

## [連載]フリーソフトによるデータ解析・マイニング 第3回

# Rでのデータの編集と演算

データの解析や処理を行うためには、データについて、その構造の確認や編集などが必要である場合がある。

## 1. データの編集

### (1) データのサイズの確認

ここでは入力されているデータ `sales2` を用いて説明する。データ `sales2` は 2 行 5 列で、データフレーム型で入力されている。

```
> sales2
  Cherry Apple Grape Banana Other
A    15    20    25    10    30
B    10    25    20    25    20
```

次に示すように関数 `ncol`, `nrow`, `dim` を用いると、データフレームおよび行列の行数、列数や配列の次元を求めることが可能である。その例を次に示す。

```
> nrow(sales2)
[1] 2
> ncol(sales2)
[1] 5
> dim(sales2)
[1] 2 5
```

### (2) データの編集エディタ

Rには、データ編集のための関数 `edit`、`edit.data.frame`、`data.entry`、`fix` などがある。`edit` は「テキストエディタの起動」、`edit.data.frame` は「データフレームと行列の編集」、`data.entry` は「データ入力のための表計算ソフト風のインタフェイス」、`fix` は「オブジェクトの編集」用として作成されている。しかし、最近のバージョンでは、データオブジェクトが作成されている場合は、上記の関数のいずれもデータを表計算ソフト風のエディタで編集を行うことができるようになっている。例えば、コマンド

```
>edit(sales2)
```

を実行すると、次のページ図1のような編集画面が開かれる。編集画面はExcelほど便利

ではないが、セル単位で編集が可能である。



	Cherry	Apple	Grape	Banana	Other
1	15	20	25	10	30
2	10	25	20	25	20
3					

図1 データ編集エディタの画面

関数 `edit` で編集したデータを保存するためには、付値の手続きが必要である。`sales2` を `edit` で編集し、その結果を `temp` という名前前で保存したいとき、`sales2` に上書きしたいときの例を次に示す。

```
>temp<-edit(sales2)
>sales2<-edit(sales2)
```

関数 `edit` を用いてデータセットの行、あるいは列を付け加えたいときには、データセットをデータフレーム型にした方がよい。

関数 `fix` を用いてデータを編集すると、編集された新しい結果が直接上書きされるので付値の手順は必要としない。

このように、異なるデータ編集用の関数はそれぞれ特徴を持っている。

### (3) データセットの結合

データフレームや行列などの行の追加や列の追加が必要な場合がある。Rにはデータフレームおよび行列の結合の関数 `rbind`、`cbind` がある。例えば、データフレーム `sales2` の最後の行の後にデータ 20、30、15、20、15 を追加したいときには次のように行う。

```
> rbind(sales2,c(20,30,15,20,15))
  Cherry Apple Grape Banana Other
A      15    20    25     10     30
B      10    25    20     25     20
      20    30    15     20     15
```

関数 `rbind`、`cbind` は同じ長さの行、あるいは列のデータフレームや行列を結合することが可能である。例えば、行列 A、B があるとする。

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 4 & 3 \\ 2 & 4 \end{bmatrix}$$

データ行列 A、B の作成と `rbind(A,B)`、`cbind(A,B)` の実行結果を示す。

```
> A<-matrix(0,2,2)
> A [1,]<-c(1,1)
> A [2,]<-c(2,4)
> B<-matrix(0,2,2)
> B [1,]<-c(4,3)
>B [2,]<-c(2,4)
> rbind(A,B)
```

```
      [,1] [,2]
[1,]    1    1
[2,]    2    4
[3,]    4    3
[4,]    2    4
```

```
> cbind(A,B)
```

```
      [,1] [,2] [,3] [,4]
[1,]    1    1    4    3
[2,]    2    4    2    4
```

関数 `rbind` を用いるためにはデータフレーム、あるいは行列の列数が一致し、`cbind` は行数が一致しなければならない。

#### (4) データの並べ替え

ベクトル、行列の要素を並べ替える関数は `sort` である。`sort` はデータを昇順に並べ替える。

```
> sort(c(3,2,4,1))
[1] 1 2 3 4
```

