Introduction to R and Data Objects

2017.09.13

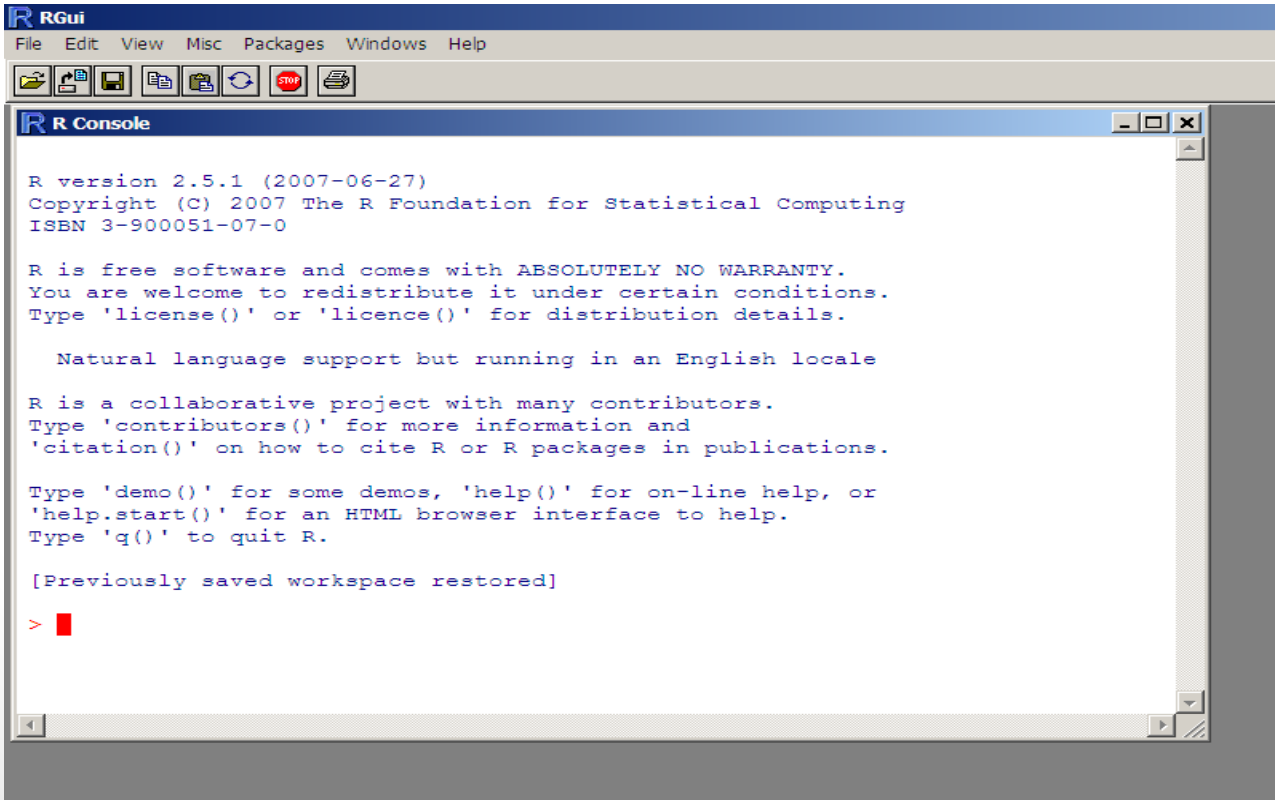
서울아산병원 임상약리학과
임형석

What is R?

