Introduction to RStudio

Yuwu Chen
HPC @ LSU
Outline

– R basics
  • What is R
  • Basic syntax
  • Data classes and objects

– RStudio basics
  • What is RStudio
  • RStudio IDE features
  • User environment and data acquisition
  • Install and load R packages
  • Coding tools
  • Use Version Control with RStudio
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.
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 3.6.3 released on Feb 29, 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 Comprehensive R Archive Network
    • Source code and binaries, user contributed packages and documentation
    — More than 15,000 packages available on CRAN (as of March 2020)
      • 6,000 five years ago
• Free to use!
Installing and Loading R

• On your PC
  – RStudio is the de facto environment for R on a desktop system
  – R console from CRAN

• On HPC cluster
  – R on all LONI and LSU HPC clusters
    • SuperMIC and QB2: r/3.5.3/INTEL-18.0.1
    • SuperMike2: r/3.5.3/INTEL-18.0.0
  – RStudio via Open OnDemand on SuperMike2
  – 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

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

- Linux/Mac/Windows version available
- Similar graphic user interface (GUI) to other Windows software, dividing the screen into panes
  - Source code
  - Console
  - Workspace
  - Others (help message, plot etc.)
On LONI and LSU HPC Clusters

• Two modes to run R on clusters
  – Interactive mode
    • Type `R` command to launch the console, then run R commands in the console
    • RStudio Server at [LSU HPC Open OnDemand](https://hpc.lsu.edu) (LSU HPC users only)
  – 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

R version 3.4.3 (2017-11-30) -- "Kite-Eating Tree"
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.

>
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
  • Basic syntax
  • Data classes and objects

– RStudio basics
  • What is RStudio
  • RStudio IDE features
  • User environment and data acquisition
  • Install and load R packages
  • Coding tools
  • Use Version Control with RStudio
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

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

- Last expression can be retrieved through an internal object `.Last.value`
  ```r
  > 2*4
  [1] 8
  > x <- .Last.value
  > x
  [1] 8
  ```

- Removing objects with the function `rm()`
  ```r
  > 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.
  ```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
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

```r
> 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
    ```r
    > x <- 2 * 4
    [1] 8
    ```
  – Spaces cannot be placed between the `<` and `-` in the assignment operator
    ```r
    > x < - -2 * 4
    [1] FALSE
    ```
Outline

– R basics
  • What is R
  • Basic syntax
    • Data classes and objects
– RStudio basics
  • What is RStudio
  • RStudio IDE features
  • User environment and data acquisition
  • Install and load R packages
  • Coding tools
  • Use Version Control with RStudio
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] TRUE
  > is.character(string)
  [1] TRUE
  > 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
Data Objects - Vectors

• Vectors can only contain elements of the same data class
• Vectors can be constructed by
  – Using the `c()` function (concatenate)
    ```r
    > 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
    ```r
    > as.numeric(d)
    ```
  – Coercion will occur when mixed objects are passed to the `c()` function, as if the `as.<Type>()` function is explicitly called
    ```r
    > 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)`
Vectors Indexing

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

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

• R matrices are constructed column-wise by default

  ```r
  > 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 – 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

```r
> # 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

```r
> 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 - 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
  – 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 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()`

```r
> 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

```r
> names(my_df) <- c("c1","c2")
> my_df
   c1 c2
 1 31 M
 2 40 F
 3 50 M
```
Matrices and Data Frames Indexing

• One can use \([<\text{index}>, <\text{index}>]\) to access individual element

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

• Indexing by columns

```r
> 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

```r
> 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

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

• the `[[ ]]` operator

