クレジット:
UTokyo Online Education 統計データ解析 Ⅱ 2018 小池祐太

東京大学 THE UNIVERSITY OF TOKYO

CC BY NC ND

統計データ解析 II (平成３０年度)

東京大学大学院数理科学研究科
統計データ解析教育研究グループ

村田　昇 (早稲田大学, 東京大学)
吉田朋広 (東京大学)
小池祐太 (東京大学)

第 1 章 R の基本的な操作

1.1. はじめに

まず，はじめに R の概要について述べる．

1.1.1. R 言語. R(または R 言語と呼ばれる) は統計計算のための言語と環境の総称であり，オープンソース・フリーソフトウェア (open source, free software) である．利用の規約は GNU General Public License (GPL) に従うので，その内容について詳しく知りたい場合は

`https://www.gnu.org/licenses/gpl-3.0.en.html` (英語)
`https://www.gnu.org/licenses/gpl-3.0.ja.html` (日本語)

を参照して欲しい．

また，多くの人により開発されている多数のパッケージ (package) によって，様々な機能 (パッケージは関数やデータの集合体と考えればよい) を追加することができる．R の本体，およびパッケージは，開発プロジェクトのサイト R Project (The R Project for Statistical Computing)

`http://www.r-project.org/`

のメニューにある CRAN (The Comprehensive R Archive Network) の中にあるミラーサイト (mirror site; 日本国内にもある) からダウンロードすることができる．R の本体は OS (Operating System; Linux, MacOS, Windows) 別に異なる配布物として公開されており，それぞれの OS に適切な方法で簡単にインストールすることができる．また，パッケージは R の中に用意された関数や GUI (Graphical User Interface; 画面上のグラフィクスとマウスなどを用いて直感的な操作を提供するユーザインタフェース) を用いてインストールすることができる．

R Project で公開されている R 本体には Windows や MacOS の場合は専用の GUI が用意されているが，UNIX 系 OS の場合はターミナル (シェル) から起動する必要がある．このため OS によって若干操作性が異なるという問題があるが，UNIX も含め様々な OS において同様に利用することができる RStudio という統合開発環境 (integrated development environment; IDE) が RStudio 社により開発され公開されている．

`https://www.rstudio.com/`

講義では OS による操作の違いをできるだけ少なくするために，RStudio を用いて説明を行う．

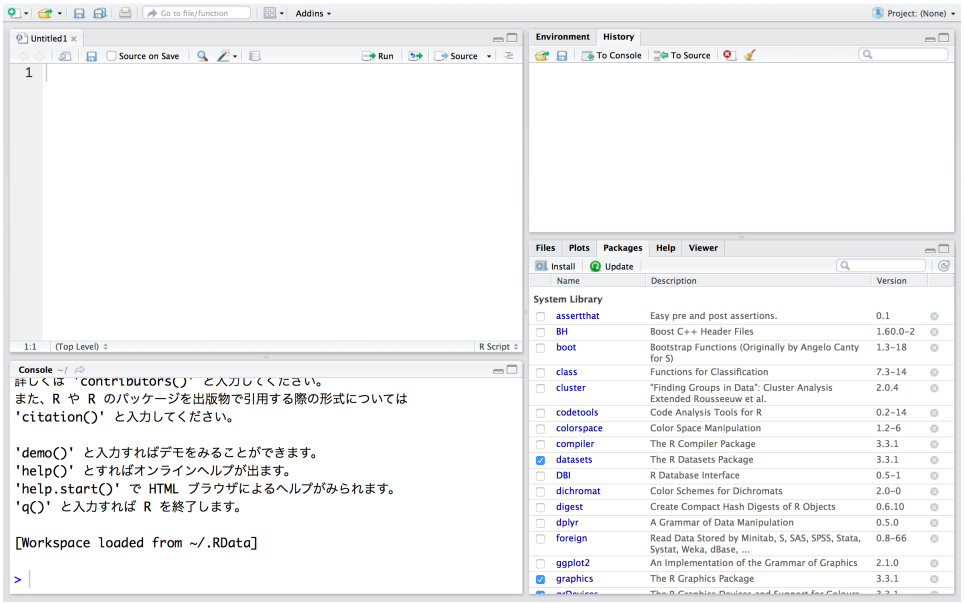
演習 1.1. R と RStudio を自身の PC にインストールしてみよう．

