# Introduction to R

Le Yan

HPC User Services @ LSU

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

# Outline

- R basics
- Case study: NOAA weather hazard data

# The 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)
- R was first announced in 1993
- 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 3.2.2

# Features of R

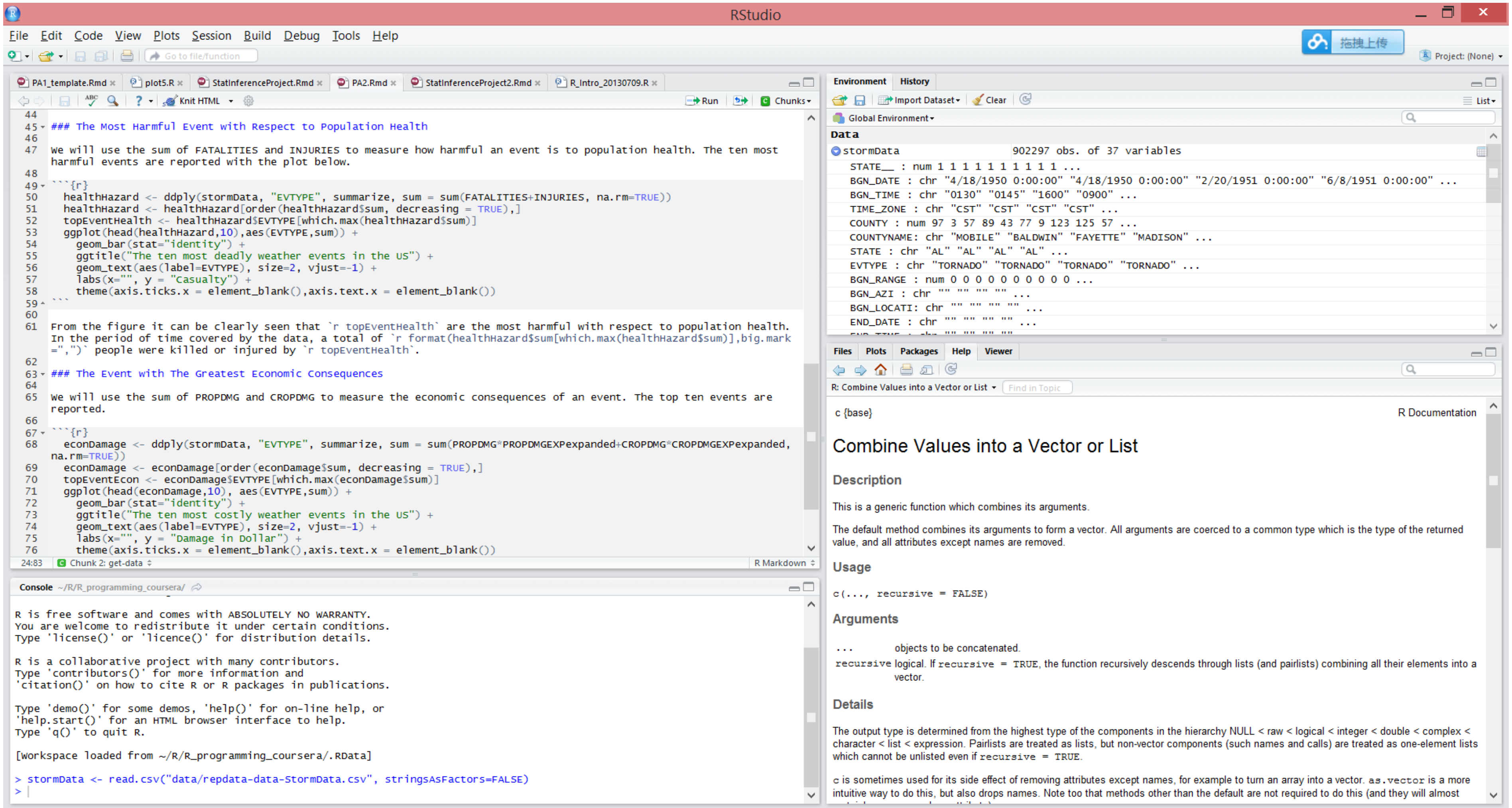
- R is a dialect of the S language
  - Language designed for statistical analysis
  - Similar syntax
- Available on most platform/OS
- Rich data analysis functionalities and sophisticated graphical capabilities
- Active development and very active community
  - CRAN: The **C**omprehensive **R** **A**rchive **N**etwork
    - Source code and binaries, user contributed packages and documentation
  - More than 6,000 packages available on CRAN (as of March 2015)
- Free to use

