

## [連載]フリーソフトによるデータ解析・マイニング 第4回

# Rでの関数オブジェクト

### 1. 既存の関数オブジェクト

Rには、通常頻繁に使われているデータの操作や標準的な統計計算は、ほとんど関数オブジェクトとしてプログラミングされている。ここでは、関数オブジェクトを略して関数と呼ぶ。データの処理および解析はその関数を用いて行う。Rは、積木の種々の形状の木片を積み上げて色々な物の形をつくるのと同じく、一つひとつの関数をパーツとみなし、それを組み合わせることにより、複雑なデータの処理・解析を行うように設計されている。関数の組み合わせの方法について例を用いて説明する。

例えば、7つの異なるグループに次の質問についてアンケート調査を行い、表1のような結果が得られたとする。

**質問：**今の生活がむなしく感じられることがある。

1. よくある    2. ときどきある    3. あまりない    4. ない

表1 質問の回答結果(合計 2407 人)

グループ	回答 1	回答 2	回答 3	回答 4
A	101	120	70	35
B	153	162	88	46
C	89	135	78	24
D	26	49	42	17
E	36	70	30	16
F	167	216	144	71
G	125	143	89	65

表の中の回答1は、質問項目の1の回答を指す。表の中の数値は回答の人数である。このようなアンケート調査結果は一般的には、まず次に示すように第1列はグループの名前、第2列は回答番号としたデータセットを作成する。Excelで入力された場合は、CSV形式に保存し、Rに読み込むことができる。例えば、Rで作成されたデータオブジェクトの名前がlifeであるとする。そのデータ形式を次に示す。

```
> life
No. グループ 回答番号
1     A      2
2     A      3
3     B      2
<後略>
```

このような調査データの分析は、通常次ページ表 1 のようなクロス表に集計し分析を行う。R では関数 **table** を次のように用いると簡単にクロス表を作成することができる。

```
>table(life[,1],life[,2])
```

読者の手元には life の素データがないので表 1 のクロス表のデータを次のように入力し、用いることにする。

```
>life1<-matrix(0,7,4)
>life1[,1]<-c(101,153,89,26,36,167,125)
>life1[,2]<-c(120,162,135,49,70,216,143)
>life1[,3]<-c(70,88,78,42,30,144,89)
>life1[,4]<-c(35,46,24,17,16,71,65)
```

このようなデータについて分析を行うためには、各グループの回答人数の合計を求める必要がある。合計を求める関数は **sum** である。グループ A は第 1 行なので、その合計は次のように求めることができる。

```
> sum(life1[1,])
[1] 326
```

同じ方法で、グループ B、C、D、E、F、G の合計を求めることができる。しかし、このような方法では 7 回のコマンドラインを実行することが必要となる。同じ作業を何十回、何百回行わなければならないのは非常に非効率である。R には行、列の操作をまとめて行う関数 **apply** がある。例えば、データ life1 のグループ別の合計は次のコマンドで求めることができる。返された値はそれぞれグループ A、B、C、D、E、F、G の合計である。

```
> apply(life1,1,sum)
[1] 326 449 326 134 152 598 422
```

関数 **apply** の引数は、まずデータ、その次は求める統計量が行単位であるか、列単位であるかの指定(行の場合は 1、列の場合は 2)、最後には求めたい統計量(例えば、平均、中央値、最大値、最小値、範囲、分散など)を指定する。



また、表1のようなデータにおいては、しばしば百分率データを用いて比較分析を行う。百分率は個々のデータをそのグループの合計で割り100を乗じた値である。life1の百分率は次のように求めることができる。

```
>(life1/apply(life1,1,sum))*100
```

このコマンドラインを実行すると小数点右数桁のデータが返される。小数点右の桁数が多いと精度はよいが、扱うのに不便である場合もある。問題によっては、小数を丸める(四捨五入)必要がある。小数を丸める関数として **round** がある。関数 **round** の書式を次に示す。

```
round(x,桁数)
```

式の中のxはデータ、あるいはデータオブジェクトである。表1の百分率のデータを小数点右2桁まで丸めるコマンドラインおよび返された結果を次に示す。

```
>round((life1/apply(life1,1,sum))*100,2)
```

```
  [,1] [,2] [,3] [,4]
[1,] 30.98 36.81 21.47 10.74
[2,] 34.08 36.08 19.60 10.24
[3,] 27.30 41.41 23.93  7.36
[4,] 19.40 36.57 31.34 12.69
[5,] 23.68 46.05 19.74 10.53
[6,] 27.93 36.12 24.08 11.87
[7,] 29.62 33.89 21.09 15.40
```

このデータは例として用いるので次のように life2 という名前で保存しておく。

```
>life2<-round((life1/apply(life1,1,sum))*100,2)
```