- R is a free software environment for statistical computing and graphics
- It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.
- Download R from CRAN (Comprehensive R Archive Network)
  - http://www.r-project.org/

Opening R

- Click the R icon located in the programs directory (e.g., C:\Program Files\R\R-3.4.1) or located on your desktop



Note

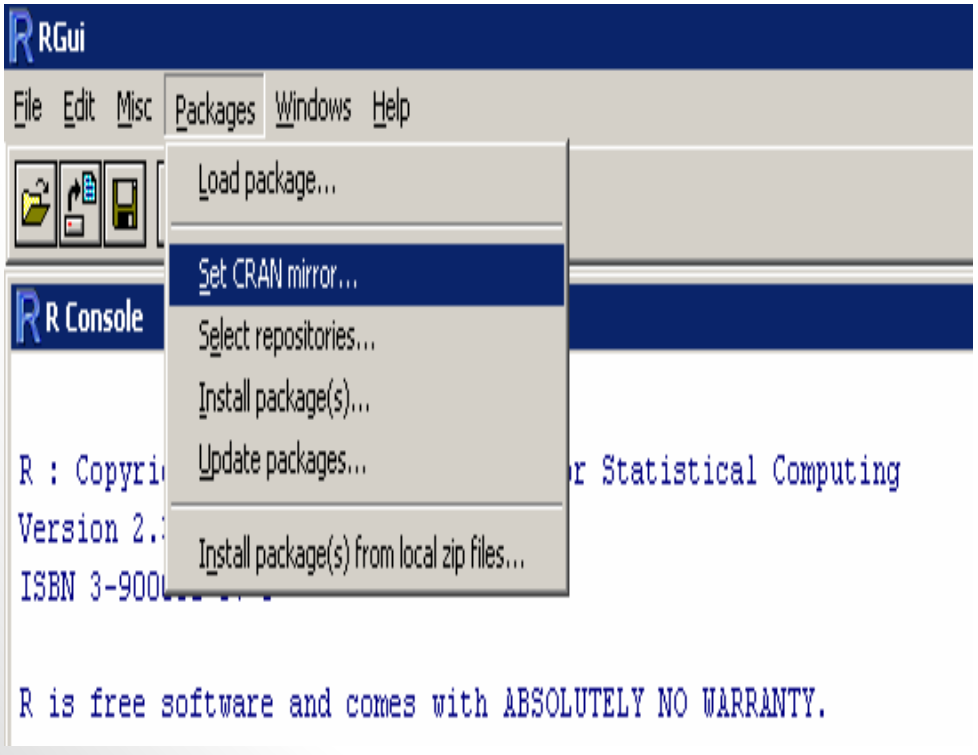
- R discriminates between upper case and lower case letters
- () for function
- [] for data
- # Notation-out

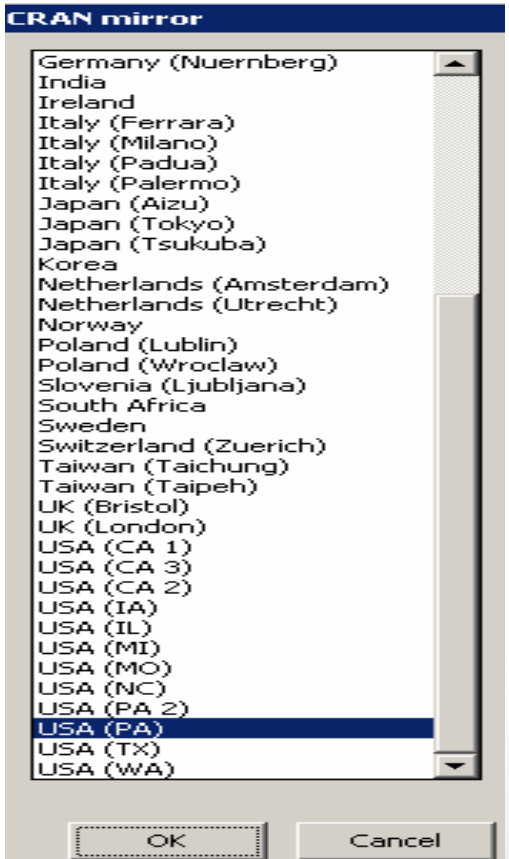
Getting help

- help(plot)
- ?plot
- args(plot)

R packages

- R packages provide a wide variety of enhancements from new functions for analysis and reporting





