Analyzing Values Below the Method Detection Limit Using R

Carolyn Huston
Simon Fraser University
AQ 3148

February 17, 2008

More on R

Jumping Into Part II!

Data

Statistics is about the analysis of data. Data can be stored as an object in R. Because R is mainly a statistical program, it provides lots of functions to store, manipulate, and perform statistical analysis on all sorts of data. There are several types of data objects in R that we will be discussing.

More on Objects

A question for you!

Question

Are any of you thinking right now ‘Carolyn has been going on and on about objects and functions, and I kind of get it, but..... HUH???’

Question

If this is you, please think of an object in the everyday world and raise your hand. We can use this example to discuss the conceptual similarities between how objects are used in the everyday world, and how objects are used in an R workspace.

...And now back to data objects!

Vectors

A vector is simply a collection of numbers. Vectors can be a convenient way to store a series of data, for example a list of measurements. Vectors are basically a data set where we have a univariate measurement, with no covariates. For example, a long row of solute concentration values.

Vectors

In R, vectors are a type of data object. To create a vector in R, use the `c()` function (c stands for ‘concatenate’)(To concatenate is to link things together in a chain or series).

For example,

```
>x=c(2,3,5,7) #1st 4 prime numbers
>x
[1] 2 3 5 7
```

Vectors

Functions and logical operators that we talked about before break will work on each element of the vector.

```
>x^2
[1]4 9 25 49
>log(x)
[1]0.693 1.098 1.609 1.9459
>x>5
[1]FALSE FALSE FALSE TRUE
```

Vectors

Two vectors can be combined using arithmetic functions, element by element:

```
>y=c(1,2,3,4)
>x+y #adding elements
[1]3 5 8 11
>x*y #multiplying elements
[1]2 6 15 28
>x==y #comparing elements
[1]FALSE FALSE FALSE FALSE
```

SFU

When two vectors have different lengths, the shorter one is repeated to match the longer one. This is called the *recycling rule*. If the length of the shorter vector is not a multiple of the length of the other, R will print a warning.

```
>y=c(1,2)
>x+y #adding elements
[1]3 5 6 9
```

Here, `x` is of length 4 while y is of length 2. To produce `x+y`, the variable `y` is repeated to make `c(1,2,1,2)`, and then added to `x` as usual.

This can cause mistakes if you are not used to it!!

Shortcuts For Creating Vectors

Creating Vectors

You might often need to create a vector that is a sequence of numbers, or perhaps create a vector where all the elements are the same. There are several shortcuts in R available to help you do this.

Creating Vectors

To create a list of numbers between 10 and 50, we could use

```
>x=10:50
[1] 10 11 12 13 14 15 16 17 18 19 20 21 22
[14] 23 24 25 26 27 28 29 30 31 32 33 34 35
[27] 36 37 38 39 40 41 42 43 44 45 46 47 48
[40] 49 50
```

The numbers in square brackets tell you which elements of the vector starts a new line. The first line starts with the first [1] element (10). The second line starts with the 14th element (23), etc. This is why R has previously put a [1] before our output; if the answer is a single number then it is a vector of length 1.

Creating Vectors

You can create the same vector using the `seq()` function

```
>seq(from=10, to=50, by=1)
```

This creates a sequence of numbers between 10 and 50 inclusive, incrementing by 1 each time. Alternatively, you can specify how many numbers you want in the result; numbers will automatically be equally spaced.

```
>seq(from=10, to=50, length=41)
```

```
>seq(from=10, to=50,length=40)
```

The result need not be in integer values!!

Creating Vectors

Another useful function that can be used to create vectors is `rep()`. This creates a vector by repeating a number (or sequence of numbers) a specified amount. For example, to repeat the number 3, 5 times

```
>rep(x=3,times=5)
[1] 3 3 3 3 3
```

SFU

Creating Vectors

An interesting feature of the function `rep()` is that the value to be repeated can also be a vector. If this is the case, we can use the argument `each` to represent the number of times to repeat each element of the vector. For example

```
>rep(x=1:3,each=3)
[1] 1 1 1 2 2 2 3 3 3
```

Creating Vectors

In my text I say that being able to use functions like `seq()`, and `rep()` are useful. What about you, can you see any applications for this type of function?

Manipulating Vectors

Manipulating Vectors

R allows you to manipulate vectors in many ways. You can extract a particular element of a vector using square brackets, [].

```
>x=1:5
```

Extract the 3rd element of x

```
>x[3]
[1] 3
```

Extract everything but the 3rd element of x

```
>x[-3]
[1] 1 2 4 5
```

Manipulating Vectors

Continuing from the previous slide..

Change the 3rd element of x to 10

```
>x[3]=10
[1] 1 2 10 4 5
```

Use another vector, i, to extract a list of elements

```
>i=c(1,3,5)
>x[i]
[1] 1 10 5
```

Manipulating Vectors

Continuing some more...

Extract all the elements except those in i

```
>x[-i]
[1] 2 4
```

Add a new element (to the end of x)

```
>x[6]=6
>x
[1] 1 2 10 4 5 6
```

Another way to add a new element, using c()

```
>x=c(x,7)
>x
[1] 1 2 10 4 5 6 7
```

Manipulating Vectors

A useful function is the `which(logical)` function. This function evaluates a logical expression. The logical expression creates a vector of TRUE and FALSE. The function `which()` reports the position of elements that are TRUE.

