Rのための言語入門

はじめに

お久しぶりです。Python に引き続きやってまいりました。

今回は、R 言語です。

Python の時以上に、見やすく、わかりやすい資料にできたらと思います。

もしわからないところなどございましたら、気軽に聞いてください。

それでは、R 言語の世界へ Let’s GO!!

目次

1 Rってなに？

1.1 R 言語について

R 言語は、統計やデータ解析に特化したプログラミング言語です。

Python などと同じく、データ解析などを得意としていて、なおかつ Python などに比べて統計解析のプログラムを比較的短く記述することができます。

拡張子は「.r」もしくは「.R」です。

1.2 開発環境の構築

では、実際に開発環境を整えていきましょう。

まず、次のサイトに行き、R をダウンロードしてください。

https://cran.ism.ac.jp/

Windows であれば”Download R for Windows”を、Mac であれば”Download R for Mac”をクリックし、”install R for the first time”をクリックします。すると、”Download R-X.X.X for windows”(Mac なら for Mac)というボタンがあります。ここを押すと、R 言語をダウンロードできます。

PC にダウンロード出来たら、ダウンロードした.exe ファイルを実行しましょう。

すると、インストールで使用する言語を選択する画面が出てきます。ここで、「日本語」を選択し、「次へ」を押します。

その後は、何も変更することはありません。どんどん「次へ」を押していきましょう。

すると、ダウンロードが完了します。

これより先は、開発する方法によって異なります。自分の好きな方法で開発しましょう。

・RStudio を使用して開発する方法

これが一番オーソドックスな開発方法かと思います。

RStudio を使用した開発環境を構築するのは簡単です。

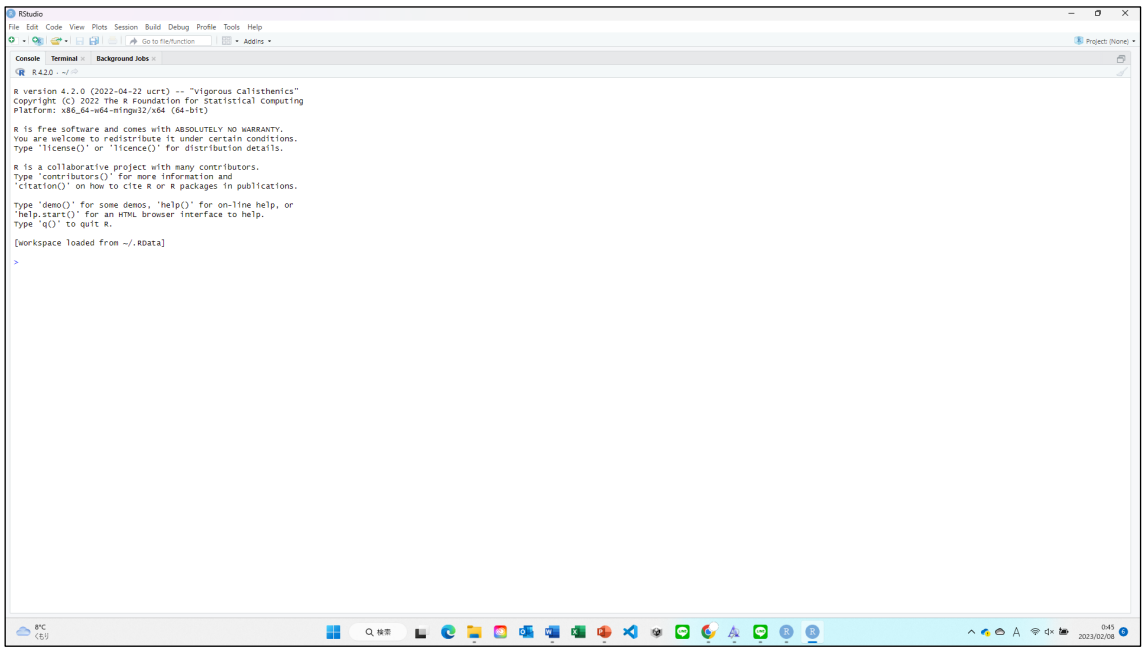
まず、次のサイトに行き、RStudio をダウンロードします。

https://posit.co/download/rstudio-desktop/

ダウンロードができたら、これもまたどんどん「次へ」を押して、インストールします。

インストールが終わったら、RStudio を開いてみましょう。

下のような画面になれば正しくインストールができています。



これにて、環境構築は終了です。

・Visual Studio Code を使用して開発する方法

この方法だと、R 以外の開発などに使用できるため、これもおすすめです。

まず、次のサイトに行き、Visual Studio Code をダウンロードしてください。

https://code.visualstudio.com/download

ここで自分のパソコンの OS に合うものをダウンロードしてください。

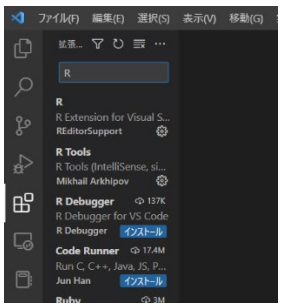
ダウンロードしたファイルを実行すると、「使用許諾契約書」と書かれた画面が出てきます。

これに同意して、「次へ」を押します。

ダウンロード先はいじる必要はないと思います。「次へ」を押します。

「追加タスクの選択」という画面で。「デスクトップ上にアイコンを作成する」にチェックを入れて「次へ」を押します。

最後に、「インストール」を押します。



インストールが完了したら、Ctrl+Shift+X で、拡張機能のダウンロードページを開き、検索窓に「R」と入力します。

そして、左の写真にある「R」というものと、「R Tools」という二つの拡張機能をダウンロードします。

これが完了したら環境構築は完了です。

1.3 作業ディレクトリの設定

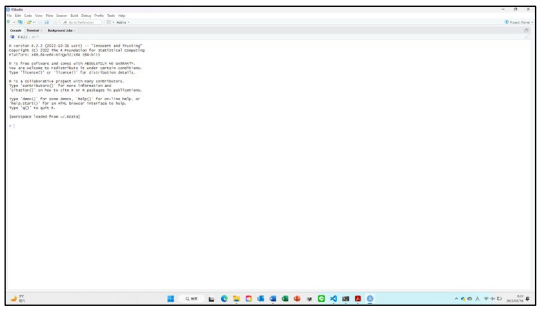
では、ファイルを作成する前に、作業を行うディレクトリを設定しましょう。

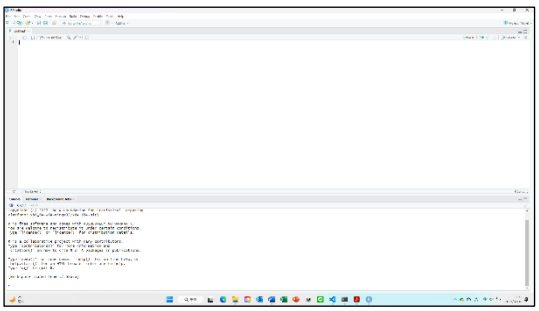
作業ディレクトリは、「作業の際に作成したファイル等を保存する場所」という解釈で大丈夫です。

