COMP 1433: Introduction to Data Analytics &
COMP 1003: Statistical Tools and Applications

Lecture 6 – Programming with R: Basics, Data Input, and Manipulation

Dr. Jing Li
Department of Computing
The Hong Kong Polytechnic University

22 & 24 Feb 2022

Why Programing?

- We are facing large-scale data!
  - Big data available from Internet, Social Media, etc.
  - Large sample to understand probability!
- Impossible to be analyzed manually!
- We need computer's help to do mathematics for us.
- But we need to tell it how to help us!
  - With a *language* between humans and computers.

Why R Programing?

- First thing first, *Free*!
- Runs on a *variety of platforms* including *Windows*, *Unix* and *MacOS*.
- Provides an unparalleled platform for programming new statistical methods in an *easy and straightforward* manner.
- Contains *advanced statistical routines* not yet available in other packages.
- It has *state-of-the-art graphics capabilities*.

Roadmap

- R Basics
  - Programming Environments
  - R Commands
  - Datasets and Packages
- Data Input and Manipulation
  - Data Types
  - Data Input and Processing
  - Data Manipulation

Roadmap

- R Basics
  - Programming Environments
  - R Commands
  - Datasets and Packages
- Data Input and Manipulation
  - Data Types
  - Data Input and Processing
  - Data Manipulation

Roadmap

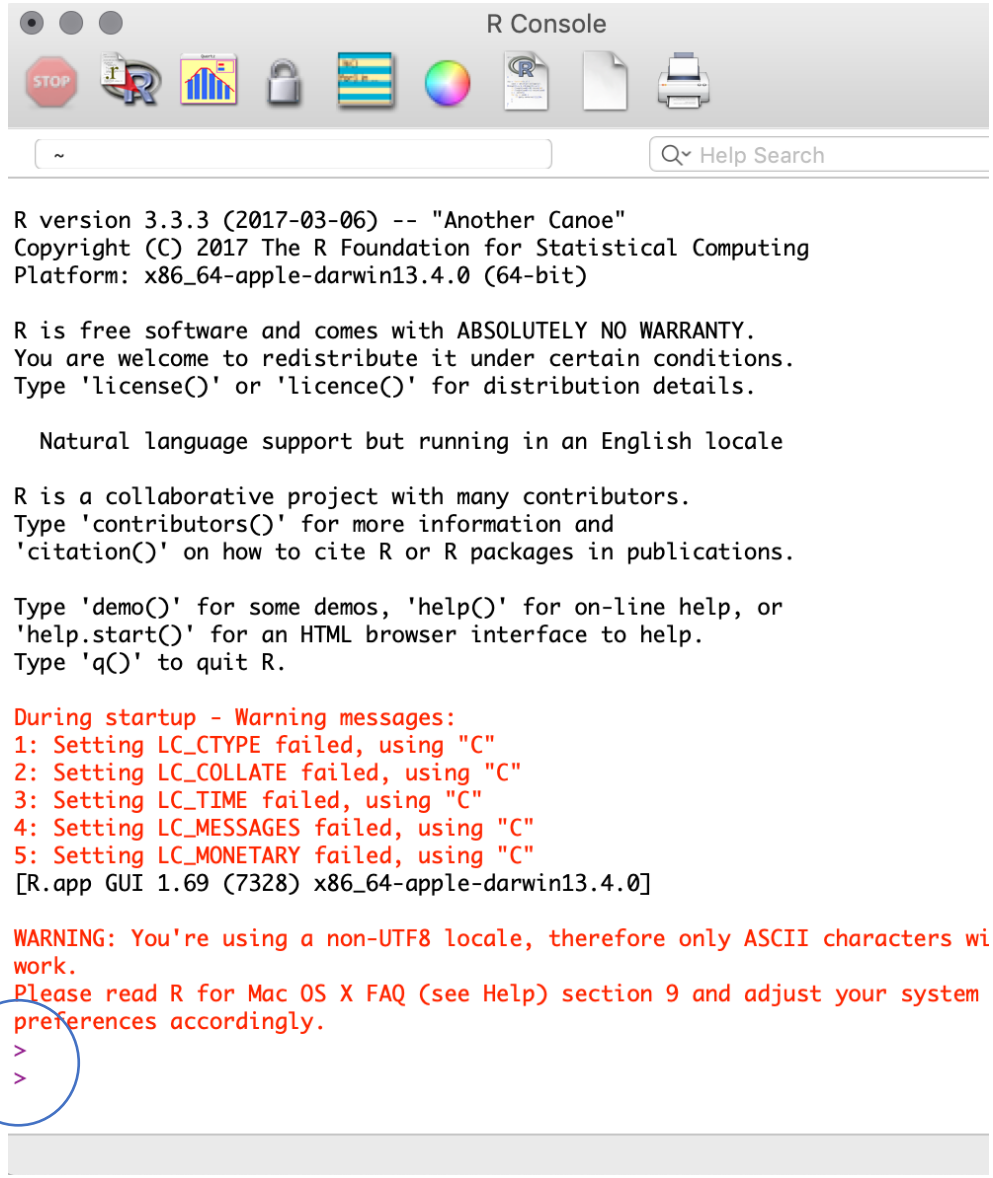
- R Basics
  - Programming Environments
  - R Commands
  - Datasets and Packages
- Data Input and Manipulation
  - Data Types
  - Data Input and Processing
  - Data Manipulation

