# Introduction to R

Le Yan

HPC @ LSU

*Some materials are borrowed from the "Data Science" course by John Hopkins University on Coursera.*

# Outline

- R basics
  - How to run R codes
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis case study: NOAA weather hazard data
  - Data acquisition and inspection
  - Data preprocessing
  - Data analysis
  - Report generation

# History of R

- R is a dialect of the S language
  - S was initiated at the Bell Labs as an internal statistical analysis environment
  - Most well known implementation is S-plus (most recent stable release was in 2010)
- The R core group was formed in 1997, who controls the source code of R (written in C)
- R 1.0.0 was released in 2000
- The current version is 3.3.3

# 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 10,000 packages available on CRAN (as of March 2017)
    - 6,000 two years ago
- Free to use!

# Outline

- R basics
  - How to run R codes
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis case study: NOAA weather hazard data
  - Data acquisition and inspection
  - Data preprocessing
  - Data analysis
  - Report generation

# Two Ways of Running R Codes

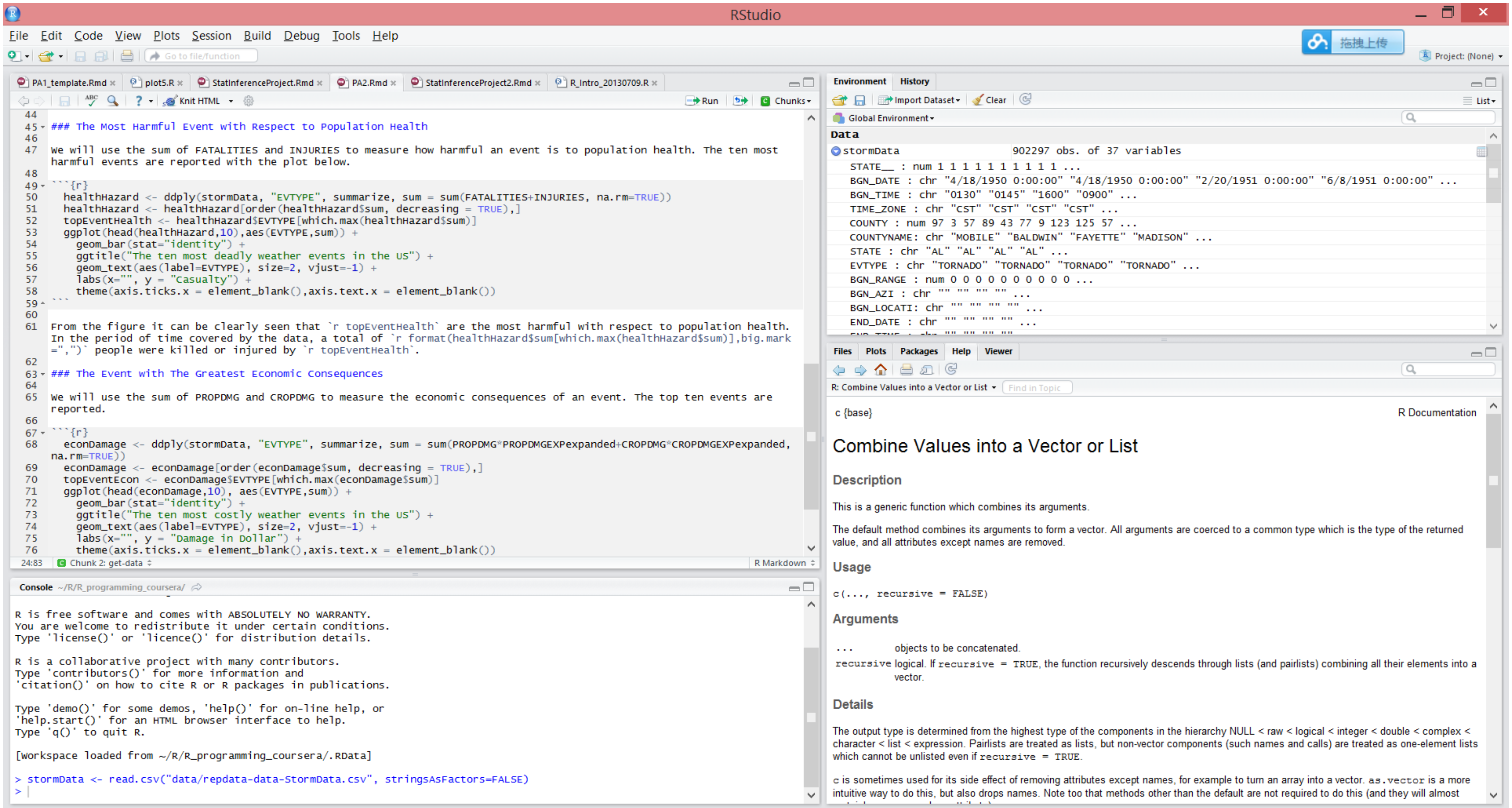
- With an IDE (Integrated Development Environment)
  - *Rstudio* is the de facto environment for R on a desktop system
- On a cluster
  - R is installed on all LONI and LSU HPC clusters
    - QB2: `r/3.1.0/INTEL-14.0.2`
    - SuperMIC: `r/3.1.0/INTEL-14.0.2`
    - Philip: `r/3.1.3/INTEL-15.0.3`
    - SuperMike2: `+R-3.2.0-gcc-4.7.2`