```r
> 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
```
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
More R Tutorials and Workshop

• Tutorials
  – Introduction to R
  – Data Analysis in R
  – Data Visualization in R
  – Parallel Computing with R

• Workshop
  – Introduction to R
  – Data Mining with R

http://www.hpc.lsu.edu/training/index.php
Outline

— R basics
  • What is R
  • Basic syntax
  • Data classes and objects

— RStudio basics
  • What is RStudio
  • RStudio IDE features
  • User environment and data acquisition
  • Install and load R packages
  • Coding tools
  • Use Version Control with RStudio
What is RStudio

• RStudio is an integrated development environment (IDE) for R
• RStudio is available in two formats:
  – RStudio Desktop
  – RStudio Server
• RStudio Desktop and RStudio Server are both available in free and fee-based (commercial) editions
• Initial release: 28 February 2011
Why RStudio

R STUDIO

... BUT WHY!

Why you guys never teach RStudio??

HPC User Services
Why RStudio

- RStudio integrates the tools you use with R into a single environment
- RStudio includes powerful coding tools
- RStudio enables rapid navigation to files and functions
- RStudio make it easy to start new or find existing projects
- RStudio has integrated support for Git and Subversion
- RStudio supports authoring HTML, PDF, Word Documents, and slide shows
- RStudio supports interactive graphics with Shiny and ggvis

Why RStudio

https://i2.wp.com/res.cloudinary.com/syknaptic/image/upload/v1521320144/tidyverse_meme_oceake.png
Installing and Loading RStudio

• On your PC
  – RStudio Desktop
    https://rstudio.com/products/rstudio/download/
  – RStudio Server (available for some Linux platforms)

• On HPC cluster
  – RStudio Server via Open OnDemand on SuperMike2
  – In your own directory as you requested (Not recommended)
Installing and Loading RStudio

• On your PC

RStudio Desktop 1.2.5033 - Release Notes

1. Install R. RStudio requires R 3.0.1+

2. Download RStudio Desktop. Recommended for your system:

   ![DOWNLOAD RSTUDIO FOR WINDOWS 1.2.5033 | 149.83MB](image)

   Requires Windows 10/8/7 (64-bit)

All Installers

Linux users may need to import RStudio's public code-signing key prior to installation, depending on the operating system's security policy.

RStudio 1.2 requires a 64-bit operating system. If you are on a 32 bit system, you can use an older version of RStudio.

<table>
<thead>
<tr>
<th>OS</th>
<th>Download</th>
<th>Size</th>
<th>SHA-256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 10</td>
<td><img src="image" alt="RStudio-1.2.5033.exe" /></td>
<td>149.83 MB</td>
<td>769d6b46</td>
</tr>
<tr>
<td>macOS 10.12+</td>
<td><img src="image" alt="RStudio-1.2.5033.dmg" /></td>
<td>326.89 MB</td>
<td>b57e9373</td>
</tr>
</tbody>
</table>
Installing and Loading RStudio

• On your PC
  – Install R, better choose the same version as on the cluster.
  https://cran.r-project.org/
Installing and Loading RStudio

• On your PC
  – Download RStudio Desktop

2. [Download RStudio Desktop](#)  Recommended for your system:

   ![Download RStudio Desktop](#)

   Requires Windows 10/8/7 (64-bit)
Outline

– R basics
  • What is R
  • Basic syntax
  • Data classes and objects

– RStudio basics
  • What is RStudio
  • RStudio IDE features
  • User environment and data acquisition
  • Install and load R packages
  • Coding tools
  • Use Version Control with RStudio
Customizing Pane Layout

Menu “Tools” -> “Global Options”-> “Pane Layout”
Toolbar and Source Pane

Creating or opening various files (e.g. R script)

Visualizing data

Coding window

New file

Open existing file

Save current document

Create a project

Note: raw data opened here CANNOT be accessed in R
• Source, console and terminal all support:
  – Automatic completion of typing file, directory or command name via the **TAB** key
  – Recall previous commands using the up arrow (↑)
Keyboard Shortcuts

• Some of the more useful shortcuts
  – Ctrl+1 — Move focus to the Source Editor
  – Ctrl+2 — Move focus to the Console
  – Ctrl+L — Clear the Console
  – Esc — Interrupt R
Environment & History & Connections

- Data to data, values to values
- Importing data (will introduce later)

List / grid switch
Clear ALL objects
Access lib objects
Way to import raw data
Environment & History & Connections

- History can be saved to / loaded from a file
- One or multiple selected commands can be sent to Console or Source
- Remove selected or clear all of the entries
The Connection Pane connects to a variety of data sources, and explore the objects and data inside the connection.
Files & Plots & Packages & Help & Viewer

<table>
<thead>
<tr>
<th>Files</th>
<th>Plots</th>
<th>Packages</th>
<th>Help</th>
<th>Viewer</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Image" /></td>
<td><img src="image.png" alt="Image" /></td>
<td><img src="image.png" alt="Image" /></td>
<td><img src="image.png" alt="Image" /></td>
<td><img src="image.png" alt="Image" /></td>
</tr>
<tr>
<td>New Folder</td>
<td>Delete</td>
<td>Rename</td>
<td>More</td>
<td></td>
</tr>
</tbody>
</table>

- .RData
- .Rhistory
- 1.RData

- 20170125-test(2825009709)
- 20170201-HPC User Environment 1(2825014737)
- 20170607-HPC User Environment 1(63922977626925319)
- 20170906-HPC User Environment 1(72269933191375799)
- 20171004-Introduction to R(74703386077833790)
- 20180207-HPC User Environment 1(85880121929509051)
- 20180307-Hands-On Practice Session(88659819603438417)
- 20180314-Introduction to R(89289146025126688)
- 20180620-HPC User Environment 2(98184207135681332)
- 20180920-HPC User Environment 1(10532000313810882)

- You CAN customize Pane Layout
Outline

– R basics
  • What is R
  • Basic syntax
  • Data classes and objects

– RStudio basics
  • What is RStudio
  • RStudio IDE features
    • User environment and data acquisition
    • Install and load R packages
    • Coding tools
    • Use Version Control with RStudio
How does R work

- R works best if you have a dedicated folder called “working directory”. Put all data files in the working directory (or in its subdirectories).