データ life2 について、個々のデータとその列の平均値との差を求めることを考えてみよう。平均を求める関数は **mean** である。回答1の個々のデータとその列の平均値との差は、

```
> life2[,1]-mean(life2[,1])
```

```
[1] 3.41 6.51 -0.27 -8.17 -3.89 0.36 2.05
```

となる。データセット life2 のすべての列に対するこのような計算は、次のコマンドで行うことができる。

```
>t(life2)-apply(life2,2,mean)
```

このように求めた結果は、第1行が回答1、第2行が回答2、第3行が回答3、第4行回答4となる。表1と同じのように列を回答1、2、3、4にするためには出力結果を転置する必要がある。また、小数点以下2桁まで丸めると次のようになる。

```
> round(t(t(life2)-apply(life2,2,mean)),2)
```

```
      [,1] [,2] [,3] [,4]
[1,]  3.41 -1.32 -1.57 -0.52
[2,]  6.51 -2.05 -3.44 -1.02
[3,] -0.27  3.28  0.89 -3.90
[4,] -8.17 -1.56  8.30  1.43
[5,] -3.89  7.92 -3.30 -0.73
[6,]  0.36 -2.01  1.04  0.61
[7,]  2.05 -4.24 -1.95  4.14
```

関数を組み合わせた場合、関数の実行の優先順位は丸括弧 ( ) を単位にコマンドの中心部から外側の方向に、階層的に中心部から計算結果を次の層にわたす。上記のコマンドの関数の実行順序を番号で次に示す。

```
round(t(t(life2)-apply(life2,2,mean)),2)
```

```
t(life2)
apply(life2,2,mean)
t(life2)-apply(life2,2,mean)
t(t(life2)-apply(life2,2,mean))
round(t(t(life2)-apply(life2,2,mean)),2)
```

## 2. 自作の関数オブジェクト

Rでデータ解析を行う際、すでに関数のオブジェクトがある場合はその関数を用いればよいが、千差万別なデータ処理および解析の方法をすべて関数として用意して置くことは不可能である。解析したい方法が関数として用意されていない場合は、前節の説明のように既存の関数を組み合わせてデータを処理・解析する。頻繁に同じの処理を行う場合は、既存関数を組み合わせた関数オブジェクトを作成しておくことと便利である。Rでは、関数 function を用いて関数オブジェクトを作成する。

## (1) function の構造

関数 `function` によるプログラムの構文を次に示す。引数は複数項目でもかまわない。

```
関数の名前<-function(引数){  
  プログラムの本体  
}
```

例えば、前章で求めた個々のデータとその列の平均との差を求める関数は次のように作成する。

```
>hensa<-function(x){  
  round(t(t(x)-apply(x,2,mean)),2)  
}
```

この簡単なプログラムが `hensa` という自作の関数オブジェクトである。この関数の引数は `x` のみで、`x` はデータ行列、あるいはデータフレームである。返す結果は、個々のデータとその列の平均との差である。より複雑な関数(プログラム)を作成するためには、比較、繰り返しの処理などを行う必要がある。

## (2) 比較・判断文

データを処理するときには、与えられた条件と比較し、判断を下さなければならないケースにしばしば遭遇する。R での比較・判断文の書式を次に示す。

```
if(条件式) 処理1 else 処理2
```

この文を `if` 文とも呼ぶ。 `if` 文の例として、データ `life2` の第 1 行第 1 列のデータが、その列の平均値より大きければ `TRUE`(真)、そうでなければ `FALSE`(偽)を返すコマンドと実行結果を次に示す。

```
>if(life2[1,1]>mean(life2[,1])) print( " TRUE " )else print( " FALSE " )  
[1] "TRUE"
```

## (3) 繰り返し文

コンピュータに同じの作業を繰り返し実行させる指示文を繰り返し文という。R 言語には 3 種類の繰り返し文 `for`、`while`、`repeat` がある。ここでは、`for` 文のみについて説明する。`for` 文は、変数を用いて繰り返しの回数をコントロールする。`for` 文の書式を次に示す。

```
for(変数 in 変数の開始値 : 変数の終了値){  
  繰り返し処理される部分  
}
```

前節の if 文を説明する際の

```
if(life2[1,1]>mean(life2[,1])) print( " TRUE " )else print( " FALSE " )
```

は、1 行 1 列のみについて比較・判断の処理を行った。1 列の 7 行のデータについてこのような処理は、for 文を次のように用いて行うことができる。

```
>for(i in 1:7){  
  if(life2[i,1]>mean(life2[,1])) print("TRUE")else print("FALSE")  
}
```

for 文を次のようにもう一回使用するとデータセットにおける、個々のデータがその列の平均値より大きいと “ TRUE ”、小さいと “ FALSE ” を返す処理ができる。

```
>for(j in 1:4)  
  for(i in 1:7){  
    if(life2[i,j]>mean(life2[,j])) print("TRUE")else print("FALSE")  
  }
```

