

[連載]フリーソフトによるデータ解析・マイニング 第2回

Rでのデータの入出力とパッケージ

データ解析・マイニングを行うためには、まずデータをツールに読み込まなければならない。Rでは、実数(numeric)、複素数(complex)、文字(character)、論理(logical)値などのデータをベクトル、行列、配列、データフレーム、リストなどの形式で扱うことができる。

Rでは、キーボードによって直接入力する方法と、R外部のファイルからRに読み込む方法がある。

1. 直接入力

(1)ベクトル

表1のような、データを入力することを例として説明する。

表1 果物の売上の割合(単位は%)

Cherry	Apple	Grape	Banana	Other
15	20	25	10	30

Rを起動し、Rのコンソールに

```
> sales<-c(15,20,25,10,30)
```

を入力し、[Enter]キーを押すと、salesに一行のデータ15,20,25,10,30が入力される。このsalesはデータセットの名前で、他のオブジェクトと区別可能な文字列であれば、どのような文字列にしてもかまわない。例えば、「hanbai」、「販売」とすることもできる。コマンドラインの">"は改行するたびに自動的に現れるコマンドラインの開始記号であるので、入力する必要はない。

コマンドラインのc()はデータをまとめる関数と呼ぶ。記述の便利のため以下には丸括弧を省略する。関数cを用いてデータを入力する際、入力データは単位ごとに半角のカンマ「,」で区切る。半角の不等記号「<」とハイフン「-」をつないだ「<-」の左側はオブジェクトの名前、右側は付値(assignment)する内容である。Rで使用可能ないくつかの付値の書式を以下に示す。これらの異なる表記形式の付値機能は等価である。

```
>sales<-c(15,20,25,10,30)
>sales=c(15,20,25,10,30)
>assign("sales",c(15,20,25,10,30))
>c(15,20,25,10,30)-> sales
```

このような 1 行、あるいは 1 列のデータをベクトルと呼び、行の場合は行ベクトル、列の場合は列ベクトルと呼ぶ。

ベクトル `sales` は 5 つのデータを一つのデータセットとしてまとめている。この場合、ベクトル `sales` の長さは 5 である。R ではベクトルの長さを、関数 `length` を用いて求めることができる。

```
> length(sales)
[1] 5
```

連続する自然数を入力する際には、「始まる値:終わる値」の書式でデータを入力することも可能である。例えば、1 から 10 までの自然数を `y` に付値するには、`y<-1:10` あるいは `y<-c(1:10)` のように書く。

文字列の入力は、文字列を半角の `"` で囲む。文字列のデータは行、列の名前、ラベルとしてよく用いられる。例えば、表 1 の果物の種類のベクトルは次のように作成することができる。

```
> fruits<-c("Cherry","Apple","Grape","Banana","Other")
```

文字列ベクトルと数値ベクトルの長さが同じである場合、文字ベクトルを数値ベクトルの要素の名前として付け加えることができる。例えば、入力した数値ベクトル `sales` と文字ベクトル `fruits` の長さは同じである。`fruits` を `sales` のラベルにしたいときには、関数 `names` を次のように用いる。

```
> names(sales)<-fruits
> sales
Cherry Apple Grape Banana Other
  15   20   25   10   30
```

(2) 行列

長さが同じである複数のベクトルを一つのデータセットとしてまとめた n 行 m 列のデータセットを行列(`matrix`、マトリックス)と呼ぶ。

行列の作成と入力の手順は、まず行列のサイズを定義し、次に行列の要素を行別、あるいは列別にベクトルの場合と同じように入力する。例えば、表 2 のようなデータがあるとする。

表 2 果物の売上の割合(単位は%)

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

表 2 のデータは 2 行 5 列であるので、まず 2 行 5 列の行列を定義する。

```
>sales2<-matrix(0,2,5)
```

`sales2` はオブジェクトの名前、`matrix` は行列を作成する関数で、引数 `0` は行列の要素がすべてゼロで、`2` は行の数、`5` は列の数である。つまり、`sales2` は各要素がゼロである 2 行 5 列の行列である。

