Introduction to the R Language

Data Types and Basic Operations

Roger Peng, Associate Professor
Johns Hopkins Bloomberg School of Public Health

Subsetting

There are a number of operators that can be used to extract subsets of R objects.

- `[` always returns an object of the same class as the original; can be used to select more than one element (there is one exception)
- `[[` 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 hat of `[[`.

Subsetting

```
> x <- c("a", "b", "c", "c", "d", "a")
> x[1]
[1] "a"
> x[2]
[1] "b"
> x[1:4]
[1] "a" "b" "c" "c"
> x[x > "a"]
[1] "b" "c" "c" "d"
> u <- x > "a"
> u
[1] FALSE TRUE TRUE TRUE TRUE FALSE
> x[u]
[1] "b" "c" "c" "d"
```

Subsetting a Matrix

Matrices can be subsetted in the usual way with (*i*,*j*) type indices.

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

By default, when a single element of a matrix is retrieved, it is returned as a vector of length 1 rather than a 1 × 1 matrix. This behavior can be turned off by setting `drop = FALSE`.

```
> x <- matrix(1:6, 2, 3)
> x[1, 2]
[1] 3
> x[1, 2, drop = FALSE]
     [,1]
[1,] 3
```

Subsetting a Matrix

Similarly, subsetting a single column or a single row will give you a vector, not a matrix (by default).

```
> x <- matrix(1:6, 2, 3)
> x[1, ]
[1] 1 3 5
> x[1, , drop = FALSE]
     [,1] [,2] [,3]
[1,]    1    3    5
```

Subsetting Lists

```
6/14
Subsetting Lists
> x <- list(foo = 1:4, bar = 0.6)
> x[1]
$foo
[1] 1 2 3 4

> x[[1]]
[1] 1 2 3 4

> x$bar
[1] 0.6
> x[["bar"]]
[1] 0.6
> x["bar"]
$bar
[1] 0.6
```

Subsetting Lists

```
> x <- list(foo = 1:4, bar = 0.6, baz = "hello")
> x[c(1, 3)]
$foo
[1] 1 2 3 4

$baz
[1] "hello"
```

Subsetting Lists

The `[[` operator can be used with *computed* indices; `$` can only be used with literal names.

```
> x <- list(foo = 1:4, bar = 0.6, baz = "hello")
> name <- "foo"
> x[[name]]  ## computed index for 'foo'
[1] 1 2 3 4
> x$name     ## element 'name' doesn't exist!
NULL
> x$foo
[1] 1 2 3 4  ## element 'foo' does exist
```

Subsetting Nested Elements of a List

The `[[` can take an integer sequence.

```
> x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))
> x[[c(1, 3)]]
[1] 14
> x[[1]][[3]]
[1] 14

> x[[c(2, 1)]]
[1] 3.14
```

Partial Matching

Partial matching of names is allowed with `[[` and `$`.

```
> x <- list(aardvark = 1:5)
> x$a
[1] 1 2 3 4 5
> x[["a"]]
NULL
> x[["a", exact = FALSE]]
[1] 1 2 3 4 5
```

Removing NA Values

A common task is to remove missing values (`NA`s).

```
> x <- c(1, 2, NA, 4, NA, 5)
> bad <- is.na(x)
> x[!bad]
[1] 1 2 4 5
```

Removing NA Values

What if there are multiple things and you want to take the subset with no missing values?

```
> x <- c(1, 2, NA, 4, NA, 5)
> y <- c("a", "b", NA, "d", NA, "f")
> good <- complete.cases(x, y)
> good
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE
> x[good]
[1] 1 2 4 5
> y[good]
[1] "a" "b" "d" "f"
```

Removing NA Values

```
> airquality[1:6, ]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
> good <- complete.cases(airquality)
> airquality[good, ][1:6, ]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
7    23     299  8.6   65     5   7
```