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

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 first stable version R 1.0.0 was released in 2000
- Latest stable version is 4.0.3 released on Oct 10, 2020

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 16,000 packages available on CRAN (as of October 2020)
    - 6,000 five 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

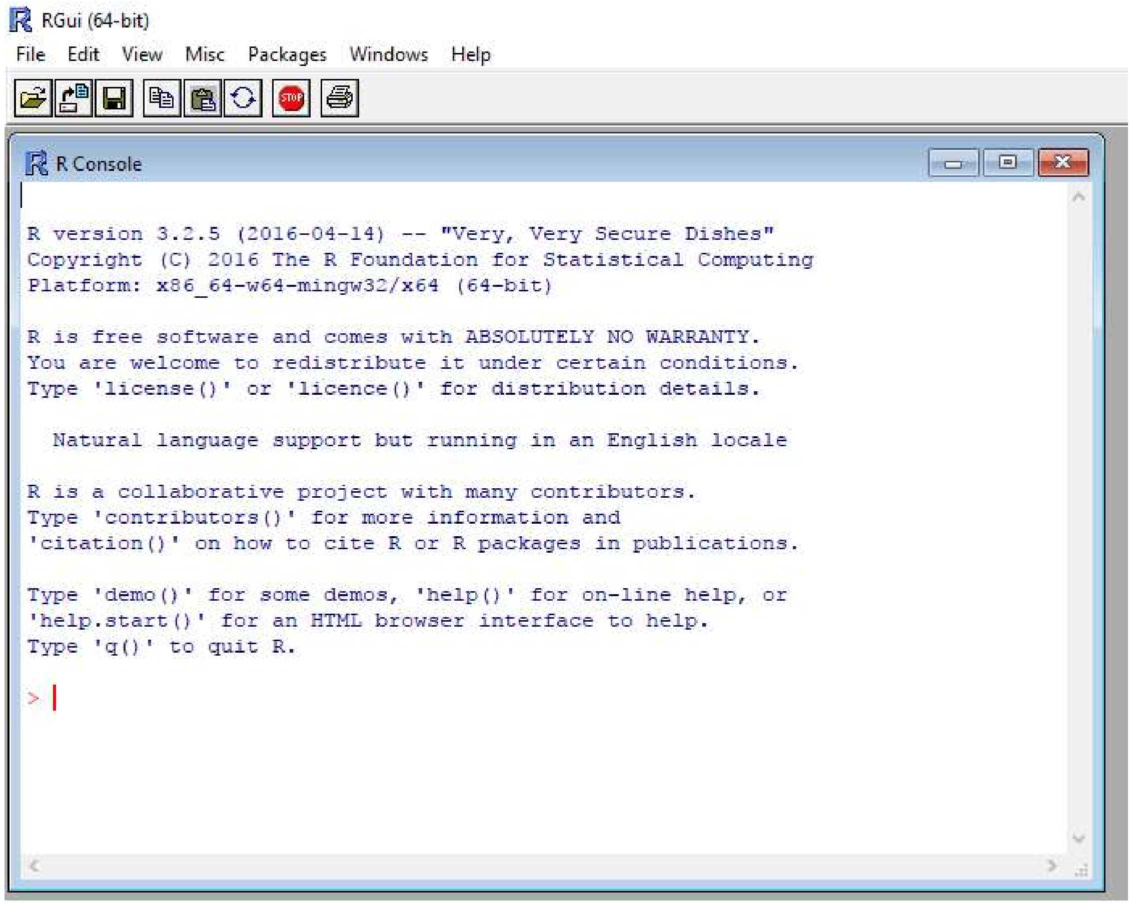
Installing and Loading R

- 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/3.5.3/INTEL-18.0.1`
    - QB3:`r/3.6.2/intel-19.0.5`
    - SuperMIC: `r/3.5.3/INTEL-18.0.1`
    - SuperMike2: `r/3.5.3/INTEL-18.0.0`
  - User requested R
    - Usually installed in user home directory

