기초

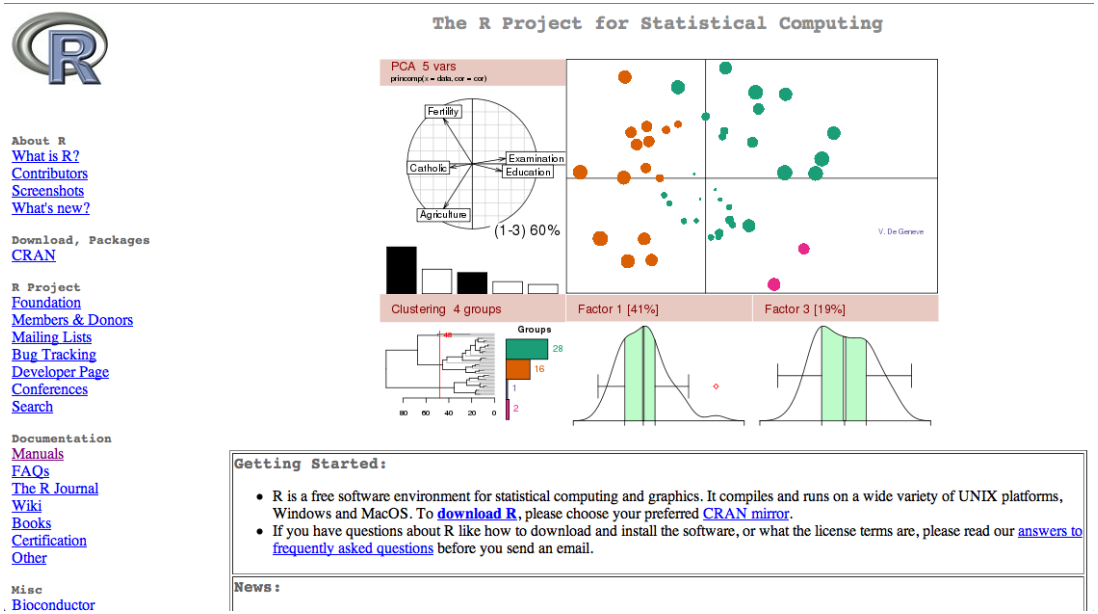
정 석 오
한국외국어대학교 통계학과

1. Introduction

R is…

- A programming language and an environment for data manipulation, (statistical) computing, and graphical display.
- Because R is now the *lingua franca* of data science, businesses are rapidly adopting R to support data science programs.
- Powerful but **FREE**!

