R - II

Bioinformatics Applications (PLPTH813)

Sanzhen Liu

2/9/2017

Outline

- your own functions
- apply family
- Other useful functions
- Simple statistical test

function/module in R

- If a procedure is repeated multiple times, it would be valuable to convert the procedure to a function/module.

- **Define a function**

```
fun_name <- function(arg_1, arg_2, ...) expression
```

or

```
fun_name <- function(arg_1, arg_2, ...) {
    expressions
}
```

- **Use a function**

```
fun_name(arg_1, arg2, ...)
```

Function example 2

- **Define a function**

name <- function(arg_1, arg_2, ...) expression

```
# example 1
threetimes <- function(x) {
  y <- 3*x
  y
}
```

```
> threetimes(6)
[1] 18

> val <- threetimes(29)
> val
[1] 87
```

Function example 2

```
# return the value of the nth element of the input vector
what_at_n <- function(in_vector, n) {
  # initiate the output value
  nth_val <- NA

  if (n <= length(in_vector)) {
    nth_val <- in_vector[n]
  }

  print_info <- paste("The value of element", n, "is", nth_val, sep=" ")
  print(print_info)
  nth_val
}
```

```
> what_at_n(c(36, 19, 13), 2)
[1] "The value of element 2 is 19"
[1] 19

> val2 <- what_at_n(c(36, 19, 13), 2)
[1] "The value of element 2 is 19"
> val2
[1] 19
```

base (build-in) functions in R

- R has many build-in functions
- What we learn is to know how to use them
- If you have choices to use a build-in function, do not use your own function (efficiency and code sharing)

"apply" functions

- **apply**
- **lapply**
- **sapply**
- **mapply**
- **tapply**

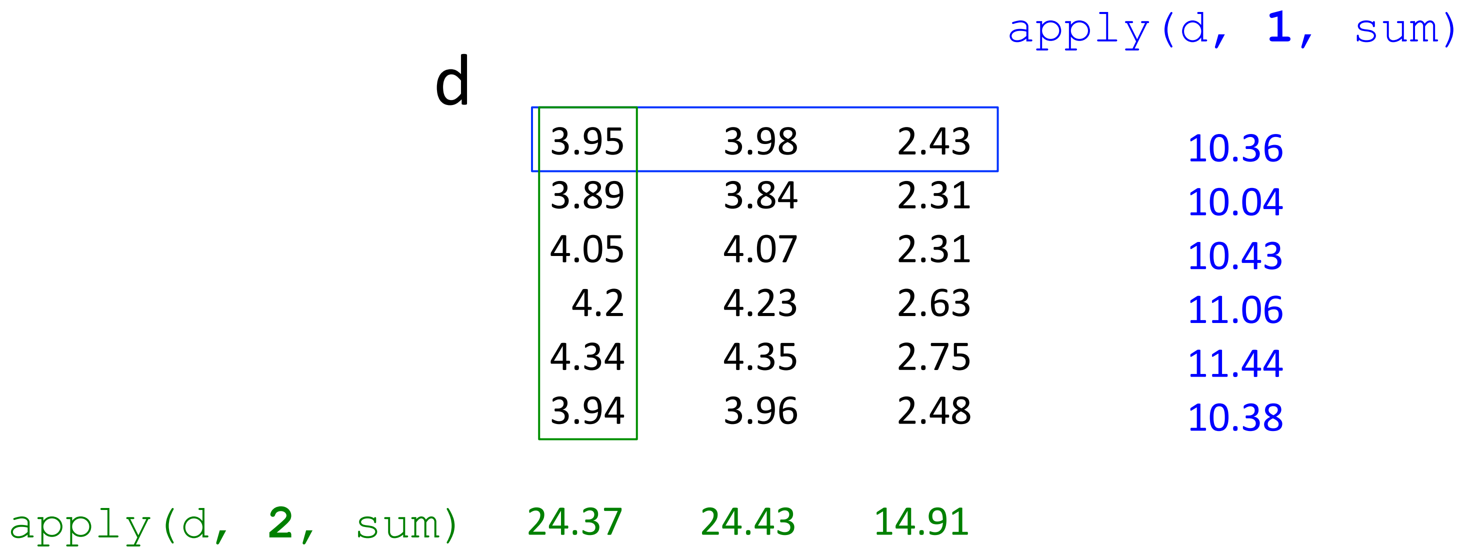
- vapply
- rapply
- ...

goal: to simplify coding and improve computation efficiency

apply()

- **apply(X, MARGIN, FUN, ...)**

apply a function to margins of an array or matrix.