R packages

- Install packages
  - e.g. install.packages(“survival”)
  - C:\Program Files\R\R-3.4.1\library
- Only when a package is loaded are its contents available.
  - e.g. library(survival)
- Which packages are currently loaded?
  - search()

Setting the Working Directory

- Setting the Working Directory
  - setwd("D:\\AMC\\Education\\UU\\2015")
  - setwd("D:/AMC/Education/UU/2015")

- To check your current working directry
  - getwd()

Data Types

Mode	Description
Numeric	Integer or real numbers
Character	Character strings
Complex	Complex numbers
Logical	Logical data (True or False)

Data Types

```
> pi # Numeric
[1] 3.141593
>
> "Hello" # Character
[1] "Hello"
>
> 3 + 5i # Complex
[1] 3+5i
>
> 3 > 5 # Logical
[1] FALSE
```

```
> mode(pi)
[1] "numeric"
>
> mode("Hello")
[1] "character"
>
> mode(4>5)
[1] "logical"
>
>
```

- Missing Data : **Missing data element -> Assigned "NA" character**

```
> c(T, F, NA) # Logical
[1]  TRUE FALSE    NA
>
> c(1, 2, NA) # Numeric
[1]  1  2 NA
>
> c(1+4i, 2-3i, NA) # Complex
[1] 1+4i 2-3i   NA
>
> c("A", "B", NA) # Character
[1] "A" "B" NA
```

Primary Structures of R data Objects

Structure	Type	Description
Vectors	Single Mode	Series of elements
Matrices	Single Mode	Rectangular structure
Arrays	Single Mode	3 or more dimensions
Lists	Multi Mode	Container for other objects
Data Frames	Multi Mode	Rectangular structure

Vectors

- Structure

  1) Single “column” of data

  2) Holding only 1 type of data

- Creating Vectors
- Combining Elements

```
> c(4, 3, 6, 5) # A simple numeric vector
[1] 4 3 6 5
>
> c("A", "B", "C") # A simple character vector
[1] "A" "B" "C"
>
> c(F, T, F) # A simple logical vector
[1] FALSE  TRUE FALSE
>
> c(3, 1, 4, "Hello") # Trying to mix data modes
[1] "3"     "1"     "4"     "Hello"
>
> mode(c(3, 1, 4, "Hello"))
[1] "character"
> myVector1 <- c(8, 3, 4, 1, 5, 4, 0, -3, 6, -3) # Print out the myVector1 object
>
> myVector1
 [1]  8  3  4  1  5  4  0 -3  6 -3
>
> myVector2 <- c(1, 5, myVector1, 3, 2) # Print out the myVector2 object
>
> myVector2
 [1]  1  5  8  3  4  1  5  4  0 -3  6 -3  3  2
```

Vectors

- The seq function ⇒ Creating a sequence of numbers, with user specified intervals

Argument	Description	Default
from	Starting value	1
To	End value	1
By	Size of interval	Depends on length
length	Length of resulting vector	Depends on by

```
> seq(1,5) # Equivalent to 1:5
[1] 1 2 3 4 5
>
> seq(1, 5, by=0.5) # Sequence from 1 to 5 by 0.5
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
>
> seq(1, 5, length=9) # 9 evenly spaced numbers between 1 and 5
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
>
> seq(10, 1, by=-0.5) # Sequence from 10 down to 1 by 0.5
 [1] 10.0  9.5  9.0  8.5  8.0  7.5  7.0  6.5  6.0  5.5  5.0  4.5  4.0  3.5  3.0  2.5  2.0  1.5  1.0
```