# Rstudio

- Free to use
- Similar user interface to other, 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

LSU

LSU INFORMATION TECHNOLOGY SERVICES




LSU CENTER FOR COMPUTATION & TECHNOLOGY

# On LONI and LSU HPC Clusters

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

```
[lyan1@qb1 ~]$ module add r
[lyan1@qb1 ~]$ R

R version 3.1.0 (2014-04-10) -- "Spring Dance"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-unknown-linux-gnu (64-bit)

…

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.

> getwd()
[1] "/home/lyan1"
> x <- 5
> x
[1] 5
>
Save workspace image? [y/n/c]: n

[lyan1@qb1 ~]$ cat hello.R
print("Hello World!")
[lyan1@qb1 ~]$ Rscript hello.R
[1] "Hello World!"
```

# Getting Help

- Command line
  - `?<command name>`
  - `??<part of command name/topic>`
  - `help(<function name>)`
- Or search in the help page in Rstudio

# Outline

- R basics
  - How to run R codes
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis case study: NOAA weather hazard data
  - Data acquisition and inspection
  - Data preprocessing
  - Data analysis
  - Report generation

# Data Classes

- R has five atomic classes
  - **Numeric** (double)
    - Numbers in R are treated as numeric unless specified otherwise.
  - **Integer**
  - **Complex**
  - **Character**
  - **Logical**
    - TRUE or FALSE
- Derivative classes
  - **Factor**
  - **Date and time**

# 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)
    - Coercion will occur when mixed objects are passed to the `c()` function, as if the `as.<Type>()` function is explicitly called
  - Using the `vector()` function
- One can use `[<index>]` to access individual element
  - Indices start from 1

```
# “#” indicates comment
# “<-” performs assignment operation (you can use “=“ as well, but
“<-” is preferred)
# Variable names are case sensitive (even for Rstudio on Windows!)

# numeric (double is the same as numeric)
> d <- c(1,2,3)
# print its content
> d
[1] 1 2 3

# character
> d <- c("1","2","3")
> d
[1] "1" "2" "3"

# you can covert at object with as.TYPE() functions
# For example, as.numeric() changes the argument to numeric
> as.numeric(d)
[1] 1 2 3

# The conversion doesn't always work though
> as.numeric("a")
[1] NA
Warning message:
NAs introduced by coercion
```

```
> x <- c(TRUE, FALSE) ## logical
> x <- c(T, F) ## logical
> x <- c("a", "b", "c") ## character
# The ":" operator can be used to generate an integer sequence (can
be either ascending or descending)
> x <- 9:6 ## integer vector (9,8,7,6)

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

# Coercion will occur when objects of different classes are mixed
> y <- c(1.7, "a") ## character
> y <- c(TRUE, 2) ## numeric
> y <- c("a", TRUE) ## character

# Can also coerce explicitly
> x <- 0:6
> class(x)
[1] "integer"
> as.logical(x)
[1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
# Example of vector indexing

> 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
```

# Vectorized Operations in R

- Lots of R operations process objects in a vectorized way
  - more efficient, concise, and easier to read.

```
> x <- 1:4; y <- 6:9
> x + y
[1] 7 9 11 13
> x > 2
[1] FALSE FALSE TRUE TRUE
> x * y
[1] 6 14 24 36
> x[x >= 3]
[1] 3 4
```

# Data Objects - Matrices

- Matrices are vectors with a dimension attribute
- R matrices can be constructed by
  - Using the `matrix()` function
  - Passing an `dim` attribute to a vector
  - Using the `cbind()` or `rbind()` functions
- R matrices are constructed column-wise
- One can use `[<index>,<index>]` to access individual element

CENTER FOR COMPUTATION & TECHNOLOGY

```
# Create a matrix using the matrix() function
> m <- matrix(1:6, nrow = 2, ncol = 3)
> m
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
> dim(m)
[1] 2 3
> attributes(m)
$dim
[1] 2 3

# Pass a dim attribute to a vector to recast it to be a matrix
> 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
```

