# Introduction to R

Dr. Arabin Kumar Dey

Assistant Professor
**Department of Mathematics**
**Indian Institute of Technology Guwahati**

**December 28, 2011**

## Outline

1. Installing R
2. Why using R ?
3. Working with Data Set
4. Using Data Available in R

## Outline

1 Installing R

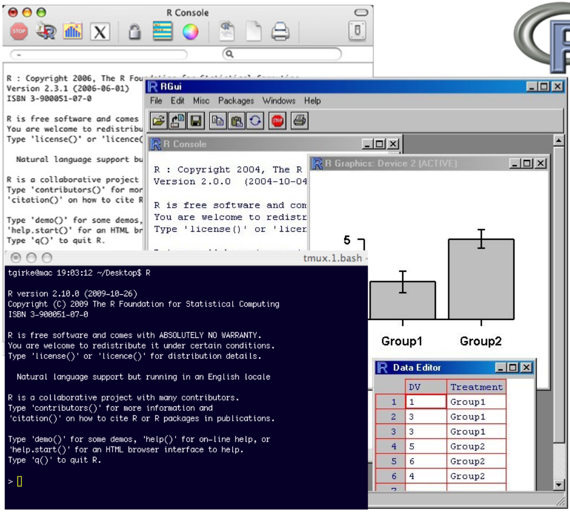
2 Why using R ?

3 Working with Data Set

4 Using Data Available in R

## Installing R on Linux/Windows

- Go to http://cran.r-project.org
- Select to download the latest version:
- Install and Open. The R window should look like :