```r
> getwd()  # Show current working directory
[1] "C:/Users/Administrator/Documents/project2"
> dir.create("data") # Create a new directory
> getwd()
[1] "/home/ychen64"
> setwd("data")
> getwd()
[1] "C:/Users/Administrator/Documents/project2/data"
> list.files()  # List files in current directory
```
How does R work

• Your objects will be automatically saved in the .RData file in the working directory.

• To quit use `q()` or `CTRL + D` or just kill the window. R will ask you “Save workspace image to .RData?”. You can choose:
  – Save: leave R without saving your results in R;
  – Don’t Save: save your results in .RData in your working directory;
  – Cancel: not quitting R.

• The commands you type in console or executed in Source Pane will be automatically saved in the .Rhistory file in the working directory.
Projects in RStudio

• RStudio Project is working directory “Pro”
  – includes the functionality of working directory
  – provides “Project Options” to set on a per-project basis to customize the behavior of RStudio
  Menu “Tools” -> “Project Options”
  – works with version control system
Creating Projects in RStudio

- When creating new project RStudio creates:
  - a project file (with an .Rproj extension) within the project directory
  - a hidden directory (named .Rproj.user) where project-specific temporary files are stored, which is also automatically added to .Rbuildignore, .gitignore, etc. if required.
Opening Projects in RStudio

• RStudio project can be opened
  – in menu “File” -> “Open Project...”
  – on the toolbar
  – by double-clicking the .Rproj file

• When a project is opened within RStudio the following actions are taken:
  – A new R session (process) is started
  – The .RData, .Rhistory and .Rprofile (if any) files in the project's main directory is sourced by R
  – The current working directory is set to the project directory
Closing Projects in RStudio

• RStudio project can be closed
  – in menu “File” -> “Close Project...” (w/o quitting RStudio)
  – in menu “File” -> “Open Project in New Session...”
  – when closing the RStudio
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()`

