R-Language

PREPARED FOR

Engineering Students

All Engineering College

(R-Language)
PREPARED BY: ***MS. SHWETA TIWARI***
Published On: March 12, 2022

Today's discussion…

- R is an open source programming language and software environment for statistical computing and graphics.

- The R language is widely used among statisticians and data miners for developing statistical software and data analytics tools

History of R

- Modelled after S & S-plus, developed at AT&T labs in late 1980s.
- R project was started by Robert Gentleman and Ross Ihaka Department of Statistics, University of Auckland (1995).
- Currently maintained by R core development team – an international team of volunteer developers (since 1997).

R resources

- http://www.r-project.org/
- http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf

Download R and RStudio

- Download R :
  http://cran.r-project.org/bin/

- Download RStudio :
  http://www.rstudio.com/ide/download/desktop

Installation

Installing R on windows PC :

- Use internet browser to point to : http://mirror.aarnet.edu.au/pub/CRAN
- Under the heading Precompiled Binary Distributions, choose the link Windows.
- Next heading is R for Windows; choose the link base.
- Click on download option(R 3.4.1 for windows).
- Save this to the folder C:\R on your PC.
- When downloading is complete, close or minimize the Internet browser.
- Double click on R 3.4.1-win32.exe in C:\R to install.

Installing R on Linux:

- sudo apt-get install r-base-core

Installation

Installing RStudio:

- Go to www.rstudio.com and click on the "Download RStudio" button.
- Click on "Download RStudio Desktop.“
- Click on the version recommended for your system, or the latest Windows version, and save the executable file. Run the .exe file and follow the installation instructions.

Version

- Get R version

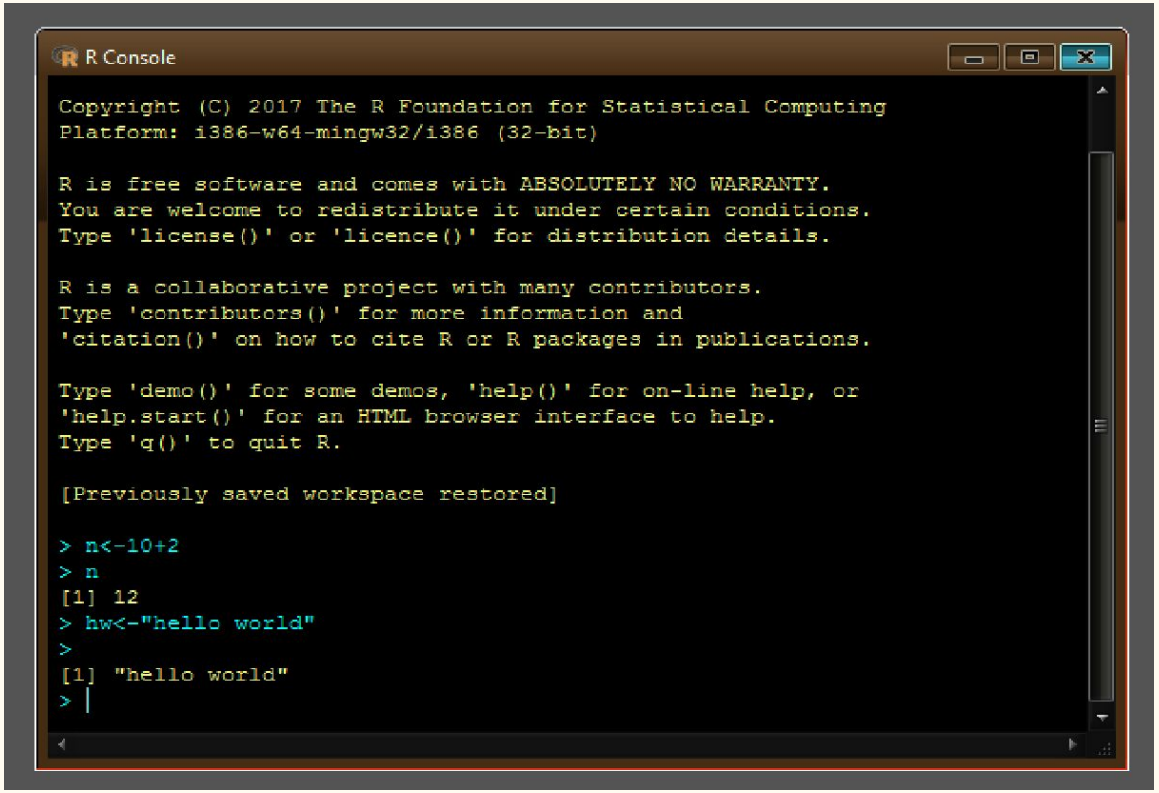
  R.Version()

