NATIONAL ENGINEERING CENTER
University of the Philippines
Diliman, Quezon City

2.0 Data Types and Simple Operators

Eugene Rex L. Jalao, Ph.D.

Associate Professor

Department Industrial Engineering and Operations Research

University of the Philippines Diliman

@thephdataminer

Module 6 of the Business Intelligence and Analytics Certification of UP NEC and the UP Center for Business Intelligence

Outline for this Training

- Introduction to R and R Studio
- **Data Types and Operators**
  - **Case Study on R Scripting**
- Reading, Manipulating and Writing Data
  - Case Study on Dataset Analysis with ETL
- Basic R Programming
  - Case Study: Writing Functions
- Graphics and Plotting
- Deploying R and Dashboard Generation
  - Case Study: Deploying a Simple Dashboard
- Deploying R with C#
  - Case Study: A Simple Standalone GUI For R Apps

Outline for This Session

- One Dimensional Vector Arithmetic
- Multi Dimensional Vector Arithmetic
- Case Study

One Dimensional Vector Arithmetic

Definition 2.1: Variable

- A variable is a memory storage location paired with an associated symbolic name (an *identifier*),
- It contains some known or unknown quantity or information referred to as a value.

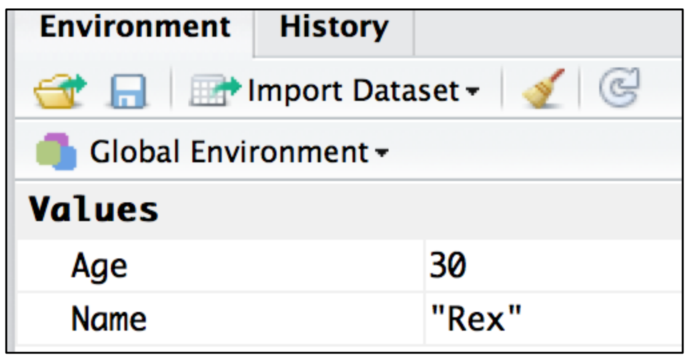
One Dimensional Vector Arithmetic

Example 2.1: Variables

- Example Type and Run the following Lines of Code

➢ `Name = "Rex"`

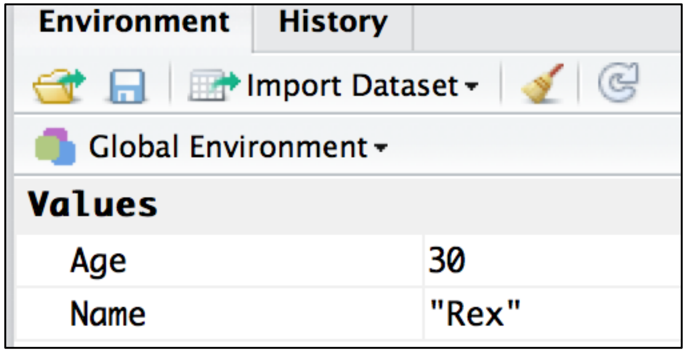
➢ `Age = 20`



One Dimensional Vector Arithmetic

Definition 2.2: The Environment

- The environment is where you can **view all** of your variables that you have declared
- Options for the Environment
  - Save Workspace Environment into an Rdata File
  - Open a Saved Workspace Environment
  - Clear Current Workspace



One Dimensional Vector Arithmetic

Definition 2.3: Vectors

- A vector is an entity consisting of an ordered collection of numbers.
- Generate vectors using the function c() which can take an arbitrary number of vector arguments and concatenates it into a single variable.
- An example of a vector:

$$x = \begin{bmatrix} 10.4 \\ 5.6 \\ 3.1 \\ 6.4 \\ 21.7 \end{bmatrix}$$

One Dimensional Vector Arithmetic

Example 2.2: Vectors

- Declare a new vector named x, consisting of five numbers, namely 10.4, 5.6, 3.1, 6.4 and 21.7 and print the result

➢ `#create a vector`

➢ `x <- c(10.4, 5.6, 3.1, 6.4, 21.7)`

➢ `#print the vector`

➢ `x`

```
> #create a vector
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> #print the vector
> x
[1] 10.4  5.6  3.1  6.4 21.7
>
```

