words, RStudio now thinks of all files and folders in relation to this location. The working directory is the new center of R's world.

Use the following command "setwd()" to set your directory to the same location as the file you are currently working in, that is, 3.RStudio.Rmd. If you are working in Posit.cloud, you should be able to type the following into the the console (without the # sign).

```
#setwd("/cloud/project/QB-2025//Week1-RStudio")
```

If you are working on your personal computer without Posit.cloud, you might change your working directory to something that looks like the following, which would reflect where your GitHub files are being locally kept:

```
setwd("~/GitHub/QB-2025/1.HandOuts/3.RStudio")
```

2) WORKING WITH MATRICES

Matrices are another data type in R. They are just two-dimensional vectors containing data of the same type (e.g., numeric, integer, character). Therefore, much of what we discussed in the pre-class exercise (Week 0) about vectors translates directly into dealing with matrices.

A. Making A Matrix

There are three common ways to create a matrix in R.

Approach 1 is to combine (or **concatenate**) two or more vectors. Let us start by creating a one-dimensional vector using a new function `rnorm()`.

```
j <- c(rnorm(100, mean = 50, sd = 25))
```

Create a new vector named `z` also with 100 elements, but with a different mean and standard deviation

```
z <- c(rnorm(100, mean = 500, sd = 150))
```

Now we will use the function `cbind` to create a matrix from the two one-dimensional vectors:

```
k <- cbind(z, j)
```

Use the `help()` function to learn about `cbind()`. Use the `dim()` function to describe the matrix you just created.

```
dim(k)
```

Approach 2 to making a matrix is to use the `matrix()` function along with arguments that specify the number of rows (`nrow`) and columns (`ncol`):

```
l <- matrix(c(2, 4, 3, 1, 5, 7), nrow = 3, ncol = 2, byrow = FALSE)
```

Approach 3 to making a matrix is to import or **load a dataset** from your working directory:

```
m <- as.matrix(read.table("data/matrix.txt", sep = "\t", header = FALSE))
```

In this case, we are reading in a tab-delimited file. The name of your file must be in quotes, and you need to specify that it is a tab-delimited file type using the `sep` argument. The `header` argument tells R whether or not the names of the variables are contained in the first line (row); in the current example, they are not.

Often, when handling and manipulating datasets, we want to be able to **transpose** a matrix. This is an easy operation in R that uses the `t()` function:

```
n <- t(m)
```

B. Indexing a Matrix

Frequently, you will need to **index** or retrieve a certain *portion* of a matrix. As with the vector example above, we will use the square brackets to return data from a matrix. Inside the square brackets, there are now two subscripts corresponding to the rows and columns, respectively, of the matrix.

The following code will create a new matrix (`n`) based on the first three rows of matrix (`m`):

```
n <- m[1:3, ]
```

Or maybe you want the first two columns of a matrix instead:

```
n <- m[ ,1:2]
```

Or perhaps you want non-sequential columns of a matrix. How do we do that? It is easy when you understand how to reference data within a matrix, along with the combine function, `c()`:

```
n <- m[, c(1:2, 5)]
```

3) BASIC DATA VISUALIZATION AND STATISTICAL ANALYSIS

A. Load Aquatic Mesocosm Dataset

In the following exercise, we will use a dataset from Lennon et al. (2003) (http://goo.gl/JplHwd), which looked at zooplankton communities along an experimental nutrient gradient in aquatic mesocosms. Inorganic nitrogen and phosphorus were added to 300-L mesocosms for six weeks at three different levels (low, medium, and high), but we also directly measured nutrient concentrations of water samples. So we have **categorical** and **continuous** predictors that we are going to use to help explain variation in zooplankton biomass.

The first thing we are going to do is load the nutrient data. To load that data, we simply extend the file path to the `data` folder within `Week1-RStudio` that contains the desired file:

```
meso <- read.table("data/zoop_nuts.txt", sep = "\t", header = TRUE)
```

Let us use the `str()` function to look at the structure of the data.

```
str(meso)
```

```
## 'data.frame':    24 obs. of  8 variables:
##  $ TANK: int  34 14 23 16 21 5 25 27 30 28 ...
##  $ NUTS: chr  "L" "L" "L" "L" ...
##  $ TP  : num  20.3 25.6 14.2 39.1 20.1 ...
##  $ TN  : num  720 750 610 761 570 ...
##  $ SRP : num  4.02 1.56 4.97 2.89 5.11 4.68 5 0.1 7.9 3.92 ...
##  $ TIN : num  131.6 141.1 107.7 71.3 80.4 ...
##  $ CHLA: num  1.52 4 0.61 0.53 1.44 1.19 0.37 0.72 6.93 0.94 ...
##  $ ZP  : num  1.781 0.409 1.201 3.36 0.733 ...
```

How does this dataset differ from the `m` dataset above? Remember, **matrices** and **vectors** are only comprised of a *single* data type. In contrast, the `meso` dataset has a combination of numeric and character data; note the `Factor`, `int`, and `num` data types generated from `str` function. So, this means we are dealing with a new type of **data structure**: a **data frame**.