実際に設定をしてみましょう。

まず最初に、現在の作業ディレクトリを確認してみましょう。

RStudio を開いてください。(VSCode を使用している場合は読み飛ばして大丈夫です。)





RStudio なら上の二つの写真のどちらかのような画面になっていると思います。

Console と書いてある場所がありますね。

この部分を、これ以降本書では「実行欄」と記述します。

では、現在の作業ディレクトリの特定をしてみましょう。

実行欄に「getwd()」と入力し、Enter を押してみましょう。

このようにして出てきたものが、現在あなたがいる作業ディレクトリです。

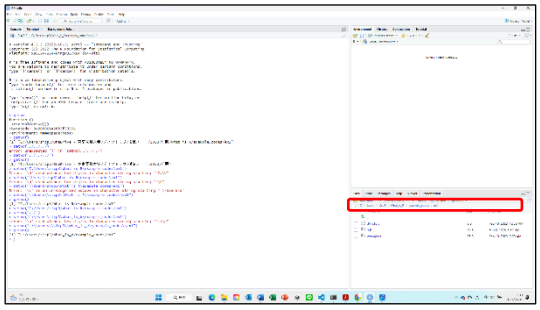
では、作業ディレクトリに任意のフォルダを設定してみましょう。

その準備として、「ex1」というフォルダを PC 上の任意の位置に作成してみましょう。

この ex1 に、この章で作成するファイルやデータを入れていこうと思います。

・R Studio で作業ディレクトリを設定する(その 1)

では、一番簡単に作業ディレクトリを設定する方法を見ていきましょう。



Console の右端にマウスを持っていき、console の幅を狭くします。

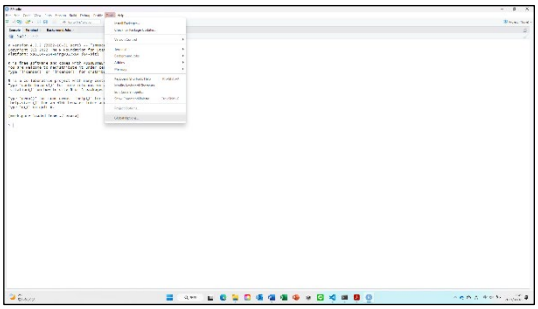
すると、以下のような画面になると思います。

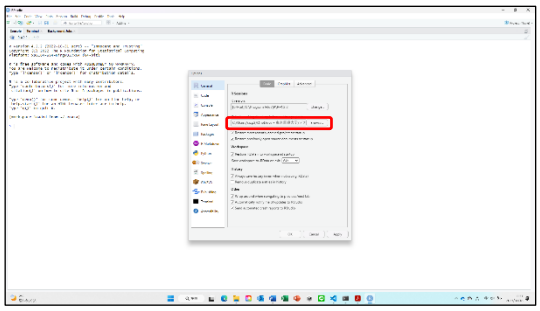
このようにできたら、赤枠の部分を操作して、先ほど作った「ex1」フォルダに移動します。

移動出来たら、More > Set As Working Directory を押します。

そうすることによって、ex1 フォルダが作業ディレクトリになりました。
getwd()を使用して確認してみてください。

・R Studio で作業ディレクトリを設定する(その 2)

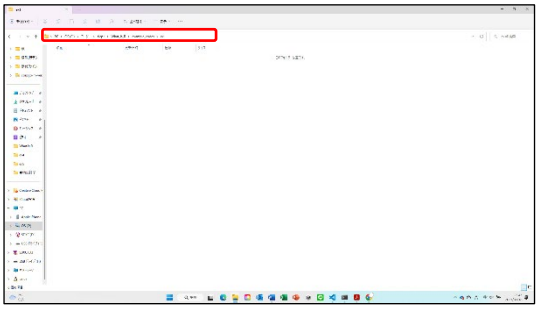




Tool を使用して設定する方法を見ていきましょう。
まず、上部にあるツールバーから Tools ＞ Global Options を選択します。
このようにしたら、下の写真のような画面が表紙されます。
そこで、赤枠で囲ってある Browse ボタンをおして、先ほど作成した ex1 フォルダを選択します。
そうしたら、Apply ＞ OK と押し、画面が閉じられたら RStudio 自体を一度閉じ、再度起動してみましょう。
起動したら実行欄に「getwd()」と入力し、「"○○/ex1"」と表示されていることを確認してください。

・R Studio で作業ディレクトリを設定する(その 3)・VSCode で設定する

では、コマンドを使用して作業ディレクトリを設定する方法を見ていきましょう。VSCode を使用する場合はこのような設定方法のみです。



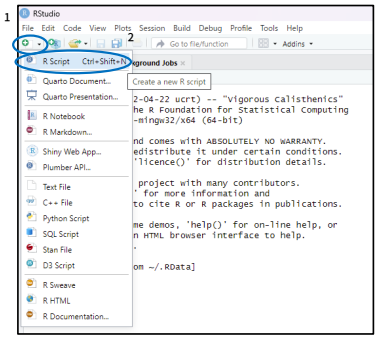
先ほど作成した ex1 フォルダをエクスプローラーで開いてみましょう。
この画面の赤枠で囲ってあるところをクリックして、「C:」から始まる文字列をコピーしましょう。
コピー出来たら、実行欄に、setwd("コピーした文字列(\となっている部分を\\に直したもの)")と入力して、Enter を押します。
入力する文字列は、コピーした状態が「C:\Users\sicp3\What_is_R\example_codes\ex1」なら、「C:\\Users\\sicp3\\What_is_R\\example_codes\\ex1」というように、\\という形に直す必要があります。

1.3 ファイルの作成・実行方法

では、各環境におけるファイルの作成・実行方法を見ていきましょう。
ではまず、簡単な R 言語のファイルを作成しましょう。

ファイル名に日本語は極力使用しないようにしましょう。

・Rstudio の場合



①1 のボタンをクリックする

②2 のボタンをクリックする

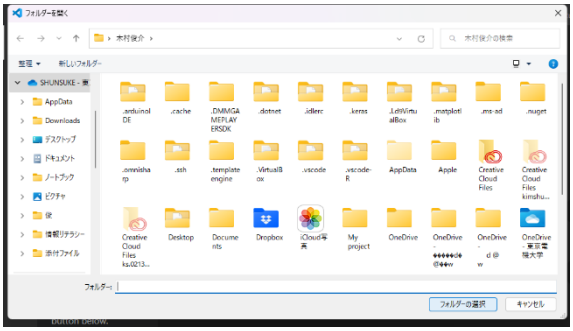
この手順を踏むと、Untitled という名前のファイルが画面上部に出てきます。