`http://www.r-project.org/` (The R Project)
`https://www.rstudio.com/` (RStudio, inc)

例えば以下のサイトがインストールの参考になる．

`http://www.okadajp.org/RWiki/?R%20` のインストール
`http://aoki2.si.gunma-u.ac.jp/R/begin.html`

1.1.2. 起動と終了. RStudio を起動すると，標準では以下のような 4 ペイン (枠; pane) のウィンドウが立ち上がる．左上がエディタ，左下がコンソール，右上が変数や履歴，右下がグラフィクスやヘルプなどを表示するペインとなる．



RStudio © RStudio, GNU AGPL v.3

FIGURE 1. RStudio の起動画面.

起動後，コンソールは以下のようなメッセージを表示し，最後に入力を促すプロンプトである ‘>’ 記号を表示して入力待ちの状態となる.

```
R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.5.0 (64-bit)

R は、自由なソフトウェアであり、「完全に無保証」です。
一定の条件に従えば、自由にこれを再配布することができます。
配布条件の詳細に関しては、'license()' あるいは 'licence()' と入力してください。

R は多くの貢献者による共同プロジェクトです。
詳しくは 'contributors()' と入力してください。
また、R や R のパッケージを出版物で引用する際の形式については
'citation()' と入力してください。

'demo()' と入力すればデモをみることができます。
'help()' とすればオンラインヘルプが出ます。
'help.start()' で HTML ブラウザによるヘルプがみられます。
'q()' と入力すれば R を終了します。

>
```

(start.r)

例えばここで終了を指示する q() を入力すれば，R は終了する.

終了時にメッセージが表示される場合があるが，これに対して “y(yes)” を入力すると，それまでに定義された変数や関数およびコマンドの履歴 (ヒストリ) が保存され，次回起動時に自動的に読み込まれる. “n(no)” にするとこのセッションで更新した内容は残らない. また，終了することを中止して計算を続ける場合は “c(cancel)” を入力する.

なお，入力文字列において ‘#’ 以降は無視されるので，以降の実行例においては#を用いて必要なコメントを記載していく.

1.1.3. ヘルプ機能. R にはオンラインのヘルプ機能が備えられていて，コンソール (左下のペイン) から関数 `help()` に関数名を，関数 `help.search()` には検索したいキーワードを渡すことによって利用することができる．なお，以下はあくまで一つの出力例であり，環境 (R のバージョンやインストールされているパッケージなど) によって出力が異なる場合があることに注意して欲しい．

```
> help(sin) # 三角関数のヘルプを見る
> ?log # help() の代わりに ? を使うことができる
> help.search("histogram") # ヒストグラムに関連する関数を探す
> ??random # help.search() の代わりに ?? を使うことができる
```

(help.r)

最初の例は sin 関数を調べたもので, 左上の “Trig” は見出しで, この内容が “trigonometric functions” に関するヘルプであることを表わしている．また中央上の “package:base” は “base” というパッケージ内の関数であることを示している．

二番目の例は “histogram” に関連する事項を検索したもので, 例えば “graphics::hist” は “graphics” というパッケージ内にある “hist” という関数がヒストグラムの作成に関連することを示している．

なお，上記の例で出てきた “base” や “graphics” は指定しなくても標準で読み込まれるパッケージである．読み込まれているパッケージを確認する方法は次節を参照して欲しい．

GUI を用いる場合は右下のペインの “Help” タブ (tab) を利用する．関数名またはキーワードを入力して必要な情報を検索することができる．

R の本体，あるいはパッケージに関するドキュメント (マニュアル) は開発プロジェクトのサイト CRAN にあるが，使い方を含め有用な情報を解説するサイトとして

`http://www.okada.jp.org/RWiki/`
`http://aoki2.si.gunma-u.ac.jp/R/`

など数多くあるので，これらも合わせて参照して欲しい．

1.1.4. パッケージ管理. CRAN では現在 9000 を越えるパッケージが公開されている．

右下のペインには “Packages” タブがあり，GUI を用いてパッケージ管理を行うことができる．必要な機能を持つパッケージ名を調べれておけば，“Packages” タブの中の “Install” から新規にパッケージをインストールすることができる．また “Update” を選ぶとインストール済のパッケージの更新を行うことができる．なお，標準でいくつかのパッケージは自動的に読み込まれており，既に読み込まれたパッケージは “Packages” タブで確認することができる．