One Dimensional Vector Arithmetic

- If an **expression** is used as a complete command, the value is printed and lost
  - ➢ `1/x`
  - – the reciprocals of the five values would be printed at the terminal (and the value of x, of course, unchanged).
- The **assignment**
  - ➢ `y <- c(x, 0, x)`
  - – would create a vector y with 11 entries consisting of two copies of x with a zero in the middle place.
- The `y` vector will be saved in the environment

One Dimensional Vector Arithmetic

Example 2.3: Assignments

- Print Assignments and Expressions

➢ #this vector is lost

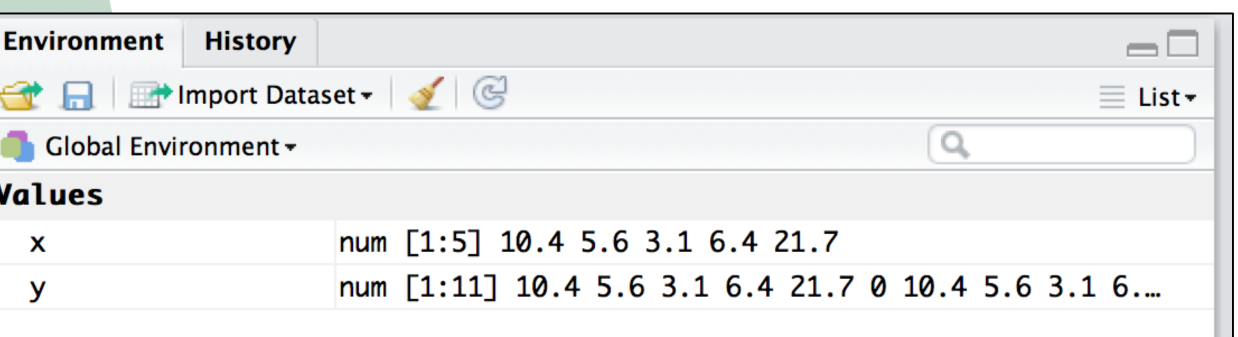
➢ 1/x

➢ #value is saved

➢ y <- c(x, 0, x)

➢ y

```
> #this vector is lost
> 1/x
[1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295
> #value is saved
> y <- c(x, 0, x)
> y
 [1] 10.4  5.6  3.1  6.4 21.7  0.0 10.4  5.6  3.1  6.4 21.7
>
```



One Dimensional Vector Arithmetic

- Alternative Assignments
- `<-`
- `=`
- `->`
- Eg

➢ `c(10.4, 5.6, 3.1, 6.4, 21.7) -> x`

One Dimensional Vector Arithmetic

- Vectors can be used in arithmetic expressions, in which case the operations are performed element by element.
- Vectors occurring in the same expression need not all be of the same length. If they are not, the value of the expression is a vector with the same length as the longest vector
- Shorter vectors in the expression are recycled as often as need be until they match the length of the longest vector.
- A constant is simply repeated.
- Example

One Dimensional Vector Arithmetic

Example 2.4: Vector Arithmetic

- Calculate for $v = 2 * x + y + 1$

➢ #Different Lengths

➢ v <- 2*x + y + 1

➢ v

```
> #Different Lengths
> v <- 2*x + y + 1
Warning message:
In 2 * x + y :
  longer object length is not a multiple of shorter object length
> v
 [1] 32.2 17.8 10.3 20.2 66.1 21.8 22.6 12.8 16.9 50.8 43.5
> 
```

One Dimensional Vector Arithmetic

- Elementary arithmetic operators : +, -, *, / and ^ for raising to a power.
- Arithmetic functions are available. `log, exp, sin, cos, tan, sqrt, max, min.`
- `range` is a function whose value is a vector of length two, namely `c(min(x), max(x)).`
- `length(x)` is the number of elements in `x, sum(x)` gives the total of the elements in `x, and prod(x)` their product.