R Console

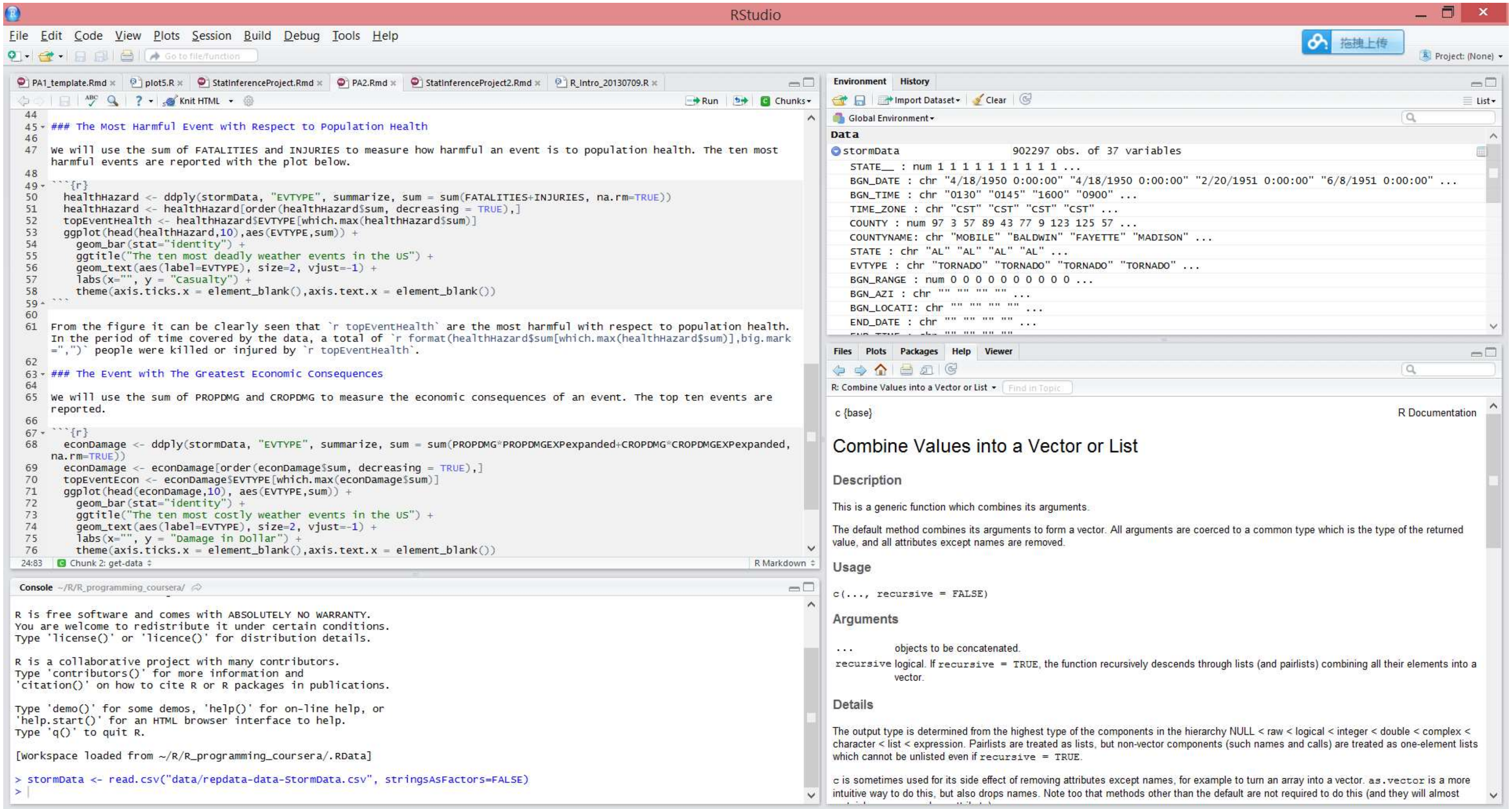
- Linux/Mac/Windows version available
- Limited graphic user interface (GUI)
- Command line interface (CLI) is similar to HPC environment