CENTER FOR
& TECHNOLOGY

```
# Row binding and column binding
> 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

# Slicing
# What we saw in vector indexing works here too
> m <- 1:10
> m[c(1,2),c(2,4)]
[,1] [,2]
[1,] 3 7
[2,] 4 8
```

LSU CENTER FOR COMPUTATION & TECHNOLOGY

# 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))
# The statement below is equivalent to the one above – argument names can be omitted as long as the order is maintained
> m <- array(1:8,c(2,2,2))
```

# Data Objects - Lists

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

LSU CENTER FOR COMPUTATION & TECHNOLOGY

```
# Use the list() function to construct a list
> x <- list(1, "a")
> y <- list("b",Sys.time())
> list_of_lists = list(x,y)
> list_of_lists
[[1]]
[[1]][[1]]
[1] 1

[[1]][[2]]
[1] "a"


[[2]]
[[2]][[1]]
[1] "b"

[[2]][[2]]
[1] "2016-03-01 16:05:59 CST"
```

LSU CENTER FOR COMPUTATION & TECHNOLOGY

# Element Names

- Elements of R objects can have names

```
> names(list_of_lists)
NULL
> names(list_of_lists) <- c("x","y")
## function str() displays structure of a R
object
> str(list_of_lists)
List of 2
 $ x:List of 2
  ..$ : num 1
  ..$ : chr "a"
 $ y:List of 2
  ..$ : chr "b"
  ..$ : POSIXct[1:1], format: "2016-03-01
16:05:59"
```

```
# Names can be used as indices
> x <- 1:3
> names(x) <- c("a","b","c")
> x["b"]
[1] 2

# Pass names to elements when creating a list
> y <- list(a = 1, b = F, x = x)
> y
$a
[1] 1

$b
[1] FALSE

$x
a b c
1 2 3

# Names can be used to refer to individual element using the
"$" operator (does not work for vectors, matrices and arrays)
> y$x["a"]
[1] 1
```

LSU CENTER FOR COMPUTATION & TECHNOLOGY

# Data Objects - Data Frames

- Data frames are used to store tabular data
  - They are a special type of lists where every element (i.e. column) has to be of the same length, but can be of different class
  - Data frames can store different classes of objects in each column
  - 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
  - Can be converted to a matrix by calling `data.matrix()`

```
# The "mtcars" data frame is part of the "datasets" package of R.
# It comprises fuel consumption and 10 aspects of design and performance
# for 32 automobiles.
> mtcars
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
……
> str(mtcars)
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
…
# Can use names and indices to access individual elements
> mtcars["Mazda RX4","cyl"]
[1] 6
> mtcars[1,2]
[1] 6
```

# Querying Object Attributes

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

# Outline

- R basics
  - How to run R codes
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis case study: NOAA weather hazard data
  - Data acquisition and inspection
  - Data preprocessing
  - Data analysis
  - Report generation

# Flow Control Structures

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

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

# Testing conditions

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

# Example of if…else if…else structure

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")
}
```

# For Loops

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

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

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

# The loop above can be replaced by:
count <- sum(1 - x %% 2)
```

# Outline

- R basics
  - How to run R codes
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis case study: NOAA weather hazard data
  - Data acquisition and inspection
  - Data preprocessing
  - Data analysis
  - Report generation

# Simple Statistic Functions

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

```
> mean(mtcars$mpg)
[1] 20.09062
> which.min(mtcars$mpg)
[1] 15
```

# 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` |

```
# Random generation from a uniform distribution.
> runif(10, 2, 4)
 [1] 2.871361 3.176906 3.157928 2.398450 2.171803 3.954051
3.084317 2.883278
 [9] 2.284473 3.482990
# You can name the arguments in the function call.
> runif(10, min = 2, max = 4)

# Given p value and degree of freedom, find the t-value.
> qt(p=0.975, df = 8)
[1] 2.306004
# The inverse of the above function call
> pt(2.306, df = 8)
[1] 0.9749998
```

# 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, MARGIN, 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
```

```
# x is a 20x10 matrix
> dim(x)
[1] 20 10

# Change the dimensions of x to 2x2x50
> dim(x) <- c(2,2,50)

# Take average over the first two dimensions
> apply(x, c(1, 2), mean)
           [,1]        [,2]
[1,] -0.0763205 -0.01840142
[2,] -0.1125101  0.11393513
> rowMeans(x, dims = 2)
           [,1]        [,2]
[1,] -0.0763205 -0.01840142
[2,] -0.1125101  0.11393513
```

# Other Apply Functions

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

# User Defined Functions

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

```
# Example of user defined function
#
# Syntax
# function_name <- function (arguments) {
#   statement
# }
#
# Define the function
> pow <- function(x, y) {
+    result <- x^y
+}

# Call the function
> c <- pow(4,2)
> c
[1] 16
```