R Overview

- Enter commands:
  - *One at a time* at the command prompt (>)
  - *Run a set of commands* from a source file
- Support a wide variety of data types
  - such as *vectors* (*numerical*, *character*, *logical*), *matrices*, *dataframes*, and *lists*.
- To quit R, use
  - *>q()*

R Interface

- Start the R system, the window with R Console will appear
- In the `*Consol*e' window the cursor is waiting for you to type in some R commands.

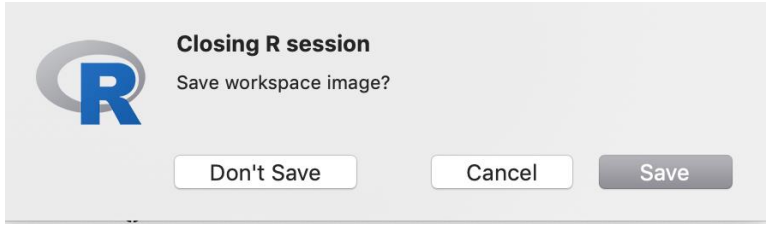


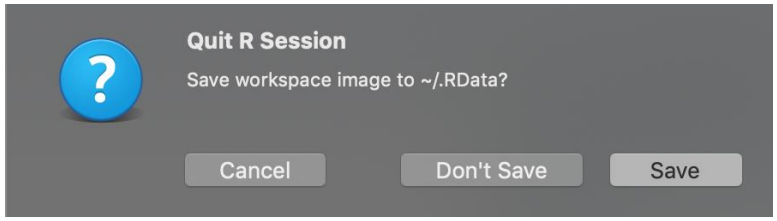
R Workspace

- Objects that you create during an R session are held in *memory*
  - **Workspace:** *the collection of objects you currently have*
- This *workspace* is not saved on disk unless you tell R to do so.
- This means that your objects are *lost* when you close R and not save the objects, or worse when R or your system crashes on you during a session.

R Workspace –Save

- When you *close* the *R IDE* (e.g., RGui and RStudio), or *R console window*, the system will ask if you want to save the workspace image.
- If you select to save the *workspace image* then all the objects in your current R session are saved in a file named *.RData*.
  - This is *a binary file* located in the working directory of R, which is *by default the installation directory of R*.





R Workspace - Save

- During your R session you can also explicitly save the workspace image.
- Go to the `File‘ menu and then select `Save Workspace...’, or use the save.image function.

```
## save to the current working directory
save.image()
## just checking what the current working directory is
getwd()
## save to a specific file and location
save.image("C:\\Program Files\\R\\R-2.5.0\\bin\\.RData")
```

R Workspace - Save

- *getwd()* # print the current working directory
- *ls()* # list the objects in the current workspace
- *setwd(mydirectory)* # change to *mydirectory*
  - setwd("*c:/docs/mydir*")

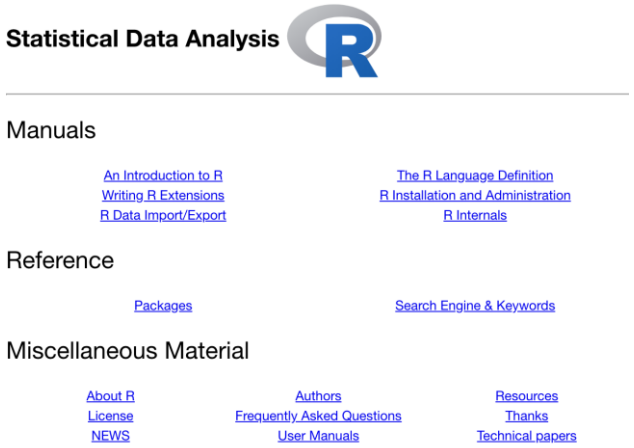
R Workspace – Load

- If you have saved a workspace image and you start R the next time, it will restore the workspace.
  - So all your previously saved objects are available again.
- You can also explicitly load a saved workspace, that could be the workspace image of someone else.
- Select `*Load workspace file…*' (e.g., under workspace menu for *RGui*).
- Or ***run command***:

  load("C:\\Program Files\\R\\R-2.5.0\\bin\\.RData")