Here is a description of the column headers:

- TANK = unique mesocosm identifier
- NUTS = categorical nutrient treatment: "L" = low, "M" = medium, "H" = high
- TP = total phosphorus concentration (µg/L)
- TN = total nitrogen concentration (µg/L)
- SRP = soluble reactive phosphorus concentration (µg/L)
- TIN = total inorganic nutrient concentration (µg/L)
- CHLA = chlorophyll *a* concentration (proxy for algal biomass; µg/L)
- ZP = zooplankton biomass (mg/L)

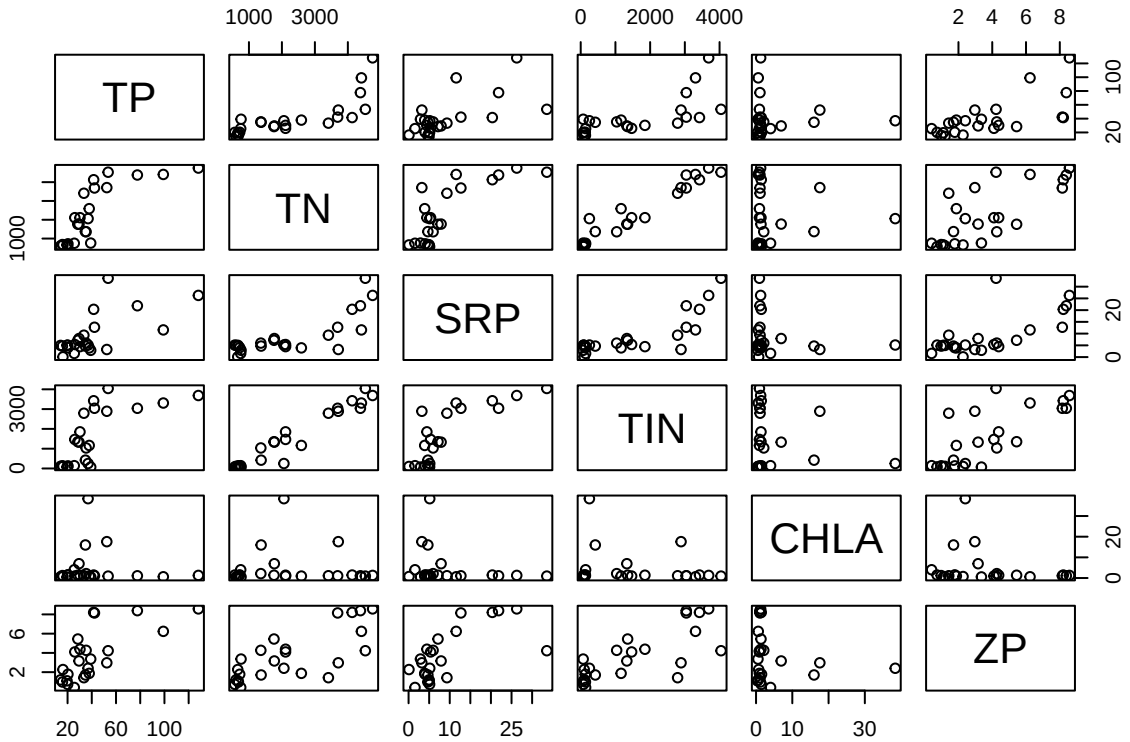
B. Correlation

A common step in data exploration is to look at correlations among variables. Before we do this, let us **index** our numerical (continuous) data in the 'meso' data frame. (Correlations typically do not work well on categorical data.)

```
meso.num <- meso[ ,3:8]
```

We can conveniently visualize pairwise **bi-plots** of the data using the following command:

```
pairs(meso.num)
```



Now let us conduct a simple **Pearson's correlation** analysis with the `cor()` function.

```
cor1 <- cor(meso.num)
```

C. Loading Contributed Packages

The base package in R will not always meet all of our wants and needs. This is why there are > 20,000 **contributed packages** that have been developed for R. This may seem overwhelming, but it also means that there are tools (and web support) for just about any problem you might encounter.

When using one of the contributed packages, the first thing we need to do is **install** them along with their dependencies (i.e., other required packages). We are going to start out by using the `psych` package http://cran.r-project.org/web/packages/psych/vignettes/overview.pdf. The `psych` package has many features, but we are going to use it specifically for the `corr.test()` function. This function generates **P-values** for each pairwise correlation. (For whatever reason, the `cor()` function in the `base` package of R does not generate *P*-values.)

You can load an R package and its dependencies using the `require()` function. If a package is not found, i.e., not installed, you will need to install it first and then load dependencies.

```
#install.packages("psych", repos="http://cran.rstudio.com/")
require("psych")
```

Note that if a package is already installed, you can bypass the install line by placing a '#' at the beginning of it. Now, let us look at the correlations among variables and assess whether they are significant:

```
cor2 <- corr.test(meso.num, method = "pearson", adjust = "BH")
print(cor2, digits = 3)
```

