Intro to Data Visualization and Statistics in R

Session #3 extra

Genome Institute of Singapore

18th October 2019

Data types | Overview

Here are the generic data types in R.

- Scalar
- Vector
- Factor
- Matrix
- **Data Frames**
- Lists

Data types | Overview

Every data type have the following features:

- way to create it
- special operations
- attributes
- way to extract elements from it

Phone contact example

Insert here

SCALARS

Scalar | Definition and create

Assign a value with a name so that it can be re-used it later.

```
x <- 100
(x + 1)^2 / x
## [1] 102.01
```

The assignment operator is "<-".

Scalar | Definition and create

You can also assign a character using quotes:

```
x <- "100"
```

(Note: re-assigning to `x` destroys the previous value)

In R, you cannot apply a mathematical function on a character.

```
x + 1
# Error in y + 1 : non-numeric argument to binary operator
```

i.e. Numeric and character value are not identical.

VECTORS

Vectors | Definition

A vector is a collection of numeric values or character values (never a mix of both).

Three types:

- numeric vectors
- character vectors
- logical vectors

Optionally, a vector can have names.

Numeric vector | Create

Let us generate a numeric vector:

```
x <- c(1, 2, 3, 4, 5)
x
## [1] 1 2 3 4 5
```

The small letter c is a function to ***c***ombine values into a vector.

Numeric vector | Create

Generating a sequence of numbers is a fairly common task. You can use the colon operator (`:`) or `seq()` for this:

```
1:5                         ## increments of 1 only
## [1] 1 2 3 4 5

seq(1, 5, by=1)             ## same as 1:5
## [1] 1 2 3 4 5

seq(-100, 100, by=25) ## more generalized form
## [1] -100  -75  -50  -25    0   25   50   75  100
```

You can check if the two vectors are the same using:

```
x <- c(1, 2, 3, 4, 5)
y <- 1:5
x == y                      ## Check element by element
## [1] TRUE TRUE TRUE TRUE TRUE
```

Numeric vector | Operations

Many functions in R are vectorized. This means the function is applied to every element of a vector without an explicit `for()` loop.

```
x + 100
## [1] 101 102 103 104 105

y^2
## [1]  1  4  9 16 25

x + 100 + y^2          ## Adds element by element
## [1] 102 106 112 120 130

x < 3
## [1]  TRUE  TRUE FALSE FALSE FALSE
```

Vectorized operations are much more compact to code and usually computationally more efficient.

Numeric vector | Re-arranging

```
r <- c(3.3, 1.1, 4.4, 5.5, 2.2)

sort(r)                    ## Sort smallest to largest
## [1] 1.1 2.2 3.3 4.4 5.5

rank(r)                    ## Smallest value is ranked 1
## [1] 3 1 4 5 2

sort(r, decreasing=T) ## Sort largest to smallest
## [1] 5.5 4.4 3.3 2.2 1.1

rank(-r)                   ## Largest value is ranked 1
## [1] 3 5 2 1 4

rev(r)                     ## Reverse (N, N-1, ..., 2, 1)
## [1] 2.2 5.5 4.4 1.1 3.3
```

Numeric vector | Summary

You can summarize a numeric vector in different ways. E.g.

```
x <- seq(0, 500, by=100)
x
## [1]   0 100 200 300 400 500

length(x)
## [1] 6

mean(x)
## [1] 250

var(x)
## [1] 35000

summary(x)  ## Demo only. Not meaningful with few values.
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0     125     250     250     375     500
```

Character vector | Creation

You can create a character vector by quoting the values:

```
y <- c("G", "C", "T", "A")
y
## [1] "G" "C" "T" "A"
```

Arithmetic operations (sum, mean, var, summary) will not work. However, the sort and rank will work alphabetically.

```
length(y)
## [1] 4

sort(y)
## [1] "A" "C" "G" "T"

rank(y)
## [1] 3 2 4 1
```

Character vector | Comparison with a numeric vector

Quoting a number makes them characters which renders them not viable for arithmetic operations:

```
y <- c("1", "2", "3", "4", "5")
y
## [1] "1" "2" "3" "4" "5"
```

You can convert this back to a numeric vector

```
y <- as.numeric(y)
y + 100
## [1] 101 102 103 104 105
```

Character vector | Always character or always numeric

Creating a vector with a mix of numeric and character will turn it to a character vector instead.

```
y <- c(1, 2, 3, "D", "E")
y
## [1] "1" "2" "3" "D" "E"
```

