Vectors, Matrices and Dataframes

Hamid Abdulsalam

vector

Vectors are one-dimensional ordered collections of values that are all stored in a single variable. Each value in the vector is referred to as element in that Vector.

The function c() is used in creating vector

```
people <- c("Sarah", "Amit", "Zhang")
people
```

```
## [1] "Sarah" "Amit"  "Zhang"
```

```
numbers <- c(1, 2, 3, 4, 5)
numbers
```

```
## [1] 1 2 3 4 5
```

Vectors

Using the length() function to determine the number of elements in a vector

```r
length(people)
```

```
## [1] 3
```

```r
length(numbers)
```

```
## [1] 5
```

There are other handy ways to create vectors. For example, the seq() function

Vectors

```r
# Make vector of numbers 1 to 90
one_to_ninety <- seq(1, 90)
print(one_to_ninety)  # [1] 1 2 3 4 5 ...
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
## [51] 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
## [76] 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
```

```r
# Make vector of numbers 1 to 10, counting by 2
odds <- seq(1, 10, 2)
print(odds)  # [1] 1 3 5 7 9
```

```
## [1] 1 3 5 7 9
```

Vectors of different Objects

```r
x <- c(1, 4, 6.5, 4.5) ## numeric
x <- c(TRUE, FALSE) ## logical
x <- c(T, F) ## logical
x <- c("c", "d", "e") ## character
x <- 19:40 ## integer
x <- c(1+0i, 2+4i) ## complex
```

Vectors

Another useful function that creates vectors is rep() that repeats it's first argument:

```r
rep("James", 5)  # Repeat James 5 times
```

```
## [1] "James" "James" "James" "James" "James"
```

Objects Mixing in R

As atomic vectors can only contain same type of elements

```r
y <- c(16, "a") ## character
y <- c(FALSE, 2) ## numeric
y <- c("a", TRUE) ## character
```

There is always coercion when objects are mixed in R. For instance, object mixed with a character becomes a character class. Logical objects mixed with a numeric object becomes a numeric object.

Explicit Coercion in R

Objects can be explicitly coerced from one class to another using the as.* functions, if available.

```r
x <- 0:6
as.numeric(x)
```

```
## [1] 0 1 2 3 4 5 6
```

```r
as.logical(x)
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```r
as.character(x)
```

```
## [1] "0" "1" "2" "3" "4" "5" "6"
```

Vector Indices

Using the bracket notation, we can retrieve the elements in a vector

```
state<-c("Florida", "New York", "California",
         "New Jersey")
# access the element at index 1
state[1]
```

```
## [1] "Florida"
```

```
# access the element at index 2
state[2]
```

```
## [1] "New York"
```

Vector Indices

```r
# You can also use variables inside the brackets
last_index <- length(state)

# last index is the length of the vector!
state[last_index]  # returns "New Jersey"
```

```
## [1] "New Jersey"
```

Multiple Indices

You can extract multiple elements from a vector in R using Multiple indices.

```r
# Create a ‘colors‘ vector
colors <- c("red", "green", "blue", "yellow", "purple")

# Vector of indices to extract and Retrieve the colors
indices <- c(1, 3, 4)
colors[indices]
```

```
## [1] "red"    "blue"   "yellow"
```

```r
# Specify the index array anonymously
colors[c(2, 5)]
```

```
## [1] "green"  "purple"
```

Matrices

Matrices in R are created through the function **matrix** or by combining multiple vectors through rows or columns.

```r
x <- matrix(nrow = 2, ncol = 3)
x
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
```

```r
dim(x)
```

```
## [1] 2 3
```

Matrices

```
attributes(x)
```

```
## $dim
## [1] 2 3
```

Matrices can be constructed by column-wise or by row-wise. By default, R construct matrices by Column-wise, but if you want the matrix to be constructed by row-wise, you need to inform R.

```
m <- matrix(1:6, nrow = 2, ncol = 3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Matrices

By row-wise Matrix

```
z <- matrix(1:6, nrow = 2, ncol = 3, byrow=TRUE)
z
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

We can also create matrix through vector by adding the dimension attribute

```
x <- c(1,2,3,4,5,6,7,8,9,10)
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Matrices

Using dim to create a matrix

```
dim(x) <- c(2, 5)
x
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

We can also create matrices in R through column-binding or row-binding with functions **cbind()** and **rbind()**

```
x <- 4:6
y <- 1:3
```

Matrices

```
cbind(x, y)
```

```
##      x y
## [1,] 4 1
## [2,] 5 2
## [3,] 6 3
```

```
rbind(x, y)
```

```
##   [,1] [,2] [,3]
## x    4    5    6
## y    1    2    3
```

Matrices

We can name the rows and columns of matrices in R using **colnames** or **rownames**

```r
m <- matrix(1:4, nrow = 2, ncol = 2)
colnames(m) <- c("a", "b")
rownames(m) <- c("c", "d")
m
```

```
##   a b
## c 1 3
## d 2 4
```

Excercise 1

- Create a vector X containing values 1:4 and another vector Y containing values 5:8.
- Use the cbind function to turn it to matrix called M.
- Also use the rbind function to turn it to another matrix call Q.
- Use the colnames and rownames function to name the row and column of the two matrices.

