

Basics

Features of R

- R is an interpreted language
- R is case sensitive
- R has powerful graphics with package ggplot2
- R can perform complex statistical calculations

Data Types in R

- atomic classes
- vectors, lists
- factors
- missing values
- matrices
- arrays
- data frames

Atomic Classes

- ❑ Character
 - ❑ "a", "Good", "Bad"
- ❑ numeric (real numbers)
 - ❑ 12.3, 0, -0.92, -1200.76
- ❑ Integer
 - ❑ 12, -13, 90, 0
- ❑ Complex
 - ❑ $2 + 4i$, $5 - 5i$
- ❑ logical
 - ❑ TRUE, FALSE

Vector

- A vector can only contain objects of the same atomic class
- It can be created with the help of `c()` function

```
> d <- c(12,23,15,19,90)
> d
[1] 12 23 15 19 90
```

```
> s <- c("ABC","L","FRE","LLL")
> s
[1] "ABC" "L"   "FRE" "LLL"
```

List

- List can contain objects of different types
- Many times we observe that the outputs of many algorithmic functions are expressed as list
- List can be created with the help of the function `list()`

```
> suchi <- list(1,"Amit",TRUE,3,FALSE,"Swapnil","Neha")
> suchi
[[1]]
[1] 1

[[2]]
[1] "Amit"

[[3]]
[1] TRUE

[[4]]
[1] 3

[[5]]
[1] FALSE

[[6]]
[1] "Swapnil"

[[7]]
[1] "Neha"
```

Accessing the contents in List

- Accessing members in the list is little bit different

```
> suchi <- list(1,"Amit",TRUE,3,FALSE,"Swapnil","Neha")
> suchi[1]
[[1]]
[1] 1

> suchi[5]
[[1]]
[1] FALSE
```

```
> vis <- list("DR",c(12,13,34),TRUE,13)
> vis[2]
[[1]]
[1] 12 13 34

> vis[1]
[[1]]
[1] "DR"

> vis[[2]][1]
[1] 12
```

Factor

- Factors are used to represent categorical data.
- Factors can be unordered or ordered.
- One can think of a factor as an integer vector where each integer has a *label*.
- *E.g.*
 - *Categorical variables like*
 - *Yes / No*
 - *Sold / Not Sold / On hold*

Missing Values

- Missing values are presented as NA or NaN.
- `is.na()` is used to test objects if they are NA
- `is.nan()` is used to test for Not a Number
- Also there are functions indicating finite / infinite values
- A NaN value is also NA but the NA is not always NaN.
- Best example of NaN is result of zero by zero

```
> f <- NA
> is.na(f)
[1] TRUE
> d <- c(NA,13,17,NA,13,NA)
> is.na(d)
[1] TRUE FALSE FALSE TRUE FALSE TRUE
```

```
> f <- 0
> p <- 0
> g <- f/p
> is.nan(g)
[1] TRUE
> is.na(g)
[1] TRUE
```

```
> r <- 23
> p <- 0
> e <- r/p
> is.finite(e)
[1] FALSE
> is.infinite(e)
[1] TRUE
```

Matrix

- Matrix can hold the data in matrix form
- The matrix can be created with the help of R function
- Commonly used syntax arguments are
 - `matrix(data , nrow , ncol)`

```
> mat <- matrix(10,3,2)
> mat
```

	[,1]	[,2]
[1,]	10	10
[2,]	10	10
[3,]	10	10

```
> mat2 <- matrix(c(2,3,1,4,2,4),3,2)
> mat2
```

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

rbind() and cbind()

- Matrices can also be created by *using* functions cbind() and rbind().
- rbind() binds the rows and cbind() binds the columns

```
> a <- 1 : 5
> a
[1] 1 2 3 4 5
> b <- 46 : 50
> b
[1] 46 47 48 49 50
> cb <- cbind(a,b)
> cb
      a  b
[1,] 1 46
[2,] 2 47
[3,] 3 48
[4,] 4 49
[5,] 5 50
```

```
> rb <- rbind(a,b)
> rb
      [,1] [,2] [,3] [,4] [,5]
a        1    2    3    4    5
b       46   47   48   49   50
```