R Help

- Once **R** is installed, there is a comprehensive built-in help system.
- At the program's command prompt you can use any of the following:




```
help.start()   # general help
help(foo)      # help about function foo
?foo           # same thing
apropos("foo") # list all function containing string foo
example(foo)   # show an example of function foo
```

Roadmap

- R Basics
  - Programming Environments
  - R Commands
  - Datasets and Packages
- Data Input and Manipulation
  - Data Types
  - Data Input and Processing
  - Data Manipulation

R Commands

- Results of calculations can be stored in objects using the assignment operators:
  - An arrow (<-) formed by a smaller than character and a hyphen without a space!
  - Or the equal character (=).
- These objects can then be used in other calculations.
- To print the object just enter the name of the object.

R Commands

- There are some restrictions when naming an object:
  - Object names cannot contain `*strange*' symbols like !, +, -, #.
  - A dot (.) and an underscore ( ) are allowed, also a name starting with a dot.
  - Object names can contain a number but cannot start with a number.
  - R is case sensitive, X and x are two different objects, as well as temp and temP.

Example R Commands

Code Comments

```
> # An example
> x <- c(1:10)
> x[(x>8) | (x<5)]
> # yields 1 2 3 4 9 10
```

```
> # How it works
> x <- c(1:10)
> x
>1 2 3 4 5 6 7 8 9 10
> x > 8
> F F F F F F F F T T
> x < 5
> T T T T F F F F F F
> x > 8 | x < 5
> T T T T F F F F T T
> x[c(T,T,T,T,F,F,F,F,T,T)]
> 1 2 3 4 9 10
```

x=c(1:10)=[1,2,3,4,5,6,7,8,9,10]

R Commands

- To list the objects that you have in your current R session use the function *ls* or *the function objects*.
  - > ls()
  - [1] "x" "y"
- So to run the function *ls* we need to enter the name followed by an opening (and a closing).
- Most functions in R accept certain arguments.
  - For example, one of the arguments of the function *ls* is pattern. To list all objects starting with the letter x:

```
> x2 = 9
> y2 = 10
> ls(pattern="x")
[1] "x2"
```

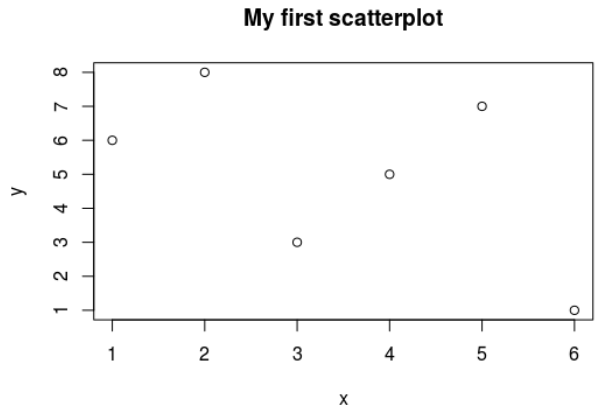
R Commands

- If you assign a value to an object that already exists, then the contents of the object will be overwritten with the new value (without a warning!).
- Use the function rm to remove one or more objects from your session.

```
> rm(x, x2)
```

- Let's create two small vectors with data and a scatterplot.

```
x <- c(1,2,3,4,5,6)
y <- c(6,8,3,5,7,1)
plot(x,y)
title("My first scatterplot")
```



Example R Commands

```
> x = sin(9)/75
> y = log(x) + x^2
> x
[1] 0.005494913
> y
[1] -5.203902
```

```
> A <- matrix(c(1,2,4,1), ncol=2)
> A
>    [,1] [,2]
[1,]    1    4
[2,]    2    1
> b <- c(2,1)
> solve(A, b)
[1] 0.2857143 0.4285714
```

Find the solution x for $Ax = b$

R Conflicting Objects

- R allows the user to give an object a name that already exists.
- R will not warn you when you use an existing name.

```
> mean = 10
> mean
[1] 10
```

- The object *"mean"* already exists in the base package. But is now masked by your object mean.
- To get a list of all masked objects use the function conflicts.

```
> conflicts
[1] "body<-" "mean"
```

R Conflicting Objects

- The object mean already exists in the base package. But is now masked by your object mean.
- To get a list of all masked objects use the function conflicts.