Vectors

- Repeating Vectors

☞ The rep function ⇒ Replicating a vector a given number of times

Argument	Description
x	Starting vector
times	End value
length	Size of interval
each	Length of resulting vector

```
> rep("A", 5) # Repeat "1" 5 times
[1] "A" "A" "A" "A" "A"
>
> rep(1:3, 3) # Repeat "1 to 3" 3 times
[1] 1 2 3 1 2 3 1 2 3
>
> rep(1:3, 1:3) # Repeat "1" once, "2" twice, "3" 3 times
[1] 1 2 2 3 3 3
>
> rep(1:3, 3) # Repeat "1 to 3" 3 times
[1] 1 2 3 1 2 3 1 2 3
>
> rep(1:3, c(3, 3, 3)) # Repeat "1 to 3" 3 times each
[1] 1 1 1 2 2 2 3 3 3
>
> rep(1:3, rep(3, 3)) # Repeat "1 to 3" 3 times each
[1] 1 1 1 2 2 2 3 3 3
>
> rep(1:3, each=3) # Repeat "1 to 3" 3 times each
[1] 1 1 1 2 2 2 3 3 3
```

Vectors

- Vector Attributes

Function	Description
length	Number of elements in our vector
names	Names of the vector elements
mode	Type of data held in the vector

```
> myVector <- c(6, 3, 4, 8, 5, 2, 7, 9, 4, 5) # Create myVector
>
> myVector # Print myVector
 [1] 6 3 4 8 5 2 7 9 4 5
>
> mode(myVector) # Vector mode
[1] "numeric"
>
> length(myVector) # No. of elements
[1] 10
>
> names(myVector) # Element names
NULL
>
> names(myVector) <- c("A", "B", "C", "D", "E", "F", "G", "H", "I", "J")
> myVector # Print myVector
A B C D E F G H I J
6 3 4 8 5 2 7 9 4 5
```

Vectors

- Subscripting Vectors
- To easily extract required information ⇒ using square brackets
- The square brackets can contain one of 5 possible types of information

Subscript Type	Description
Blank	Return all elements
Vector of positive integers	Indices of elements to return
Vector of negative integers	Indices of elements to omit
Vector of logical values	Only “TRUE” values are return
Vector of characters	Names of elements to return

Vectors

- Subscripting Vectors

```
> myVector[]
A B C D E F G H I J
6 3 4 8 5 2 7 9 4 5
>
> LETTERS[]
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
>
> letters[]
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
>
> pi[]
[1] 3.141593
```

- Positive Integers

```
> myVector[1] # The first element
A
6
>
> myVector[1:5] # The first 5 elements
A B C D E
6 3 4 8 5
>
> myVector[1:length(myVector)] # All elements
A B C D E F G H I J
6 3 4 8 5 2 7 9 4 5
>
> myVector[length(myVector):1] # Elements in reverse order
J I H G F E D C B A
5 4 9 7 2 5 8 4 3 6
>
> myVector[c(1,5)] # The first and fifth elements
A E
6 5
```

Vectors

- Negative Integers

```
> myVector[-1] # All but first element
B C D E F G H I J
3 4 8 5 2 7 9 4 5
>
> myVector[-length(myVector)] # All but the last element
A B C D E F G H I
6 3 4 8 5 2 7 9 4
>
> myVector[-(1:3)] # All but the first 3 elements
D E F G H I J
8 5 2 7 9 4 5
```

- Logical Values

```
> myVector
A B C D E F G H I J
6 3 4 8 5 2 7 9 4 5
>
> myVector[c(T,F,F,T,F,F,T,T,F,F)]
A D G H
6 8 7 9
>
> LETTERS[c(T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F)]
 [1] "A" "C" "E" "G" "I" "K" "M" "O" "Q" "S" "U" "W" "Y"
>
> letters[c(T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F,T,F)]
 [1] "a" "c" "e" "g" "i" "k" "m" "o" "q" "s" "u" "w" "y"
```