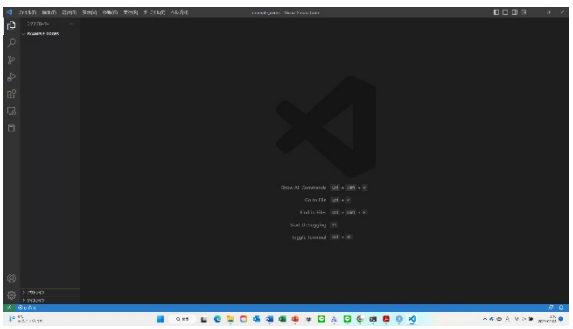
ここで Ctrl+S で、自分の好きなフォルダに好きなファイル名.r で保存をします。

以上が、RStudio におけるファイルの作成方法です。

・Visual Studio Code の場合

まず、Visual Studio Code(以降 VSCode と表記)を起動します。





起動したら Ctrl+K,Ctrl+O を続けて押し、フォルダ選択画面を出します。

ここで、自分の好きなフォルダに移動(場合によってはここでフォルダを作成)し、「フォルダを選択」をクリックし、フォルダを開きます。

すると、下の写真のような画面になります。

ここで、Ctrl+N を押し、Untitled という名前のファイルを作成します。

そうしたら、同じように Ctrl+S で、「任意のファイル名.r」という名前で、名前を付けて保存します。

VSCode で気を付けることは、保存の際に拡張子を自分で変える必要があるという事です。

では、ファイルの作成ができたところで、簡単なプログラムを書いて、実行してみましょう。

以下の文を、先ほど作成したファイルに打ち込んで、保存してください。

1.r

```
print("Hello world")
```

保存したら、コードを全選択した後、Rstudio なら「RUN」ボタンを、VSCode なら Ctrl+Enter を押してみてください。

すると、どちらも画面下のターミナルに、

[1] "Hello world"
と表示されたと思います。
これが表示されていれば、正しく実行できています。

これ以降、プログラムと、その実行結果例を以下のように表記します。

```
print("This is Example") #プログラム例

> print("This is Example") #プログラム例
[1] "This is Example"
```

以降省略

また、実行結果欄に入力したものは、以下のようにピンク字で表記し、それによる実行結果は>>>を省略して表記します。

```
print("This is Example") #プログラム例

>This is Key input. #入力したもの
[1] "This is Example"
```

また、章ごとにプログラムを保存するフォルダは変えるようにしましょう。

2 Rのキソ

R の環境構築、実行方法が分かったところで、いよいよ R 言語の学習が始まります。

Let's GO!

2.1 文字の表示

では、まず最初に文字の表示をしてみましょう。

新しいファイルを作成し、以下のように打ち込んで実行してみましょう。

1.r

```
print("Hello world")


[1] "Hello world"
```

どうでしょう。きちんと表示されましたか？

このように、Python と同じ方法で文字の表示ができるということが分かります。

2.2 R における変数

では次に、変数について見ていきましょう。

R 言語も、Python と同じく、インタープリタ言語です。

そのため、C や Java などのように、最初に int や、char などを使用して変数型を定義する必要がありません。

これが Python が初心者におすすめと言われる理由であり、同時に分かりにくい理由です。

さて、R での変数も、Python と同じように定義ができます。

~R での変数名に関するルール~

①使えるのは文字・数字・ドット・アンダーバー

②名前の先頭は必ず文字かドット

③予約語(if,else など)は使用できない

名前の付け方についてのルールは上記のとおりです。

では、実際に変数を作成し、そこに文字列を代入して表示してみましょう。

以下のコードを打ち込み、実行してみましょう。

2.r

```
moji <- "文字"
print(moji)

[1] "文字"
```

実行できましたか？

R では、代入をするときに「 = 」ではなく「 <- 」を使用します。

それ以外はここも Python と大差ありません。

2.3 データの種類

では次に、R で扱うことのできるデータの種類を見ていきましょう。

下の表を見てみましょう。

データ型	例	変数に代入して型を確認する (3.r)

真偽(Logical)	TRUE,FALSE	`v <- TRUE`
		`print(class(v))`
		
		`[1] "logical"`

数値(Numeric)	12.3,4,100	`v <- 12.3`
		`print(class(v))`
		
		`[1] "numeric"`

整数(Integer)	2L,34L,0L	`v <- 12L`
		`print(class(v))`
		
		`[1] "integer"`

複素数(Complex)	3+2i	`v <- 3+2i`
		`print(class(v))`
		
		`[1] "complex"`

文字列(Character)	“a”,”TDU”,”12.3”	`v <- "TDU"`
		`print(class(v))`
		
		`[1] "character"`

文字コード(Raw)	“Hello”の例 48 5 6c 6c 6f	`v <- charToRaw("Hello")`
		`print(class(v))`
		
		`[1] "raw"`

リスト(List)	21.3,2L,”Hello”	`v <- list(21.3,2L,"Hello")`

		print(class(v)) [1] "list"

これらは今までの Python でも見てきたものたちなので、見覚えがあるでしょう。

では、**R だからこそ扱うことのできるデータ型**を見ていきましょう。

・ベクトル(Vector)

R 言語では、ベクトルを扱うことができます。

ベクトルとは言っても、数学で勉強したベクトルとは少し違い、これは「数字の列」です。

では、定義方法を見ていきましょう。

以下のコードを打ち込み、実行してみましょう。

4.r

```
x <- c(1,2,3)
y <- c(1:5,3:1)
z <- c(rep(1,10))

> x
[1] 1 2 3
> y
[1] 1 2 3 4 5 3 2 1
> z
[1] 1 1 1 1 1 1 1 1 1 1
```

詳しく見ていきましょう。

まず、R でベクトルを定義するときは、c()を使います。

1 行目では、変数 x に 1,2,3 という数字の列のベクトルを代入しています。このため、実行画面で x を入力して Enter を押すと、[1]1,2,3 と表示されています。

2 行目では、変数 y に 1~5 の数字の列+3~1 の数字の順番の列のベクトルを代入しています。このため、実行画面で y を入力して Enter を押すと、[1]1,2,3,4,5,3,2,1 と表示されています。

3 行目では、変数 y に 1 を 10 回繰り返したものを数字の列としたベクトルを代入しています。このため、実行画面で z を入力して Enter を押すと、[1]1,1,1,1,1,1,1,1,1,1 と表示されています。

・行列(Matrix)

R では、行列を扱うこともできます。

以下のコードを打ち込み、実行してみましょう。

5.r

```
mat1 <- matrix(c(1:6),nrow=2,ncol=3,byrow = TRUE)

> mat1
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

詳しく見ていきましょう。

まず、R で行列を定義するときは、matrix()を使います。

1 行目で、変数 x に 1~6 の数字のベクトルを、2 行 3 列、1 行目から順にデータを埋める方式で行列化したものを代入しています。このため、実行画面で x を入力し、Enter を押すと、2 行 3 列で行列が表示されます。

行数・列数の指定方法を見てみましょう。

まず、行数の指定方法です。

行数の指定は、nrow= で行います。この後の数値を変えることで、行数の指定が可能です。

今回、ベクトルに含まれる数字の数は 6 個です。ここで、もし 4 や 5 など、割り切れない数値を指定すると、エラーになってしまうので、注意が必要です。

次に、列数の指定方法です。

列数の指定は、ncol= で行います。この後の数値を変えることで、列数の指定が可能です。

また、行数と同じように、数値には注意が必要です。

では、先ほどのコードの byrow= を FALSE に変えて実行するとどうなるでしょう。

先ほど作成したファイルに、以下のように追記し、実行してみましょう。