コンソールから直接インストールする場合には，関数 `install.packages()` を用いればよい．以下は一つの出力例であり，環境によっては異なる場合もあることに注意して欲しい．

```
> install.packages("ggplot2")
Installing package into ‘/usr/local/lib/R/3.3/site-library’
(as ‘lib’ is unspecified)
 URL 'https://cran.ism.ac.jp/src/contrib/ggplot2_2.1.0.tar.gz' を試しています
Content type 'application/x-gzip' length 1571788 bytes (1.5 MB)
==================================================
downloaded 1.5 MB

* installing *source* package ‘ggplot2’ ...
**  パッケージ ‘ggplot2’ の解凍および MD5 サムの検証に成功しました
** R
** data
*** moving datasets to lazyload DB
```

```
** inst
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** installing vignettes
** testing if installed package can be loaded
* DONE (ggplot2)

### 以下省略
```

(install.r)

パッケージを取り扱う関数についての更に詳しい情報は help("install.packages") や help("update.packages"), または help("INSTALL") などを利用して調べて欲しい.

1.2. 基本的な使い方

1.2.1. 式の入力. 四則演算や一般的な関数は C 言語などとほぼ同じ名称で使うことができ, ほぼ直感に沿った文法で計算を実行することができる.

```
> (1 + 3) * (2 + 4) / 6 # 四則演算
[1] 4
> 1.8 + 5 - 0.04 + 8.2 / 3 # 計算順に注意
[1] 9.493333
> pi # π (パイ) は定義されている
[1] 3.141593
> print(pi,digits=22) # 桁数を変更して表示
[1] 3.1415926535897931159979
> sqrt(2) # 平方根
[1] 1.414214
> 8^(1/3) # 冪乗
[1] 2
> exp(10) # 指数関数
[1] 22026.47
> exp(1) # 自然対数の底
[1] 2.718282
> log(10) # 対数関数 (log, log10, log2)
[1] 2.302585
> sin(pi/2) # 三角関数 (sin, cos, tan)
[1] 1
> sinpi(2/3) # sinpi(x) = sin(pi*x)
[1] 0.8660254
> acos(1/2) # 逆三角関数 (asin, acos, atan)
[1] 1.047198
```

(calc.r)

1.2.2. 数の扱い. R では実数および複素数を取り扱うことができ, 指数表記にも対応している. また, 無限大や不定な数など特殊なものを扱うこともできる.

```
> (1.5+3.5i) * (2-4i) # 複素数の計算 (i の前に数字がある場合のみ虚数とみなす)
[1] 17+1i
> 1i * 1i # 虚数単位は i ではなく 1i
```

```
[1] -1+0i
> 1.38e10 * 3.68e-37 / 0.34e-5 # 指数表記の計算
[1] 1.493647e-21
> -log(0) # 無限大（非常に大きな値）
[1] Inf
> 3 * log(0) # 数として扱える（計算はできる）
[1] -Inf
> sqrt(-1) # Not a Number（非数）
[1] NaN
> sqrt(-1) + 1 # 数として扱えない（計算はできない）
[1] NaN
> log(0) / log(0) # これも Not a Number（非数）
[1] NaN
```

(numbers.r)

なお，これらの数値は C 言語にあるような int や double などの数値データの型を気にする必要はない．

1.2.3. 変数への代入. 文字列を変数名として，数値を保持することができる．また，変数をそのまま計算に用いることもできる．

```
> x <- sin(pi/3) # x に代入
> print(x) # x の値を確認
[1] 0.8660254
> y <- cos(pi/3) # y に代入
> y # print(y) と同じ，y の値を確認
[1] 0.5
> z <- x - y # 計算結果を代入
> (z) # print(z) と同じ，z の値を確認
[1] 0.3660254
> (w <- x * y) # print(w <- x * y) と同じ，代入結果を表示
[1] 0.4330127
> w # 代入結果を確認（上と同じ値が表示される）
[1] 0.4330127
```

(variables.r)