# Two Ways of Running R

- With an IDE
  - 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 IDEs or software such as Matlab; provides panes for
  - Source code
  - Console
  - Workspace
  - Others (help message, plot etc.)
- Rstudio in a desktop environment is better suited for development and/or a limited number of small jobs


RStudio
File Edit Code View Plots Session Build Debug Tools Help
PA1_template.Rmd
plot5.R
StatInferenceProject.Rmd
PA2.Rmd
StatInferenceProject2.Rmd
R_Intro_20130709.R
Knit HTML
Run
Chunks
### The Most Harmful Event with Respect to Population Health
We will use the sum of FATALITIES and INJURIES to measure how harmful an event is to population health. The ten most harmful events are reported with the plot below.
```{r}
healthHazard <- ddply(stormData, "EVTYPE", summarize, sum = sum(FATALITIES+INJURIES, na.rm=TRUE))
healthHazard <- healthHazard[order(healthHazard$sum, decreasing = TRUE),]
topEventHealth <- healthHazard$EVTYPE[which.max(healthHazard$sum)]
ggplot(head(healthHazard,10),aes(EVTYPE,sum)) +
geom_bar(stat="identity") +
ggtitle("The ten most deadly weather events in the US") +
geom_text(aes(label=EVTYPE), size=2, vjust=-1) +
labs(x="", y = "Casualty") +
theme(axis.ticks.x = element_blank(),axis.text.x = element_blank())
```
From the figure it can be clearly seen that `r topEventHealth` are the most harmful with respect to population health. In the period of time covered by the data, a total of `r format(healthHazard$sum[which.max(healthHazard$sum)],big.mark =",")` people were killed or injured by `r topEventHealth`.
### The Event with The Greatest Economic Consequences
We will use the sum of PROPDMG and CROPDMG to measure the economic consequences of an event. The top ten events are reported.
```{r}
econDamage <- ddply(stormData, "EVTYPE", summarize, sum = sum(PROPDMG*PROPDMGEXPexpanded+CROPDMG*CROPDMGEXPexpanded, na.rm=TRUE))
econDamage <- econDamage[order(econDamage$sum, decreasing = TRUE),]
topEventEcon <- econDamage$EVTYPE[which.max(econDamage$sum)]
ggplot(head(econDamage,10), aes(EVTYPE,sum)) +
geom_bar(stat="identity") +
ggtitle("The ten most costly weather events in the US") +
geom_text(aes(label=EVTYPE), size=2, vjust=-1) +
labs(x="", y = "Damage in Dollar") +
theme(axis.ticks.x = element_blank(),axis.text.x = element_blank())
Console ~/R/R_programming_coursera/
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[workspace loaded from ~/R/R_programming_coursera/.RData]
> stormData <- read.csv("data/repdata-data-StormData.csv", stringsAsFactors=FALSE)
Environment History
Import Dataset
Clear
Global Environment
Data
stormData 902297 obs. of 37 variables
STATE__ : num 1 1 1 1 1 1 1 1 1 1 ...
BGN_DATE : chr "4/18/1950 0:00:00" "4/18/1950 0:00:00" "2/20/1951 0:00:00" "6/8/1951 0:00:00" ...
BGN_TIME : chr "0130" "0145" "1600" "0900" ...
TIME_ZONE : chr "CST" "CST" "CST" "CST" ...
COUNTY : num 97 3 57 89 43 77 9 123 125 57 ...
COUNTYNAME: chr "MOBILE" "BALDWIN" "FAYETTE" "MADISON" ...
STATE : chr "AL" "AL" "AL" "AL" ...
EVTYPE : chr "TORNADO" "TORNADO" "TORNADO" "TORNADO" ...
BGN_RANGE : num 0 0 0 0 0 0 0 0 0 0 ...
Files Plots Packages Help Viewer
c {base}
R Documentation
Combine Values into a Vector or List
Description
This is a generic function which combines its arguments.
The default method combines its arguments to form a vector. All arguments are coerced to a common type which is the type of the returned value, and all attributes except names are removed.
Usage
c(..., recursive = FALSE)
Arguments
... objects to be concatenated.
recursive logical. If recursive = TRUE, the function recursively descends through lists (and pairlists) combining all their elements into a vector.
Details
The output type is determined from the highest type of the components in the hierarchy NULL < raw < logical < integer < double < complex < character < list < expression. Pairlists are treated as lists, but non-vector components (such names and calls) are treated as one-element lists which cannot be unlisted even if recursive = TRUE.
```

