Having a taste of R language

My colleagues have found the R language interesting and would like me to show them some very basic steps in applying the language in statistical evaluation. Below are some illustrations on simple steps used in the R language. We begin by discussing what "vector" is about:

For the construction of a vector from given values, we use: c(….) operator. For example,

```
> c(4,5,2,8,6,9,1)
[1] 4 5 2 8 6 9 1
> c(pi,2*pi,3*pi)
[1] 3.141593 6.283185 9.424778
> c("This is your first R program line")
[1] "This is your first R program line"
> c(TRUE,FALSE,TRUE)
[1]  TRUE FALSE  TRUE
```

If the arguments to c(…) operator are themselves vectors, we can flatten them and combine them into one single vector:

```
> x <- c(1,2,3)
> y <- c(4,5,6)
> c(x,y)
[1] 1 2 3 4 5 6
```

However, if we were to sum up x+y, we got:

```
> z <- c (x+y)
>z
[1] 5 7 9
```

We use n:m to create a sequence of numbers, such as:

```
> 0:8
[1] 0 1 2 3 4 5 6 7 8
```

Interestingly, R knows that 8 is larger than 0, so it counts backward from the starting to ending value:

```
> 8:0
[1] 8 7 6 5 4 3 2 1 0
```

But, the colon operator works for sequences that grow by 1 only. We use "seq" function to build sequences but can support it with an optional third argument "by", which is the increment:

```
> seq(from=0,to=10, by=2)
[1]  0  2  4  6  8 10
>
> seq(from=5, to=30, by=5)
[1]  5 10 15 20 25 30
```

Alternatively, we can specify a length for the output sequence and then R will calculate the necessary increment:

```
> seq(from=10, to=20, length.out=5)
[1] 10.0 12.5 15.0 17.5 20.0
```

We take note of some key properties of vectors:

1. Vectors are homogeneous (meaning all elements of a vector must have the same type or, in R terminology, the same mode)
2. Vectors can be indexed by position (for example, w[2] refers to the second element of w). *Note the use of bracket signs* [ ].
3. Vectors can be indexed by multiple positions, returning a sub-vector (for example, w[c(2,3)] is a sub-vector of w that consists of the second and third elements)
4. Vectors elements can have names
   It means that vectors have a 'names' property, the same length as the vector itself, that gives names to the elements:

   ```
   > w<-c(5,50,100)
   > names(w)<-c("ABC", "DEF", "GHI")
   > print(w)
   ABC DEF GHI
     5   50 100
   ```
5. If vector elements have names then we can select them by name:
   Continuing the previous example:

   ```
   > w["DEF"]
   DEF
   50
   ```

In R, a scalar is simply a vector that contains exactly one element. Consider the built-in constant *pi* (π). It is indeed a scalar:

```
> pi
[1] 3.141593
```

Since a scalar is a one-element vector, we can use vector functions on *pi*:

```
> length(pi)
[1] 1
```

and, this scalar does not have a second element as shown below:

```
> pi[2]
[1] NA
```

In R, a matrix is just a vector that has dimensions. We can transform a vector into a matrix simply by giving it dimensions.

To continue with this discussion, we note that vector has an attribute called "dim", which is initially NULL, as shown here:

```
> A<-1:6
> dim(A)
NULL
> print(A)
[1] 1 2 3 4 5 6
```

We can give dimensions to the vector when we set its "dim" attribute. See below which shows the vector can be reshaped into a 2x3 matrix:

```
> A<-1:6
> dim(A)<-c(2,3)
> print(A)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

It is interesting that R can generate matrices to a 3-dimensional or even *n*-dimensional structures by just assigning more dimensions to the underlying vector.

The following example creates a 3-D array with dimensions 2x3x2:

```
> D<-1:12
> dim(D)<-c(2,3,2)
> print(D)
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

Note that R prints one "slice" of the structure at a time because it is not possible for us to print a 3-D structure on a 2-D medium!

Now, let's see how we calculate basic statistics: mean, median, sample standard deviation and sample variance (which is square of standard deviation):

```
> x<- c(3,6,4,1,8,7,4,9,12,10,6)
> x
[1]  3  6  4  1  8  7  4  9 12 10  6
> mean(x)
[1] 6.363636
> median(x)
[1] 6
> sd(x)
[1] 3.26413
> var(x)
[1] 10.65455
> x-mean(x)
 [1] -3.3636364 -0.3636364 -2.3636364 -5.3636364  1.6363636  0.6363636
 [7] -2.3636364  2.6363636  5.6363636  3.6363636 -0.3636364
```

Another example of the R calculation:

```
> a<- c(4,6,2,4,8)
> mean(a)
[1] 4.8
> sd(a)
[1] 2.280351
> z<- (a-mean(a))/sd(a)
> z
[1] -0.3508232  0.5262348 -1.2278812 -0.3508232  1.4032928
```

All these functions are picky about values that are not available (NA). Even one NA value in the vector argument can cause any of these functions to return NA:

```
> y<-c(2,6,4,9,3,NA)
> mean(y)
[1] NA
> sd(y)
[1] NA
```

However, if we want to tell R to do the right calculation, we use the function "rm" (remove memory) to reset the default FALSE to TRUE:

```
> y<-c(2,6,4,9,3,NA)
> mean(y, na.rm=TRUE)
```

```
[1] 4.8
> sd(y, na.rm=TRUE)
[1] 2.774887
```

Note: R prefers to use the assignment operator "<-" instead of equals-sign assignment"=" to prevent potential confusion with its expressions in test for equality. Make sure this assignment operator is formed from a less-than character (<) and a hyphen (-) with no space between them.