				`apply(d, 1, sum)`
d				
	3.95	3.98	2.43	10.36
	3.89	3.84	2.31	10.04
	4.05	4.07	2.31	10.43
	4.2	4.23	2.63	11.06
	4.34	4.35	2.75	11.44
	3.94	3.96	2.48	10.38
`apply(d, 2, sum)`	24.37	24.43	14.91	

```
rowSums
colSums
```

apply - example

```
> head(diamonds)
  carat       cut color clarity depth table price    x    y    z
1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48
```

```
> apply(diamonds[, c("carat", "price")], 2, mean)
       carat        price
   0.7979397 3932.7997219
```

combine your own function with apply

```
sumsqrt <- function(x) {
   sum(sqrt(x))
}
apply(d, 1, sumsqrt)
```

or

```
apply(d, 1, function(x) sum(sqrt(x)))
```

3.95	3.98	2.43	5.54
3.89	3.84	2.31	5.45
4.05	4.07	2.31	5.55
4.2	4.23	2.63	5.73
4.34	4.35	2.75	5.83
3.94	3.96	2.48	5.55

sapply and lapply

sapply () and lapply()

work in a similar way, calling the specified function for each item of a list or vector.

```
> sapply(1:3, function(x) x^2)
[1] 1 4 9
```

lapply returns a list rather than a vector:

```
> lapply(1:3, function(x) x^2)
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9
```

mapply

`mapply()`

vectorize arguments to a function that is not usually accepting vectors as arguments.

```
> rep(1:3, 3)
[1] 1 2 3 1 2 3 1 2 3

> mapply(rep, 1:3, 3)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
[3,]    1    2    3

> mapply(rep, 1:3, 3:1)
[[1]]
[1] 1 1 1
[[2]]
[1] 2 2
[[3]]
[1] 3
```

1. apply each element from the 3rd argument to each element in the 2nd argument using the function specified in the 1st argument

2. combine them by column or organize them in a data frame or list format

tapply

- **tapply()**

Applying *a function* to each element of *a vector* given by the category of each element, provided by *the other vector*.

```
> head(diamonds)
  carat       cut color clarity depth table price    x    y    z
1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48

> tapply(diamonds$price, diamonds$cut, mean)
     Fair      Good Very Good   Premium     Ideal 
 4358.758  3928.864  3981.760  4584.258  3457.542 
```

aggregate

aggregate(X, by, FUN, ...)

Splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

```
carat       cut color clarity depth table price     x    y    z
1  0.23     Ideal     E     SI2  61.5    55    326 3.95 3.98 2.43
2  0.21   Premium     E     SI1  59.8    61    326 3.89 3.84 2.31
3  0.23      Good     E     VS1  56.9    65    327 4.05 4.07 2.31
4  0.29   Premium     I     VS2  62.4    58    334 4.20 4.23 2.63
5  0.31      Good     J     SI2  63.3    58    335 4.34 4.35 2.75
6  0.24 Very Good     J    VVS2  62.8    57    336 3.94 3.96 2.48
```

```
> aggregate(diamonds$price, by=list(diamonds$cut), FUN=mean)
 Group.1        x
1      Fair 4358.758
2      Good 3928.864
3 Very Good 3981.760
4   Premium 4584.258
5     Ideal 3457.542
```

```
> tapply(diamonds$price, diamonds$cut, FUN=mean)
 Fair      Good Very Good   Premium     Ideal
 4358.758  3928.864  3981.760  4584.258  3457.542
```

table

- **table()**

Determining counts for each category

```
> head(diamonds)
  carat       cut color clarity depth table price    x    y    z
1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48


> table(diamonds$cut)

     Fair       Good Very Good    Premium      Ideal
     1610       4906     12082      13791      21551
```

Outline

- your own functions
- apply family
- Other useful functions
- Simple statistical test

