CHAPTER 18

Simple Value Assignment

In this chapter, we will discuss different ways to assign a value or values to a variable. Again, if you read chapters 1 and 2, you can skip this chapter.

18.1. Several Ways to Assign a Value to a Variable

The simplest way to assign a value to a variable is to use <-.

```
> x<-10
```

To show the value of a variable, simply type its name.

```
> x
[1] 10
```

To assign a value to a variable, we can use = or -> as well.

```
> y=2
> 10->x
```

The -> assignment can make our debugging efforts easier. Assume that we want to test a program to estimate the present value of $100 received in two years with an 8% annual discount rate. We can have the following:

```
> 100/(1+0.08)^2
[1] 85.73388
```

After hitting the Enter key to get our result, we change our mind trying to sign the result to a variable, such as **pv**. To save time, we simply use the up arrow key to recall the previous command. Then we add **->pv** at the end of the above command.

```
> 100/(1+0.08)^2->pv
> pv
[1] 85.73388
```

To assign a set of values, we use **c(1,2.6,4.3,5.25)**, where c stands for "column."

```
# assign a vector (column values)
> X<-c(1,2,4,6)
```

To assign a set of consecutive integers, we can use n1:n2, such as 1:10.

```
> y<-1:50
> x<-c(1:5,8:12)
> x
[1] 1 2 3 4 5 8 9 10 11 12
```

We can input data from high to low (i.e., reverse the order).

```
> y<-5:1
```

The rev() function can be used to reverse an input data set.

```
> x<-5:1
> x<-rev(1:5) # same as above
```

18.2. Viewing Objects Using the ls() Function

We can use the ls() function to list all objectives including variables.

```
> ls() # list all variables
```

18.3. The seq() Function

The seq() function is used to generate a set of values (see an example below).

```
> seq(1, 19, by = 2)
[1] 1 3 5 7 9 11 13 15 17 19
```

The following command uses pi as an incremental value.

```
> seq(1, 11, by = pi)
[1] 1.000000 4.141593 7.283185 10.424778
```

The complete command has the following format:

```
> seq(from=1,to=3, by =0.5)
```

18.4. Position and Keyword Approaches

There are two ways to input data: position and keyword. In the following one-line code, we use the position-variable approach. In other words, the meaning of the input variable depends on its position in the set of input variables.

```
> x<-seq(1,3,0.5) # position variable approach
```

For the keyword approach, we add a keyword in front of each input value, such as from=1. One advantage of the keyword approach is that the order of input variables does not play a role. The following three statements are equivalent.

```
> seq(from=1,to=3,by=0.5) # they are equivalent
> seq(to=3,from=1,by=0.5)
> seq(by=0.5,to=3,from=1)
```

18.5. Inputting Data via scan()

Another easy way to input data from your keyboard is to use the **scan()** function.

```
> x<-scan()
1: 1
2: 3
3: 4
4: 2.5
5: 5
6:
Read 5 items
> x
[1] 1.0 3.0 4.0 2.5 5.0
```

If you plan to input multiple columns, you can input them as a vector first then use the **matrix()** function to convert it to what you need. The desired input format (two columns) is given in the right panel below.

```
> x<-scan()
1: 1 3 3 6 5 6 7 8
9:
Read 8 items
> y<-matrix(x,4,2,byrow=T)
> y
[,1] [,2]
[1,] 1 3
[2,] 3 6
[3,] 5 6
[4,] 7 8
```

```
1 3
3 6
5 6
7 8
```

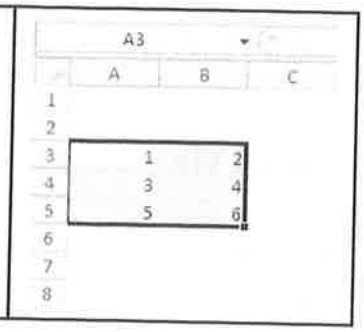
In the above example, we input data according to row. On the other hand, if we input data according to column, we have to change our codes a little bit.

```
> x<-scan()
1: 1 3 5 7 3 6 6 8
9:
Read 8 items
> y<-matrix(x,4,2,byrow=F) # or use default y<-matrix(x,4,2)
```

18.6. Getting Data from an Excel File

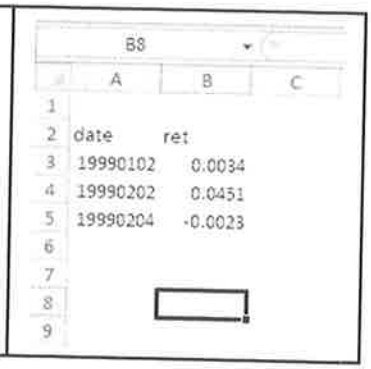
Assume that we have the following Excel spreadsheet (in the right panel below). To input the data into R, we can highlight and copy the data then issue **x<-read.table("clipboard")**.

```
> x<-read.table("clipboard")
> x
  V1 V2
1  1  2
2  3  4
3  5  6
```



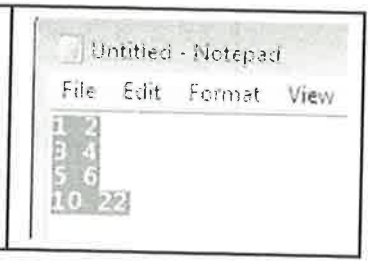
If the data set has headers (column names), just add **header=T**.

```
> x<-read.table("clipboard",header=T)
> x
date ret
1 19990102 0.0034
2 19990202 0.0451
3 19990204 -0.0023
```

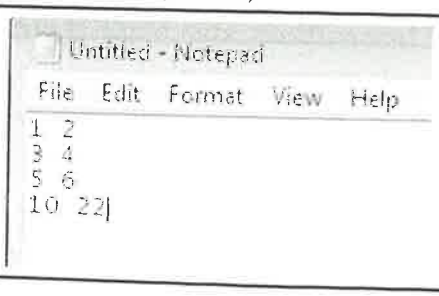
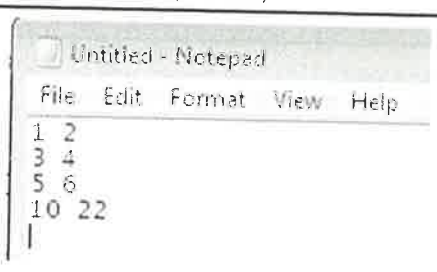


The above example is true when we open a Notepad or an MS Word file (see below).

```
> y<-read.table("clipboard")
```



We should pay attention to the last row, which should be the entry only (see the comparison for the two formats below).

Wrong format (last line)	Correct format (last line)
	

For the format shown in the above left panel, we will get the following warning message when issuing the command **x<-read.table("clipboard")**. Fortunately the variable will take values it is supposed to get.

```
> x<-read.table("clipboard")
Warning message:
In read.table("clipboard"):
 incomplete final line found by readTableHeader on 'clipboard''
```

18.7. Precision of R

Most times, the precision of our R software is not an issue for most researchers for calculation. However, knowing how to find it would be helpful if you have such an issue in the future.

```
> .Machine$double.eps
[1] 2.220446e-16
```

Exercises

18.1. Use the **scan()** function to input twenty pairs of x and y.
18.2. Write an R program to input an Excel data set.
18.3. Input values for x ranging from 1 to 100 and 202 to 300.
18.4. Reverse the input values in 18.3.