```
mat2 <- matrix(c(1:6),nrow=2,ncol=3,byrow = FALSE)

> mat2
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

実行してみると、先ほどとは違い、1 列から順にデータが埋められていることが分かります。

・配列(Array)

配列は、ベクトルと似ていますが、次元を自由に設定できます。

以下のコードを打ち込み、実行してみましょう。

6.r

```
arr1 <- array(1:24,c(3,4))

> arr1
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

詳細を見ていきましょう。

まず、R で配列を定義するときは、array()を使用します。

array(入れる数字,c(行数,列数))という風に定義します。

しかし、このままではベクトルと何ら変わりがありません。

では、配列の特徴は何でしょう。

それは、**任意の次元の指定ができること**です。

ここで、配列の次元についておさらいしていきましょう。

以下のコードを見てみましょう。

7.r

```
arr2 <- array(1:24,c(24))      #1 次元
arr3 <- array(1:24,c(4,6))     #2 次元
arr4 <- array(1:24,c(3,4,2))   #3 次元

> arr1
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
> arr2
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
> arr3
, , 1

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

```
, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

まず、arr1 が入力されたときの実行結果を見てみましょう。

このように、座標系に換算した際に x 座標のみで数値が特定できるものが、1 次元の配列でした。

では、arr2 が入力されたときの実行結果を見てみましょう。

ここでは、座標系に換算したときに x,y 座標を使用して数値を特定する必要があります。このようなものが 2 次元配列でした。

最後に、arr3 が入力されたときの実行結果を見ていきましょう。

この場合、座標系に換算した際に、x,y,z 座標を使用して数値を特定する必要があります。これが 3 次元配列でした。

実際に 3 次元配列を定義してみましょう。

では、以下のコードを打ち込み、実行してみましょう。

8.r

```r
arr4 <- array(1:24,c(3,4,2))
```

```
> arr4
, , 1

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

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

このようにすると、3 次元の配列が作成できていることが分かります。

さて、今まで見てきたベクトル・行列・配列のうち、次元を持つものは行列・配列の 2 つです。

この二つは、座標を指定して特定の値を出力させることができます。

では、先ほど書いたプログラムを実行し、実行画面に以下のように入力してみましょう。

```
> arr4[2,3]
[1] 8
```

このように、特定の値のみ出力できるということが分かります。

また、先ほどの入力を arr4[2,]とすれば、2 行目の値が全て、arr4[,3]とすれば 3 列目の値が全て表示されます。

2.4 コメントの挿入

では、プログラム中にメモなどのコメントを挿入する方法を見ていきましょう。

R でのコメントの挿入方法は、Python と同じく「 # 」を付けることです。

以下のコードを打ち込み、実行してみましょう。

9.r

```
print("この行は実行されます。")
# print("この行は実行されません。")
# これはメモです。

>[1] "この行は実行されます。"
```

このように、#を置くとそれ以降は改行するまで実行されないことが分かります。

2.5 章末問題

問題①

ベクトル、行列、配列のうち、次元を持つものを全て選びなさい。

問題②

以下のように表示されるベクトル「Vec_Q2」を作成しなさい。

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

問題③

以下のように表示される行列「Mat_Q3」を作成しなさい。

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
```

入力しなさい。

問題⑤

以下のように表示される配列「Arr_Q4」を作成しなさい。

```
, , 1

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

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24

, , 3

     [,1] [,2] [,3] [,4]
[1,]   25   28   31   34
[2,]   26   29   32   35
[3,]   27   30   33   36
```

3 Rと行列

この章では、行列について見ていきます。
線形代数で勉強したいろいろが出てきます。頑張りましょう。

3.1 行列の四則演算

では、行列の四則演算の方法を見ていきましょう。

まず、以下のコードを打ち込んだファイルを用意しておきましょう。

1.r

```
mat1 <- matrix(c(1:9),nrow = 3,ncol = 3)
mat2 <- matrix(c(10:18),nrow = 3,ncol = 3)
```

・各要素に値を足す

行列の各要素に値を足す方法を見ていきましょう。

実行欄に以下のように入力してみましょう。

```
> mat1+3
     [,1] [,2] [,3]
[1,]    4    7   10
[2,]    5    8   11
[3,]    6    9   12
```

このように、すべての要素に 3 が足されていることが分かります。

・各要素から値を引く

では、行列の各要素から値を引く方法を見ていきましょう。

実行欄に以下のように入力してみましょう。

```
> mat1-3
     [,1] [,2] [,3]
[1,]   -2    1    4
[2,]   -1    2    5
[3,]    0    3    6
```

このように、すべての要素から 3 が引かれていることが分かります。

・各要素に値をかける

では、行列の各要素に値をかける方法を見ていきましょう。

```
> mat1*2
     [,1] [,2] [,3]
[1,]    2    8   14
[2,]    4   10   16
[3,]    6   12   18
```

このように、すべての要素に 2 がかけられていることが分かります。

・各要素の値を割る

では、行列の各要素の値を割る方法を見ていきましょう。

```
> mat1/2
     [,1] [,2] [,3]
[1,]  0.5  2.0  3.5
[2,]  1.0  2.5  4.0
[3,]  1.5  3.0  4.5
```

このように、すべての要素に 2 が割られていることが分かります。

・行列の対応する要素同士の足し算

まずは、足し算の方法です。

先ほど用意したファイルを実行して、以下のように入力してみましょう。

```
> mat1+mat2
     [,1] [,2] [,3]
[1,]   11   17   23
[2,]   13   19   25
[3,]   15   21   27
```

このような入力を行うと、行列の対応する要素の足し算を行うことができます。

また、実行欄に入力せずに、print(mat1+mat2)と記述する方法もあります。

・行列の対応する要素同士の引き算

では、引き算を見ていきましょう。

実行欄に以下のように入力してみましょう。

```
> mat1-mat2
     [,1] [,2] [,3]
[1,]   -9   -9   -9
[2,]   -9   -9   -9
[3,]   -9   -9   -9
```

このような入力を行うと、行列の対応する要素の引き算を行うことができます。

・行列の同じ位置にある要素同士の掛け算

では、掛け算を見ていきましょう。

実行欄に以下のように入力してみましょう。

```
> mat1*mat2
     [,1] [,2] [,3]
[1,]   10   52  112
[2,]   22   70  136
[3,]   36   90  162
```

このような入力を行うと、同じ位置にある要素同士の掛け算を行うことができます。

・行列の同じ位置にある要素同士の割り算

では、割り算を見ていきましょう。

実行欄に以下のように入力してみましょう。