  > conflicts

  [1] "body<-" "mean“

- **What if you face conflicts?**
  - You can safely remove the object mean with the function *rm()* without risking deletion of the mean function.
  - Calling *rm()* removes only objects in your working environment by default.
  - E.g., *> rm(mean)*

Roadmap

- R Basics
  - Programming Environments
  - R Commands
  - Datasets and Packages
- Data Input and Manipulation
  - Data Types
  - Data Input and Processing
  - Data Manipulation

R Datasets

- R comes with some sample datasets that you can experiment with.
- Type

  > data( )

to see the available datasets. The results will depend on which packages you have loaded.

- Type

  > help("CO2")

for details on a sample dataset.

```
Data sets in package 'datasets':

AirPassengers               Monthly Airline Passenger Numbers 1949-1960
BJsales                     Sales Data with Leading Indicator
BJsales.lead (BJsales)      Sales Data with Leading Indicator
BOD                         Biochemical Oxygen Demand
CO2                         Carbon Dioxide Uptake in Grass Plants
ChickWeight                 Weight versus age of chicks on different diets
DNase                       Elisa assay of DNase
EuStockMarkets              Daily Closing Prices of Major European Stock Indices,
                            1991-1998
Formaldehyde                Determination of Formaldehyde
HairEyeColor                Hair and Eye Color of Statistics Students
Harman23.cor                Harman Example 2.3
Harman74.cor                Harman Example 7.4
Indometh                    Pharmacokinetics of Indomethacin
InsectSprays                Effectiveness of Insect Sprays
JohnsonJohnson              Quarterly Earnings per Johnson & Johnson Share
LakeHuron                   Level of Lake Huron 1875-1972
LifeCycleSavings            Intercountry Life-Cycle Savings Data
Loblolly                    Growth of Loblolly pine trees
Nile                        Flow of the River Nile
Orange                      Growth of Orange Trees
OrchardSprays               Potency of Orchard Sprays
PlantGrowth                 Results from an Experiment on Plant Growth
Puromycin                   Reaction Velocity of an Enzymatic Reaction
Seatbelts                   Road Casualties in Great Britain 1969-84
Theoph                      Pharmacokinetics of Theophylline
```

CO2 {datasets} R Documentation

Carbon Dioxide Uptake in Grass Plants

Description

The `CO2` data frame has 84 rows and 5 columns of data from an experiment on the cold tolerance of the grass species *Echinochloa crus-galli*.

Usage

```
CO2
```

Format

An object of class `c("nfnGroupedData", "nfGroupedData", "groupedData", "data.frame")` containing the following columns:

Plant
: an ordered factor with levels `Qn1` < `Qn2` < `Qn3` < ... < `Mc1` giving a unique identifier for each plant.

Type
: a factor with levels `Quebec` `Mississippi` giving the origin of the plant

Treatment

R Packages

- The system allows you to *write new functions* and package those functions in a so called `***R package***' (or `R library').
- The R package may also contain other R objects, for example *datasets or documentation*.
- There is a *lively R user community* and many R packages have been written and made *available on CRAN* for other users.
  - For example, there are packages for *portfolio optimization, drawing maps, exporting objects to html, time series analysis, spatial statistics* and the list goes on and on.

R Packages

- When you download R, already a number (around 30) of packages are downloaded as well.
- To use a function in an R package, that package should be attached to the system.
- When you start R some of the packages are attached to the system by default (10 for my case).
- You can use the function search to see a list of packages that are currently attached to the system, this list is also called the search path.

```
> search()
 [1] ".GlobalEnv"        "tools:rstudio"     "package:stats"     "package:graphics"
 [5] "package:grDevices" "package:utils"     "package:datasets"  "package:methods"
 [9] "Autoloads"         "package:base"
```