- Get RStudio version

  RStudio: Toolbar at top > Help > About RStudio

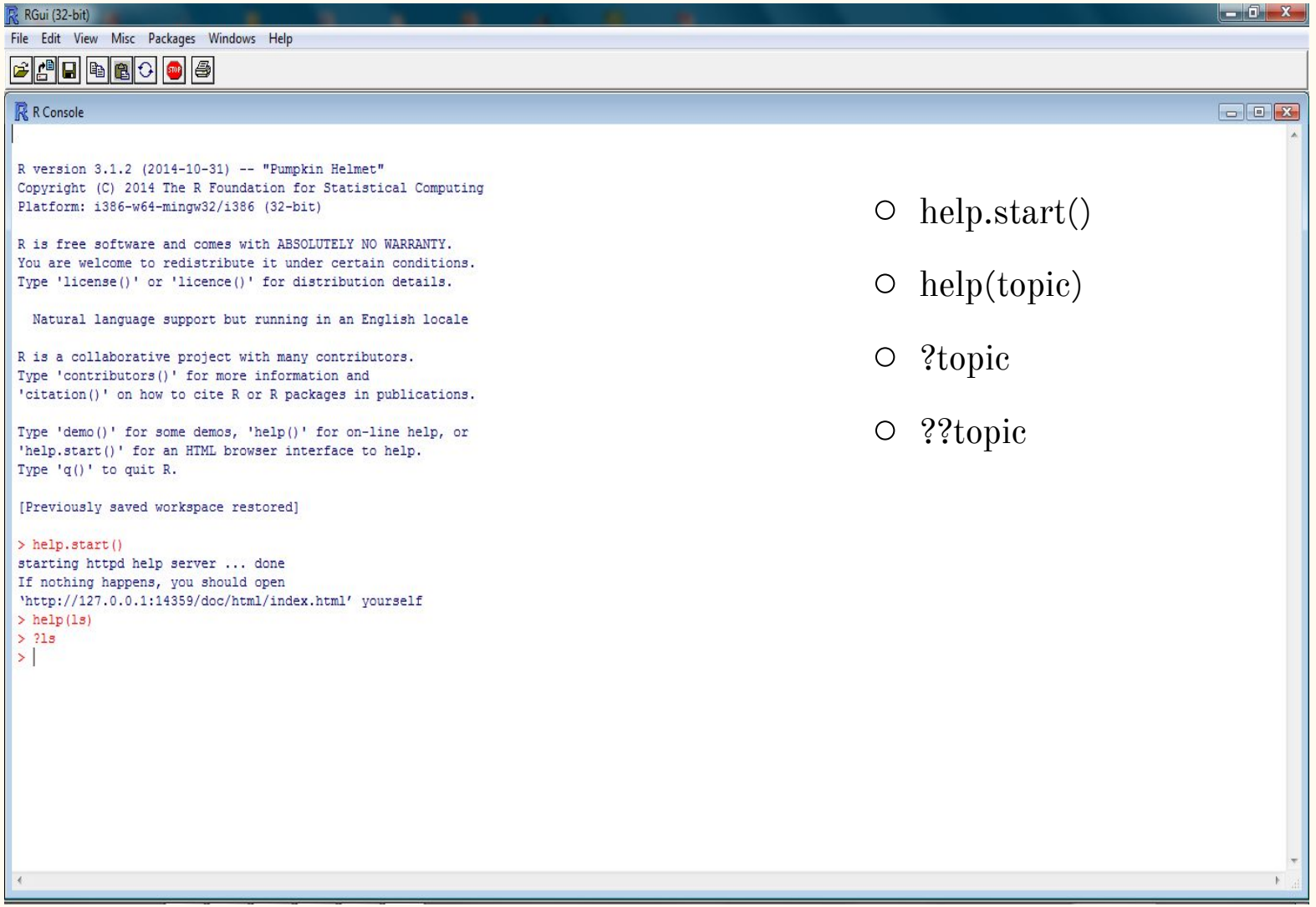
A test run with R in Windows

- Double click the R icon on the Desktop and the R Console will open.
