Introduction to R

Yuwu Chen
HPC @ LSU

Some materials are borrowed from the EXST 7142/7152 data mining courses by Dr. Bin Li at Statistics Dept.

Outline

- R basics
  - What is R
  - How to run R codes
  - Basic syntax
  - R as a calculator
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis
  - Data acquisition and inspection
  - Data preprocessing
  - Statistical analysis
  - Report generation

What is R

- R is an integrated suite of software facilities for
  - importing, storing, exporting and manipulating data;
  - scientific computation;
  - conducting statistical analyses;
  - displaying the results by tables, graphs, etc.
- Highly customizable via thousands of freely available packages.
- R is also a platform for the development and implementation of new algorithms.
- Many graphical user interface to R both free and commercial

  (e.g. Rstudio and Revolution R (now Microsoft R)).

What is R

- R mailing lists: http://www.R-project.org/mail.html
  - R-announce: announcements of major R developments.
  - R-packages: announcements of new R packages.
  - R-help: main discussion list.
  - R-devel: discussion on code development in R.
  - Special interest group (e.g. R-SIG-Finance).

History of R

- R is a dialect of the S language
  - S was created in 1976 at the Bell Labs as an internal statistical analysis environment
  - Goal of S was “to turn ideas into software, quickly and faithfully".
  - Most well known implementation is S-plus (most recent stable release was in 2010). S-Plus integrates S with a nice GUI interface and full customer support.
  - R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand.
- The R core group was formed in 1997, who controls the source code of R (written in C)
- The stable beta version R 1.0.0 was released in 2000
- Latest version is 3.4.2 released on September 28, 2017

Features of R

- R is a language designed for statistical analysis
- Available on most platform/OS
- Rich data analysis functionalities and sophisticated graphical capabilities
- Active development and very active community
  - CRAN: The **C**omprehensive **R** **A**rchive **N**etwork
    - Source code and binaries, user contributed packages and documentation
  - More than 11,000 packages available on CRAN (as of September 2017)
    - 6,000 two years ago
- Free to use!

Outline

- R basics
  - What is R
  - How to run R codes
  - Basic syntax
  - R as a calculator
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis
  - Data acquisition and inspection
  - Data preprocessing
  - Statistical analysis
  - Report generation

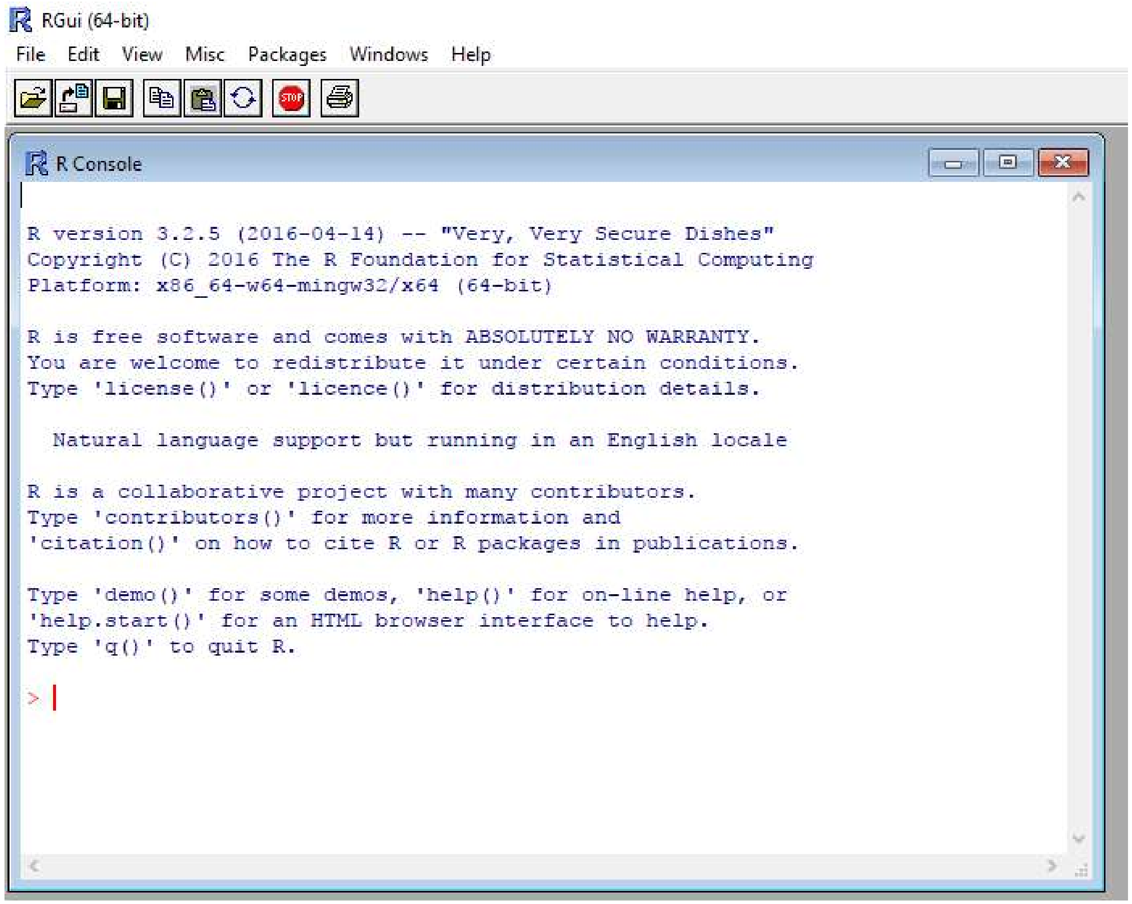
Installing and loading R

- On your PC
  - R console can be downloaded from: http://cran.r-project.org/
  - Rstudio is the de facto environment for R on a desktop system
- On a cluster
  - R is installed on all LONI and LSU HPC clusters
    - QB2: `r/3.1.0/INTEL-14.0.2`
    - SuperMIC: `r/3.1.0/INTEL-14.0.2`
    - Philip: `r/3.1.3/INTEL-15.0.3`
    - SuperMike2: `+R-3.3.3-gcc-4.7.2`

R console

- Linux/Mac/Windows version available
- Limited graphic user interface (GUI)
- Command line interface (CLI) is similar to HPC environment

