



**PEARSON**



# R Introduction

An introduction to data management, analysis, and  
programming in the R language

Psychometrics conf: 2012

Ou Zhang

# Synopsis

- Introduce some basic programming & statistics with R
- Introduce fundamental R concepts
- Introduce an R programming editor-RStudio
- Introduce some fun parts in R

After this presentation you will be able to

- know the history of R
- know how to compile/run R-scripts in RStudio and R-base
- manipulate data into different formats
- conduct some statistical analysis in R that you would in SAS or SPSS

# History of R

- Statistical programming language S developed at Bell Labs since 1976 (at the same time as UNIX)
- Exclusively licensed to Insightful corp. Product name: S-Plus
- R: Open source platform similar to S developed by **R. Gentleman** and **R. Ihaka** (U of Auckland, NZ) during the 1990s



R. Gentleman



R. Ihaka

- Since 1997: international “R-core” developing team (15 core experts & 1000s code writers and statisticians to share their codes and libraries)
- Updated versions available every couple months

# What R does and does not

- data handling and storage:  
numeric, textual
- matrix algebra
- hash tables and regular expressions
- high-level data analytic and statistical functions
- classes
- graphics
- programming language: loops, branching, subroutines
- is not a database, but connects to DBMSs
- has no graphical user interfaces, but connects to Java, C, C++, Fortran
- language interpreter can be very slow, but allows to call own C/C++ code
- no spreadsheet view of data, but connects to Excel/MsOffice  
(but with RStudio the spreadsheet view works)
- no professional / commercial support

# Advantages

- Fast and free.
- State of the art: Statistical researchers provide their methods as R packages. SPSS and SAS are years behind R.
- 2<sup>nd</sup> only to MATLAB for graphics.
- Excellent for simulation, programming, computer intensive analyses, etc.
- Forces you to *think* about your analysis.
- Interfaces with database storage software (SQL)