`sales2` へのデータの 입력は行単位、あるいは列単位でできる。第 1 行にデータ 15,20,25,10,30 を入力したいときには、ベクトルの場合と同じく次のように入力する。

```
>sales2[1,]<-c(15,20,25,10,30)
```

ここの `[1,]` は行列の第 1 行を指す。よって第 2 行の場合は `[2,]` となる。第 1 列の場合は `[,1]`、第 2 列の場合は `[,2]` で指定する。第 1 行と同じく第 2 行にデータを入力する。

```
>sales2[2,]<-c(10,25,20,25,20)
```

```
> sales2
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]  15   20   25   10   30
[2,]  10   25   20   25   20
```

行列が何行何列により構成されているかというサイズについては関数 `dim` を用いて求めることが可能である。

```
> dim(sales2)
```

```
[1] 2 5
```

行列の中のデータは行番号と列の番号(添字)で参照することができる。例えば、`sales2` の第 2 行、3 列の要素は、`sales2[2,3]` で呼び出すことが可能である。

```
> sales2[2,3]
```

```
[1] 20
```

第 2 行、3 列の要素 `20` を `30` に置き換えるのには次のように行う。

```
> sales2[2,3]<-30
```

関数 `rownames`, `colnames` を用いて行、列に名前を加えることができる。例として、入力した表 2 のデータ `sales2` に行、列の名前を付けることにする。果物の種類に関する文字列のベクトルはすでに `fruits` として作成しているので、それを用いる。

```
> colnames(sales2)<-fruits
```

```
> rownames(sales2)<-c("A","B")
```

```
> sales2
```

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

(3)配列

表 2 では売上の割合を示しているが、果物屋では、売上、仕入れの価額、売り出しの価額、種類別の利益のように異なる視点からデータを集計することも考えられる。

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

図 1 配列の構造のイメージ

このような複数のデータ表を一つのオブジェクトとしてまとめたものを配列(分割表)という。配列は、行列の拡張である。配列は関数 `array` を用いて作成する。

```
> array(1:30,c(2,5,3))
,, 1
  [,1] [,2] [,3] [,4] [,5]
[1,]  1   3   5   7   9
[2,]  2   4   6   8  10
,, 2
  [,1] [,2] [,3] [,4] [,5]
[1,] 11  13  15  17  19
[2,] 12  14  16  18  20
,, 3
  [,1] [,2] [,3] [,4] [,5]
[1,] 21  23  25  27  29
[2,] 22  24  26  28  30
```

(4)リスト

リストは、ベクトル、行列、配列、リストなど異なる型を一つのオブジェクトとして扱うことが可能なオブジェクトである。リストは関数 `list` を用いて作成する。

```
> L1<-list(c(1:8),c("A","B"),matrix(1:12,2,6))
> L1[1]
[[1]]
[1] 1 2 3 4 5 6 7 8

> L1[2]
[[1]]
[1] "A" "B"

> L1[3]
[[1]]
  [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  1   3   5   7   9  11
[2,]  2   4   6   8  10  12
```

2. データをファイルから読み込む

データファイルにはさまざまな形式があるが最も広く使用されているのはテキスト形式である。テキスト形式ファイルとは、保存されたファイルの拡張子が txt になっているファイルである。R の外部のテキストファイルを R に読み込むのには、関数 `read.table`、`scan` がよく使用されている。

(1) `read.table` による読み込み

関数 `read.table` は小、中規模のデータ表を R に読み込むのに適している。例えば、図 2 のように個々のデータをスペース、あるいは Tab キーで区切り、C ドライブのフォルダ RW に `data1.txt` という名前で保存したとする。



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

図 2 テキストデータファイルの作成画面

次のコマンドを実行すると、C ドライブのフォルダ RW の中に保存している `data1.txt` が `data1` という名前で R に読み込まれる。

```
>data1<-read.table("C:/RW/data1.txt",header=T,row.names=1)
```

引数 `header=T`(あるいは `header=TRUE`)は、データの第 1 行が列の名前であることを示し、`row.names=1` は第 1 列が行の名前であることを示す。行、あるいは列の名前がない場合は、対応する引数を省略する。