Vectors

- Logical Values

```
> myVector > 5
    A     B     C     D     E     F     G     H     I     J
 TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE
>
> myVector <= 4
    A     B     C     D     E     F     G     H     I     J
FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
>
> myVector != 4 # != : not equal
    A     B     C     D     E     F     G     H     I     J
 TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
>
> myVector == 5 # == : equal
    A     B     C     D     E     F     G     H     I     J
FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE
>
> myVector > 2 & myVector < 9 # & : and
    A     B     C     D     E     F     G     H     I     J
 TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE
>
> myVector < 3 | myVector > 6 # | : or
    A     B     C     D     E     F     G     H     I     J
FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE
> myVector[myVector > 5]
A D G H
6 8 7 9
>
> myVector[myVector != 4]
A B D E F G H J
6 3 8 5 2 7 9 5
```

Vectors

- Logical Values

```
> vec1 <- c("A","B","A","A","C")
> vec2 <- c(5,2,3,7,4)
>
> length(vec1) # Vector is of length 5
[1] 5
>
> length(vec2) # Vector is of length 5
[1] 5
>
> vec1 == "A"
[1]  TRUE FALSE  TRUE  TRUE FALSE
>
> vec1[vec1 == "A"] # Elements of vec1 which are equal to "A"
[1] "A" "A" "A"
>
> vec2[vec1 == "A"] # Elements of vec2 where vec1 equal to "A"
[1] 5 3 7
>
> vec2 > 3
[1]  TRUE FALSE FALSE  TRUE  TRUE
> vec2[vec2 > 3] # Elements of vec2 which are greater than 3
[1] 5 7 4
>
> vec1[vec2 > 3] # Elements of vec1 where vec2 greater than 3
[1] "A" "A" "C"
```

Vectors

- Character Vectors

```
> myVector[c("A","B","E")] # Select A, B and E elements
A B E
6 3 5
>
> myVector[LETTERS[1:3]] # Equivalent to myVector[1:3] here
A B C
6 3 4
```

Matrices

- Structure

1) 2-dimensional data structure with rows and columns

2) Single mode

- Creating a Matrix
- Binding rows or columns

☞ cbind ⇒ combining vectors as columns of a matrix

☞ rbind ⇒ combining vectors as rows of a matrix

```
> cbind(1:3, c(6,3,4), -1:1) # Column bind 3 vectors
     [,1] [,2] [,3]
[1,]    1    6   -1
[2,]    2    3    0
[3,]    3    4    1
>
> rbind(1:3, 4:6, 7:9) # Row bind 3 vectors
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> cbind(1:3, 4:6, 3) # The last vector is repeated
     [,1] [,2] [,3]
[1,]    1    4    3
[2,]    2    5    3
[3,]    3    6    3
```

Matrices

- Building a Matrix from a vector

Logical Operator	Description	Default
data	Input vector	NA
nrow	Number of rows	1
ncol	Number of columns	1
byrow	Read in “by row”?	F
dimnames	Dimension names	NULL

```
> myMatrix <- matrix(1:9, nrow=3) # Create matrix
>
> dim(myMatrix)
[1] 3 3
>
> nrow(myMatrix)
[1] 3
>
> dimnames(myMatrix) <- list(LETTERS[1:3], c("x1","x2","x3"))
> myMatrix # Matrix with dimension names
  x1 x2 x3
A  1  4  7
B  2  5  8
C  3  6  9
```

Matrices

- Subscripting Matrices
- Using a comma to separate the subscripting information within the square brackets
- Matrix[row subscripts, column subscripts]
- Blank

```
> myMatrix[ , ] # Blank
  x1 x2 x3
A  1  4  7
B  2  5  8
C  3  6  9
```

- Positive Integers

```
> myMatrix[1:2, ] # Rows 1 and 2, all columns
  x1 x2 x3
A  1  4  7
B  2  5  8
>
> myMatrix[ , 2:3] # All rows, columns 2 and 3
  x2 x3
A  4  7
B  5  8
C  6  9
>
> myMatrix[1:2,2:3] # Rows 1 and 2, columns 2 and 3
  x2 x3
A  4  7
B  5  8
```

```
> myMatrix[ , 1]  # All rows, column 1
A B C
1 2 3
>
> myMatrix[ , 1, drop=F] # Returns a 1-column matrix
  x1
A  1
B  2
C  3
```