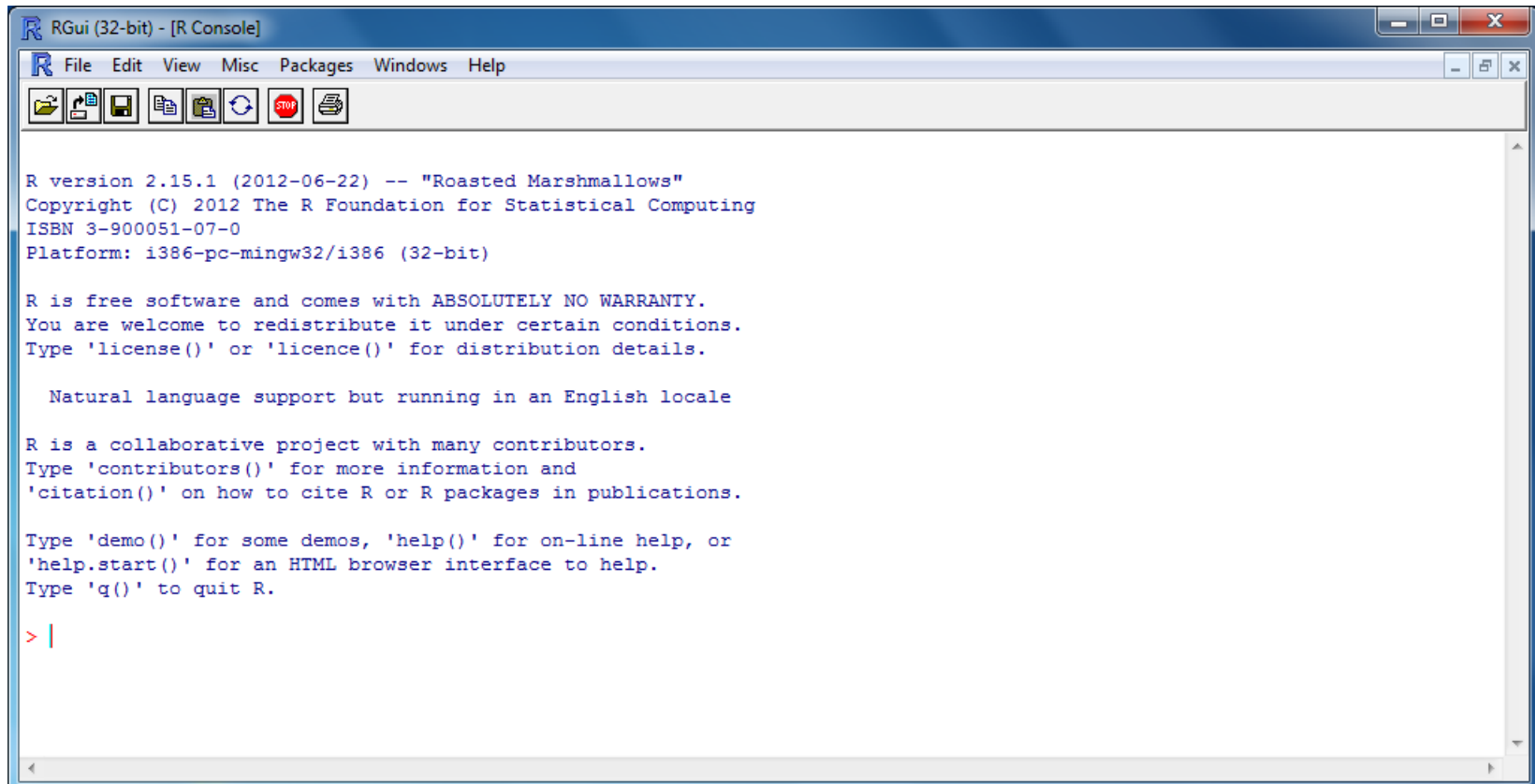
# Disadvantages

- Not user friendly @ start - steep learning curve, minimal GUI.
- No commercial support; figuring out correct methods or how to use a function on your own can be frustrating.
- Easy to make mistakes and not know.
- Working with large datasets is limited by RAM
- Data prep & cleaning can be messier & more mistake prone in R vs. SPSS or SAS
- Some users complain about hostility on the R listserve

# Getting Started (R-base and RStudio)

- Where to get R?
- Go to <http://cran.r-project.org/mirrors.html> to choose a mirror near you
- Click on your favorite operating system (Linux, Mac, or Windows)
- Download and install the “base”
- To install additional packages  
Start R on your computer  
Choose the appropriate item from the “Packages” menu
- R script/code can be coded in any text editor tool: Notepad, R script editor, Tinn-R, RStudio.
- I recommend RStudio because it really improves user/program GUI and programming environment ( <http://www.rstudio.org>).

# R base workspace and RStudio Editor



The image shows a screenshot of the RGui (32-bit) - [R Console] window. The window has a standard Windows-style title bar and menu bar. The menu bar includes File, Edit, View, Misc, Packages, Windows, and Help. Below the menu bar is a toolbar with icons for file operations and execution. The main area of the window displays the R startup screen, which includes the version number (2.15.1), copyright information (© 2012 The R Foundation for Statistical Computing), and platform details (i386-pc-mingw32/i386 (32-bit)). It also contains a disclaimer about the software being free and having no warranty, and provides instructions on how to use various help functions like 'license()', 'demo()', and 'q()'.

```
RGui (32-bit) - [R Console]
File Edit View Misc Packages Windows Help
R version 2.15.1 (2012-06-22) -- "Roasted Marshmallows"
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (32-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.

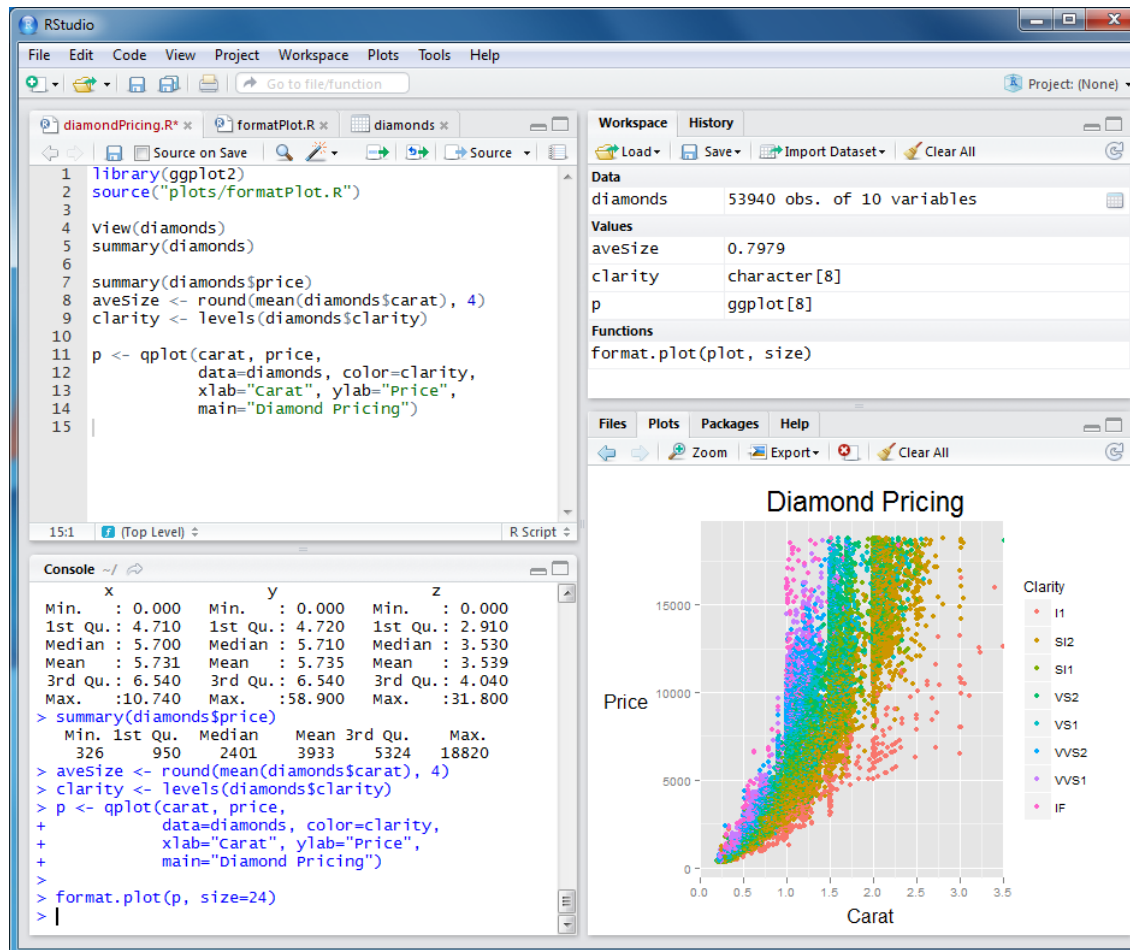
Natural language support but running in an English locale

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.

> |
```

