REVOLUTION ANALYTICS

Introduction to R Data Structures: Vectors to data frames

Revolution Analytics

Overview

In this session you establish a foundation for R, by getting to know about:

- Data structures
  - vectors, arrays, lists, and data frames
  - factors
- Operations on data, including subsetting
- Calculations on vectors

Outline

Vectors

A vector in R is the simplest type of object.

Vectors always have a single `mode`:

- `logical`
- `integer`
- `numeric`, synonym with `double`
- `character`
- `complex`
- `raw`

For help, see `?vector`

Getting started with vectors

```
1:10
```

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

```
(1:10)^2
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
length(month.name)
```

```
## [1] 12
```

Assignment

Assign an object with the operator <− (pronounced as “gets” or “gets the value”)

```
x <- 1:10
x
```

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

```
sum(x)
```

```
## [1] 55
```

```
class(x)
```

```
## [1] "integer"
```

Combining vectors

Combining vectors: you use `c()` to combine vectors

```
x <- c(1, 1, 2, 3, 5, 8)
sum(x)
```

```
## [1] 20
```

```
y <- c(x, 13, 21)
y
```

```
## [1]  1  1  2  3  5  8 13 21
```

Character vectors

You create character vectors in a similar way to numeric vectors:

```
authors <- c("Ross", "Robert")
authors
```

```
## [1] "Ross"   "Robert"
```

```
length(authors)
```

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

Pasting and concatenation of vectors

You use the function `paste()` to concatenate strings:

```
paste(authors, c("Ihaka", "Gentleman"))
```

```
## [1] "Ross Ihaka"       "Robert Gentleman"
```

```
paste("row", 1:5, sep = "_")
```

```
## [1] "row_1" "row_2" "row_3" "row_4" "row_5"
```

paste0

A convenience function

```
paste("row", 1:5)
```

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

```
paste0("row", 1:5)
```

```
## [1] "row1" "row2" "row3" "row4" "row5"
```

Logical vectors

The elemens of logical vectors take the value `TRUE` or `FALSE`

```
authors == "Ross"
```

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

```
1:10 > 7
```

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

```
sum(1:10 > 7)
```

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

Outline

Subsetting vectors

Use `[]` to extract subsets of vectors and matrixes:

```
letters[5]
```

```
## [1] "e"
```

```
letters[1:5]
```

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

```
letters[c(1, 3, 5)]
```

```
## [1] "a" "c" "e"
```

Subsetting vectors 2

Use negative integers to drop elements

```
letters[-1]
```

```
##  [1] "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r"
## [18] "s" "t" "u" "v" "w" "x" "y" "z"
```

```
letters[-(1:5)]
```

```
##  [1] "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v"
## [18] "w" "x" "y" "z"
```

Subsetting vectors 3

Convenience functions

```
head(letters)
```

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

```
tail(letters)
```

```
## [1] "u" "v" "w" "x" "y" "z"
```

Named vectors

Each element in a vector can have an associated name.

```
x <- 1:5
names(x) <- letters[1:5]
x
```

```
## a b c d e
## 1 2 3 4 5
```

Subset by names

You can subset by names too.

```
x["c"]
```

```
## c
## 3
```

```
x[c("c", "d")]
```

```
## c d
## 3 4
```

Summary of subsetting rules

Syntax	Rule	Effect
[1:3]	Positive integer	Return elements at these positions
[-1]	Negative integer	Drop elements at these positions
[“a”]	Character	Return named elements
[]	Empty	Return entire vector
[NULL]	Null	Empty vector

Vector Summary

- Simplest Object type
- Every element has same `mode`
- Subset via `[]`
  - positive integer
  - negative integer
  - name

Questions?

Outline

Factors

Factors are a type of 1-dimensional data structure that represents categorical variables.

The have both numeric and character qualities.