```r
```
**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.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read.table</td>
<td>write.table</td>
<td>for reading/writing tabular data</td>
</tr>
<tr>
<td>read.csv</td>
<td>write.csv</td>
<td></td>
</tr>
<tr>
<td>readLines</td>
<td>writeLines</td>
<td>for reading/writing lines of a text file</td>
</tr>
<tr>
<td>source</td>
<td>dump</td>
<td>for reading/writing in R code files</td>
</tr>
<tr>
<td>dget</td>
<td>dput</td>
<td>for reading/writing in R code files</td>
</tr>
<tr>
<td>load</td>
<td>save</td>
<td>for reading in/saving workspaces</td>
</tr>
</tbody>
</table>
# read.csv() is identical to read.table() except that the default separator is a comma.

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

Reading Data in Environment Pane

Carefully choose the options of import
Steps for Data Analysis

• Get the data
• Read the data to R
• Inspect the data
• Preprocess the data (remove missing and dubious values, discard columns not needed etc.)
• Analyze the data
• Generate the report
More R Tutorials for Forbes Case Study

• Tutorials
  – Introduction to R
  – Data Analysis in R

http://www.hpc.lsu.edu/training/index.php
Outline

— R basics
  • What is R
  • Basic syntax
  • Data classes and objects

— RStudio basics
  • What is RStudio
  • RStudio IDE features
  • User environment and data acquisition
  • Install and load R packages
  • Coding tools
  • Use Version Control with RStudio
Installing and Loading R Packages

• libraries that R currently searching can be shown with .libPaths() in the R console
  
  ```r
  > .libPaths()
  [1] "/home/ychen64/packages/R/libraries"  # user’s own directory
  [2] "/home/packages/r/3.4.3/INTEL-18.0.0/lib64/R/library"  # system path
  ```

• Installation:
  – Option 1: menu “Tools” -> “Install Packages”
  – Option 2: run `install.packages("<package name>")` function in the console

• Loading: the `library <package name>` function load previously installed packages
Installing and Loading R Packages - HPC Cluster

• You do NOT own a directory to install your packages by default, you need to specify it
  – Option 1: Point the environment variable `R_LIBS_USER` to a desired location (doesn’t apply for RStudio)

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

  – Option 2: Save the path of packages to `.Rprofile`
    (particularly useful with RStudio project)

```
$ cat /home/ychen64/project1/.Rprofile
myPaths <- .libPaths() #system path
myPaths <- c("/home/ychen64/project1" "/project/ychen64/packages/R-3.5.3/libraries",myPaths)
.libPaths(myPaths)
```

A new directory dependent to the project
Your own directory by default
Listing and Unloading
R Packages - Command Lines

• List all available packages `library()`
• List all packages in the default system library `library(lib = .Library)`
• Show currently loaded libraries: `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"
[10] "package:base"
> packageVersion("swirl")
> detach(package:swirl)
### Listing and Unloading R Packages - RStudio GUI

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>abind</td>
<td>Combine Multidimensional Arrays</td>
<td>1.4-5</td>
</tr>
<tr>
<td>AmesHousing</td>
<td>The Ames Iowa Housing Data</td>
<td>0.0.3</td>
</tr>
<tr>
<td>askpass</td>
<td>Safe Password Entry for R, Git, and SSH</td>
<td>1.1</td>
</tr>
<tr>
<td>assertthat</td>
<td>Easy Pre and Post Assertions</td>
<td>0.2.1</td>
</tr>
<tr>
<td>backports</td>
<td>Reimplementations of Functions Introduced Since R-3.0.0</td>
<td>1.1.3</td>
</tr>
<tr>
<td>base64enc</td>
<td>Tools for base64 encoding</td>