http://www.r-project.org

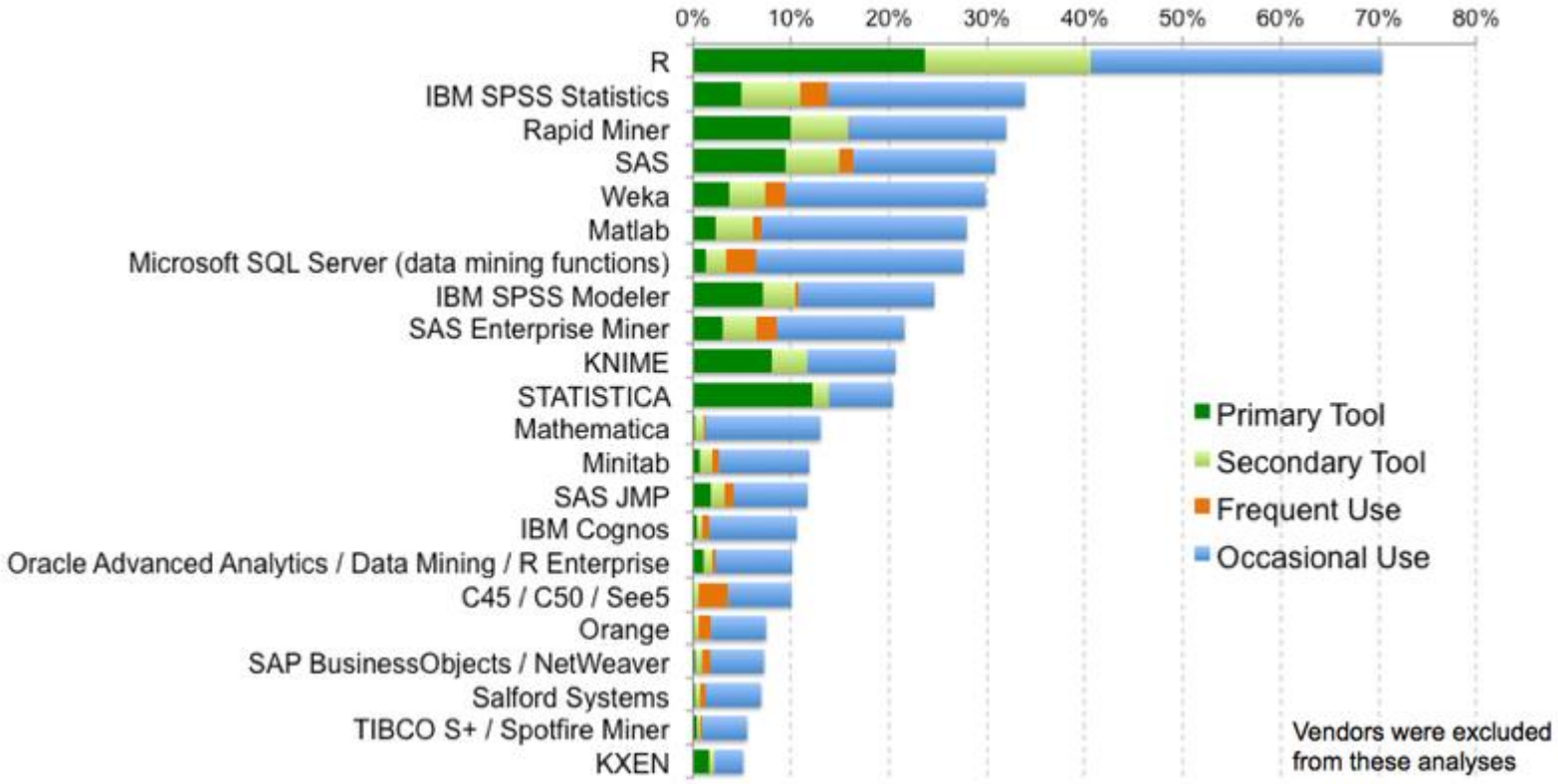


패키지 packages

- 모든 R 함수 및 dataset은 패키지로 묶여서 저장되어 있음
- 따라서 특정 함수 및 dataset은 해당 패키지를 load 해야 사용 가능
  - 메모리 관리 및 검색 시 효율적임
  - 패키지 개발자들이 기존 패키지와 충돌 걱정 없이 개발 가능
  - 확장성 ↑
- 패키지 설치 및 업데이트는 인터넷을 통해 항상 가능
- Standard (or base) package들은 R source code의 일부이기도 함
  - 기본적인 R 함수, 통계 및 데이터 분석을 위한 함수, 시각화 등을 위한 함수 등을 포함
  - R 설치 시 함께 설치되고 따로 load 할 필요 없음