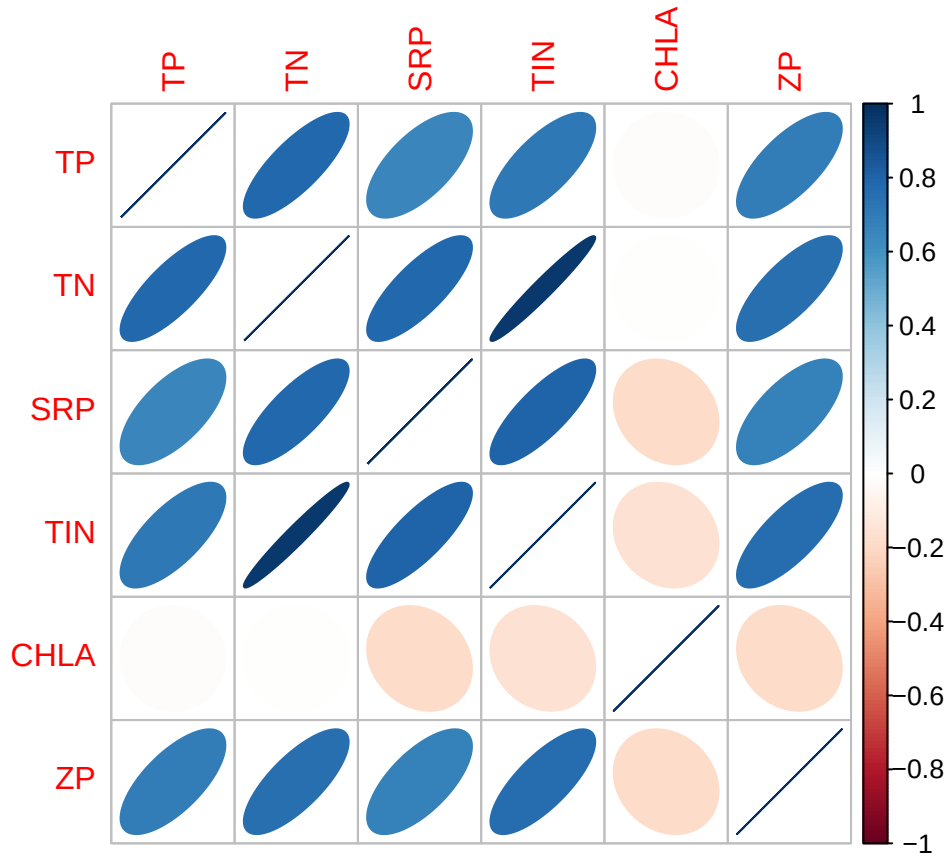
Notes on corr.test:

a) for rank-based correlations (i.e., non-parametric), use method = "kendall" or "spearman". Give it a try!

b) the adjust = "BH" statement supplies the Benjamini & Hochberg-corrected *P*-values in the upper right diagonal of the square matrix; the uncorrected p-values are below the diagonal. This process corrects for **false discovery rate**, which arises when making multiple comparisons. In other words, the probability of detecting a "significant" *P*-value – just by chance – increases with the number of tests that are performed, which is also known as a Type-1 error.

Now, let us load another package that will let us visualize the direction and strength of the correlations:

```
#install.packages("corrplot", repos="http://cran.rstudio.com/")
require("corrplot")
corrplot(cor1, method = "ellipse")
```




```
dev.off()
```

D. Linear Regression

It seems that total nitrogen (TN) is a fairly good predictor of zooplankton biomass (ZP) and this is something that we directly manipulated. This gives us license to conduct a linear regression analysis. We can do this in R using the `lm()` function:

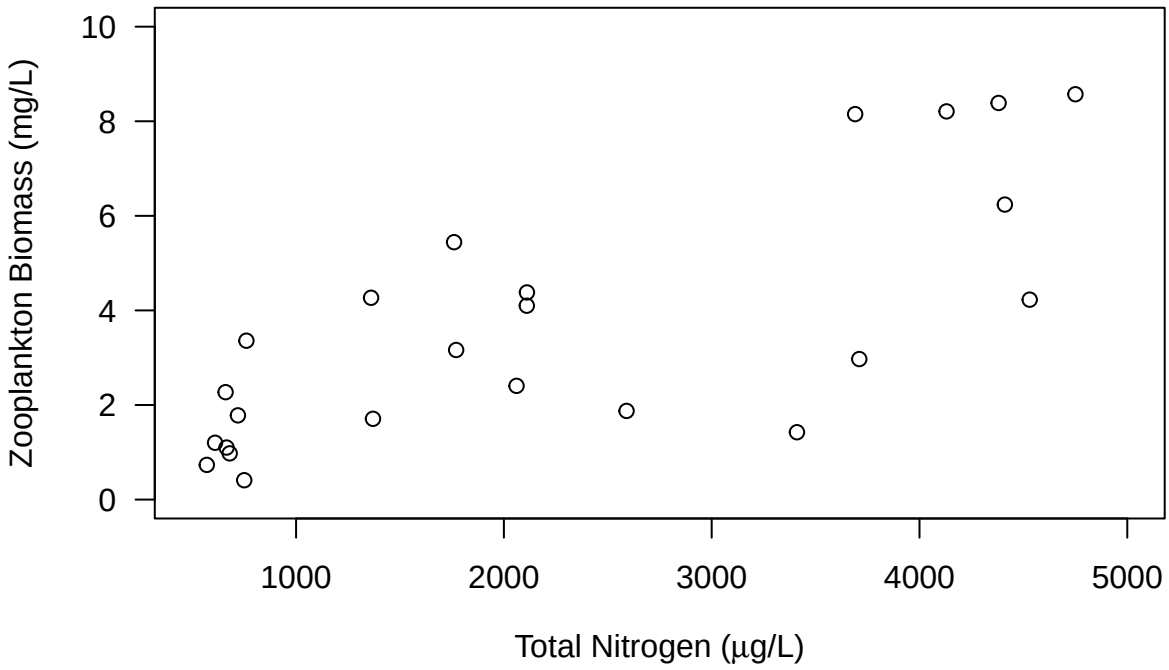
```
fitreg <- lm(ZP ~ TN, data = meso)
```