<td>0.1-3</td>
</tr>
<tr>
<td>BH</td>
<td>Boost C++ Header Files</td>
<td>1.0-0-1</td>
</tr>
<tr>
<td>bindr</td>
<td>Parametrized Active Bindings</td>
<td>0.1.1</td>
</tr>
<tr>
<td>bindrcpp</td>
<td>An 'Rcpp' Interface to Active Bindings</td>
<td>0.2.2</td>
</tr>
<tr>
<td>bitops</td>
<td>Bitwise Operations</td>
<td>1.0-6</td>
</tr>
<tr>
<td>callr</td>
<td>Call R from C</td>
<td>3.2.0</td>
</tr>
<tr>
<td>caret</td>
<td>Companion to Applied Regression</td>
<td>3.0-2</td>
</tr>
<tr>
<td>caretData</td>
<td>Companion to Applied Regression Data Sets</td>
<td>3.0-2</td>
</tr>
<tr>
<td>caret2</td>
<td>Classification and Regression Training</td>
<td>6.0-82</td>
</tr>
<tr>
<td>cellranger</td>
<td>Translate Spreadsheet Cell Ranges to Rows and Columns</td>
<td>1.1.0</td>
</tr>
<tr>
<td>cli</td>
<td>Helpers for Developing Command Line Interfaces</td>
<td>1.1.0</td>
</tr>
<tr>
<td>clipr</td>
<td>Read and Write from the System Clipboard</td>
<td>0.5.0</td>
</tr>
<tr>
<td>clisymbols</td>
<td>Unicode Symbols at the R Prompt</td>
<td>1.2.0</td>
</tr>
<tr>
<td>colorspace</td>
<td>A Toolbox for Manipulating and Assessing Colors and Palettes</td>
<td>1.4-1</td>
</tr>
</tbody>
</table>

- **Load/unload package by selecting/deselecting**
- **Scroll down to check system packages**
- **Package version**
Updating and Uninstall R Packages - PC and Cluster

- **Update** `update.packages("<package name>")`
- **Uninstall** `remove.packages("<package name>")`

Do you need to update a package to the newest? Please use good judgement.
Outline

— R basics
  • What is R
  • Basic syntax
  • Data classes and objects

— RStudio basics
  • What is RStudio
  • RStudio IDE features
  • User environment and data acquisition
  • Install and load R packages
  • Coding tools
  • Use Version Control with RStudio
RStudio Coding Window Overview

- Position of the focus
- File type
- Show in a new window
- Run the line where the focus is at
- Source the entire file

1: x <- c(2,5,3,9,8,11,6)
2: count <- 0
3: for (i in x) {
4:   if (i %% 2 == 0) count <- count + 1
5: }
6: count
Code Diagnostics in RStudio

- Diagnostics can be enabled and options can be set in menu “Tools” -> “Global Options” -> “Code”

Code Diagnostics in RStudio

• “Check arguments to R function calls” tries to detect whether a particular call to a function will succeed.

Code Diagnostics in RStudio

- “Warn if variable is defined but not used” helps to identify if a variable is created but never used.

“Provide R style diagnostics (e.g. whitespace)” checks to see if your code conforms to Hadley Wickham’s style guide, and reports style warnings when encountered.

Code Debugging in RStudio

- RStudio has integrated the R debugging tools.
- In order to enter debug mode, you’ll need to tell R when you want to pause the computation.
  - R doesn’t have a “pause now” feature (not useful as most R computations are too fast to stop in the middle)
  - Pick best way to pause calculation

Entering the Debug Mode

- Stopping on a line
  - Editor breakpoints
  - `browser()` breakpoints
- Stopping when a function executes
- Stopping when an error occurs

Stopping on a Line

- Editor breakpoints
  - is the most common (and easiest) way to stop on a line
  - takes effect immediately and don’t require you to change your code
  - works by injecting some tracing code into the R function object.

Stopping on a Line

- `browser()` **breakpoints**
  - halts execution and invokes an environment browser when it is called
  - is actually part of the code, so it needs to be applied like any other code change in order to become active (e.g. source it)