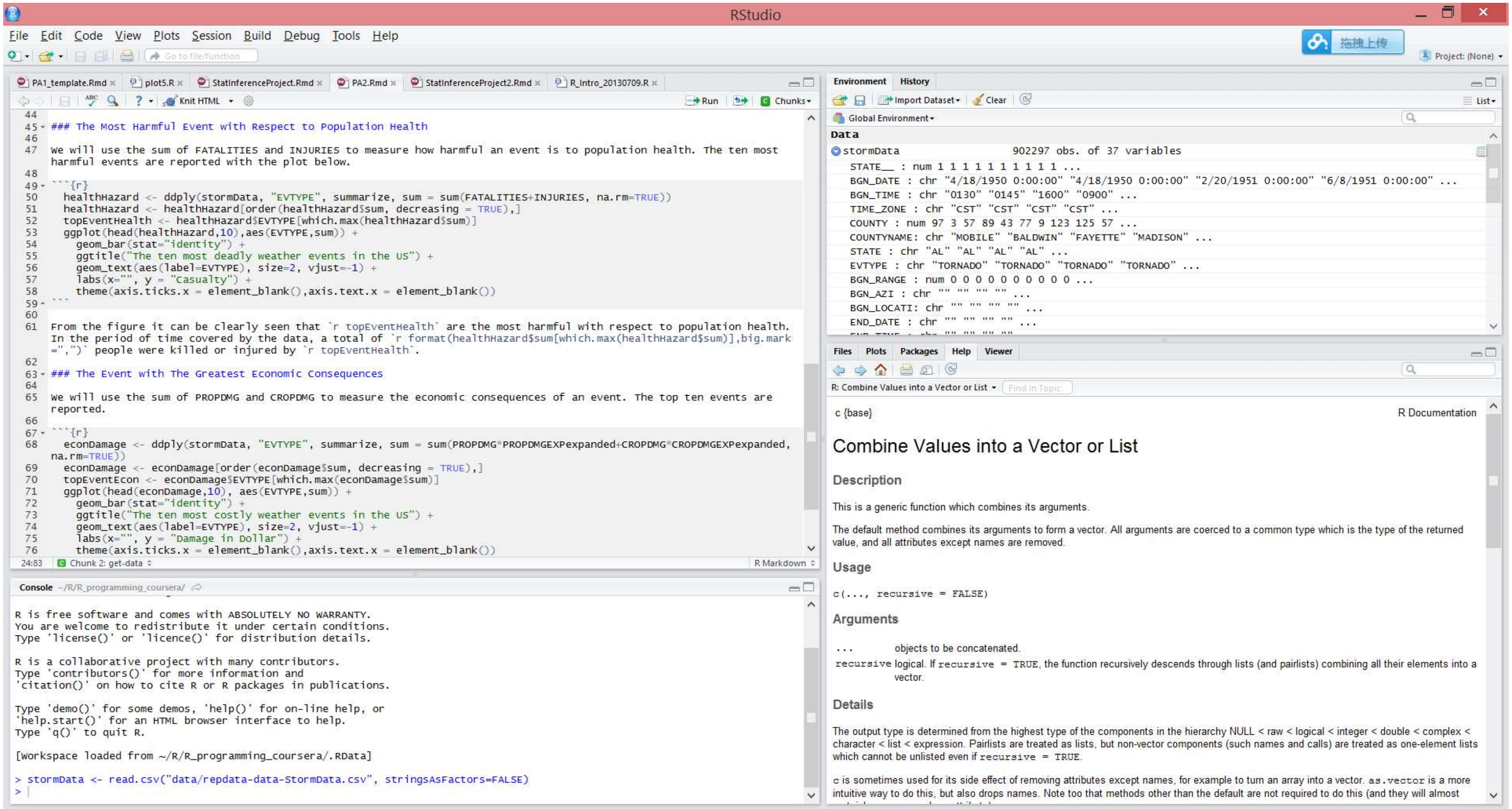
R console



RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
R version 3.2.5 (2016-04-14) -- "Very, Very Secure Dishes"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
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.
>

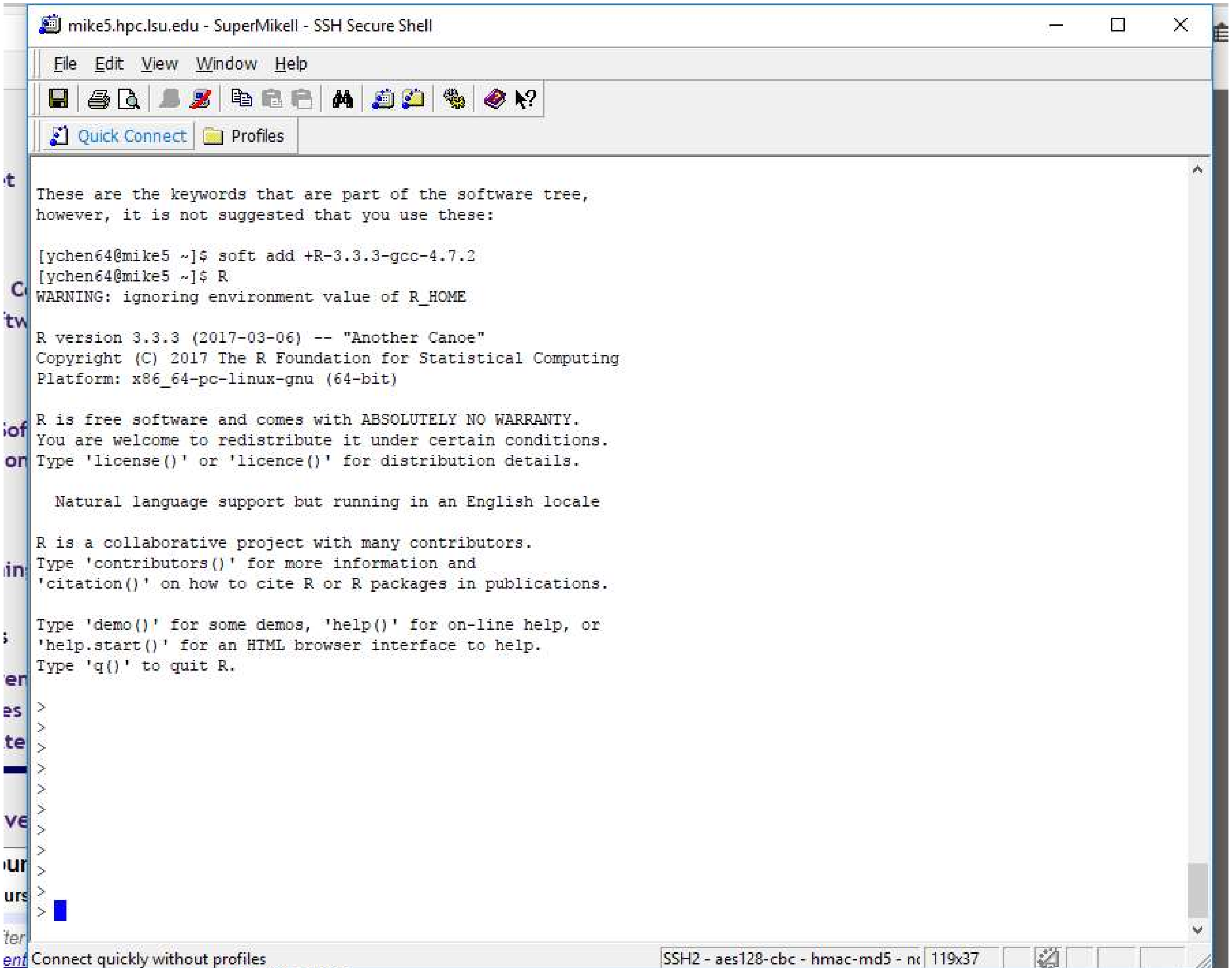
RStudio

- Similar graphic user interface (GUI) to other Windows software, dividing the screen into panes
  - Source code
  - Console
  - Workspace
  - Others (help message, plot etc.)
- Rstudio in a desktop environment is better suited for development and/or a limited number of small jobs