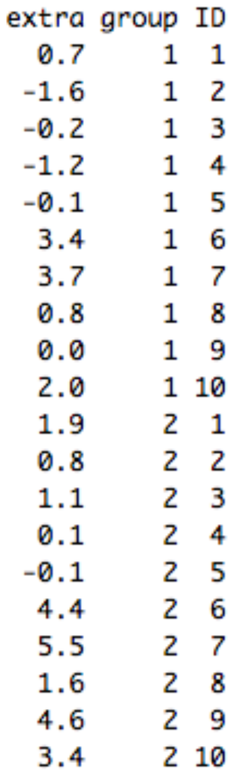
t-test

t.test

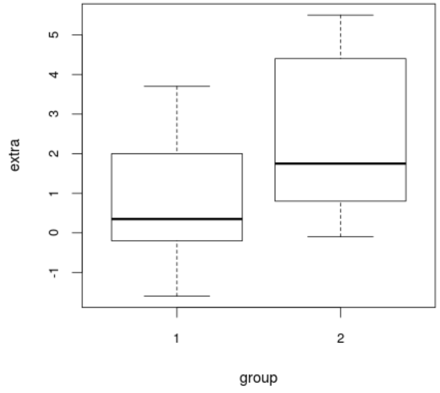
Performs one and two sample t-tests on vectors of data.

```
# Student's sleep data
plot(extra ~ group, data = sleep)

# t-test
with(sleep, t.test(extra[group == 1],
extra[group == 2]))

# Formula
t.test(extra ~ group, data = sleep)
```

data: sleep



```
extra group ID
  0.7     1  1
 -1.6     1  2
 -0.2     1  3
 -1.2     1  4
 -0.1     1  5
  3.4     1  6
  3.7     1  7
  0.8     1  8
  0.0     1  9
  2.0     1 10
  1.9     2  1
  0.8     2  2
  1.1     2  3
  0.1     2  4
 -0.1     2  5
  4.4     2  6
  5.5     2  7
  1.6     2  8
  4.6     2  9
  3.4     2 10
```



Linear models (I)

Fitting a linear model

lm(formula, data = data.frame)

```
pc <- lm(price ~ carat, data=diamonds)
summary(pc)
```

```
Residuals:
     Min       1Q   Median       3Q      Max 
-18585.3   -804.8    -18.9    537.4  12731.7 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -2256.36      13.06  -172.8   <2e-16 ***
carat        7756.43      14.07   551.4   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 

Residual standard error: 1549 on 53938 degrees of freedom
Multiple R-squared:  0.8493,  Adjusted R-squared:  0.8493 
F-statistic: 3.041e+05 on 1 and 53938 DF,  p-value: < 2.2e-16
```

ANOVA (I)

ANOVA

anova(model)

```
pcc <- lm(price ~ carat + cut, data=diamonds)
anova(pcc)
```

```
Analysis of Variance Table

Response: price
             Df     Sum Sq    Mean Sq   F value    Pr(>F)
carat         1 7.2913e+11 7.2913e+11 319162.11 < 2.2e-16 ***
cut           4 6.1332e+09 1.5333e+09    671.17 < 2.2e-16 ***
Residuals 53934 1.2321e+11 2.2845e+06
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

ANOVA (II)

Comparing two models

anova(model1, model2)

```
pc <- lm(price ~ carat, data=diamonds)
pcc <- lm(price ~ carat + cut, data=diamonds)
anova(pc, pcc)
```

```
Analysis of Variance Table

Model 1: price ~ carat
Model 2: price ~ carat + cut
  Res.Df        RSS Df  Sum of Sq      F    Pr(>F)
1  53938 1.2935e+11
2  53934 1.2321e+11  4 6133201436 671.17 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

chi-square test

chisq.test

```
d <- c(12, 36, 24, 70)
dm <- matrix(d, nrow=2, byrow=T)
chisq.test(dm)
```

```
data:  dm
X-squared = 0, df = 1, p-value = 1
```

	B	
A	12	36
	24	70

Online resources

"apply" function family

- https://www.datacamp.com/community/tutorials/r-tutorial-apply-family#gs.YUI=Luc

Statistical modeling with R

- https://www.datacamp.com/courses/statistical-modeling-in-r-part-1
- http://www.analyticsforfun.com/2014/06/performing-anova-test-in-r-results-and.html