Week 1 Exercise: Basic R

Z620: Quantitative Biodiversity, Indiana University

January 16, 2015

OVERVIEW

This exercise introduces some of the basic features of the R (http://www.r-project.org/) computing environment. We will briefly cover operators, data types, and simple commands that will be useful for you during the course and beyond. In addition to using R's base package, we will also use contributed packages, which together will allow us to visualize data and peform relatively simple statistics (e.g., linear regression and ANOVA).

1) HOW WE WILL BE USING R AND OTHER TOOLS

During the course, we will useRStudio (http://www.rstudio.com/), which is a user-friendly integrated developement environment (IDE) that allows R to interface with other tools. For example, the document you are reading was generated in R Markdown (http://rmarkdown.rstudio.com). Markdown is a simple formatting syntax for authoring HTML, PDF, and other documents.

We will also use a tool called knitr (http://yihui.name/knitr/), which is a package that generates reports from R script and Markdown text. For example, when you click the **Knit PDF** button in the scripting window of Rstudio, a document will be generated that includes LaTeX (http://www.latex-project.org/) typesetting as well as the output of any embedded R code.

However, if there are errors in the Rcode contained in your markdown document, you will not be able to knit a PDF file. Assignments in this class will require that you successfully create a Markdown-generated PDF using knitr; you will then **push** this document to the course **respository** hosted on IU's GitHub (https://github.iu.edu) and generate a **pull request**.

2) 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:

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

Now we want to set the working directory. This is where your R script and output will be saved. It's also a logical place to put data files that you plan to import into R. The following command will return your current working directory:

```r
getwd()
```

Use the following command to change your directory (but note that you will need to modify to reflect *your* actual directory):

```r
setwd("~/GitHub/QuantitativeBiodiversity/Assignments/Week1")
```

3) USING R AS A CALCULATOR

R is capable of performing various calcuations using simple operators and built-in **functions**

Addition:

```
1 + 3
```

Subtraction:

```
3 - 1
```

Multiplication (with an exponent):

```
3 * 10^2
```

Division (using a built-in constant; pi):

```
10 / pi
```

Trigonometry with a simple built-in function (i.e., *sin*) that takes an **argument** (i.e., '4'):

```
sin(4)
```

Logarithms (another example of functions and arguments)

```
log10(100)  # log base 10
log(100)    # log base e "natural log"
```

4) ASSIGNING VARIABLES

You will often find it useful and necessary to assign values to a **variable**, also known as an **object** in R. Generally speaking, in R, it's best to use `<-` rather than `=` as an assignment operator.

```
a <- 10
b <- a + 20
```

What is the value of `b`?

Now let's reassign a new value to `a`:

```
a <- 200
```

Now, what is the value of `b`? What's going on?

R held onto the original value of `a` that was used when assigning values to `b`. You can correct this using the `rm` function, which removes objects from your R **environment**.

```
rm("b")
```

What happens if we reassign `b` now?

```
b <- a + 20
```

Sometimes it's good practice to clear all variables from your R environment (e.g., you've been working on multiple projects during the day). This can be done in a couple of ways. For example, you can just click `clear` in the **Environment/History pane** window of R Studio. The same procedure can be performed at the **command line** in the Rstudio **Console pane** or **Script Editor pane**. To do this, you can use the `ls` function to view a list of all the objects in the R environment:

```
ls()
```

```
## [1] "a" "b"
```

You can now clear all of the stored variables from R's memory using two functions: `rm` and `ls`. (Note: we did this above prior to setting our working directory).

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

5) WORKING WITH VECTORS

Basic Features Of Vectors

Vectors are the fundamental data type in R. Often, vectors are just a collection of data of a similar type, either numeric (e.g., 17.5), integer (e.g., 2), or character (e.g., "low"). The simplest type of vector is a single value, sometimes referred to as a **scalar** in other programming languages:

```
w <- 5
```

We can create longer one-dimensional vectors in R like this:

```
x <- c(2, 3, 6, w, w + 7, 12, 14)
```

What is the function `c`() that we just used to create a vector? To answer this question, trying typing `help`() function at the command line. Let's try it out:

```
help(c)
```

What happens when you multiply a vector by a "scalar"?

```
y <- w * x
```

What happens when you multiply two vectors of the same length?

```
z <- x * y
```

You may need to reference a specific **element** in a vector. We will do this using the square brackets. In this case, the number inside of the square brackets tells R that we want call the ith element of vector `z`:

```
z[2]
```

```
## [1] 45
```

You can also reference **multiple elements** in a vector using the square brackets and the colon symbol:

```
z[2:5]
```

```
## [1]  45 180 125 720
```

In some instances, you may want to change the value of an element in a vector. Here's how you can substitute a new value for the second element of `z`:

```
z[2] <- 583
```

Summary Statictics of Vectors

It's pretty easy to perform summary statistics on a vector using the built-in fuctions of R:

```
max(z)    # maximum
min(z)    # minimum
sum(z)    # sum
mean(z)   # mean
median(z) # median
var(z)    # variance
sd(z)     # standard deviation
```

What happens when you take the standard error of the mean (`sem`) of `z`?