RStudio
File Edit Code View Plots Session Build Debug Tools Help
PA1_template.Rmd
plot5.R
StatInferenceProject.Rmd
PA2.Rmd
StatInferenceProject2.Rmd
R_Intro_20130709.R
Run
Chunks
Knit HTML
The Most Harmful Event with Respect to Population Health
We will use the sum of FATALITIES and INJURIES to measure how harmful an event is to population health. The ten most harmful events are reported with the plot below.
```{r}
healthHazard <- ddply(stormData, "EVTYPE", summarize, sum = sum(FATALITIES+INJURIES, na.rm=TRUE))
healthHazard <- healthHazard[order(healthHazard$sum, decreasing = TRUE),]
topEventHealth <- healthHazard$EVTYPE[which.max(healthHazard$sum)]
ggplot(head(healthHazard,10),aes(EVTYPE,sum)) +
geom_bar(stat="identity") +
ggtitle("The ten most deadly weather events in the US") +
geom_text(aes(label=EVTYPE), size=2, vjust=-1) +
labs(x="", y = "Casualty") +
theme(axis.ticks.x = element_blank(),axis.text.x = element_blank())
```
From the figure it can be clearly seen that `r topEventHealth` are the most harmful with respect to population health. In the period of time covered by the data, a total of `r format(healthHazard$sum[which.max(healthHazard$sum)],big.mark=",")` people were killed or injured by `r topEventHealth`.
The Event with The Greatest Economic Consequences
We will use the sum of PROPDMG and CROPDMG to measure the economic consequences of an event. The top ten events are reported.
```{r}
econDamage <- ddply(stormData, "EVTYPE", summarize, sum = sum(PROPDMG*PROPDMGEXPexpanded+CROPDMG*CROPDMGEXPexpanded, na.rm=TRUE))
econDamage <- econDamage[order(econDamage$sum, decreasing = TRUE),]
topEventEcon <- econDamage$EVTYPE[which.max(econDamage$sum)]
ggplot(head(econDamage,10), aes(EVTYPE,sum)) +
geom_bar(stat="identity") +
ggtitle("The ten most costly weather events in the US") +
geom_text(aes(label=EVTYPE), size=2, vjust=-1) +
labs(x="", y = "Damage in Dollar") +
theme(axis.ticks.x = element_blank(),axis.text.x = element_blank())
Chunk 2: get-data
R Markdown
Console ~/R/R_programming_coursera/
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.
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.
[workspace loaded from ~/R/R_programming_coursera/.RData]
> stormData <- read.csv("data/repdata-data-StormData.csv", stringsAsFactors=FALSE)
Environment History
Import Dataset
Clear
Global Environment
Data
stormData 902297 obs. of 37 variables
STATE__ : num 1 1 1 1 1 1 1 1 1 1 ...
BGN_DATE : chr "4/18/1950 0:00:00" "4/18/1950 0:00:00" "2/20/1951 0:00:00" "6/8/1951 0:00:00" ...
BGN_TIME : chr "0130" "0145" "1600" "0900" ...
TIME_ZONE : chr "CST" "CST" "CST" "CST" ...
COUNTY : num 97 3 57 89 43 77 9 123 125 57 ...
COUNTYNAME: chr "MOBILE" "BALDWIN" "FAYETTE" "MADISON" ...
STATE : chr "AL" "AL" "AL" "AL" ...
EVTYPE : chr "TORNADO" "TORNADO" "TORNADO" "TORNADO" ...
BGN_RANGE : num 0 0 0 0 0 0 0 0 0 0 ...
BGN_AZI : chr "" "" "" "" ...
BGN_LOCATI: chr "" "" "" "" ...
END_DATE : chr "" "" "" "" ...
Files Plots Packages Help Viewer
R: Combine Values into a Vector or List
c {base}
R Documentation
Combine Values into a Vector or List
Description
This is a generic function which combines its arguments.
The default method combines its arguments to form a vector. All arguments are coerced to a common type which is the type of the returned value, and all attributes except names are removed.
Usage
c(..., recursive = FALSE)
Arguments
... objects to be concatenated.
recursive logical. If recursive = TRUE, the function recursively descends through lists (and pairlists) combining all their elements into a vector.
Details
The output type is determined from the highest type of the components in the hierarchy NULL < raw < logical < integer < double < complex < character < list < expression. Pairlists are treated as lists, but non-vector components (such names and calls) are treated as one-element lists which cannot be unlisted even if recursive = TRUE.
c is sometimes used for its side effect of removing attributes except names, for example to turn an array into a vector. as.vector is a more intuitive way to do this, but also drops names. Note too that methods other than the default are not required to do this (and they will almost
```

On LONI and LSU HPC Clusters

- Two modes to run R on clusters
  - Interactive mode
    - Type `R` command to launch the console
    - Run R commands in the console
  - Batch mode
    - Write the R script first, then submit a batch job to run it (use the `Rscript` command)
    - This mode is better for production runs
- Clusters are better for resource-demanding jobs


mike5.hpc.lsu.edu - SuperMikeII - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
These are the keywords that are part of the software tree,
however, it is not suggested that you use these:
[ychen64@mike5 ~]$ soft add +R-3.3.3-gcc-4.7.2
[ychen64@mike5 ~]$ R
WARNING: ignoring environment value of R_HOME
R version 3.3.3 (2017-03-06) -- "Another Canoe"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
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.
Connect quickly without profiles
119x37

Outline

- R basics
  - What is R
  - How to run R codes
  - Basic syntax
  - R as a calculator
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis
  - Data acquisition and inspection
  - Data preprocessing
  - Statistical analysis
  - Report generation

Basic syntax

- The default R prompt is the greater-than sign (>)

```
> 2*4
[1] 8
> options(prompt="R>")
R>
```

- If a line is not syntactically complete, a continuation prompt (+) appears.

```
> 2*
+ 4
[1] 8
```

- Assignment operators are the left arrow (<−) and =. They both assign the value of the object on the right to the object on the left.

```
> x <- 2*4
```

- The contents of the object x can be viewed by typing value at the R prompt

```
> x
[1] 8
```

Basic syntax

- Last expression can be retrieved through an internal object `.Last.value`

```
> 2*4
[1] 8
> x <- .Last.value
> x
[1] 8
```

- Removing objects with the function `rm()`

```
> rm(x)
> x
Error: object 'value' not found
```

- Legal R Names
  - names for R objects can be any combination of letters, numbers and periods (.) but must not start with a number nor period
- **Note: R is case sensitive. X and x are different in R.**

```
> X
Error: object 'X' not found
```

Basic syntax

- Avoid assignment to built in functions
  - R has a number of built in functions e.g. `c, T, F, t`
  - An easy way to avoid this problem is to check the contents of the object you wish to use, this also stops you from overwriting the contents of a previously saved object

```
> X    # object with no value assigned
Error: object 'value' not found
> x     # object with a value assigned
[1] 8
> T      # Built in R value
[1] TRUE
> t      # Built in R function
function (x)
UseMethod("t")
```

- Spaces
  - R will ignore extra spaces between object names and operators

```
> x <- 2 * 4
[1] 8
```

  - Spaces cannot be placed between the < and - in the assignment operator

```
> x < - -2 * 4
[1] FALSE
```

R as a calculator

- Arithmetic operators and parentheses

```
>   (1+2)/(3*2)
>   [1] 0.5
```

- Power operator

```
> 2^3
[1] 8
> 4^0.5
[1] 2
> sqrt(4)
[1] 2
```

- Scientific notation

```
> 2.1e2
[1] 210
```

R as a calculator

- Exponential function

```
> exp(1); exp(0) # ; is the newline separate commands
[1] 2.718282
[1] 1
```