CENTER FOR COMPUTATION & TECHNOLOGY

# Outline

- R basics
  - How to run R codes
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis case study: NOAA weather hazard data
  - Data acquisition and inspection
  - Data preprocessing
  - Data analysis
  - Report generation

# Installing and Loading R Packages - Rstudio

- Installation:
  - Option 1: menu item “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 R and R Packages - Clusters

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

```
[lyan1@qb1 R]$ export R_LIBS_USER=/home/lyan1/packages/R/libraries
[lyan1@qb1 R]$ R

R version 3.1.0 (2014-04-10) -- "Spring Dance"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-unknown-linux-gnu (64-bit)
…

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

# Outline

- R basics
  - How to run R codes
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis case study: NOAA weather hazard data
  - Data acquisition and inspection
  - Data preprocessing
  - Data analysis
  - Report generation

# Steps for Data Analysis

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

# Case Study: NOAA Weather Hazard Data

- Hazardous weather event data from US National Oceanic and Atmospheric Administration
  – Records time, location, damage etc. for all hazardous weather events in theUS between year 1950 and 2011
  – BZ2 compressed CSV data
- Objectives
  – Rank the type of events according to their threat to public health (fatalities plus injuries per occurrence)
    - Report the top 10 types of events
    - Generate a plot for the result

CENTER FOR COMPUTATION & TECHNOLOGY

# Outline

- R basics
  - How to run R codes
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis case study: NOAA weather hazard data
  - Data acquisition and inspection
  - Data preprocessing
  - Data analysis
  - Report generation

# Getting Data

- Display and set current working directory
  - `getwd()` and `setwd()`
- Downloading files from internet
  - `download.file()`
- File manipulation
  - `file.exists()`, `list.files()` and `dir.create()`

```
# Show current directory
> getwd()
[1] "/project/lyan1/R"
# Create a new directory
> dir.create("data")
> getwd()
[1] "/project/lyan1/R"
> setwd("data")
> getwd()
[1] "/project/lyan1/R/data"
# Download the data
>
download.file("https://filestogeaux.lsu.edu/public/download.php?FILE=lyan1/77545by
0Q5E", "repdata-data-StormData.csv.bz2", method="curl")
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 46.8M  100 46.8M    0     0  32.6M      0  0:00:01  0:00:01 --:--:-- 37.2M
# List files in the current directory
> list.files()
[1] "repdata-data-StormData.csv.bz2"
```

# Reading and Writing Data

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

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

# Reading Data with `read.table` (1)

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

# Reading Data with `read.table` (2)

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

# Reading Data with `read.table` (3)

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

```
> stormData <- read.table("repdata-data-StormData.csv.bz2",
        header = T, sep = ',')
```

# Inspecting Data (1)

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

```
> head(stormData)
  STATE__           BGN_DATE BGN_TIME TIME_ZONE COUNTY COUNTYNAME STATE  EVTYPE
1       1  4/18/1950 0:00:00     0130       CST     97     MOBILE    AL TORNADO
2       1  4/18/1950 0:00:00     0145       CST      3    BALDWIN    AL TORNADO
3       1  2/20/1951 0:00:00     1600       CST     57    FAYETTE    AL TORNADO
4       1   6/8/1951 0:00:00     0900       CST     89    MADISON    AL TORNADO
5       1 11/15/1951 0:00:00     1500       CST     43    CULLMAN    AL TORNADO
6       1 11/15/1951 0:00:00     2000       CST     77 LAUDERDALE    AL TORNADO
……
```

# Inspecting Data (2)

```
# Summary of the "stormData" dataframe.
> str(stormData)
'data.frame':   902297 obs. of  37 variables:
 $ STATE__   : num  1 1 1 1 1 1 1 1 1 1 ...
 $ BGN_DATE  : Factor w/ 16335 levels "10/10/1954 0:00:00",..: 6523 6523 4213
11116 1426 1426 1462 2873 3980 3980 ...
 $ BGN_TIME  : Factor w/ 3608 levels "000","0000","00:00:00 AM",..: 212 257 2645
1563 2524 3126 122 1563 3126 3126 ...
 $ TIME_ZONE : Factor w/ 22 levels "ADT","AKS","AST",..: 7 7 7 7 7 7 7 7 7 7 ...
 $ COUNTY    : num  97 3 57 89 43 77 9 123 125 57 ...
 $ COUNTYNAME: Factor w/ 29601 levels "","5NM E OF MACKINAC BRIDGE TO PRESQUE
ISLE LT MI",..: 13513 1873 4598 10592 4372 10094 1973 23873 24418 4598 ...
 $ STATE     : Factor w/ 72 levels "AK","AL","AM",..: 2 2 2 2 2 2 2 2 2 2 ...
```

# Inspecting Data (3)