# On LONI and LSU HPC Clusters

- Two modes to run R on clusters
  - Interactive mode
    - Type R command to enter the console, then run R commands there
  - Batch mode
    - Write the R script first, then submit a batch job to run it (use the `Rscript` command)
    - This is 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

# Data Classes

- R has five atomic classes
  - Numeric
    - Double is equivalent to numeric.
    - Numbers in R are treated as numeric unless specified otherwise.
  - Integer
  - Complex
  - Character
  - Logical
    - TRUE or FALSE
- You can convert data from one type to the other using the `as.<Type>` functions

# Data Objects - Vectors

- Vectors can only contain elements of the same 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)

# numeric (double is the same as numeric)
> d <- c(1,2,3)
> 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(0.5, 0.6) ## numeric
> x <- c(TRUE, FALSE) ## logical
> x <- c(T, F) ## logical
> x <- c("a", "b", "c") ## character
# The ":" operator can be used to generate integer sequences
> x <- 9:29 ## integer
> x <- c(1+0i, 2+4i) ## complex

> 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
```

# Vectorized Operations

- 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
> print( x[x >= 3] )
[1] 3 4
```

# Data Objects - Matrices

- Matrices are vectors with a dimension attribute
- R matrices can be constructed
  - 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

```
# 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
> 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
```

```
# 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
> m <- 1:10
> m[c(1,2),c(2,4)]
[,1] [,2]
[1,] 3 7
[2,] 4 8
```

# Data Objects - Lists

- Lists are a special kind of vector that contains objects of different classes
- Lists can be constructed by using the `list()` function
- Lists can be indexed using `[[]]`

LSU CENTER FOR COMPUTATION & TECHNOLOGY

```
# Use the list() function to construct a list
> x <- list(1, "a", TRUE, 1 + 4i)
> x
[[1]]
[1] 1

[[2]]
[1] "a"

[[3]]
[1] TRUE
```

# Names

- R objects can have names

```
# Each element in a vector can have a name
> x <- 1:3
> names(x)
NULL
> names(x) <- c("a","b","c")
> names(x)
[1] "a" "b" "c"
> x
a b c
1 2 3
```

```
# Lists
> x <- list(a = 1, b = 2, c = 3)
> x
$a
[1] 1

$b
[1] 2

$c
[1] 3

# Names can be used to refer to individual element
> x$a
[1] 1

# Columns and rows of matrices
> m <- matrix(1:4, nrow = 2, ncol = 2)
> dimnames(m) <- list(c("a", "b"), c("c", "d"))
> m
  c d
a 1 3
b 2 4
```

# Data Objects - Data Frames

- Data frames are used to store tabular data
  - They are a special type of list where every element of the list has to have the same length
  - Each element of the list can be thought of as a column
  - Data frames can store different classes of objects in each column
  - Data frames also have a special attribute called `row.names`
  - Data frames are usually created by calling `read.table()` or `read.csv()`
    - More on this later
  - Can be converted to a matrix by calling `data.matrix()`

```
> 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 ...
…
> 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 (does not work with vectors)
  - Class
  - Names
  - Dimensions
  - Length
  - User defined attributes
- They work on all objects (including functions)

```
> m <- matrix(1:10, nrow = 2, ncol = 5)
> str(matrix)
function (data = NA, nrow = 1, ncol = 1, byrow = FALSE,
dimnames = NULL)
> str(m)
 int [1:2, 1:5] 1 2 3 4 5 6 7 8 9 10