- Inf means "non-finite numeric value"

```
> x <- 1/0
> x
[1] Inf
> y <- -1/0
> y
[1] -Inf
```

- NaN means "not a number"

```
> x+y
[1] NaN
```

- pi

```
> pi
[1] 3.141593
> help(pi) # Get help from R. You can also use ?pi
```

CENTER FOR COMPUTATION & TECHNOLOGY

Outline

- R basics
  - What is R
  - How to run R codes
  - Basic syntax
  - R as a calculator
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis
  - Data acquisition and inspection
  - Data preprocessing
  - Statistical analysis
  - Report generation

Data Classes

- R has five atomic classes
  - **Two numeric classes** (integer or double)
    - Numbers in R are treated as numeric unless specified otherwise.

```
> x <- 605
> x
[1] 605
```

  - **Complex**

```
> cn <- 2 + 3i
> cn
[1] 2 + 3i
```

  - **Character**

```
> string <- “Hello World”
> string
[1] “Hello World”
```

  - **Logical**
    - TRUE or FALSE

```
> 2 < 4
[1] TRUE
```

Data Classes

- The function `class()` can be used to determine the class of each object

```
> class(x)
[1] "numeric"
> class(string)
[1] "character"
> class(cn)
[1] "complex"
```

- The code missing values in R is `NA`. The `is.<type>()` functions can be used to check for the data classes

```
> is.numeric(x)
[1] TURE
> is.character(string)
[1] TURE
> value <- NA
> is.na(value)
[1] TRUE
```

Data Objects

- R Data objects
  - **Vector**: elements of same class, one dimension
  - **Matrix**: elements of same class, two dimensions
  - **Array**: elements of same class, 2+ dimensions
  - **Lists**: elements can be any objects
  - **Data frames**: “datasets” where columns are variables and rows are observations

CENTER FOR COMPUTATION & TECHNOLOGY

Data Objects - Vectors

- Vectors can only contain elements of the same data class
- Vectors can be constructed by
  - Using the `c()` function (concatenate)

```
> d <- c(1,2,3)  ##numeric
> d <- c("1","2","3") ##character
> value.logical <- c(F,F,T)  ##logical
```

  - you can convert an object with `as.TYPE()` functions

```
> as.numeric(d)
```

  - Coercion will occur when mixed objects are passed to the `c()` function, as if the `as.<Type>()` function is explicitly called

```
> y <- c(1.7, "a") ## character
> y <- c(TRUE, 2) ## numeric
> y <- c("a", TRUE) ## character
```

Data Objects - Vectors

- Vectors can also be constructed by
  - Using the `vector()` function

```
> x <- vector("numeric", length = 10)
> x
 [1] 0 0 0 0 0 0 0 0 0 0
```

  - Using `seq()` or `rep()` function

```
> x <- 0:6
> x <- seq(from=2,to=10,by=2)
> x <- seq(from=2,to=10,length=5)
> x <- rep(5,6)
```

- Vectors can be created using a combination of these functions.

```
> value1 <- c(1,3,4,rep(3,4),seq(from=1,to=6,by=2))
> value2 <- rep(c(1,2),3)
> value3 <- rep(c(1,2),each=3)
```

Data Objects - Vectors

- NA in R means missing value

```
> weight <- c(60, 72, NA, 90, 95, 72)
> weight
[1] 60 72 NA 90 95 72
> height <- c(1.75,1.80,1.65,1.90,1.74,1.91)
```

- Vector based operations are very fast!

```
> bmi <- weight/height^2
> bmi
[1] 19.59184     22.22222     NA 24.93075      31.37799        19.73630
> mean(weight)
[1] NA
> mean(weight, na.rm=TRUE)
[1] 77.8
> sd(weight, na.rm=T)
[1] 14.39444
> median(weight, na.rm=T)
[1] 72
> round(height, d=1)
[1] 1.8 1.8 1.6 1.9 1.7 1.9
```

Vectors Indexing

- One can use `[<index>]` to access individual element of interest
  - Indices start from 1

```
> x <- 1:10
> x[4] ## individual element of a vector
> x[1,4] ## how about multiple elements?
Error in x[1,4] : incorrect number of dimensions
> x[c(1,4)] ## this is the correct way
[1] 1 4
> x[c(1,8:9,3)] ## not necessarily in order
[1] 1 8 9 3
> x[-1] ## negative indices drop elements
[1] 2 3 4 5 6 7 8 9 10
> x[-1:-5]
[1] 6 7 8 9 10
> x[c(T,T,T,T,T,F,F,F,F,F)] ## Can use logical values as indices
[1] 1 2 3 4 5
> x[c(T,F)] ## Use a pattern
[1] 1 3 5 7 9
```

Data Objects - Matrices

- Matrices are vectors with a dimension attribute
- R matrices can be constructed by
  - Using the `matrix()` function

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

- R matrices are constructed column-wise by default

```
> m <- matrix(1:12,nrow=3,ncol=4,byrow=F) ## is the same as x <- matrix(1:12,nrow=3,ncol=4)
> m <- matrix(1:12,nrow=3,ncol=4,byrow=T) ## try this one
```

Data Objects - Matrices

- R matrices can also be constructed by
  - Passing an `dim` attribute to a vector

```
> m <- 1:10
> m
 [1]  1  2  3  4  5  6  7  8  9 10
> dim(m) <- c(2, 5)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

  - Using `cbind()` or `rbind()` functions

```
> x <- 1:3
> y <- 10:12
> cbind(x, y)
x y
[1,] 1 10
[2,] 2 11
[3,] 3 12
> rbind(x, y)
[,1] [,2] [,3]
x 1 2 3
y 10 11 12
```

Data Objects – Arrays

- Elements of same class with a number of dimensions
  - Vectors and matrices are arrays of 1 and 2 dimensions
  - Function `array()` creates an array with given dimensions

```
> # An array with 8 elements and 3 dimensions
> m <- array(data = 1:8,dim = c(2,2,2))
```

Data Objects - Lists

- Lists are an ordered collection of objects (which can be of different types or classes and different lengths)
- Lists can be constructed by using the `list()` function

```
> x <- c(31, 32, 40)
> y <- factor(c("F", "M", "M", "F"))
> z <- c("London", "New York")
> my_list <- list(x,y,z)
> my_list
[[1]]
[1] 31 32 40

[[2]]
[1] F M M F
Levels: F M

[[3]]
[1] "London"   "New York"
```

Data Objects - Lists

- Elements of R objects can have names, `names()` function can display:

```
> names(my_list)
NULL
```

- Names can be assigned

```
> names(my_list) <- c("age","sex","city")
> names(my_list)
[1] "age"  "sex"  "city“
```

- Or can be assigned when creating a list.