When values specified \neq ncols / nrows

- While using any binding function in case if no. of rows / columns are not equal to no. of values specified then the values get repeated in that order with a warning.

```
> a <- 1:3
> b <- 20:30
> rbind(a,b)
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
a    1    2    3    1    2    3    1    2    3    1    2
b   20   21   22   23   24   25   26   27   28   29   30
Warning message:
In rbind(a, b) :
  number of columns of result is not a multiple of vector length (arg 1)
> cbind(a,b)
      a  b
[1,] 1 20
[2,] 2 21
[3,] 3 22
[4,] 1 23
[5,] 2 24
[6,] 3 25
[7,] 1 26
[8,] 2 27
[9,] 3 28
[10,] 1 29
[11,] 2 30
Warning message:
In cbind(a, b) :
  number of rows of result is not a multiple of vector length (arg 1)
```

Array

- Array can be treated as a vector with multiple dimensions
- Array can be created with the function `array` specifying the dimension using `dim` as argument
- By default the values are NA

Syntax : `array(dim=c(r,c,...))`

```
> k <- array(dim=c(8))  
> k  
[1] NA NA NA NA NA NA NA NA
```

```
> d <- array(dim=c(3,4))  
> d  
      [,1] [,2] [,3] [,4]  
[1,]  NA  NA  NA  NA  
[2,]  NA  NA  NA  NA  
[3,]  NA  NA  NA  NA
```

Data Frames

- Data frame are used hold tabular data
- We can combine several vectors of same length into a data frame
- Data frames provide a simpler way for handling the data than lists
- Data frame gets usually created by calling functions like `read.csv()`, `read.xlsx` etc.

```
> a <- c(12,23,14,15)
> b <- c("X","Y","Z","W")
> df <- data.frame(a,b)
> df
  a b
1 12 X
2 23 Y
3 14 Z
4 15 W
```

```
> bollywood <- read.csv("D:/Data Science Training/Datasets/Bollywood_2015.csv")
> bollywood
```

	Movie.Name	BO_Collection	Budget	Box_Office_Verdict
1	Pyaar Ka PUNCHnama 2	53.25	25.0	Hit
2	Shandaar	38.28	68.0	Flop
3	Singh is Bliing	74.87	92.0	Flop
4	Jazbaa	24.30	30.0	Flop
5	Talvar	24.00	22.0	Plus
6	Kis Kisko Pyaar Karoon	44.80	20.0	Hit
7	Calendar Girls	7.00	12.0	Flop
8	Katti Batti	22.96	36.0	Flop

colnames

- Colnames are set of row or column names of a matrix-like object.

	Order.ID	Order.Date	Place.of.Shipment	Payment.Terms	Item.ID	Qty
1	32 90 001	31-Dec-10	Pune	Cheque	121 021	7
2	32 90 001	31-Dec-10	Pune	Cheque	121 003	49
3	32 90 001	31-Dec-10	Pune	Cheque	121 023	1
4	32 90 001	31-Dec-10	Pune	Cheque	121 018	9
5	32 90 001	31-Dec-10	Pune	Cheque	121 015	29
6	32 90 001	31-Dec-10	Pune	Cheque	121 014	44

```
> colnames(ords)
[1] "order.ID"      "order.Date"      "Place.of.Shipment" "Payment.Terms"    "Item.ID"
[6] "Qty"
```

names

- For knowing the names of the elements of any object `names()` function can be used

```
> names(df)
[1] "a" "b"
```

```
> names(bollywood)
[1] "Movie.Name"      "BO_Collection"    "Budget"
[4] "Box_Office_Verdict"
```

```
> names(ords)
[1] "order.ID"          "order.Date"        "Place.of.Shipment" "Payment.Terms"     "Item.ID"
[6] "Qty"
```

```
> fit <- lm(Qty ~ Payment.Terms , data = ords)
> names(fit)
[1] "coefficients"  "residuals"    "effects"      "rank"         "fitted.values" "assign"
[7] "qr"           "df.residual"  "contrasts"    "xlevels"      "call"         "terms"
[13] "model"
> colnames(fit)
NULL
```