One Dimensional Vector Arithmetic

Example 2.4 (Cont.) : Vector Arithmetic

- Do some arithmetic operations on vector x

```
#Arithmetic Operations
z <- (2*x + 5)^2
z
# The Range of X
xrange = range(x)
xrange
# The Num of Elements in X
xlength = length(x)
xlength
```

One Dimensional Vector Arithmetic

```
> #Arithmetic Operations
> z <- (2*x + 5)^2
> z
[1]  665.64  262.44  125.44  316.84 2342.56
> # The Range of X
> xrange = range(x)
> xrange
[1]  3.1 21.7
> # The Num of Elements in X
> xlength = length(x)
> xlength
[1] 5
> 
```

One Dimensional Vector Arithmetic

- Elementary Statistical Operations
  - Calculating the average of a vector:
    - `mean(x)` which calculates the sample mean, which is the same as `sum(x)/length(x)`
  - Calculating the variance of a vector:
    - `var(x)` which is equal to `sum((x-mean(x))^2)/(length(x)-1)` or sample variance.
  - Calculating the standard deviation of a vector:
    - `sqrt(var(x))`
  - Calculating the median of a vector
    - `median(x)`

One Dimensional Vector Arithmetic

Example 2.5: Statistical Operations

- Do some arithmetic operations on vector x

➢ `#Mean of x`
➢ `mean(x)`
➢ `#Variance of x`
➢ `var(x)`
➢ `#Standard deviation of x`
➢ `sqrt(var(x))`
➢ `#Median of x`
➢ `median(x)`

```
> #Mean of x
> mean(x)
[1] 9.44
> #Variance of x
> var(x)
[1] 53.853
> #Standard deviation of x
> sqrt(var(x))
[1] 7.33846
> #Median of x
> median(x)
[1] 6.4
>
```

One Dimensional Vector Arithmetic

- Generating a sequential vector:
  - `1:30` is the vector `c(1, 2, ..., 29, 30)`
- The colon operator has a high priority within an expression,
- Example
  - `2*1:15` is the vector `c(2, 4, ..., 28, 30).`
- General format for generating sequences:
  - `seq(from, to, by=)`
  - `seq(length, from, to, by=)`
- Generate repeating valued vector
  - `rep(vector, times=)`

One Dimensional Vector Arithmetic

Example 2.6: Arithmetic Sequences

- `#Arithmetic Sequence from 1 to 30`
- `1:30`
- `#Priority of Operations`
- `2*1:15`
- `#Sequence Function`
- `seq(from=-5, to=5, by=.2)`
- `seq(length=51, from=-5, by=.2)`
- `#Repeat Function`
- `rep(1, times=5)`

One Dimensional Vector Arithmetic

```
> #Arithmetic Sequence from 1 to 30
> 1:30
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
[29] 29 30
> #Priority of Operations
> 2*1:15
 [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30
> #Sequence Function
> seq(from=-5, to=5, by=.2)
 [1] -5.0 -4.8 -4.6 -4.4 -4.2 -4.0 -3.8 -3.6 -3.4 -3.2 -3.0 -2.8 -2.6 -2.4 -2.2 -2.0 -1.8
[18] -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2  0.0  0.2  0.4  0.6  0.8  1.0  1.2  1.4  1.6
[35]  1.8  2.0  2.2  2.4  2.6  2.8  3.0  3.2  3.4  3.6  3.8  4.0  4.2  4.4  4.6  4.8  5.0
> seq(length=51, from=-5, by=.2)
 [1] -5.0 -4.8 -4.6 -4.4 -4.2 -4.0 -3.8 -3.6 -3.4 -3.2 -3.0 -2.8 -2.6 -2.4 -2.2 -2.0 -1.8
[18] -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2  0.0  0.2  0.4  0.6  0.8  1.0  1.2  1.4  1.6
[35]  1.8  2.0  2.2  2.4  2.6  2.8  3.0  3.2  3.4  3.6  3.8  4.0  4.2  4.4  4.6  4.8  5.0
> #Repeat Function
> rep(1, times=5)
[1] 1 1 1 1 1
> 
```

One Dimensional Vector Arithmetic

- The elements of a logical vector can have the values TRUE, FALSE, and NA (for “not available”).
- The logical operators are <, <=, >, >=, == for exact equality and != for inequality.
- Logical vectors are generated by conditions.