```
> mat1/mat2
          [,1]      [,2]      [,3]
[1,] 0.1000000 0.3076923 0.4375000
[2,] 0.1818182 0.3571429 0.4705882
[3,] 0.2500000 0.4000000 0.5000000
```

このような入力を行うと、同じ位置にある要素同士の割り算を行うことができます。

・行列の掛け算

さて、今までやってきた計算は、「対応する要素」「同じ位置にある要素」同士の四則演算でした。

では、行列同士の掛け算を行う方法を見てみましょう。

実行欄に以下のように入力してみましょう。

```
> mat1%*%mat2
     [,1] [,2] [,3]
[1,]  138  174  210
[2,]  171  216  261
[3,]  204  258  312
```

このように、行列の掛け算ができていることが分かります。

3.2 行列に使用できる関数

では、行列に使用できる関数を見ていきましょう。

以下のコードを打ち込んだファイルを用意し、実行しましょう。

2.r

```
mat3 <- matrix(c(1:4),nrow = 2,ncol = 2)
```

・行列式

まず、行列式を求める関数です。

実行欄に以下のように入力してみましょう。

```
> det(mat3)
[1] -2
```

このように、行列式が求められています。

・逆行列

次に、逆行列を求める関数です。

実行欄に、以下のように入力してみましょう。

```
> solve(mat3)
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

このように、逆行列が求められています。

・単位行列

単位行列を作成することも可能です。

実行欄に以下のように入力してみましょう。

```
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

このように、単位行列を作る際は、diag()関数を使用し、()内に行数を入れます。

3.3 章末問題

・問題 1

以下のように表示される行列「Mat_Q1_1」を作成し、この行列の各要素の値を 3 倍したものを代入した行列「Mat_Q1_2」を作成しなさい。

```
> Mat_Q1_1
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

・問題 2

以下のように表示される行列「Mat_Q2_1」、「Mat_Q2_2」を作成し、この二つの行列をかけた行列「Mat_Q2_3」を作成しなさい。

```
> Mat_Q2_1
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> Mat_Q2_2
     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18
```

・問題 3

以下のように表示される行列「Mat_Q3_1」を作成し、この行列の逆行列を代入した「Mat_Q3_2」を作成なさい。

```
> Mat_Q3_1
     [,1] [,2]
[1,]   13   14
[2,]   15   16
```

・問題 4

以下のように表示される行列「Mat_Q4_1」の行列式を代入した「Mat_Q4_2」を作成しなさい。

```
> Mat_Q4_1
     [,1] [,2] [,3]
[1,]   11   14   17
[2,]   12   15   18
[3,]   13   16   19
```

4 Rとデータ

R で統計解析などを行うときに、実験の結果などを入力したファイルを読み込んで使用することがあります。
この章では、CSV ファイル(エクセルから変換ができます)と、テキストファイルを読み込む方法を見ていきましょう。
自分で CSV ファイルを作成するときは、セルの結合はしないように注意してください。

4.1 R と CSV

では、まず最初に CSV ファイルの読み込みかたを見ていきましょう。

以下のページにサンプルの CSV ファイルがあります。まずはページにアクセスして CSV ファイルをダウンロードしてください。

https://kimshun0213kr.github.io/R_tutorial_for_RB/sample/4/example.csv

ダウンロード出来たら、そのファイルをこの章で作成するコードを保存するフォルダに移動させてください。

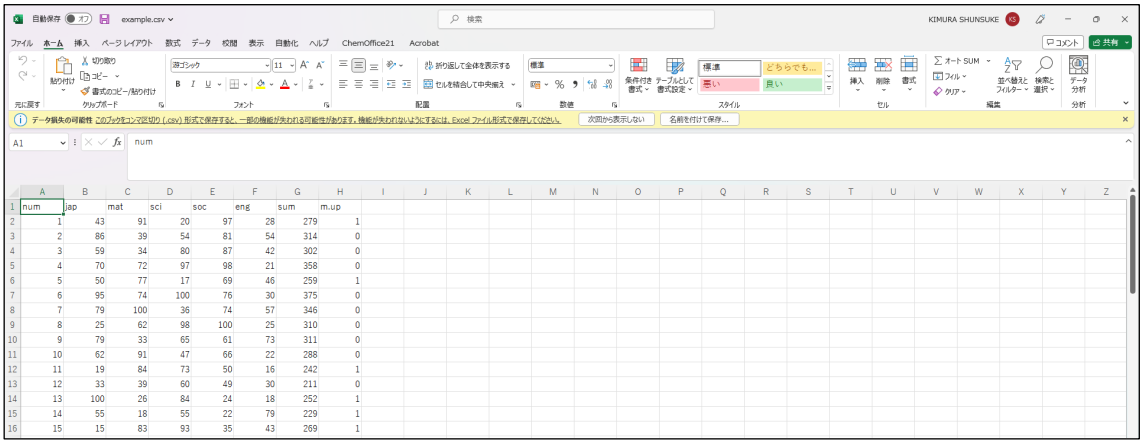
移動が出来たら、以下のコードを打ち込んだファイルを作成してみましょう。

1.r

```
test <- read.csv("example.csv",header = TRUE,row.names = 1)
> test
    jap mat sci soc eng sum m.up
1    43  91  20  97  28 279    1
2    86  39  54  81  54 314    0
3    59  34  80  87  42 302    0
.....
```

このように、ズラーっと数字が表示されたと思います。

では、csv ファイルがどのようになっているのか、開いて確認してみましょう。



num	jap	mat	sci	soc	eng	sum	m.up
1	43	91	20	97	28	279	1
2	86	39	54	81	54	314	0
3	59	34	80	87	42	302	0
4	70	72	97	98	21	358	0
5	50	77	17	69	46	259	1
6	95	74	100	76	30	375	0
7	79	100	36	74	57	346	0
8	25	62	98	100	25	310	0
9	79	33	65	61	73	311	0
10	62	91	47	66	22	288	0
11	19	84	73	50	16	242	1
12	33	39	60	49	30	211	0
13	100	26	84	24	18	252	1
14	55	18	55	22	79	229	1
15	15	83	93	35	43	269	1