- Wait while the program loads. You observe something like this.



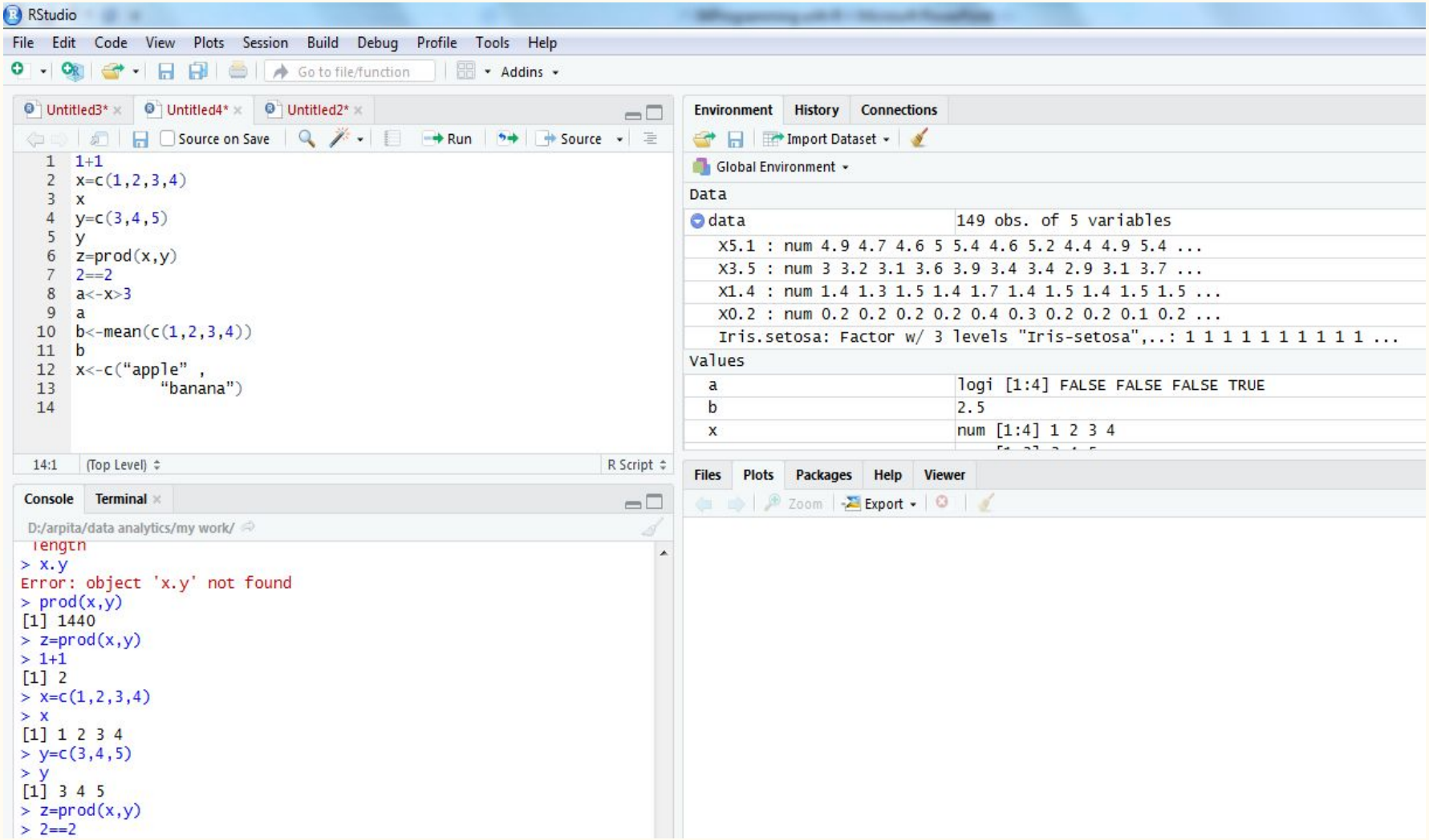
- You can type your own program at the prompt line >.