One Dimensional Vector Arithmetic

Example 2.7: Logical Vectors

- Create a variable temp to determine which elements in x are greater than 10

```
➢ #Logical Vectors
➢ x
➢ temp <- x > 10
➢ temp
```

```
> #Logical Vectors
> x
[1] 10.4  5.6  3.1  6.4 21.7
> temp <- x > 10
> temp
[1]  TRUE FALSE FALSE FALSE  TRUE
> 
```

One Dimensional Vector Arithmetic

- In some cases the components of a vector may not be completely known.
- When an element or value is "not available" or a "missing value" it will be assigned the special value NA.
- Any operation on an NA becomes an NA.
  - The motivation for this rule is simply that if the specification of an operation is incomplete, the result cannot be known and hence is not available.
- The function `is.na(x)` gives a logical vector of the same size as `x` with value TRUE if and only if the corresponding element in `x` is NA.

One Dimensional Vector Arithmetic

Example 2.8: Missing Values

- #Working with NA
- z <- c(1:3,NA)
- z
- ind <- is.na(z)
- ind

```
> #Working with NA
> z <- c(1:3,NA)
> z
[1]  1  2  3 NA
> ind <- is.na(z)
> ind
[1] FALSE FALSE FALSE  TRUE
>
```


eljalao@up.edu.ph

One Dimensional Vector Arithmetic

- Character quantities and character vectors are used frequently in R, for example as plot labels.
- They are denoted by a sequence of characters delimited by the double quote character, e.g., "x-values", "New iteration results".
- Character strings are entered using either matching double (") or single (’) quotes, but are printed using double quotes
- The `paste()` function takes an arbitrary number of arguments and concatenates them one by one into character strings.

One Dimensional Vector Arithmetic

Example 2.9: Character Vectors

- #This is a character vector
- "x-values"
- #The use of paste
- paste("Eugene","-", "Rex")
- labs <- paste(c("X"), 1:10, sep="")
- labs

```
> #This is a character vector
> "x-values"
[1] "x-values"
> #The use of paste
> paste("Eugene","-", "Rex")
[1] "Eugene - Rex"
> labs <- paste(c("X"), 1:10, sep="")
> labs
 [1] "X1"  "X2"  "X3"  "X4"  "X5"  "X6"  "X7"  "X8"  "X9"  "X10"
>
```

One Dimensional Vector Arithmetic

- Dates are represented as the number of days since 1970-01-01, with negative values for earlier dates.
- use as.Date( ) to convert strings to dates
- Sys.Date( ) returns today's date.
- date() returns the current date and time.

One Dimensional Vector Arithmetic

Example 2.10: Date Vectors

- `mydates <- as.Date(c("2007-06-22", "2004-02-13"))`
- `# number of days between 6/22/07 and 2/13/04`
- `days <- mydates[1] - mydates[2]`
- `days`

```
> mydates <- as.Date(c("2007-06-22", "2004-02-13"))
> # number of days between 6/22/07 and 2/13/04
> days <- mydates[1] - mydates[2]
> days
Time difference of 1225 days
>
```

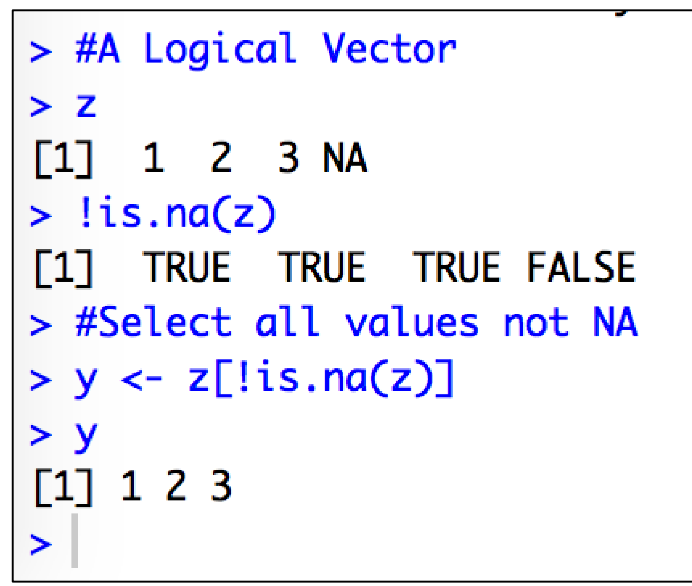
One Dimensional Vector Arithmetic

- Indexing
  - Subsets of the elements of a vector may be selected by appending to the name of the vector an index vector in square brackets.
  - Such index vectors can be any of four distinct types.
    - A Logical Vector: Selects all True Indices
    - A Sequence of Positive Integers: Selects all items from 1 to the end of the sequence
    - A Sequence of Negative Integers : Selects all except items from -1 to the end of the sequence
    - A Vector of Character Strings: Select only the ones in the character string

