生物信息学系
Department of Bioinformatics

Lesson 4

R: Data Transformations
Strings and Dates

Xiaoqin Yang

School of Biology and Basic Medical Sciences
Soochow University

yangxiaoqin@suda.edu.cn

Splitting a Vector into Groups

- **Problem**
  - You have a vector. Each element belongs to a different group, and the groups are identified by a grouping factor. You want to split the elements into the groups.

- **Solution**
  - Suppose the vector is x and the factor is f. You can use the **split** function:

    ```
    > groups <- split(x, f)
    ```

  - You can also use the **unstack** function:

    ```
    > groups <- unstack(data.frame(x,f))
    ```

  - Both functions return a list of vectors, where each vector contains the elements for one group.

Splitting a Vector into Groups

```
>  library(MASS)
> g <- split(Cars93$MPG.city, Cars93$Origin)
> g
$USA
 [1] 22 19 16 19 16 16 25 25 19 21 18 15 17 17 20 23 20 29 23 22 17 21 18 29
20
[26] 31 23 22 22 24 15 21 18 17 18 23 19 24 23 18 19 23 31 23 19 19 19 28

$`non-USA`
 [1] 25 18 20 19 22 46 30 24 42 24 29 22 26 20 17 18 18 29 28 26 18 17 20 19
29
[26] 18 29 24 17 21 20 33 25 23 39 32 25 22 18 25 17 21 18 21 20

> median(g[[1]])
[1] 20
> median(g[[2]])
[1] 22
```

Applying a Function to Each List Element

- **Problem**
  - You have a list, and you want to apply a function to each element of the list.

- **Solution**
  - Use either the **lapply** function or the **sapply** function, depending upon the desired form of the result. **lapply** always returns the results in list, whereas **sapply** returns the results in a vector if that is possible:

```
> lst <- lapply(lst, fun)
> vec <- sapply(lst, fun)
```

Applying a Function to Each List Element

```
> scores <- list(S1, S2, S3, S4)
> lapply(scores, length)
[[1]]
[1] 36

[[2]]
[1] 39

[[3]]
[1] 38

[[4]]
[1] 36
```

Applying a Function to Each List Element

```
> sapply(scores, length)
[1] 36 39 38 36
> sapply(scores, mean)
[1] 88.77778 89.79487 89.23684 88.86111
> sapply(scores, sd)
[1]  7.720515 10.543592  7.178926  8.208542
> sapply(scores, range)
     [,1] [,2] [,3] [,4]
[1,]   68   60   75   63
[2,]   98  100   99   99
```

Applying a Function to Every Row

- **Problem**
  - You have a matrix. You want to apply a function to every row, calculating the function result for each row.

- **Solution**
  - Use the **apply** function. Set the second argument to 1 to indicate row-by-row application of a function:

    ```
    > results <- apply(mat, 1, fun)
    ```

  - The **apply** function will call fun once for each row, assemble the returned values into a vector, and then return that vector.

Applying a Function to Every Row

```
> mat <- matrix(sample(1:100, 20, replace=TRUE), 4,5)
> apply(mat, 1, mean)
[1] 35.4 64.8 39.4 75.4
> apply(mat, 1, range)
     [,1] [,2] [,3] [,4]
[1,]    5   38   15   37
[2,]   71   86   78   99
```

Applying a Function to Every Column

- **Problem**
  - You have a matrix or data frame, and you want to apply a function to every column.
- **Solution**
  - For a matrix, use the **apply** function. Set the second argument to 2, which indicates column-by-column application of the function:

    ```
    > results <- apply(mat, 2, fun)
    ```
  - For a data frame, use the **lapply** or **sapply** functions:

    ```
    > lst <- lapply(dfrm, fun)
    > vec <- sapply(dfrm, fun)
    ```
  - You can use apply on data frames, too, but only if the data frame is homogeneous.

Applying a Function to Every Column

```
> head(Cars93)
> sapply(Cars93, class)
```

```
##       Manufacturer              Model               Type
##           "factor"           "factor"           "factor"
##         Min.Price              Price          Max.Price
##          "numeric"          "numeric"          "numeric"
##           MPG.city        MPG.highway             AirBags
##          "integer"          "integer"           "factor"
##         DriveTrain          Cylinders         EngineSize
##           "factor"           "factor"          "numeric"
##         Horsepower                RPM       Rev.per.mile
##          "integer"          "integer"          "integer"
##    Man.trans.avail Fuel.tank.capacity         Passengers
##           "factor"          "numeric"          "integer"
##             Length          Wheelbase              Width
##          "integer"          "integer"          "integer"
##        Turn.circle     Rear.seat.room       Luggage.room
##          "integer"          "numeric"          "integer"
##             Weight             Origin               Make
##          "integer"           "factor"           "factor"
```

Applying a Function to Groups of Data

- **Problem**
  - How to process the data by groups?

- **Solution**
  - Create a grouping factor that identifies the group of each corresponding datum. Then use the **tapply** function, which will apply a function to each group of data:

    ```
    > tapply(x, f, fun)
    ```

Applying a Function to Groups of Data

```
> library(MASS)
> tapply(Cars93$MPG.city, Cars93$Origin, mean)
     USA  non-USA
20.95833 23.86667
> tapply(Cars93$MPG.city, Cars93$Origin, length)
  USA non-USA
 48      45
> tapply(Cars93$MPG.city, Cars93$Origin, sum)
 USA non-USA
 1006    1074
```

Applying a Function to Groups of Rows

- **Problem**
  - You want to apply a function to groups of rows within a data frame.

- **Solution**
  - Define a grouping factor for every row in your data frame. For each such group of rows, the **by** function puts the rows into a temporary data frame and calls your function with that argument. The **by** function collects the returned values into a list and returns the list:

```
> by(dfrm, fact, fun)
```

Applying a Function to Groups of Rows

```
> dfrm <- data.frame(f=c(rep("A",50), rep("B",30), rep("C", 20)),
+          v=c(rnorm(50)-0.2,rnorm(30), rnorm(20)+0.2))
> by(dfrm, dfrm$f, summary)
dfrm$f: A
 f           v
 A:50    Min.      :-2.2870
 B: 0    1st Qu.  :-0.8492
 C: 0    Median  :-0.2448
         Mean     :-0.2351
         3rd Qu. : 0.4240
         Max.     : 1.4526
------------------------------------------------------------
dfrm$f: B
f           v
A: 0    Min.      :-1.6488
B:30    1st Qu.  :-0.1468
… …
```

Applying a Function to Parallel Vectors or Lists

- **Problem**
  - You have a function, say **f**, that takes multiple arguments. You want to apply the function element-wise to vectors and obtain a vector result. Unfortunately, the function is not vectorized; that is, it works on scalars but not on vectors.

- **Solution**
  - Use the **mapply** function. It will apply the function f to your arguments element-wise:

    > mapply(f, vec_1, vec_2, ..., vec_N)

Applying a Function to Parallel Vectors or Lists

```
> gcd <- function(a,b) {
+         if (b == 0) return(a)
+         else return(gcd(b, a %% b)) }
> gcd(c(25,9,12), c(9,6,3))
[1]   1 NaN NaN
Warning messages:
1: In if (b == 0) return(a) else return(gcd(b, a%%b)) :
  the condition has length > 1 and only the first element will be used
… …
> mapply(gcd, c(25,9,12), c(9,6,3))
[1] 1 3 3
```