Subsetting

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

- `[` 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 `[[`(reference elements by name)

```
> x <- c("a", "b", "c")
> x[1] ## Numeric Index, in that case is 1
[1] "a"
> x[1:3]
[1] "a" "b" "c"
> x[x > "a"] ## Logical Index
[1] "b" "c"
> u <- x > "a" ## a subset u which element is greater than a
> u
[1] FALSE  TRUE  TRUE
```

Lists

```
> x <- list(foo = 1:4 , bar = 0.6) ## foo is element 1 and bar is 2
> x[1] ## export element 1
$foo
[1] 1 2 3 4

> x[[1]] ## get the sequence of element 1
[1] 1 2 3 4

> x$bar ## get the element named bar, dont have remember place
[1] 0.6
> x[["bar"]]
[1] 0.6
> x["bar"]
$bar
[1] 0.6
```

Subsetting List

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

$baz
[1] "hello"
```

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$name ## element "name" does not exist!(name is variable)
NULL
> x$foo ## elememt "foo" exists
[1] 1 2 3 4
```

Subsetting Nested Elements of a List

The `[[` cam take an integer sequence(recurses into the list)

```
> x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))
> x[[c(1, 3)]] ## third element of the list
[1] 14
> x[[1]][[3]] ## equalivent to the fitst
[1] 14
> x[[c(2, 1)]]
[1] 3.14
```

Subsetting a Matrix

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

```
> x <- matrix(1:6, 2, 3) ## x is a 2 by 3 matrix
> x[1, 2] ## first row, second column
[1] 3
> x[2, 1]
[1] 2
```

Indices can also be missing

```
> x[1, ] ## first row
[1] 1 3 5
> x[, 2] ## second column
[1] 3 4
```

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

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

Similarly, subsetting a single column or a single row will give you a vector, not a matrix

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

Partial Matching

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

```
> x <- list(aardvark = 1:5)
> x$a ## looks for a name contain "a"
[1] 1 2 3 4 5
> x[["a"]] ## exact match
NULL
> x[["a", exact = FALSE]] ## close exact match
[1] 1 2 3 4 5
```

Removing NA Values

A common task is to remove missing values (`NAS`)

```
> x <- c(1, 2, NA, 4, NA, 5)
> bad <- is.na(x) ## check which one is NA
> x[!bad] ## get the element which does not belong to bad
[1] 1 2 4 5
```

When multiple NA vector occurs,

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

Vectorized Operations

Many operations in R are vectorized making code more efficient

```
> x <- 1:4; y <- 6:9
> x + y
[1]  7  9 11 13
> x > 2
[1] FALSE FALSE  TRUE  TRUE
> x >= 2
[1] FALSE  TRUE  TRUE  TRUE
> y == 8
[1] FALSE FALSE  TRUE FALSE

> x * y
[1]  6 14 24 36
> x / y
[1] 0.1666667 0.2857143 0.3750000 0.4444444

> x <- matrix(1:4, 2, 2); y <- matrix(rep(10, 4), 2, 2)
> x * y ## Element-wise multiplication
     [,1] [,2]
[1,]   10   30
[2,]   20   40
> x /y
     [,1] [,2]
[1,]  0.1  0.3
[2,]  0.2  0.4
> x %*% y ## ture matrix multiplication
     [,1] [,2]
[1,]   40   40
[2,]   60   60
```