```
# Statistical summary of the “stormData” dataframe.
> summary(stormData)
    STATE__                        BGN_DATE                   BGN_TIME
 Min.   : 1.0   5/25/2011 0:00:00:   1202   12:00:00 AM: 10163
 1st Qu.:19.0   4/27/2011 0:00:00:   1193   06:00:00 PM:  7350
 Median :30.0   6/9/2011 0:00:00 :   1030   04:00:00 PM:  7261
 Mean   :31.2   5/30/2004 0:00:00:   1016   05:00:00 PM:  6891
 3rd Qu.:45.0   4/4/2011 0:00:00 :   1009   12:00:00 PM:  6703
 Max.   :95.0   4/2/2006 0:00:00 :    981   03:00:00 PM:  6700
                (Other)          :895866    (Other)    :857229
   TIME_ZONE          COUNTY             COUNTYNAME            STATE
 CST    :547493   Min.   :  0.0   JEFFERSON :   7840   TX     : 83728
 EST    :245558   1st Qu.: 31.0   WASHINGTON:   7603   KS     : 53440
 MST    : 68390   Median : 75.0   JACKSON   :   6660   OK     : 46802
 PST    : 28302   Mean   :100.6   FRANKLIN  :   6256   MO     : 35648
 AST    :  6360   3rd Qu.:131.0   LINCOLN   :   5937   IA     : 31069
 HST    :  2563   Max.   :873.0   MADISON   :   5632   NE     : 30271
 (Other):  3631                   (Other)   :862369    (Other):621339
```

# Outline

- R basics
  - How to run R codes
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis case study: NOAA weather hazard data
  - Data acquisition and inspection
  - Data preprocessing
  - Data analysis
  - Report generation

# 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
- Method 1: use indices and names

```
# Extract the 1st, 2nd and 4th observations of variables
MAG, COUNTY and STATE
> stormData[c(1,2,4),c("MAG","COUNTY","STATE")]
  MAG COUNTY STATE
1   0     97    AL
2   0      3    AL
4   0     89    AL
```

# Preprocessing - Subsetting Data (2)

- Method 2: use conditions

```
# Extract the values of MAG, COUNTY and STATE for observations whose value
of MAG is greater than 300

> stormData300 <- stormData[stormData$MAG > 300,c("MAG","COUNTY","STATE")]
> class(stormData300)
[1] "data.frame"
> nrow(stormData300)
[1] 1636
```

# Preprocessing - Subsetting Data (3)

- Method 3: use the `subset` function

```
# Extract the values of MAG, COUNTY and STATE for observations whose value
of MAG is greater than 300


> str(subset(stormData, MAG > 300, select=c(MAG,COUNTY,STATE)))
'data.frame':   1636 obs. of  3 variables:
 $ MAG   : num  350 400 350 400 350 400 400 350 350 800 ...
 $ COUNTY: num  25 91 97 9 97 65 65 125 143 65 ...
 $ STATE : Factor w/ 72 levels "AK","AL","AM",..: 2 2 2 7 5 5 5 5 5 5 ...
```

# Dealing with 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
  - `is.nan()` is used to test for NaN
  - NA values have a class also, so there are integer NA, character NA, etc.
  - A `NaN` value is also NA but the converse is not true
- The `complete.cases()` function can be used to identify complete observations
- Many R functions have a logical "`na.rm`" option
  - `na.rm=TRUE` means the NA values should be discarded
- Note: Not all missing values are marked with "NA" in raw data!

```
# Extract the values of EVTYPE, FATALITIES and
# INJURIES for observations whose EVTYPE is not “?”.
# Here the missing value is not represented by NA or
# NaN.

