Week 1 Exercise

Z620: Quantitative Biodiversity, Indiana University

November 4, 2014

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

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.

RETRIEVING AND SETTING YOUR WORKING DIRECTORY

USING R AS A CALCULATOR

addition

subtraction

multiplication (with exponent)

division (using a built-in constant)

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

logarithms (another example of function and argument)

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.

What is the value of b?

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

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

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

WORKING WITH SCALARS, VECTORS, AND MATRICES

Create a *scalar* by assigning a numeric value to a character

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

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

What happens when you multiply a vector by a scalar?

What happens when you multiply two vectors?

Here is how you reference an element in a vector

Here is how you reference multiple elements in a vector

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

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

What happens when you take the standard error of the mean (sem) of z? Sometimes you need to make your own functions:

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

What happens when you apply your sem function to vector i? One solution is to tell R to remove NA from the dataset:

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'

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

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:

Approach 3 to making a matrix is to import or 'load' a dataset from your working directory (or elsewhere)

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

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:

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

Or perhaps you want non-sequential columns of a matrix:

Student Questions

1. This is an example Question

    ```
    Answers will be here.
    ```

Basic Plotting

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 simple use the `{r} help` function

```
help(cars)
```

```
## starting httpd help server ... done
```

Use the '{r} summary()' function to see basic summary statistics about this dataset

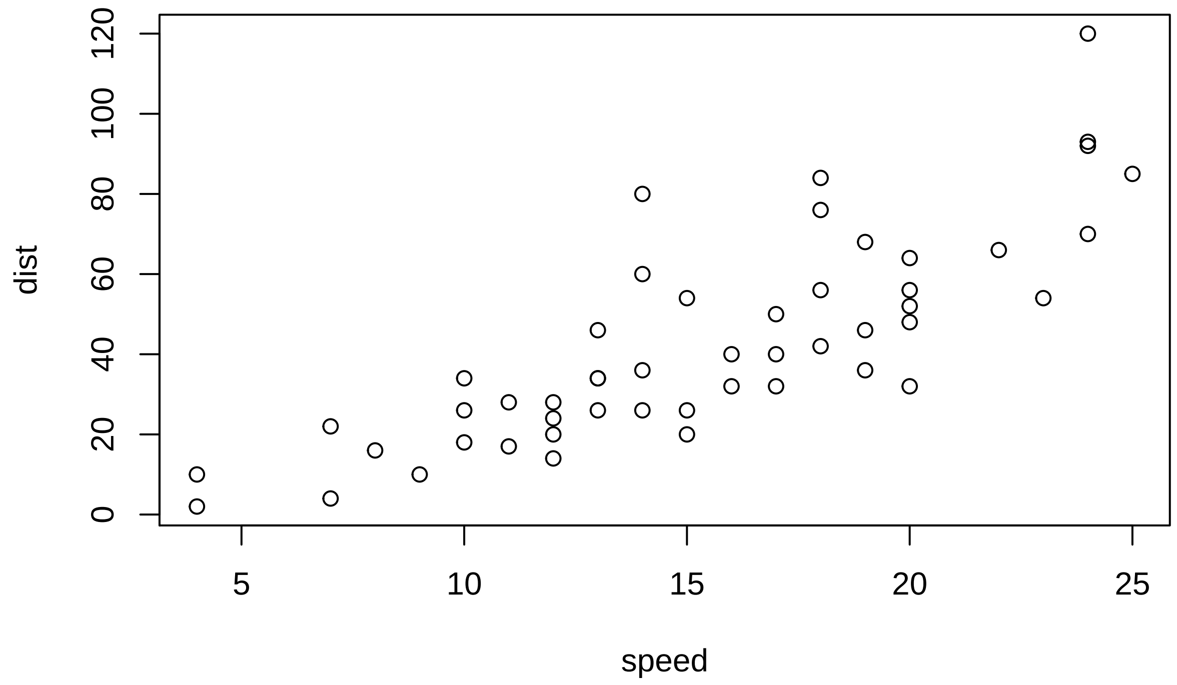
```
summary(cars)
```

```
##      speed           dist    
##  Min.   : 4.0   Min.   :  2  
##  1st Qu.:12.0   1st Qu.: 26  
```

