R Lecture #1

October 11th, 2017

Hyeokkoo Eric Kwon (KAIST)

hkkwon7@business.kaist.ac.kr

About R

- Statistical computing language
- Used by statisticians and data analysts
- Open source version of "S"
- Large number of built-in statistical functions
- Easily configurable via packages

Why R

- Free!!!!!
- Open source code
- A lot of training materials
- Its best data visualization tools
- High compatibility
- The latest and broadest range of statistical algorithms

How to install R & R Studio

- Download *R* from http://cran.r-project.org/
- Download *R Studio* from https://www.rstudio.com/
- See
  1. *http://youtu.be/cX532N_XLIs*
  2. *http://youtu.be/MFfRQuQKGYg*

Working Directory

- If you want to read files from a specific location or write files to a specific location, you will need to set working directory in R.
  1. *Get working directory : getwd()*
  2. *Set working directory : setwd("C:/Documents and Settings/Data")*

- you must use the forward slash / or double backslash \\ in R.

Data Types Used in R

- **Vector** : a collection of ordered homogeneous elements
- **Matrix** : a vector with two-dimensional shape information
- **Data.frame** : a set of heterogeneous vectors of equal length.

- **List** : a vector with possible heterogeneous elements.

Vector

Vector

- In R the "base" type is a vector, not a scalar.
- A vector is an indexed set of values that are all of the same type. The type of the entries determines the class of the vector. The possible vectors are:
  1. *integer*
  2. *numeric*
  3. *character*
  4. *complex*
  5. *logical*
- Integer is a subclass of numeric.
- Cannot combine vectors of different modes.

Creating Vectors

- Assignment of a value to a variable is done with <- (two symbols, no space).
- Vectors can only contain entries of the same type: numeric or character; you can't mix them. Note that characters should be surrounded by " ".
- The most basic way to create a vector is with *c(x1, . . . , xn)*, and it works for characters and numbers alike.

1. *x <- c("a", "b", "c")*
2. *x*
3. *y <- c(1, 2, 3, 4)*
4. *y*
5. *length (x)*
6. *length (y)*

Vector Names

- Entries in a vector can and normally should be named. It is a way of associating a numeric reading with a sample id, for example.

1. *score <- c(math = 100, eng= 30, science= 70)*
2. *score*
3. *sort(score)*
4. *names(score)*

5. *score2 <- c(100,30,70)*
6. *names(score2) <- c("math", "eng", "science")*

Vector Arithmetic

- Numeric vectors can be used in arithmetic expressions, in which case the operations are performed element by element to produce another vector.
- The elementary arithmetic operations are the usual +, -, *, /, ^.

1. *x <- c(1, 2, 3)*
2. *y <- c(5, 0, 1)*
3. *x + 1*
4. *x - y*
5. *x * y*
6. *y / x*
7. *x ^ y*

Vector Arithmetic (Cont'd)

- Also *log*, *exp*, *sqrt* etc.
  1. *z <- c(0, exp(1), exp(2),10,100)*
  2. *log(z)*
  3. *exp(z)*
  4. *log10(z)*
  5. *sqrt(z)*

More Vector Arithmetic

- In studying data you will make frequent use of *sum*, *max*, *min*, *mean*, and *var(x)*

1. *x<-c(1,3,5,7,9)*
2. *sum(x)*
3. *max(x)*
4. *min(x)*
5. *mean(x)*
6. *var(x)*
7. *summary(x)*

Generating Regular Sequences

- Using *seq* and *rep*
- seq(from, to , by)
- rep(x, each, times)

1. *seq(from = 0, to = 10)*
2. *seq(from = 10, to = 0)*
3. *seq(from = 0, to = 10, by = 2)*

4. *x1<-1:2*
5. *x2<-rep(x = x1, each = 3, times = 2)*
6. *x3<-rep(x = x1, each = 2, times = 3)*

Logical Vectors

- Working with data often involves comparing numbers. Comparisons in R output logical values TRUE, FALSE or NA, for "not available".
- Just as with arithmetic operations, logical comparisons with a vector are applied to each entry and output as a vector of truth values; i.e., a logical vector.

1. *v<-seq(-3,3)*
2. *v*
3. *tf<- v>0*
4. *tf*

Logical Comparisons

- Logical vectors are largely used to extract entries from a dataset satisfying certain conditions. The logical operators are: <, <=, >, >=, == for exact equality and != for inequality.

- If *c1* and *c2* are logical expressions, then *c1 & c2* is their intersection ("and"), *c1 | c2* is their union ("or"), and *!c1* is the negation of c1.