> str(matrix)
function (data = NA, nrow = 1, ncol = 1, byrow = FALSE,
dimnames = NULL)

> str(str)
function (object, ...)
```

# Data Class - Factors

- Factors are used to represent categorical data.
- Factors can be unordered or ordered.
- Factors are treated specially by modelling functions like `lm()` and `glm()`

```
# Use the factor() function to construct a vector of factors
# The order of levels can be set by the levels keyword
> x <- factor(c("yes", "yes", "no", "yes", "no"),
levels = c("yes", "no"))
> x
[1] yes yes no yes no
Levels: yes no
```

# Data Class - Date and Time

- R has a Date class for date data while times are represented by POSIX formats
- One can convert a text string to date using the `as.Date()` function
- The `strptime()` function can deal with dates and times in different formats.
- The package "`lubridate`" provides many additional and convenient features

```
# Dates are stored internally as the number of days since 1970-01-01
> x <- as.Date("1970-01-01")
> x
[1] "1970-01-01"
> as.numeric(x)
[1] 0
> x+1
[1] "1970-01-02“

# Tmes are stored internally as the number of seconds since 1970-01-01
> x <- Sys.time()
> x
[1] "2015-03-17 09:40:43 CDT"
> as.numeric(x)
[1] 1426603244
> p <- as.POSIXlt(x)
> names(unclass(p))
 [1] "sec"    "min"    "hour"   "mday"   "mon"    "year"   "wday"   "yday"
 [9] "isdst"  "zone"   "gmtoff"
> p$sec
[1] 43.88181
```

# Simple Statistic Functions

| | |
|---|---|
| `min()` | Minimum value |
| `max()` | Maximum value |
| `which.min()` | Location of minimum value |
| `which.max()` | Location of maximum value |
| `pmin()` | Element-wise minima of several vectors |
| `pmax()` | Element-wise maxima of several vectors |
| `sum()` | Sum of the elements of a vector |
| `mean()` | Mean of the elements of a vector |
| `prod()` | Product of the elements of a vector |

```
> dim(x)
[1]  2  2 50
> min(x)
[1] -2.665878
> which.min(x)
[1] 123
```

# 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
```

# 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

# Control Structures

- Control structures allow one to control the flow of execution.

| | |
|---|---|
| `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: !, &&, ||

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")
}
```

# Outline

- R basics
- Case study: NOAA weather hazard data

# 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

# 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 a report

# 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()`

CENTER FOR COMPUTATION & TECHNOLOGY

```
> getwd()
[1] "/project/lyan1/R"
> dir.create("data")
> getwd()
[1] "/project/lyan1/R"
> setwd("data")
> getwd()
[1] "/project/lyan1/R/data"
> download.file("https://tigerbytes2.lsu.edu/users/hpctraining/web/2015-
Fall/repdata-data-StormData.csv.bz2", "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()
[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 |
| `unserialize` | `serialize` | for reading/writing single R objects in binary form |

CENT
& TECHNOLOGY

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

```
> 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)
```

LSU CENTER FOR COMPUTATION & TECHNOLOGY

# 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 a #
  - 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 = ',')
```

CENTER FOR COMPUTATION & TECHNOLOGY

# Viewing Data Information

- `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
……
```

# Viewing Data Information

```
> 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 ...
```

# Summarizing Data

- `summary` function:

```
> 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
```

# Subsetting Data (1)

- There are a number of different ways of extracting a subset of R objects
- Using indices and names

```
> stormData[c(1,2,4),c("MAG","COUNTY","STATE")]
  MAG COUNTY STATE
1   0     97    AL
2   0      3    AL
4   0     89    AL
```

LSU CENTER FOR COMPUTATION & TECHNOLOGY

# Subsetting Data (2)

- Using conditions

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

# Subsetting Data (3)

- Using the subset function

```
> 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
- Many R functions have a logical “na.rm” option
  - `na.rm=`TRUE means the NA values should be discarded
- Not all missing values are marked with “NA” in raw data

