Subsetting

Jayant

2/22/2020

Operators

1. [
2. [[
3. $

1. [

Always gives the object of same class as original. This rule has one exception which is for when we use it with matrix.(done later)

Can be used to select more than one element of an object

```
x <- c("a","b","c","c","d","a")
# Using numeric index
x[1]
```

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

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

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

```
# Using logical index
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"
```

```r
x <- list(foo = 1:4, bar = 0.6)
x[1]
```

```
## $foo
## [1] 1 2 3 4
```

Here we get a list containing 1 to 4

2. [[

Extracts elements of a list or data frame , the returned object class can be diff.

Can only extract a single element

```r
x <- list(foo = 1:4, bar = 0.6)
x[[1]]
```

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

Here we get just the seq 1 to 4

3. $

Extracts elements of a list or data frame that have names. The returned object class can be diff.

```r
x <- list(foo = 1:4, bar = 0.6)
x$bar
```

```
## [1] 0.6
```

```r
x$foo
```

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

Lists

Extracting single element

```r
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"]]   # Same as using $bar
```

```
## [1] 0.6
```

```
x["bar"]     # Gives list with element bar in it
```

```
## $bar
## [1] 0.6
```

Extracting multiple elements

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

```
## $foo
## [1] 1 2 3 4
##
## $baz
## [1] "Hello"
```

```
name <- "foo"
x[[name]]     #Computed index for "foo"
```

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

```
x$name        #element "name" doesn't exsist
```

```
## NULL
```

```
x$foo
```

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

List inside list

```r
x <- list(a = list(10,12,14), b = c(3.18,2.81))
x[[c(1,3)]]
```

```
## [1] 14
```

```r
x[[1]][[3]]
```

```
## [1] 14
```

```r
x[[c(2,1)]]
```

```
## [1] 3.18
```

Matrices

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

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

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

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

```r
x[1,]
```

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

```r
x[,2]
```

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

These are giving us vector values

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

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

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

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

```
x[1,2,drop=FALSE]
```

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

```
x[1,]
```

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

```
x[1,,drop=FALSE]
```

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

By default Drop = TRUE and so it drops the dimension.

Partial matching

```
x <- list(wabalabadubdub = 1:4)
x$w
```

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

```
x[["w"]]
```

```
## NULL
```

```
x[["w",exact=FALSE]]
```

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

Removing missing values

For single object

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

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

For multiple objects

```
x <- c(1,2,NA,4,NA,5)
y <- c("a","b",NA,"d",NA,"f")
good <- complete.cases(x,y)   #Pos where values for both are not missing
x[good]
```

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

```
y[good]
```

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

complete.cases() can remove missing values from big data frames as well.