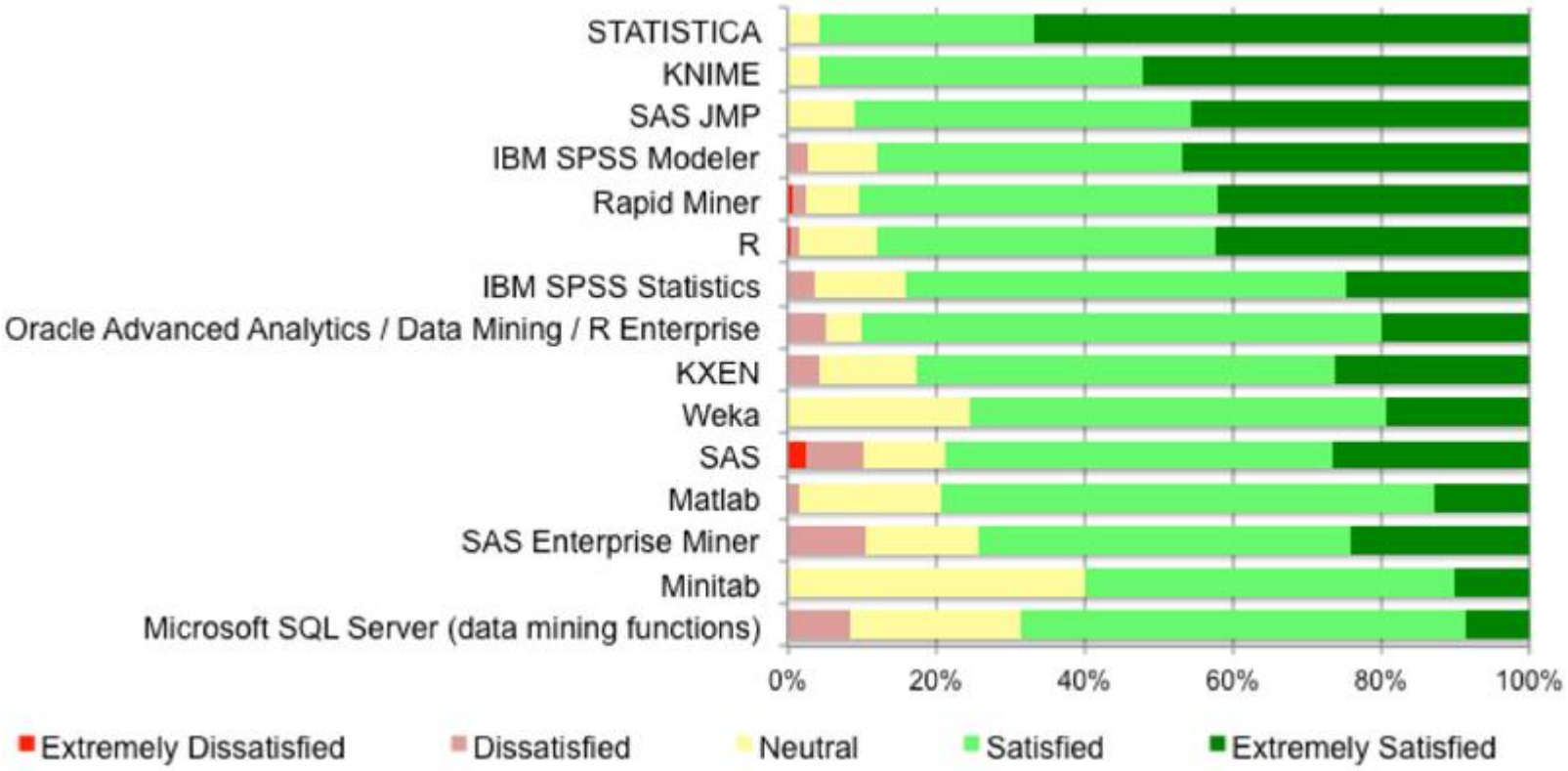
  c.f. Contributed packages

R is now the most popular tool…



(발췌) http://www.r-bloggers.com/r-usage-skyrocketing-rexer-poll/

Most users are satisfied with R...



(발췌) http://www.r-bloggers.com/r-usage-skyrocketing-rexer-poll/

Download & Installation

http://www.r-project.org

About R
What is R?
Contributors
Screenshots
What's new?