```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 {
  `browser()`
  print("x is greater or equal to 5")
}
```

Stopping when a Function Executes

• Why need this type of stopping?
  – Sometimes you don’t have the source file for the code you want to debug.

• The breakpoint causes the debugger to activate immediately when the function is run.
  – a one-shot breakpoint: `debugonce()`
  
  ```r
  # R Code Example
  pow <- function(x, y) {
    result <- x * y
  }
  debugonce(pow)
  pow(4, 2)
  ```

  – debug a function every time it executes: `debug()` and `undebug()`

Stopping when an Error Occurs

- RStudio halts execution at the point where the error is raised

Using the Debugger

Using the Debugger

Using the Debugger

Outline

– R basics
  • What is R
  • Basic syntax
  • Data classes and objects

– RStudio basics
  • What is RStudio
  • RStudio IDE features
  • User environment and data acquisition
  • Install and load R packages
  • Coding tools
  • Use Version Control with RStudio
What is and Why Version Control

• Version Control is the management of changes to documents, computer programs, large web sites, and other collections of information.
• Version control is not only good for team collaboration but also benefits for individual work.
  – Why should I use version control?
  – R and version control for the solo data analyst
• RStudio IDE has integrated support for version control.:
  – Git
  – Subversion
Installing and Activating Git in RStudio

- **Installation**
  - HPC Clusters: load the Module key for Git
- in menu “Tools” -> “Global Options” -> “Git/SVN”
Various Scenarios of Initializing Git in RStudio

• Create a new project
  – with a totally new Git repository
  – based on an existing remote Git repository (Github)
  – using a directory already under version control

• Add version control to an existing project
  – by remote repositories
  – by Git locally
Various Scenarios of Initializing Git in RStudio

- Create a new project with a brand new Git repository

Check this box when creating a new project
Various Scenarios of Initializing Git in RStudio

• Create a new project from an existing remote Git repo (Github)

Equals to “git clone”, then creating a new project based on the directory already under version control
Various Scenarios of Initializing Git in RStudio

- Create a new project using a directory already under version control
Various Scenarios of Initializing Git in RStudio

- Add version control to an existing project
  - by remote repositories (rare)
  - by Git locally
in menu “Tools” -> “Project Options” -> “Git/SVN”
Git Panes

• The Git pane shows the file status (in terms of git)
• Git commands can be typed in Terminal
Working with Git

Untracked file, not in the Git yet (or no one cares about its change)
Working with Git

- Staged file, needs to take actions (to commit or unstaged back to modified etc)

- Use “git add” command to staged file, or
- check the Staged box in the Git pane
Working with Git

- Modified file, may use "git add" to staged file, or the change can be reversed to let the status be unmodified.

- Two icons for the same file ("dual" status in RStudio)
  - File has never been committed.
Working with Git

Once a file is committed, its status is “unmodified” and will not shown in the Git pane (unless been changed/removed again)

- git commit -m “commit message”
Ignoring Files

• Any Files such as temporary, very large or R project log files that you don’t want Git to automatically add or even show you as being untracked can be
  – added to .gitignore

$ cat .gitignore
*.o[oa]
# tells Git to ignore any files ending in ".o" or ".a" object
  – selected in the Git pane
Push Current Repository to GitHub

• If the current repository was created from an existing repository on Github
  
  – git remote and git push commands:

  $ git remote add origin https://github.com/dbxmcf/project2.git

  $ git push -u origin master

  – or push button in the Git pane:

• If the current repository hasn’t been connected to Github:
  
  – Create a new repo on GitHub: https://github.com/new. Give it the same name as your project.

  – same git remote and git push commands or the Git pane
Take-home message

• R Studio
  – What is R and RStudio
  – How to install RStudio
  – Basic IDE features
  – RStudio as a coding tool
  – Version control structures
Learning RStudio

- User documentation on RStudio
  - https://support.rstudio.com/hc/en-us
- 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

• Run HPC jobs with Agave Web Interface, April 1st.
• Weekly trainings during regular semester
  – Wednesdays “9:00am-11:00am” session, Frey 307 CSC
• Programming/Parallel Programming workshops
  – Usually in summer
More R Tutorials – Introduction to R

• R basics
  – What is R
  – How to run R codes
  – Basic syntax
  – Data classes and objects in R
• Flow control structures
• Statistical functions
• How to install and load R packages
• http://www.hpc.lsu.edu/training/archive/tutorials.php
More R Tutorials – Data Analysis in R

- Data analysis fundamentals with applications in R.
  - The data pre-processing
  - Basic statistical analysis methods such as linear regression, classification as well as re-sampling methods for the basic machine learning will be covered
- [http://www.hpc.lsu.edu/training/archive/tutorials.php](http://www.hpc.lsu.edu/training/archive/tutorials.php)
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](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?
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 1 - solution

1. \[ x <- \text{seq}(\text{from}=1, \text{to}=100, \text{by}=2) \]
2. \[ x[x>60\&x<80] \]
3. \[ R <- \text{c}(2.27, 1.98, 1.69, 1.88, 1.64, 2.14) \]
   \[ H <- \text{c}(8.28, 8.04, 9.06, 8.70, 7.58, 8.34) \]
   \[ V <- \frac{1}{3}\pi R^2 H \]
   \[ V \]
   \text{[1]} 44.67974 33.00768 27.09756 32.20057 21.34939 39.99652
   \[ \text{data.frame}(R,H,V) \]
Exercises 2

1. Create a RStudio project called “Forbes”
2. Download the raw data of Forbes to the working/project directory of “Forbes”, unzip it
3. Import dataset in RStudio, save it as “forbes”
4. Run the following commands in the console:
   - head(forbes)
   - str(forbes)
   - summary(forbes)
What is R

- R mailing lists: [http://www.R-project.org/mail.html](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).
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
Data Objects - Vectors

- NA in R means missing value