Getting help from R console



- help.start()
- help(topic)
- ?topic
- ??topic

R command in integrated environment



How to use R for simple maths

- > 3+5
- > 12 + 3 / 4 – 5 + 3*8
- > (12 + 3 / 4 – 5) + 3*8
- > pi * 2^3 – sqrt(4)
- >factorial(4)
- >log(2,10)
- >log(2, base=10)
- >log10(2)
- >log(2)

Note

- **R ignores spaces**

How to store results of calculations for future use

- > x = 3+5
- > x
- > y = 12 + 3 / 4 – 5 + 3*8
- > y
- > z = (12 + 3 / 4 – 5) + 3*8
- > z
- > A <- 6 + 8 ## no space should be between < & -
- > a ## Note: R is case sensitive
- >A

Identifiers naming

- Don't use underscores (_) or hyphens (-) in identifiers.
- The preferred form for variable names is all lower case letters and words separated with dots (variable.name) but variableName is also accepted.
- Examples:

avg.clicks	GOOD
avgClicks	OK
avg_Clicks	BAD

- Function names have initial capital letters and no dots (e.g., FunctionName).

Using C command

- > data1 = c(3, 6, 9, 12, 78, 34, 5, 7, 7) ## numerical data
- > data1.text = c('Mon', 'Tue', "Wed") ## Text data
- ## Single or double quote both ok
- ##copy/paste into R console may not work
- > data1.text = c(data1.text, 'Thu', 'Fri')

Scan command for making data

- > data3 = scan() ## data separated by Space / Press
  ## Press Enter key twice to exit
- 1: 4 5 7 8
- 5: 2 9 4
- 8: 3
- 9:

Read 8 items

- > data3
- [1] 4 5 7 8 2 9 4 3

Scan command for making data

- **> d3 = scan(what = 'character')**
- 1: mon
- 2: tue
- 3: wed thu
- 5:
- **> d3**
- [1] "mon" "tue" "wed" "thu"
- **> d3[2]**
- [1] "tue"
- **> d3[2]='mon'**
- **> d3**
- [1] "mon" "mon" "wed" "thu"

- > d3[6]='sat'
- > d3
- [1] "mon" "mon" "wed" "thu" NA "sat"
- > d3[2]='tue'
- > d3[5] = 'fri'
- > d3
- [1] "mon" "tue" "wed" "thu" "fri" "sat"

Concept of working directory

- >getwd()
- [1] "C:\Users\DShweta\R\Database"
- > setwd('D:\Data Analytics\Project\Database)
- > dir() ## working directory listing
- >ls() ## Workspace listing of objects
- >rm('object') ## Remove an element "object", if exist
- > rm(list = ls()) ## Cleaning

Reading data from a data file

- > setwd("D:/shweta/data analytics/my work") #Set the working directory to file location
- > getwd()
- [1] "D:/shweta/data analytics/my work"
- > dir()
- [1] "Arv.txt" "DiningAtSFO" "LatentView-DPL" "TC-10-Rec.csv" "TC.csv"
- rm(list=ls(all=TRUE)) # Refresh session
- > data=read.csv('iris.csv', header = T, sep=",")
- (data = read.table('iris.csv', header = T, sep = ','))
- > ls()
- [1] "data"
- > str(data)
- 'data.frame': 149 obs. of 5 variables:
- $ X5.1 : num 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 5.4 ...
- $ X3.5 : num 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 ...
- $ X1.4 : num 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 ...
- $ X0.2 : num 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 0.2 ...
- $ Iris.setosa: Factor w/ 3 levels "Iris-setosa",..: 1 1 1 1 1 1 1 1 1 1 ...

