Chapter 4
Subsetting Objects

Arthur Li

The Subsetting Operators

- The subsetting operators: `[`, `[[`, and `$`

- Deciding which operator to use depends on the object

- The main behavior difference across these three objects:
  - Using the `[` operator returns the same data type
  - `[` can extract any numbers of elements of an object,
  - you can only use `[[` and `$` to extract one element
  - `$` does not evaluate its argument, while `[[` and `[` do
  - `$` uses partial matching to extract elements, while `[[` and `[` do not

The Subsetting Operators

```
> alist = list(name1 = c("john", "ken"), station =
+ "AM640", time = "M-F: 3:00pm")
> alist[c(1, 2)]
$name1
[1] "john" "ken"
$station
[1] "AM640"
```

Using the [operator returns a list

```
> alist["name"]
$<NA>
NULL
> alist[["name"]]
NULL
> alist$name
[1] "john" "ken"
```

$ uses partial matching to extract elements

The Subsetting Operators

```
> foo = "station"
> alist[foo]
$station
[1] "AM640"
> alist[[foo]]
[1] "AM640"
> alist$foo
NULL
```

$ does not evaluate its argument

Subsetting Vectors

- Subsetting vectors, matrices, or arrays can be referred to as *indexing*
- subsetting vector **`x`** can be done by **`x[index.vector]`**
- 5 types of index vectors

Type 1: A logical vector

- Let X = the “target” vector, L = logical index vector
- **`length (X)` = `length(L)`**
- The resulting vector = values from X corresponding to TRUE
- Values corresponding to NA returns NA

```
> a = c(1, 3, 5, NA, 7)
> b = a[!is.na(a)]
> b
[1] 1 3 5 7
> ind = a > 3
> ind
[1] FALSE FALSE TRUE NA TRUE
> a[ind]
[1] 5 NA 7
```

Type 1: A logical vector

- Let X = the “target” vector, L = logical index vector
- If **`length(L) < length(X)`** → the element of **`L`** will be recycled
- If **`length(X)`** is not a multiple of the **`length(L)`**, no warning message is generated

```
> a
[1]  1  3  5 NA  7
> a[c(T, F, T)]
[1] 1 5 NA
```

Type 1: A logical vector

- Let X = the “target” vector, L = logical index vector
- If **`length(L) > length(X)`** → the selected value will be extended to **`NA`**

```
> a
[1]  1  3  5 NA  7
> a[c(rep(T, 7), NA)]
[1] 1 3 5 NA 7 NA NA NA
```

Type 2: A vector of positive integer

- Let X = the "target" vector, P= index vector w/ positive integer
- `P[i]`: 1, 2, …length(X)

```
> a
[1]  1  3  5 NA  7
> a[c(1:3, 2)]
[1] 1 3 5 3
```

```
> a[6]
[1] NA
```

If `P[i] > length(X)` → selection = NA

```
> a[c(0, 3)]
[1] 5
```

If the elements of `P` are 0 → no corresponding result

```
> a[3.8]
[1] 5
```

A non-integer value will be truncated towards 0

```
> a[c(1, NA)]
[1] 1 NA
```

Elements of the `P` that are NA → generate a NA

Type 3: A vector of negative integer

- The index vector specifies the value to be excluded

```
> a
[1]  1  3  5 NA  7
> a[-c(1, 4)]
[1] 3 5 7
```

- Elements of the index vector that are 0 generate no corresponding values

- NA is not allowed to be included in the index vector

- You can not mix positive and negative index together

Type 4: A vector of character strings

- Only applies to an object that has names
- If the target vector has no names, it will result in an NA

```
> a
[1]  1  3  5 NA  7
> a[c("a", "c")]
[1] NA NA
> names(a) = c(letters[1:4], "d")
> a
a b c  d d
1 3 5 NA 7
> a[c("a", "c")]
a c
1 5
> a["d"]
d
NA
```

only the value with the lowest index is returned

Type 4: A vector of character strings

- Use **`is.element`** to find all occurrences of duplicated names

```
> a
a b c  d d
1 3 5 NA 7
> findD = is.element(names(a), "d")
> findD
[1] FALSE FALSE FALSE TRUE TRUE
> a[findD]
d d
NA 7
```