R Packages

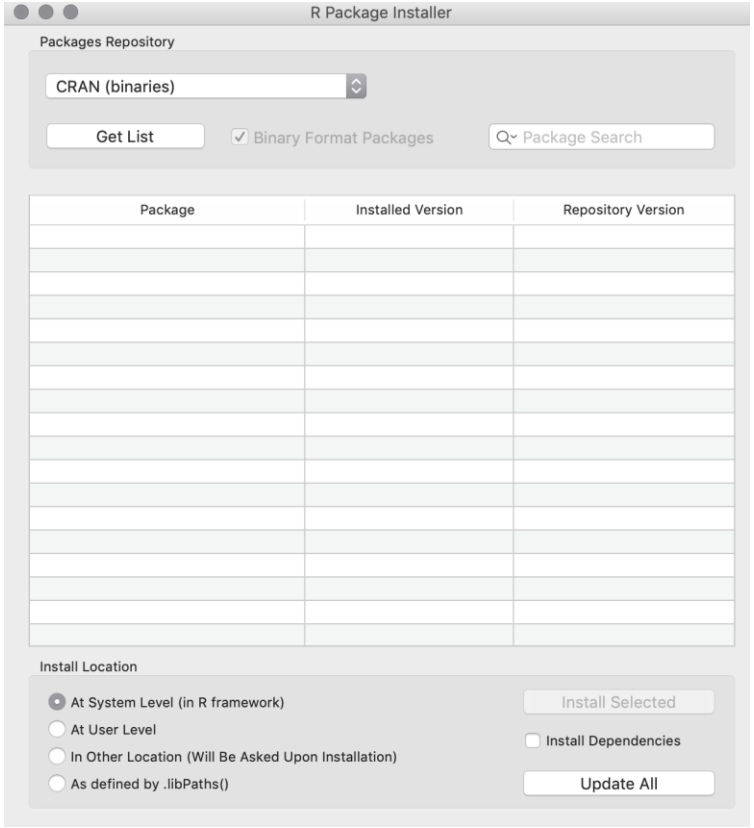
- To attach another package to the system you can use the menu or the library function.
- Via the menu: select *R package Installer*, e.g., from *Packages&Data* menu for *RGui*.
- Via the library function:

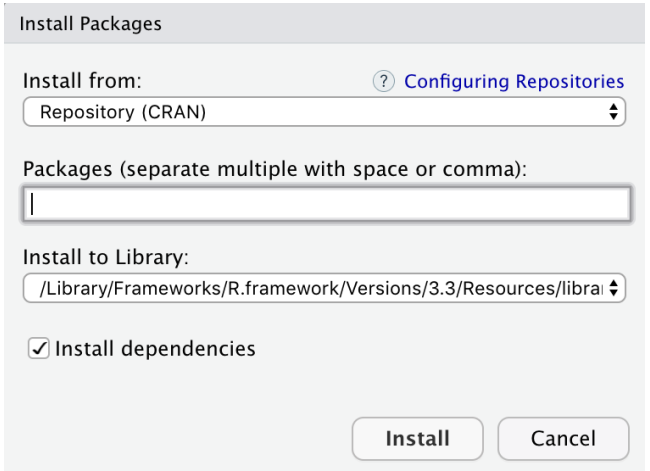
Modern Applied Statistics with S

```
> library(MASS)
> shoes
```

Shoe wear data of Box, Hunter and Hunter

```
$A
[1] 13.2 8.2 10.9 14.3 10.7 6.6
    9.5 10.8 8.8 13.3
$B
[1] 14.0 8.8 11.2 14.2 11.8 6.4
    9.8 11.3 9.3 13.6
```





R Packages

- The function library can also be used to list all the available libraries on your system with a short description.
- Run the function without any arguments

> library()

```
Packages in library '/Library/Frameworks/R.framework/Versions/3.3/Resources/library':

base                    The R Base Package
boot                    Bootstrap Functions (Originally by Angelo Canty for S)
class                   Functions for Classification
cluster                 "Finding Groups in Data": Cluster Analysis Extended
                        Rousseeuw et al.
codetools               Code Analysis Tools for R
compiler                The R Compiler Package
datasets                The R Datasets Package
foreign                 Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat,
                        Weka, dBase, ...
graphics                The R Graphics Package
grDevices               The R Graphics Devices and Support for Colours and Fonts
grid                    The Grid Graphics Package
KernSmooth              Functions for Kernel Smoothing Supporting Wand & Jones
                        (1995)
lattice                 Trellis Graphics for R
MASS                    Support Functions and Datasets for Venables and Ripley's
                        MASS
Matrix                  Sparse and Dense Matrix Classes and Methods
methods                 Formal Methods and Classes
mgcv                    Mixed GAM Computation Vehicle with GCV/AIC/REML Smoothness
                        Estimation
nlme                    Linear and Nonlinear Mixed Effects Models
nnet                    Feed-Forward Neural Networks and Multinomial Log-Linear
                        Models
```

Roadmap

- R Basics
  - Programming Environments
  - R Commands
  - Datasets and Packages
- Data Input and Manipulation
  - Data Types
  - Data Input and Processing
  - Data Manipulation

Roadmap

- R Basics
  - Programming Environments
  - R Commands
  - Datasets and Packages
- Data Input and Manipulation
  - Data Types
  - Data Input and Processing
  - Data Manipulation

Data Types

- **R** has a wide variety of *data types* including *scalars, vectors (numerical, character, logical), matrices, dataframes, and lists*.