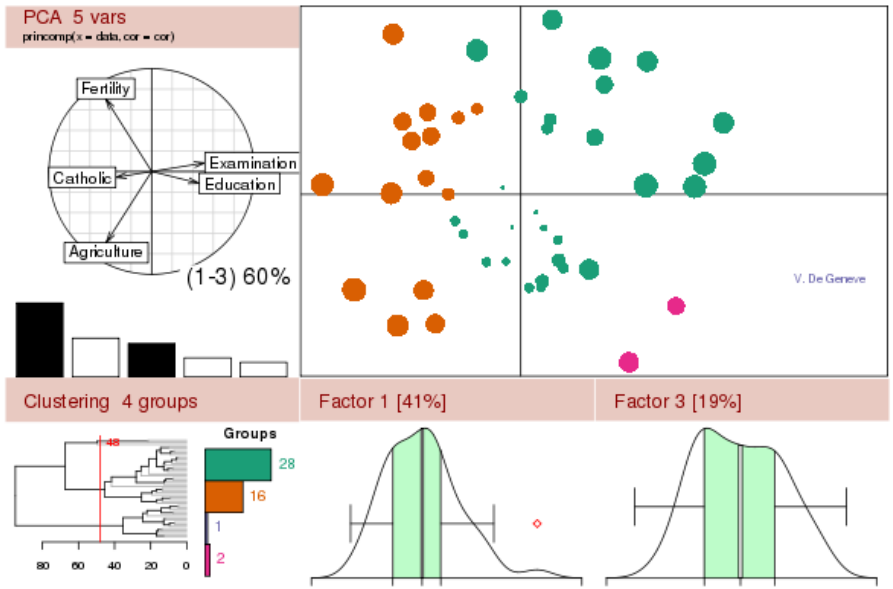
Download, Packages
CRAN

R Project
Foundation
Members & Donors
Mailing Lists
Bug Tracking
Developer Page
Conferences
Search

Documentation
Manuals
FAQs
The R Journal
Wiki
Books
Certification
Other

Misc
Bioconductor

The R Project for Statistical Computing



Getting Started:

- R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.
- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

News:

Console

- Run R, then a R-GUI window will appear.
- In R-GUI window, you'll see another window called 'R console'.
- Command prompt:

```
> 3+2
[1] 5
> pi
[1] 3.141593
```

Working Directory

- Working directory is the default location for all file input and output.
- Use `getwd()` to report the current working directory, and use `setwd()` to change your working directory.

```
> getwd()
> setwd("c:/users/mywork")
```

- Or, from the main menu, select
  "File" → "Change dir…"

Help

- Need a help for `persp()`? Just type in the command prompt
  `> ? persp`
  or
  `> help(persp)`
- Need an extended help on “log”? Type
  `> ?? log`
  or
  `> help.search(“log”)`
- Online documentation: Visit R-project website and click on “Manuals”.

Package

- All R functions and datasets are stored in packages.
- Installation of a package

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

- Loading a package

```
> library(MASS)
```

- Unloading a package

```
> detach(package:MASS)
```

R command

- For variable names, we may use alphabets, numbers, period(.), underscore(_), etc.
- For assignment, <- is used. You may use =, but not preferable.
- All names should begin with alphabet or period(.).
- Semicolon(;) separates multiple commands.

```
> beta.0 <- 3 ; beta.1 <- 2
```

- Comments begins with #.

```
> rnorm(100) # to generate 100 random numbers
```

R command

- Use arrows for recalling former commands.
- Type the name of a variable to print its value onto console.

```
> beta.0
[1] 3
> beta.0 + 1
[1] 4
> pi
[1] 3.141592
```

R command

- Case-sensitive

```
> a <- 1
> A <- 2
> a==A
[1] FALSE
```

- The objects are stored in R's database.

```
> ls()   # list the objects stored in database
```

- Run the script files.

```
> source("practice.R")
```

2. Data Manipulation

Data Types

- Vector

```
> x <- c(1,2,3,4,5)
> y <- c("Saenuri", "Minju", "Jinbo")
> x[3]
```

- Array

```
> z <- array(1:20, dim=c(4,5))
> z[4,5]
```

- Matrix

```
> z <- matrix(1:20, 4, 5)
> A <- matrix(2, 4, 5)
> z[3,2]
```

Data Types

- List

```
> Jeong <- list(first.name="Seok-Oh", age=42,
citizenship=”South Korea”)
```

- Data frame

```
> x <- c(100, 75, 80)
> y <- c(“A302043”, “A302044”, “A302045”)
> z <- data.frame(score=x, ID=y)
```

- Factor

```
> blood.type <- c(“A”, “B”, “AB”, “O”, “O”, “B”)
> blood.type <- factor(blood.type)
```

Vectors

- Concatenation

```
> a <- c(2,2,2,2,2,2)
> a <- c(1, 2, 3); b <- c(5, 6)
> x <- c(a, 4, b) # x <- c(1,2,3,4,5,6)
```

- Sequence

```
> x <- seq(from=0, to=1, by=0.1)
> y <- seq(from=0, to=1, length=11)
> z <- 1:10
> rep(1, 10)
```

Vectors

- Arithmetic: componentwise

```
> x <- 1:3; y <- c(2,2,2)
> x+y
> x-y
> x*y
> x/y
> x^y
> z <- rep(2, 5)
> x+z
> y-3
```

