

Week 1 Exercise: Basic R

Z620: Quantitative Biodiversity, Indiana University

January 16, 2015

In this exercise, we provide an introduction to some of the basic features of the R computing environment. We emphasize calculations, data types, and simple commands that will be useful for you during the course and beyond.

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>. When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

SETTING YOUR WORKING DIRECTORY (explain what and why)

```
getwd()
```

```
## [1] "/Users/lennonj/GitHub/Quantitative_Biodiversity/Assignments/Week1"
```

```
#The following line needs to be updated
```

```
#setwd("~/GitHub/Quantitative_Biodiversity/Assignments/Week1")
```

USING R AS A CALCULATOR

addition

```
1 + 3
```

```
## [1] 4
```

subtraction

```
3 - 1
```

```
## [1] 2
```

multiplication (with exponent)

```
3 * 10^2
```

```
## [1] 300
```

division (using a built-in constant)

```
10 / pi
```

```
## [1] 3.183
```

trigonometry with a simple built-in **function** (i.e., *sin*) and **argument** (i.e., '4')

```
sin(4)
```

```
## [1] -0.7568
```

logarithms (another example of function and argument)

```
log10(100)
```

```
## [1] 2
```

```
log(100)
```

```
## [1] 4.605
```

DEFINING VARIABLES

In R, you will often find it useful and necessary to assign values to a variable. Generally speaking, it's best to use `<-` rather than `=` as an assignment operator.

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

What is the value of b?

```
a <- 200
```

Now what is the value of b? Can you explain? Fix? It can help to examine variables with the following function

```
ls()
```

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

You can clear variables from R memory with following function (example of nested function)

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

You can also examine variables in the Environment window of R Studio. By clicking 'clear' in this window, you can erase variables from memory

→ Time for discussing R as a calculator and assigning of variables

WORKING WITH SCALARS, VECTORS, AND MATRICES

There is a hierarchy of mathematical elements. First, a **scalar** is single numeric value. Let's assign a numeric value to a character:

```
w <- 5
```

A **vector** (or array) is a one-dimensional row of numeric values. You can create a vector in R: like this:

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

What is the function `c()`? The `help()` function is your friend. 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?

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

Here is how you reference an element in a vector

```
z[2]
```

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

Here is how you reference multiple elements in a vector

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

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

Here is how you can change the value of an element in a vector

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

It's pretty easy to perform summary statistics on a vector using built-in functions

```
max(z)      # maximum
```

```
## [1] 980
```

```
min(z)    # minimum
```

```
## [1] 20
```

```
sum(z)    # sum
```

```
## [1] 3328
```

```
mean(z)   # mean
```

```
## [1] 475.4
```

```
median(z) # median
```

```
## [1] 583
```

```
var(z)    # variance
```

```
## [1] 133881
```

```
sd(z)     # standard deviation
```

```
## [1] 365.9
```

What happens when you take the standard error of the mean (**sem**) of z ? The standard error of the mean is defined as $SEM = \frac{sd(x)}{\sqrt{n}}$. Sometimes you need to make your own functions. Let's give it a try:

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

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 ? One solution is to tell R to remove NA from the dataset:

```
sem(i, na.rm = TRUE)
```

```
## [1] 16.03
```

There are three common ways to create a matrix (two dimensional vectors) in R. **Approach 1** is to combine (or concatenate) two or more vectors. Let's start by creating a vector using a new function **rnorm**

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

What does the `rnorm` function do? What are arguments doing? Now we will use the function `cbind` to create a matrix

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

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

Approach 2 to making a matrix is to use the `matrix` function:

```
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 (or elsewhere)

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

Often, when handling datasets, we want to be able to transpose a matrix. This is easy in R:

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

Also, you will find that you need to subset data in a matrix:

For example, maybe you want to take first three rows of a matrix:

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

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

```
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:

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

Basic Plotting and Statistics

In the following exercise, we will use a dataset from [Lennon et al. \(2003\)](#), which looked at zooplankton community assembly along a trophic state gradient. Nutrients (nitrogen and phosphorus) were added to mesocosms over the course of six weeks at three different levels (low, medium, and high). We also directly measured nutrients. So we have categorical and continuous predictors that we're going to use to help explain variation in zooplankton biomass.