Accessing elements from a file

- > data$X5.1
- [1] 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7
- > data$X5.1[7]=5.2
- > data$X5.1
- [1] 4.9 4.7 4.6 5.0 5.4 4.6 5.2 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7
  #Note: This change has happened in workspace only not in the file.
- How to make it permanent?
- write.csv / write.table
- >write.table(data, file =‘iris_mod.csv', row.names = FALSE, sep = ',')
- If row.names is TRUE, R adds one ID column in the beginning of file.
- So its suggested to use row.names = FALSE option
- >write.csv(data, file ==‘iris_mod.csv', row.names = TRUE) ## to test

Different data items in R

- **Vector**
- **Matrix**
- **Data Frame**
- **List**

Vectors in R

- >x=c(1,2,3,4,56)
- >x
- > x[2]
- > x = c(3, 4, NA, 5)
- >mean(x)
- [1] NA
- >mean(x, rm.NA=T)
- [1] 4
- > x = c(3, 4, NULL, 5)
- >mean(x)
- [1] 4

More on Vectors in R

- **>y = c(x,c(-1,5),x)**
- **>length(x)**
- **>length(y)**
- **There are useful methods to create long vectors whose elements are in arithmetic progression:**
- **> x=1:20**
- **> x**

- **If the common difference is not 1 or -1 then we can use the seq function**
- **> y=seq(2,5,0.3)**
- **> y**
- [1] 2.0 2.3 2.6 2.9 3.2 3.5 3.8 4.1 4.4 4.7 5.0
- **> length(y)**
- [1] 11

More on Vectors in R

- > x=1:5
- > mean(x)
- [1] 3
- > x
- [1] 1 2 3 4 5
- > x^2
- [1] 1 4 9 16 25
- > x+1
- [1] 2 3 4 5 6
- > 2*x
- [1] 2 4 6 8 10
- > exp(sqrt(x))
- [1] 2.718282 4.113250 5.652234 7.389056 9.356469

- It is very easy to add/subtract/multiply/divide two vectors entry by entry.
- > y=c(0,3,4,0)
- > x+y
- [1] 1 5 7 4 5
- > y=c(0,3,4,0,9)
- > x+y
- [1] 1 5 7 4 14
- Warning message:
- In x + y : longer object length is not a multiple of shorter object length
- > x=1:6
- > y=c(9,8)
- > x+y
- [1] 10 10 12 12 14 14

Matrices in R

- Same data type/mode – number , character, logical
- a.matrix <- matrix(vector, nrow = r, ncol = c, byrow = FALSE, dimnames = list(char-vector-rownames, char-vector-col-names))

  ## dimnames is optional argument, provides labels for rows & columns.
- > y <- matrix(1:20, nrow = 4, ncol = 5)
- >A = matrix(c(1,2,3,4),nrow=2,byrow=T)
- >A
- >A = matrix(c(1,2,3,4),ncol=2)
- >B = matrix(2:7,nrow=2)
- >C = matrix(5:2,ncol=2)
- >mr <- matrix(1:20, nrow = 5, ncol = 4, byrow = T)
- >mc <- matrix(1:20, nrow = 5, ncol = 4)
- >mr
- >mc

More on matrices in R

- >dim(B) #Dimension
- >nrow(B)
- >ncol(B)
- >A+C
- >A-C
- >A%*%C #Matrix multiplication. Where will be the result?
- >A*C #Entry-wise multiplication
- >t(A) #Transpose
- >A[1,2]
- >A[1,]
- >B[1,c(2,3)]
- >B[,-1]

Lists in R

- Vectors and matrices in R are two ways to work with a collection of objects.
- Lists provide a third method. Unlike a vector or a matrix a list can hold different kinds of objects.
- One entry in a list may be a number, while the next is a matrix, while a third is a character string (like "Hello R!").
- Statistical functions of R usually return the result in the form of lists. So we must know how to unpack a list using the $ symbol.