Setting the current working directory

- Instead of calling the same path multiple no. of times we can set the working directory to the frequently used file path with the help of the function `setwd()`

```
> setwd("D:/Data Science Training/Datasets")
```

Hence instead of typing as...

```
> read.csv("D:/Data Science Training/Datasets/Bollywood_2015.csv")
```

We can simply type as...

```
> read.csv("Bollywood_2015.csv")
```

FILE INPUT AND OUTPUT

Reading Data

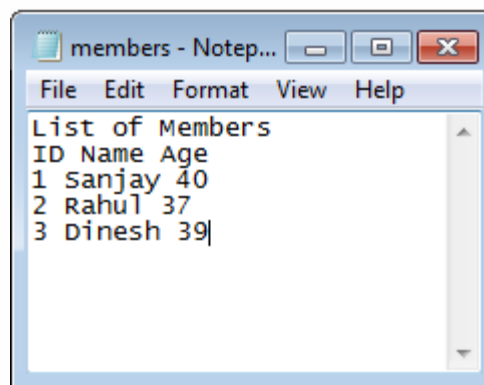
- The data from files read gets directly stored into data frame objects
- For reading data from flat (notepad) files we can use any of the functions:
 - `read.csv()`
 - `read.table()`
- For reading data from Excel format (.xlsx) we can use `read.xlsx` from `xlsx` package

read.table() - Some Important Arguments

```
read.table(file, header = FALSE, sep = "",  
row.names, col.names, colClasses = NA, skip = 0,...)
```

- This function is a general form of read.csv() and read.csv2()
- The default values have been shown above
- file : the name of the file which the data are to be read from
- header : (optional) a logical value indicating whether the file contains the names of the variables as its first line. Here default is FALSE
- row.names : a vector of row names
- col.names : a vector of column names
- colClasses : a character vector indicating the class of each column in the dataset
- skip : no. of lines to skip from the top of the file
- sep : separator (optional). Default is space
- stringsAsFactors : (optional) logical indicating should the character vectors be converted to factors?

Example of read.table()



```
mem <- read.table("D:/Data Science Training/Datasets/members.txt",header=TRUE,  
  colClasses = c("character","character","integer"),skip=1,  
  sep=" ")
```

ID	Name	Age
1	Sanjay	40
2	Rahul	37
3	Dinesh	39

read.csv()

- This function assumes comma as a delimiter by default while we read a file
- It assumes that file already has a header i.e. a line indicating the names of the variables separated by a delimiters

```

Bollywood_2015 - Notepad
File Edit Format View Help
Movie Name,BO_
Collection,Budget,Box_Office_Verdict
Pyaar Ka PUNCHnama 2,53.25,25,Hit
Shandaar ,38.28,68,Flop
Singh is Bliing ,74.87 ,92,Flop
Jazbaa ,24.3,30,Flop
Talvar,24,22,Plus
Kis Kisko Pyaar Karoon ,44.8,20,Hit
Calendar Girls ,7,12,Flop
Katti Batti ,22.96,36,Flop
Hero ,31.75,28,Plus
welcome Back ,96.88,100,Average
Phantom ,49.84,70,Flop
All is well,12.65,36,Flop
Manjhi,13.36,8.5,Plus
Brothers ,77.18,82,Flop
Drishyam ,65.22 ,65,Average
Bajrangi Bhaijaan,318.14,125,All Time
  
```

```

> bollywood <- read.csv("D:/Data Science Training/Datasets/Bollywood_2015.csv")
> bollywood

```

	Movie.Name	BO_Collection	Budget	Box_Office_Verdict
1	Pyaar Ka PUNCHnama 2	53.25	25.0	Hit
2	Shandaar	38.28	68.0	Flop
3	Singh is Bliing	74.87	92.0	Flop
4	Jazbaa	24.30	30.0	Flop
5	Talvar	24.00	22.0	Plus
6	Kis Kisko Pyaar Karoon	44.80	20.0	Hit
7	Calendar Girls	7.00	12.0	Flop
8	Katti Batti	22.96	36.0	Flop

read.csv() arguments

```
read.csv(file, header = TRUE, sep = ",", stringsAsFactors ...)
```

