Stats 406 Lab 1

GAO Zheng

September 12, 2016

## Lab info

* Lab section: Thursday, 11:30 am - 1:00 am (Gao, Zheng)
* Office hours: (Science Learning Center, Chemistry building, attend any)
  + Tuesday: 10:00a.m. – 11:30a.m. (Gao, Zheng)
  + Wednesday 4:00p.m.-5:30p.m. (Liu, Boang)

Face to face conversation is prefered to email conversations.

I am roughly following the **TutorialsPoint.com**'s R tutorial: <http://www.tutorialspoint.com/r/>

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

## Getting started

# get working directory  
getwd()

## [1] "/home/gaozheng/Teaching/Stats406/R\_scripts/lab1"

# set appropriate working directory  
setwd("/home/gaozheng/Teaching/Stats406/R\_scripts")

## Data types

There are 6 types of basic data types in R: Logical, Numeric, Integer, Complex, Character, Raw

v <- TRUE   
print(class(v))

## [1] "logical"

v <- 23.5  
print(class(v))

## [1] "numeric"

v <- 2L  
print(class(v))

## [1] "integer"

v <- 2+5i  
print(class(v))

## [1] "complex"

v <- "TRUE"  
print(class(v))

## [1] "character"

v <- charToRaw("Hello")  
print(class(v))

## [1] "raw"

## Vectors and Lists

Vectors contains objects of the *same* type. List doesn't require members to be of the same type.

### Vectors

# initialize a vector (concatenation)  
vector1 <- c(1,'A',T,2.3)  
# print a vector (works inside functions)  
print(vector1)

## [1] "1" "A" "TRUE" "2.3"

# another way to display a variable (doesn't work inside functions)  
vector1

## [1] "1" "A" "TRUE" "2.3"

# another way to initialize a vector  
x <- 1:10  
# yet another way to initialize a vector  
x <- seq(from = 1, to = 10, by = 1);x

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

# you can leave out the paramenter names as long as the orders are correct  
x <- seq(1,10,1);x

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

# initialize a variable using existing ones  
y <- x^2  
# Notice that you don't have to write a loop to define y  
# R is 'vector-friendly'  
# In fact you should AVOID writting loops in R unless necessary  
print(y)

## [1] 1 4 9 16 25 36 49 64 81 100

### Lists

# Create a list.  
list1 <- list(c(2,5,3),21.3,sin)  
# Now things inside list1 do NOT get converted to the same type!  
print(list1)

## [[1]]  
## [1] 2 5 3  
##   
## [[2]]  
## [1] 21.3  
##   
## [[3]]  
## function (x) .Primitive("sin")

### Subsetting a vector / list

# You can ``access'' any elements of a vector / list by providing its index inside the [ ]  
vector1[1]

## [1] "1"

# Or by providing a vector of indices  
vector1[1:3]

## [1] "1" "A" "TRUE"

# Or by providing a vector of Logicals  
index <- c(T,F,F,T)  
vector1[index]

## [1] "1" "2.3"

#### Quiz: what happens if the length of the Logical index and the vector disagrees?

# vector1[c(T,F,F)]

Subsetting a list is **tricky**!

# Subsetting a list with [ ], and you still get a list!  
list1[1]

## [[1]]  
## [1] 2 5 3

class(list1[1])

## [1] "list"

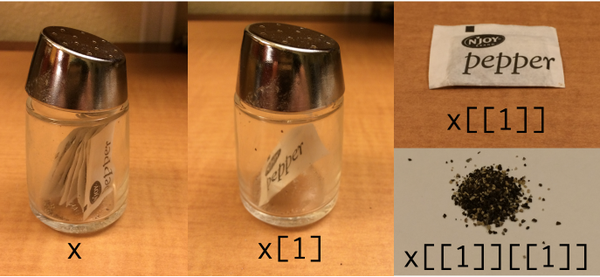
# To extract the vector, use double [[ ]]  
list1[[1]]

## [1] 2 5 3

class(list1[[1]])

## [1] "numeric"

Provocative imagery by [Hadley Wickham](https://twitter.com/hadleywickham/status/643381054758363136):



## Matrices

Think of them as ``2-D vectors''.

### Creating a matrix

M <- matrix( c(4,2,6,3,7,1), nrow = 2, ncol = 3, byrow = TRUE)  
print(M)

## [,1] [,2] [,3]  
## [1,] 4 2 6  
## [2,] 3 7 1

# Matrix transpose  
t(M)

## [,1] [,2]  
## [1,] 4 3  
## [2,] 2 7  
## [3,] 6 1

### Naming the columns and rows (always good to have meaningful names)

colnames(M) <- c('col1','col2','col3')  
rownames(M) <- c('row1','row2')  
M

## col1 col2 col3  
## row1 4 2 6  
## row2 3 7 1

#### Quiz: what happens if the length of the content and the matrix size disagrees?

# M <- matrix(1:5, nrow = 2, ncol = 3)  
# print(M)

Subsetting of a matrix is siimlar to that of a vector, except you need to provide two indices for the two dimensions:

M[2,c(1,3)]

## col1 col3   
## 3 1

### Matrix mutiplication

Dimensions must agree! Vectors are treated by default as column vectors.