Let us examine the output of the regression model:

```
summary(fitreg)
```

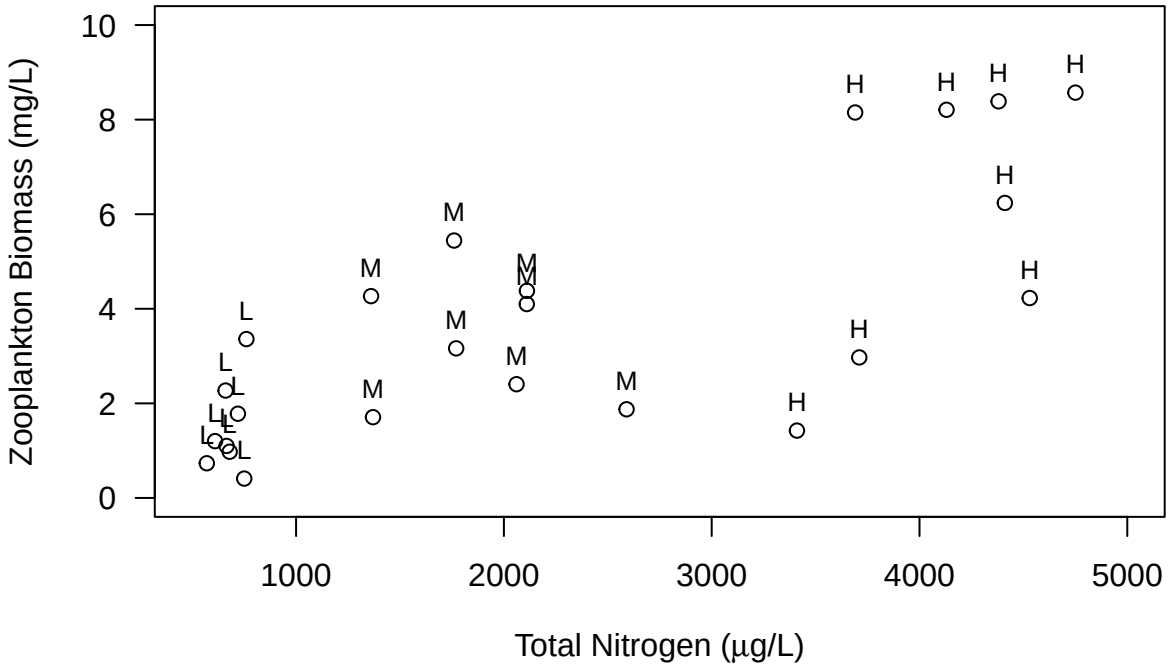
Now, let us look at a plot of the data used in the regression analysis:

```
plot(meso$TN, meso$ZP, ylim = c(0, 10), xlim = c(500, 5000),
     xlab = expression(paste("Total Nitrogen (", mu,"g/L)")),
     ylab = "Zooplankton Biomass (mg/L)", las = 1)
```



We can add some text to the plot to visualize the categorical nutrient treatments:

```
text(meso$TN, meso$ZP, meso$NUTS, pos = 3, cex = 0.8)
```