このようになっているのが分かります。

これは、100 人の生徒の 5 教科のテストの各教科の点数と合計点、追試教科(20 点以下の教科)の有無を打ち込んだファイルです。

では、コードを詳しく見ていきましょう。

まず、読み込む際に使用する関数です。

読み込むには、read.csv(“ファイル名”)と記述します。そして、header = TRUE の部分で 1 行目を列の名前として扱うか否かを、row.names = 1 の部分で 1 行目を読み込みの際の行の名前に指定しています。

header = FALSE とすると、1 行目は列の名前として読み込まれません。今回は 1 行目に列の名前を入力してあるので、TRUE にします。

さて、csv ファイルを R で表示すると、行列のように表示がされました。

しかし、csv を読み込んだとき、これはデータフレームになっています。

データフレームは、行列と違い、各データが列名・行名を持っていて、列名・行名を使用しての操作が可能になります。

4.2 データの参照方法

データフレームの特徴は、先ほど述べた通り、列名・行名を指定して特定の要素にアクセスできるという事です。

列名・行名の事を、ラベルと言います。

では、見ていきましょう。

・列を指定しての参照

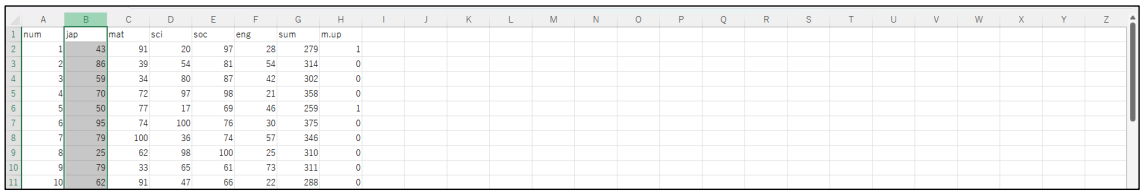
まずは、列を指定しての参照方法です。

では、先ほどの R ファイルを実行した状態で以下のように実行欄に入力してみましょう。

```
> test[,1]
  [1]  43  86  59  70  50  95  79  25  79  62  19  33 100  55  15  41
87  29
......
```

このように表示されたと思います。

これは、どこのデータが表示されているのかというと、以下の場所です。



	num	jap	mat	sci	soc	eng	sum	m.up
2	1	43	91	20	97	28	279	1
3	2	86	39	54	81	54	314	0
4	3	59	34	80	87	42	302	0
5	4	70	72	97	98	21	358	0
6	5	50	77	17	69	46	259	1
7	6	95	74	100	76	30	375	0
8	7	79	100	36	74	57	346	0
9	8	25	62	98	100	25	310	0
10	9	79	33	65	61	73	311	0
11	10	62	91	47	66	22	288	0

jap の列の 2 行目以降が表示されています。

なぜ 2 行目以降が表示されているかというと、header=TRUE としているからです。

header=TRUE としていることで、1 行目を列の名前としているので、データは 2 行目以降ということになります。

このように、変数名[,列番号]とすることで、指定した列番号のデータを参照することができます。

では、せっかく 1 行目を列名としているのですから、列名を使用してデータを参照する方法も見ていきましょう。

実行欄に以下のように打ち込んでみましょう。

```
> test$jap
  [1]  43  86  59  70  50  95  79  25  79  62  19  33 100  55  15  41
87  29
……
```

このように、変数名$列名とすることで、指定した列のデータを参照することができます。

では、複数の列を指定して参照する方法を見てみましょう。

実行欄に以下のように入力してみましょう。

```
> test[,c(1:3)]
    jap mat sci
1    43  91  20
2    86  39  54
……
```

このように、1~3 列目のデータが、番号と列名付きで表示されています。

参照する範囲の指定の際に、c(1:3)とせず、1:3 としても参照することは可能です。

・行を指定しての参照

次に、行を指定しての参照方法です。

実行欄に以下のように入力してみましょう。

```
> test[1,]
  jap mat sci soc eng sum m.up
1  43  91  20  97  28 279    1
```

このように、1 人目のデータのみを参照することができました。

行は番号のみしかリンクしていないため、名前を指定しての取り出しはできません。

では、複数の行を指定して参照する方法を見ていきましょう。

実行欄に以下のように入力してみましょう。

```
> test[c(1:3),]
  jap mat sci soc eng sum m.up
1  43  91  20  97  28 279    1
2  86  39  54  81  54 314    0
3  59  34  80  87  42 302    0
```

このように、1~3 人目のデータのみを参照することができました。

・行と列を指定しての参照

では、行と列のどちらも指定しての参照方法を見ていきましょう。

実行欄に以下のように入力してみましょう。

```
> test[1,3]
[1] 20
```

このように表示されていると思います。

これは、1 人目の 3 列目のデータ(sci の点数)です。

このように、配列と同じように特定の一つの要素を参照することが可能です。

では、複数行と複数列を組み合わせて呼び出す方法も見ていきましょう。

実行欄に以下のように入力してみましょう。

```
> test[c(1:3),c(1:3)]
  jap mat sci
1  43  91  20
2  86  39  54
3  59  34  80
```

このように表示されたと思います。

これは、1~3 人目の jap,mat,sci の点数のみを表示している状態です。

このように、範囲指定をしての参照方法も可能であることが分かります。

では、列名を指定した後に、行を指定しての参照は可能なのでしょうか。

実行欄に以下のように入力してみましょう。

```
> test$jap[1:3]
[1] 43 86 59
```

このように、変数名$列名[行範囲]とすることで、列名と行を指定しての参照が可能であることが分かります。

4.3 データの取り出し方法

参照方法を学んだところで、次は取り出し方法を見ていきましょう。

参照と取り出しの異なるところは、参照は範囲を使用し、取り出しは条件を使用するというところです。

では見ていきましょう。

・追試対象の人(m.up が 1 の人)のみ取り出す

ではまず、追試対象の人のデータのみを取り出してみます。

実行欄に以下のように入力してみましょう。

```
> subset(test,test$m.up==1)
    jap mat sci soc eng sum m.up
1    43  91  20  97  28 279    1
5    50  77  17  69  46 259    1
11   19  84  73  50  16 242    1
……
```

このように、m.up の列が 1 の人のデータのみ取り出されていることが分かります。

この取り出し方は 3 種類あります。ほかの方法も見てみましょう。

```
> subset(test,test[,7]==1)
    jap mat sci soc eng sum m.up
1    43  91  20  97  28 279    1
5    50  77  17  69  46 259    1
11   19  84  73  50  16 242    1
……
```

```
> test[test$m.up==1,]
    jap mat sci soc eng sum m.up
1    43  91  20  97  28 279    1
5    50  77  17  69  46 259    1
11   19  84  73  50  16 242    1
……
```

この 3 つの方法があります。

新しく出てきた subset 関数について詳しく見ていきましょう。

subset(変数,条件)と記述します。

ここでは、変数は test、条件は test$m.up==1 です。

条件に使用できる条件演算子の一部を見ていきましょう。

演算子	使い方
==	一致
<=,>=	以下、以上
<,>	より大きい、未満
&	かつ(「変数かつ変数」とする場合は&&)

これらの条件演算子を使用することで、複雑な条件に一致するデータのみ取り出すことが可能です。

また、これらを使用して取り出したデータは、変数に代入することも可能です。

4.4 txt ファイルを読み込む

csv ファイル以外に、txt ファイルを読み込むこともできます。

では、以下のページにアクセスして、sample.txt をダウンロードしてください。その後、ダウンロードしたファイルをこの章で作成する R ファイルを保存するフォルダに移動してください。

https://kimshun0213kr.github.io/R_tutorial_for_RB/sample/4/example.txt

では、読み込んでみましょう。

以下のコードを打ち込んだファイルを作成し、実行してみましょう。

2.r

```
test_tx <- read.table("example.txt",header = TRUE,sep = "\t",skip = 0)


> test_tx
    num jap mat sci soc eng sum m.up
1     1  43  91  20  97  28 279    1
2     2  86  39  54  81  54 314    0
3     3  59  34  80  87  42 302    0
……
```

txt ファイルを読み込んだときは、1 列目に行番号が追加されます。

txt ファイルの読み込みの際は、read.table(“ファイル名”,header = ,sep = “区切りの文字”,skip = コメント行の行番号)と記述します。

