

# 第3回 データフレームの基本操作 その1

平成 29 年 5 月 16 日

## 目次

1	ここで学ぶ事	1
2	データフレーム (data.frame) とは?	1
2.1	チェックリスト	2
3	データフレームの作成	3
3.1	列の追加	3
3.2	行の追加	4
3.3	チェックリスト	5
4	データフレームの属性	5
4.1	チェックリスト	6
5	実習	6
6	番外編: ちょっと便利な文字列生成関数	6
7	宿題	8

## 1 ここで学ぶ事

- データフレームと呼ばれるデータ構造体の作成方法

## 2 データフレーム (data.frame) とは?

表計算ソフトのテーブル(表)のようなデータ構造で、行(row)と列(column)を持つ。一般的にテーブル形式で表されるデータでは、行が各データを表し(これをレコードと呼ぶ)、列が各レコードの属性データを表す。表1はテーブル形式の例である。テーブル形式では列に同じタイプのデータが並ぶ。Rのデータ型(mode)に当てはめるなら、「身長, 体重, 年齢」は Numeric 型, 「氏名, 性別, 出身地」は Character 型になる。

氏名	身長	体重	年齢	性別	出身地
山田太郎	173.5	66	19	男	福岡県
西南花子	166.4	58	20	女	鹿児島県
市東治子	168	NA	18	女	千葉県
三宅信二	170.3	81	20	男	岡山県

表 1: 個人データ

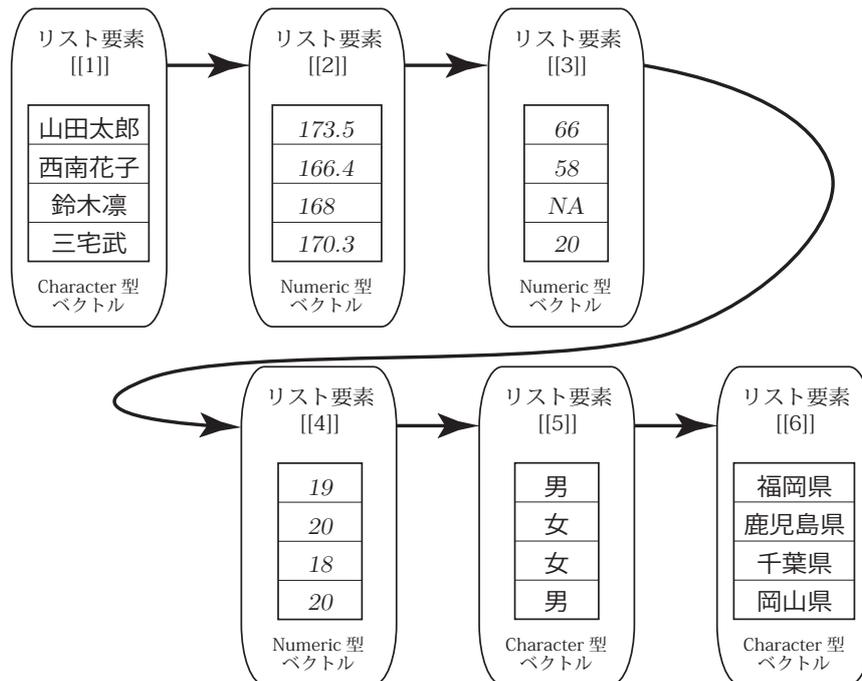


図 1: データフレームの構造

R の `data.frame` は表 1 のような列に同じデータ型が並んだテーブル構造を保持するためのデータ構造だ。 `data.frame` の実体は、各列をベクトルに格納し連結させたリスト構造<sup>1</sup>である。ベクトルがリスト状に連結されている様子を表したものが図 1 である<sup>2</sup>。

市東治子の体重が「NA」となっているが、これはデータが Not Available、すなわち欠損値 (missing values) を表す。何らかの理由でデータが存在しない場合、R では NA という logical 型の値で欠損値を表す。NA だけは他の型のベクトルに混入できる。