The standard error of the mean is defined as $\frac{sd(x)}{\sqrt{n}}$. This function does not exist in the base package of R. Therefore, you need to write your own function. Let's give it a try:

```
sem <- function(x){
  sd(x)/sqrt(length(x))
  }
```

There are number of functions inside of `sem`. Take a moment to think about and describe what is going on here. Now, use the `sem` function you just created on the vector `y` from above

Often, datasets have missing values (designated as 'NA' in R):

```
i <- c(2, 3, 9, NA, 120, 33, 7, 44.5)
```

What happens when you apply your `sem` function to vector `i`? This is a problem!

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

Now run `sem` on the vector `i`.

5) 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). Thereofore, much of what we just discussed about vectors translates directly into dealing with matrices.

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's start by creating a one-dimensional vector using a new function `rnorm` based on information contained in vector `z` above.

```
j <- c(rnorm(length(z), mean = z))
```

Q1: What does the `rnorm` function do? What are the arguments specifying? Remember to use `help()` or type `?rnorm`.

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)
```

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're 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; in the current example, they are not.

Often, when handling 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)
```

Q2: What are the dimensions of the matrix you just transposed?

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`):

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

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

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

Or perhaps you want non-sequential columns of a matrix. How do we do that? It's easy when you understand how to reference data within a matrix:

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

Q3: Describe what we just did in the last indexing operation?

6) BASIC DATA VISUALIZATION AND STATISTICAL ANALYSIS

Load Zooplankton 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 mesocosms for six weeks at three different levels (low, medium, and high), but we also direcly measured nutrient concentrations of water samples. So we have **categorical** and **continuous** predictors that we're going to use to help explain variaiton in zooplankton biomass.

The first thing we're going to do is load the data:

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

Let's use the `str` function to look at the structure of the data.

```r
str(meso)
```

```
## 'data.frame':    24 obs. of  8 variables:
##  $ TANK: int  34 14 23 16 21 5 25 27 30 28 ...
##  $ NUTS: Factor w/ 3 levels "H","L","M": 2 2 2 2 2 2 2 2 3 3 ...
##  $ 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? The answer is, we're now dealing with a new type of **data structure**. Specifically, the `meso` dataset is a **data frame** since it has a combination of numeric and character data (i.e., note the `Factor`, `int`, and `num` data types generated from `str` function). (Remember, **matrices** and **vectors** are only comprised of a *single* data type.)

Here is a description of the column headers:

- TANK = unique mesocosm identifier
- NUTS = categorical nutrient treament: "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)

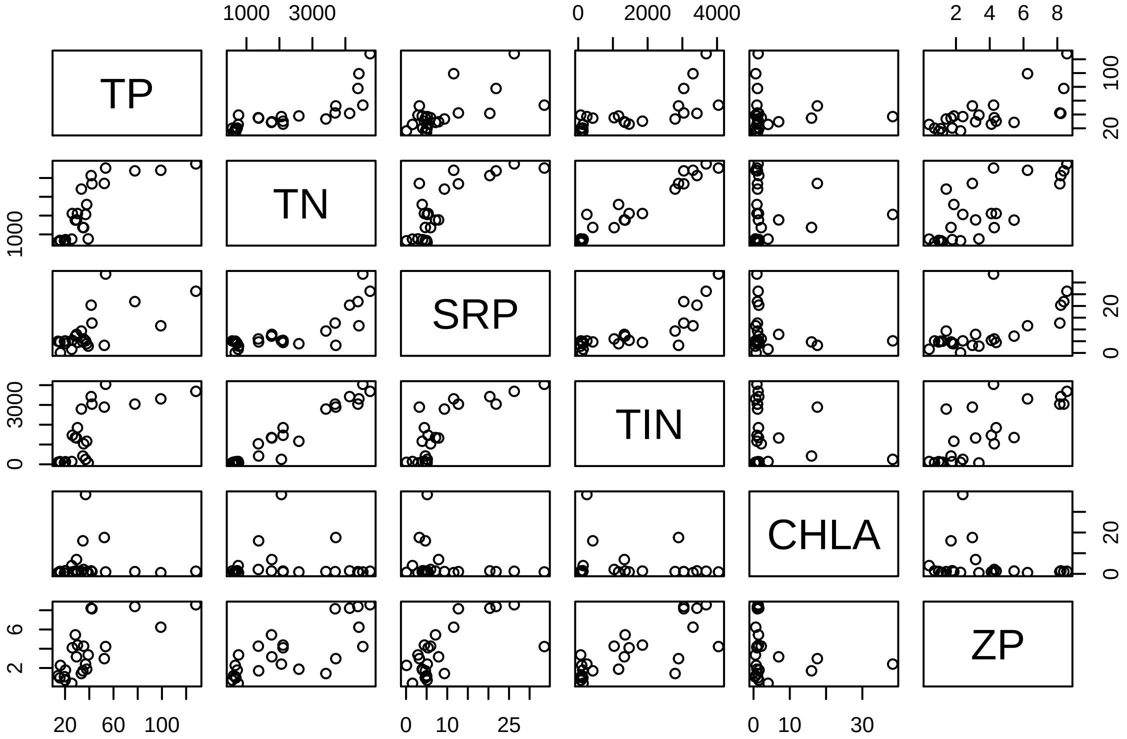
Correlation

A common step in data exploration is to look at correlations among variables. Before we do this, let's **index** our numerical (continuous) data in the 'meso' dataframe. (Correlations typiclly don't 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's conduct a simple **Pearson's correlation** analysis with the `cor()` function.

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

Q4: Describe some of the general features based on the visualization and correlation anaylysis above?

Loading Contributed Packages

The base pakcage in R won't always meet all of our needs. This is why there are > 6,000 **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 can think of.

When using one of the contributed packages, the first thing we need to do is **install** them along with their dependencies (other required packages). We're going to start out by using the *psych* package. The *psych* package has many features, but we're 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:

