Working with Vectors and Matrices

INT232

Working with Vectors

Suppose **'x'** is a vector and **'df'** is a data frame

```
X=c(a=1, b=2)
df=data.frame(x=1:3,y=5:7)
is.vector(x)                    # TRUE
as.vector(x)                    #1    2

is.list(df)                     # TRUE
is.data.frame(df)               # TRUE
!is.vector(df)                  # TRUE
```

If we want to create a vector of consecutive numbers, the `:` operator is very helpful.

Example 1: Creating a vector using : operator

```
> x <- 1:7; x
[1] 1 2 3 4 5 6 7
> y <- 2:-2; y
[1]  2  1  0 -1 -2
```

More complex sequences can be created using the `seq()` function, like defining number of points in an interval, or the step size.

Example 2: Creating a vector using seq() function

```
> seq(1, 3, by=0.2)             # specify step size
[1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
> seq(1, 5, length.out=4)     # specify length of the vector
[1] 1.000000 2.333333 3.666667 5.000000
```

How to access Elements of a Vector?

Elements of a vector can be accessed using vector indexing. The vector used for indexing can be logical, integer or character vector.

Using integer vector as index

Vector index in R starts from 1, unlike most programming languages where index start from 0.

We can use a vector of integers as index to access specific elements.

We can also use negative integers to return all elements except that those specified.

But we cannot mix positive and negative integers while indexing and real numbers, if used, are truncated to integers.

```
> x
[1]  0  2  4  6  8 10
> x[3]              # access 3rd element
[1] 4
> x[c(2, 4)]      # access 2nd and 4th element
[1] 2 6
> x[-1]             # access all but 1st element
[1]  2  4  6  8 10
> x[c(2, -4)]     # cannot mix positive and negative integers
Error in x[c(2, -4)] : only 0's may be mixed with negative subscripts
> x[c(2.4, 3.54)]     # real numbers are truncated to integers
[1] 2 4
```

How to modify a vector in R?

We can modify a vector using the assignment operator.

We can use the techniques discussed above to access specific elements and modify them.

If we want to truncate the elements, we can use reassignments.

```
> x
[1] -3 -2 -1  0  1  2
> x[2] <- 0; x          # modify 2nd element
[1] -3  0 -1  0  1  2
> x[x<0] <- 5; x    # modify elements less than 0
[1] 5 0 5 0 1 2
> x <- x[1:4]; x        # truncate x to first 4 elements
[1] 5 0 5 0
```

How to delete a Vector?

We can delete a vector by simply assigning a `NULL` to it.

```
> x
[1] -3 -2 -1  0  1  2
> x <- NULL
> x
NULL
> x[4]
NULL
```

Working with matrices

A **matrix** is a collection of data elements arranged in a two-dimensional rectangular layout. The following is an example of a matrix with 2 rows and 3 columns.

$$A = \begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

```
> A = matrix(
+   c(2, 4, 3, 1, 5, 7), # the data elements
+   nrow=2,              # number of rows
+   ncol=3,              # number of columns
+   byrow = TRUE)        # fill matrix by rows

> A                      # print the matrix
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

An element at the m^{th} row, n^{th} column of A can be accessed by the expression A[m, n].

```
> A[2, 3]      # element at 2nd row, 3rd column
[1] 7
```

The entire m^{th} row A can be extracted as A[m,].

```
> A[2, ]       # the 2nd row
[1] 1 5 7
```

Similarly, the entire n^{th} column A can be extracted as A[,n].

```
> A[ ,3]       # the 3rd column
[1] 3 7
```

We can also extract more than one rows or columns at a time.

```
> A[ ,c(1,3)]  # the 1st and 3rd columns
     [,1] [,2]
[1,]    2    3
[2,]    1    7
```

If we assign names to the rows and columns of the matrix, than we can access the elements by names.

```
> dimnames(A) = list(
+   c("row1", "row2"),         # row names
+   c("col1", "col2", "col3")) # column names

> A                  # print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7

> A["row2", "col3"] # element at 2nd row, 3rd column
[1] 7
```