# R base workspace and RStudio Editor (cont.)



The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for loading data, summarizing it, and creating a scatter plot.
- Workspace:** Shows the 'diamonds' dataset with 53940 observations and 10 variables.
- Console:** Shows the execution of the code, including summary statistics for 'x', 'y', and 'z' (representing carat, price, and clarity), and the creation of the 'p' plot object.
- Plots Panel:** Displays a scatter plot titled 'Diamond Pricing' showing Price (Y-axis, 0 to 15000) versus Carat (X-axis, 0.0 to 3.5). The points are colored by Clarity, with a legend on the right showing categories: I1, SI2, SI1, VS2, VS1, VVS2, VVS1, and IF.

```
1 library(ggplot2)
2 source("plots/formatPlot.R")
3
4 view(diamonds)
5 summary(diamonds)
6
7 summary(diamonds$price)
8 aveSize <- round(mean(diamonds$carat), 4)
9 clarity <- levels(diamonds$clarity)
10
11 p <- qplot(carat, price,
12           data=diamonds, color=clarity,
13           xlab="Carat", ylab="Price",
14           main="Diamond Pricing")
15
```

Console Output:

```
Min. : 0.000 Min. : 0.000 Min. : 0.000
1st Qu.: 4.710 1st Qu.: 4.720 1st Qu.: 2.910
Median : 5.700 Median : 5.710 Median : 3.530
Mean : 5.731 Mean : 5.735 Mean : 3.539
3rd Qu.: 6.540 3rd Qu.: 6.540 3rd Qu.: 4.040
Max. :10.740 Max. :58.900 Max. :31.800
> summary(diamonds$price)
  Min. 1st Qu. Median Mean 3rd Qu. Max.
 326    950   2401   3933   5324  18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+           data=diamonds, color=clarity,
+           xlab="Carat", ylab="Price",
+           main="Diamond Pricing")
>
> format.plot(p, size=24)
>
```



# R Environments

- Command Prompt: >
- The command line is not complete: +
- Current working direction: `getwd()`
- Change working direction: `setwd("c:/Myfile")`
- R gets confused if you use a back slash "\ " path in your code like  
`c:\mydocuments\myfile.txt`

This is because R sees "\" as an escape character. Instead, use  
Double back slasp "\\ " or Slash "/"