Matrices

- Negative Integers

```
> myMatrix[ , -1] # All rows, all but the first column
  x2 x3
A  4  7
B  5  8
C  6  9
>
> myMatrix[-3, -3] # First 2 rows, first 2 columns
  x1 x2
A  1  4
B  2  5
>
> myMatrix[ , -(2:3)] # First column only (returns a vector)
A B C
1 2 3
>
> myMatrix[ , -(2:3), drop=F] # First column only (returns a matrix)
  x1
A  1
B  2
C  3
```

Matrices

- Logical Values

```
> myMatrix[ , 1] # First column of myMatrix
A B C
1 2 3
>
> myMatrix[ , 1] > 1 # Generating a logical vector based on column 1
    A     B     C
FALSE  TRUE  TRUE
>
> myMatrix[myMatrix[ , 1] > 1 , ] # Subscripting using logical vector
  x1 x2 x3
B  2  5  8
C  3  6  9
```

- Character Values

```
> myMatrix[ , "x1"] # All rows, only the "x1" column
A B C
1 2 3
>
> myMatrix[c("A","B"), c("x1","x2")] # All rows, only the "x1" column
  x1 x2
A  1  4
B  2  5
>
> myMatrix[LETTERS[2:3], ] # Rows "B" and "C"
  x1 x2 x3
B  2  5  8
C  3  6  9
```

Arrays

- Structure

1) N-dimensional Single mode data structure

2) Creating Arrays

```
> myArray <- array(1:18, c(3,3,2))
> myArray
, , 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
```

- Array Attributes

Function	Description
length	Total number of elements in our array
dim	Structure “dimension” c(rows, columns, 3D,…)
mode	Mode of data stored
dimnames	Dimension names (a list containing “N” vectors

Arrays

- Subscripting Arrays
  - Extracting data from arrays using similar syntax to the matrix notation, with “N-1” commas within the square brackets

☞ Array[row subscripts, column subscripts, 3D subscripts]

Lists

- Structure
  - ☞ Container in which to hold other objects

```
> unnamedList <- list(myVector, myMatrix)
> unnamedList
[[1]]
A B C D E F G H I J
6 3 4 8 5 2 7 9 4 5

[[2]]
  x1 x2 x3
A  1  4  7
B  2  5  8
C  3  6  9

>
>
> length(unnamedList) # Number of elements in list
[1] 2
>
> names(unnamedList) # Names of elements in list
NULL
```

```
> namedList <- list(x = myVector, y = myMatrix)
> namedList
$x
A B C D E F G H I J
6 3 4 8 5 2 7 9 4 5

$y
  x1 x2 x3
A  1  4  7
B  2  5  8
C  3  6  9

>
>
> names(namedList)
[1] "x" "y"
```

- List attribute

Function	Description	namedList	unnamedList
length	Number of list elements	2	2
names	Names of list elements	c(“x”, “y”)	NULL

Lists

- Subscripting Lists

☞ 2 ways in which we can extract data from a list

1) Using the double square brackets notation

2) Using the $ symbol to directly extract the elements

```
> unnamedList[[1]] # First element of list
A B C D E F G H I J
6 3 4 8 5 2 7 9 4 5
> unnamedList[[2]] # Second element of list
  x1 x2 x3
A  1  4  7
B  2  5  8
C  3  6  9
> namedList$x # The "x" element of list
A B C D E F G H I J
6 3 4 8 5 2 7 9 4 5
> namedList$y # The "y" element of list
  x1 x2 x3
A  1  4  7
B  2  5  8
C  3  6  9
```

Lists

- Adding List Elements

☞ Directly adding elements to a list using the $ or [[]] approaches

```
> unnamedList[[3]] <- 1:5 # Add 3rd element to list
> unnamedList # Print unnamed list
[[1]]
A B C D E F G H I J
6 3 4 8 5 2 7 9 4 5

[[2]]
  x1 x2 x3
A  1  4  7
B  2  5  8
C  3  6  9

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