One Dimensional Vector Arithmetic

Example 2.11: Indexing

- #A Logical Vector
- z
- !is.na(z)
- #Select all values not NA
- y <- z[!is.na(z)]
- y



One Dimensional Vector Arithmetic

Example 2.11 (Cont.): Indexing

- `v`
- `#Include Using Positive Integers`
- `temp = v[1:10]`
- `temp`
- `#Exclude Using Negative Integers`
- `temp2 = v[-(1:5)]`
- `temp2`
- `#Include Using Any Number`
- `temp3 = v[c(1,4,6)]`
- `temp3`

One Dimensional Vector Arithmetic

Example 2.11 (Cont.): Indexing

```
> v
 [1] 32.2 17.8 10.3 20.2 66.1 21.8 22.6 12.8 16.9 50.8 43.5
> #Include Using Positive Integers
> temp = v[1:10]
> temp
 [1] 32.2 17.8 10.3 20.2 66.1 21.8 22.6 12.8 16.9 50.8
> #Exclude Using Negative Integers
> temp2 = v[-(1:5)]
> temp2
[1] 21.8 22.6 12.8 16.9 50.8 43.5
> #Include Using Any Number
> temp3 = v[c(1,4,6)]
> temp3
[1] 32.2 20.2 21.8
```

One Dimensional Vector Arithmetic

Example 2.11 (Cont.): Indexing

- `#A Vector of Character Strings`
- `fruit <- c(5, 10, 1, 20)`
- `names(fruit) <- c("orange", "banana", "apple", "peach")`
- `fruit`
- `lunch <- fruit[c("apple","orange")]`
- `lunch`

One Dimensional Vector Arithmetic

Example 2.11 (Cont.): Indexing

```
> #A Vector of Character Strings
> fruit <- c(5, 10, 1, 20)
> names(fruit) <- c("orange", "banana", "apple", "peach")
> fruit
orange banana  apple  peach
     5     10      1     20
> lunch <- fruit[c("apple","orange")]
> lunch
 apple orange
     1      5
> |
```

One Dimensional Vector Arithmetic

- An indexed expression can also appear on **the receiving end of an assignment**, in which case the assignment operation is performed only on those elements of the vector.
- The expression must be of the form `vector[index_vector]`

One Dimensional Vector Arithmetic

Example 2.12: Indexing and Assignment of Vectors

- #Assignment of operators
- z
- is.na(z)
- z[is.na(z)] <- 0
- z

```
> #Assignment of operators
> z
[1]  1  2  3 NA
> is.na(z)
[1] FALSE FALSE FALSE  TRUE
> z[is.na(z)] <- 0
> z
[1] 1 2 3 0
```

One Dimensional Vector Arithmetic

- R caters for changes of data types almost anywhere it could be considered sensible to do so, (and a few where it might not be).
- To know the **current type** of the object use: `mode(object)`
- To convert a **numeric vector to a string** use: `as.character(vector)`
- To convert a character vector (with numeric characters) to **an integer vector** use: `as.integer(vector)`

One Dimensional Vector Arithmetic

Example 2.13: Changing Data Types

- `#Changing Data Types`
- `digits <- as.character(z)`
- `mode(digits)`
- `digits <- as.integer(digits)`
- `mode(digits)`

```
> #Changing Data Types
> digits <- as.character(z)
> mode(digits)
[1] "character"
> digits <- as.integer(digits)
> mode(digits)
[1] "numeric"
>
```

One Dimensional Vector Arithmetic

Definition 2.4: Factors

- A factor is a vector object used to specify a discrete classification (grouping) of the components of other vectors of the same length.
- A factor is created using the `factor()` function

One Dimensional Vector Arithmetic

Example 2.14: Factors

- Suppose that we are looking at 5 individuals and their corresponding monthly income. We would like to treat their gender as a factor.