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

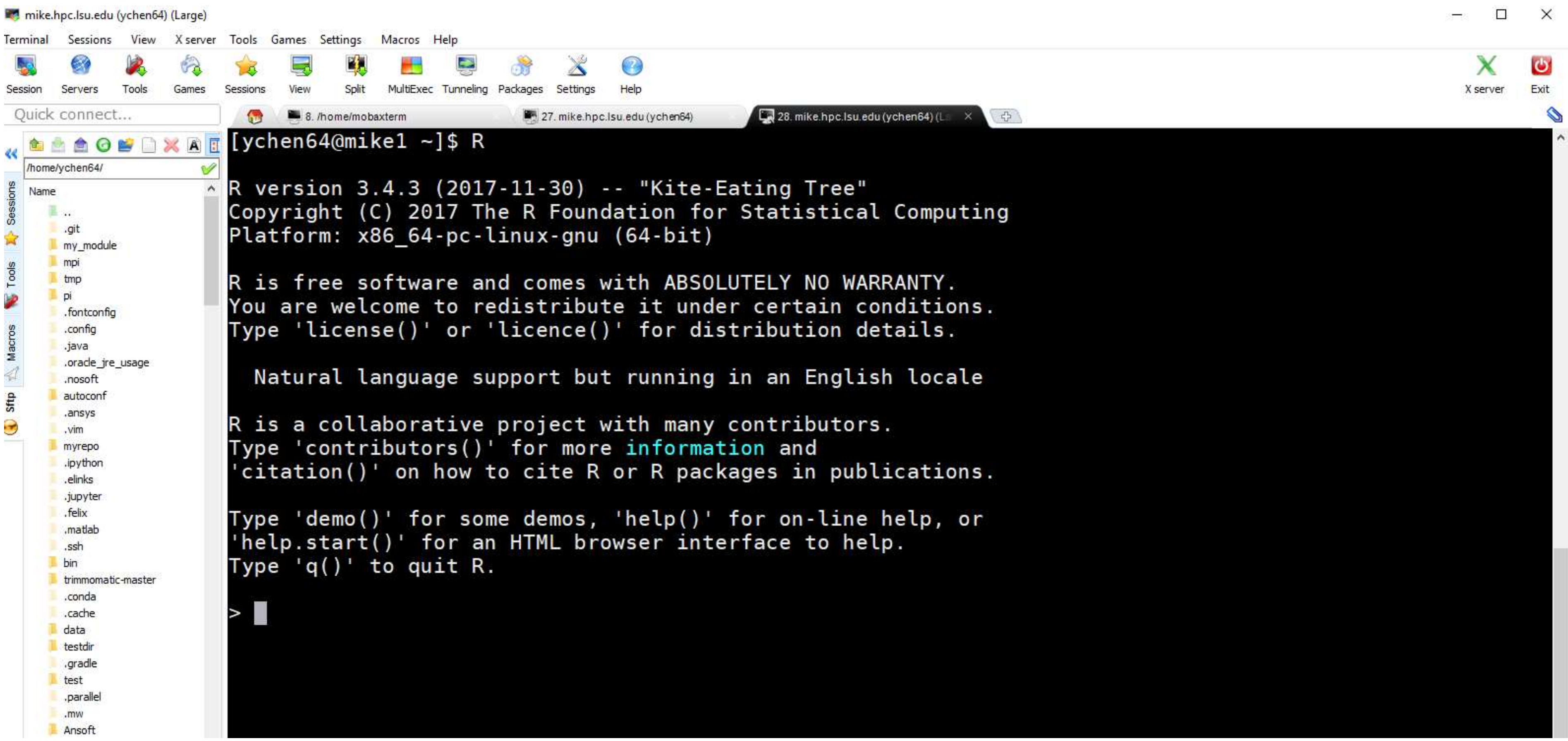
- Similar graphic user interface (GUI) to other Windows software, dividing the screen into panes
  - Source code
  - Console
  - Workspace
  - Others (help message, plot etc.)
- Rstudio in a desktop environment is better suited for development and/or a limited number of small jobs


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
Knit HTML Run Chunks
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())
24:83 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 List
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