`c:\\my documents\\myfile.txt`

or

`c:/mydocuments/myfile.txt`

# Objects

- Types of objects: **variable, vector, factor, array, matrix, data.frame, list.**
- Attributes
  - mode: numeric, character, complex, logical
  - length: number of elements in object
- Creation
  - assign a value
  - create a blank object

# Caution

- R is a case sensitive language.
  - FOO
  - Foo
  - foo

These three are three different objects

# R Help and R as a Calculator

## Help:

two types of help command:

option 1:

```
help(vector)
```

```
?vector
```

option 2:

```
help.search("vector")
```

```
??Matrix
```

## Quit R:

```
q()
```

```
quit()
```

## R as a calculator:

Addition:  $1+5$

Subtraction:  $2-1$

Multiplication:  $4*5$

Division:  $10/2$

Power:  $3^2$

Comment mark : **#**

```
# comment line.
```

Kill current operation:

**Esc** or **Ctrl-C**



# Built-in R Function

## Built-in function:

```
exp(5) # exponential
function
log(5) # logarithm
cos(0) # cosine
sum(c(1,2,3,4,5,6)) #
sum
mean(c(1,2,3,4,5,6)) #
mean
# standard deviation
sd(c(1,2,3,4,5,6))
```

## Function can be nested:

"from the inside out" according to the following order of operations:

- terms inside parentheses or brackets
- exponents and roots
- multiplication and division  
As they appear left to right
- addition and subtraction  
As they appear left to right



# Object Type in R: Variables

```
> a <- 49  
> sqrt(a)  
[1] 7
```

numeric

```
> a <- c("The dog ate my homework")  
[1] "The dog ate my homework"
```

character string

```
> a <- (1+1==3)  
> a  
[1] FALSE
```

logical

# Object Type in R: Variables (cont.)

```
# store 5 in variable
(object) x "x gets 5"
x <- 5
# print the value of x on
screen
x
>x # print the value of x
on screen
[1] 5
# assign and print -> use
parentheses
(x <- 5)
[1] 5
# store 3 in variable y
(y <- 3)
[1] 3
```

```
# sum of two variables
option1
(z <- x + y)
# sum of two variables
option2
(z <- sum(x, y))
# Other assignment
(z <- x*y)
(z <- x^y)
(z <- sqrt(x))
(z <- exp(y))
(z <- log(exp(y)))
```

(please find more examples in the example script)



# Object Type in R: Vector

vector: an ordered collection of data of the same type

```
> a = c(1,2,3)
```

```
> a*2
```

```
[1] 2 4 6
```

In R, a single number is the special case of a vector with 1 element.

Other vector types: character strings, logical



# Object Type in R: Matrices and Arrays

- Matrix: a rectangular table of data of the same type
- Example:

```
# number 1-10, two column matrix
```

```
mat1 <- matrix(1:10, ncol = 2)
```

```
> mat1
```

```
      [, 1] [, 2]
[1, ]    1    6
[2, ]    2    7
[3, ]    3    8
[4, ]    4    9
[5, ]    5   10
```

- Array: Arrays are similar to matrices but can have more than two dimensions. See **help(array)** for details.
- Example: 3-,4-,...dimensional matrix

# Object Type in R: Data frames

Data Frame: is a flexible data table much like what you would be used to in a SAS or SPSS data set - like a spreadsheet.

- Columns are variable of a single type
- Rows are observations (e.g., people)

It is a rectangular table with rows and columns; data within each column has the same type (e.g., number, text, logical), but different columns may have different types (e.g., numeric, character, string, etc.).

Example:

```
> a
```

	<b>city</b>	<b>temprature</b>	<b>rain</b>
<b>XX001</b>	<b>New York</b>	<b>60</b>	<b>FALSE</b>
<b>XX002</b>	<b>Boston</b>	<b>57</b>	<b>TRUE</b>
<b>XX003</b>	<b>San Antonio</b>	<b>70</b>	<b>FALSE</b>

# Object Type in R: List

list: an ordered collection of data of arbitrary types.

```
> doe <- list(name="john",  
              age=28,married=F)
```

```
> doe$name
```

```
[1] "john"
```

```
> doe$age
```

```
[1] 28
```

Typically, vector elements are accessed by their index (an integer), list elements by their name (a character string). But both types support both access methods.