```
require("psych")
```

```
## Loading required package: psych
```

If the package is not found, you will need to install it first and then load dependencies:

```
install.packages("psych")
require("psych")
```

Now, let's look at the correlations among variables and assess whether they are signficant:

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

```
## Call:corr.test(x = meso.num, method = "pearson", adjust = "BH")
## Correlation matrix
##          TP     TN    SRP    TIN   CHLA     ZP
## TP    1.000  0.787  0.654  0.717 -0.017  0.697
## TN    0.787  1.000  0.784  0.969 -0.004  0.756
## SRP   0.654  0.784  1.000  0.801 -0.189  0.676
## TIN   0.717  0.969  0.801  1.000 -0.157  0.761
## CHLA -0.017 -0.004 -0.189 -0.157  1.000 -0.183
## ZP    0.697  0.756  0.676  0.761 -0.183  1.000
## Sample Size
## [1] 24
## Probability values (Entries above the diagonal are adjusted for multiple tests.)
##         TP    TN   SRP   TIN  CHLA    ZP
## TP   0.000 0.000 0.001 0.000 0.983 0.000
## TN   0.000 0.000 0.000 0.000 0.983 0.000
## SRP  0.001 0.000 0.000 0.000 0.491 0.000
## TIN  0.000 0.000 0.000 0.000 0.536 0.000
## CHLA 0.938 0.983 0.376 0.464 0.000 0.491
## ZP   0.000 0.000 0.000 0.000 0.393 0.000
##
##  To see confidence intervals of the correlations, print with the short=FALSE option
```

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 whe making multiple comparisons.

Q5: Describe what you learned from corr.test. Specifically, are the results sensitive to whether you use parametric (i.e., Pearson's) or non-parametric methods? With the Pearson's method, is thre evidence for false discovery rate due to multiple comparisons?

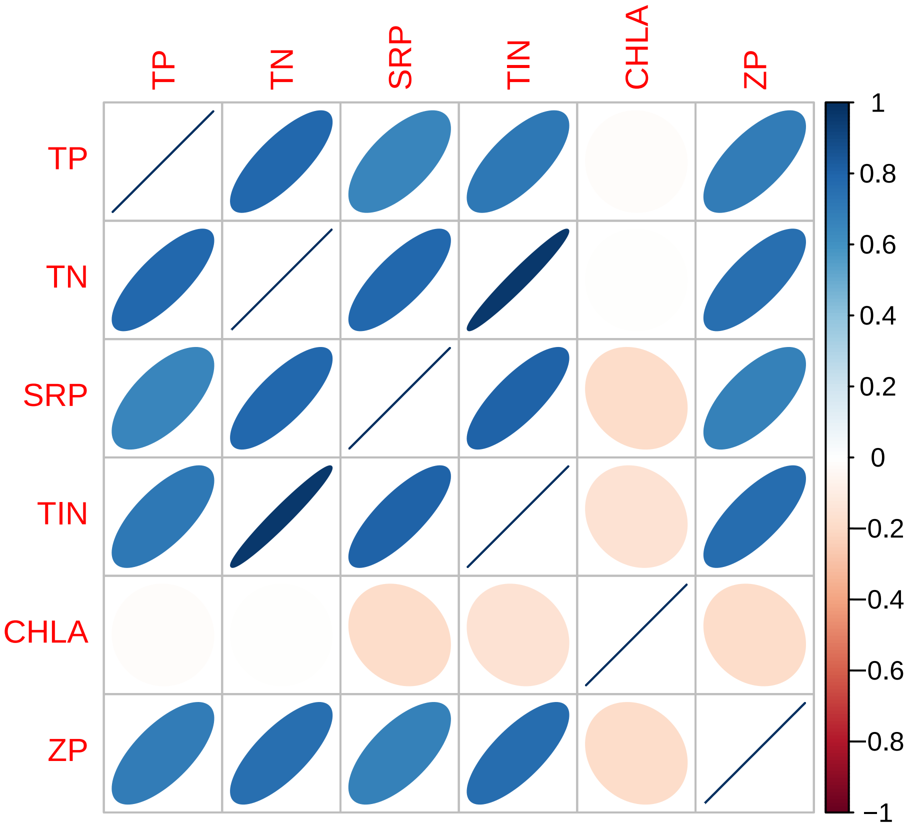
Now, let's load another package that will let us visualize the sign and strength of the correlations:

```
require("corrplot")||install.packages("corrplot");require("corrplot")
```

```
## Loading required package: corrplot
```

```
## [1] TRUE
```

```
corrplot(cor1, method = "ellipse")
```



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)
```

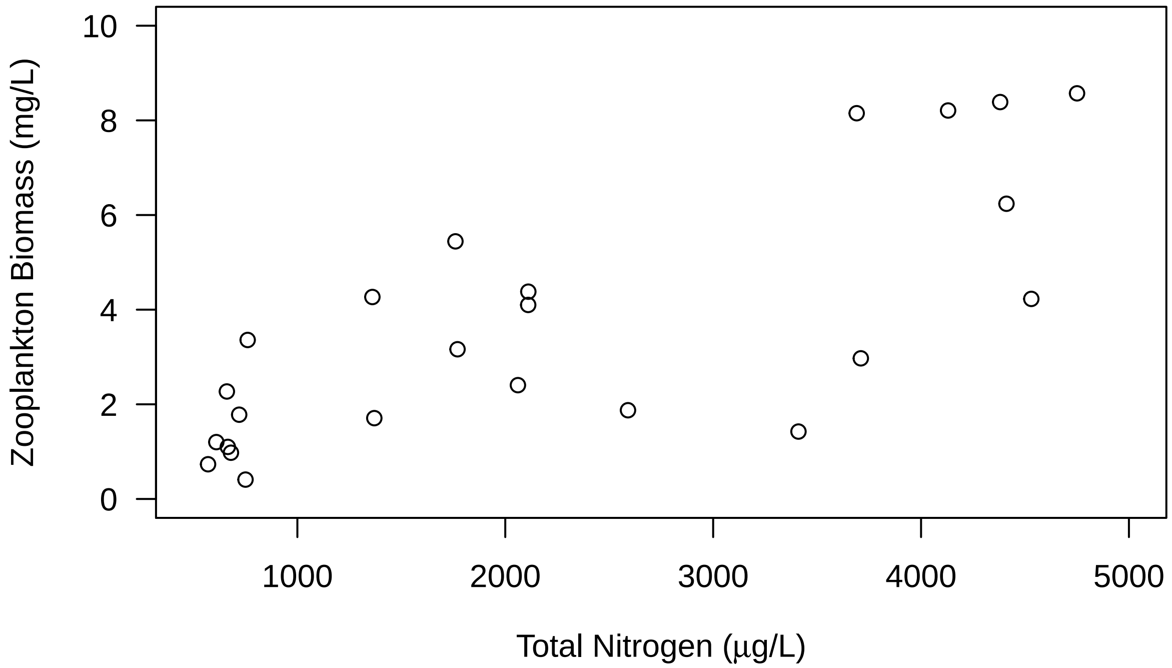
Let's examine the output of the regression model:

```
summary(fitreg)
```

Q6: Interpret the results from the regression model

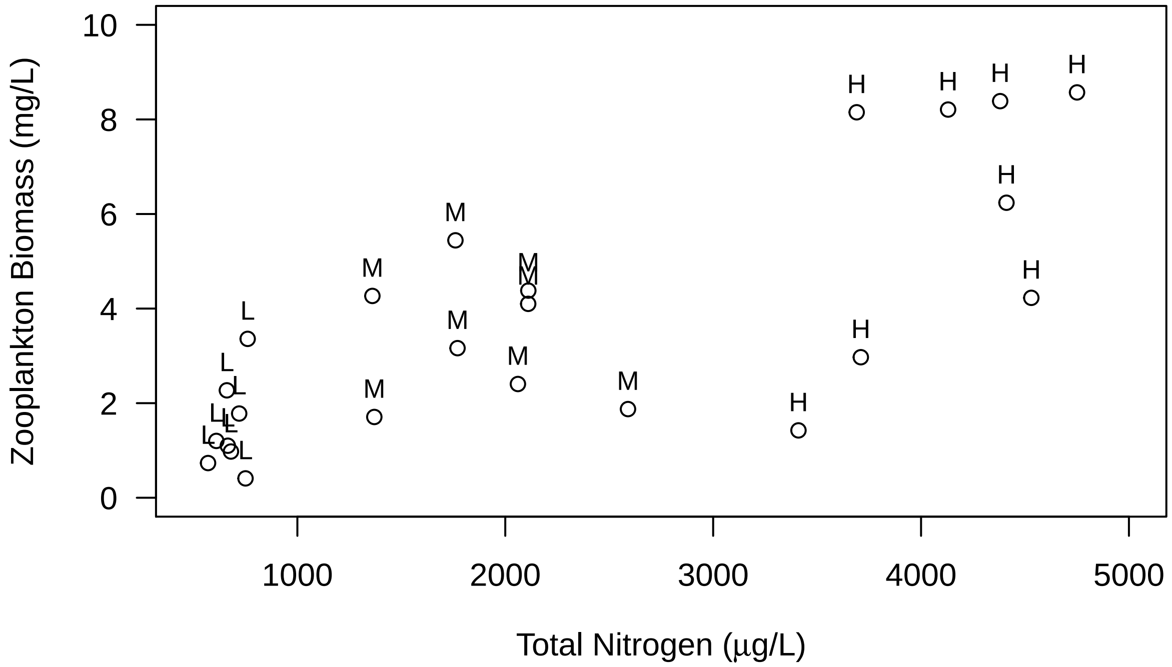
Now, let's look at a plot of the data used in the regression analysis:

```
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)
```