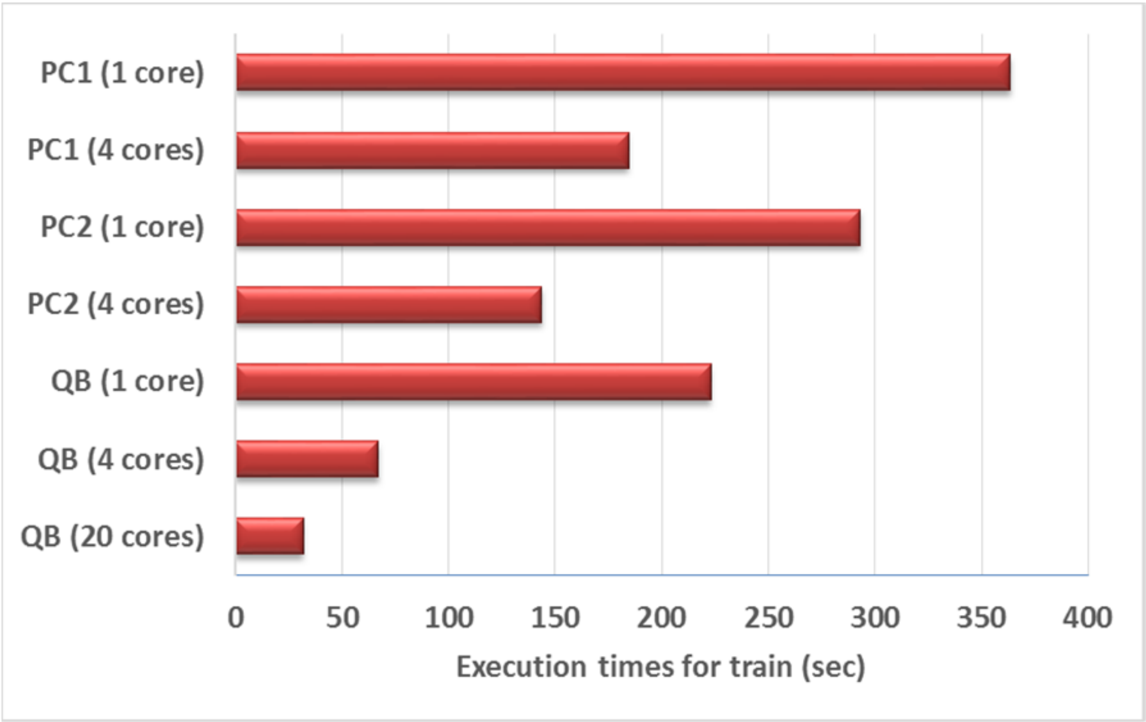
- Two modes to run R on clusters
  - Interactive mode
    - Type `R` command to launch the console
    - Run R commands in the console
  - Batch mode
    - Write the R script first, then submit a batch job to run it (use the `Rscript` command)
    - This mode is better for production runs

On LONI and LSU HPC Clusters



Clusters are Better for Resource-demanding Jobs

Training random forest model
Resampling method: 10-fold cross-validation



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

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 <- 8
> X
Error: object 'X' not found
```

Basic Syntax

- Function to clear the console in R and Rstudio

```
> cat("\014")
```

- The code above is the same as `CTRL + L`
- The saved object or function will not be affected

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

- In R, any line starting with “#” will be interpreted as a comment

```
> # z <- 2*4
> # Nothing will happen
```

Basic Syntax

- R allows automatic completion of type function or object name via the TAB key
  - Convenient, also error-proof
  - If there is no unique name, all matching names will show
- R allows using the up arrow key "↑" -- the previous command you entered shows up on the command line

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
```

R as a Calculator

- Comparisons: <, <=, >, >=, ==, !=

```
> 1 > 2
> 1 !=2
```

- Logical operations
  - `NOT: !`
  - `AND (element wise): &`
  - `OR(element wise): |`

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 Classes

- R has five atomic classes
  - **Two numeric classes** (numeric (double) or special type integer)
    - 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, FALSE or NA. The code missing values in R is NA.

```
> 2 < 4
[1] TRUE
> value <- NA
> is.logical(value)
```