変数名は自由に決めて用いることができる (例:x, y, abc など) が，sin，log，pi など R の仕様として使われているものは，用いることができない訳ではないが混乱を招く元なので使わない方が良い．

なお, R では, 変数や関数, および関数の実行結果等を総称して「オブジェクト」と呼ぶ

1.2.4. 乱数の生成. 代表的な確率分布に従う乱数の生成を行うことができる. ここでは一様乱数，正規乱数およびランダムサンプリングを行う関数 runif()，rnorm()，sample() を紹介する．

```
> ### 関数 runif の使い方
> runif(5,min=-1,max=1) # [-1,1] 上の一様乱数を 5 個生成
[1] -0.5918908  0.8065636 -0.6365265  0.3623380 -0.4739526
> runif(5) # 最大最小について指示がなければ [0,1] 上の一様分布
[1] 0.3796288 0.5554349 0.6294269 0.9439813 0.8978049
```

```
> ### 関数 rnorm の使い方
> rnorm(5,mean=3,sd=2) # 平均 3，標準偏差 2 の正規乱数を 5 個生成
[1] 4.1475048 0.8727831 1.8968064 5.0898932 6.2644172
> rnorm(5) # 平均と標準偏差について指示がなければ標準正規分布
[1] -0.08079564  1.08373064 -0.90697933  0.50875084  0.57380880
> ### 関数 sample の使い方
> sample(1:10,size=5) # 1 から 10 の整数からランダムに 5 つ抽出
[1] 8 1 9 6 3
> sample(1:10,10) # 1 から 10 の整数をランダムに並べ替え，"size=" は省略可
 [1]  4  6  9  7  8  5  2 10  1  3
> sample(1:10,10,replace=TRUE) # 1 から 10 の整数から 10 個復元抽出
 [1] 10  4  9  5  3  4  5  5  6  4
```

(random.r)

これらの関数は数値シミュレーションを行う場合に重要な役割を果たす．

演習 1.2. R を電卓として使ってみよう．

(1) 四則演算の計算順を確認する．
(2) 複素数の扱い方を確認する．
(3) 数学で用いらるどういった関数が R で利用可能か確認する．
(4) 計算機で生成される乱数の性質を確認する．(ヒント: `help("Random")`, `help("hist")`)

1.3. データ構造

R には，以下のようなデータ構造が用意されている．

- ベクトル (vector)
- 行列 (matrix)
- 配列 (array)
- リスト (list)
- データフレーム (data frame)

また，これらのデータは適当な変数を割り当てて保存しておくことができる．

以下ではデータ解析において基本的な役割を担うベクトル，行列, リスト, データフレームについて説明する．

1.3.1. ベクトル. ベクトルはスカラー値の集合 (1 次元配列) である．

スカラー値として扱われるものには，実数と複素数以外に，文字列，論理値などが含まれる．

```
> ### 実数
> (x <- 4) # 変数 x に実数 4 を代入
[1] 4
> x^10
[1] 1048576
> x^100
[1] 1.606938e+60
> x^1000 # 実数として保持できる最大値を越える
[1] Inf
> ### 複素数
> 1i # "i"の直前に数値を書く
[1] 0+1i
> (1+2i)*(2+1i)
[1] 0+5i
```

```
> ## i # i だけでは複素数とみなされずエラーになる
> ### 文字列
> (y <- "foo") # 文字列は ' または " で括る
[1] "foo"
> (z <- "bar") # ("foo" や "bar" は意味のない文字列として良く用いられる)
[1] "bar"
> paste(y,z) # 文字列の足し算，省略時は区切り文字 (separator) は " "(空白)
[1] "foo bar"
> paste(y,z,sep="") # 区切り文字 (separator) を ""(無) に指定
[1] "foobar"
> ## y+z # 足し算はできずエラーになる
> ### 論理値
> TRUE # 論理値 (真)
[1] TRUE
> T # 論理値 (真) の省略形
[1] TRUE
> FALSE # 論理値 (偽)
[1] FALSE
> F # 論理値 (偽) の省略形
[1] FALSE
> as.numeric(TRUE) # as.numeric は数値に変換する関数
[1] 1
> as.numeric(F)
[1] 0
```

(scalar.r)