Gender	Monthly Income (000)
M	30
F	29
M	35
M	20
F	40

One Dimensional Vector Arithmetic

Example 2.14 (Cont.): Factors

```
#Working with Factors
gender <- c("M","F","M","M","F")
genderfactors <- factor(gender)
levels(genderfactors)
#Get average income per factor
income <- c(30,29,35,20,40)
incmeans <- tapply(income, genderfactors, mean)
incmeans
```

One Dimensional Vector Arithmetic

Example 2.14 (Cont.): Factors

```
> #Working with Factors
> gender <- c("M","F","M","M","F")
> genderfactors <- factor(gender)
> levels(genderfactors)
[1] "F" "M"
> #Get average income per factor
> income <- c(30,29,35,20,40)
> incmeans <- tapply(income, genderfactors, mean)
> incmeans
       F        M
34.50000 28.33333
```

Outline for This Session

- One Dimensional Vector Arithmetic
- **Multi Dimensional Vector Arithmetic**
- Case Study

Multi Dimensional Vector Arithmetic

Definition 2.5: Arrays

- An array can be considered as a multiple subscripted collection of data entries
- Converting a one-dimensional vector into a two dimensional matrix using a non-negative dimensional vector we use the `dim()` function

eljalao@up.edu.ph

Multi Dimensional Vector Arithmetic

Example 2.15: Dimensional Arrays

- #2-Dimensional Array
- z
- zmatrix <- z
- dim(zmatrix) = c(2,2)
- zmatrix

```
> #2-Dimensional Array
> z
[1] 1 2 3 0
> zmatrix <- z
> dim(zmatrix) = c(2,2)
> zmatrix
     [,1] [,2]
[1,]    1    3
[2,]    2    0
```

Multi Dimensional Vector Arithmetic

- Individual elements of an array may be **referenced** by giving the name of the array followed by the subscripts in square brackets, separated by commas.
- More generally, **subsections** of an array may be specified by giving a sequence of index vectors in place of subscripts

Multi Dimensional Vector Arithmetic

Example 2.16: Subsetting Arrays

- Consider the following matrix , select item [1,3], [2,2], [3,1] and replace them with zeros.

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

Multi Dimensional Vector Arithmetic

Example 2.16: Subsetting Arrays

```
# Generate a 4 by 5 array.
xindex <- array(1:20, dim=c(4,5))
xindex
#select item [1,3], [2,2], [3,1]
indextoselect <- array(c(1:3,3:1),
dim=c(3,2))
xindex[indextoselect]
#replace with zeroes
xindex[indextoselect]<-0
xindex
```

Multi Dimensional Vector Arithmetic

```
> # Generate a 4 by 5 array.
> xindex <- array(1:20, dim=c(4,5))
> xindex
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
> #select item [1,3], [2,2], [3,1]
> indextoselect <- array(c(1:3,3:1), dim=c(3,2))
> xindex[indextoselect]
[1] 9 6 3
> #replace with zeroes
> xindex[indextoselect]<-0
> xindex
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    0   13   17
[2,]    2    0   10   14   18
[3,]    0    7   11   15   19
[4,]    4    8   12   16   20
> 
```

Multi Dimensional Vector Arithmetic

Definition 2.6: Matrix

- A matrix is just an array with two subscripts.
- R contains many operators and functions that are available only for matrices. For example `t(X)` is the matrix transpose function, as noted above.
- The functions `nrow(A)` and `ncol(A)` give the number of rows and columns in the matrix A respectively.

Multi Dimensional Vector Arithmetic

Example 2.17: Transpose of a Matrix

- `#To Transpose a Matrix`
- `xindex`
- `xtranspose = t(xindex)`
- `xtranspose`
- `nrow(xtranspose)`
- `ncol(xtranspose)`

Multi Dimensional Vector Arithmetic

```
> #To Transpose a Matrix
> xindex
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    0   13   17
[2,]    2    0   10   14   18
[3,]    0    7   11   15   19
[4,]    4    8   12   16   20
> xtranspose = t(xindex)
> xtranspose
     [,1] [,2] [,3] [,4]
[1,]    1    2    0    4
[2,]    5    0    7    8
[3,]    0   10   11   12
[4,]   13   14   15   16
[5,]   17   18   19   20
> nrow(xtranspose)
[1] 5
> ncol(xtranspose)
[1] 4
>
```

Multi Dimensional Vector Arithmetic

- Review of Linear Algebra
  - Let $x = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $y = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix}$
  - Element-wise product

$$x * y = \begin{bmatrix} 1*4 & 2*3 \\ 3*2 & 4*1 \end{bmatrix} = \begin{bmatrix} 4 & 6 \\ 6 & 4 \end{bmatrix}$$