CENTER FOR COMPUTATION & TECHNOLOGY

Data Classes

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

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

- The `is.<type>()` functions check the data classes

```
> is.numeric(x)
[1] TURE
> is.character(string)
```

- The `as.<type>()` funtions convert an object to a different type

```
> as.character(x)
```

CENTER FOR COMPUTATION & TECHNOLOGY

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

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

  - Using “:” operator

```
> x <- 0:6
```

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

```
> value1 <- c(1,2:4,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)   # unit is kg, contents after the # sign are comments
> weight
[1] 60 72 NA 90 95 72
> height <- c(1.75,1.80,1.65,1.90,1.74,1.91)   # unit: meter
```

- Vector based operations are very fast!

```
> bmi <- weight/height^2   # bmi stands for body mass index
> 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
```

CENTER FOR COMPUTATION & TECHNOLOGY

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
```

Exercises 1

1. Create a vector of the positive odd integers less than 100, save it to “x” (Hint: use seq function).
2. Only access the values less than 18 in x.

Exercises 1 - solution

1. `x <- seq(from=1,to=100,by=2)`
2. `x[c(1:9)]`

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))
>m
, , 1

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

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

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“
```

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" or "variable") has to be of **the same length**, but can be of different class
  - Why do we need data frames if it is simply a list? - More efficient storage, and indexing!
  - Data frames can have special attributes such as `row.names`
  - Data frames can be created by
    - combining elements with the same length using `data.frame()` functions
    - reading data files, using functions such as `read.table()` or `read.csv()`

LSU CENTER FOR COMPUTATION & TECHNOLOGY

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
```

- Column names can be assigned

```
> names(my_df) <- c("c1","c2")
> my_df
  c1 c2
1 31  M
2 40  F
3 50  M
```

Data Objects - Data Frames

- Row names are automatically assigned and are by default labelled “1”, “2”, “3”, …

```
> row.names(my_df)
[1] "1" "2" "3"
```

- These can also be renamed if desired

```
> row.names(my_df)<-c("r1","r2","r3")
> my_df
   c1 c2
r1 31  M
r2 40  F
r3 50  M
```

LSU CENTER FOR COMPUTATION & TECHNOLOGY

Matrices and Data Frames 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 Data Frames 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 Data Frames Indexing

- Indexing can be conditional on the variable itself or 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)
- More examples in the “Data inspection” section

Exercises 2

1. In the Exercises 1, a vector x <- seq(from=1,to=100,by=2) has been created. What if we want to exclude the values greater than 40 and less than 80?

2. Create a data frame called “cone” with three elements. The first two elements are radiuses and heights of the cones:

   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 - solution

1. x[x>40 & x<80]
2. ```
   > 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)
   > V <- 1/3*pi*R^2*H
   > V
   [1] 44.67974 33.00768 27.09756 32.20057 21.34939 39.99652
   > data.frame(R,H,V)
   ```

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

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 (element wise)

`&&`: AND (only leftmost element)

`|`: OR (element wise)

`||`: OR (only leftmost element)

An example if.R

```
> 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
}

An example for.R

```
> 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

Simple Statistical 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)  ## order() returns the indices of the vector in sorted order
[1] 2 4 1 3
```

Sorting and Random Samples

- 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
```

CENTER FOR COMPUTATION & TECHNOLOGY

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 (data frame)** and evaluate a function on each element
- `sapply` - Same as `lapply` but try to simplify the result

```
## lapply & sapply example
> x <- list(a = 1, b = 1:3, c = 10:100)
> lapply(x, FUN = length)
> sapply(x, FUN = length)
> lapply(x, FUN = sum)
> sapply(x, FUN = sum)
```

Other `apply` Functions

- In statistics, one of the most basic activities is computing statistic of variables
- `tapply` - Apply a function over subsets of a **vector**
- `mapply` - Multivariate version of `lapply`

```
## generate medical data for tapply example (https://www.r-bloggers.com/r-function-of-the-day-tapply-2/)
> medical.example <-
+     data.frame(patient = 1:100,
+                age = rnorm(100, mean = 60, sd = 12),
+                treatment = gl(2, 50,
+                  labels = c("Treatment", "Control")))
> tapply(medical.example$age, medical.example$treatment, mean)
Treatment   Control
  61.7065   59.9123
```

