R Objects

August 22,2013

Outline

The fundamental data type in R is the vector.

- **Null (empty object): Null**
- **Logical (Boolean): TRUE, FALSE or T, F**
- **Numeric (real number): 1,2,3, pi, 1e-5**
- **Complex(complex number): 2+i, 2i**
- **Character(chain of characters): “ABC”, “hi”**

In order to know the mode of an object, say x, in R we can use
> mode(x)

Example:

```
> x=c(1,2,3,4,5,6)
>  is.null(x)
[1] FALSE
>  is.logical(x)
[1] FALSE
>  is.numeric(x)
[1] TRUE
>  is.complex(x)
[1] FALSE
>  is.complex(x)
[1] FALSE
> is.character(x)
[1] FALSE
```

Numeric Vectors

- Use collect function or concatenate: **c**
- Construction by **sequence operator**
- Construction by **rep function**
- Construction by **scan function**

Example:

```
> x=c(2, 4, 5, 6.8, 7.2) # Numeric vector with 5 entries
> x
[1] 2.0 4.0 5.0 6.8 7.2
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(1,10, by=3) # We can simply use seq(1,10,3)
[1]  1  4  7 10
> seq(1,50,length=8)
[1]  1  8 15 22 29 36 43 50
> rep(1,6)
[1] 1 1 1 1 1 1
> rep(c(1,2,3,4), each=3)
 [1] 1 1 1 2 2 2 3 3 3 4 4 4
> rep(c(1,2,3,4), 3)
 [1] 1 2 3 4 1 2 3 4 1 2 3 4
```

Example: scan

```
> y
[1] 3 5 7
> y<-scan()
1: 2
2: 5
3: 6
4: 7
5: 8
6: 9
7: 9
8:
Read 7 items
> y
[1] 2 5 6 7 8 9 9
```

Character Vectors

It is possible to create character vectors in the same way as the numeric vectors with functions **c** or **rep**

```
> x<-c("A", "BB", "CCC", "Example")
>  x
[1] "A"       "BB"      "CCC"     "Example"
>  y<-rep("A", 10)
> y
 [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
> z<-rep('STAT',8)
> z
 [1] "STAT" "STAT" "STAT" "STAT" "STAT" "STAT" "STAT" "STAT
```

Note that in R " and ' are essentially the same.

Logical Vectors

Boolean vectors are usually generated with logical operations: “$>$”, “$>=$”, “$<$”, “$<=$”, “$==$”, “$!=$” etc.

```
> 1>0
[1] TRUE
> 2<=5
[1] TRUE
> 2>5
[1] FALSE
> 2>=5
[1] FALSE
> 7!=10
[1] TRUE
```

Using all() and any()

```
> x <- 1:10
> any(x > 8)
[1] TRUE
> any(x > 88)
[1] FALSE
> all(x > 88)
[1] FALSE
> all(x > 0)
[1] TRUE
```

which() and match()

```
> x<-c(4,7,2,12,9,0)
> y <- rep(1:5, times=5:1)
> x
[1]  4  7  2 12  9  0
> y
 [1] 1 1 1 1 1 2 2 2 2 3 3 3 4 4 5
> which(x>4)
[1] 2 4 5
> which.max(x) # Location of the maximum x value
[1] 4
> which.min(x)# Location of the minimum x value
[1] 6
> x[which.max(x)] # Value of the maximum x value
[1] 12
> match(1:5,y)# It matches the locations
[1]  1  6 10 13 15
```

Selecting Part of a Vector

Selections are made using the selection operator [ ] and a selection vector:> x[indexvector]

```
> x=c(2,4,6,8,4,6,7,8,9,0,12,13,14,15)
> x
 [1]  2  4  6  8  4  6  7  8  9  0 12 13 14 15
> x[5]
[1] 4
> x[-5]
 [1]  2  4  6  8  6  7  8  9  0 12 13 14 15
> v=1:15
> v
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> v[(v<4)|(v>12)] # | means OR
[1]  1  2  3 13 14 15
> v[(v<4)&(v>12)] # & means AND
integer(0)
```

which() and match()

- **which()** Indices of a vector where the condition is TRUE
- **which.max()** Location of the maximum element of a numeric vector
- **which.min()** Location of the minimum element of a numeric vector
- **match()** First position of an element in a vector

```
> x<-c(4,7,2,12,9,0)
> which(x>4)
[1] 2 4 5
> which.max(x)
[1] 4
> x[which.max(x)] # Exact value of the maximum value
[1] 12
> y <- rep(1:5, times=5:1)
> match(1:5,y)
[1]  1  6 10 13 15
```

Testing Vector Equality

```
> x <- 1:3
> y <- c(1,3,4)
> x = = y
[1] TRUE FALSE FALSE
> identical(x,y)
[1] FALSE
```

Note that “:” produces integers while “c()” produces floating-point numbers.

```
> x=c(1,2,3); y=1:3
[1] 1 2 3
[1] 1 2 3
> identical(x,y)
[1] FALSE
> typeof(x)
[1]  "double" # Double is same as numeric
> typeof(y)
[1] "integer"
```

Missing Value

For a number of reasons, certain elements of data may not be collected during and experiment or study. In R, missing values are represented by the symbol **NA** (not available) but impossible values (e.g., dividing by zero) are represented by the symbol **NaN** (not a number) and **Inf** for infinity.

```
> var(5) # Variance of a single observation
[1] NA
> log(-2)
[1] NaN
Warning message:
In log(-2) : NaNs produced
> exp(1e10)
[1] Inf
```

Detecting Missing Value

```
> x<-c(1,2,3,4,5,6,7,8,4,5,7)
> is.na(x)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> y<-c(1,4,NA,6,9,NA,7)
> is.na(y)
[1] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
>y<-c(1,4,NA,6,9,NA,7)
> any(is.na(y)) # checking if there is any missing value
[1] TRUE
>which (is.na(y)) # Which one is NA?
[1] 3  6      # Answer: the third one and sixth one
```

Dealing with Missing Value

The user can specify if **NA** should be last or first in a sorted order by indicating TRUE or FALSE for the **na.last** argument.

```
>y<-c(1,4,NA,6,9,NA,7)
> sort(y, na.last = TRUE)
[1]  1  4  6  7  9 NA NA
```

na.omit and **na.exclude** returns the object with observations removed if they contain any missing values

```
> x <- c(88,NA,12,168,13)
> x
[1] 88 NA 12 168 13
> mean(x)
[1] NA
> mean(x, na.rm=T)
[1] 70.25
> mean(na.omit(x))
[1] 70.25
> mean(na.exclude(x))
[1] 70.25
```

NA Vs. NULL

Once can use “NULL” if the values really are counted as nonexistent

```
> x <- c(88,NULL,12,168,13) # R has skipped the 2nd observa
> mean(x)
[1] 70.25
> u <- NULL
> length(u)
[1] 0
> v <- NA
> length(v)
[1] 1
```