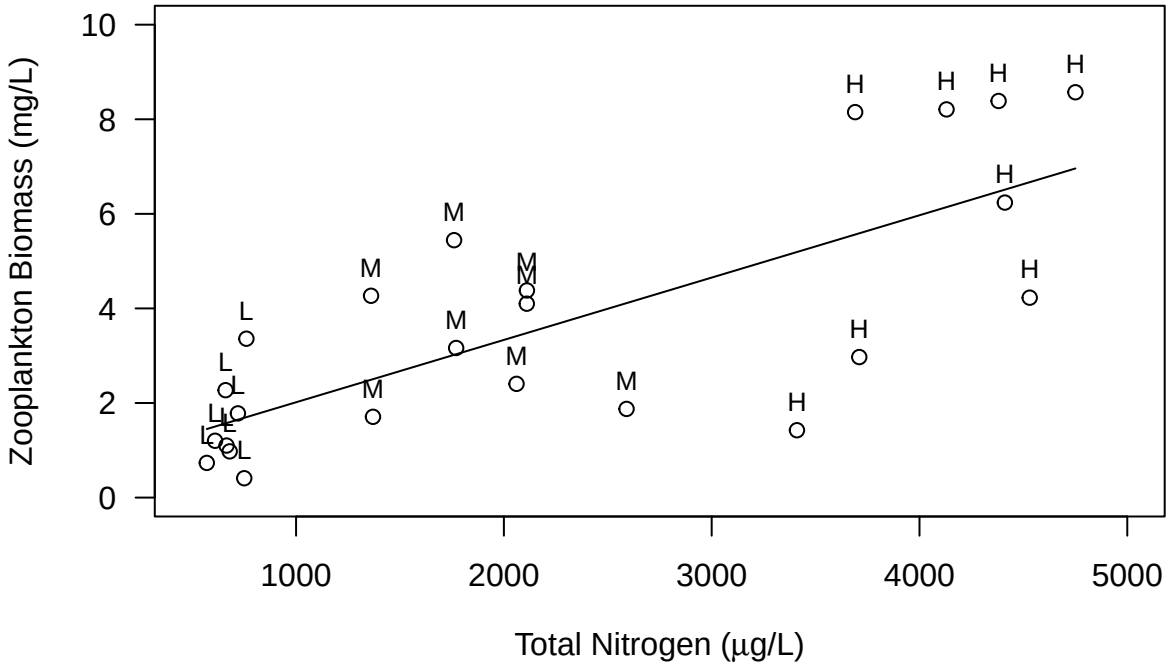


There is somewhat of a steep learning curve for plotting in R. However, R is also very flexible and can make beautiful figures. Perhaps one of the biggest initial obstacles is getting familiar with all of the **plotting arguments**. The following website provides a nice overview http://www.statmethods.net/advgraphs/parameters.html and there are plenty of other websites and books that are useful to have close by. Outside of `base` R, there are packages that aid in plotting. One such package is `ggplot2` https://ggplot2.tidyverse.org/.

It belongs to a group of packages knows as the **Tidyverse** https://www.tidyverse.org/, which includes great tools for "wrangling" data. We will learn more about this in QB in future weeks.

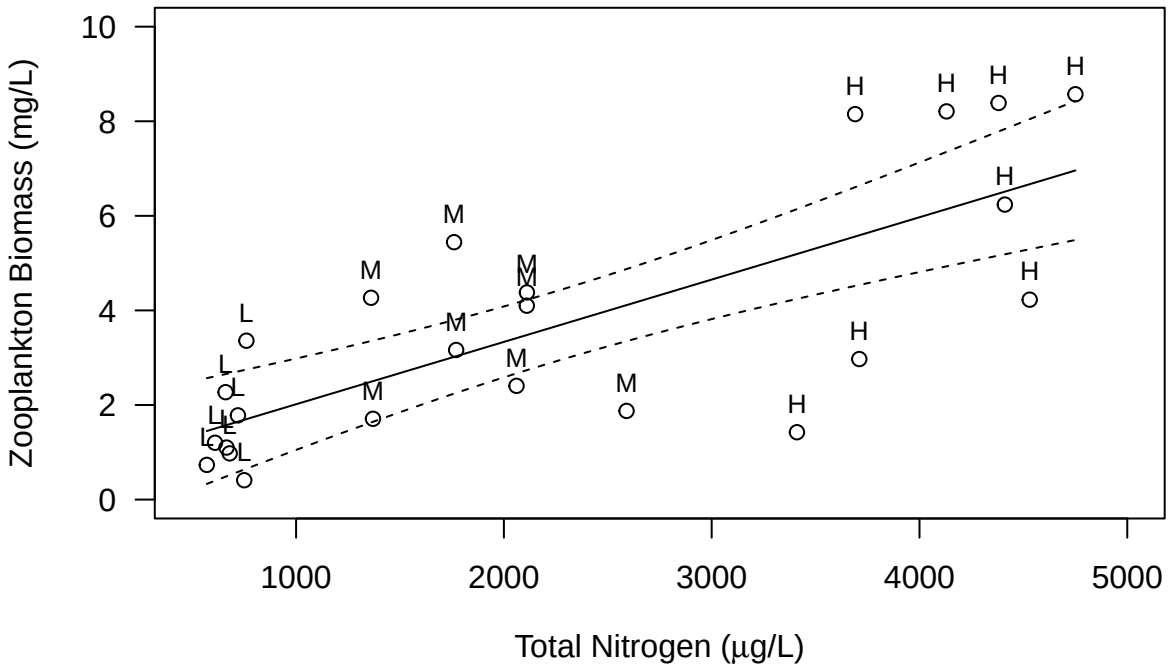
Now, let us continue with our regression analysis. To add the regression line, the first thing we need to do is identify a range of x-values and then generate the corresponding **predicted values** from our regression model:

```
text(meso$TN, meso$ZP, meso$NUTS, pos = 3, cex = 0.8)
newTN <- seq(min(meso$TN), max(meso$TN), 10)
regline <- predict(fitreg, newdata = data.frame(TN = newTN))
lines(newTN, regline)
```