R Console
RGui
File Edit Misc Packages Windows Help
R Console
R Graphics: Device 2 (ACTIVE)
5
Group1
Group2
Data Editor
tmux.1.bash

```
R : Copyright 2006, The R Fou
Version 2.3.1 (2006-06-01)
ISBN 3-900051-07-0

R is free software and comes
You are welcome to redistribu
Type 'license()' or 'licence(

  Natural language support bu

R is a collaborative project
Type 'contributors()' for mor
'citation()' on how to cite R

Type 'demo()' for some demos,
'help.start()' for an HTML br
Type 'q()' to quit R.
```

```
R : Copyright 2004, The R
Version 2.0.0  (2004-10-04

R is free software and com
You are welcome to redistr
Type 'license()' or 'licer
```

```
tgirke@mac 19:03:12 ~/Desktop$ R

R version 2.10.0 (2009-10-26)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

| | DV | Treatment |
|---|---|---|
| 1 | 1 | Group1 |
| 2 | 3 | Group1 |
| 3 | 3 | Group1 |
| 4 | 5 | Group2 |
| 5 | 6 | Group2 |
| 6 | 4 | Group2 |

## Outline

1 Installing R

2 Why using R ?

3 Working with Data Set

4 Using Data Available in R

- Complete statistical package and programming language
- Efficient functions and data structures for data analysis
- Powerful graphics
- Access to fast growing number of analysis packages
- Technical advantages: free, open-source, available for all OSs

- Complete statistical package and programming language
- Efficient functions and data structures for data analysis
- Powerful graphics
- Access to fast growing number of analysis packages
- Technical advantages: free, open-source, available for all OSs

- Complete statistical package and programming language
- Efficient functions and data structures for data analysis
- Powerful graphics
- Access to fast growing number of analysis packages
- Technical advantages: free, open-source, available for all OSs

- Complete statistical package and programming language
- Efficient functions and data structures for data analysis
- Powerful graphics
- Access to fast growing number of analysis packages
- Technical advantages: free, open-source, available for all OSs

- Complete statistical package and programming language
- Efficient functions and data structures for data analysis
- Powerful graphics
- Access to fast growing number of analysis packages
- Technical advantages: free, open-source, available for all OSs

## Startup/Closing Behavior

- Starting R The R GUI versions under Windows and Mac OS X can be opened by double-clicking their icons.
- Alternatively, one can start it by typing ‘R’ in a terminal (default under Linux).
- The R environment is controlled by hidden files in the startup directory:.RData, .Rhistory and .Rprofile (optional).
- ## Closing R
  $> q()$
  Save workspace image?
  $[y/n/c]$ :
- Note When responding with ‘y’, then the entire R workspace will be written to the .RData file which can become very large. Often it is sufficient to just save an analysis protocol in an R source file. This way one can quickly regenerate all data sets and objects.

## Startup/Closing Behavior

- Starting R The R GUI versions under Windows and Mac OS X can be opened by double-clicking their icons.
- Alternatively, one can start it by typing ‘R’ in a terminal (default under Linux).
- The R environment is controlled by hidden files in the startup directory:.RData, .Rhistory and .Rprofile (optional).
- ## Closing R
  $> q()$
  Save workspace image?
  $[y/n/c]$ :
- Note When responding with ‘y’, then the entire R workspace will be written to the .RData file which can become very large. Often it is sufficient to just save an analysis protocol in an R source file. This way one can quickly regenerate all data sets and objects.

## Startup/Closing Behavior

- Starting R The R GUI versions under Windows and Mac OS X can be opened by double-clicking their icons.
- Alternatively, one can start it by typing 'R' in a terminal (default under Linux).
- The R environment is controlled by hidden files in the startup directory:.RData, .Rhistory and .Rprofile (optional).
- ## Closing R
  $> q()$
  Save workspace image?
  $[y/n/c]$ :
- Note When responding with 'y', then the entire R workspace will be written to the .RData file which can become very large. Often it is sufficient to just save an analysis protocol in an R source file. This way one can quickly regenerate all data sets and objects.

## Startup/Closing Behavior

- Starting R The R GUI versions under Windows and Mac OS X can be opened by double-clicking their icons.
- Alternatively, one can start it by typing ‘R’ in a terminal (default under Linux).
- The R environment is controlled by hidden files in the startup directory:.RData, .Rhistory and .Rprofile (optional).
- ## Closing R
  $> q()$
  Save workspace image?
  $[y/n/c]$ :
- Note When responding with ‘y’, then the entire R workspace will be written to the .RData file which can become very large. Often it is sufficient to just save an analysis protocol in an R source file. This way one can quickly regenerate all data sets and objects.

## Startup/Closing Behavior

- Starting R The R GUI versions under Windows and Mac OS X can be opened by double-clicking their icons.
- Alternatively, one can start it by typing ‘R’ in a terminal (default under Linux).
- The R environment is controlled by hidden files in the startup directory:.RData, .Rhistory and .Rprofile (optional).
- ## Closing R
  $> q()$
  Save workspace image?
  $[y/n/c]$ :