```r
> 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!

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

• R matrices can also be constructed by
  – **Passing an `dim` attribute to a vector**
    ```r
    > m <- 1:10
    > m
    [1]  1  2  3  4  5  6  7  8  9 10
    > dim(m) <- c(2, 5)
    > m
    [1,]    1    3    5    7    9
    [2,]    2    4    6    8   10
    ```
  – **Using `cbind()` or `rbind()` functions**
    ```r
    > 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 - 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
    ```r
    > my_list[[1]]
    [1] 31 32 40
    ```
  – the “$” sign if the elements of list have names
    ```r
    > my_list$age
    [1] 31 32 40
    ```
• Referring individual element
  ```r
  > my_list$age[1]
  [1] 31
  ```
Data Objects - Data Frames

• Row names are automatically assigned and are by default labelled “1”, “2”, “3”, ...

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

• These can also be renamed if desired

```r
> row.names(my_df) <- c("r1","r2","r3")
> my_df
  c1 c2
 r1 31 M
 r2 40 F
 r3 50 M
```
Matrices and Data Frames Indexing

- Indexing can be conditional on another variable!

```r
> 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
```
Flow Control Structures

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

<table>
<thead>
<tr>
<th>Control Structure</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>if ... else</td>
<td>testing a condition</td>
</tr>
<tr>
<td>for</td>
<td>executing a loop (with fixed number of iterations)</td>
</tr>
<tr>
<td>while</td>
<td>executing a loop when a condition is true</td>
</tr>
<tr>
<td>repeat</td>
<td>executing an infinite loop</td>
</tr>
<tr>
<td>break</td>
<td>breaking the execution of a loop</td>
</tr>
<tr>
<td>next</td>
<td>skipping to next iteration</td>
</tr>
<tr>
<td>return</td>
<td>exit a function</td>
</tr>
</tbody>
</table>
Testing Conditions

# Comparisons: <, <=, >, >=, ==, !=

# Logical operations:
# !: NOT
# &: AND (elementwise)
# &&: AND (only leftmost element)
# |: OR (element wise)
# ||: OR (only leftmost element)

An example if.R

```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

```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 Statistic Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min()</td>
<td>Minimum value</td>
</tr>
<tr>
<td>max()</td>
<td>Maximum value</td>
</tr>
<tr>
<td>which.min()</td>
<td>Location of minimum value</td>
</tr>
<tr>
<td>which.max()</td>
<td>Location of maximum value</td>
</tr>
<tr>
<td>sum()</td>
<td>Sum of the elements of a vector</td>
</tr>
<tr>
<td>mean()</td>
<td>Mean of the elements of a vector</td>
</tr>
<tr>
<td>sd()</td>
<td>Standard deviation of the elements of a vector</td>
</tr>
<tr>
<td>quantile()</td>
<td>Show quantiles of a vector</td>
</tr>
<tr>
<td>summary()</td>
<td>Display descriptive statistics</td>
</tr>
</tbody>
</table>

```r
> 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

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Distribution name in R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>unif</td>
</tr>
<tr>
<td>Binomial</td>
<td>binom</td>
</tr>
<tr>
<td>Poisson</td>
<td>pois</td>
</tr>
<tr>
<td>Geometric</td>
<td>geom</td>
</tr>
<tr>
<td>Gamma</td>
<td>gamma</td>
</tr>
<tr>
<td>Normal</td>
<td>norm</td>
</tr>
<tr>
<td>Log Normal</td>
<td>lnorm</td>
</tr>
<tr>
<td>Exponential</td>
<td>exp</td>
</tr>
<tr>
<td>Student’s t</td>
<td>t</td>
</tr>
</tbody>
</table>
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()`.

```r
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
  ```
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 evaluates 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, ...)
```
```r
> x <- matrix(rnorm(200), 20, 10)
# Row means
> apply(x, 1, mean)
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
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
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

```r
## 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 summaries of variables
- `tapply` - Apply a function over subsets of a vector
- `mapply` - Multivariate version of `lapply`

```r
## 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
```

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
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
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).