> healthDamage <- subset(stormData, EVTYPE != "?",
select=c(EVTYPE,FATALITIES,INJURIES))
> head(healthDamage)
   EVTYPE FATALITIES INJURIES
1 TORNADO          0       15
2 TORNADO          0        0
3 TORNADO          0        2
4 TORNADO          0        2
5 TORNADO          0        2
6 TORNADO          0        6
```

# Outline

- R basics
  - How to run R codes
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis case study: NOAA weather hazard data
  - Data acquisition and inspection
  - Data preprocessing
  - Data analysis
  - Report generation

# Analyzing Data: Split-Apply-Combine

| | EVTYPE | FATALITIES | INJ |
|---|---|---|---|
| 1 | A | 0 | 15 |
| 2 | A | 0 | 0 |
| 3 | A | 0 | 2 |
| 4 | B | 0 | 2 |
| 5 | B | 0 | 2 |
| 6 | B | 0 | 6 |
| 7 | C | 0 | 15 |
| 8 | C | 0 | 0 |
| 9 | C | 0 | 2 |

| | EVTYPE | FATALITIES | INJ |
|---|---|---|---|
| 1 | A | 0 | 15 |
| 2 | A | 0 | 0 |
| 3 | A | 0 | 2 |

| | EVTYPE | FATALITIES | INJ |
|---|---|---|---|
| 4 | B | 0 | 2 |
| 5 | B | 0 | 2 |
| 6 | B | 0 | 6 |

| | EVTYPE | FATALITIES | INJ |
|---|---|---|---|
| 7 | C | 0 | 15 |
| 8 | C | 0 | 0 |
| 9 | C | 0 | 2 |

| EVTYPE | AVG_CASUALTIES |
|---|---|
| A | 5.7 |
| B | 3.3 |
| C | 5.7 |

# Analyzing Data: Split-Apply-Combine

| | EVTYPE | FATALITIES | INJ |
|---|---|---|---|
| 1 | A | 0 | 15 |
| 2 | A | 0 | 0 |
| 3 | A | 0 | 2 |
| 4 | B | 0 | 2 |
| 5 | B | 0 | 2 |
| 6 | B | 0 | 6 |
| 7 | C | 0 | 15 |
| 8 | C | 0 | 0 |
| 9 | C | 0 | 2 |

**Split**

| | EVTYPE | FATALITIES | INJ |
|---|---|---|---|
| 1 | A | 0 | 15 |
| 2 | A | 0 | 0 |
| 3 | A | 0 | 2 |

| | EVTYPE | FATALITIES | INJ |
|---|---|---|---|
| 4 | B | 0 | 2 |
| 5 | B | 0 | 2 |
| 6 | B | 0 | 6 |

| | EVTYPE | FATALITIES | INJ |
|---|---|---|---|
| 7 | C | 0 | 15 |
| 8 | C | 0 | 0 |
| 9 | C | 0 | 2 |

**Apply**

**Combine**

| EVTYPE | AVG_CASUALTIES |
|---|---|
| A | 5.7 |
| B | 3.3 |
| C | 5.7 |

# Analyzing Data: Split-Apply-Combine

- In data analysis you often need to **split** up a big data structure into homogeneous pieces, **apply** a function to each piece and then **combine** all the results back together
- This split-apply-combine procedure is what the `plyr` package is for.

```
> library(plyr)
# Use the ddply() function to perform split-apply-merge
> healthByType <- ddply(healthDamage, "EVTYPE", summarize,
casualty=sum(FATALITIES+INJURIES), freq=length(EVTYPE),
perEvt=casualty/freq)
> head(healthByType)
                EVTYPE casualty freq perEvt
1        ABNORMALLY DRY        0    2      0
2        ABNORMALLY WET        0    1      0
3       ABNORMAL WARMTH        0    4      0
4 ACCUMULATED SNOWFALL        0    4      0
5   AGRICULTURAL FREEZE        0    6      0
6         APACHE COUNTY        0    1      0
# Sort the result and get the top 10 events
> healthByType[order(healthByType$perEvt,decreasing=TRUE),][1:10,]
                        EVTYPE casualty freq    perEvt
272                  Heat Wave       70    1 70.00000
846       TROPICAL STORM GORDON       51    1 51.00000
954                 WILD FIRES      153    4 38.25000
755              THUNDERSTORMW       27    1 27.00000
832 TORNADOES, TSTM WIND, HAIL       25    1 25.00000
359         HIGH WIND AND SEAS       23    1 23.00000
274          HEAT WAVE DROUGHT       19    1 19.00000
645            SNOW/HIGH WINDS       36    2 18.00000
973   WINTER STORM HIGH WINDS       16    1 16.00000
405          HURRICANE/TYPHOON     1339   88 15.21591
```

# Parallel Processing in R

- `doParallel` package
  - Used as a parallel backend by other packages such as `foreach` and `plyr`
  - **Tutorial: Parallel computing with R, March 22nd, 2017**

```
# Sequential
> system.time(foreach(i=1:4) %do% rnorm(1e8))
   user  system elapsed
 33.512   0.432  33.948

# Paralle with 4 workers
> library(doParallel)
> cl <- makeCluster(4)
> registerDoParallel(cl)
> system.time(foreach(i=1:4) %dopar% rnorm(1e8))
   user  system elapsed
  1.090   1.491  12.439
> stopCluster(cl)
```

```
## Sequential ddply
> system.time(healthByType <- ddply(healthDamage, "EVTYPE",
casualty=sum(FATALITIES+INJURIES)))
   user  system elapsed
  2.849   0.091   2.940