- Note When responding with ‘y’, then the entire R workspace will be written to the .RData file which can become very large. Often it is sufficient to just save an analysis protocol in an R source file. This way one can quickly regenerate all data sets and objects.

## Installation of Cran package

- Linux - install.packages("mypackage_1.0.tar.gz", repos=NULL)
- Hands-on Experience $\cdots$

- ## Create an object with the assignment operator '$<-$' (or '$=$')
  $> object <- \cdots$
- ## List objects in current R session
  $> ls()$
- ## Return content of current working directory
  $> dir()$
- ## Return path of current working directory
  $> getwd()$
- ## Change current working directory
  $> setwd("/home/user")$

## Basic R syntex

- ## General R command syntax
  > object < − function(arguments)
  > object < − object[arguments]
- ## Execute an R script
  > source("my script.R")
- ## Execute an R script from command-line
  > R CMD BATCH my_script.R
  > R –slave < my_script.R
- ## Finding help
  >?function
- ## Load a library
  > library("my_library")
- ## Summary of all functions within a library
  > library(help="my_library")

## Basic R syntex

- ## General R command syntax
  $>$ object $<-$ function(arguments)
  $>$ object $<-$ object[arguments]
- ## Execute an R script
  $>$ source("my script.R")
- ## Execute an R script from command-line
  $>$ R CMD BATCH my_script.R
  $>$ R –slave $<$ my_script.R
- ## Finding help
  $>$?function
- ## Load a library
  $>$ library("my_library")
- ## Summary of all functions within a library
  $>$ library(help="my_library")

## Basic R syntex

- ## General R command syntax
  $>$ object $<-$ function(arguments)
  $>$ object $<-$ object[arguments]
- ## Execute an R script
  $>$ source("my script.R")
- ## Execute an R script from command-line
  $>$ R CMD BATCH my_script.R
  $>$ R –slave $<$ my_script.R
- ## Finding help
  $>$?function
- ## Load a library
  $>$ library("my_library")
- ## Summary of all functions within a library
  $>$ library(help="my_library")

## Basic R syntex

- ## General R command syntax
  $>$ object $<-$ function(arguments)
  $>$ object $<-$ object[arguments]
- ## Execute an R script
  $>$ source("my script.R")
- ## Execute an R script from command-line
  $>$ R CMD BATCH my_script.R
  $>$ R –slave $<$ my_script.R
- ## Finding help
  $>$?function
- ## Load a library
  $>$ library("my_library")
- ## Summary of all functions within a library
  $>$ library(help="my_library")

## Basic R syntex

- ## General R command syntax
  > object < − function(arguments)
  > object < − object[arguments]
- ## Execute an R script
  > source("my script.R")
- ## Execute an R script from command-line
  > R CMD BATCH my_script.R
  > R –slave < my_script.R
- ## Finding help
  >?function
- ## Load a library
  > library("my_library")
- ## Summary of all functions within a library
  > library(help="my_library")

## Basic R syntex

- ## General R command syntax
  $>$ object $<-$ function(arguments)
  $>$ object $<-$ object[arguments]
- ## Execute an R script
  $>$ source("my script.R")
- ## Execute an R script from command-line
  $>$ R CMD BATCH my_script.R
  $>$ R –slave $<$ my_script.R
- ## Finding help
  $>$?function
- ## Load a library
  $>$ library("my_library")
- ## Summary of all functions within a library
  $>$ library(help="my_library")

## Data Type

- ## Numeric data: 1, 2, 3 $> x < -c(1, 2, 3)$; x; is.numeric(x); as.character(x)
- ## Character data: "a", "b", "c" $> x < -c("1", "2", "3")$; x; is.character(x); as.numeric(x)
- ## Logical data: TRUE, FALSE, TRUE
  $> x < -1 : 10 < 5; x$
  $>!x$
- ## Return indices for the 'TRUEs' in logical vector
  $> which(x)$

## Data Type

- ## Numeric data: 1, 2, 3 $> x < -c(1, 2, 3)$; x; is.numeric(x); as.character(x)
- ## Character data: "a", "b", "c" $> x < -c("1", "2", "3")$; x; is.character(x); as.numeric(x)
- ## Logical data: TRUE, FALSE, TRUE
  $> x < -1 : 10 < 5; x$
  $> !x$
- ## Return indices for the 'TRUEs' in logical vector
  $> which(x)$

## Data Type

- ## Numeric data: 1, 2, 3 $> x < -c(1, 2, 3)$; x; is.numeric(x); as.character(x)
- ## Character data: "a", "b", "c" $> x < -c(''1'', ''2'', ''3'')$; x; is.character(x); as.numeric(x)
- ## Logical data: TRUE, FALSE, TRUE
  $> x < -1 : 10 < 5; x$
  $>!x$
- ## Return indices for the 'TRUEs' in logical vector
  $> which(x)$

## Data Type

- ## Numeric data: 1, 2, 3 $> x < -c(1, 2, 3)$; x; is.numeric(x); as.character(x)
- ## Character data: "a", "b", "c" $> x < -c("1", "2", "3")$; x; is.character(x); as.numeric(x)
- ## Logical data: TRUE, FALSE, TRUE
  $> x < -1 : 10 < 5; x$
  $>!x$
- ## Return indices for the 'TRUEs' in logical vector
  $> which(x)$

## Caution !!

On the other hand, if you give a variable the same name as an existing function, R will treat the identifier as a variable if used as a variable, and will treat it as a function when it is used as a function:
$c < -2$ #typing c yields "2"
$c(c, c)$ #yields a vector containing two 2s.

**Better to avoid using 'c' as a variable**

## Data Objects

- ## Vectors (1D)
  > myVec < − 1:10; names(myVec) < − letters[1:10]
  > myVec[1:5]; myVec[c(2,4,6,8)]; myVec[c("b", "d", "f")]
- ## Matrices (2D), Data Frames (2D) and Arrays (≥2D)
  > myMA <- matrix(1:30, 3, 10, byrow = T)
  > myDF <- data.frame(Col1=1:10, Col2=10:1)
  > myDF[1:4, ]; myDF[ ,c("Col2", "Col1", "Col1")]
- ## Lists: containers for any object type
  > myL < − list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))
  > myL[[4]][1:2]