```
> namedList$x2 <- 1:5 # Add "x2" element to list
> namedList # Print list
$x
A B C D E F G H I J
6 3 4 8 5 2 7 9 4 5

$y
  x1 x2 x3
A  1  4  7
B  2  5  8
C  3  6  9

$x2
[1] 1 2 3 4 5
```

Data Frames

- Comparisons with Lists

☞ A “Data Frame” is a list structure with the following additional “rules”

1) Every list element must have a name

2) It can only hold vector of data

3) All elements (vectors) must be of the same length

```
> myDataset <- list(x=1:10, y=letters[1:10])
> myDataset
$x
 [1]  1  2  3  4  5  6  7  8  9 10

$y
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

Data Frames

- Creating Data Frames
  - ☞ Creating a data frame using the data.frame function

```
> myDf <- data.frame(x=1:10, y=letters[1:10]) # Create data frame
> myDf # Print data frame
    x y
1   1 a
2   2 b
3   3 c
4   4 d
5   5 e
6   6 f
7   7 g
8   8 h
9   9 i
10 10 j
> length(myDf) # Number of elements (columns)
[1] 2
> names(myDf) # Names of elements (columns)
[1] "x" "y"
```

Data Frames

- Data Frame Attributes

☞ 2 very useful functions to extract information about the data frame's structure

Function	Description	myDf
length	Number of list elements (columns)	2
names	Names of list elements (columns)	c("x", "y")

- Subscripting Data Frames

```
> myDf$x # Extract the x column
 [1]  1  2  3  4  5  6  7  8  9 10
> myDf$y # Extract the y column
 [1] a b c d e f g h i j
Levels: a b c d e f g h i j
> myDf$y[myDf$x > 4] # Elements of y where x greater than 4
[1] e f g h i j
Levels: a b c d e f g h i j
```

Data Frames

- Subscripting Data Frames

☞ Referencing data within a data frame using square brackets with row and column subscripts

```
> myDf[1:3, ] # First 3 rows of myDf
  x y
1 1 a
2 2 b
3 3 c
> myDf[ , 1] # Column 1-returns a vector
 [1]  1  2  3  4  5  6  7  8  9 10
> myDf[ , 1, drop=F] # Column 1-returns a data frame
    x
1   1
2   2
3   3
4   4
5   5
6   6
7   7
8   8
9   9
10 10
```

Basic Calculations

```
> 2+2
[1] 4
> 4-2
[1] 2
> 2*3
[1] 6
> 6/3
[1] 2
```

```
> 10^3
[1] 1000
> sqrt(4)
[1] 2
> abs(-5)
[1] 5
> log(exp(1))
[1] 1
> log(10, 10)
[1] 1
> exp(1)
[1] 2.718282
```

Vectorized Calculations

```
> a <- 1:5
> a + 2
[1] 3 4 5 6 7
> a - 2
[1] -1  0  1  2  3
> a*3
[1]  3  6  9 12 15
> a/5
[1] 0.2 0.4 0.6 0.8 1.0
> a^3
[1]   1   8  27  64 125
```

```
> sqrt(a)
[1] 1.000000 1.414214 1.732051
2.000000 2.236068
> abs(-a)
[1] 1 2 3 4 5
> log(exp(a))
[1] 1 2 3 4 5
> log(a, 10)
[1] 0.0000000 0.3010300 0.4771213
0.6020600 0.6989700
> exp(a)
[1]   2.718282   7.389056  20.085537
54.598150 148.413159
```

Basic Functions

```
vec <- 1:10
> mean(vec)
[1] 5.5
> median(vec)
[1] 5.5
> min(vec)
[1] 1
> max(vec)
[1] 10
> range(vec)
[1]  1 10
> sd(vec)
[1] 3.02765
> var(vec)
[1] 9.166667
> sum(vec)
[1] 55
> length(vec)
[1] 10
> quantile(vec,0.95)
 95%
9.55
> pi
[1] 3.141593
```