## 2.1 チェックリスト

- テーブル
- レコード
- row , column

<sup>1</sup>リストとはデータを連結させたデータ構造で、ここではベクトルを連結させて保持している。

<sup>2</sup>図 1 はデータが連結されてリストを構成することを示す概念図で、コンピュータ・メモリ上の構成ではない。通常の R プログラミングではメモリ上での構成まで理解する必要はない。

- リスト
- 欠損値, NA, missing values

### 3 データフレームの作成

データフレームは列毎にベクトルを連結させたものなので、まずベクトルを作成し、列データを `data.frame()` 関数に渡してやるとデータフレームが生成される。

```
> name <- c("山田太郎", "西南花子", "市東治子", "三宅信二") # 名前データをベクトルで用意
> height <- c(173.5, 166.4, 168, 170.3) # 身長データをベクトルで用意
> weight <- c(66, 58, NA, 81) # 体重データ. NA は欠損値
> age <- c(19, 20, 18, 20) # 年齢データ
> personal <- data.frame(name, height, weight, age) # 列データからデータフレームを作成
> personal
  name height weight age
1 山田太郎 173.5    66 19
2 西南花子 166.4    58 20
3 市東治子 168.0    NA 18
4 三宅信二 170.3    81 20
```

作成されたデータフレームには自動的に行番号が付され、列名は変数名になる。

#### 3.1 列の追加

作成したデータフレームに列を追加したい場合、`cbind()` 関数 (column bind) を使う。下の例では「性別」列を `personal` に追加し、その結果を再び `personal` に代入している。

```
> 性別=c("男", "女", "女", "男") # 追加する列をベクトルで作成
> personal <- cbind(personal, 性別) # 変数名を日本語にすると列名も日本語になる
> personal
  name height weight age 性別
1 山田太郎 173.5    66 19  男
2 西南花子 166.4    58 20  女
3 市東治子 168.0    NA 18  女
4 三宅信二 170.3    81 20  男
```

下のように引数に直接データを渡すこともできる。引数データに名前を付けて渡すことによって列名も引数名になる。関数に渡す引数に名前を付けたものを「名前付き引数」と呼ぶ。

```
> personal <- cbind(personal, 出身地=c("福岡県", "鹿児島県", "千葉県", "岡山県"))
> personal
  name height weight age 性別 出身地
1 山田太郎 173.5    66 19  男  福岡県
2 西南花子 166.4    58 20  女  鹿児島県
3 市東治子 168.0    NA 18  女  千葉県
4 三宅信二 170.3    81 20  男  岡山県
```

### 3.2 行の追加

データフレームに行を追加するには `rbind()` 関数 (row bind) を使う。まず、新たに追加する2行からなるデータフレームを作成する。

```
> extra <- data.frame(名前=c("山村英二", "平野康介"),
+                     身長=c(169, 159),
+                     体重=c(61, 63),
+                     年齢=c(23, 20),
+                     性別=c("男", "男"),
+                     出身地=c("福岡県", "神奈川県"))
> extra
  名前 身長 体重 年齢 性別 出身地
1 山村英二 169  61  23  男  福岡県
2 平野康介 159  63  20  男  神奈川県
```

`rbind()` 関数を以下のように実行するとエラーが出る。

```
> rbind(personal, extra) # 「名前が以前の名前と一致しません」というエラーが出る
```

データフレーム同士を結合する場合、列名をそろえる必要がある。`personal` の列名を日本語に変更する。`colnames()` 関数は引数で与えられたデータ構造の列名を返す。この返した値に、新たに設定したい列名をベクトルで与えると列名が更新される。

```
> colnames(personal) # 現在の列名 (column names)
[1] "name" "height" "weight" "age" "性別" "出身地"
> colnames(personal) <- c("名前", "身長", "体重", "年齢", "性別", "出身地")
> personal
  名前 身長 体重 年齢 性別 出身地
1 山田太郎 173.5  66  19  男  福岡県
2 西南花子 166.4  58  20  女  鹿児島県
3 市東治子 168.0  NA  18  女  千葉県
4 三宅信二 170.3  81  20  男  岡山県
```