```
> 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
```

# 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
```

```
> dim(x)
[1] 20 10

# Change the dimensions of x
> 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`

# 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.

# Split-Apply-Combine

| | mpg | cyl | gear |
|---|---|---|---|
| Mazda RX4 | 21.0 | 6 | 4 |
| Mazda RX4 Wag | 21.0 | 6 | 4 |
| Datsun 710 | 22.8 | 4 | 4 |
| Hornet 4 Drive | 21.4 | 6 | 3 |
| Hornet Sportabout | 18.7 | 8 | 3 |
| Valiant | 18.1 | 6 | 3 |
| Duster 360 | 14.3 | 8 | 3 |
| Merc 240D | 24.4 | 4 | 4 |
| Merc 230 | 22.8 | 4 | 4 |
| Merc 280 | 19.2 | 6 | 4 |
| Merc 280C | 17.8 | 6 | 4 |
| Merc 450SE | 16.4 | 8 | 3 |
| Merc 450SL | 17.3 | 8 | 3 |
| Merc 450SLC | 15.2 | 8 | 3 |

| | cyl | gear | avgMPG |
|---|---|---|---|
| 1 | 4 | 3 | 21.500 |
| 2 | 4 | 4 | 26.925 |
| 3 | 4 | 5 | 28.200 |
| 4 | 6 | 3 | 19.750 |
| 5 | 6 | 4 | 19.750 |
| 6 | 6 | 5 | 19.700 |
| 7 | 8 | 3 | 15.050 |
| 8 | 8 | 5 | 15.400 |

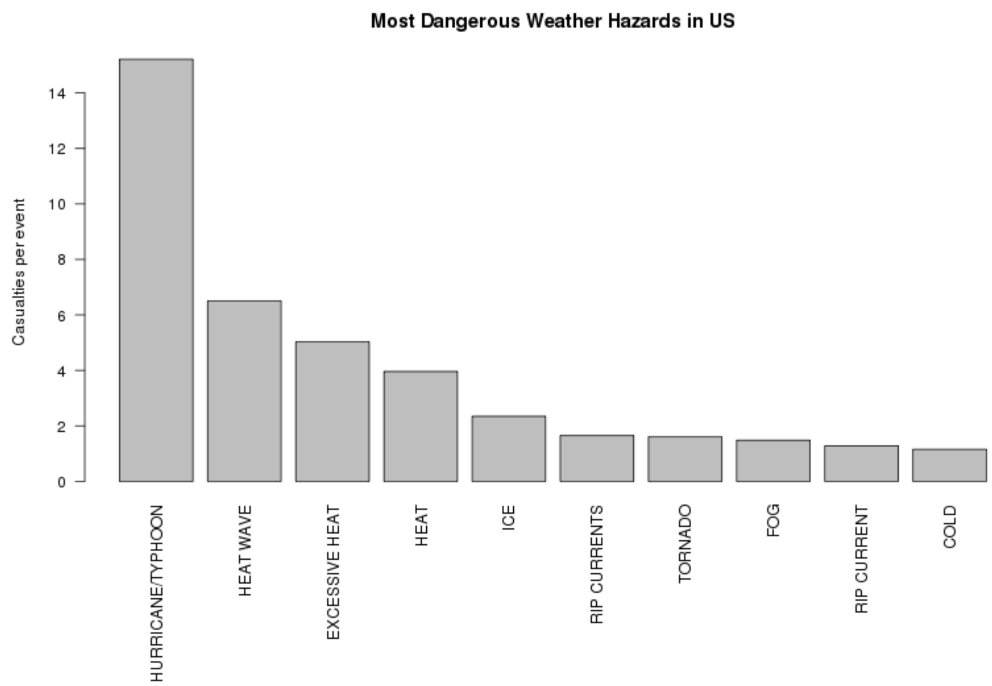
```
ddply(mtcars, c("cyl","gear"),avgMPG=mean(mpg))
```