```
> my_list2 <- list(age=x,sex=y,city=z)
> names(my_list2)
[1] "age"  "sex"  "city“
```

LSU CENTER FOR COMPUTATION & TECHNOLOGY

Lists Indexing

- Using two equivalent ways to access the first component (e.g. age in my_list):
  - the `[[]]` operator

```
> my_list[[1]]
[1] 31 32 40
```

  - the "$" sign if the elements of list have names

```
> my_list$age
[1] 31 32 40
```

- Referring individual element

```
> my_list$age[1]
[1] 31
```

Data Objects - Data Frames

- Data frames are used to store tabular data
  - They are a special type of lists where every element (i.e. column) has to be of **the same length**, but can be of different class
  - Data frames can have special attributes such as `row.names`
  - Data frames can be created by reading data files, using functions such as `read.table()` or `read.csv()`
    - More on this later

Data Objects - Data Frames

- Data frames can be created directly by calling `data.frame()`

```
> my_df <- data.frame(age=c(31,40,50), sex=c("M","F","M"))
> my_df
  age sex
1  31   M
2  40   F
3  50   M
```

- Why do we need data frames if it is simply a list? - More efficient storage, and indexing!

Matrices and Dataframes Indexing

- One can use `[<index>,<index>]` to access individual element

```
> my_df[1,2]
[1] M
```

- Indexing by columns

```
> my_df[,1]
[1] 31 40 50
> my_df[,1:2]
  age sex
1  31   M
2  40   F
3  50   M
```

- Indexing by rows

```
> my_df[1,]
  age sex
1  31   M
> my_df[2:3,]
  age sex
2  40   F
3  50   M
```

Matrices and Dataframes Indexing

- the "$" sign if the elements of matrix/dataframe have names

```
> my_df$sex
[1] M F M
Levels: F M
> my_df$sex[2] ## Referring individual element

[1] F
Levels: F M
```

- the `[[]]` operator

```
> my_df[[1]]
[1] 31 40 50
> my_df[[1]][1]
[1] 31
> my_df[[3]][1]
Error in .subset2(x, i, exact = exact) : subscript out of bounds
```

Matrices and Dataframes Indexing

- Indexing can be conditional on another variable!

```
> pain <- c(0, 3, 2, 2, 1)
> sex <- factor(c("M", "M", "F", "F", "M"))
> age <- c(45, 51, 45, 32, 90)
> which(sex=="M")
[1] 1 2 5
> pain[sex=="M"]
[1] 0 3 1
> pain[age>32]
[1] 0 3 2 1
> pain[(age>32)&(sex=="M")]
[1] 0 3 1
> pain[(age>=49)|(age<41)]
[1] 3 2 1
> my_df
  age sex
1  31   M
2  40   F
3  50   M
> my_df$age[my_df$sex=="M"]
[1] 31 50
```

Querying Object Attributes

- The `length()` function
- The `class()` function
- The `dim()` function
- The `str()` function
- The `attributes()` function reveals attributes of an object
  - Class
  - Names
  - Dimensions
  - Length
  - User defined attributes
- They work on all objects (including functions)

Outline

- R basics
  - What is R
  - How to run R codes
  - Basic syntax
  - R as a calculator
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis
  - Data acquisition and inspection
  - Data preprocessing
  - Statistical analysis
  - Report generation

Flow Control Structures

- Control structures allow one to control the flow of execution.
  - Similar to other script languages

`if … else`	testing a condition
`for`	executing a loop (with fixed number of iterations)
`while`	executing a loop when a condition is true
`repeat`	executing an infinite loop
`break`	breaking the execution of a loop
`next`	skipping to next iteration
`return`	exit a function

Testing conditions

Comparisons: <, <=, >, >=, ==, !=
Logical operations:
!: NOT
&: AND (elementwise)
&&: AND (only leftmost element)
|: OR (element wise)
||: OR (only leftmost element)

```
> x <- 10
> if(x > 3 && x < 5) {
+   print ("x is between 3 and 5")
+ } else if(x <= 3) {
+   print ("x is less or equal to 3")
+ } else {
+   print ("x is greater or equal to 5")
+ }
[1] "x is greater or equal to 5"
```

For Loops

```
# Syntax
# for (value in sequence) {
#   statements
# }
```

Example

```
> x <- c(2,5,3,9,8,11,6)
> count <- 0
> for (i in x) {
+   if (i %% 2 == 0) count <- count+1
+ }
> count
[1] 3
```

Loops are not very frequent used because of many inherently vectorized operations and the family of `apply()` functions (more on this later)

Outline

- R basics
  - What is R
  - How to run R codes
  - Basic syntax
  - R as a calculator
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis
  - Data acquisition and inspection
  - Data preprocessing
  - Statistical analysis
  - Report generation

Simple Statistic Functions

`min()`	Minimum value
`max()`	Maximum value
`which.min()`	Location of minimum value
`which.max()`	Location of maximum value
`sum()`	Sum of the elements of a vector
`mean()`	Mean of the elements of a vector
`sd()`	Standard deviation of the elements of a vector
`quantile()`	Show quantiles of a vector
`summary()`	Display descriptive statistics

```
> mean(weight,na.rm=T)
[1] 77.8
> which.min(weight)
[1] 1
> min(weight,na.rm=T)
[1] 60
>
```

Distributions and Random Variables

- For each distribution R provides four functions: density (d), cumulative density (p), quantile (q), and random generation (r)
  - The function name is of the form `[d|p|q|r]<name of distribution>`
  - e.g. `qbinom()` gives the quantile of a binomial distribution

Distribution	Distribution name in R
Uniform	`unif`
Binomial	`binom`
Poisson	`pois`
Geometric	`geom`
Gamma	`gamma`
Normal	`norm`
Log Normal	`lnorm`
Exponential	`exp`
Student's t	`t`

Distributions and Random Variables

- Generating random number from normal distribution

```
> set.seed(1)
> rnorm(2,mean=0,sd=1)
[1] -0.6264538  0.1836433

> pnorm(1.96)
[1] 0.9750021
```

- The inverse of the above function call

```
> qnorm(0.975)
[1] 1.959964
```

Sorting and random samples

- Sort and order elements: `sort()`, `rank()` and `order()`.

```
> x <- c(1.2,0.4,2.3,0.9)
> sort(x)  ## sort x in ascending order
> sort(x,decreasing=T) ## sort x in descending order
> rank(x)
[1] 3 1 4 2
> order(x)
[1] 2 4 1 3
```

- Random sampling function `sample()`.

```
> sample(1:4,4,replace=F)
> sample(1:10,10,replace=F)
> sample(1:10,10,replace=T)  ## will be different from the last run
> sample(1:4,10,replace=T,prob=c(.2,.5,.2,.1))
```

- Using the same seed value through `set.seed()` can reproduce the same outcome.

```
> set.seed(1)
> sample(1:4,10,replace=T)
 [1] 2 2 3 4 1 4 4 3 3 1
> set.seed(1)
> sample(1:4,10,replace=T)
 [1] 2 2 3 4 1 4 4 3 3 1
```

The table Function

- The `table()` function is useful to tabulate factors or find the frequency of an object
- Example: The quine dataset consists of 146 rows describing children's ethnicity (Eth), age (Age), sex (Sex), days absent from school (Days) and their learning ability (Lrn).
  - If we want to find out the frequency of the age classes in quine dataset

```
> library(MASS)
> table(quine$Age)
F0 F1 F2 F3
27 46 40 33
```

  - If we need to know the breakdown of ages according to sex

```
> table(quine$Sex,quine$Age)

    F0 F1 F2 F3
  F 10 32 19 19
  M 17 14 21 14
```

The apply Function

- The `apply()` function evaluate a function over the margins of an array
  - More concise than the `for` loops (not necessarily faster)

X: array objects
MARGIN: a vector giving the subscripts which the function will be applied over
FUN: a function to be applied