Data Types – Vectors

- **R** has a wide variety of *data types* including *scalars, vectors (numerical, character, logical), matrices, dataframes, and lists*.

```
> a
[1]  1.0  2.0  5.3  6.0 -2.0  4.0
```

```
> a <- c(1,2,5.3,6,-2,4) # numeric vector
> b <- c("one","two","three") # character vector
> c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # logical vector
```

```
> # Refer to elements of a vector using subscripts.
> a[c(2,4)] # 2nd and 4th elements of vector
> a[2:4] # 2nd to 4th elements of vector
```

```
> a[c(2,4)]
[1] 2 6
```

```
> a[2:4]
[1] 2.0 5.3 6.0
```

Starts at index 1 instead of 0

Data Types - Matrices

- **R** has a wide variety of *data types* including *scalars, vectors (numerical, character, logical), matrices, dataframes, and lists*.

- *mymatrix <-* ***matrix(****vector,* ***nrow=****r,* ***ncol=****c,* ***byrow=****FALSE,****dimnames=list(****char_vector_rownames, char_vector_colnames****))***

Fill the matrices by columns (default)

provides optional labels for the columns and rows

Data Types - Matrices

```
# generates 5 x 4 numeric matrix
    y<-matrix(1:20, nrow=5,ncol=4)
# another examples
    cells <- c(1,26,24,68)
    rnames <- c("R1", "R2")
    cnames <- c("C1", "C2")
    mymatrix <- matrix(cells, nrow=2, ncol=2,
    byrow=TRUE, dimnames=list(rnames, cnames))
```

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

```
> mymatrix
   C1 C2
R1  1 26
R2 24 68
```

```
#Identify rows, columns or elements using subscripts.
y[,4] # 4th column of matrix
    y[3,] # 3rd row of matrix
    y[2:4,1:3] # rows 2,3,4 of columns 1,2,3
```

```
> y[2:4,1:3]
     [,1] [,2] [,3]
[1,]    2    7   12
[2,]    3    8   13
[3,]    4    9   14
```

Data Types - Arrays

- **R** has a wide variety of *data types* including *scalars, vectors (numerical, character, logical), matrices, dataframes, and lists*.
  - ***Arrays*** are similar to matrices but can have more than two dimensions. See **help(array)** for details.

Multi-way Arrays

Description

Creates or tests for arrays.

Usage

```
array(data = NA, dim = length(data), dimnames = NULL)
as.array(x, ...)
is.array(x)
```

Data Types - Dataframes

- **R** has a wide variety of *data types* including *scalars, vectors (numerical, character, logical), matrices, dataframes, and lists*.

A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.).

```
d <- c(1,2,3,4)
e <- c("red", "white", "red", NA)
f <- c(TRUE,TRUE,TRUE,FALSE)
mydata <- data.frame(d,e,f)
names(mydata) <- c("ID","Color","Passed") #variable names
```

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

Data Types - Dataframes

A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.).

```
d <- c(1,2,3,4)
e <- c("red", "white", "red", NA)
f <- c(TRUE,TRUE,TRUE,FALSE)
mydata <- data.frame(d,e,f)
names(mydata) <- c("ID","Color","Passed") #variable names
```

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

There are a variety of ways to identify the elements of a dataframe.

```
mydata[1:2] # columns 1,2 of dataframe
mydata[c("ID",”Passed")] # columns ID and Passed from dataframe
mydata$ID # variables ID in the dataframe
```

Data Types - Lists

- **R** has a wide variety of *data types* including *scalars, vectors (numerical, character, logical), matrices, dataframes, and lists*.
  - ***A list*** *is an ordered collection of objects (components)*. It allows you to gather a variety of (even unrelated) objects under one name. **Examples**:

```
# example of a list with 4 components -
# a string, a numeric vector, a matrix, and a scaler
    w <- list(name="Fred", mynumbers=a,
    mymatrix=y, age=5.3)

    # example of a list containing two lists
    v <- c(list1,list2)
```

Data Types - Lists

- An ordered collection of objects (components). A list allows you to gather a variety of (possibly unrelated) objects under one name. **Examples**:

```
# example of a list with 4 components -
# a string, a numeric vector, a matrix, and a scaler
    w <- list(name="Fred", mynumbers=a,
    mymatrix=y, age=5.3)

    # example of a list containing two lists
    v <- c(list1,list2)
```

Identify elements of a list using the [[]] convention.

```
w[[2]] # 2nd component of the list
```

```
> w[[2]]
[1]  1.0  2.0  5.3  6.0 -2.0  4.0
```