テキストファイルの中のデータの要素がカンマで区切られ、`data.txt` として保存された場合、`read.table` で読み込むのには引数「`sep=','`」を次のように追加する。

```
>read.table("C:/RW/data.txt",sep=',')
```

関数 `read.table` を用いて読み込んだデータを、データフレーム(data frame)と言う。データフレーム型のデータ構造は行列に似ているが、数値、文字、論理値が混在しているデータを扱うことができるのが大きな特徴である。データフレームはオブジェクトの一つの属性である。

R には、表計算ソフト Excel で作成した CSV 形式のファイルはを読み込む関数 `read.csv` が用意されている。

関数 `read.table` はデータを読み込むのには非常に便利であるが、大量のデータの場合に

は以下で説明する関数 `scan` と比較すると効率がよくない。

(2)関数 `scan` による読み込み

関数 `scan` は、大規模のデータセットを R に読み込むのに適している。

例えば、データが図 3 のように数値のみにより構成され、C ドライブのフォルダ RW に `data2.txt` として保存されたとする。関数 `scan` の使用例を次に示す。

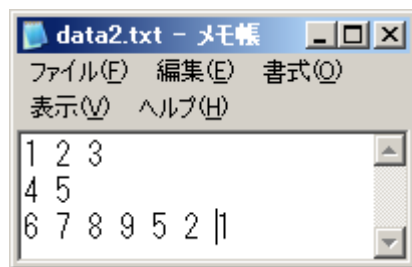


図 3 data2 のテキストファイル

```
> scan("C:/RW/data2.txt")
Read 12 items
[1] 1 2 3 4 5 6 7 8 9 5 2 1
```

もし、テキストファイルのデータがカンマで区切られた場合は、`read.table` を用いるときと同じく、関数 `scan` の中に引数「`sep=","`」を加える。関数 `scan` を用いるとデータが 1 行に読み込まれる。よって、関数 `scan` で読み込まれたデータを解析するためには、必要な形式への変換が必要である。

例えば、読み込んだ 12 の数値データを 3 行 4 列の行列に変更したいときには、関数 `matrix` と `scan` を次のように組み合わせればよい。

```
>matrix(scan("C:/RW/data2.txt"),3,4)
Read 12 items
      [,1] [,2] [,3] [,4]
[1,]    1    4    7    5
[2,]    2    5    8    2
[3,]    3    6    9    1
```

`data2.txt` では、データが横に 1、2、3 の順になっているが、作成した行列はデータが縦に 1、2、3 の順になっている。

関数 `matrix` に引数 `byrow=T` (あるいは `byrow=TRUE`) を加えるとデータが横の順に並べられた行列になる。

```
>data2<- scan("C:/RW/data2.txt")
>data2.M<-matrix(data2,3,4,byrow=T)
>data2.M
      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
```

```
[2,] 5 6 7 8
[3,] 9 5 2 1
```

データに数値、文字が混在している場合、関数 `scan` を用いて読み込むのには、列ごとにデータの属性を明記する必要がある。例えば、下記のようなデータ `data3.txt` があるとする。

```
A 1 5
B 2 4
C 3 3
```

このデータを、関数 `scan` を用いて読み込むときには、`list` を用いて列のデータの属性を明記する。文字の場合は`"`、数値の場合は `0` で表記する。

```
> data3 <- scan("C:/RW/data3.txt", list(label="", x=0, y=0))
```

このように読み込まれたデータはリストである。R 内部の行列、データフレーム、リストはお互いに変換することができる。行列、リストをデータフレームに変換するには関数 `data.frame` を用いる。

```
> data.frame(data3)
```

```
  label x y
1     A 1 5
2     B 2 4
3     C 3 3
```

```
> data2.F <- data.frame(data2.M)
```

```
> data2.F
```

```
 X1 X2 X3 X4
1  1  2  3  4
2  5  6  7  8
3  9  5  2  1
```

データフレームを行列へ変換するには関数 `as.matrix`、あるいは `data.matrix` を用いる。

```
> as.matrix(data2.F)
```

```
 X1 X2 X3 X4
1  1  2  3  4
2  5  6  7  8
3  9  5  2  1
```