Vectors

- Mathematical functions

```
> x <- 1:10
> log(x)
> exp(x)
> sin(x)  # cos(x), tan(x)
> abs(x)
> sqrt(x)
> sort(x)
> length(x)
> min(x)
> max(x)
> mean(x)
> sum(x)
> prod(x)
```

Vectors

- Logical vectors

```
> x <- 1:10; y <- rep(5, 10)
> z <- x<5          # less than
> sum(z)
> x<=5              # less than or equal to
> x==5              # equal
> x!=5              # not equal
> (x>5)&(y<2)       # and
> (x<5)|(y<2)       # or
```

- Missing values

```
> x <- c(1, 2, 3, NA, 5)
> is.na(x)
```

Vectors

- Index vectors

```
> x <- -10:10
> x[3]
> x[1:3]
> x[c(1,3,5)]
> y <- x[x<0]
> x[x<0] <- -x[x<0]
> x <- c(1, 2, 3, NA, 5)
> x[!is.na(x)]
> x[is.na(x)] <- 4
> fruit <- c(5, 3, 2)
> names(fruit) <- c("apple", "orange", "peach")
> fruit[c("apple", "peach")]
```

Arrays and matrices

- To generate an array and a matrix

```
> z <- array(1:20, dim=c(4,5))
> A <- matrix(1:20, 4, 5)
> B <- matrix(2, 4, 5)
> z[3,4]    # Indexed by the position
> A[3,4]
> x <- c(1,2,3)
> y <- c(4,5,6)
> cbind(x, y)
> rbind(x, y)
> cbind(B, 1:4)
> C <- cbind(A, B)
```

Arrays and matrices

- Arithmetic: componentwise

```
> A <- matrix(1:20, 4, 5)
> B <- matrix(1:20, 4, 5)
> A+B
> A-B
> A*B
> A/B
```

- Arithmetic: matrix multiplication, inverse

```
> A <- matrix(runif(20), 4, 5)
> B <- A%*%t(A)      # t(): transpose
> solve(B)           # inverse
```

Lists

- A list is an object consisting of a collection of objects called as components.

```
> Jeong <- list(first.name=”Seok-Oh”, age=42,
married=T, no.children=2, child.ages=c(9, 6))
> Jeong$age
> Jeong[[1]]
> Jeong$child.ages
> Jeong[[5]][1]
```

3. Data Import / Export

Read data

- Use `scan()` to read your data from the console.

```
> x <- scan()
1: 1
2: 2
3: 3
4:
Read 3 items
> x
[1] 1 2 3
```

Read data

- From a file: `scan()`

```
> x <- scan(file="c:/mydata/data_x.txt")
> y <- matrix(scan("c:/mydata/data_y.txt"),
ncol=3, byrow=T)
```

- From a file: `read.table()`

```
> x <- read.table(file="table.txt", header=T,
sep=" ")
```

- From a file: `read.csv()`

```
> x <- read.csv(file="table.csv", header=T)
```

Read data

- Accessing built-in datasets:

```
> library("MASS")
> data("geyser")
```

- Want the list of built-in datasets contained in the currently loaded packages? Just type `data()`.

```
> data()
```

Export data

- To the console: `print()`

```
> x <- scan()
1: 1
2: 2
3: 3
4:
Read 3 items
> print(x)
[1] 1 2 3
```

Export data

- To a file: `write()`

```
> x <- seq(from=0, to=1, by=0.1)
> write(x, file="output.txt")
```

- To a file: `write.table()`

```
> x <- matrix(1:20, 4, 5)
> write.table(x, file="table.txt")
```