Examples of lists in R

- >x = list(name="Arun Patel", nationality="Indian", height=5.5, marks=c(95,45,80))

- >names(x)
- >x$name

- >x$hei #abbreviations are OK
- >x$marks
- >x$m[2]

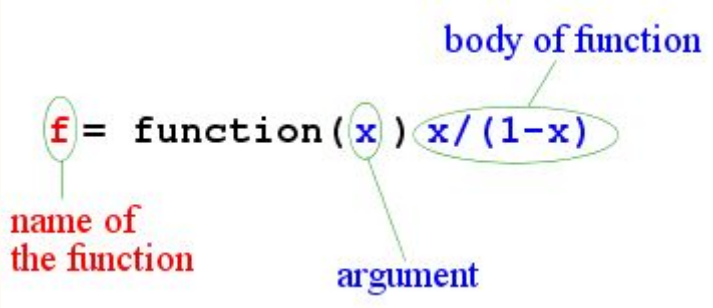
Data frame in R

- 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)
- >myframe <- data.frame(d,e,f)
- >names(myframe) <- c("ID","Color","Passed") # Variable names
- >myframe
- >myframe[1:3,] # Rows 1 , 2, 3 of data frame
- >myframe[,1:2] # Col 1, 2 of data frame
- >myframe[c("ID","Color")] #Columns ID and color from data frame
- >myframe$ID # Variable ID in the data frame

Factors in R

- In R we can make a variable is nominal by making it a factor.
- The factor stores the nominal values as a vector of integers in the range [1... k] (where k is the number of unique values in the nominal variable).
- An internal vector of character strings (the original values) mapped to these integers.
- # Example: variable gender with 20 "male" entries and
  # 30 "female" entries
  >gender <- c(rep("male",20), rep("female", 30))
  >gender <- factor(gender)
  # Stores gender as 20 1's and 30 2's
- # 1=male, 2=female internally (alphabetically)
  # R now treats gender as a nominal variable
  >summary(gender)

Functions in R



- >g = function(x,y) (x+2*y)/3
- >g(1,2)
- >g(2,1)

Useful Functions in R

```
length(object) # number of elements or components
str(object)    # structure of an object
class(object)  # class or type of an object
names(object)  # names
c(object,object,...)  # combine objects into a vector
cbind(object, object, ...) # combine objects as columns
rbind(object, object, ...) # combine objects as rows
ls()       # list current objects
rm(object) # delete an object
newobject <- edit(object) # edit copy and save a newobject
fix(object)               # edit in place
```

Importing Data

Importing data into **R** is fairly simple.

For Stata and Systat, use the **foreign** package.

For SPSS and SAS I would recommend the **Hmisc** package for ease and functionality.

See the **Quick-R** section on **packages**, for information on obtaining and installing the these packages.

Example of importing data are provided below.

From A Comma Delimited Text File

```
# first row contains variable names, comma is separator
# assign the variable id to row names
# note the / instead of \ on mswindows systems

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

From Excel

The best way to read an Excel file is to export it to a comma delimited file and import it using the method above.

On windows systems you can use the **RODBC** package to access Excel files. The first row should contain variable/column names.

```
# first row contains variable names
# we will read in workSheet mysheet
library(RODBC)
channel <- odbcConnectExcel("c:/myexel.xls")
mydata <- sqlFetch(channel, "mysheet")
odbcClose(channel)
```

From SAS

- # save SAS dataset in trasport format
  libname out xport 'c:/mydata.xpt';
  data out.mydata;
  set sasuser.mydata;
  run;
- library(foreign)

#bsl=read.xport(“mydata.xpt")

Keyboard Input

Usually you will obtain a dataframe by importing it from **SAS**, **SPSS**, **Excel**, **Stata**, a database, or an ASCII file. To create it interactively, you can do something like the following.

