Vector Out-of-Order Indexes

- The index vector can even be out-of-order. Here is a vector slice with the order of first and second members reversed.
- For example,

```
> s = c("aa", "bb", "cc", "dd", "ee")
> s[c(2, 1, 3)]
[1] "bb" "aa" "cc"
```

Vector Range Index

- To produce a vector slice between two indexes, we can use the colon operator ":".
- For example,

```
> s = c("aa", "bb", "cc", "dd", "ee")
> s[2:4]
[1] "bb" "cc" "dd"
```

Named Vector Members

- We can assign names to vector members.
- For example, the following variable v is a character string vector with two members.

```
> v = c("Mary", "Sue")
> v
[1] "Mary" "Sue"
```

- We now name the first member as First, and the second as Last.

```
> names(v) = c("First", "Last")
> v
 First   Last
"Mary"  "Sue"
```

Matrices

- A matrix is a collection of data elements arranged in a row-column layout.
- A matrix can be regarded as a generalization of a vector.
- As with vectors, all the elements of a matrix must be of the same data type.
- A matrix can be generated in several ways.
  - ✓ Use the function dim
  - ✓ Use the function matrix

Matrices

- Use the function dim

```
> x <- 1:8
> dim(x) <- c(2,4)
> X
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

- Use the function matrix

```
> A = matrix(c(2, 4, 3, 1, 5, 7), nrow=2, ncol=3, byrow = T)
> A
> A <- matrix(c(2, 4, 3, 1, 5, 7),2,3,byrow=T)
> A
```

```
A    [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

Accessing Matrices

- An element at the m^{th} row, n^{th} column of A can be accessed by the expression A[m, n].

```
> A[2, 3]
[1] 7
```

- The entire m^{th} row A can be extracted as A[m,].

```
> A[2, ]
[1] 1   5   7
```

- We can also extract more than one rows/columns at a time.

```
> A[ ,c(1,3)]
     [,1] [,2]
[1,]    2    3
[2,]    1    7
```

Calculations on matrices

- We construct the transpose of a matrix by interchanging its columns and rows with the function t .

```
> t(A)          # transpose of A
     [,1] [,2]
[1,]    2    1
[2,]    4    5
[3,]    3    7
```

- We can deconstruct a matrix by applying the c function, which combines all column vectors into one.

```
> c(A)
[1] 2 4 3 1 5 7
```

Arrays

- In R, Arrays are generalizations of vectors and matrices.
- A vector is a one-dimensional array and a matrix is a two dimensional array.
- As with vectors and matrices, all the elements of an array must be of the same data type.
- An array of one dimension of two element may be constructed as follows.

```
> x = array(c(T,F),dim=c(2))
> print(x)
[1] TRUE FALSE
```

Arrays

- A three dimensional array - 3 by 3 by 3 - may be created as follows.

```
> z = array(1:27,dim=c(3,3,3))
> dim(z)
[1] 3 3 3
> print(z)
, , 1

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

, , 2

     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18

, , 3

     [,1] [,2] [,3]
[1,]   19   22   25
[2,]   20   23   26
[3,]   21   24   27
```

Accessing Arrays

- R arrays are accessed in a manner similar to arrays in other languages: by integer index, starting at 1 (not 0).
- For example, the third dimension is a 3 by 3 array.

```
> z[,,3]
     [,1] [,2] [,3]
[1,]   19   22   25
[2,]   20   23   26
[3,]   21   24   27
```

- Specifying two of the three dimensions returns an array on one dimension.

```
>z[,3,3]
[1] 25 26 27
```

Accessing Arrays

- Specifying three of three dimension returns an element of the 3 by 3 by 3 array.

```
> z[3,3,3]
[1] 27
```

- More complex partitioning of array may be had.

```
> z[,c(2,3),c(2,3)]

, , 1
     [,1] [,2]
[1,]   13   16
[2,]   14   17
[3,]   15   18

, , 2
     [,1] [,2]
[1,]   22   25
[2,]   23   26
[3,]   24   27
```

Lists

- A list is a collection of R objects.
- list() creates a list. unlist() transform a list into a vector.
- The objects in a list do not have to be of the same type or length.

```
>x <- c(1:4)
>y <- FALSE
> z <-
matrix(c(1:4),nrow=2,ncol=2)
> myList <- list(x,y,z)
> myList

[[1]]
[1] 1 2 3 4

[[2]]
[1]
FALSE

[[3]]
[,1] [,2]
[1,] 1 2
[2,] 3 4
```

Data Frame

- A data frame is used for storing data like spreadsheet(table).
- It is a list of vectors of equal length.
- Most statistical modeling routines in R require a data frame as input.
- For example,

```
> weight = c(150, 135, 210, 140)
> height = c(65, 61, 70, 65)
> gender = c("Fe","Fe","Ma","Fe")
> study = data.frame(weight,height,gender) # make the data frame
> study
  weight height gender
1    150     65     Fe
2    135     61     Fe
3    210     70     Ma
4    140     65     Fe
```

Creating a data frame

- The dataframe may be created directly using data.frame().
- For example, the dataframe is created - naming each vector composing the dataframe as part of the argument list.

```
> patientID <- c(1, 2, 3, 4)
> age <- c(25, 34, 28, 52)
> diabetes <- c("Type1", "Type2", "Type1", "Type1")
> status <- c("Poor", "Improved", "Excellent", "Poor")
> patientdata <- data.frame(patientID, age, diabetes, status)
> patientdata
```

	patientID	age	diabetes	status
1	1	25	Type1	Poor
2	2	34	Type2	Improved
3	3	28	Type1	Excellent
4	4	52	Type1	Poor

Accessing data frame elements

- Use the subscript notation/specify column names to identify the elements in the patient data frame [1] 25 34 28 52

```
>patientdata[1:2]
```

	patientID	age
1	1	25
2	2	34

```
>patientdata[c("diabetes", "status")]
```

	diabetes	status
1	Type1	Poor
2	Type2	Improved
3	Type1	Excellent
4	Type1	Poor

```
>table(patientdata$diabetes, patientdata$status)
```

	Excellent	Improved	Poor
Type1	1	0	2
Type2	0	1	0

Importing and Exporting Data

- There are many ways to get data in and out.
- Most programs (e.g. Excel), as well as humans, know how to deal with rectangular tables in the form of tab-delimited text files.
- Normally, you would start your R session by reading in some data to be analysed. This can be done with the read.table function. Download the sample data to your local directory...

```
>x <- read.table("sample.txt", header = TRUE)
```
Also: read.delim, read.csv, scan

```
>write.csv(x, file = "samplenew.csv")
```
Also: write.matrix, write.table, write

HANDSON

Accessing data frame elements

- Use the subscript notation/specify column names to identify the elements in the patient data frame [1] 25 34 28 52

```
>patientdata[1:2]
  patientID age
1         1  25
2         2  34
3         3  28
4         4  52
```

```
>patientdata[c("diabetes", "status")]
  diabetes  status
1    Type1    Poor
2    Type2 Improved
3    Type1 Excellent
4    Type1    Poor
```

```
>table(patientdata$diabetes, patientdata$status)
       Excellent Improved Poor
Type1          1        0    2
Type2          0        1    0
```