```
> library(plyr)
> 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
> 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
```

# Graphics in R

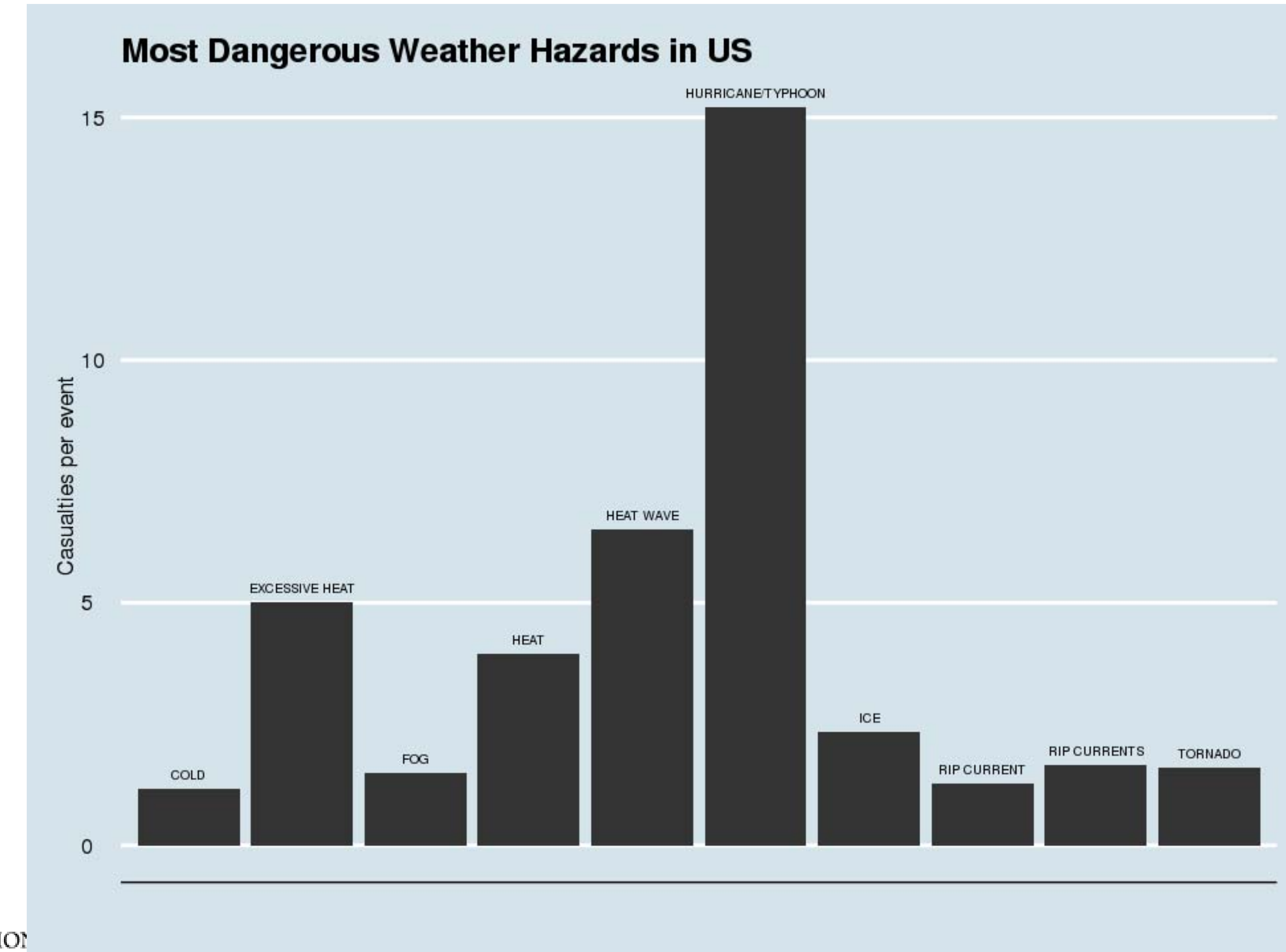
- There are three plotting systems in R
  - Base
    - Convenient, but hard to adjust after the plot is created
  - Lattice
    - Good for creating conditioning plot
  - Ggplot2
    - Powerful and flexible, many tunable feature, may require some time to master
- Each has its pros and cons, so it is up 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())
```

# Rscript

- Run R commands in batch mode

```
[lyan1@philip1 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%2FS
tormData.csv.bz2"
                , "data/repdata-data-StormData.csv.bz2", method="curl")
}
…

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

# Data Analysis with Reporting

- `knitr` is a R package that allows one to generate dynamic report by weaving R code and human readable texts together
  - It uses the markdown syntax
  - The output can be HTML, PDF or (even) Word

LSU

LSU INFORMATION TECHNOLOGY SERVICES