また、R にはデータを逆の順序で並べ替える関数 `rev` がある。

```
> rev(c(3,2,4,1))
[1] 1 4 2 3
```

データフレーム、あるいは行列を、ある行、列を基準にし、並べ替えるのには、関数 `sort.list` あるいは `order` を用いて行うことが可能である。

例えば、データセット `sales2` を、第 1 行を基準にし、昇順に並べ替えは次のように行うことができる。関数 `order` を用いても結果は同じである。

```
> sales2[,sort.list(sales2[1,])]
```

	Banana	Cherry	Apple	Grape	Other
A	10	15	20	25	30
B	25	10	25	20	20

## 2. データの演算

Rでは、算術演算、論理演算、比較演算、行列の演算などの演算を行うことができる。

### (1) 算術演算

主な算術演算は、加算、減算、乗算、除算、べき算で、それぞれの演算は記号 +、-、\*、/、^を用いる。演算の名称と記号との対応関係およびその演算例を表 1 に示す。

表 1 算術演算の表記

名称	表記	例	結果
加算	+	3+2	5
減算	-	3 - 2	1
乗算	*	3*2	6
除算	/	3/2	1.5
べき算	^	2^3	8
		8^(1/3)	2
除算の商	%/%	5%/%2	2
除算の余	%%	5%%%2	1

表 1 に示す算術演算はベクトル、データフレーム、行列にも有効である。ベクトル、データフレーム、行列では、対応する要素の間で表 1 の例のように算術演算を行う。両ベクトルの算術演算のコマンドおよびその結果を次に示す。

```

> x<-c(1,2,3,4)
> y<-c(4,2,3,1)
> x+y
[1] 5 4 6 5
> x-y
[1] -3 0 0 3
> x*y
[1] 4 4 9 4
> x/y
[1] 0.25 1.00 1.00 4.00
> x^2
[1] 1 4 9 16
> x%/%2
[1] 0 1 1 2
> x%%%2

```

## [1] 1 0 1 0

行列に関しては、作成した行列 A, B を用いて演算を行ってみてください。算術演算における優先順位は次の通りである。

括弧

べき算(^)

除算の商と余(%%、%)

乗算と除算(\*、/)

加算と減算(+、-)

## (2) 比較演算

比較演算は、より大きいか、小さいか、等しいかなどを比較するための演算である。表 2 に比較演算子の名称と記号との対応関係および例を示す。

表 2 比較演算の表記

名称	表記	例	結果
より大	>	3>2	TRUE
より大か等しい	>=	3>=4	FALSE
より小	<	3<2	FALSE
より小か等しい	<=	3<=4	TRUE
等しい	==	3==4	FALSE
等しくない	!=	3!=4	TRUE

前節で作成したベクトル x, y を用いた比較演算とその結果を次に示す。

```
> x>y
[1] FALSE FALSE FALSE TRUE
> x>=y
[1] FALSE TRUE TRUE TRUE
> x!=y
[1] TRUE FALSE FALSE TRUE
> x==y
[1] FALSE TRUE TRUE FALSE
```

## (3) 論理演算

論理演算は、真(TRUE)、偽(FALSE) の 2 値の演算である。表 3 に論理演算の名称と記号との対応関係および例を示す。表の中の T は TRUE、F は FALSE の略である。

表 3 論理演算の記号

名称	表記	例	結果
否定(NOT)	!	!(F)	TRUE
積(AND)	&	T & T	TRUE
積(AND)	&&	T && T	TRUE
和(OR)		T   F	TRUE
和(OR)		T    F	TRUE
排他的論理和	xor	xor(T,T)	FALSE

論理ベクトルの作成およびその演算と結果を次に示す。

```
> Lx<-c(T,T,F,F)
```

```
> Ly<-c(T,F,F,T)
```

```
> !(Lx)
```

```
[1] FALSE FALSE TRUE TRUE
```

```
> Lx & Ly
```

```
[1] TRUE FALSE FALSE FALSE
```

```
> Lx && Ly
```

```
[1] TRUE
```

```
> Lx | Ly
```

```
[1] TRUE TRUE FALSE TRUE
```

```
> Lx || Ly
```

```
[1] TRUE
```

```
> xor(Lx,Ly)
```

```
[1] FALSE TRUE FALSE TRUE
```