Roadmap

- R Basics
  - Programming Environments
  - R Commands
  - Datasets and Packages
- Data Input and Manipulation
  - Data Types
  - Data Input and Processing
  - Data Manipulation

Import and Export Data - Text File

- **Import Data**

  # first row contains variable names, comma is separator

  # assign the variable *id* to row names

  # note the / instead of \ on MS windows systems

```
mydata <- read.table("c:/mydata.csv",
header=TRUE, sep=",", row.names="id")
```

- **Export Data**

  #To A Tab Delimited Text File

```
write.table(mydata, "c:/mydata.txt", sep="\t")
```

Viewing Data

- **There are a number of functions for listing the contents of an object or dataset.**

```
# list objects in the working environment
  ls()
# list the variables in mydata
  names(mydata)
# list the structure of mydata
  str(mydata)
# dimensions of an object
  dim(object)
```

```
# class of an object (numeric, matrix, dataframe, etc)
  class(object)
# print mydata
  mydata
# print first 10 rows of mydata
  head(mydata, n=10)
# print last 5 rows of mydata
  tail(mydata, n=5)
```

Import Data – Keyboard Input

```
# create a dataframe from scratch
age <- c(25, 30, 56)
gender <- c("male", "female", "male")
weight <- c(160, 110, 220)
mydata <- data.frame(age,gender,weight)
```

```
> mydata
  age gender weight
1  25   male    160
2  30 female    110
3  56   male    220
```

Missing Data

- In **R**, missing values are represented by the symbol ***NA*** (*not available*) .
- Impossible values (e.g., dividing by zero) are represented by the symbol ***NaN*** (*not a number*).
- **Testing for Missing Values**

```
is.na(x) # returns TRUE of x is missing
y <- c(1,2,3,NA)
is.na(y) # returns a vector (F F F T)
```

Missing Data

- **Excluding Missing Values from Analyses**
  - Arithmetic functions on missing values yield missing values.

```
x <- c(1,2,NA,3)
mean(x)        # returns NA
mean(x, na.rm=TRUE) # returns 2
```

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

- The function ***complete.cases()*** returns a logical vector indicating which cases are complete.

```
# list rows of data that have missing values
mydata[!complete.cases(mydata),]
```

```
  ID Color Passed
4  4  <NA>  FALSE
```

- The function **na.omit()** returns the object with listwise deletion of missing values.

```
# create new dataset without missing data
newdata <- na.omit(mydata)
```

```
> newdata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
```

Roadmap

- R Basics
  - Programming Environments
  - R Commands
  - Datasets and Packages
- Data Input and Manipulation
  - Data Types
  - Data Input and Processing
  - Data Manipulation

Creating New Variables

- Use the *assignment operator* "**<-**" or "=" to create new variables.

```
# Three examples for doing the same computations
mydata$sum <- mydata$x1 + mydata$x2
mydata$mean <- (mydata$x1 + mydata$x2)/2
```

```
attach(mydata)
mydata$sum <- x1 + x2
mydata$mean <- (x1 + x2)/2
detach(mydata)
```

```
mydata <- transform(mydata, sum = x1 + x2, mean = (x1 + x2)/2 )
```

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
***	multiplication
/	division
*^ or ** *	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
*x	y*	x OR y
x & y	x AND y
isTRUE(x)	test if x is TRUE

Control Structures

- **R** has the standard control structures you would expect.
- **expr** can be multiple (compound) statements by enclosing them in braces {}.
- It is more *efficient* to use built-in functions rather than control structures whenever possible.

if-else

if (*cond*) *expr*
if (*cond*) *expr1* else *expr2*

for

for (*var* in *seq*) *expr*

while

while (*cond*) *expr*

switch

switch(*expr*, ...)

ifelse

ifelse(*test*,*yes*,*no*)

Control Structures - Example

```
# transpose of a matrix
# a poor alternative to built-in t() function
    mytrans <- function(x) {
     if (!is.matrix(x)) {
       warning("argument is not a matrix:
    returning NA")
       return(NA_real_)}
     y <- matrix(1, nrow=ncol(x),
    ncol=nrow(x))
     for (i in 1:nrow(x)) {
       for (j in 1:ncol(x)) {
         y[j,i] <- x[i,j]
       }}
    return(y)}
```

```
> z <- matrix(1:10, nrow=5, ncol=2)
> z
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
> tz <- mytrans(z)
> tz
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
```

R Built-in Functions

- Almost everything in **R** is done through functions.
- Many of the functions can be applied to vectors and matrices as well.