Trying to convert this to a numeric vector will only convert quoted the numbers. The characters will be turning into missing values (encoded as `NA` in R).

```
as.numeric(y)
## Warning: NAs introduced by coercion
## [1]  1  2  3 NA NA
```

Logical vectors | Definition and creation

This is a vector of TRUE and FALSE. The `which()` function pulls the **index** of the TRUE elements.

```
y <- c(TRUE, FALSE, TRUE, FALSE, TRUE)
y
## [1]  TRUE FALSE  TRUE FALSE  TRUE
which(y)
## [1] 1 3 5
```

Logical vectors can be created when you make a comparison.

```
x <- seq(-50, 50, by=25)
x
## [1] -50 -25   0  25  50
neg <- ( x < 0 )
neg
## [1]  TRUE  TRUE FALSE FALSE FALSE
which(neg)
## [1] 1 2
```

Vector | One element vector reduces to scalar

A vector with only one element reduces to a scalar

```
c(1)
## [1] 1

c("G")
## [1] "G"

c(TRUE)
## [1] TRUE
```

This concept of reduction is useful to know. Likewise, pulling a column from matrix or dataframe will reduce it to a vector.

Vector | Naming

Let's create a vector to hold the number of days in a non-leap year.

```
nDays <- c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
nDays
##  [1] 31 28 31 30 31 30 31 31 30 31 30 31
```

It is difficult to see which month has how many days in this numeric vector. Let's set the `names()` attribute.

Vector | Naming

Let's create a vector to hold the number of days in a non-leap year.

```
nDays <- c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
nDays
##  [1] 31 28 31 30 31 30 31 31 30 31 30 31
```

It is difficult to see which month has how many days in this numeric vector. Let's set the `names()` attribute.

```
mths <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
          "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
names(nDays) <- mths
nDays
## Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
##  31  28  31  30  31  30  31  31  30  31  30  31
```

Using names makes it easier to read. We can also extract specific elements using their name (coming up in the next few slides).

Creating a named vector

You can choose to create and name a vector simultaneously:

```
nDays <- c(Jan=31, Feb=28, Mar=31, Apr=30, May=31, Jun=30,
           Jul=31, Aug=31, Sep=30, Oct=31, Nov=30, Dec=31)

nDays
## Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
##  31  28  31  30  31  30  31  31  30  31  30  31
```

The quote around the month names is not necessary when creating named vectors unless it contains special characters (e.g. "Jan-2019", "Jan 2019"). Also, you can extract the names via:

```
names(nDays)
## [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun"
## [7] "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

Extracting from a vector

Several ways of extracting element(s) from a vector using a square bracket and one of the following:

1) position index. e.g. x[1]

2) position index but with a negative sign to drop elements. e.g. x[-1]

3) name if the vector is named. e.g. nDays[“Jan”]

You cannot combine a negative with a name. i.e. nDays[-“Jan”] will give an error.

Extracting a single element from a vector

```
nDays
## Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
##  31  28  31  30  31  30  31  31  30  31  30  31

nDays[ 1 ]   ## Select only the first element
## Jan
##  31
nDays[ -1 ]  ## Select everything but the first element
## Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
##  28  31  30  31  30  31  31  30  31  30  31

nDays[ "Jan" ]
## Jan
##  31

keep <- "Jan";  nDays[ keep ]  ## using variable
## Jan
##  31
```

Extracting multiple elements from a vector

You can use a vector of positions to extract specific elements:

```
x <- c("e", "h", "l", "o", "w", "r", "d")
x[ c(4, 1) ]  # Extract the 4th and 1st element
## [1] "o" "e"

y <- c(2, 1, 3, 3, 4)
x[y]          # Extract using another vector
## [1] "h" "e" "l" "l" "o"

paste(x[y], collapse="") # collapse to a scalar string
## [1] "hello"
```

Note:

1) The elements are returned in order requested
2) You can request the same index several times (e.g. here we requested 3 twice to get "ll")

Extracting multiple elements from a vector

You can use the negative sign to drop some elements from the vector:

```
x
## [1] "e" "h" "l" "o" "w" "r" "d"

y
## [1] 2 1 3 3 4

x[-y]
## [1] "w" "r" "d"
```

Extracting elements using logic operators

```
nDays
# Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
#  31  28  31  30  31  30  31  31  30  31  30  31

nDays==30
##   Jan   Feb   Mar   Apr   May  Jun
## FALSE FALSE FALSE  TRUE FALSE TRUE
##   Jul   Aug   Sep   Oct   Nov   Dec
## FALSE FALSE  TRUE FALSE  TRUE FALSE

which(nDays==30)                # position of matches
## Apr Jun Sep Nov
##   4   6   9  11

names(which(nDays==30))     # get the names instead
## [1] "Apr" "Jun" "Sep" "Nov"

## or longer: names(nDays)[ which(nDays==30) ]
```

Requesting elements that do not exist

You get missing value (NA) if you request something non-existent.