1. *v1 <- seq(-2,2)*
2. *v2 <- seq(1,5)*
3. *tf1 <- v1>0 & v2 > 0*
4. *tf2 <- v1==0 | v2 != 3*
5. *tf3 <- !tf2*

Missing Values

- One smart feature of R is that it allows for missing values in vectors and datasets; it denotes them as NA. You don't have to coerce missing values to, say 0, in order to have a meaningful vector.
  1. *x <- c(1:3, NA)*
  2. *is.na(x)*

Extracting Subsequences of a Vector

- Getting elements of a vector with desired properties is extremely common, so there are robust tools for doing it.
- An element of a vector *v* is assigned an index by its position in the sequence, starting with *1*.

- The basic function for subsetting is [ ].
- *v[1]* is the first element, *v[length(v)]* is the last. The subsetting function takes input in many forms.

Subsetting with Positive & Negative Integral Sequences

1. *v <- c("a", "b", "c", "d", "e")*
2. *J <- c(1, 3, 5)*

1. *v[1, 3, 5]*
2. *v[c(1, 3, 5)]*
3. *v[J]*
4. *v[1:3]*
5. *v[2:length(v)]*

1. *v[-J]*
2. *v[-1]*
3. *v[-length(v)]*

Subsetting with Logical Vector

- Given a vector *x* and a logical vector *L* of the same length as *x*, *x[L]* is the vector of entries in *x* matching a *TRUE* in *L*.

1. *v <- c("a", "b", "c", "d", "e")*
2. *L <- c(TRUE, FALSE, TRUE, FALSE, TRUE)*
3. *v[L]*

4. *x<- seq(-3, 3)*
5. *x>=0*
6. *x[x>=0]*

Subsetting by Names

- If *x* is a vector with names and *A* is a subsequence of *names(x)*, then *x[A]* is the corresponding subsequence of *x*.
  1. *x <- seq(-1,1)*
  2. *names(x)<-c("N1", "N2", "N3")*
  3. *x[c("N1", "N3")]*

Assignment to a Subset

- A subset expression can be on the receiving end of an assignment, in which case the assignment only applies the subset and leaves the rest of the vector alone.

1. *z <- seq(1,4)*
2. *z[1] <- 0*
3. *z[z<=2] <- 10*

4. *w <- c(1:3, NA, NA)*
5. *w[is.na(w)] <- 0*

Matrix

Matrix

- The entries in a matrix X are arranged in rows and columns. Think of it as a two-dimensional version of a numeric vector.
- X is $n \times m$ if it has n rows and m columns.

- Create a 3×4 matrix all of whose entries are 0:
  1. `x <- matrix(0, nrow = 3, ncol = 4)`
  2. `nrow(x)`
  3. `ncol(x)`
  4. `dim(x)`

More Examples

1. `Y <- matrix(1:12, nrow = 3, ncol = 4)`
2. `Y <- matrix(1:12, nrow = 3, ncol = 4, byrow=FALSE)`
3. `Y[1, 3]`
4. `Y[1, ]`
5. `Y[, 2]`

6. `x <- 1:15`
7. `dim(x) <- c(3, 5)` **Matrix=Vector with Dimension**
8. `x`

9. `z <- x[1:2, 3:4]` **Submatrices**

10. `x[1, 3] <- 1` **Matrix Assignment**
11. `x[ , 1] <- c(-1, -2, -3)`
12. `x[3, ] <- 2`

More Examples (Cont'd)

1. *Y <- matrix(1:6, nrow = 3, ncol = 2)*
2. *r <- matrix(1:6, nrow = 3, ncol = 2)*

3. *Y * r*
4. *Y %*% r*

5. *t(r)* **Transpose**
6. *Y %*% t(r)*

7. *diag(Y %*% t(r))* **Diagonal vector**

Row and Column Stats

- Frequently we'll want to extract statistics from the rows or columns of a matrix.
- Let *f* be a function that produces a number given a vector.
- If *X* is a matrix *apply(X, 1, f)* is the result of applying *f* to each row of *X; apply(X, 2, f)* to the columns.

1. *Y <- matrix(1:12, nrow = 3, ncol = 4)*
2. *apply(Y, 1, min)*
3. *apply(Y, 2, mean)*

Names

- Just as you can name indices in a vector you can (and should!) name columns and rows in a matrix with *colnames(X)* and *rownames(X)*. These can be used in subsetting just like vectors.

1. `Y <- matrix(1:12, nrow = 3, ncol = 4)`
2. `colnames(Y) <- c("X1", "X2", "X3", "X4")`

Example

- Generate matrix ***Y***:

	var1	var2	var3	var4
obs1	1	4	7	10
obs2	2	5	8	11
obs3	3	6	9	12

- Generate vector ***max_Y*** using matrix ***Y***:

max_var1	max_var2	max_var3	max_var4
3	6	9	12

Example (Cont'd)

1. *Y <- matrix(1:12, nrow = 3, ncol = 4)*
2. *rownames(Y) <- c("obs1", "obs2", "obs3")*
3. *colnames(Y) <- c("var1", "var2", "var3", "var4")*

4. *max_Y <- apply(Y, 2, max)*
5. *names(max_Y)<-paste("max", colnames(Y), sep="_")*

Data.frame

Data.frame

- Like matrix, data frames are tabular data objects.
- Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical.
- It is a set of vectors of equal length.