```
> str(apply)
function (X, 2, FUN, ...)
```

```
> x <- matrix(rnorm(200), 20, 10)
# Row means
> apply(x, 1, mean)
 [1] -0.23457304  0.36702942 -0.29057632 -0.24516988 -0.02845449  0.38583231
 [7]  0.16124103 -0.10164565  0.02261840 -0.52110832 -0.10415452  0.40272211
[13]  0.14556279 -0.58283197 -0.16267073  0.16245682 -0.28675615 -0.21147184
[19]  0.30415344  0.35131224

# Column sums
> apply(x, 2, sum)
 [1]  2.866834  2.110785 -2.123740 -1.222108 -5.461704 -5.447811 -4.299182
 [8] -7.696728  7.370928  9.237883

# 25th and 75th Quantiles for rows
> apply(x, 1, quantile, probs = c(0.25, 0.75))
           [,1]       [,2]       [,3]       [,4]      [,5]       [,6]
25% -0.52753974 -0.1084101 -1.1327258 -0.9473914 -1.176299 -0.4790660
75%  0.05962769  0.6818734  0.7354684  0.5547772  1.066931  0.6359116
          [,7]       [,8]       [,9]       [,10]      [,11]      [,12]
25% -0.1968380 -0.5063218 -0.8846155 -1.54558614 -0.8847892 -0.2001400
75%  0.7910642  0.3893138  0.8881821 -0.06074355  0.5042554  0.9384258
         [,13]       [,14]      [,15]      [,16]      [,17]      [,18]
25% -0.5378145 -1.08873676 -0.5566373 -0.3189407 -0.6280269 -0.6979439
75%  0.6438305 -0.02031298  0.3495564  0.3391990 -0.1151416  0.2936645
        [,19]      [,20]
25% -0.259203 -0.1798460
75%  1.081322  0.8306676
```

Other Apply Functions

- `lapply` - Loop over a list and evaluate a function on each element
- `sapply` - Same as `lapply` but try to simplify the result
- `tapply` - Apply a function over subsets of a vector
- `mapply` - Multivariate version of `lapply`

User Defined Functions

- Similar to other languages, functions in r are defined by using the `function()` directives
- The return value is the last expression in the function body to be evaluated
- Functions can be nested
- Functions are R objects
  - For example, they can be passed as an argument to other functions

CENTER FOR COMPUTATION & TECHNOLOGY

Example of User Defined Function

Syntax
function_name <- function (arguments) {
statement
}
#
Define the function for the power calculation

```
> pow <- function(x, y) {
+    result <- x^y
+}
```

Call the function

```
> c <- pow(4,2)
> c
[1] 16
```

Outline

- R basics
  - What is R
  - How to run R codes
  - Basic syntax
  - R as a calculator
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis
  - Data acquisition and inspection
  - Data preprocessing
  - Statistical analysis
  - Report generation

CENTER FOR COMPUTATION & TECHNOLOGY

Installing and Loading R Packages - PC

- Installation:
  - Option 1: menu item
  - Option 2: run `install.packages("<package name>")` function in the console
- Loading: the `library(<package name>)` function load previously installed packages

Installing R and R Packages - Clusters

- Installation
  - You most likely do NOT have root privilege, so you need to
  - Point the environment variable `R_LIBS_USER` to desired location, then
  - Use the `install.packages("<package name>")` function
- Loading: the `library(<package name>)` function load previously installed packages
- Documentation page: http://www.hpc.lsu.edu/docs/faq/installation-details.php

```
[ychen64@mike2 ~]$ export R_LIBS_USER=/home/ychen64/packages/R/libraries
[ychen64@mike2 ~]$ echo $R_LIBS_USER
/home/ychen64/packages/R/libraries
[ychen64@mike2 ~]$ R

R version 3.3.3 (2017-03-06) -- "Another Canoe"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
…

> install.packages("swirl")
```

Outline

- R basics
  - What is R
  - How to run R codes
  - Basic syntax
  - R as a calculator
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis
  - Data acquisition and inspection
  - Data preprocessing
  - Statistical analysis
  - Report generation

Steps for Data Analysis

- Get the data
- Read and inspect the data
- Preprocess the data (remove missing and dubious values, discard columns not needed etc.)
- Analyze the data
- Generate the report

How does R work

- R works best if you have a dedicated folder for each separate project - the working folder. Put all data files in the working folder (or in subfolders).

```
> getwd()  #Show current directory
[1] "/home/ychen64"
> dir.create("data") #Create a new directory
> getwd()
[1] "/home/ychen64"
> setwd("data")
> getwd()
[1] "/home/ychen64/data"
```

- Work on the project - your objects can be automatically saved in the .RData file
- To quit use `q()` or just kill the window. R will automatically ask you “Save workspace image?”. You can choose:
  – No: leave R without saving your results in R;
  – Yes: save your results in .RData in your working directory;
  – Cancel: not quitting R.

Case Study: Forbes Fortune List

- The forbes dataset consists of 2000 rows (observations) describing companies' rank, name, country, category, sales, profits, assets and market value.

Getting Data

- Downloading files from internet
  - Manually download the file to the working directory
  - or with R function `download.file()`

```
> download.file("http://www.hpc.lsu.edu/training/weekly-materials/Downloads/Forbes2000.csv.zip", "Forbes2000.csv.zip")
> unzip("Forbes2000.csv.zip","Forbes2000.csv")
```

Reading and Writing Data

- R understands many different data formats and has lots of ways of reading/writing them (csv, xml, excel, sql, json etc.)

`read.table` `read.csv`	`write.table` `write.csv`	for reading/writing tabular data
`readLines`	`writeLines`	for reading/writing lines of a text file
`source`	`dump`	for reading/writing in R code files
`dget`	`dput`	for reading/writing in R code files
`load`	`save`	for reading in/saving workspaces

Reading Data with `read.table` (1)

```
# List of arguments of the read.table() function
> str(read.table)
function (file, header = FALSE, sep = "", quote = "\"'", dec = ".", row.names,
col.names, as.is = !stringsAsFactors, na.strings = "NA", colClasses = NA, nrows = -1,
skip = 0, check.names = TRUE, fill = !blank.lines.skip, strip.white = FALSE,
blank.lines.skip = TRUE, comment.char = "#", allowEscapes = FALSE, flush = FALSE,
stringsAsFactors = default.stringsAsFactors(), fileEncoding = "", encoding = "unknown",
text, skipNul = FALSE)
```

Reading Data with `read.table` (2)

- `file` - the name of a file, or a connection
- `header` - logical indicating if the file has a header line
- `sep` - a string indicating how the columns are separated
- `na.strings` - a character vector of strings which are to be interpreted as `NA` values
- `nrows` - the number of rows in the dataset
- `comment.char` - a character string indicating the comment character
- `skip` - the number of lines to skip from the beginning
- `stringsAsFactors` - should character variables be coded as factors?

Reading Data with `read.table` (3)

- The function will
  - Skip lines that begin with #
  - Figure out how many rows there are (and how much memory needs to be allocated)
  - Figure out what type of variable is in each column of the table
- Telling R all these things directly makes R run faster and more efficiently.
- `read.csv()` is identical to `read.table()` except that the default separator is a comma.

```
> forbes <- read.csv("Forbes2000.csv",header=T,stringsAsFactors =
FALSE,na.strings ="NA",sep=",")
```

CENTER FOR COMPUTATION & TECHNOLOGY

Reading EXCEL spreadsheets

- The XLConnect library can open both .xls and .xlsx files. It is Java-based, so it is cross platform. But it may be very slow for loading large datasets.