Factors 2

Factors are useful for calculations on *categorical* variables, e.g. gender, treatment status, or group label.

```
mtcars$cyl
```

```
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```

```
as.factor(mtcars$cyl)
```

```
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
## Levels: 4 6 8
```

Factor Properties

Numeric property The index of the unique level the observation represents

Character property The `label` of the unique level

Why use factors

- Efficient storage of character values
  - Each unique character value is stored just once as the label.
  - The data itself is stored internally as a vector of integers.
- Advantages in working with modeling and graphing functions

Caveat: When working with factors, you need to keep in mind their unique structure.

Outline

More Complex Data Structures

- Matrices
- Arrays
- Lists
- Data frames

Creating matrices

In R, a `matrix` is a 2-dimensional array. An object of class `array` is similar, but typically has more than 2 dimensions.

Use the function `matrix()` to create a matrix

```
matrix(1:9, nrow = 3, ncol = 3)
```

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

Creating matrices 2

R will guess `ncol` if you leave the argument out…

```
mat <- matrix(1:25, nrow = 5)
mat
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    3    8   13   18   23
## [4,]    4    9   14   19   24
## [5,]    5   10   15   20   25
```

Subsetting matrices

Two positions: `[rows, columns]`

```
mat[2:4, 3:5]
```

```
##      [,1] [,2] [,3]
## [1,]   12   17   22
## [2,]   13   18   23
## [3,]   14   19   24
```

Remember that the operator `:` generates an integer sequence.

Subsetting matrices 2

Negative subscripts work the same way

```
mat[-3, -4]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   21
## [2,]    2    7   12   22
## [3,]    4    9   14   24
## [4,]    5   10   15   25
```

Subsetting matrices 3

An empty position indicates that you want to retain *all* of that dimension

```
mat[-3, ]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    4    9   14   19   24
## [4,]    5   10   15   20   25
```

Outline

Arrays

Arrays can have many dimensions.

You create an array with the `array()` function.

```
array.one <- array(1:12, dim = c(3, 4))
array.two <- array(1:24, dim = c(3, 4, 2))
```

Subsetting arrays

An n-dimensional array can be accessed just like a matrix. There are now just `n` positions.

```
array.two[, , 1]
```

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

Row and column names

You assign row and column names using `rownames()` and `colnames()`.

```
colnames(mat) <- paste("col", 1:5, sep = "")
rownames(mat) <- paste0("row", 1:5)
mat
```

```
##      col1 col2 col3 col4 col5
## row1    1    6   11   16   21
## row2    2    7   12   17   22
## row3    3    8   13   18   23
## row4    4    9   14   19   24
## row5    5   10   15   20   25
```

Subsetting with row and column names

Since the row and column names are named vectors, you can subset using the names:

```
mat[c("row2", "row3"), c("col1", "col5")]
```

```
##      col1 col5
## row2    2   22
## row3    3   23
```

Exercise:

Your turn: Explore basic types

- Create a vector containing values `1 - 10` and `21 - 35`
- Change this vector into a 5 x 5 matrix
- Extract a 3 x 3 subset from that matrix
- Assign row and column names to the matrix.
  - Hint: use the `rownames()` and `colnames()` functions

Outline

Lists

You can combine vectors of different class with `list()`. Lists are a very important class in R and form the basis of data frames.

```
xl <- list(num = 1:10, char = letters[1:15], log = rep(c(TRUE, FALSE),
  10))
xl
```

```
## $num
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $char
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
##
## $log
...
```

Subsetting lists

Use `[]` to subset and return a list, and `[[]]` to extract a specific list element.

The usual form of indexing is `[`. `[[` can be used to select a single element dropping names, whereas `[` keeps them, e.g., in `c(abc = 123)[1]`

Single bracket example

```
out.single <- xl[1]
length(out.single)
```

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

