Vectors, Matrices, Arrays, Lists, and Data Frames

Vector – a collection of **ordered homogeneous** elements.

We can think of matrices, arrays, lists and data frames as deviations from a vector. The deviaitions are related to the two characteristics **order** and **homogeneity**.

Matrix - a vector with two-dimensional **shape** information.

```
> xx = matrix(1:6, nrow=3, ncol =2)
> xx
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

> class(xx)                     [1] "matrix"
> is.vector(xx)                 [1] FALSE
> is.matrix(xx)                 [1] TRUE
> length(xx)                    [1] 6
> dim(xx)                       [1] 3 2
```


Also, matrices can have row and column names, which can be determined and/or assigned by **rownames** and **colnames**. Other functions **nrow**, **ncol**, **dimnames**.


```
> yy = array(1:12, c(2,3,2))
> yy
, , 1

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

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

> length(yy)                    [1] 12
> dim(yy)                       [1] 2 3 2
> is.matrix(yy)                 [1] FALSE
> is.array(yy)                  [1] TRUE
```


Lists

A vector with possible **heterogeneous** elements. The elements of a list can be numeric vectors, character vectors, matrices, arrays, and lists.

myList = list(a = 1:10, b = ”def”, c(TRUE, FALSE, TRUE))

```
$a
 [1]  1  2  3  4  5  6  7  8  9 10
$b
 [1] "def"
[[3]]
 [1]  TRUE FALSE  TRUE
```

- **length(myList)** – there are 3 elements in the list
- **class(myList)** – the class is a “list”
- **names(myList)** – are “a”, “b” and the empty character “”
- **myList[1:2]** – returns a list with two elements
- **myList[1]** – returns a **list** with one element. What is length(myList[1]) ?
- **myList[[1]]** – returns a **vector** with ten elements, the numbers 1, 2, ..., 10 What is length(myList[[1]]) ?

- **names(intel)** – returns the element names of the list, which are the names of each of the vectors: ”Date”, ”Transistors”, ”Microns” etc.
- **class(intel)** – a ”data.frame”
- **dim(intel)** – as a rectangular list, the data frame supports some matrix features: 10 7
- **length(intel)** – the length is the number of elements in the list, NOT the combined number of elements in the vectors, i.e. it is ?
- **class** of **intel[”Date”]** versus **intel[[”Date”]]** – recall the **[]** returns an object of the same type, i.e. a list but **[[ ]]** returns the element in the list.
- What is the class of the speed element in intel?

```
> intel[["speed"]]
 [1] MHz MHz MHz MHz MHz MHz MHz MHz GHz GHz
 Levels: GHz MHz
```


Computations involving Vectors and Lists

- Write code using vectorized function calls
  e.g. nchar(y), x[ ] = 0, z + w
- Use the apply mechanism
  - **lapply** and **sapply** for lists
  - **apply** for matrices and arrays
  - **tapply** for ragged arrays as vectors
- With these functions we can avoid looping, and write code that is meaningful in a statistical setting, e.g. if we have a list of rainfall data where each element represents the measurements taken at a different weather station, when we think about studying the average rainfall at each station we don’t think in terms of loops.

Data Frames

A list with possible **heterogeneous** vector elements of the **same length**. The elements of a data frame can be numeric vectors, factor vectors, and logical vectors, but they must all be of the same length.

```
> intel
           Date Transistors Microns Clock speed Data    MIPS
8080       1974        6000    6.00   2.0   MHz    8    0.64
8088       1979       29000    3.00   5.0   MHz   16    0.33
80286      1982      134000    1.50   6.0   MHz   16    1.00
80386      1985      275000    1.50  16.0   MHz   32    5.00
80486      1989     1200000    1.00  25.0   MHz   32   20.00
Pentium    1993     3100000    0.80  60.0   MHz   32  100.00
PentiumII  1997     7500000    0.35 233.0   MHz   32  300.00
PentiumIII 1999     9500000    0.25 450.0   MHz   32  510.00
Pentium4   2000    42000000    0.18   1.5   GHz   32 1700.00
Pentium4x  2004   125000000    0.09   3.6   GHz   32 7000.00
```


Subsetting a Data Frame

Using the fact that a data frame is a list which also support some matrix features, fill in the table specifying the **class** (data.frame or integer) and the **length** and **dim** of the subset of the data frame. Note that some responses will be NULL.

Subset	class	length	dim
intel			
intel[1]			
intel[[1]]			
intel[,1]			
intel[“Date”]			
intel[, “Date”]			
intel$Date			

Apply

apply(xx, 1, sum) for the matrix xx, the sum function is applied across the columns so that the row dimension (i.e. dim 1) is preserved.

```
> xx
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> apply(xx, 1, sum)
[1]  9 12
```


apply(aa, c(1,2), sum) for the array aa, the sum function is applied across the pages so that the row and column dimensions (i.e. dim 1 and 2) are preserved.

```
> aa
, , 1
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2
      [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

> apply(aa,c(1,2),sum)
       [,1] [,2] [,3]
[1,]    8   12   16
[2,]   10   14   18
```

apply(aa, 2, sum) **apply(aa, c(2, 3), sum)** **apply(aa, c(3, 2), sum)**


```
> apply(aa,c(2),sum)
[1] 18 26 34

> apply(aa,c(2,3),sum)
     [,1] [,2]
[1,]    3   15
[2,]    7   19
[3,]   11   23

> class(apply(aa,c(2,3),sum))
[1] "matrix"

> apply(aa,c(3,2),sum)
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]   15   19   23
```


Applying functions to list elements

The **lapply** and **sapply** both apply a specified function to each element of a list. The former returns a list object and the latter a vector when possible.

```
> ll
[[1]]
[1] 1 2 3 4 5

[[2]]
[1] 2 2 2

[[3]]
[1]  0.0546  0.6851  0.8388 -0.1199  0.7995 -0.2518
[7] -0.0585 -0.1581  0.6912  0.3957

> lapply(ll, sum)
```

tapply

This function is useful to apply a function to subsets of a vector.

```
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> v
 [1]  1  1  1  0  0  0  1  1  1  0

> tapply(x, v, mean)
   0    1
  6.25 5.00

> tapply(x, v, median)
   0   1
  5.5 5.0
```



```
[[1]]
[1] 15

[[2]]
[1] 6

[[3]]
[1] 2.87678

> sapply(ll, sum)
[1] 15.00000  6.00000  2.87678
```