Numeric Functions

Function	Description
abs(x)	absolute value
sqrt(x)	square root
ceiling(x)	ceiling(3.475) is 4
floor(x)	floor(3.475) is 3
trunc(x)	trunc(5.99) is 5
round(x, digits=n)	round(3.475, digits=2) is 3.48
signif(x, digits=n)	signif(3.475, digits=2) is 3.5
cos(x), sin(x), tan(x)	also acos(x), cosh(x), acosh(x), etc.
log(x)	natural logarithm
log10(x)	common logarithm
exp(x)	e^x

Character Functions

Function	Description
substr(x, start=n1, stop=n2)	Extract or replace substrings in a character vector. x <- "abcdef" substr(x, 2, 4) is "bcd" substr(x, 2, 4) <- "22222" is "a222ef"
grep(pattern, x , ignore.case=FALSE, fixed=FALSE)	Search for pattern in x. If fixed =FALSE then pattern is a regular expression. If fixed=TRUE then pattern is a text string. Returns matching indices. grep("A", c("b","A","c"), fixed=TRUE) returns 2
sub(pattern, replacement, x, ignore.case =FALSE, fixed=FALSE)	Find pattern in x and replace with replacement text. If fixed=FALSE then pattern is a regular expression. If fixed = T then pattern is a text string. sub("\\s",".","Hello There") returns "Hello.There"
strsplit(x, split)	Split the elements of character vector x at split. strsplit("abc", "") returns 3 element vector "a","b","c"
paste(..., sep="")	Concatenate strings after using sep string to seperate them. paste("x",1:3,sep="") returns c("x1","x2" "x3") paste("x",1:3,sep="M") returns c("xM1","xM2" "xM3") paste("Today is", date())
toupper(x)	Uppercase
tolower(x)	Lowercase

Probability Functions

Function	Description
dnorm(x)	normal density function (by default m=0 sd=1) # plot standard normal curve x <- pretty(c(-3,3), 30) y <- dnorm(x) plot(x, y, type='l', xlab="Normal Deviate", ylab="Density", yaxs="i")
pnorm(q)	cumulative normal probability for q (area under the normal curve to the right of q) pnorm(1.96) is 0.975
qnorm(p)	normal quantile. value at the p percentile of normal distribution qnorm(0.9) is 1.28 # 90th percentile
rnorm(n, m=0,sd=1)	n random normal deviates with mean m and standard deviation sd. #50 random normal variates with mean=50, sd=10 x <- rnorm(50, m=50, sd=10)

Probability Functions (CONT.)

Function	Description
dbinom(x, size, prob) *pbinom(q, size, prob)* *qbinom(p, size, prob)* *rbinom(n, size, prob)*	binomial distribution where size is the sample size and prob is the probability of a heads (pi) # prob of 0 to 5 heads of fair coin out of 10 flips dbinom(0:5, 10, .5) # prob of 5 or less heads of fair coin out of 10 flips pbinom(5, 10, .5)
dpois(x, lamda) *ppois(q, lamda)* *qpois(p, lamda)* *rpois(n, lamda)*	poisson distribution with m=std=lamda #probability of 0,1, or 2 events with lamda=4 dpois(0:2, 4) # probability of at least 3 events with lamda=4 1- ppois(2,4)
dunif(x, min=0, max=1) *punif(q, min=0, max=1)* *qunif(p, min=0, max=1)* *runif(n, min=0, max=1)*	uniform distribution, follows the same pattern as the normal distribution above. #10 uniform random variates x <- runif(10)

Statistical Functions

Function	Description
mean(x, trim=0, na.rm=FALSE)	mean of object x # trimmed mean, removing any missing values and # 5 percent of highest and lowest scores mx <- mean(x,trim=.05,na.rm=TRUE)
sd(x)	standard deviation of object(x). also look at var(x) for variance and mad(x) for median absolute deviation.
median(x)	median
quantile(x, probs)	quantiles where x is the numeric vector whose quantiles are desired and probs is a numeric vector with probabilities in [0,1]. # 30th and 84th percentiles of x y <- quantile(x, c(.3,.84))

Statistical Functions (CONT)

Function	Description
range(x)	range
sum(x)	sum
diff(x, lag=1)	lagged differences, with lag indicating which lag to use
min(x)	minimum
max(x)	maximum
scale(x, center=TRUE, scale=TRUE)	column center or standardize a matrix.

A slide to take away

- What do the *R programming environments* and the *R commands* look like?
- How to attach the external *datasets and packages*?
- What are the major *data types* for R programming?
- How to *read the data* for the analytics with R?
- How to *manipulate data* with operations, control structures, and functions?