詳細を見ていきましょう。

Header については csv の時と変わりません。

Sep は、この txt ファイルは tab 区切りの為、\t を指定しています。

Skip ですが、データに含めないメモを記述している場合は、その行番号を、していない場合は 0 を指定します。

データの参照方法・取り出し方法は csv と変わりません。

4.5 章末問題

・問題 1

csv ファイルや txt ファイルを R で読み込んだときのデータの型を答えなさい。

・問題 2

Sample.csv を代入した変数「Q2_csv」を作成し、Q2_csv から mat、eng の点数がどちらも 70 以上の生徒のデータのみを代入した変数「Q2_csv70」を作成しなさい。

・問題 3

以下のリンクから、Q3.txt というファイルをダウンロードして、コードを保存するフォルダに移動させてください。

https://tdu.box.com/s/iy05qr5zfy4kdp6xmo52vyplw7oh0018

このファイルの 12 行目はデータに含まれないメモで、区切りは tab、header はありません。

正しくこのファイルを読み込ませて、「Q3_txt」という変数に代入してください。

5 Rと統計①

この章から、統計について見ていきましょう。
まずは基礎の部分です。

5.1 度数分布とヒストグラム

ではまず、度数分布とヒストグラムを表示する方法を見ていきましょう。

・度数分布の表示

ではまず、度数分布の表示をしてみましょう。

以下のリンクから csv ファイルをダウンロードし、現在の作業ディレクトリに移動させてください。

https://kimshun0213kr.github.io/R_tutorial_for_RB/sample/5/example.csv

そして、以下のコードを打ち込んだファイルを作成し、実行してみましょう。

1.r

```
data1 <- read.csv("example.csv",header = TRUE,row.names = 1)
data_jap <- data1$jap

> table(data_jap)
data_jap
 15 16 17 19 24 25 26 27 29 30 32 33 35 36 38 39 41 42 43 45
 2  1  1  2  1  1  2  1  2  1  1  1  3  2  1  1  2  2  3  2
 46 50 51 52 53 55 56 59 60 61 62 63 64 66 68 69 70 71 75 76
 1  1  1  2  1  2  1  2  3  3  2  2  3  2  1  2  3  5  2  2
......
```

実行結果だけを見るとわかりにくいかもしれませんが、これで度数分布の表示ができています。

この表示結果を表に直してみます。

点数	15	16	17	19	24	25	26	27	28
度数	2	1	1	2	1	1	2	1	2

このようになります。

ここから、度数分布の表示には table()を使用することが分かります。

度数が 0 の所は表示されないため、点数が飛んでいるところがありますが、これは仕様です。気にしない

でください。

もし、範囲を指定して、度数が 0 の所も表示する場合は実行欄に以下のように入力しましょう。

```
> table(factor(data_jap,levels = 1:100))
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  1  1  0  2  0
.....
```

このように、度数が 0 の部分も表示されるようになりました。

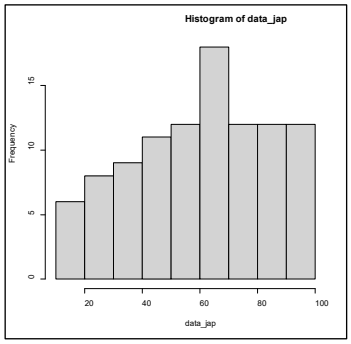
度数が 0 の所を含めて、範囲を指定して表示する場合は、table(factor(データ名,levels = 最小値:最大値))と記述します。

・ヒストグラムの描写

では次に、ヒストグラムの描写方法を見ていきましょう。

実行欄に以下のように入力してみましょう。

```
> hist(data_jap)
```



右のようなヒストグラムが表示されたと思います。

これが、data_jap(国語の得点)のヒヒストグラムです。

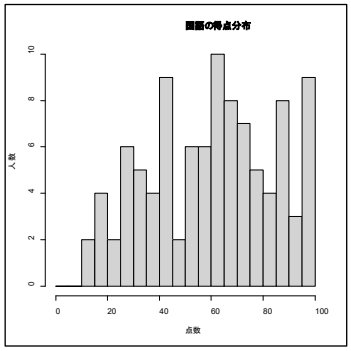
このように、hist(データ名)と記述することで、ヒストグラムを描写することができます。

しかし、この状態だと、名前や階級の幅が扱いやすいものではありません。

そこで、このヒストグラムの名前や階級幅、範囲を指定する方法を見ていきましょう。

実行欄に以下のように入力してみましょう。

```
>  hist(data_jap,breaks=seq(0,100,by=5),main="国語の得点分布",xlab="点数
",ylab="人数")
```



このように記述すると、グラフの名前が「国語の得点分布」に、x 軸の表示が「点数」に、y 軸の表示が「人数」になっていることが分かります。

また、階級の幅が、5 に変更されていることも見て取れます。

このように、階級の範囲と幅を指定するときは、breaks=seq(最小値,最大値,by=階級幅)と記述します。

また、グラフの名前は main=”名前”、x 軸の表示は xlab=”名前”、y 軸の表示は ylab=”名前”で変更できます。

また、ヒストグラムは、ウィンドウを右クリックすることで、画像として保存することができます。

メタファイルを選択すると、emf ファイルとして保存が可能です。これは、数式によって構成される画像ファイルの為、どれだけ拡大しても画質が落ちることはありません。

5.2 外部、または他のフォルダのコードの組み込み

では、外部にある R のファイルを参照して、それを組み込む方法を見ていきましょう。

以下のコードを打ち込んだファイルを作成し、実行してみましょう。

2.r

```
data1 <- read.csv("example.csv",header = TRUE,row.names = 1)
data_jap <- data1$jap

source("https://raw.githubusercontent.com/kimshun0213kr/R_tutorial_for
_RB/main/example_codes/ex5/source.r")

kekka <- table_percent(data_jap,5,1,100,1)

> kekka
    class c_value freq c_freq  rel com_rel
1     1~5       3    0      0 0.00    0.00
2    6~10       8    0      0 0.00    0.00
3   11~15      13    2      2 0.02    0.02
4   16~20      18    6      6 0.06    0.06
……
```

ここでは、以下のリンクにある R のコードを参照しています。

https://raw.githubusercontent.com/kimshun0213kr/R_tutorial_for_RB/main/example_codes/ex5/source.r

このサイトにあるコードは、任意の幅で 1~100 点の人数のヒストグラムを表示するコードです。

内容が気になった方はリンクから見てみてください。

外部のファイルを組み込むには、source(“URL”)と記述します。

このようにして組み込ませた後は、そのファイルの関数名や変数名が使用可能になります。

そのため、ここでは関数「table_percent」が使用可能になっています。

この関数は、「データファイル形式でのデータ(1 列)、度数の幅、取る度数の最小値、取る度数の最大値、度数の幅」の順に引数を取るため、「table_percent(data_jap,5,1,100,1)」と記述しています。