- The default values have been shown above
 - file : the name of the file which the data are to be read from
 - header : (optional) a logical value indicating whether the file contains the names of the variables as its first line
 - sep : separator (optional)
 - stringsAsFactors : (optional) logical indicating should the character vectors be converted to factors?

read.csv2() arguments

```
read.csv2(file, header = TRUE, sep = ";", stringsAsFactors...)
```

- The default values have been shown above
 - file : the name of the file which the data are to be read from
 - header : (optional) a logical value indicating whether the file contains the names of the variables as its first line
 - sep : separator (optional). Default is semi-colon
 - stringsAsFactors : (optional) logical indicating should the character vectors be converted to factors?

Reading from MS Excel Sheet

- There are various alternatives for reading Excel sheet. One of them is provided by package `xlsx`
- We can use `read.xlsx()` to read the Excel sheet

Syntax: `read.xlsx(file,sheetNo)`

file : file path

sheetNo : sheet number of the sheet to be read

```
install.packages("xlsx")  
library(xlsx)  
bank <- read.xlsx("D:\\Data Science Training\\Datasets\\bankruptcy.xlsx",3)
```

Writing to a file

- For writing to file the functions `write.table()`, `write.csv()` etc. can be used

Syntax Usage : `write.table(data frame , file path)`

`write.csv(data frame , file path)`

```
write.table(Pens3,"Pen.txt")
```

```
write.csv(Pens3,"Pen.csv")
```

SUBSETTING THE DATA

Subsetting a vector

- A vector can be subsetting by typing various options in between the brackets []

```
> x <- c(12,23,52,78,90,10,28,93,95,92,95,79)
> x[1:5]
[1] 12 23 52 78 90
> x>50
[1] FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
> x[x>50]
[1] 52 78 90 93 95 92 95 79
```

Subsetting a list

- A list can be subsetting by typing various options in between the brackets []

```
> f <- list(a=1:7, b="XYZ",c=FALSE,d=c(23.4,14,6))
> f[1]
$a
[1] 1 2 3 4 5 6 7

> f[2]
$b
[1] "XYZ"

> f$c
[1] FALSE
> f[["c"]]
[1] FALSE
> f["c"]
$c
[1] FALSE
```

Subsetting a matrix

- Matrix can be subsetting by specifying the row or column numbers

```
> m <- matrix(c(1:12),4,3)
> m
      [,1] [,2] [,3]
[1,]     1     5     9
[2,]     2     6    10
[3,]     3     7    11
[4,]     4     8    12
> m[3,2]
[1] 7
> m[2,]
[1]  2  6 10
> m[,3]
[1]  9 10 11 12
> m[,3,drop=FALSE]
      [,1]
[1,]     9
[2,]    10
[3,]    11
[4,]    12
```

Specifying the column / row indices

- Subsetting can be done to a data frame by specifying rows / columns

```
> items[2,] # only 2nd row
Item.ID          Item.Name Item.Type Brand Price  UOM
2 121 002 Pilot V7 Liquid Ink Roller Ball Pen (2 Blue + 1 Black) Pen Pilot 135 Pack
> items[c(1,3,5,6),] # only rows 1,3,5,6
Item.ID          Item.Name Item.Type Brand Price  UOM
1 121 001 Parker Quink Roller Ball Pen Refill, Blue Pen Parker 69 Piece
3 121 003 Parker Beta Premium Gold Ball Pen Pen Parker 125 Piece
5 121 005 Reynolds 045 Fine Carbure Blue Ballpen, Pack of 10 Pen Reynolds 60 Pack
6 121 006 Pilot FriXion Roller Ball Pen, Blue Pen Parker 92 Piece
> items[,4] # only 4th column
[1] Parker      Pilot      Parker      Pilot      Reynolds    Parker      Staedtler    Parker
[9] Puro         Cello      Staedtler    Staedtler    Sheaffer     Lamy         Pierre Cardin Pierre Cardin
[17] Reynolds    Camlin     Camlin      Camlin      Artline     Luxor        Pilot        Camlin
[25] Artline
Levels: Artline Camlin Cello Lamy Luxor Parker Pierre Cardin Pilot Puro Reynolds Sheaffer Staedtler
> items[,c(2,4)] # only 2nd and 4th column
          Item.Name          Brand
1 Parker Quink Roller Ball Pen Refill, Blue Parker
2 Pilot V7 Liquid Ink Roller Ball Pen (2 Blue + 1 Black) Pilot
3 Parker Beta Premium Gold Ball Pen Parker
4 Pilot V5 Liquid Ink Roller Ball Pen - 1 Blue + 1 Black + 1 Red Pilot
5 Reynolds 045 Fine Carbure Blue Ballpen, Pack of 10 Reynolds
6 Pilot FriXion Roller Ball Pen, Blue Parker
7 Staedtler Triangular Ball Pen (Set of 10 Colours) Staedtler
8 Parker Vector Metallix CT Roller Ball Pen (Blue) + Swiss Knife Parker
```

```
> items[,-c(2,4)] # exclude 2nd and 4th column
Item.ID Item.Type Price  UOM
1 121 001 Pen 69 Piece
2 121 002 Pen 135 Pack
3 121 003 Pen 125 Piece
4 121 004 Pen 135 Pack
```