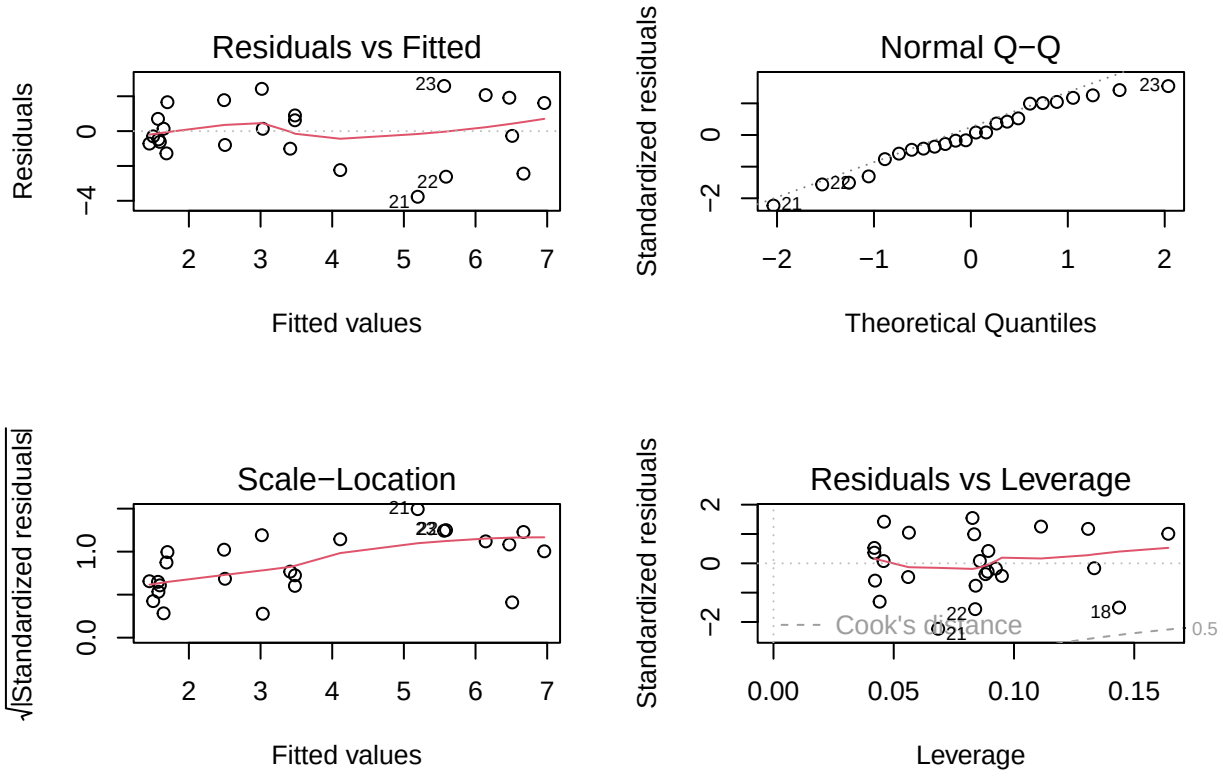
Now let us create and plot the **95% confidence intervals** using the same procedure as above, that is, use `newTN` to generate corresponding confidence intervals from our regression model:

```
text(meso$TN, meso$ZP, meso$NUTS, pos = 3, cex = 0.8)
newTN <- seq(min(meso$TN), max(meso$TN), 10)
regline <- predict(fitreg, newdata = data.frame(TN = newTN))
lines(newTN, regline)
# the line above calls the previous figure object
conf95 <- predict(fitreg, newdata = data.frame(TN = newTN),
                  interval = c("confidence"), level = 0.95, type = "response")
matlines(newTN, conf95[, c("lwr", "upr")], type="l", lty = 2, lwd = 1, col = "black")
```



We should also look at the residuals (i.e., observed values - predicted values) to see if our data meet the assumptions of linear regression. Specifically, we want to make sure that the **residuals** are normally distributed and that they are homoscedastic (i.e., equal variance). We can look for patterns in our residuals using the following diagnostics:

```
par(mfrow = c(2, 2), mar = c(5.1, 4.1, 4.1, 2.1))
plot(fitreg)
```



- Upper left: is there a random distribution of the residuals around zero (horizontal line)?
- Upper right: is there a reasonably linear relationship between standardized residuals and theoretical quantiles? Try `help(qqplot)`
- Bottom left: again, looking for a random distribution of $\sqrt{\text{standardized residuals}}$
- Bottom right: leverage indicates the influence of points; contours correspond with Cook's distance, where values $> |1|$ are "suspicious"

E. Analysis of Variance (ANOVA)

We also have the option of looking at the relationship between zooplankton biomass and nutrients where the manipulation is treated categorically (low, medium, high). First, let us order the categorical nutrient treatments from low to high (R's default is to order alphabetically):

```
NUTS <- factor(meso$NUTS, levels = c('L', 'M', 'H'))
```

Before plotting, we need to calculate the means and standard errors for zooplankton biomass in our nutrient treatments. We are going to use an important function called `tapply`. This allows us to apply a function (e.g., mean) to a vector (e.g., ZP) based on information in another column (e.g., nutrient treatment).