## Parallel ddply
> library(doParallel)
> cl <- makeCluster(4)
> registerDoParallel(cl)
> system.time(healthByType <- ddply(healthDamage, "EVTYPE",
casualty=sum(FATALITIES+INJURIES), .parallel=TRUE))
   user  system elapsed
  2.294   0.023   2.317

## In this example the sequential version does not take much
## time and the parallel version fails to speed it up
## significantly to complete due to overhead.
```

# Put Everything Together

- Run R commands in batch mode with `Rscript`

```
[lyan1@philip025 R]$ cat noaa_analysis.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("data/repdata-data-StormData.csv.bz2")) {

download.file("https://d396qusza40orc.cloudfront.net/repdata%2Fdata%2FStormData.csv.bz2"
                , "data/repdata-data-StormData.csv.bz2", method="curl")
}
…

[lyan1@philip025 R]$ Rscript noaa_analysis.R
```

# Outline

- R basics
  - How to run R codes
  - Data classes and objects in R
  - Flow control structures
  - Functions
  - How to install and load R packages
- Data analysis case study: NOAA weather hazard data
  - Data acquisition and inspection
  - Data preprocessing
  - Data analysis
  - Report generation

# Report Generation with R Markdown

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

```
[lyan1@philip1 R]$ cat noaa_analysis.Rmd
…
### The Most Harmful Event with Respect to Population Health