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

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

Let's use the `str` function to look at the structure of the data. How does this dataset differ from the 'm' dataset above?

```
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 "high","low","medium": 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 ...
```

Now, let's explore the data set a little bit: *TANK = mesocosm identifier

*NUTS = categorical nutrient treatment

*TP = total phosphorus concentration ($\mu\text{g/L}$)

*TN = total nitrogen concentration

*SRP = soluble reactive phosphorus concentration

*TIN = total inorganic nutrient concentration

CHLA = *chlorophyll a** (proxy for algal biomass)

*ZP = zooplankton biomass

look at correlations among variables `cor.data <- cor(data[,3:8])`

`install.packages("corrplot") library(corrplot) corrplot(cor.data, method = "circle")`

Included in R and various R packages are some basic datasets that are useful for testing functions and learning about R features and functions. One such dataset is **cars**. To learn about this dataset you can simply use the `{r} help` function

```
help(cars)
```

Use the `{r} str()` and `'{r} summary()'` functions to see basic summary statistics about this dataset

```
str(cars)
```

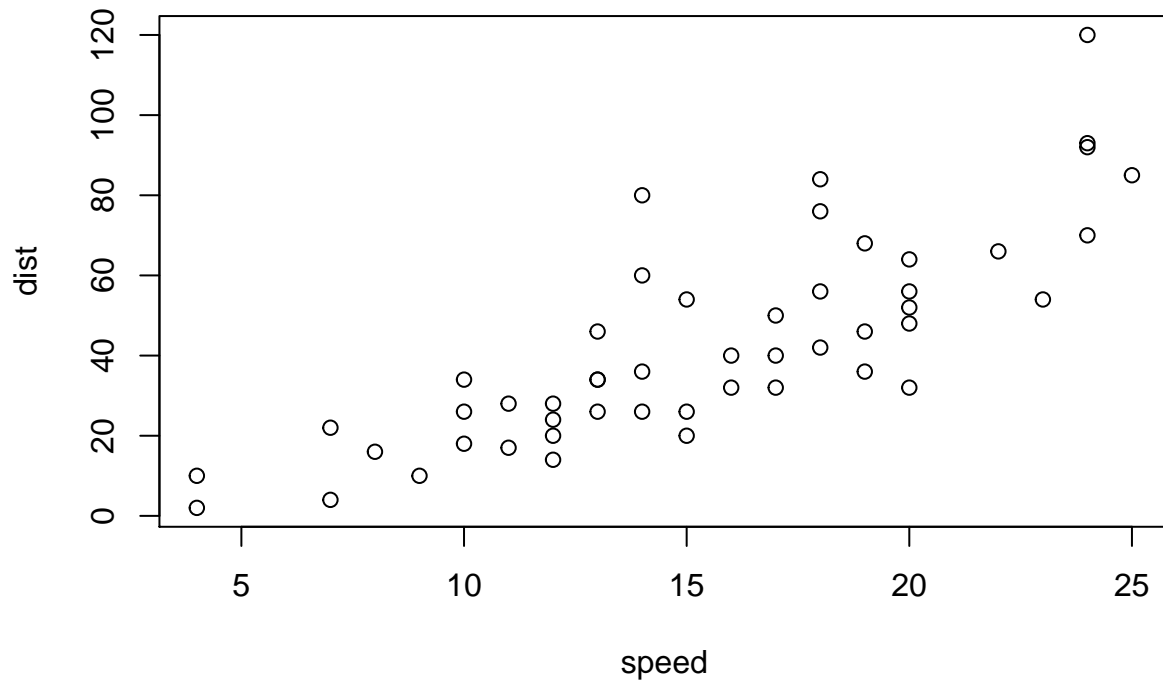
```
## 'data.frame': 50 obs. of 2 variables:
## $ speed: num 4 4 7 7 8 9 10 10 10 11 ...
## $ dist : num 2 10 4 22 16 10 18 26 34 17 ...
```

```
summary(cars)
```