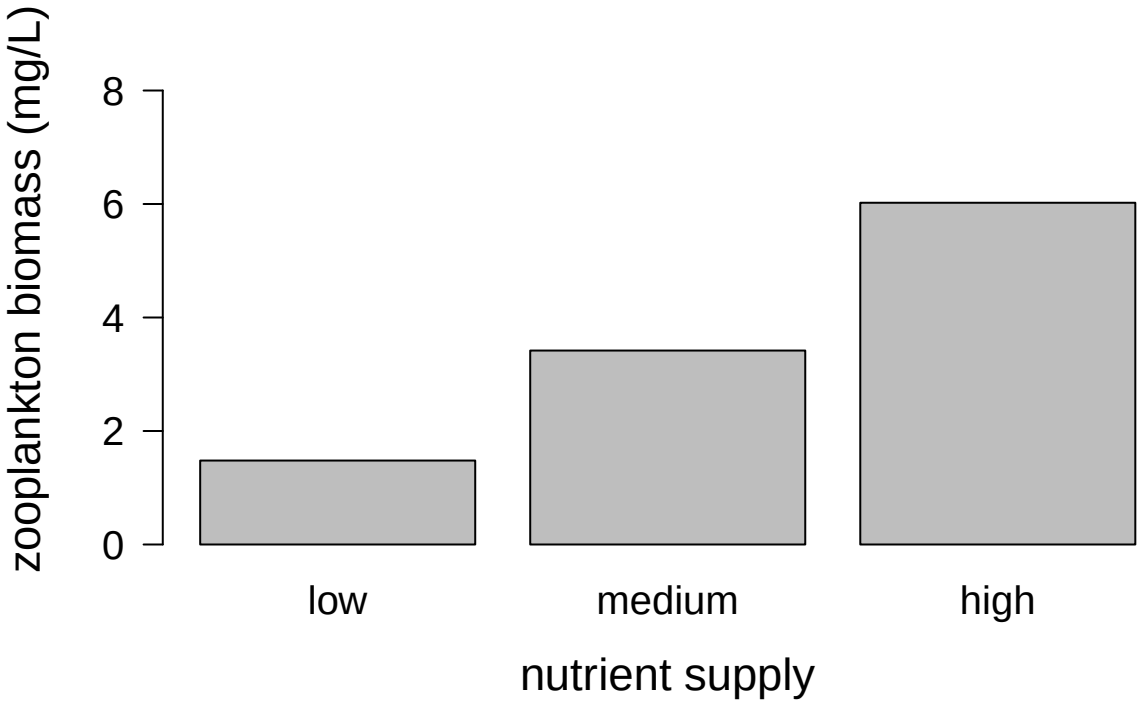
```
zp.means <- tapply(meso$ZP, NUTS, mean)

sem <- function(x){
  sd(na.omit(x))/sqrt(length(na.omit(x)))
  }

zp.sem <- tapply(meso$ZP, NUTS, sem)
```

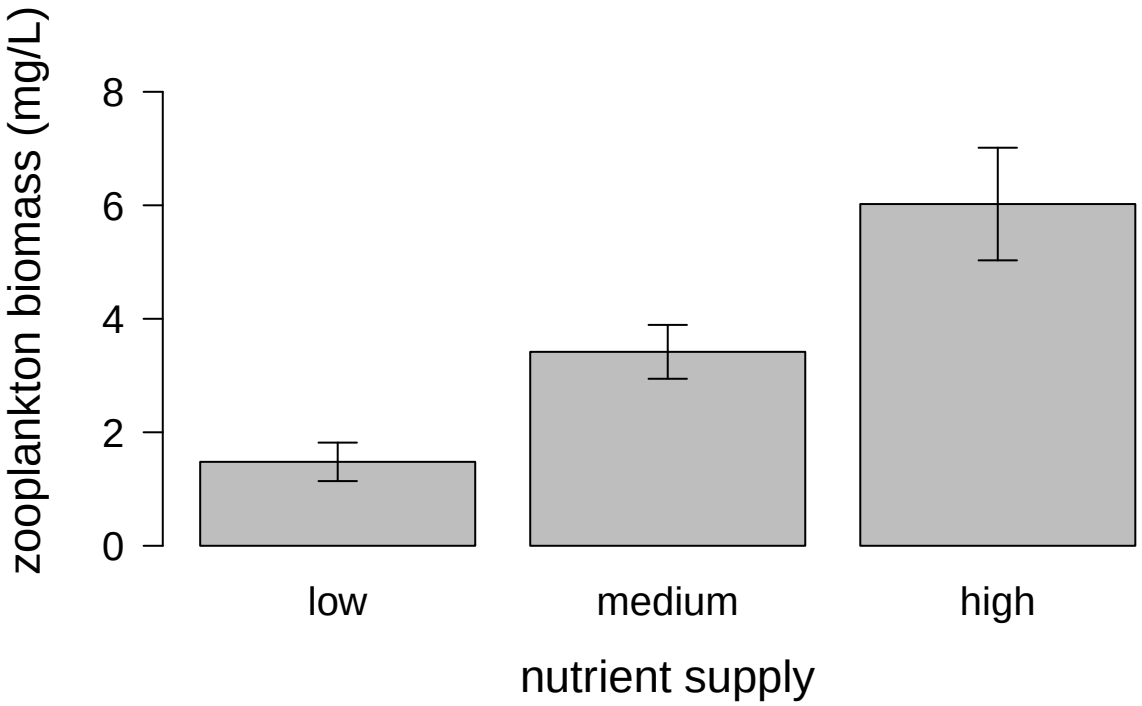
Now let us make the barplot:

```
bp <- barplot(zp.means, ylim =c(0, round(max(meso$ZP), digits = 0)),
              pch = 15, cex = 1.25, las = 1, cex.lab = 1.4, cex.axis = 1.25,
              xlab = "nutrient supply",
              ylab = "zooplankton biomass (mg/L)",
              names.arg = c("low", "medium", "high"))
```