CENTER FOR COMPUTATION & TECHNOLOGY

User Defined Functions

- Similar to other languages, functions in rare 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

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

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
- Libraries that R currently searching can be shown with `.libPaths()`

LSU CENTER FOR COMPUTATION & TECHNOLOGY

Installing and Loading R Packages - Cluster

- Installation
  - You most likely do NOT have root privilege
  - Point the environment variable `R_LIBS_USER` to a desired location
  - Use the `install.packages(“<package name>”)` function to install a library
- Loading: the `library(<package name>)` function load previously installed packages
- Libraries that R currently searching can be shown with `.libPaths()`

```
[ychen64@mike002 ~]$ export R_LIBS_USER=/home/ychen64/packages/R/libraries
[ychen64@mike002 ~]$ echo $R_LIBS_USER
/home/ychen64/packages/R/libraries
[ychen64@mike002 ~]$ 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)
…

> .libPaths ()
[1] "/home/ychen64/packages/R/libraries"
[2] "/home/packages/r/3.4.3/INTEL-18.0.0/lib64/R/library“

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

| Hi! Type swirl() when you are ready to begin.
```

Listing and Unloading R Packages - PC and Cluster

- List all available packages `library()`, press “q” to quit
- List all packages in the default library (on the SuperMike2 cluster the default is /home/packages/r/3.5.3/INTEL-18.0.0/lib64/R/library) `library(lib = .Library)`
- Show currently loaded libraries: the `search()` function or `sessionInfo()` function
- Check package version: `packageVersion("<package name>")`
- Unload `detach(package:<package name>)`

```
[ychen64@mike002 ~]$ 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)
…

> library()
> library(lib = .Library)

> search()
 [1] ".GlobalEnv"         "package:swirl"     "package:stats"
 [4] "package:graphics"   "package:grDevices" "package:utils"
 [7] "package:datasets"   "package:methods"   "Autoloads"
[10] "package:base"
> packageVersion("swirl")
> detach(package:swirl)
```

Updating and Uninstall R Packages - PC and Cluster

- Update `update.packages("<package name>")`
- Uninstall `remove.packages("<package name>")`
- Documentation page: http://www.hpc.lsu.edu/docs/faq/installation-details.php

```
[ychen64@mike002 ~]$ 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)
…

> update.packages("swirl")
> remove.packages("swirl")
```

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

Take-home message

- R basics
  - What is R
  - How to run R codes on PC and cluster
  - Basic syntax (variable assignment)
  - R as a calculator
  - Data classes and objects in R (dataframe!)
  - Flow control structures
  - Functions (basic statistical functions)
  - How to install and load R packages

Not Covered

- Data acquisition
- Data inspection
- Report generation
- Data manipulation
- Statistical analysis (e.g regression models, machine learning/data mining)
- Advanced 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)
- Blogs
  - https://www.r-bloggers.com
- Educational R packages
  - Swirl: Learn R in R

Next HPC Training

- Introduction to Python, October 28.
- Weekly trainings during regular semester
  - Wednesdays “9:00am-11:00am” session, pure Zoom
- Programming/Parallel Programming workshops
  - Usually in summer

More R Tutorials – Data Analysis in R

- You will learn the data analysis fundamentals with applications in R.
- The data pre-processing using R will be introduced first, then some basic statistical analysis methods such as linear regression, classification as well as re-sampling methods for the basic machine learning will be covered

More R Tutorials – Data Visualization in R

- This training provided an introduction to the R graphics in detail
- An overview on how to create and save graphs in R, then focus on the ggplot2 package.
- http://www.hpc.lsu.edu/training/archive/tutorials.php

More R Tutorials – Parallel Computing with R

- This training focused on how to use the "parallel" package in R and a few related packages to parallelize and enhance the performance of R programs
- http://www.hpc.lsu.edu/training/archive/tutorials.php

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?