一般にベクトルは関数 c() を用いて生成することができる．ベクトルの要素を取り出すには [] を付けて要素の番号を指定すればよい．

これ以外に規則的な系列を生成するための関数として，等間隔の系列を作るために関数 seq()，繰り返しの系列を作るために関数 rep() などがある．また，ベクトルの長さを求めるために関数 length() が用意されている．

```
> ### 関数 c の使い方
> (x <- c(0,1,2,3,4)) # スカラーを並べてベクトルを作成
[1] 0 1 2 3 4
> (y <- c("foo","bar")) # 文字列のベクトル
[1] "foo" "bar"
> x[2] # ベクトルの 2 番目の要素
[1] 1
> y[2]
[1] "bar"
> x[c(1,3,5)] # 複数の要素は要素番号のベクトルで指定
[1] 0 2 4
> ### 関数 seq の使い方
> (x <- seq(0,3,by=0.5)) # 0 から 3 まで 0.5 刻みの系列
[1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0
> (y <- seq(0,3,length=5)) # 0 から 3 まで長さ 5 の系列
[1] 0.00 0.75 1.50 2.25 3.00
> (z <- 1:10) # seq(1,10,by=1) と同様
 [1]  1  2  3  4  5  6  7  8  9 10
> (z <- 10:1) # 10 から 1 まで 1 刻みの逆順
```

```
 [1] 10  9  8  7  6  5  4  3  2  1
> z[3:8] # z の 3 番目から 8 番目の要素
[1] 8 7 6 5 4 3
> ### 関数 rep の使い方
> (x <- rep(1,7)) # 1 を 7 回繰り返す
[1] 1 1 1 1 1 1 1
> (y <- rep(c(1,2,3),times=3)) # (1,2,3) を 3 回繰り返す
[1] 1 2 3 1 2 3 1 2 3
> (z <- rep(c(1,2,3),each=3)) # (1,2,3) をそれぞれ 3 回繰り返す
[1] 1 1 1 2 2 2 3 3 3
> ### その他の操作
> (x <- seq(0,2,by=0.3))
[1] 0.0 0.3 0.6 0.9 1.2 1.5 1.8
> length(x) # ベクトルの長さ
[1] 7
> y <- 2:5
> (z <- c(x,y)) # ベクトルの連結
 [1] 0.0 0.3 0.6 0.9 1.2 1.5 1.8 2.0 3.0 4.0 5.0
> rev(z) # rev はベクトルを反転する関数
 [1] 5.0 4.0 3.0 2.0 1.8 1.5 1.2 0.9 0.6 0.3 0.0
> LETTERS # アルファベット 1 文字を要素とするベクトルは定義されている
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"
> letters[1:10] # 小文字の場合
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
                                                                    (vector.r)
```

1.3.2. 行列. 一般に行列は関数 `matrix()` を用いて生成することができる．行列の (i, j) 成分を取り出すには，`[i,j]` をつければよい．

```
> ## 関数 matrix の使い方
> x <- c(2, 3, 5, 7, 11, 13)
> matrix(x, 2, 3) # (2,3) 行列
     [,1] [,2] [,3]
[1,]    2    5   11
[2,]    3    7   13
> (X <- matrix(x, ncol = 3)) # 列数指定 (行数はそれに合わせて決まる)
     [,1] [,2] [,3]
[1,]    2    5   11
[2,]    3    7   13
> (Y <- matrix(x, ncol = 3, byrow = TRUE)) # 横に並べる
     [,1] [,2] [,3]
[1,]    2    3    5
[2,]    7   11   13
> # その他の操作
> nrow(X) # 行数
[1] 2
> ncol(X) # 列数
[1] 3
> X[1, 2] # (1,2) 成分
[1] 5
> X[2, ] # 2 行目
[1]  3  7 13
```

```
> X[, 3] # 3 列目
[1] 11 13
> as.vector(X) # ベクトル x に戻る
[1]  2  3  5  7 11 13
> dim(x) <- c(2, 3) # ベクトルに次元属性を与えて行列化
> x # 行列 X になる
     [,1] [,2] [,3]
[1,]    2    5   11
[2,]    3    7   13
```

(matrix.r)