```
class(out.single)
```

```
## [1] "list"
```

Double bracket example

```
out.double <- xl[[1]]
length(out.double)
```

```
## [1] 10
```

```
class(out.double)
```

```
## [1] "integer"
```

Data frames

A `data.frame` is one of the fundamental data structures used by most of the modeling functions in R.

- Technically, a `data.frame` is a special type of `list`, where each element has the same length.
- Concretely, a `data.frame` is rectangular in shape, with rows and columns.
- You can think of a `data.frame` as similar to a single table in a database, or as a neatly rectangular single worksheet in a spreadsheet.

Example data.frame

The built-in data set `mtcars` is a `data.frame`:

```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Data frame properties

Properties of a data frame:

- A list with all elements the same length
- Like lists, each element can be a different class
- Has row names and column names

data.frame Structure

You can extract a lot of information about an object by getting its structure with `str()`.

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
...
```

Data frame row and column names

```
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```
colnames(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```
rownames(mtcars)
```

```
##  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
##  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
##  [7] "Duster 360"          "Merc 240D"           "Merc 230"
## [10] "Merc 280"            "Merc 280C"           "Merc 450SE"
## [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
## [19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
```

data.frame vs. arrays:

- Arrays coerce data into a single class
  - All values numeric, or all values character, etc.
- Data frames can have a different class for each column
  - Very useful to represent tabular data
  - Think of a data frame as very similar to a single table in a database

Data frames contain many modes

You can easily combine vectors of different modes in a data frame with the function `data.frame()`

```
dat <- data.frame(char = letters[1:5], num = 1:5, log = c(TRUE, FALSE,
  TRUE, FALSE, TRUE))

head(dat)
```

```
##   char num   log
## 1    a   1  TRUE
## 2    b   2 FALSE
## 3    c   3  TRUE
## 4    d   4 FALSE
## 5    e   5  TRUE
```

Data frames

Recall that `class()` returns the class name from which the R object inherits certain attributes. The function `typeof()` returns the internal storage type of an object:

```
class(mtcars)
```

```
## [1] "data.frame"
```

```
typeof(mtcars)
```

```
## [1] "list"
```

Using $ to subset lists and data frames

Lists and dataframes have another option for subsetting, the $ operator.

The operator $ is a way of pointing to a list element with a name:

```
cars$speed[1]
```

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

Helper functions on data.frames

Prior subsetting of vectors: `head()` and `tail()`

They work on `data.frames` as well.

```
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

Number of columns and rows

The functions `nrow()` and `ncol()` return the number of rows and columns of a data frame. The function `dim()` returns both dimensions.

```
nrow(mtcars)
```

```
## [1] 32
```

```
ncol(mtcars)
```

```
## [1] 11
```

```
dim(mtcars)
```

```
## [1] 32 11
```

Exercise: Practice subsetting

- What are the dimensions of `cars[1]`?
- How about `cars[[1]]`?
- Why the difference?
- Extract the final 10 rows from both columns of cars and store as another object.

What do all the brackets mean?

Operator	Meaning
()	Contains a comma-separated list of function arguments
[]	Subsets arrays, lists, and dataframes
[[]]	Extracts the value of a specific list element
{}	Defines an environment, typically for loops and functions.

Closing thoughts

- All R commands work on data structures or `objects`.
- All objects have a `class` that can be checked with the function class.

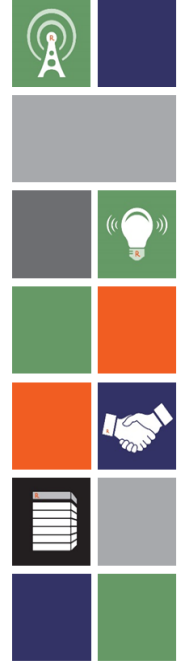
For more information, see

```
help(class)
```

Module review questions

- What is a `class`?
- What notation do you use to extract a subset of data?
- How about a specific list element?
- What is a ‘logical vector,’ and how can it be used to subset / modify an object?

REVOLUTION ANALYTICS



Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com
1.855.GET.REVO
Twitter: @RevolutionR