Creating Data Frames

	gender	height	weight
1	Male	152.0	81
2	Male	171.5	93
3	Female	165.0	78
	Character	Numeric	Numeric

```
BMI   <-   data.frame  (
                        gender = c("Male", "Male","Female"),
                        height = c(152, 171.5, 165),
                        weight = c(81,93, 78)
                        )

str(BMI)
```

Creating Data Frames (Cont'd)

	gender	height	weight
Eric	Male	152.0	81
James	Male	171.5	93
Babel	Female	165.0	78

1. *BMI [1,2]*
2. *names(BMI)*
3. *rownames(BMI) <- c("Eric", "James", "Babel")*

Components as Vectors

	gender	height	weight
Eric	Male	152.0	81
James	Male	171.5	93
Babel	Female	165.0	78

- The components of a data frame can be extracted as a vector:

1. *height <-BMI$height*
2. *names(height) <- rownames(BMI)*

Expanding Data Frames

	gender	height	weight	age	tel
Eric	Male	152.0	81	42	SK
James	Male	171.5	93	38	KT
Babel	Female	165.0	78	26	LG

- Components can be added to a data frame:
  1. *BMI$age <- c(42,38,26)*

  1. *tel <- c("SK", "KT", "LG")*
  2. *BMI <- cbind(BMI, tel)*

Expanding Data Frames (Cont'd)

	gender	height	weight	age	tel	score	overweight
Eric	Male	152.0	81	42	SK	35.05886	TRUE
James	Male	171.5	93	38	KT	31.61948	TRUE
Babel	Female	165.0	78	26	LG	28.65014	FALSE

- Components can be added to a data frame:
  1. *BMI$score <- 10000 * BMI$weight / (BMI$height^2)*
  2. *BMI$overweight <- BMI$score >= 30*

Expanding Data Frames (Cont'd)

	gender	height	weight
Eric	Male	152.0	81
James	Male	171.5	93
Babel	Female	165.0	78
Sanchez	Male	160.0	50
Yoon	Male	170	60

- Components can be added to a data frame:
  1. *BMI <- BMI[1:3,1:3]*
  2. *add_row <- data.frame(gender="Male", height=160, weight =50)*
  3. *rownames(add_row) <- "Sanchez"*
  4. *BMI <- rbind(BMI, add_row)*

  5. *BMI[5,] <- c("Male", 170, 60)*
  6. *rownames(BMI)[5] <- "Yoon"*

List

List

- A list is a set of objects.
- Each element in a list can be a:
  1. *Vector*
  2. *Matrix*
  3. *Data.frame*
  4. *List*

Example 1

```
1. vec_n <- c(1:10)
2. vec_c <- c("Eric", "James")
3. mat   <- matrix(1:10,nrow=5, ncol=2)
4. df    <- data.frame(
                    gender = c("Male", "Male","Female"),
                    height = c(152, 171.5, 165),
                    weight = c(81,93, 78)
                    )
4. lst_ex <- list(vec_n, vec_c, mat, df)
```

Indexing

- Elements within objects are indexed using *[]* and *[[]]*.
- **Vectors**: *[i]* for the *i*th element
- **Matrices and Data.frames**: *[i,j]* for the *i*th row, *j*th column
- **Lists**: *[[i]]* for the *i*th element

1. *lst_ex[[1]]*
2. *lst_ex[[1]][4]*
3. *lst_ex[[2]]*
4. *lst_ex[[2]][2]*
5. *lst_ex[[3]]*
6. *lst_ex[[3]][3,2]*
7. *lst_ex[[4]]*
8. *lst_ex[[4]][1,3]*

➢ list

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

[[2]]
[1] "Eric"  "James"

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

[[4]]
  gender height weight
1   Male  152.0     81
2   Male  171.5     93
3 Female  165.0     78
```

Indexing with Names

1. ```
   Lst <- list(
           name = "Joe",
           height = 182,
           no.children = 3,
           child.ages = c(5, 7, 10)
           )
   ```
2. ```
   Lst[[2]]
   ```
3. ```
   Lst[["height"]]
   ```
4. ```
   Lst$height
   ```

- Lst

```
$name
[1] "Joe"

$height
[1] 182

$no.children
[1] 3

$child.ages
[1]  5  7 10
```