```
>library(XLConnect)
wb <- loadWorkbook("Forbes2000.xls")
setMissingValue(wb, value = c("NA"))
forbes <- readWorksheet(wb, sheet=1, header=TRUE)>dim(forbes)
[1] 2000    8
```

- There are at least two other ways: read.xlsx from library(xlsx) (slow for large datasets) and read.xls from library(gdata) (require PERL installed).

```
>library(xlsx)
>forbes <- read.xlsx("Forbes2000.xls", 1)
```

- Note: the libraries above requires both Java Dev Kit and rJava library. The former is not available on SuperMike2, while the later is not available for R version on QB2 and SuperMic.

Inspecting Data (1)

- `head()`: print the first part of an object
- `tail()`: print the last part of an object

```
> head(forbes)
  rank                name        country             category  sales profits
1    1           Citigroup  United States              Banking  94.71   17.85
2    2    General Electric  United States        Conglomerates 134.19   15.59
3    3 American Intl Group  United States            Insurance  76.66    6.46
4    4          ExxonMobil  United States Oil & gas operations 222.88   20.96
5    5                  BP United Kingdom Oil & gas operations 232.57   10.27
6    6     Bank of America  United States              Banking  49.01   10.81
   assets marketvalue
1 1264.03      255.30
2  626.93      328.54
3  647.66      194.87
4  166.99      277.02
5  177.57      173.54
6  736.45      117.55
```

Inspecting Data (2)

- Summary of the “forbes” dataframe.

```
> str(forbes)
'data.frame':   2000 obs. of  8 variables:
 $ rank       : num  1 2 3 4 5 6 7 8 9 10 ...
 $ name       : chr  "Citigroup" "General Electric" "American Intl Group" "ExxonMobil" ...
 $ country    : chr  "United States" "United States" "United States" "United States" ...
 $ category   : chr  "Banking" "Conglomerates" "Insurance" "Oil & gas operations" ...
 $ sales      : num  94.7 134.2 76.7 222.9 232.6 ...
 $ profits    : num  17.85 15.59 6.46 20.96 10.27 ...
 $ assets     : num  1264 627 648 167 178 ...
 $ marketvalue: num  255 329 195 277 174 ...
```

Inspecting Data (3)

- Statistical summary of the “Forbes” dataframe.

```
> summary(forbes)
      rank             name             country            category
 Min.   :   1.0   Length:2000        Length:2000        Length:2000
 1st Qu.: 500.8   Class :character   Class :character   Class :character
 Median :1000.5   Mode  :character   Mode  :character   Mode  :character
 Mean   :1000.5
 3rd Qu.:1500.2
 Max.   :2000.0

     sales             profits             assets            marketvalue
 Min.   :  0.010   Min.   :-25.8300   Min.   :   0.270   Min.   :  0.02
 1st Qu.:  2.018   1st Qu.:  0.0800   1st Qu.:   4.025   1st Qu.:  2.72
 Median :  4.365   Median :  0.2000   Median :   9.345   Median :  5.15
 Mean   :  9.697   Mean   :  0.3811   Mean   :  34.042   Mean   : 11.88
 3rd Qu.:  9.547   3rd Qu.:  0.4400   3rd Qu.:  22.793   3rd Qu.: 10.60
 Max.   :256.330   Max.   : 20.9600   Max.   :1264.030   Max.   :328.54
                   NA's   :5
```

- There are missing values in the profits category.

Outline

- R basics
  - What is R
  - How to run R codes
  - Basic syntax
  - R as a calculator
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis
  - Data acquisition and inspection
  - Data preprocessing
  - Statistical analysis
  - Report generation

Preprocessing - Missing Values

- Missing values are denoted in `R` by `NA` or `NaN` for undefined mathematical operations.
  - `is.na()` is used to test objects if they are NA
- Make sure when reading data `R` can recognize the missing values. E.g. `setMissingValue(wb, value = c("NA"))` when using `XLConnect`
- Many R functions also have a logical "`na.rm`" option
  - `na.rm=TRUE` means the NA values should be discarded

```
mean(weight,na.rm=T)
```

- **Note: Not all missing values are marked with "NA" in raw data!**

Preprocessing - Missing Values

- There are many statistical techniques that can deal with the missing values, but the simplest way is removing them.
  - If a row (observation) has a missing value, remove the row with `na.omit()`. e.g.

```
> forbes <- na.omit(forbes)
> dim(forbes)
```

  - If a column (variable) has a high percentage of the missing value, remove the whole column or just don't use it for the analysis

Preprocessing - Subsetting Data (1)

- At most occasions we do not need all of the raw data
- There are a number of methods of extracting a subset of R objects
- Subsetting data can be done either by row or by column

Preprocessing - Subsetting Data (2)

- Subsetting by row: use conditions

```
# Find all companies with negative profit
>forbes[forbes$profits < 0,c("name","sales","profits","assets")]
                          name sales profits  assets
350          Allianz Worldwide 96.88   -1.23  851.24
354                   Vodafone 47.99  -15.51  256.28
364           Deutsche Telekom 56.40  -25.83  132.01
```

Preprocessing - Subsetting Data (3)

- Subsetting by row: use the `subset()` function

```
# Find the business category to which most of the
Bermuda island companies belong.
```

```
>Bermudacomp <- subset(forbes, country == "Bermuda")
>table(Bermudacomp[,"category"]) #frequency table of categories

          Banking       Capital goods       Conglomerates
                1                   1                   2
Food drink & tobacco     Food markets           Insurance
                1                   1                  10
            Media Oil & gas operations  Software & services
                1                   2                   1
```

CENTER FOR COMPUTATION & TECHNOLOGY

Preprocessing - Subsetting Data (4)

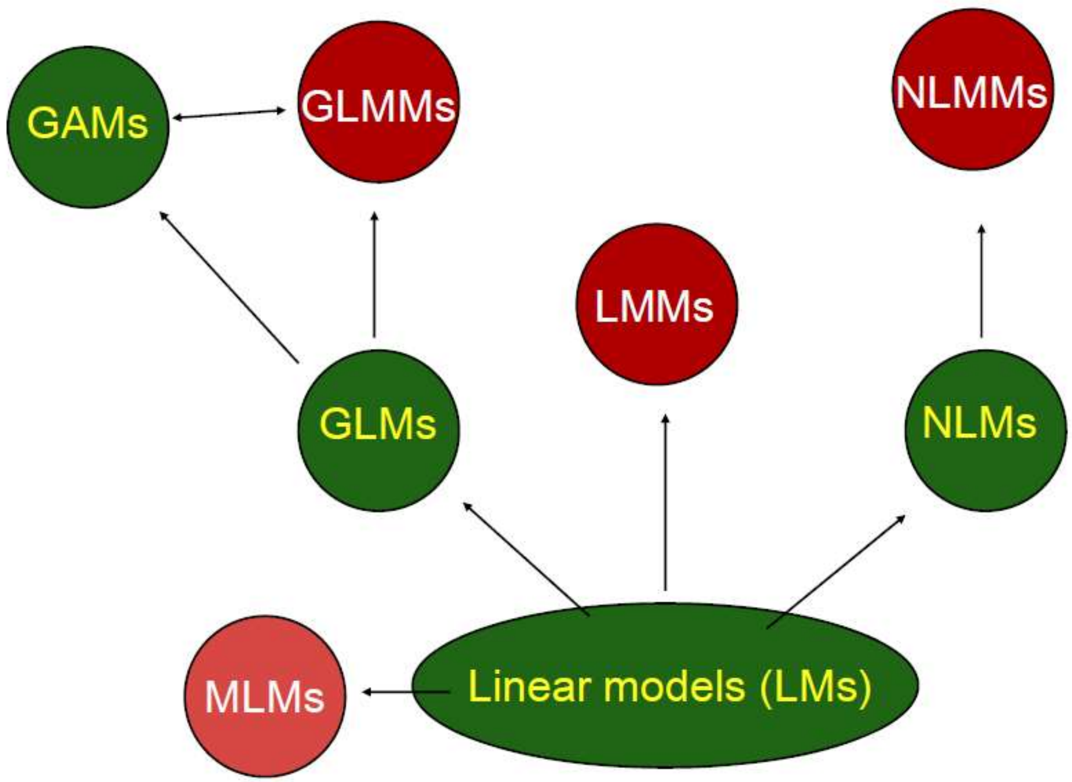
- Subsetting by column