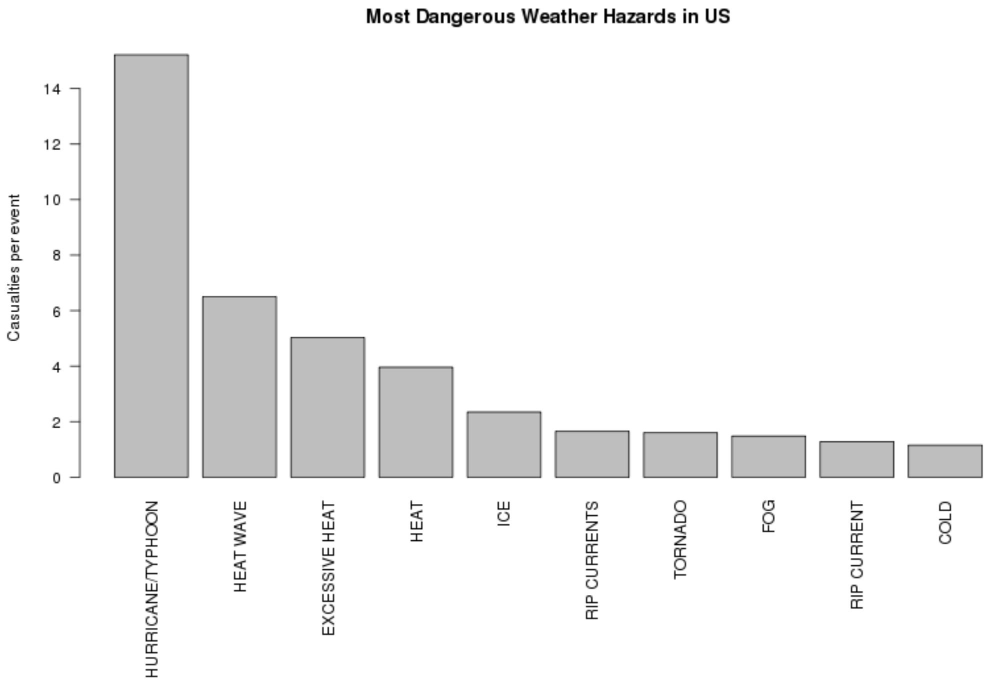
We will use casualties per event (the sum of FATALITIES and
INJURIES divided by the number of occurrence) to measure how
harmful an event is to population health. The ten most harmful
events are reported with the plot below.

```{r}
# Split-apply-combine: calculate total casualties,
# frequency and casualties per event for all event types.
healthByType <- ddply(healthDamage, "EVTYPE", summarize,
casualty=sum(FATALITIES+INJURIES), freq=length(EVTYPE),
perEvt=casualty/freq)
…
[lyan1@philip1 R]$ Rscript -e "library(knitr);
stitch('noaa_analysis.Rmd')"
```
```

# Graphics in R

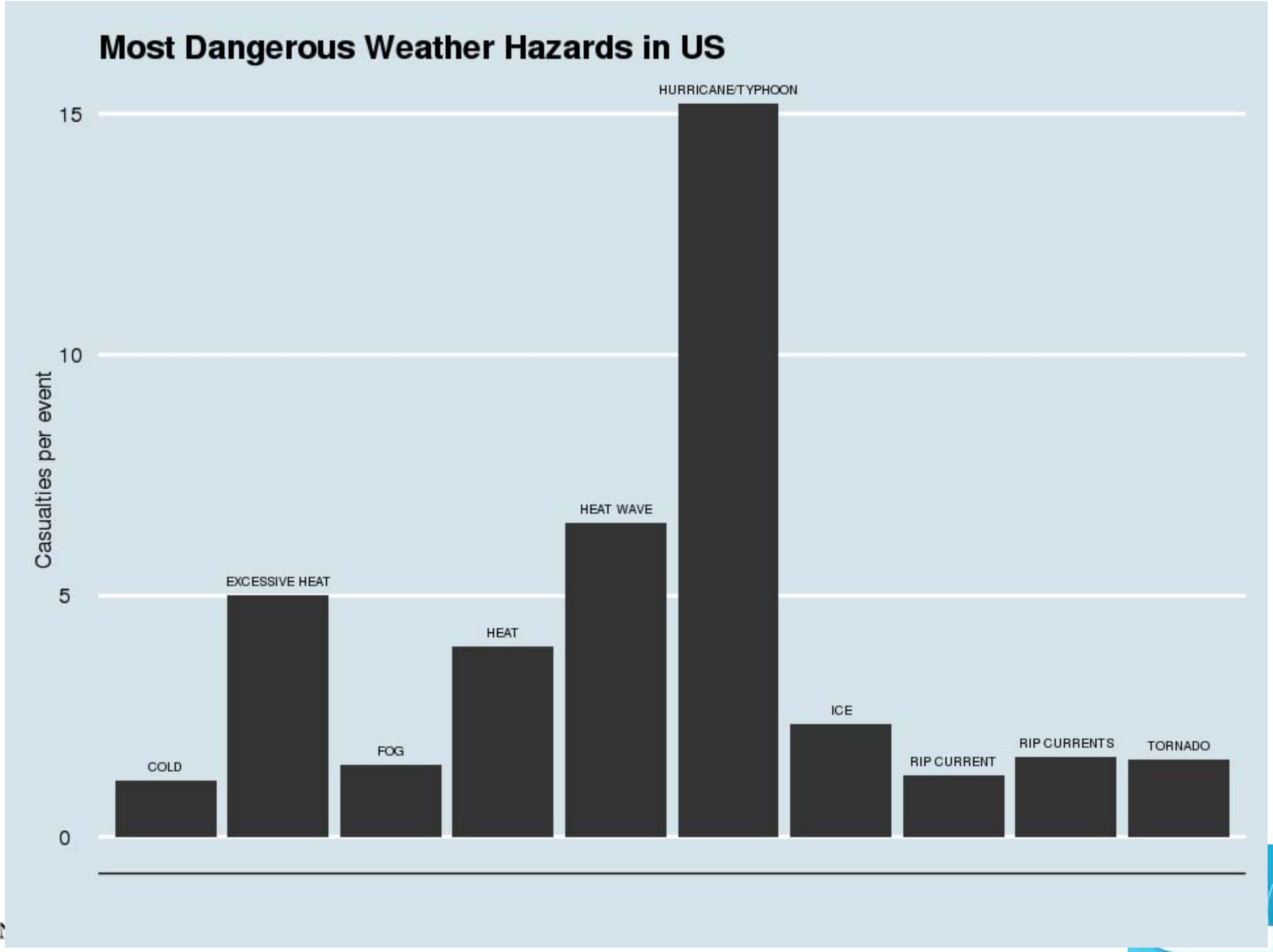
- There are three plotting systems in R
  – base
    - Convenient, but hard to adjust after the plot is created
  – lattice
    - Good for creating conditioning plot
  – ggplot2
    - Powerful and flexible, many tunable feature, may require some time to master
- Each has its pros and cons, so it is up to the users which one to choose

# Barplot - Base




```
barplot(top10Evts$perEvt, names.arg=top10Evts$EVTYPE,
        main="Most Dangerous Weather Hazards in US",
       ylab="Casualties per event", las=2)
```

# Barplot – ggplot2



# Barplot – ggplot2

```
ggplot(top10Evts, aes(EVTYPE, perEvt)) +
        geom_bar(stat="identity") +
        ggtitle("Most Dangerous Weather Hazards in US") +
        geom_text(aes(label=EVTYPE), size=2, vjust=-1) +
        labs(x="", y="Casualties per event") +
        theme_economist() + scale_colour_economist() +
        theme(axis.ticks.x = element_blank(),
               axis.text.x = element_blank())
```

# Not Covered

- Statistical analysis (e.g regression models, machine learning/data mining)
- Profiling and debugging
- …
- Chances are that R has something in store for you whenever it comes to data analysis

# Learning R

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

# Next Tutorial – Introduction to Python

- This training will provide a brief introduction to the python programming language, introduce you to some useful python modules for system management and scientific computing.
- Date: March 15th, 2017

# Getting Help

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

# Questions?