`extra` の列名を `personal` と同じにしたいので、次のようにしても同じ。

```
> colnames(personal) <- colnames(extra) # extra の列名を personal の列名に設定
> personal
  名前 身長 体重 年齢 性別 出身地
1 山田太郎 173.5  66  19  男  福岡県
2 西南花子 166.4  58  20  女  鹿児島県
3 市東治子 168.0  NA  18  女  千葉県
4 三宅信二 170.3  81  20  男  岡山県
```

これで列名が同じになったので、`rbind()` で行を追加しよう。追加した結果は再び `personal` 変数に代入する。

```
> personal <- rbind(personal, extra)
> personal
  名前 身長 体重 年齢 性別 出身地
1 山田太郎 173.5 66 19 男 福岡県
2 西南花子 166.4 58 20 女 鹿児島県
3 市東治子 168.0 NA 18 女 千葉県
4 三宅信二 170.3 81 20 男 岡山県
5 山村英二 169.0 61 23 男 福岡県
6 平野康介 159.0 63 20 男 神奈川県
```

### 3.3 チェックリスト

- データフレームを生成する関数は？
- データフレームに列を追加する関数は？
- 名前付き引数
- データフレームに行を追加する関数は？
- `colnames()` 関数

## 4 データフレームの属性

各データフレームを特徴付ける列名, 行名, 列数, 行数等を一般に「属性 (attributes)」と呼ぶ<sup>3</sup>. このうち, 各データフレームがデータとして保持している属性は `attributes()` 関数で調べることができる.

```
> attributes(personal)
$names
[1] "名前" "身長" "体重" "年齢" "性別" "出身地"

$row.names
[1] 1 2 3 4 5 6

$class
[1] "data.frame"
```

上の結果は `personal` データフレームは `names` 属性, `row.names` 属性, `class` 属性の3つの属性を持つことが分かる. `names` 属性は列名を保持し, `row.names` 属性は行名を保持している. `class` 属性はオブジェクト指向プログラミングを学ぶまで必要ない. 前節で学んだ `colnames()` 関数は `names` 属性を読み書きする関数である. `row.names` 属性は `rownames()` 関数を用いて読み書きでき, これで行名を指定できる.

<sup>3</sup>オブジェクト指向プログラミングで用いられる「属性」はもっと狭い意味で用いられる. これについてはオブジェクト指向プログラミングの学習時に説明する.

以下，データフレームの様々な属性を取得してみよう．

```
> rownames(personal) # 行名の取得
[1] "1" "2" "3" "4" "5" "6"
> dim(personal)      # 次元 (dimension) の取得 . 行数 , 列数の順 .
[1] 6 6
> dim(personal)[1]   # 行数
[1] 6
> dim(personal)[2]   # 列数
[1] 6
```

## 4.1 チェックリスト

- 属性
- `attributes()` 関数
- `rownames()` 関数
- データフレームの行数と列数の取得方法 .

## 5 実習

`personal` データフレームに行名を設定してみよう . 1 行目から `ID1` , `ID2` , ... , と `ID` を振って以下の行名を持つデータフレームを完成させよ .

```
> personal
  名前 身長 体重 年齢 性別 出身地
ID1 山田太郎 173.5 66 19 男 福岡県
ID2 西南花子 166.4 58 20 女 鹿児島県
ID3 市東治子 168.0 NA 18 女 千葉県
ID4 三宅信二 170.3 81 20 男 岡山県
ID5 山村英二 169.0 61 23 男 福岡県
ID6 平野康介 159.0 63 20 男 神奈川県
```

## 6 番外編: ちょっと便利な文字列生成関数

上の実習では , 行名に指定する `ID` を自分でタイプした . 大量のデータの場合 , こうした文字列を自動生成できたら便利である . ここでは , 文字列生成に便利な `paste()` 関数の使い方を学ぶ .