# Object Type in R: List

Most flexible data structure.

```
> list1 <- list(c(4),vec0,mat1)
```

```
> list1
```

```
[[1]]
```

```
[1] 4
```

```
[[2]]
```

```
[1] 1 2 3 4 5
```

```
[[3]]
```

```
      [,1] [,2]
```

```
[1,]    1    6
```

```
[2,]    2    7
```

```
[3,]    3    8
```

```
[4,]    4    9
```

```
[5,]    5   10
```

PEARSON



# Object Type in R: Factors

- Factors only exist in data frames
- Factor is used in the ANOVA sense
- Factors are categorical variables
- Factors are described by their levels
- Many R functions use factors to make it easy to run analyses
- You can turn any variable into a factor using `factor()`

# Logical Function: Branching

```
if (logical expression) {  
    statements  
} else {  
    alternative statements  
}
```

**else** branch is optional

# Logical Function: Loops

When the same or similar tasks need to be performed multiple times; for all elements of a list; for all columns of an array; etc.

```
for (i in 1:10) {  
    print(i*i)  
}
```

```
i=1
```

```
while (i<=10) {  
    print(i*i)  
    i=i+sqrt(i)  
}
```

# Logical Function: lapply

When the same or similar tasks need to be performed multiple times for all elements of a list or for all columns of an array. May be easier and faster than “for” loops

```
lapply( li, fct )
```

To each element of the list `H4`, the function `fct` is applied. The result is a list whose elements are the individual `fct` results.

```
> H4 <- list("klaus", "martin", "georg")
> lapply(H4, toupper)
> [[1]]
> [1] "KLAUS"
> [[2]]
> [1] "MARTIN"
> [[3]]
> [1] "GEORG"
```

**PEARSON**



# Logical Function: sapply

**Sapply** (H4, fct )

Like apply, but tries to simplify the result, by converting it into a vector or array of appropriate size

```
> H4 <- list("klaus", "martin", "georg")
> sapply(H4, toupper)
[1] "KLAUS" "MARTIN" "GEORG"

> fct = function(x) {return(c(x, x*x, x*x*x)) }
> sapply(1:5, fct)
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    1    4    9   16   25
[3,]    1    8   27   64  125
```

# Logical Function: `apply`

```
apply( arr, margin, fct )
```

Applies the function `fct` along some dimensions of the array `arr`, according to `margin`, and returns a vector or array of the appropriate size.

```
> x
      [,1] [,2] [,3]
[1,]    5    7    0
[2,]    7    9    8
[3,]    4    6    7
[4,]    6    3    5
```

```
> apply(x, 1, sum)
[1] 12 24 17 14
```

```
> apply(x, 2, sum)
[1] 22 25 20
```

# Logical Function: Function

- Functions get assigned to objects
- Keyword “function” starts function definition
- Braces “{}” mark start and stop

```
hi <- function() {  
    cat("Hello World \n")  
}
```

- Functions do things with data
  - “Input”: function arguments (0,1,2,...)
  - “Output”: function result (exactly one)

# Logical Function: Function (cont.)

- If you put variable names in the parentheses, that variable can be used in the function
  - The variable only exists inside the function
- When you call the function you put the value or object you want represented into the parentheses

```
> hi <- function(name) {  
  cat("Hello", name, "\n")  
}
```

```
> hi("Everybody")
```

# Package

- Packages are collections of **R** functions, data, and compiled code in a well-defined format.
- The directory where packages are stored is called the library.
- **R** comes with a standard set of packages. Others are available for download and installation.
- Once installed, they have to be loaded into the session to be used.

Let's open RStudio and try to install an R package.

# Importing and exporting data

There are many ways to get data into R and out of R.

```
> x = read.delim("filename.txt")
```

also: read.table, read.csv, read.xls, read.ssd

```
> write.table(x, file="x.txt", sep="\t")
```

also: save

Please find more examples in the hands-on R script.

# Storing data

Every R object can be stored into and restored from a file with the commands “save” and “load”.

This uses the XDR (external data representation) standard of Sun Microsystems and others, and is portable between MS-Windows, Unix, Mac.

```
> save(x, file="x.Rdata")
```

```
> load("x.Rdata")
```

# Graphics and Statistics

- R has a series of very powerful graphics
- R also has a huge number of functions and plug-ins for statistical testing and statistics.
  - Most build-in functions and packages deal with statistics and data analysis
  - many statistical researchers provide their methods as R packages
- Due to time limits, I will skip these two important parts.



# Fun Part of R

As a programming language, R also can do a lot of work.

- Web scraping-Use R to control your twitter (TwitterR)
- Draw a map in R (map, mapdata)
- Play games (fun)
- Play some basic music (seewave)
- Animation (animation)

# The End

Thank you!