```
nDays
## Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
##  31  28  31  30  31  30  31  31  30  31  30  31

nDays[ "June" ]    # Should be Jun not June
## <NA>
##   NA

nDays[ 13 ]        # ghost month?
## <NA>
##   NA

nDays[ c("May", "June", "July") ]  # May is found
##  May <NA> <NA>
##   31   NA   NA
```

Aside | Set operators on vectors

Set operators can be useful when finding elements that are common/unique to each set. Example:

```
left  <- c("A", "A", "B", "C")
right <- c("B", "C", "C", "C", "E", "D")

setdiff(left, right)    # Repeated items in a set ignored
## [1] "A"

setdiff(right, left)
## [1] "E" "D"

intersect(left, right) # same as intersect(right, left)
## [1] "B" "C"

union(left, right)      # same as union(right, left)
## [1] "A" "B" "C" "E" "D"
```

FACTORS

Factors | Definition

A factor is simply a vector where the value of the elements are limited and have an inherent order (set by the levels argument).

There are two main use cases for factors:

- order the output of a table or graph (e.g. axis order, facet)
- in a linear model, the effect of a categorical variable is reported with respect to a reference value

Factors | Motivation

Here are the WHO classification of BMI for 5 people:

```
BMI_class <- c("over", "normal", "obese", "over", "over")
```

Table gives the counts but in alphabetical order which is messy.

```
table(BMI_class)
## BMI_class
## normal  obese   over
##      1      1      3
```

Factors | Create

Let's convert it to a factor and add in an extra level for future use.

```
levs <- c("under", "normal", "over", "obese")

BMI_class <- factor(BMI_class, levels=levs)

table(BMI_class)     ## Columns are ordered correctly
## BMI_class
##  under normal   over  obese
##      0      1      3      1

levels(BMI_class)    ## Levels of the factor
## [1] "under"  "normal" "over"   "obese"

nlevels(BMI_class)   ## Number of levels
## [1] 4
```

MATRIX

Matrix | Definition

A matrix is a collection of data elements

- arranged in a two-dimensional rectangular layout with rows and columns.
- each column is a vector.
- all elements will be either numeric or characters (never a mix of both).

Optionally, a matrix can have row names and/or column names.

Matrix | Create

A matrix is usually created from a dataframe (either after subsetting or coersion) - we will see how to do this with the iris data later.

You can also explicitly create it manually

AA

cor(iris[, -5])

t()

. . . . . .

	$Sample_1$	$Sample_2$	$Sample_3$	$Sample_4$	$Sample_5$
$Gene_1$	12.5	8.1	11.7	8.1	11.5
$Gene_2$	2.9	3.1	3.7	3.2	2.8
. . .	. . .	. . .	. . .	. . .	. . .
$Gene_N$	7.6	7.8	7.7	7.7	7.9

All entries will be numeric ***or*** character, but not a mix of both.

Data types | Data Frames

Same 2-dimensional structure as a matrix but it can be a mix of character columns and numeric columns.

	Age	Sex	Status
Subject_1	55	Male	Healthy
Subject_2	49	Female	Cancer
Subject_3	80	Female	Health
...	...	...	...

Data import functions store in a data frame e.g. `read_excel()`, `read_csv()`, `read_delim()`

Data types | List

A list contain elements of other data types. Here is how to create one:

```
experiment <- list(

  Requester = "Meng How, Tan",

  Library_Kit = "Illumina TruSeq",

  Read = c(Length="150", type="paired-end"),

  out = data.frame(ID      = c("MUX1", "MUX2", "MUX3"),
                   Machine = c("HS08", "HS08", "HS06"),
                   Date    = c("1Jun", "2Jun", "1Jun"))
)
```

Data types | List

```
experiment
## $Requester
## [1] "Meng How, Tan"
##
## $Library_Kit
## [1] "Illumina TruSeq"
##
## $Read
##       Length          type
##        "150" "paired-end"
##
## $out
##     ID Machine Date
## 1 MUX1    HS08 1Jun
## 2 MUX2    HS08 2Jun
## 3 MUX3    HS06 1Jun
```