ここでは、for 文の説明のため二重の for 文を用いているが、これは非常に効率が悪いプログラムである。R は、for 文のような繰り返し文の処理効率がよくないため、データサイズが大きい場合は繰り返し文の重ねをなるべく避けるのが賢明である。

### 3. プログラムの入力と編集

R でのプログラムの入力方法はいくつかある。

#### (1) 直接 R のコンソールに入力

コンソール上で 1 行の入力が終わったら改行する。入力されたプログラムが終了していない場合は、自動的にコマンドラインの先頭に記号 “ + ” が返される。記号 “ + ” の右にプログラム文を続けて入力する。この方法は、入力途中でプログラムにエラーがあると、入力した部分は R に保存されないため、行数が多いプログラムには向いていない。

#### (2) テキストエディタを使用

メモ帳、秀丸、ワードなどのテキストエディタ上で、プログラムを作成し、そのプログラム

をコピーし、R のコンソールに貼り付ける。プログラムにエラーがあり、正常に作動しなかったら、テキストエディタ上で修正し、コピー、貼り付けの作業を繰り返す。

### (3)関数 fix、edit を使用

関数 fix を次のように用いると

```
>fix(program1)
```

図 1 のようなテキストエディタが開かれる。コマンドラインの中の program1 は自作関数の名前で、各自が自由に名前をつけることができる。



図 1 関数編集エディタ画面

図 2 のように開かれたテキストエディタの括弧()の中に引数、括弧{}の中にプログラム本文を入力する。入力が終わったら上書保存し、テキストエディタを終了すると、プログラムにエラーがない場合、関数オブジェクトが作成される。さらに編集を行う場合は、fix(program1)を実行するとテキストエディタが開かれる。



図 2 自作関数オブジェクトの編集画面

関数 edit を用いて関数オブジェクトを編集することも可能である。ただし、その際には、次のように付値を行わないと編集されたプログラムが保存されない。

```
>program1<-edit(program1)
```

作成した関数オブジェクトを実行するためには、開かれているテキストエディタを終了しなければならない。最後に、通常頻繁に使われる関数とその簡潔な説明と例を次ページ表 2 に示す。

表2 頻繁に使う関数

	関数名	説明と例		関数名	説明と例
基本統計量	sum	合計、 $\sum_{i=1}^n x_i$	名前	names	データ項目に名前を付ける
	mean	算術平均、 $\frac{1}{n} \sum_{i=1}^n x_i$		colnames	列に名前を付ける
	max	最大値		rownames	行に名前を付ける
	min	最小値	初等代数	abs	絶対値 $abs(-2) =  -2  = 2$
	range	範囲(最大値 - 最小値)		exp	自然指数 $exp(1) = 2.7182\dots$
	median	中央値		sqrt	平方根 $sqrt(4) = \sqrt{4} = 2$
	var	分散 $\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$		log	自然対数、 $log(exp(1)) = 1$
	apply	行・列別の基本統計量を返す		log10	常用対数 $log10(10) = 1$
	table	個々のデータの出現頻度を返す。クロス表の作成		cos,sin,tan	三角関数
character	文字型	acos,asin,atan		逆三角関数	
complex	複素数型	round		四捨五入 $round(3.1415,3) = 3.142$	
integer	整数型	並べ替え		sort	昇順に並べ替える
logical	論理型		rev	逆の順に並べ替える	
numerical	実数型		ran	ランクを返す	
null	空データ型		sort.list	昇順に整列された場合のデータの順番を返す	
データ型とデータセットの操作	vector	ベクトル型	結合と編集	rbind	データを縦に結合
	data.frame	データフレーム型		cbind	データを横に結合
	matrix	行列型		edit	エディタを起動
	array	配列型		fix	オブジェクトの編集
	list	リスト型	data.entry	表計算風のインタフェース	
	class	オブジェクトの属性を返す	edit.data.frame	データオブジェクトを編集する	
	is.***** *****はデータ型の関数名	is.matrix(x)は、xが行列の場合は"TRUE"、そうではない場合は"FALSE"を返す	入出力	scan	ファイルを読み込む
	as.*****	as.matrix(x)は、データxを行列型に変換する		read.table	データフレーム型として読み込む
	c	データオブジェクトの作成		write	ファイルを出力する
	t	データを転置		sink	画面に返される結果をファイルとして出力する
データのサイズ	nrow	列の数を返す	rm	rm(x)は、オブジェクトxを削除する	
	ncol	行の数を返す	ls	Ls()は、オブジェクトのリストを返す	
	length	データオブジェクトの長さを返す			
	dim	行列、配列のサイズを返す			