# with a vector  
M%\*%c(1,2,3)

## [,1]  
## row1 26  
## row2 20

# with another matrix  
M%\*%t(M)

## row1 row2  
## row1 56 32  
## row2 32 59

#### Quiz: what happens if the orientation is misaligned for the vector?

# M%\*%t(c(1,2,3))  
# What about this?  
# t(c(1,2))%\*%M

## Dataframes

Data frame is an extension to matrices, allowing columns to take different types.

col1 <- (1:5)/10; col2 <- letters[1:5]; col3 <- round(sin(1:5),3);

Numerical objects are converted to characters when concatenating A1 to A3, same as when you create a vector of mixed data types.

A <- cbind(col1 , col2 , col3) # cbind for 'column bind'  
print(A)

## col1 col2 col3   
## [1,] "0.1" "a" "0.841"   
## [2,] "0.2" "b" "0.909"   
## [3,] "0.3" "c" "0.141"   
## [4,] "0.4" "d" "-0.757"  
## [5,] "0.5" "e" "-0.959"

Data frames keep numerical variables numerical, and convert characters to factors.

dataframe1 = data.frame(col1 , col2 , col3)  
dataframe1

## col1 col2 col3  
## 1 0.1 a 0.841  
## 2 0.2 b 0.909  
## 3 0.3 c 0.141  
## 4 0.4 d -0.757  
## 5 0.5 e -0.959

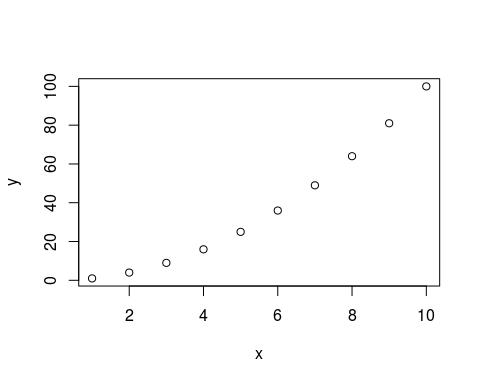
Viewing dataframe summaries

summary(dataframe1)

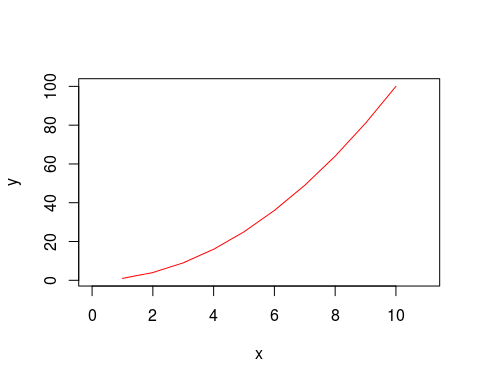
## col1 col2 col3   
## Min. :0.1 a:1 Min. :-0.959   
## 1st Qu.:0.2 b:1 1st Qu.:-0.757   
## Median :0.3 c:1 Median : 0.141   
## Mean :0.3 d:1 Mean : 0.035   
## 3rd Qu.:0.4 e:1 3rd Qu.: 0.841   
## Max. :0.5 Max. : 0.909

## Basic plotting

# plot a series against another  
plot(x,y)



# connecting the dots, add colors, plotting ranges  
plot(x,y,type = 'l',col = 'red', xlim = c(0,11))



## Look for variables in environment

# list variables in the environment  
ls()

## [1] "A" "col1" "col2" "col3" "dataframe1"  
## [6] "index" "list1" "M" "v" "vector1"   
## [11] "x" "y"

# remove a variable from the environment  
rm(list='vector1')  
# Now 'vector1' is gone  
ls()

## [1] "A" "col1" "col2" "col3" "dataframe1"  
## [6] "index" "list1" "M" "v" "x"   
## [11] "y"

## Functions

### How to use an existing function from package

# look for manual for a function  
?rm  
# fuzzy search  
??remove

I find Google search more helpful than ``??''.

### How to define a new function

# use 'function' function  
f <- function(x){  
 x^2-2\*x+1  
}  
# last line is returned, you can also use 'return' for clarity  
f = function(x){  
 return (x^2-2\*x+1)  
}  
# Now apply it  
f(x) # works on numericals

## [1] 0 1 4 9 16 25 36 49 64 81

# incompatible operation for strings  
# f('test')