Manipulating Vectors

```
>x=c(10,7,9,10,12,9,10)
```

A logical expression returning TRUE and FALSE

```
>x==10
[1]  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE
[8]  TRUE
```

The location of the elements equalling 10

```
>which(x==10)
[1]  1 4 5 8
```

Manipulating Vectors

A new vector, y where elements equalling 10 have been extracted

```
>y=x[which(x==10)]  #NB!!
[1]  10 10 10 10
```

Values of x which are less than 10

```
>x[which(x<10)]
[1]  7 9 9
```

Functions of Vectors

Earlier, we saw and used some mathematical functions that could also be applied to vectors. These functions calculate a value for each element of a vector. There are also some useful functions that work on the whole vector to provide a single answer. A table of these functions is shown on the next page.

Functions of Vectors

Function Name	**Value returned**
`length()`	The number of elements
`max()`	The largest value
`min()`	The smallest value
`sum()`	The sum of the elements
`prod()`	The product of the elements
`mean()`	The mean of the elements
`sd()`	The standard deviation of the elements
`var()`	The variance of the elements
...	And lots lots more

Functions of Vectors

For example, to find the length of a vector, use the `length()` function

```
>length(x)
[1]  8
```

Data Frames

A data frame is another type of data object in R. It is used to store complete datasets. For example, if there was a dataset representing the heights and weights of 10 different people, the data frame in R will be a table with 10 rows, one per person, and 2 columns, one for weight and one for height. Data frames can also have row and column labels, making them ideal for storing datasets. Information in a data frame is much like you would see in a spreadsheet. There is one column for each variable, and one row for each observation. Different types of data can also be stored (ie. words).

Data Frames

For example, height and weight data might be stored in a data frame called `hgtdata`:

```
>hgtdata

   height weight
1     169     61
2     167     69
3     166     70
4     172     76
5     162     60
6     185     58
7     165     61
8     171     72
9     170     55
10    163     63
```

Data Frames

You can access the heights vector by itself using the data name and column name separated by the $ operator:

```
>hgtdata$height
[1] 169 167 166 172 162 185 165 171 170 163
```

Data Frames

Once you have accessed the height vector, you can use the functions that we have learned about previously. For example

```
>mean(hgtdata$height)
[1] 169
```

Data Frames

Data rows, columns, and individual values can all be accessed using square brackets, [], similar to what we saw with vectors.

To access row 3, all columns

```
>hgtdata[3,]
```

```
  height weight
3    166     70
```

Data Frames

To access the height column (an alternative to $)

```
>hgtdata[,1]
[1] 169 167 166 172 162 185 165 171 170 163
```

To access a specific value in the matrix, for example the observation in the 5th row and 2nd column

```
>hgtdata[5,2]
[1] 60
```

Data Frames

In addition to extracting rows, columns, and values that can be used in functions, there are also some functions that can be applied to the data frame itself.

The `names()` function will give you the names of the columns. Notice the quotations around the words.

```
>names(hgdata)
 [1] "height" "weight"
```

File Input and Output

Entering Data

Often you will read data that comes from another source, for example that has been typed into an Excel spreadsheet. If the data is stored in a text file, you can read it into R using the `read.table()` function. It is straightforward to save an Excel spreadsheet as a text file, so this is my preferred method to import data into R. Some alternative methods to import files into R from Excel are offered in the guidance document.

The `read.table()` function has many optional arguments depending on how the data is written in the file. However, the basic function is simple.

Entering Data

```
>data=read.table("h:/hgtdata.csv")
```

Data can also have .dat, and .txt extensions. Using a .doc or .rtf extension will not work, because such files have embedded formatting that R does not like.

By default, when reading in data using `read.table"()`, each variable(column) will be labelled V1, V2, . . .

Entering Data

If a data file you are using specifies the column names in the first line, you can get R to use the labels by using the argument `header==TRUE`

```
>data=read.table("h:/hgtdata.csv",
header=TRUE)
```

Another EXTREMELY useful option is to specify what character has been used to separate one variable from the next. Often, a comma is used

```
>data=read.table("h:/hgtdata.csv",
header=TRUE, sep=",")
```

Entering Data

If the data are in a *tab-separated* (*tab-delimited)* text (.txt) file, use the special character combination '`\t`'. Any other separation formats can be written as they are found.

Entering Data

Another very useful argument for `read.table()` is `colClasses`. This allows you to specify the data type of the different columns that are being read into R, which can be useful if you have a factor column that has been entered as 1's, 2's, etc. Data can be of multiple different 'types', including `numeric`, `logical`, `factor`, etc.

```
>data=read.table("h:/hgtdata.csv",
header=TRUE, sep=",",
colClasses=c("numeric","numeric")
```

Entering Data

There are other arguments that can be used in the `read.table()` function, but I have tried to outline those that are the most useful. You can use the help files to learn about other options when reading in data.

Writing Data

If you have a data frame you want to save to a file, you can use the function `write.table()`. Again, this command has many options. To save a data frame in the same format as I have been reading them in, you can use the following commands.

```
>data=write.table(hgtdata,
file="h:hgtdata.csv",col.names=T, sep=",")
```

For unexplained reasons, `write.table()` uses the option `col.names` instead of `header` to indicate that column names are present in the file.

Recording Output

Usually R output prints directly to the screen. You can use the `sink()` command so that R prints results to a file instead.

Creates new file in the h: directory

```
>sink("h:/results.r")
```

No output to screen, just to h:

```
>mean(x)
```

Closing the file

```
sink()
```

This output will appear on screen

```
>mean(x)
```

Now for some exercises!!