以上の例からわかるように、&, |, はベクトルのすべての要素について論理演算を行うが、&&, || はベクトルにおける最初の要素の間の論理演算が成り立つと、その次の演算は行わない。

### 3. 行列の演算

#### (1) 行列の加、減算

同じ数の行と列をもつ複数の行列の間には加、減算を行うことが可能である。行列の加、減算は、行列の対応する位置の要素について次のように演算を行う。

$$A + B = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix} + \begin{bmatrix} 4 & 3 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 1+4 & 1+3 \\ 2+2 & 4+4 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 4 & 8 \end{bmatrix}$$

$$A - B = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix} + \begin{bmatrix} 4 & 3 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 1-4 & 1-3 \\ 2-2 & 4-4 \end{bmatrix} = \begin{bmatrix} -3 & -2 \\ 0 & 0 \end{bmatrix}$$

Rでは、行列 A、B の加、減算は算術演算の表記と同じである。

> A+B

```
      [,1] [,2]
[1,]    5    4
[2,]    4    8
```

> A-B

```
      [,1] [,2]
[1,]   -3   -2
[2,]    0    0
```

## (2) 行列の乗算

行列の乗算には、算術演算のように対応する行列の要素の間に乗算を行う演算と線形代数で定義されている積演算がある。

### 行列の算術乗算

ここでいう行列の算術演算とは、行列の要素の間に算術乗算を行うことである。前節に用いた行列 A、B を用いたその計算のコマンドと実行結果を次に示す。

> A\*B

```
      [,1] [,2]
[1,]    4    3
[2,]    4   16
```

### 行列のスカラー倍

行列 A の  $k$  倍は、行列 A のすべての要素を  $k$  倍にする。これを行列のスカラー倍という。スカラー (scalar) は行列ではなく数値であることである。例えば、行列 A の 2 倍は

$$2A = \begin{bmatrix} 2 \times 1 & 2 \times 1 \\ 2 \times 2 & 2 \times 4 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 4 & 8 \end{bmatrix}$$

となる。R での計算の書き式と結果を示す。

> 2\*A

```
      [,1] [,2]
[1,]    2    2
[2,]    4    8
```

## 行列の積演算

Rでの行列の積演算は、3種類(行列積、行列クロス積 **crossprod**、行列外積 **outer**) があるが、ここでは行列積のみについて説明する。例えば、次のような行列 A、D があるとする。

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix}, \quad D = \begin{bmatrix} 4 & 3 & 2 \\ 2 & 4 & 1 \end{bmatrix}$$

行列の積演算のルールは、左辺の行列における行の要素と右辺の行列の列に対応する要素の積の和を行列の要素とする。

例えば、上記の A の第 1 行は [1 1] で、D の第 1 列は  $\begin{bmatrix} 4 \\ 2 \end{bmatrix}$  である。  $1 \times 4 + 1 \times 2$  が積行列 AD の第 1 行、第 1 列の要素となる。

$$\begin{aligned} AD &= \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix} \times \begin{bmatrix} 4 & 3 & 2 \\ 2 & 4 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 \times 4 + 1 \times 2 & 1 \times 3 + 1 \times 4 & 1 \times 2 + 1 \times 1 \\ 2 \times 4 + 4 \times 2 & 2 \times 3 + 4 \times 4 & 2 \times 2 + 4 \times 1 \end{bmatrix} \\ &= \begin{bmatrix} 6 & 7 & 3 \\ 16 & 22 & 8 \end{bmatrix} \end{aligned}$$

行列の積演算の前提条件は、左辺の行列の列数が右辺の行列の行数と一致することである。この条件を満たさないと行列の積演算は成り立たない。行列 A は 2 列、D は 2 行であるので、行列 A と行列 D の積演算は可能であるが、積演算 DA は成り立たない。これは D が 3 列で、A が 2 行であるからである。左辺の行列が  $n$  行、右辺の行列が  $m$  列の場合、行列の積演算で得られた新たな行列は  $n$  行  $m$  列となる。