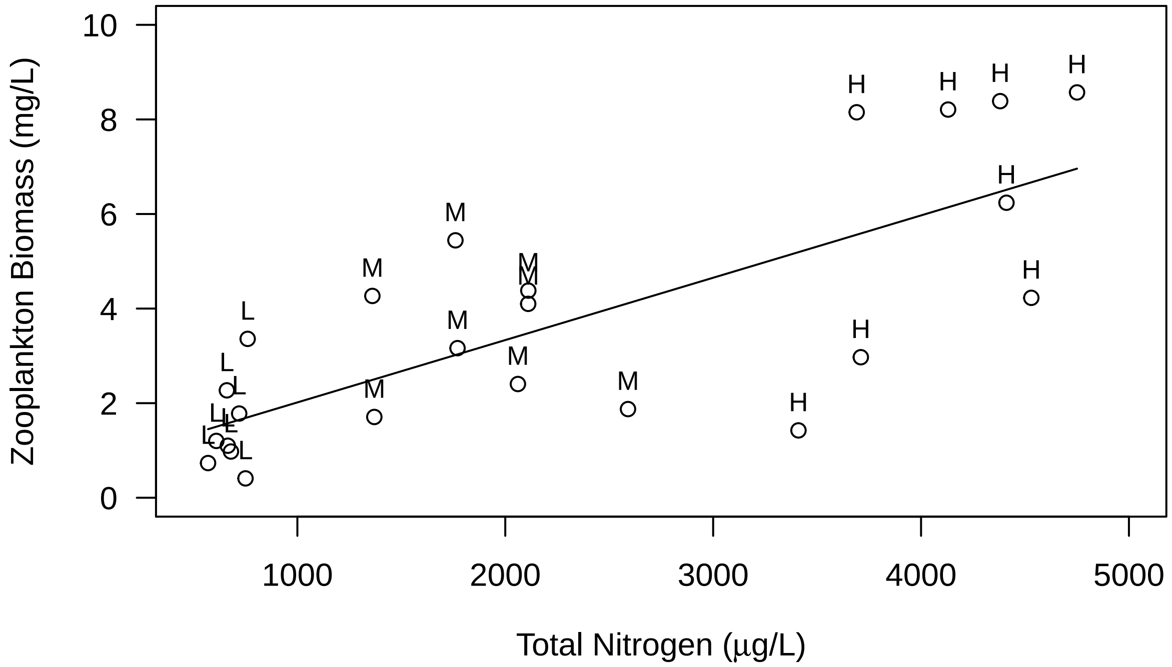
We can add some text to the plot to visualize the categorical nutrient treatments:

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



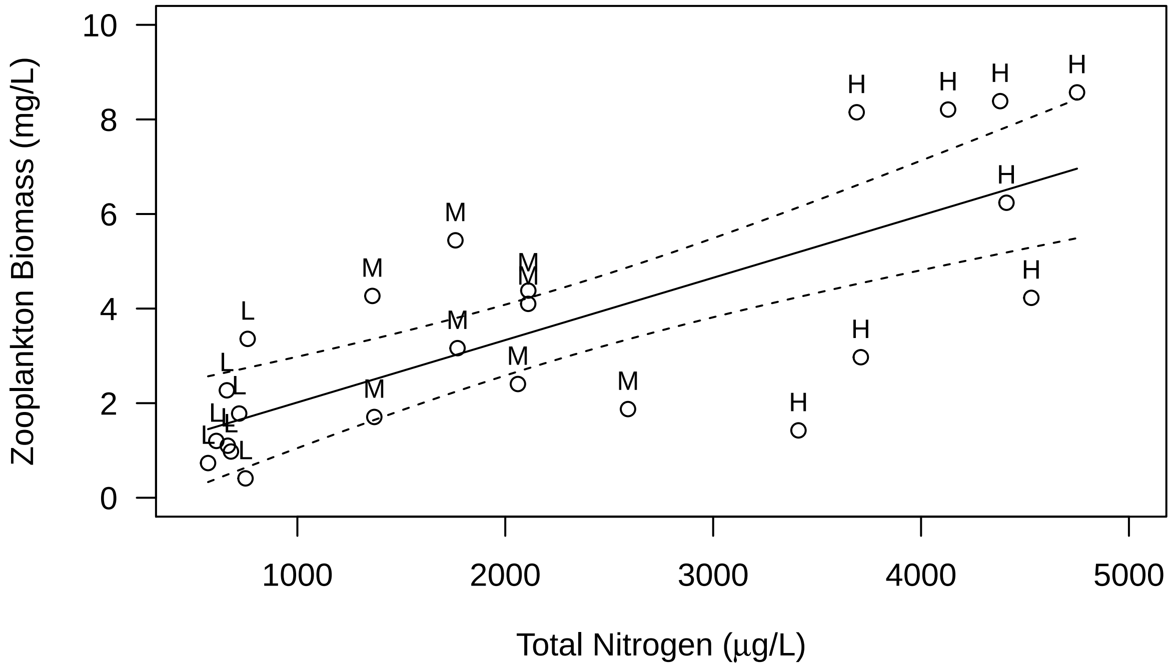
To add the regression line, the first thing we need to do is identify a range of x-values and then generate the corresponding **predicted values** from our regression model:

```
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's 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:

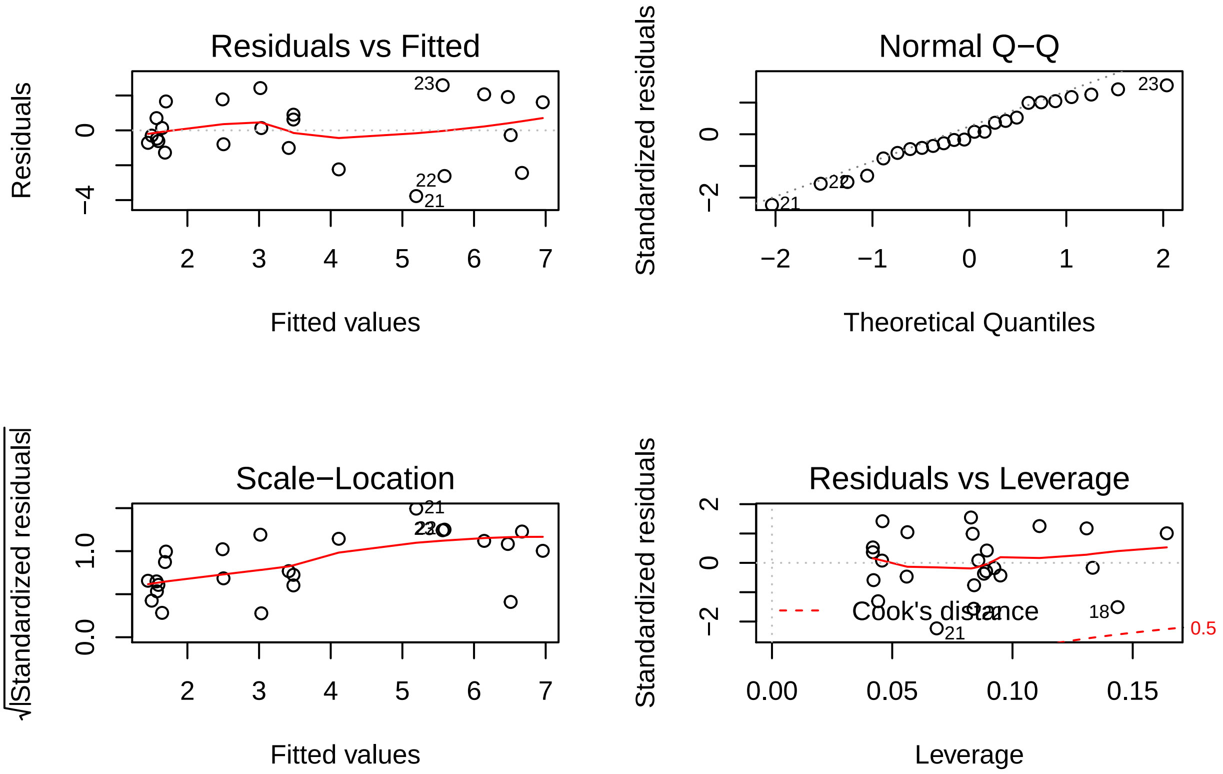
```
text(meso$TN, meso$ZP,meso$NUTS,pos=3,cex=0.8)
newTN <- seq(min(meso$TN), max(meso$TN), 10)
regline <- predict(fitreg, newdata = data.frame(TN = newTN))
lines(newTN, regline)
conf95 <- predict(fitreg, newdata = data.frame(TN = newTN),
                  interval = c("confidence"), level = 0.95, type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



Q7: Explain what you were just using the `predict()` function for?