```
# 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)
```

Keyboard Input

You can also use **R**'s built in spreadsheet to enter the data interactively, as in the following example.

```
# enter data using editor
  mydata <- data.frame(age=numeric(0),
  gender=character(0), weight=numeric(0))
  mydata <- edit(mydata)
  # note that without the assignment in the line above,
  # the edits are not saved!
```

Exporting Data

There are numerous methods for exporting **R** objects into other formats . For SPSS, SAS and Stata. you will need to load the **foreign** packages. For Excel, you will need the **xlsReadWrite** package.

Exporting Data

To A Tab Delimited Text File

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

To an Excel Spreadsheet

```
library(xlsReadWrite)
   write.xls(mydata, "c:/mydata.xls")
```

To SAS

```
library(foreign)
   write.foreign(mydata, "c:/mydata.txt",
   "c:/mydata.sas",   package="SAS")
```

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)
# list levels of factor v1 in mydata
    levels(mydata$v1)
# dimensions of an object
    dim(object)
```

Viewing Data

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

```
# 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)
```

Variable Labels

R's ability to handle variable labels is somewhat unsatisfying.

If you use the **Hmisc** package, you can take advantage of some labeling features.

```
library(Hmisc)
label(mydata$myvar) <- "Variable label for variable myvar"
describe(mydata)
```

Variable Labels

Unfortunately the label is only in effect for functions provided by the **Hmisc** package, such as **describe()**. Your other option is to use the variable label as the variable name and then refer to the variable by position index.

```
names(mydata)[3] <- "This is the label for variable 3"
mydata[3] # list the variable
```

Value Labels

To understand value labels in **R**, you need to understand the data structure factor.

You can use the factor function to create your own value lables.

```
# variable v1 is coded 1, 2 or 3
# we want to attach value labels 1=red, 2=blue,3=green
    mydata$v1 <- factor(mydata$v1,
    levels = c(1,2,3),
    labels = c("red", "blue", "green"))
# variable y is coded 1, 3 or 5
# we want to attach value labels 1=Low, 3=Medium, 5=High
```

Value Labels

```
mydata$v1 <- ordered(mydata$y,
    levels = c(1,3, 5),
    labels = c("Low", "Medium", "High"))
```

Use the **factor()** function for **nominal data** and the **ordered()** function for **ordinal data**. **R** statistical and graphic functions will then treat the data appropriately.

Note: factor and ordered are used the same way, with the same arguments. The former creates factors and the later creates ordered factors.

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). Unlike SAS, **R** uses the same symbol for character and numeric data.

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

Recoding Values to Missing

```
# recode 99 to missing for variable v1
# select rows where v1 is 99 and recode column v1
mydata[mydata$v1==99,"v1"] <- NA
```

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
```

Missing Data

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),]
```

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

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

Missing Data

Advanced Handling of Missing Data

Most modeling functions in **R** offer options for dealing with missing values. You can go beyond pairwise of listwise deletion of missing values through methods such as multiple imputation. Good implementations that can be accessed through **R** include **Amelia II**, **Mice**, and **mitools**.

Date Values

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
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]
```

Sys.Date( ) returns today's date.

Date() returns the current date and time.

Date Values

The following symbols can be used with the format( ) function to print dates.

Symbol	Meaning	Example
%d	day as a number (0-31)	01-31
%a	abbreviated weekday	Mon
%A	unabbreviated weekday	Monday
%m	month (00-12)	00-12
%b	abbreviated month	Jan
%B	unabbreviated month	January
%y	2-digit year	07
%Y	4-digit year	2007

Date Values

```
# print today's date
today <- Sys.Date()
format(today, format="%B %d %Y")
      "June 20 2007"
```

Any Question?