CHAPTER 20

Simple Data Manipulation

In this chapter, we discuss ways to manipulate data, such as combining several columns to form a matrix, manipulating matrices, and choosing a subset from a big one.

20.1. The Functions head() and tail()

When a data set is big, it is a great idea to view the first and the last couple of lines. In this case, we use the **head()** and the **tail()** functions.

```
> x<-seq(1,500,1.5)
> head(x)
[1] 1.0 2.5 4.0 5.5 7.0 8.5
> tail(x)
[1] 491.5 493.0 494.5 496.0 497.5 499.0
```

We could specify number of lines we want to show, such as **head(x,20)** or **tail(x,20)**.

```
> tail(x,20)
[1] 470.5 472.0 473.5 475.0 476.5 478.0 479.5 481.0 482.5 484.0 485.5
[12] 487.0 488.5 490.0 491.5 493.0 494.5 496.0 497.5 499.0
>
```

20.2. The Function summary()

If **x** is a vector or matrix and we want to know more about the variable, we can use the **summary()** function.

```
> x<-1:500
> summary(x)
   Min. 1st Qu. Median Mean 3rd Qu. Max.
   1.0 125.8 250.5 250.5 375.2 500.0
```

20.3. Function ls() vs. ls(pattern='my_pattern')

While the **ls()** function is used to list all objectives, we use **ls(pattern="keyword")** to list objectives with a specified keyword or pattern.

```
> ls(pattern='my')
[1] "my_double" "my_pv"
```

20.4. Types of Variables

To know the types of variables, we can use **typeof(x)**.

```
> x<-1:5
> typeof(x)
[1] "integer"
```

```
> x<-'I love apples'> typeof(x)
[1] "character"
```

The following table lists different types of variables: definitions plus examples

Table 20.1. Types of variables

Name	Examples	Dimension
Scalar	>x<-10	> x<-10 > length(x) [1] 1
Vector	>y<-c(0.1,0.34,0.98) >x<-1:20 >t<-seq(1,10,by=0.5)	> x<-seq(1,50,2.34) > length(x) [1] 21
Matrix	> x<-1:100 > y<-matrix(x,2,50)	> x<-matrix(0,3,2) > dim(x) [1] 3 2
List	> x<-list(c(1,2,3),1:20,4)	> x<-list(c(1,2,3),1:20,4) > length(x) [1] 3
Factor	> x<-c(1,2,4,1,1,1,2) > y<-as.factor(x) > y [1] 1 2 4 1 1 1 2 Levels: 1 2 4	

20.5. The Function is.vector() and Similar Functions

We can use the **is.vector()** function and other similar functions to identify whether the variable is a vector, matrix, integer, or others.

```
> x<-1:10 # here are the answers
> is.vector(x)
[1] TRUE
> is.matrix(x)
[1] FALSE
> is.integer(x)
[1] TRUE
> is.list(x)
[1] FALSE
> is.real(x)
[1] FALSE
```

See a few similar examples below:

```
> is.character(2)
[1] FALSE
> is.numeric("Hi")
[1] FALSE
> x<-c(1,2,4,1,1,1,2)
> y<-as.factor(x)
> is.factor(y)
[1] TRUE
```

20.6. Functions length() vs. dim()

For a vector, we use the **length()** function to find out its number of values (length) while we use the **dim()** function to find out the dimensions of a matrix.

```
> x<-11:8
> length(x)
[1] 20 # there are 20 values
> y<-matrix(x,4,5)
> dim(y)
[1] 4 5 # dimensions of y is 4 by 5
```

20.7. The Function cbind()

We can use the **cbind()** function to combine two vectors (columns), as in the example below:

```
> x<-1:10
> y<-2:11
> cbind(x,y)
      x  y
 [1,] 1  2
 [2,] 2  3
 [3,] 3  4
 [4,] 4  5
 [5,] 5  6
 [6,] 6  7
 [7,] 7  8
 [8,] 8  9
 [9,] 9 10
[10,] 10 11
```

When a vector is shorter than another, R recycles the values from the shorter vector.

```
> x<-1:5
> y<-c(0.2,0.3)
> cbind(x,y)
 x y
[1,] 1 0.2
[2,] 2 0.3
[3,] 3 0.2
[4,] 4 0.3
[5,] 5 0.2
Warning message:
In cbind(x, y) :
 number of rows of result is not a multiple of vector length (arg 2)
```

Sometimes we can use this property to add a constant (see below).

```
> x<-1:10
> cbind(20,x)
x
 [1,] 20 1
 [2,] 20 2
 [3,] 20 3
 [4,] 20 4
 [5,] 20 5
 [6,] 20 6
 [7,] 20 7
 [8,] 20 8
 [9,] 20 9
 [10,] 20 10
```

Since a matrix has the same data types for all its values, when a column is a character, , all columns become characters when using the **cbind()** function. This is true for the **rbind()** function.

```
> x<-1:5
> cbind("ibm",x)
x
           x
[1,] "ibm" "1"
[2,] "ibm" "2"
[3,] "ibm" "3"
[4,] "ibm" "4"
[5,] "ibm" "5"
```

20.8. Converting a Vector into a Matrix

Below, first we generate a vector then convert it into a matrix.

```
> x<-1:100
> y<-matrix(x,5,20) # nrow first and ncol second
```