1.3.3. リスト. リストは異なる構造のデータをまとめて 1 つのオブジェクトとして扱えるようにしたものである. リストの各要素は種類がバラバラであってもよい (例えばベクトルと行列が混在していてもよい). 一般に関数 `list()()` を用いてリストを作成する. 要素を取り出すには `[[]]` をつけて要素の番号を指定する. もしくは, 各成分に名前をつけることができるので, それを用いて各成分を参照することもできる.

```
> ## 関数 list の使い方
> # 各成分のデータ型はバラバラでよい
> (L1 <- list(c(1, 2, 5, 4), matrix(1:4, 2), c("Hello", "World")))
[[1]]
[1] 1 2 5 4

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

[[3]]
[1] "Hello" "World"
> L1[[1]] # L1 の第 1 成分の取り出し
[1] 1 2 5 4
> L1[[2]][2, 1] # 2 番目の要素の (2,1) 成分
[1] 2
> L1[[c(3, 2)]] # 3 番目の要素の 2 番目
[1] "World"
> L1[[3]][[2]] # 上と同じ
[1] "World"
> L1[c(1, 3)] # 複数成分を同時に抽出
[[1]]
[1] 1 2 5 4

[[2]]
[1] "Hello" "World"
> L1[1] # 第 1 成分をリストとして取り出す
[[1]]
[1] 1 2 5 4
> (L2 <- list(Info = "統計データ解析", List1 = L1)) # 名前付きリスト生成
$Info
[1] "統計データ解析"

$List1
$List1[[1]]
```

```
[1] 1 2 5 4

$List1[[2]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$List1[[3]]
[1] "Hello" "World"
> L2[["Info"]] # 要素名で取り出し
[1] "統計データ解析"
> L2$Info # 別記法
[1] "統計データ解析"
> names(L1) <- c("vector", "matrix", "character") # L1 の要素に名前をつける
> L1
$vector
[1] 1 2 5 4

$matrix
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$character
[1] "Hello" "World"
```

(list.r)

1.3.4. データフレーム. データフレームは同じ長さのベクトルを束ねたものであり，解析するデータを纏めた表と考えることができる．一般に関数 data.frame() を用いてデータフレームを作成する．要素を取り出すには [,] を付けて要素の行番号・列番号を指定すればよい．また，各行・各列には名前を付けることができるので，それを用いてデータを参照することもできる．

```
> ### 関数 data.frame の使い方
> (x <- data.frame( # 各項目が同じ長さのベクトルを並べる
+      month=c(4,5,6,7),            # 月
+      price=c(900,1000,1200,1100), # 価格
+      deal=c(100,80,50,75)))       # 取引量
  month price deal
1     4   900  100
2     5  1000   80
3     6  1200   50
4     7  1100   75
> x[2,3] # 2 行 3 列を取り出す
[1] 80
> x[3,] # 3 行目を取り出す
  month price deal
3     6  1200   50
> x[,2] # 2 列目を取り出す
[1]  900 1000 1200 1100
> x$price # price の列を取り出す
[1]  900 1000 1200 1100
> x[2]    # 2 列目だけからなるデータフレームを取り出す
  price
1   900
```

```
2  1000
3  1200
4  1100
> ### 行・列の名前の操作
> rownames(x) # 行の名前の表示
[1] "1" "2" "3" "4"
> rownames(x) <- c("Apr","May","Jun","Jul") # 行の名前の変更
> colnames(x) # 列の名前の表示
[1] "month" "price" "deal"
> colnames(x) <- c("tsuki","kakaku","torihiki") # 列の名前の変更
> x # 変更されたデータフレームの表示
    tsuki kakaku torihiki
Apr     4    900      100
May     5   1000       80
Jun     6   1200       50
Jul     7   1100       75
> x["May","kakaku"] # 特定の要素を名前で参照
[1] 1000
```

(data.frame.r)

演習 1.3. 実際のデータに基づいてデータフレームを作成してみよう.

(1) 長さの等しいベクトルを作成する.
(2) ベクトルを束ねてデータフレームを作成する.
(3) データフレームの行・列に適当な名前に変更する.

1.4. 参考文献

1. 金明哲著「R によるデータサイエンス (第 2 版)」, 森北出版 (2017 年).
2. U. リゲス著, 石田基広訳「R の基礎とプログラミング技法」, 丸善出版 (2012 年).