- ## Functions: piece of code > myfct < − function(arg1, arg2, $\cdots$)

## General Subsetting Rules

- ## Subsetting by indices
  $>$ myVec $< -$ 1:26; names(myVec) $< -$ LETTERS
  $> myVec[1:4]$
- ## Subsetting by same length logical vectors
  $>$ myLog $< -$ myVec $>$ 10
  $>$ myVec[myLog]
- ## Subsetting by field names
  $>$ myVec[c("B", "K", "M")]
- ## Special case
  $>$ iris$Species

## Basic Operators and Calculations

- Comparison operators: $==, != , <, >, <=, >=$
  ## Example:
  $> 1 == 1$
- Logical operators: $AND$ : &, $OR$ : |, $NOT$ :!
  ## Example:
  $> x < -1:10;\ y < -10:1$
  $> x > y\ \&\ x > 5$
- Calculations: ## Example:
  $> x + y; sum(x); mean(x), sd(x); sqrt(x)$
  > apply(matrix(c(1,2,3,4,4,5,6,7,8,9,9,4),3,4), 1, mean)

## Outline

1 Installing R

2 Why using R ?

3 Working with Data Set

4 Using Data Available in R

- ## Import Data into R
  > read.delim("myData.csv") ??
  > a< −read.table("myData.txt");
- Extract the portion you want to use, for example, $x < -a[,2]$
- ## Export Data from R to File
  > write.table(myframe, file="myfile.csv", sep="\t", quote=F)

## Some Great R Functions

- The **unique()** function to make vector entries unique
  > unique(iris$Sepal.Length); length(unique(iris$Sepal.Length))
- The **table()** function counts the occurrences of entries
  > table(iris$Species)
- The **aggregate()** function computes statistics of data aggregates
  > aggregate(iris[,1:4], by=list(iris$Species), FUN=mean, na.rm=T)
- The %in% function returns the intersect between two vectors
  > month.name[month.name %in% c("May", "July")]
- The **merge()** function joins data frames based on a common key column
  > merge(frame1, frame2, by.x=1, by.y=1, all = TRUE)

## Some Great R Functions

- The **unique()** function to make vector entries unique
  > unique(iris$Sepal.Length);
  length(unique(iris$Sepal.Length))
- The **table()** function counts the occurrences of entries
  > table(iris$Species)
- The **aggregate()** function computes statistics of data aggregates
  > aggregate(iris[,1:4], by=list(iris$Species), FUN=mean, na.rm=T)
- The %in% function returns the intersect between two vectors
  > month.name[month.name %in% c("May", "July")]
- The **merge()** function joins data frames based on a common key column
  > merge(frame1, frame2, by.x=1, by.y=1, all = TRUE)

## Some Great R Functions

- The **unique()** function to make vector entries unique
  > unique(iris$Sepal.Length);
  length(unique(iris$Sepal.Length))
- The **table()** function counts the occurrences of entries
  > table(iris$Species)
- The **aggregate()** function computes statistics of data aggregates
  > aggregate(iris[,1:4], by=list(iris$Species), FUN=mean, na.rm=T)
- The %in% function returns the intersect between two vectors
  > month.name[month.name %in% c("May", "July")]
- The **merge()** function joins data frames based on a common key column
  > merge(frame1, frame2, by.x=1, by.y=1, all = TRUE)

## Some Great R Functions

- The **unique()** function to make vector entries unique
  > unique(iris$Sepal.Length); length(unique(iris$Sepal.Length))
- The **table()** function counts the occurrences of entries
  > table(iris$Species)
- The **aggregate()** function computes statistics of data aggregates
  > aggregate(iris[,1:4], by=list(iris$Species), FUN=mean, na.rm=T)
- The %in% function returns the intersect between two vectors
  > month.name[month.name %in% c("May", "July")]
- The **merge()** function joins data frames based on a common key column
  > merge(frame1, frame2, by.x=1, by.y=1, all = TRUE)

## Some Great R Functions

- The **unique()** function to make vector entries unique
  > unique(iris$Sepal.Length);
  length(unique(iris$Sepal.Length))
- The **table()** function counts the occurrences of entries
  > table(iris$Species)
- The **aggregate()** function computes statistics of data aggregates
  > aggregate(iris[,1:4], by=list(iris$Species), FUN=mean, na.rm=T)
- The %in% function returns the intersect between two vectors
  > month.name[month.name %in% c("May", "July")]
- The **merge()** function joins data frames based on a common key column
  > merge(frame1, frame2, by.x=1, by.y=1, all = TRUE)

## Outline

1 Installing R

2 Why using R ?

3 Working with Data Set

4 Using Data Available in R

To use a data set available in one of the R packages, install that package (if needed). Load the package into R, using the library() function.

```
> library(MASS)
```

Extract the data set you want from that package, using the data() function. In our case, the data set is called JohnsonJohnson.

```
> data(JohnsonJohnson)
```

## Working with Datasets in R

To use the variable names when working with data, use attach():
> data(JohnsonJohnson)
> attach(JohnsonJohnson)
After the variable names have been "attached", to see the variable names, use names():
> names(JohnsonJohnson) To see the descriptions of the variables, use ?:
> ?JohnsonJohnson

After modifying variables, use detach() and attach() to save the results:

```
# Make a copy of the data set
johnson.copy <- JohnsonJohnson;
detach(JohnsonJohnson)
attach (johnson.copy)
# Change the 10 th observation for JohnsonJohnson
johnson.copy[10,1] <- 999
```

## Caution!!

Avoid using attach() if possible. Many strange things can occur if you accidentally attach the same data frame multiple times, or forget to detach. Instead, you can refer to a variable using $.

**Moral of the story**

**"attach at your own risk!"**