```
# Create another data frame with only numeric
variables

forbes2 <- data.frame(sales=forbes$sale,profits=forbes$profits,
           assets=forbes$assets, mvalue=forbes$marketvalue)
str(forbes2)

# Or simply use indexing
forbes3 <- forbes[,c(5:8)]
str(forbes3)
```

Outline

- R basics
  - What is R
  - How to run R codes
  - Basic syntax
  - R as a calculator
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis
  - Data acquisition and inspection
  - Data preprocessing
  - Statistical analysis
  - Report generation

Roadmap of generalizations of linear models



Explanation of Acronyms

Models	Acronym	R function
Linear Models	LM	lm, aov
MultivariateLMs	MLM	manova
Generalized LMs	GLM	glm
Linear Mixed Models	LMM	lme, aov
Non-linear Models	NLM	nls
Non-linear Mixed Models	NLMM	nlme
Generalized LMMs	GLMM	glmmPQL
Generalized Additive Models	GAM	gam

Symbol Meanings in Model Formulae

<table>
<tr><th>Symbol</th><th>Example</th><th>Meaning</th></tr>
<tr><td>+</td><td>+X</td><td>Include this variable in the model</td></tr>
<tr><td>-</td><td>-X</td><td>Exclude this variable in the model</td></tr>
<tr><td>:</td><td>X:Z</td><td>Include the interaction between X and Z</td></tr>
<tr><td>*</td><td>X*Z</td><td>Include X and Z and the interactions</td></tr>
<tr><td>|</td><td>X|Z</td><td>Conditioning: include X given Z</td></tr>
<tr><td>^</td><td>(A+B+C)^3</td><td>Include A, B and C and all the interactions up to three way</td></tr>
<tr><td>/</td><td>/(X*Z)</td><td>As is: include a new variable consisting of these variables multiplied</td></tr>
</table>

Model Formulae

General form: response ~ $term_1$ + $term_2$

Example	Meaning
y ~ x	Simple regression
y ~ -1 + x	LM through the origin
y ~ x + x^2	Quadratic regression
y ~ x1 + x2 + x3	Multiple regression
y ~ .	All variables included
y ~ . - x1	All variables except X1
y ~ A + B + A : B	Add interaction
y ~ A * B	Same above
y ~ (A+B)^2	Same above

A Linear Regression Example

market value ~ profits + sales + assets

```
> fit1 <- lm(mvalue ~ ., data=forbes2[1:1500,])
> summary(fit1)
Call:
lm(formula = mvalue ~ ., data = forbes2[1:1500, ])

Residuals:
     Min       1Q   Median       3Q      Max
-119.475   -5.186   -2.514    0.826  224.474

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.878870   0.576865   6.724 2.51e-11 ***
sales       0.560050   0.028367  19.743  < 2e-16 ***
profits     4.606250   0.265004  17.382  < 2e-16 ***
assets      0.047932   0.004734  10.125  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 18.98 on 1496 degrees of freedom
Multiple R-squared:  0.5251,    Adjusted R-squared:  0.5242
F-statistic: 551.4 on 3 and 1496 DF,  p-value: < 2.2e-16
```

Put Everything Together

- Run R commands in batch mode with `Rscript`

```
[ychen64@mike001 R]$ cat forbes.R
# Check if the data directory exists; if not, create it.
if (!file.exists("data")) {
        dir.create("data")
}

# Check if the data file has been downloaded; if not, download it.
if (!file.exists("Forbes2000.csv")) {
        download.file("http://www.hpc.lsu.edu/training/weekly-
materials/Downloads/Forbes2000.csv.zip", "Forbes2000.csv.zip")
}
…

[ychen64@make001 R]$ Rscript forbes.R
```

Outline

- R basics
  - What is R
  - How to run R codes
  - Basic syntax
  - R as a calculator
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis
  - Data acquisition and inspection
  - Data preprocessing
  - Statistical analysis
  - Report generation

Report Generation with R Markdown

- R markdown
  – Allows one to generate dynamic report by weaving R code and human readable texts together
- The `knitr` and `rmarkdown` packages can convert them into documents of various formats
- Help make your research reproducible

Not Covered

- Statistical analysis (e.g regression models, machine learning/data mining)
- Advanced missing data treatment
- Advanced data manipulation
- Categorical data (factor)
- Graphics in R
- Parallel Processing in R

Learning R

- User documentation on CRAN
  - An Introduction on R: http://cran.r-project.org/doc/manuals/r-release/R-intro.html
- Online tutorials (tons of them)
  - http://www.cyclismo.org/tutorial/R/
- Online courses (e.g. Coursera)
- Educational R packages
  - Swirl: Learn R in R

Next Tutorial – Introduction to R Graphics

- This training will provide an introduction to the R graphics in detail
- Date: Oct 11th, 2017

More R Tutorial – Parallel Computing with R

- This training will help you take advantage of the processing power of HPC clusters, computer programs need to be able to run in parallel.
- How to use the "parallel" package in R and a few related packages to parallelize and enhance the performance of R programs
- Date: Oct 25th, 2017

Getting Help

- User Guides
  - LSU HPC: http://www.hpc.lsu.edu/docs/guides.php#hpc
  - LONI:http://www.hpc.lsu.edu/docs/guides.php#loni
- Documentation: http://www.hpc.lsu.edu/docs
- Contact us
  - Email ticket system: sys-help@loni.org
  - Telephone Help Desk: 225-578-0900

Questions?

Exercises 1

1. Create a vector of the positive odd integers less than 100 (Hint: use seq function).
2. Remove the values greater than 60 and less than 80.
3. Create a data frame called cone with two elements:

R <- c(2.27, 1.98, 1.69, 1.88, 1.64, 2.14)

H <- c(8.28, 8.04, 9.06, 8.70, 7.58, 8.34)

Recall the volume of a cone with radius R and height H is given by $\frac{1}{3}\pi R^2 H$. Make the third element as V, which is the volume of the cone.

Exercises 2

1. Import dataset forbes, save it as forbes
2. Run the following commands:

```
head(forbes)
str(forbes)
summary(forbes)
```

3. Remove the observations with missing values
4. Find all German companies with negative profit
5. Find the 50 companies in the Forbes dataset with the highest profit
6. Find the average value of sales for the companies in each country (Hint: use tapply function)
7. Find the number of companies in each country with profits above 5 billion US dollars