4. Graphics: visualization of data

One-dimensional data

- Qualitative data
  - Bar chart: `barplot()`
  - Pie chart: `pie()`
- Quantitative data
  - Stem-and-leaf plot: `stem()`
  - Histogram: `hist()`
  - Boxplot: `boxplot()`

```r
## Beer Preference example

beer <- c(3, 4, 1, 1, 3, 4, 3, 3, 1, 3, 2, 1, 2,
   1, 2, 3, 2, 3, 1, 1, 1, 1, 4, 3, 1)
# (1) Domestic can  (2) Domestic bottle,
# (3) Microbrew     (4) Import

barplot(table(beer))
barplot(table(beer)/length(beer),
  col=c("lightblue", "mistyrose", "lightcyan","cornsilk"),
  names.arg=c("Domestic can", "Domestic bottle", "Microbrew",
  "Import"), ylab="Relative frequency",
  main="Beer Preference Survey")

beer.counts <- table(beer) # store the table result
pie(beer.counts) # first pie -- kind of dull
names(beer.counts) <- c("Domestic\n can","Domestic\n bottle",
 "Microbrew","Import") # give names
pie(beer.counts) # prints out names
```

```r
## Stem-and-leaf

scores <- c(2, 3, 16, 23, 14, 12, 4, 13, 2, 0, 0, 0,
   6, 28, 31, 14, 4, 8, 2, 5)
stem(scores)

## histogram

x <- rnorm(1000)  # To generate 1,000 random numbers from N(0,1)
hist(x, xlab="data")
hist(x, probability=T, xlab="data")
z <- seq(from=-3, to=3, by=0.01)
lines(z, dnorm(z), col=2)

## Boxplot

growth <- c(75,72,73,61,67,64,62,63) # the size of flies
sugar <- c("C","C","C","F","F","F","S","S") # diet
fly <- list(growth=growth, sugar=sugar)
boxplot(fly$growth)
jpeg(file="flygrowth.jpg", width=480, height=360)
```

Multi-dimensional data

- Categorical & Quantitative
  - Boxplot: `boxplot()`
- Quantitative & Quantitative
  - Scatterplot: `plot()`

```r
## Boxplot

boxplot(growth~sugar, xlab="Sugar Type", ylab="Growth",
   main="Growth against sugar types", data=fly)


## Scatterplot

plot(cars$speed, cars$dist)
# the speed of cars and the distances taken to stop
attach(cars)
plot(speed, dist, col="blue", pch="+",
   ylab="Distance taken to stop", xlab="Speed",
   ylim=c(-20, 140))
lm(dist~speed)
abline(-17.579, 3.932, col="red")
title(main="Scatterplot with best fit line", font.main=4)
```

```r
## Scatterplot matrix

attach(iris)
pairs(iris[,1:4])
pairs(iris[Species=="virginica", 1:4])


## 2D Histogram

library(hexbin)
plot(hexbin(iris[,3], iris[,4]),
  xlab="Petal Length", ylab="Petal Width")
```

5. Advanced Programming

Conditional execution

- A conditional statement by 'if-else'.

```
> if (x<3) print("x<3") else print("x>4")
```

- Commands can be grouped by braces.

```
> x <- 4
> if ( x < 3 ) {print("x<3"); z <- "M"} else
{print("x>3"); z <- "F"}
```

Iteration, loop

- Loop: A repeatedly executed instruction cycle
- for-loop: loop over all elements in a vector

```
x <- 1:10
n <- length(x)
y <- rep(0, n)
for ( i in 1:n ) {
   y[i] <- x[i]^2
}
z <- x^2
print(cbind(y, z))
```

Iteration, loop

- while-loop: for which we don't know in advance how many iterations where will be.

```
n <- 0
sum.so.far <- 0
while ( sum.so.far <= 1000 ) {
    n <- n+1
    sum.so.far <- sum.so.far + n
}
print(c(n, sum.so.far))
sum(1:45)
```

✓**Whenever possible, try to avoid loops.**

Applying a function to every row/column

- To apply a function to every row[or column], use `apply()`.

```
> A <- matrix(1:20, 4, 5)
> apply(A, 1, sum)   # to every row
> apply(A, 2, sum)   # to every column
```

Writing a new function

```
my.stat <- function(x)
  {
     m <- mean(x); s <- sd(x)
     res <- list(x=x, m=m, s=s)

     par(mfrow=c(1,2))
     boxplot(x, main="Boxplot")
     hist(x, prob=T, col="lightgray", main="Histrogram",
          xlab="data")
     z <- seq(from=min(x), to=max(x), by=0.01)
     lines(z, dnorm(z, mean=3, sd=1), col=2, lwd=3, lty=2)
     return(res)
  }

  data <- rnorm(1000, mean=3, sd=1)
  my.stat(x=data)
```

Q & A