Type 4: A vector of character strings

- If one of the elements in the vector has no name (NA); including NA in the index vector will only return NA

```
> names(a)[3] = NA
> a
a b <NA>  d d
1 3    5 NA 7
> a[c(NA, "b")]
<NA> b
  NA 3
```

```
> a[is.na(names(a)) | names(a) == "b"]
b <NA>
3    5
```

Type 4: A vector of character strings

- If the target vector doesn't contain the names that match the values in the index vector, NA will be returned. Warning and Error Messages will not be produced

```
> a[c("e", "f")]
<NA> <NA>
  NA   NA
```

Type 5: The index position may be empty

- When the index vector is left empty, all the components of the vector are selected

```
> a[] = 0
> a
a b <NA> d d
0 0    0 0 0
```

Only valid when `a` already exists as a vector object and it does not change the attributes

Index Vector: Replacement

- Replacement: A index vector is on the LH side of an assignment

```
> x = c(3, 6, NA, -1)
> x
[1]  3  6 NA -1
> x[is.na(x)] = 0
> x
[1]  3  6  0 -1
```

```
> x[1:3] = 1:3
> x
[1]  1  2  3 -1
```

```
> x[7] = 8
> x
[1]  1  2  3 -1 NA NA  8
```

an index outside the range of 1, ..., length(x)

Index Vector: Replacement

```
> x[-c(2, 5)] = 10
> x
[1] 10 2 10 10 NA 10 10
```

```
> names(x) = c(NA, letters[1:6])
> x
<NA> a  b  c  d  e  f
  10 2 10 10 NA 10 10
> x[c(NA, "a")] = 3
> x
<NA> a  b  c  d  e f <NA>
  10 3 10 10 NA 10 10    3
```

Missing values create a new element of the vector

Subsetting Matrices and Arrays
Array Indexing

❖ A vector is an array ↔ it has **`dim`** attribute/dimension vector

```
> x = c(1:20, rep(NA, 4))
> dim(x) = c(2, 3, 4)
> dimnames(x) =  list(d1 = c("i", "ii"), d2 = c("I", "II", "III"),
+ d3 = letters[1:4])
> x
, , d3 = a

    d2
d1   I II III
  i  1  3   5
  ii 2  4   6

, , d3 = b

    d2
d1   I II III
  i  7  9  11
  ii 8 10  12
...
...
```

Array indexing

❖ An array is still a vector, we can use vector indexing method

```
> x[3:6]
[1] 3 4 5 6
> x[!is.na(x)]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

❖ Or we can use separate indices, separated by commas

```
> x[1,2,3]
[1] 15
> x[2,1, 2:3]
 b  c
 8 14
```

Array indexing

❖ For a k-fold indexed array, any of the 5 forms of indexing is allowed in each index position

❖ For empty index, the range = full range for the index position

❖ **x** is 2 X 3 X 4 →
x[, c("I", "III"), -c(2, 4)] is 2 X 2 X 2

```
> x[, c("I", "III"), -c(2, 4)]
, , d3 = a
     d2
d1    I III
   i  1   5
  ii  2   6

, , d3 = c
     d2
d1    I III
   i 13 17
  ii 14 18
```

❖ If character values in a character index do not match with any of the values in the **dimname** attributes, an "subscript out of bounds" error message will be produced

Dropping Indices

```
> square.matrix
     [,1] [,2] [,3]
[1,]    1    4    0
[2,]    2    0    8
[3,]    0    6    9
```

❖ Example: create a 3 X 1 matrix by subsetting the first column

```
> single = square.matrix[,1]
> single
[1] 1 2 0
> dim(single)
NULL
```

❖ If the index range reduced to 1 value, **`dim`** is removed

Dropping Indices

- Set **`drop`** option to **`F`** override it

```
> single = square.matrix[, 1, drop = F]
> single
     [,1]
[1,]    1
[2,]    2
[3,]    0
> dim(single)
[1] 3 1
```