We should pay attention when we convert a vector into a matrix: column first or row first. Since we know that x has values from 1, 2, 3, up to 100, we can see how the matrix is arranged by printing the first several lines (see the codes below).

```
> x<-1:100
> y<-matrix(x,50,5)
> dim(y)
[1] 20 5
> head(y)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1   21   41   61   81
[2,]    2   22   42   62   82
[3,]    3   23   43   63   83
[4,]    4   24   44   64   84
[5,]    5   25   45   65   85
[6,]    6   26   46   66   86
```

Obviously the default setting is the column first. If we want to sort a vector into a matrix by row, we have to specify this condition by using **byrow=T**, where T stands for "true."

```
> y<-matrix(x,20,5,byrow=T)
> head(y)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
[5,]   21   22   23   24   25
[6,]   26   27   28   29   30
```

To generate a matrix (n by m) with all zeros in it, we can use the following codes.

```
> x<-matrix(0,5,3)
> x
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
[4,]    0    0    0
[5,]    0    0    0
```

20.9. Adding Column Names Using colnames()

It is a good programming practice to give columns meaningful names. It is quite convenient and sometimes critical especially when we have many columns.

```
> x<-1:5
> y<-rnorm(5) # assume those are returns
> z<-cbind(x,y)
> z
  x y
[1,] 1 0.7108900
[2,] 2 1.2676018
[3,] 3 -0.1431511
[4,] 4 -0.5150289
[5,] 5 1.4828912
> colnames(z)<-c("date","ret")
> z
  date ret
[1,] 1 0.7108900
[2,] 2 1.2676018
[3,] 3 -0.1431511
[4,] 4 -0.5150289
[5,] 5 1.4828912
```

After inputting a set of variables with their names **(header=T)**, we can use the $ sign to refer to one specific column (see an example below). The input file is presented in the right panel.

```
> k<-read.table('clipboard', header=T)
> k
date time
1 1 2
2 3 4
> k$date
[1] 1 3
```

```
date time
1 2
3 4
```

20.10. Getting Specific Rows or Columns from a Matrix

Assume that x is a matrix. It is a good idea that we know the structure of the variable first by using **head()**, **tail()**, **summary()**, **length()**, or **dim()** to acquire basic knowledge related to this specific data set. For a specific column, we specify the second dimension.

```
> y<-x[,3:5] # get columns 3 to 5
```

Similarly we can choose specific rows.

```
> y<-x[1:100,3:5] # rows 1 to 100 for columns 3 to 5
```

20.11. Retrieving a Subset Based on Certain Conditions

Assume that we generate twenty random numbers and choose all positive ones.

```
> set.seed(12345)
> x<-rnorm(20)
> x
 [1] 0.5855288 0.7094660 -0.1093033 -0.4534972 0.6058875
 [6] -1.8179560 0.6300986 -0.2761841 -0.2841597 -0.9193220
[11] -0.1162478 1.8173120 0.3706279 0.5202165 -0.7505320
[16] 0.8168998 -0.8863575 -0.3315776 1.1207127 0.2987237
> y<-x[x>0]
> y
 [1] 0.5855288 0.7094660 0.6058875 0.6300986 1.8173120
 [6] 0.3706279 0.5202165 0.8168998 1.1207127 0.2987237
```

20.12. Row Names

Most of time, we care about the column names only. Occasionally we can use row names to save some space. In the following example, we use row names to separate data for different companies. Alternatively we can add an extracolumn called firm to accomplish the same task.

```
> x<-c(1,2,3,4)
> y<-c(0.23,0.14,-0.11,0.55)
> z<-cbind(x,y)
> z
     x y
[1,] 1 0.23
[2,] 2 0.14
[3,] 3 -0.11
[4,] 4 0.55
> rownames(z)<-c('firm1','firm2','firm3','firm4')
> z
      x y
firm1 1 0.23
firm2 2 0.14
firm3 3 -0.11
firm4 4 0.55
```

20.13. Converting a List into a Matrix

In the following example, we download the financial statement for IBM first then convert it into a matrix.

```
> library(XML)
x<-readHTMLTable("http://www.marketwatch.com/investing/stock/IBM/financials")
typeof(x)
> typeof(x)
[1] "list"
> length(x)
[1] 2
>x1<-as.matrix(x[[1]])
```

20.14. Combining Two Matrices By Row

The example below shows how to use the **rbind()** function to combine rows.

```
> library(XML)
x<-readHTMLTable("http://www.marketwatch.com/investing/stock/IBM/financials")
>x1<-as.matrix(x[[1]])
> dim(x1)
[1] 11 7
>x2<-as.matrix(x[[2]])
> dim(x2)
[1] 46 7
> y<-rbind(x1,x2)
> dim(y)
[1] 57 7
```

Exercises

20.1. When can we use the head() or tail() functions?
20.2. Assuming x is a matrix, how do we view its first twenty lines?
20.3. How can we be sure that x is a matrix?
20.4. Assuming y is a vector, how do we view the first sixteen data records?
20.5. How do we know the type of a variable?
20.6. Assuming there are three columns in x, could it be possible that those three columns have different types?
20.7. Generate random numbers of x and y drawn from a uniform distribution in order to form a twentyby-two matrix. The related random number function is **runif()**.
20.8. Generate 1,000 consecutive integers starting from -20. Convert them into a matrix with 500 by-2 dimensions. How many ways can we form such a matrix?
20.9. How do we add column names to a matrix?
20.10. What is the use of column names?
20.11. How do we add row names? What are their uses?