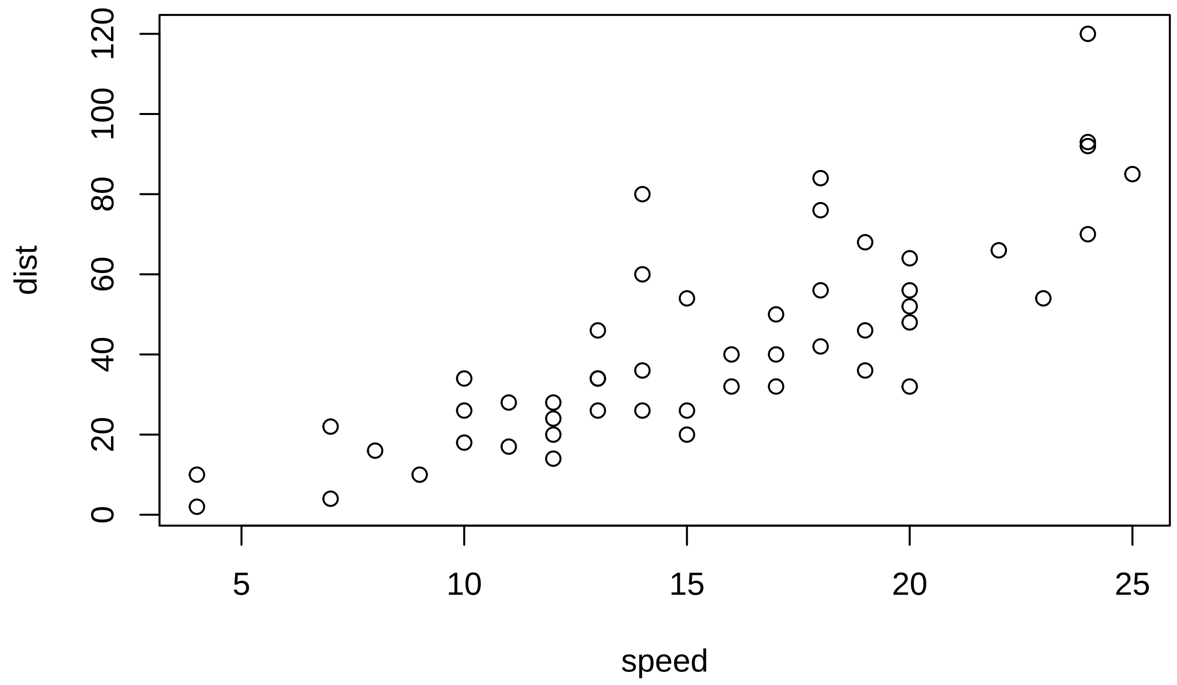
```
##  Median :15.0   Median : 36
##  Mean   :15.4   Mean   : 43
##  3rd Qu.:19.0   3rd Qu.: 56
##  Max.   :25.0   Max.   :120
```

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

```
plot(cars)
```



You can also embed plots, for example:



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 Fucntions: Sorting, Subsetting, Sampling

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

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

sort by mpg (ascending) and cyl (descending)

```
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

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

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

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

```r
rev(z)
```

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

And we can order z from smallest to largest

```r
sort(z)
```

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

We can also identify the ordering of z

```r
order(z)
```

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

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

Additionally, we can idenify the maximum values this way:

```r
max(z)
```

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

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

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

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

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

```r
x[-3]
```

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

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

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

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

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

|Logic Operator|Meaning| |! 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 elments)|

You can learn more about this 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 4 4 4 7 4 4 4 7 3
```

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

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

```
##  [1] 358 477 492 476  51  29  82 235 475 132
```