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

ANALYSIS OF VARIANCE (ANOVA)

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

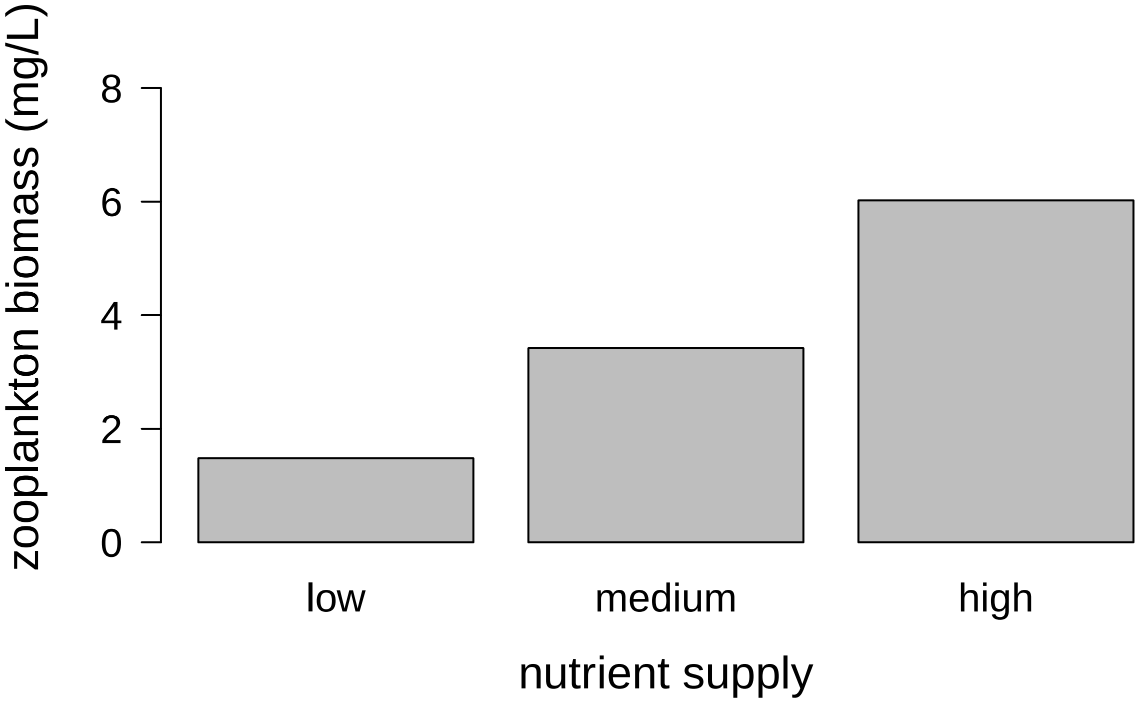
```
NUTS <- factor(meso$NUTS, levels = c('L', 'M', 'H'))
```

Before plotting, we need to calcualte the means and standard errors for zooplankton biomass in our nutrient treatments. We are goint 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$ZP, NUTS, mean)
zp.sem <- tapply(meso$ZP, NUTS, sem)
```

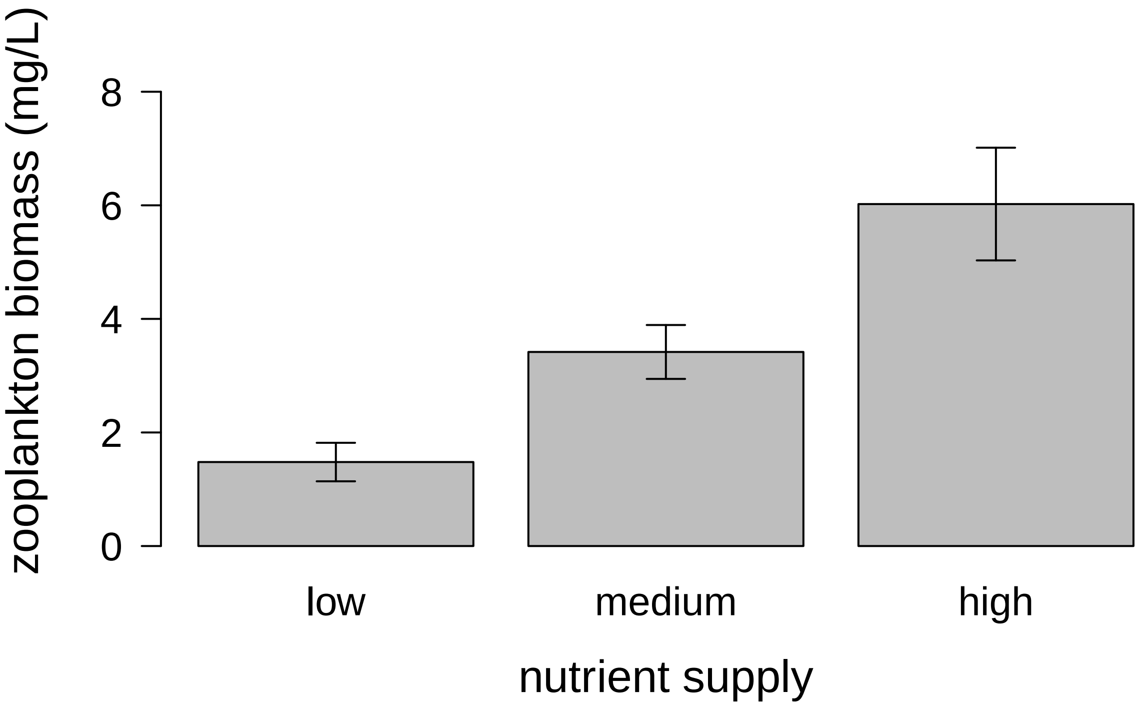
Now let's make the barplot:

```
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's look at the output in more detail (just as we did with regression):

```
summary(fitanova)
```

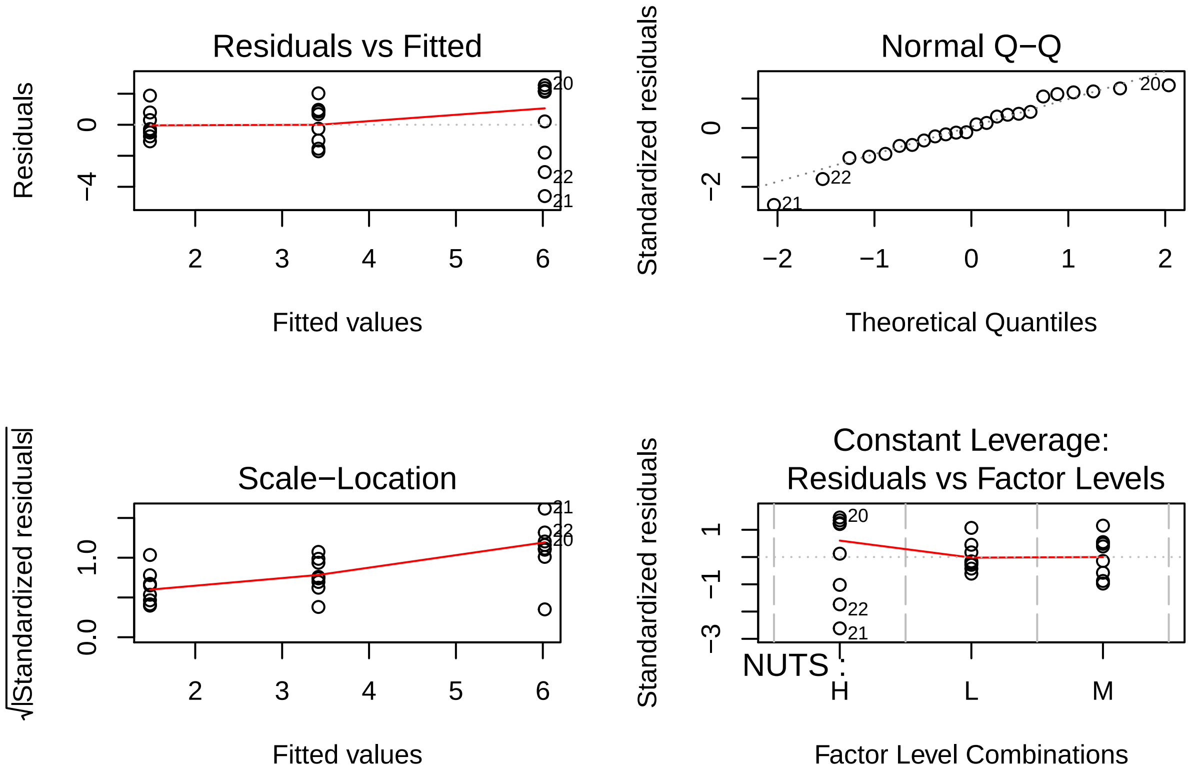
Finally, we can conduct a post-hoc comparison of treatments using Tukey's HSD (Honest Significant Differences). This will tell us whether or not there are differences (alpha = 0.05) among pairs of the three nutrient treatments

```
TukeyHSD(fitanova)
```

Q8 How do you interpret the ANOVA results relative to the regression results? Do you have any concerns about this analysis?

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

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



HOMEWORK

1) Complete the entire exercise in the Week1_R.Rmd

2) In addition, redo the section on linear regression (including summary statistics, plotting, and diagnostics) using log10-transformed zooplankton biomas data. Does the transformation help meet assumption of linear regression or change the interpretation of the analysis in anyway? Describe.

3) Use knitr to create a pdf of your completed Weed1_R.rmd document, push it to GitHub, and create a pull request.