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

- Work on the project - your objects can be automatically saved in the .RData file.
- To quit use `q()` or `CTRL + D` 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 (recommended);
  - 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()`

```r
```

Before R3.3, “unsupported URL scheme” error may occur when downloading from https method="curl"
### 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.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>read.table</code></td>
<td><code>write.table</code></td>
<td>for reading/writing tabular data</td>
</tr>
<tr>
<td><code>read.csv</code></td>
<td><code>write.csv</code></td>
<td></td>
</tr>
<tr>
<td><code>readLines</code></td>
<td><code>writeLines</code></td>
<td>for reading/writing lines of a text file</td>
</tr>
<tr>
<td><code>source</code></td>
<td><code>dump</code></td>
<td>for reading/writing in R code files</td>
</tr>
<tr>
<td><code>dget</code></td>
<td><code>dput</code></td>
<td>for reading/writing in R code files</td>
</tr>
<tr>
<td><code>load</code></td>
<td><code>save</code></td>
<td>for reading in/saving workspaces</td>
</tr>
</tbody>
</table>
Reading Data with `read.table()` (1)

```r
# List of arguments of the read.table() function
forbes <- read.csv("Forbes2000.csv",header=T,stringsAsFactors = FALSE,na.strings = "NA",sep=";")
```
Reading Data with `read.table()`

- **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()`

- 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.

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

```r
library(XLConnect)
w <- 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).

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

- Note: the libraries above requires both Java Dev Kit and rJava library. The later is not available for R version installed on QB2 and SuperMic.
Inspecting Data (1)

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

```r
> 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.

```r
> 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 ...
```
Inspectioning Data (3)

- **Statistical summary of the “Forbes” dataframe.**

```r
> summary(forbes)

<table>
<thead>
<tr>
<th>rank</th>
<th>name</th>
<th>country</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Qu.</td>
<td>Class :character</td>
<td>Class :character</td>
<td>Class :character</td>
</tr>
<tr>
<td>Median</td>
<td>Mode :character</td>
<td>Mode :character</td>
<td>Mode :character</td>
</tr>
<tr>
<td>Mean</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd Qu.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sales</th>
<th>profits</th>
<th>assets</th>
<th>marketvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>-25.8300</td>
<td>0.270</td>
<td>0.02</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>0.0800</td>
<td>4.025</td>
<td>2.72</td>
</tr>
<tr>
<td>Median</td>
<td>0.2000</td>
<td>9.345</td>
<td>5.15</td>
</tr>
<tr>
<td>Mean</td>
<td>0.3811</td>
<td>34.042</td>
<td>11.88</td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>0.4400</td>
<td>22.793</td>
<td>10.60</td>
</tr>
<tr>
<td>Max.</td>
<td>20.9600</td>
<td>1264.030</td>
<td>328.54</td>
</tr>
</tbody>
</table>

- **There are missing values in the profits category.**
Inspecting Data (4) - Basic Plots

• R offers a remarkable variety of graphics.
  > attach(forbes) # attach the data frame
  > boxplot(sales) # boxplot
  > plot(sales,assets) # scatterplot

• The result of a graphical function cannot be assigned to an object but is sent to a graphical device (i.e. a graphical window or a file)

• Save plots. For example:
  • pdf, two plots will be saved into one pdf file
    > pdf('rplot%03d.pdf')
    > boxplot(sales)
    > plot(sales,assets)
    > dev.off() # must turn off the graphical device
  • jpg, two plots will be saved into two jpg files
    > jpeg('rplot%03d.jpg')
    > boxplot(sales)
    > plot(sales,assets)
    > dev.off() # must turn off the graphical device
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
  – Data inspection
    – 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 \texttt{knitr} and \texttt{rmarkdown} packages can convert them into documents of various formats

• Help make your research reproducible
Put Everything Together

• Run R commands in batch mode with **Rscript**

```r
[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")) {
}
...

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

```r
# 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.

```r
> Bermudacomp <- subset(forbes, country == "Bermuda")
> table(Bermudacomp[, "category"])
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banking</td>
<td>1</td>
</tr>
<tr>
<td>Capital goods</td>
<td>1</td>
</tr>
<tr>
<td>Conglomerates</td>
<td>2</td>
</tr>
<tr>
<td>Food drink &amp; tobacco</td>
<td>1</td>
</tr>
<tr>
<td>Food markets</td>
<td>1</td>
</tr>
<tr>
<td>Insurance</td>
<td>10</td>
</tr>
<tr>
<td>Media Oil &amp; gas operations</td>
<td>1</td>
</tr>
<tr>
<td>Software &amp; services</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
• Subsetting by column

# Create another data frame with only numeric variables

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

# Or simply use indexing

```r
forbes3 <- forbes[,c(5:8)]
str(forbes3)
```