  - Matrix Inner Product

$$x \circ y = \begin{bmatrix} 1*4+2*2 & 1*3+2*1 \\ 3*4+2*4 & 3*3+4*1 \end{bmatrix} = \begin{bmatrix} 8 & 5 \\ 20 & 13 \end{bmatrix}$$

  - Vector*Matrix Inner Product, $z = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$$z^T * x = \begin{bmatrix} 1*1+3*2 & 1*2+4*2 \end{bmatrix} = \begin{bmatrix} 7 & 10 \end{bmatrix}$$

Multi Dimensional Vector Arithmetic

- The * operator is the element-wise product multiplication
  - `A * B`
- While the following is a matrix inner product
  - `A %*% B`

Multi Dimensional Vector Arithmetic

Example 2.18: Matrix Multiplications

- `x<- array(c(1,3,2,4), dim=c(2,2))`
- `y<- array(c(4,2,3,1), dim=c(2,2))`
- `z<- array(c(1,2),dim=c(2,1))`
- `#Element Wise Multiplication`
- `x*y`
- `#Matrix Inner Product`
- `x%*%y`
- `#Vector Times Matrix Multiplication`
- `t(z)%*%x`

Multi Dimensional Vector Arithmetic

```
> #Matrix Multiplications
> x<- array(c(1,3,2,4), dim=c(2,2))
> y<- array(c(4,2,3,1), dim=c(2,2))
> z<- array(c(1,2),dim=c(2,1))
> #Element Wise Multiplication
> x*y
     [,1] [,2]
[1,]    4    6
[2,]    6    4
> #Matrix Inner Product
> x%*%y
     [,1] [,2]
[1,]    8    5
[2,]   20   13
> #Vector Times Matrix Multiplication
> t(z)%*%x
     [,1] [,2]
[1,]    7   10
>
```

Multi Dimensional Vector Arithmetic

- The function `cbind()` forms matrices by binding together matrices horizontally, or column-wise, and `rbind()` vertically, or row-wise.
  - `X <- cbind(arg_1, arg_2, arg_3, ...)`
- the arguments to `cbind()` must be either vectors of any length, or matrices with the same number of rows.
- The result is a matrix with the concatenated arguments `arg 1, arg 2, . . .` forming the columns.

Multi Dimensional Vector Arithmetic

- The function `rbind()` does the corresponding operation for rows. In this case any vector argument are of course taken as row vectors.
- Suppose `X1` and `X2` have the same number of rows. To combine these by columns into a matrix `X`, together with an initial column of 1s we can use
  - `X <- cbind(1, X1, X2)`
- The result of `rbind()` or `cbind()` is always a matrix. Hence `cbind(x)` and `rbind(x)` are possibly the simplest ways explicitly to allow the vector `x` to be treated as a column or row matrix respectively.

Multi Dimensional Vector Arithmetic

Example 2.19: CBIND and RBIND

- #cbind and rbind
- x
- y
- colbinded = cbind(x,y)
- colbinded
- rowbinded = rbind(x,y)
- rowbinded

```
> #cbind and rbind
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> y
     [,1] [,2]
[1,]    4    3
[2,]    2    1
> colbinded = cbind(x,y)
> colbinded
     [,1] [,2] [,3] [,4]
[1,]    1    2    4    3
[2,]    3    4    2    1
> rowbinded = rbind(x,y)
> rowbinded
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    4    3
[4,]    2    1
>
```

Multi Dimensional Vector Arithmetic

- The function `table()` allows frequency tables to be calculated from equal length factors.
- If there are k factor arguments, the result is a k-way array of frequencies.

Multi Dimensional Vector Arithmetic

Example 2.20: Tables and Frequencies

- `#frequency`
- `table(genderfactors)`

```
> #frequency
> table(genderfactors)
genderfactors
F M
2 3
>
```

Outline for This Session

- One Dimensional Vector Arithmetic
- Multi Dimensional Vector Arithmetic
- **Case Study**

Case Study 1

- R Scripting Exercises

Outline for This Session

- One Dimensional Vector Arithmetic
- Multi Dimensional Vector Arithmetic
- Case Study

References

- http://www.r-tutor.com

eljalao@up.edu.ph