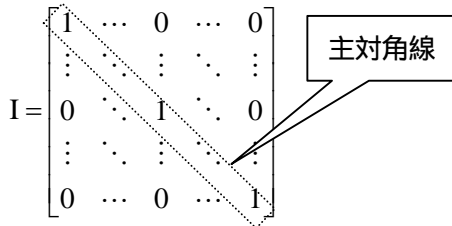
R では、コマンド `%*%` で行列の積演算を行う。行列 D の作成と積演算 AD、及びその結果を次に示す。

```
> D<-matrix(0,2,3)
> D[1,]<-c(4,3,2)
> D[2,]<-c(2,4,1)
> A%*%D
```

```
      [,1] [,2] [,3]
[1,]    6    7    3
[2,]   16   22    8
```

### (3)単位行列と逆行列

行列の主対角線の要素がすべて1で、その以外の要素がゼロである行列を単位行列という。単位行列は、一般的にI、あるいはEで表す。ここではIを用いることにする。

$$I = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & \cdots & 1 & \cdots & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$


算術演算では、 $k$  に  $\frac{1}{k}$  を乗じると  $k \times \frac{1}{k} = 1$  となる。これと似たように、行列にある行列を乗じると単位行列になる場合がある。

$$AZ = ZA = I$$

このとき、行列  $Z$  は行列  $A$  の逆行列と呼び、 $A^{-1}$  で表示する。その逆も言える。

$$AA^{-1} = A^{-1}A = I$$

ある行列の逆行列を求める計算は行列の積演算より煩雑である。ここではその計算の規則の説明は省略する。ほとんどの線形代数に関する専門書には逆行列の計算方法が説明されている。

R 言語での逆行列を求める関数は **solve** である。行列

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix}$$

の逆行列は次のように求める。

```
> solve(A)
     [,1] [,2]
[1,]   2 -0.5
[2,]  -1  0.5
```

行列 A とその逆行列の積は次に示すように単位行列になる。

```
>A%*%solve(A)
      [,1] [,2]
[1,]  1    0
[2,]  0    1
> solve(A)%*%A
      [,1] [,2]
[1,]  1    0
[2,]  0    1
```

#### (4) 固有値

$n$  行  $n$  列の正方行列  $A$  と要素が同時にゼロではない  $n$  次元のベクトル  $X$  について、

$$AX = \lambda X$$

が成り立つとき、スカラー係数  $\lambda$  を行列  $A$  の固有値(eigen value)、 $X$  を固有ベクトル(eigen vector)と言う。固有値、固有ベクトルはデータ解析に多く用いられている。Rには正方行列の固有値、固有ベクトルを求める関数 **eigen** が用意されている。例として、次の行列  $A$  の固有値、固有ベクトルを求めてみよう。

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix}$$

```
>A<-matrix(c(1,2,1,4),2,2)
> eigen(A)
$values
 [1] 4.5615528 0.4384472
$vectors
      [,1] [,2]
[1,] -0.2703230 -0.8719282
[2,] -0.9627697  0.4896337
```

行列が  $n$  次元のとき、固有値と固有ベクトルの最多数は  $n$  個である。固有値(values)は先頭から数え第 1 固有値、第 2 固有値・・・第  $n$  固有値、固有ベクトル(vectors)は列単位で左から第 1 固有ベクトル、第 2 固有ベクトル・・・第  $n$  固有ベクトルと呼ぶ。

得られた固有値と固有ベクトルを用いて、固有値、固有ベクトルの定義に用いた等式が成り立つかどうかを確認することができる。説明の便利のため、固有値を  $\lambda$ 、固有ベクトルを  $X$  というオブジェクトの名前にする。

```
>lamuda=eigen(A)$values
>X=eigen(A)$vectors
> A%*%X[,1]
      [,1]
[1,] -1.233093
[2,] -4.391725
> lamuda[1]*X[,1]
[1] -1.233093 -4.391725
```

返されている両結果が、縦と横に並べられた形式のこと以外には、対応する値はまったく同じである。これで第1固有値と固有ベクトル用いて場合、等式が成り立つことが確認できた。同じの方法で、第2固有値と固有ベクトルを用いた場合の結果を確認することができる。