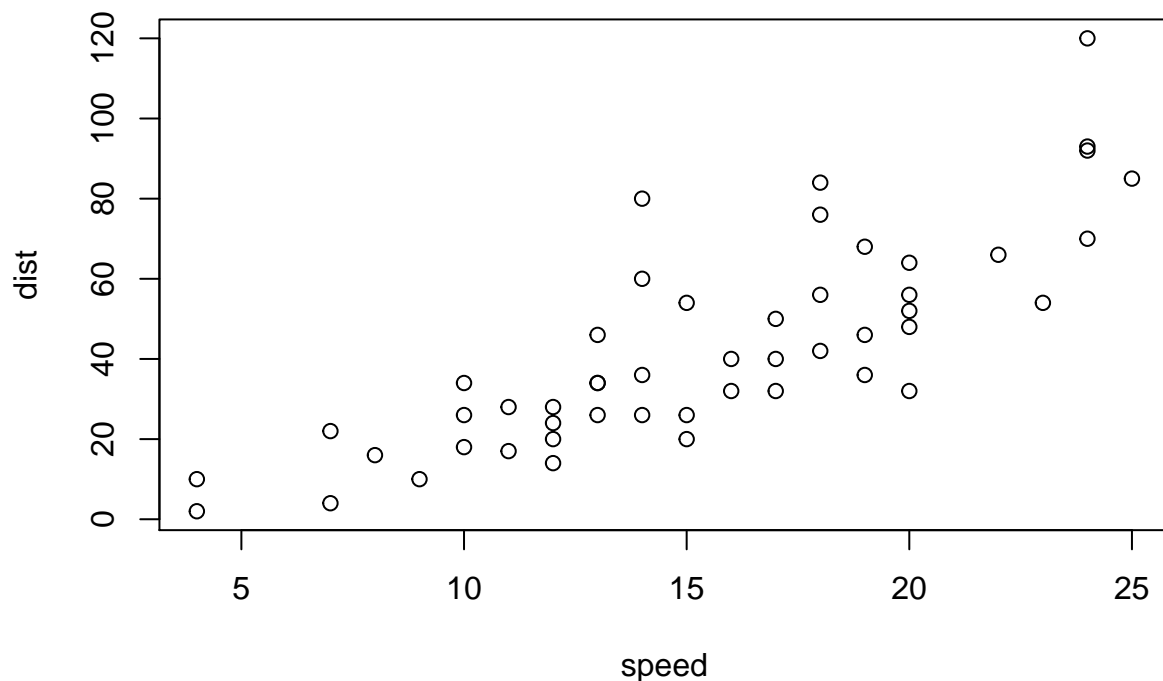
```
##      speed      dist
## Min.   : 4.0    Min.   : 2
## 1st Qu.:12.0    1st Qu.: 26
## Median :15.0    Median : 36
## Mean   :15.4    Mean   : 43
## 3rd Qu.:19.0    3rd Qu.: 56
## Max.   :25.0    Max.   :120
```

To visualize this data you can generate a simple plot with the `{r} plot()` function

```
plot(cars)
```



You can also embed plots, for example: `# JTL`, line by line, got an error; plus not sure how useful? what's point?



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

Other Useful Features and Functions: Sorting, Subsetting, Sampling

JTL: seems like some of this could be combined with stuff above

Sorting We can use another dataset (`mtcars`) to practice sorting (ordering) data. Learn about `mtcars` via `{r} help(mtcars)`

sort by mpg

```
newdata <- mtcars[order(mtcars$mpg),]
```

sort by mpg and cyl # JTL: not sure how effect the cyl sort is

```
newdata <- mtcars[order(mtcars$mpg, mtcars$cyl),]
```

sort by mpg (ascending) and cyl (descending) #JTL: same as above?

```
newdata <- mtcars[order(mtcars$mpg, - mtcars$cyl),]
```

Now, Let's make a new vector of data


```
z <- c(1.5, 1/6, 1/3)
```

If we only want to view the first two decimal places of z

```
round(z,2)
```

```
## [1] 1.50 0.17 0.33
```

Now, we can reverse the order of the elements in z

```
rev(z)
```

```
## [1] 0.3333 0.1667 1.5000
```

And we can order z from smallest to largest

```
sort(z)
```

```
## [1] 0.1667 0.3333 1.5000
```

We can also identify the ordering of z #JTL: with respect to what?

```
order(z)
```

```
## [1] 2 3 1
```

i.e., the 2nd number is the min and the 1st number is the max

Additionally, we can identify the maximum values this way:

```
max(z)
```

```
## [1] 1.5
```

Subsetting Let's create a original object vector, x:

```
x <- c(3, 4, 7)
x
```

```
## [1] 3 4 7
```

Now, let's subset this vector and keep only the first three values

```
x[-3]
```

```
## [1] 3 4
```

Now, let's subset this vector and keep only the values greater than or equal to 5

```
x[x >= 5]
```

```
## [1] 7
```

Notice that we did this using a logic statement `{r} >=`. Here is a list of other logical operators that you might find useful:

[Logic Operator|Meaning| # confusing to start using new symbols “|”? |! x | Is Not “x”| |x & y| “x” and “y” (element by element)| |x && y| “x” and “y” (across all elements)| |x | y | “x” or “y” (element by element)| |x || y | “x” or “y” (across all elements)]

You can learn more about these commands (`{r} help(Logic, package=base)`)

Sampling

First, let's create a sequence of numbers

```
seq(1,3,length=5)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0
```

```
# Create the same sequence in a slightly different way:  
seq(1,3,by=0.5)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0
```

```
# Create another sequence by going from 3 to 1:  
seq(3,1,by= -0.5)
```

```
## [1] 3.0 2.5 2.0 1.5 1.0
```

To randomly sample from an existing vector:

```
sample(x,10,replace=T)
```

```
## [1] 4 7 3 3 7 7 7 4 7 4
```

Or to randomly sample from a sequence of numbers from 1 to 500:

```
sample(1:500,10,replace=F)
```

```
## [1] 316 120 450 361 234 17 16 46 231 415
```