List

List can be regarded as vector but a special type of vector that can contain elements of different classes.We create a list in R by using the **list** function.

```
m <- list(50, "b", TRUE, 1 + 4i)
```

List

```
m
```

```
## [[1]]
## [1] 50
##
## [[2]]
## [1] "b"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+4i
```

List

Lists can also have names.

```
x <- list(a = 1, b = "c", c = 3)
x
```

```
## $a
## [1] 1
##
## $b
## [1] "c"
##
## $c
## [1] 3
```

Factors

Most times, as a data scientist. You will come across data that contains categorical data. R represent categorical data as Factors. They can be unordered or ordered.

In modelling, factors are treated specially by modelling functions such as lm() and glm()

- As a data analyst or data scientist, it is always good to use factors with labels than using integers because factors are self-describing; having a variable that has values "Male" and "Female" is better than a variable that has values 1 and 2.

Factors

We create factors in R using the function factor

```
x <- factor(c("yes", "yes", "no", "yes", "no"))
x
```

```
## [1] yes yes no  yes no
## Levels: no yes
```

```
 table(x)
```

```
## x
##  no yes
##   2   3
```

```
class(x)
```

```
## [1] "factor"
```

Missing Values

R denote Missing values by NA It also denote NaN for undefined mathematical operations. ▪ is.na() is used to test objects if they are NA ▪ is.nan() is used to test for NaN

Missing Values

```r
x <- c(1, 2, NA, 10, 3)
is.na(x)
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE
```

```r
x <- c(1, 2, NaN, 20, 4)
is.nan(x)
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE
```

Data Frames

Data frames are used to store tabular data

- Unlike matrices, data frames can store different classes of objects in each column (just like lists); matrices must have every element be the same class
- Data frames are usually created by calling read.table() or read.csv()
- Can be converted to a matrix by calling data.matrix()

Data Frames

We create data frame in R using function data.frame in R

```
x <- data.frame(num = 1:4, log = c(T, T, F, F))
x
```

```
##   num   log
## 1   1  TRUE
## 2   2  TRUE
## 3   3 FALSE
## 4   4 FALSE
```

Data Frames

Checking numbers of rows and columns

```
nrow(x)
```

```
## [1] 4
```

```
ncol
```

```
## function (x)
## dim(x)[2L]
## <bytecode: 0x0000000013a63310>
## <environment: namespace:base>
```

Subsetting

Subsetting is an important component of R. It allows you to be able to extract any element from a dataframe. There are a number of operators that can be used to extract subsets of R objects. - [can be used to select one or more than one element in a vector, list, matrix or dataframe

- [[is used to extract elements of a list or a data frame; it can only be used to extract a single element and the class of the returned object will not necessarily be a list or data frame
- $ is used to extract elements of a list or data frame by name; semantics are similar to that of [[.

Subsetting

```r
x <- c("a", "b", "c", "d")
x[1]
```

```
## [1] "a"
```

```r
x[2]
```

```
## [1] "b"
```

```r
x[1:4]
```

```
## [1] "a" "b" "c" "d"
```

```r
x[x > "b"]
```

```
## [1] "c" "d"
```

Subsetting a Matrix

Matrices in R are usually subsetted using the indices type (i,j)

```
x <- matrix(1:6, 2, 3)
```

Subsetting a Matrix

```
x[1, 2]
```

```
## [1] 3
```

```
x[2, 1]
```

```
## [1] 2
```

Indices can also be missing.

```
 x[1, ]
```

```
## [1] 1 3 5
```

```
x[, 2]
```

```
## [1] 3 4
```

Subsetting a Matrix

We can also subset a range of rows and columns from a matrix

```r
x <- matrix(1:12, 3, 4)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```r
x[2:3,2:4]
```

```
##      [,1] [,2] [,3]
## [1,]    5    8   11
## [2,]    6    9   12
```

Task 2

- Create a matrix Y with 4 rows and 4 columns.
- Fill it with integers starting from number 1 to 16
- Extract column range 3:4 and row range 1:3