We need to add the error bars (± sem) "manually" as follows:

```
arrows(x0 = bp, y0 = zp.means, y1 = zp.means - zp.sem, angle = 90,
       length = 0.1, lwd = 1)
arrows(x0 = bp, y0 = zp.means, y1 = zp.means + zp.sem, angle = 90,
       length = 0.1, lwd = 1)
```



Last, we can conduct a one-way analysis of variance (ANOVA) as follows:

```
fitanova <- aov(ZP ~ NUTS, data = meso)
```

Let us look at the output in more detail (just as we did with regression):

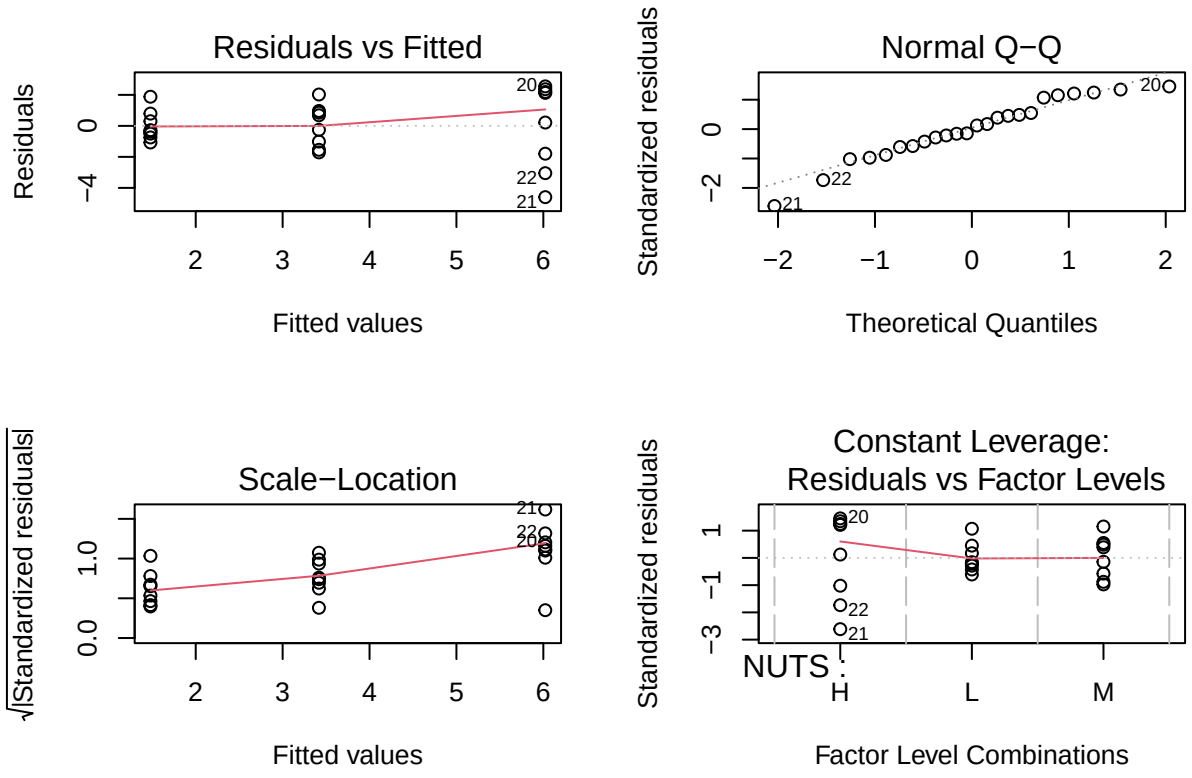
```
summary(fitanova)
```

Finally, we can conduct a post-hoc comparison of treatments using Tukey's HSD (Honest Significant Differences). This will tell us whether or not there are differences (alpha = 0.05) among pairs of the three nutrient treatments

```
TukeyHSD(fitanova)
```

Just like the regression analysis above, it is good to look at the residuals:

```
par(mfrow = c(2, 2), mar = c(5.1, 4.1, 4.1, 2.1))
plot(fitanova)
```



4) AN INTRODUCTION TO THE SITE-BY-SPECIES MATRIX

During QB this semester, you will be introduced to a few **primary data structures** that are critical for analyzing biodiversity data. The first of these is the **site-by-species matrix**. This site-by-species matrix contains abundances, relative abundances, or the presence of species (or other taxonomic units) observed at different sampling sites. The site-by-species matrix is also very similar to other types of data that you might encounter if you are working with genomic, metagenomic, transcriptomic, or metabolomic datasets. This means that, even if you don no care about biodiversity *per se*, the tools that you are learning in QB can be used to address questions in other systems! The table below is an example of what a typical site-by-species matrix looks like:

	Species1	**Species2**	**Species3**	**Species4**	...
Site1	90	76	1	0	
Site3	86	47	0	123	
Site3	23	89	0	46	
...					

Each row in the site-by-species matrix corresponds to a site (or sample), while each column corresponds to a species (or other taxonomic unit). Throughout the course, we will draw inferences about aspects of diversity from the site-by-species matrix.