````
---
title: "NOAA Weather Hazard Events Analysis"
author: "Le Yan"
date: "Oct 28, 2015"
output: pdf_document
---

## Overview

Storms and other severe weather events can cause severe public health problems for communities and municipalities. Many severe events can result in fatalities and injuries, and preventing such outcomes to the extent possible is a key concern. The U.S. National Oceanic and Atmospheric Administration's (NOAA) storm database tracks characteristics of major storms and weather events in the United States, including when and where they occur, as well as estimates of any fatalities, injuries, and property damage. In this project, we will use a data set from the NOAA database to find out which event types have the most significant consequences on population health.

## Data Processing

Load some R packages:

```{r}
library(plyr)
library(ggplot2)
library(ggthemes)
```

Download the data:

```{r get-data, cache=TRUE}

# 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")
}

```

Read the data:

```{r read-data, cache=TRUE}
stormData <- read.table("data/repdata-data-StormData.csv.bz2",
        header = T, sep = ',')
```

Extract relevant data:
```{r}
healthDamage <- subset(stormData, EVTYPE != "?", select=c(EVTYPE,FATALITIES,INJURIES))
```

## Results
```

15:1 (Top Level) R Markdown

Console

- Overview
- Data Processing
- Results
- Conclusions


# NOAA Weather Hazard Events Analysis

*Le Yan*

*Oct 28, 2015*

**Overview**

Storms and other severe weather events can cause severe public health problems for communities and municipalities. Many severe events can result in fatalities and injuries, and preventing such outcomes to the extent possible is a key concern. The U.S. National Oceanic and Atmospheric Administration's (NOAA) storm database tracks characteristics of major storms and weather events in the United States, including when and where they occur, as well as estimates of any fatalities, injuries, and property damage. In this project, we will use a data set from the NOAA database to find out which event types have the most significant consequences on population health.

**Data Processing**

Load some R packages:

```
library(plyr)
library(ggplot2)
library(ggthemes)
```

Download the data:

```
# 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")
}
```

Read the data:

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

Extract relevant data:

```
healthDamage <- subset(stormData, EVTYPE != "?", select=c(EVTYPE,FATALITIES,INJURIES))
```

1

**Results**

**The Most Harmful Event with Respect to Population Health**


LSU CENTER FOR COMPUTATION & TECHNOLOGY

# Installing and Loading R Packages

- Installation
  - With R Studio
    - You most likely have root privilege on your own computer
    - Use the `install.packages("<package name>")` function (double quotation is mandatory), or
    - Click on "install packages" in the menu
  - On a cluster
    - You most likely do NOT have root privilege
    - To install a R packages
      - Point the environment variable `R_LIBS_USER` to desired location, then
      - Use the `install.packages` function
- Loading: the `library()` function load previously installed packages

```
[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")
```

# R with HPC

- There are lots of efforts going on to make R run (more efficiently) on HPC platforms
  - http://cran.r-project.org/web/views/HighPerformanceComputing.html

# Not Covered

- Profiling and debugging
- Regression Models
- Machine learning/Data Mining
- …
- 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
  - http://www.cyclismo.org/tutorial/R/
- Online courses (e.g. Coursera)
- Educational R packages
  - Swirl: Learn R in R

# Next Tutorial – HPC in Engineering

- Engineering computation problems often requires the solution of different types of partial differential equations (PDEs) which are highly computational intensive. This training will introduce several open source and commercial engineering packages that run on HPC and LONI clusters including OpenFOAM, Ansys Fluent, DualSPhysics, LIGGGHTS/CFDEM, etc. Representative examples based on Euler, Lagrangian or coupled Euler/Lagrangian methods using these packages will be presented to demonstrate the typical procedures to solve engineering problems with
- Date: November 4th, 2015

# 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
- Online courses: http://moodle.hpc.lsu.edu
- Contact us
  - Email ticket system: sys-help@loni.org
  - Telephone Help Desk: 225-578-0900
  - Instant Messenger (AIM, Yahoo Messenger, Google Talk)
    - Add “lsuhpchelp”

# Questions?