まずは `paste()` 関数のマニュアルを読んでみよう .

```
> help(paste)
```

「Description」の項には、ベクトルを文字列に変換して繋げる (concatenate) と書いてある。次の「Usage」には関数の使い方 (呼び出し方) が記載されている。

Usage:

```
paste (... , sep = " ", collapse = NULL)
paste0 (... , collapse = NULL)
```

`paste()` 関数の最初の引数は... となっているが、Usage の下にある「Arguments (引数)」の解説部分を読むと、ここには1から複数個のRオブジェクトを渡すべきであることがわかる。Rで使用する変数、関数、あらゆるデータはすべてRオブジェクトなので、要は何を渡しても良いということだ。それらは文字列に変換されて `paste()` によって結合される。

2番目の引数に「`sep = " "`」とあるが、これは以前学んだ「名前付き引数」で、文字列をこの名前の引数に渡すと、「...」部分で指定したベクトルを繋げる際、この文字列を間に挟む。つまり、ベクトルを連結する際、この文字列で `separate` するということだ。マニュアルにある「`sep=" "`」というのはデフォルトで `sep` 引数にはスペース (空白1文字) が指定されているということだ。

以下、例を見てみよう。

```
> paste("A", "B", "C", "D")           # sepはデフォルトでスペース1文字
[1] "A B C D"
> paste("A", "B", "C", "D", sep="--") # sepに"--"を指定
[1] "A--B--C--D"
> paste("A", "B", "C", "D", sep="")   # sepに空の文字列を指定
[1] "ABCD"
```

上の例は、4つの引数に文字列を1つずつ渡している。以前、Rではすべてのデータはベクトルになることを学んだが、上の例の各文字列は要素が1個のベクトルと同じである。つまり、「A」も「B」も要素が1個の文字列ベクトルである。`paste()` 関数は4つのベクトルを結合して

`paste()` 関数が威力を発揮するのは、引数に与えるベクトルの要素が複数個になった時である。

```
> letters <- c("A", "B", "C", "D")
> paste(letters, 1:length(letters), sep="")
[1] "A1" "B2" "C3" "D4"
```

上の例では、第1引数が `letters` で4つの要素からなるベクトル、第2引数が「`1:length(letters)`」でこれは1から4の数字からなるベクトルである。`paste()` は各引数に与えられたベクトルの要素同士を結合する。

この方法を使って先ほどの実習のIDを生成してみよう。

```
> paste("ID", 1:15, sep="")
[1] "ID1" "ID2" "ID3" "ID4" "ID5" "ID6" "ID7" "ID8" "ID9" "ID10" "ID11"
[12] "ID12" "ID13" "ID14" "ID15"
> paste("ID", 1:dim(personal)[1], sep="")
[1] "ID1" "ID2" "ID3" "ID4" "ID5" "ID6"
```

2017年の1月から12月までのデータの列名を作りたいときは、以下のようにすれば良い。

```
> paste(2017, "/", 1:12, sep="")
[1] "2017/1" "2017/2" "2017/3" "2017/4" "2017/5" "2017/6" "2017/7" "2017/8"
[9] "2017/9" "2017/10" "2017/11" "2017/12"
```

第1引数と第2引数は要素が1のベクトルだが、第3引数のベクトルが12個の要素からなるので、それに合わせて第1引数と第2引数の要素が繰り返し使われる。これは非常に便利で、わざわざ第3引数に合わせて、以下のように実行する必要はない。

```
> paste(rep(2017, 12), rep("/", 12), 1:12, sep="") # 冗長! 上のやり方でよい!
[1] "2017/1" "2017/2" "2017/3" "2017/4" "2017/5" "2017/6" "2017/7" "2017/8"
[9] "2017/9" "2017/10" "2017/11" "2017/12"
```

## 7 宿題

以下の文字列を `paste()` 関数を用いて生成せよ。

```
> results
[1] "2014年1月" "2014年2月" "2014年3月" "2014年4月" "2014年5月" "2014年6月"
[7] "2014年7月" "2014年8月" "2014年9月" "2014年10月" "2014年11月" "2014年12月"
[13] "2015年1月" "2015年2月" "2015年3月" "2015年4月" "2015年5月" "2015年6月"
[19] "2015年7月" "2015年8月" "2015年9月" "2015年10月" "2015年11月" "2015年12月"
[25] "2016年1月" "2016年2月" "2016年3月" "2016年4月" "2016年5月" "2016年6月"
[31] "2016年7月" "2016年8月" "2016年9月" "2016年10月" "2016年11月" "2016年12月"
```