subset ()

- subset() function gives the data frame object

Syntax: subset(data frame , condition, select, ...)

```
> Pens2 <- subset(items,Item.Type=="Pen")
```

	Item.ID	Item.Name	Item.Type	Brand	Price	UOM
1	121 001	Parker Quink Roller Ball Pen Refill, Blue	Pen	Parker	69	Piece
2	121 002	Pilot V7 Liquid Ink Roller Ball Pen (2 Blue + 1 Black)	Pen	Pilot	135	Pack
3	121 003	Parker Beta Premium Gold Ball Pen	Pen	Parker	125	Piece
4	121 004	Pilot V5 Liquid Ink Roller Ball Pen - 1 Blue + 1 Black + ...	Pen	Pilot	135	Pack
5	121 005	Reynolds 045 Fine Carbure Blue Ballpen, Pack of 10	Pen	Reynolds	60	Pack
6	121 006	Pilot Frixion Roller Ball Pen, Blue	Pen	Parker	92	Piece
7	121 007	Staedtler Triangular Ball Pen (Set of 10 Colours)	Pen	Staedtler	160	Pack

subset() Contd...

```
ExpPens <- subset(items,Item.Type=="Pen" & Price>200)
```

	Item.ID	Item.Name	Item.Type	Brand	Price	UOM
8	121 008	Parker Vector Mettalix CT Roller Ball Pen (Blue) + Swis...	Pen	Parker	316	Pack
11	121 011	Staedtler 334 SB4 Triplus Fineliner Pen - Multicolor B...	Pen	Staedtler	320	Pack
12	121 012	Staedtler Luna RiteClic Ball Pen - Transparent Body, B...	Pen	Staedtler	300	Pack
16	121 016	Pierre Cardin Kriss Satin Nickle Roller Pen and Ball Pe...	Pen	Pierre Cardin	300	Pack

```
HighRef <- subset(items,Item.Type=="Highlighter" | Item.Type=="Refill" )
```

	Item.ID	Item.Name	Item.Type	Brand	Price	UOM
14	121 014	Lamy M63 Blue Rollerball Refill	Refill	Lamy	310	Piece
18	121 018	Camlin Office Highlighter - Pack of 5 Assorted Colors	Highlighter	Camlin	100	Pack
19	121 019	Camlin Office Highlighter Pen, Yellow	Highlighter	Camlin	190	Pack
23	121 023	Pilot Frixion Colour Highlighter (Pack of 6)	Highlighter	Pilot	465	Pack

subset() Contd...

```
isItem <- subset(items , select = c(Item.ID, Brand, Price))
```

	Item.ID	Brand	Price
1	121 001	Parker	69
2	121 002	Pilot	135
3	121 003	Parker	125
4	121 004	Pilot	135
5	121 005	Reynolds	60
6	121 006	Parker	92
7	121 007	Staedtler	160
8	121 008	Parker	316
9	121 009	Puro	179
10	121 010	Cello	90
11	121 011	Staedtler	320
12	121 012	Staedtler	300