画面上に返されたデータの構造だけでは、データフレームであるか、行列であるか初心者には見分けが付かない場合もある。

関数 `class` を用いるとオブジェクトの属性を確認することができる。

```
> class(data3)
```

```
[1] "list"
```

```
> class(data2.M)
```

```
[1] "matrix"
```

```
> class(data2.F)
[1] "data.frame"
> class(fruits)
[1] "character"
> class(sales)
[1] "numeric"
> class(c(TRUE,FALSE))
[1] "logical"
```

3. データの出力

計算結果が少量であれば、キーボードで打ち移すか、コンソールに出力された結果をコピーし、レポートや論文の文章の中に貼り付ければよいが、出力の結果が大量でかつ他のツールで利用したいときには、ファイルとして出力しておくとう便利である。

R のデータをテキストファイルとして R の外に出力する関数は **write**、**write.table**、**sink** などがある。

例えば、R に下記のようなデータが **rdata** というオブジェクトの名前で作成されたとする。

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

コマンド

```
>write(rdata,"C:/RW/OutData.txt")
```

により **rdata** が **OutData.txt** という名前で C ドライブのフォルダ **RW** の中に出力される。

OutData.txt と **rdata** は、データの並びの方向が異なる。出力されたデータと R の中のデータの並び方向を一致させるためには **t** 関数を用いて R の中のデータを転置(回転)した後出力する。転置の関数は **t** である。

```
>write(t(rdata),"C:/RW/ OutData.txt ")
```

これで R の内部のデータと R の外部に出力されたデータの並び方が同じになる。また、出力する際、データの列の数を指定することも可能である。例えば、次のように書くとデータが 2 列で出力される。

```
>write(t(rdata),"C:/RW/ OutData.txt",columns=2)
```

また出力されたデータファイル **OutData.txt** に他のデータを後で追加したいときには、下記のように引数 **append=TRUE** を加えておく必要がある。

```
>write(rdata2,"C:/RW/OutData.txt", append=TRUE)
```

関数 **write.table** はデータフレームを出力する関数である。データフレームを **write.table** で出力すると、文字列は””で囲まれ、数値データはそのまま出力される。行列を **write.table** を用いて出力するとすべてのデータが文字列として扱われる。関数 **write.table** の書式は **write** の書式と同じである。

関数 **sink** はコンソールに返される内容をファイルとして出力する関数である。

```
> sink("C:/RW/temp.txt")
> data2.M
> data2.F
> sink()
```

この 3 行のコマンドにより、**data2.M**、**data2.F** が C ドライブのフォルダ **RW** の中に **temp.txt** というファイル名で出力される。

関数 **sink** の書式は、出力が開始するコマンド **sink("****")** と出力を終了するコマンド **sink()** がペアとなっている。関数 **sink** でも **write** と同じく、引数 **append=TRUE** を用いると、出力したファイルに、後でデータを追加することができる。

また、パッケージ **foreign** を用いると、**SAS**、**SPSS**、**Stata**、**S-PLUS** のファイル形式からのデータを読み込むことが可能である。

4. パッケージ

R における関数は R の本体に組み込まれた基礎部分 (**base**) とパッケージ (**package**) に分けられる。パッケージには最新の研究結果が一般の市販のソフトより早く入手できるものもあり、現段階では 200 を超えるパッケージが利用可能である。

R でパッケージを利用するためには、次の 2 段階の作業が必要である。

- (1) パッケージをインストールする
- (2) パッケージを読み込む

R をインストールする際、頻繁に使用されると思われる 20 数種類のパッケージが自動的にインストールされる。後述の図 4 の **Select one** ダイアログボックスのリストからインストールされているパッケージを確認することができる。リストアップされていないパッケージをインストールする方法は 2 とおりある。

- (1) インターネットが接続されている環境で、メニューの **Packages Install Package(s) from CRAN** (あるいは **Install Package(s) from Bioconductor**) をクリックし、**Select** ダイアログボックスから必要となるパッケージをクリックするとダウンロード・インストールが始まる。インストールが終了するとコンソール画面にダウンロードしたファイルを削除するかどうかを尋ねる「**Delete downloaded files (y/N)?**」というメッセージが返される。いずれかを選択してもよい。ここでは推奨されている **"N"** を入力