また、別のフォルダにあるコードの変数や関数を使用したい場合は、URL をファイルパスに変えて記述することで使用が可能になります。

5.3 平均値など代表値の求め方

ここからは、統計などでよく使われる値の求め方を見ていきましょう。

・平均値を求める

ではまず、平均値の求め方を見ていきましょう。

以下のコードを打ち込んだファイルを作成し、実行してみましょう。

3.r

```
data1 <- read.csv("example.csv",header = TRUE,row.names = 1)
data_jap <- data1$jap

ave1 <- sum(data_jap)/length(data_jap)
ave2 <- mean(data_jap)

> ave1
[1] 60.57
> ave2
[1] 60.57
```

このコードでは、2 種類の方法で平均を求めています。

ave1 は、すべての値の合計を「sum()」を使用して、値の個数を「length()」を使用してそれぞれ求め、合計を個数で割るという方法で平均を求めています。

合計値は sum(変数名、またはデータそのもの)、値の個数は length(変数名、またはデータそのもの)と記述することで求められます。

しかしながら、R は統計に強い言語です。

平均を求める関数がもともと内蔵されています。

ave2 は、平均を求める関数「mean()」を使用して平均を求めています。

このように、平均は「mean(変数名、またはデータそのもの)」と入力することで求めることができます。

・中央値を求める

次に、中央値の求め方を見ていきましょう。

先ほど作成したファイルに以下のコードを追記して実行してみましょう。

```
med1 <- median(data_jap)

> med1
[1] 62
```

このように、median(変数名、またはデータそのもの)と記述することで、中央値を求めることができます。

・最小値・最大値を求める

では、最小値・最大値の求め方を見ていきましょう。

先ほどのファイルに以下のコードを追記して、実行してみましょう。

```
min1 <- min(data_jap)
max1 <- max(data_jap)

> min1
[1] 15
> max1
[1] 100
```

このように、min(変数名、またはデータそのもの)と記述することで最小値を、max(変数名、またはデータそのもの)と記述することで最大値を求めることができます。

・その他の代表値を求める

では、これら以外の代表値を含むものの求め方を見ていきましょう。

先ほどのファイルに以下のコードを追記して、実行してみましょう。

```
summ1 <- summary(data_jap)

> summ1
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  15.00   41.75   62.00   60.57   79.00  100.00
```

このように、summary(変数名、またはデータそのもの)と入力すると、最小値、第一四分位数、中央値、平均値、第三四分位数、最大値の順で値を表示させることができます。

5.4 章末問題

問題①

第 4 章の章末問題で使用した Q3.txt をコピーしたもの(または、このリンクからダウンロードしたファイル https://kimshun0213kr.github.io/R_tutorial_for_RB/sample/5/Q3.txt)を現在の作業ディレクトリに移動させ、このファイルを適切な形で読み込み、2 列目のデータについてのヒストグラムを、グラフの名前を「テスト結果」に、x 軸の表示を「点数」に、y 軸の表示を「人数」に、最小値 1、最大値 100、

階級の幅を 5 に指定した状態で表示させなさい。

問題②

問題①で読み込んだデータの 3 列目の最小値、第一四分位数、中央値、平均値、第三四分位数、最大値を全て一度に表示させなさい。

6 Rと統計②

さて、先ほどの章で、ヒストグラムの描写方法と統計の基礎部分について学びました。
この章では、統計についてもっと踏み込んでみます。

6.1 散布度、分散、標準偏差

ではまず、散布度、分散、標準偏差について見ていきましょう。

散布度

まずは散布度です。

分布の特徴を把握しようとするときに、中心だけを見ていては不十分であり、データがどのように散布しているか（ばらついているか）を観察する必要があります。

この、データのばらつき具合の事を「散布度」と言います。

分散

では、ここでは分散の求め方を見ていきましょう。

以下のリンクから、CSV ファイルをダウンロードし、そのファイルをこの章で作成したファイルを保存するフォルダに移動させてください。

https://kimshun0213kr.github.io/R_tutorial_for_RB/sample/6/example.csv

以下のコードを打ち込んだファイルを作成し、実行してみましょう。

1.r

```r
data1 <- read.csv("example.csv",header = FALSE)

data2 <- data1[,1]
data_mean <- mean(data2)

dev1 <- data2 - data_mean
dev2 <- dev1^2
sum_dev <- sum(dev2)

data_length <- length(data2)

disp <- sum_dev/data_length
```

```
sqrt1 <- sqrt(disp)
var1 <- var(data2)

> disp
[1] 33.89
> var1
[1] 35.67368
```

ここでは、標本分散の値が disp に、不偏分散の値が var1 にそれぞれ代入されています。

これらの値が異なっているのは間違いではありません。

では、これら 2 つの分散の違いについて見ていきましょう。

まずは、標本分散です。

これは高校の時に数学で習ったかもしれません。

以下の式で求めることができます。

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$$

nはデータの個数、x_iが一つ一つの値、$\bar{x}$がこれらの平均値です。

この、標本偏差は、標本のみを考えての分散です。

では、不偏分散について見ていきましょう。

これは、以下の式で求めることができます。

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

この、不偏分散は、母集団も考えるという点で、標本分散と異なっています。

この二つを使うことで、母集団を推測することもできます。

標準偏差

では、標準偏差も見ていきましょう。

先ほどのファイルに、以下の内容を追記して、実行してみましょう。

```
sd1 <- sd(data2)
sqrt2 <- sqrt(sd(data2)^2*(data_length-1)/data_length)

> sd1
[1] 5.972745
> sqrt1
[1] 5.821512
> sqrt2
[1] 5.821512
```

sd1 が標準偏差の値、sqrt1 は標本分散の平方根の値、sqrt2 は sd 関数を使用して sqrt1 と同じ値を出す方法です。

この説明からも分かるように、sqrt1、sqrt2 の値は標本分散「disp」の平方根に一致しています。

また、sd1 の値は、var1 不偏分散「var1」の平方根に一致しています。

6.2 平均偏差、範囲、標準化、偏差値

では次に、平均偏差や範囲の求め方、標準化について見ていきましょう。

平均偏差、範囲

まずは、平均偏差と範囲について見ていきましょう。

以下のように打ち込んだファイルを作成し、実行してみましょう。

2.r

```
vec1 <- c(8,6,5,9,13)
mean1 <- mean(vec1)
mean_dev <- mean(abs(vec1-mean1))
range1 <- max(vec1)-min(vec1)

> mean_dev
[1] 2.24
> range1
[1] 8
```

このような結果になったと思います。

変数「mean_dev」が、平均偏差、「range1」が範囲です。

標準偏差と平均偏差の違いは、絶対値を用いるか否かです。

標準偏差は、各値と平均値との差を 2 乗したものの平均値の平方根です。

それに対して、平均偏差は各値と平均値の差の絶対値の平均値です。

どちらもデータのばらつきを確認するのに使うことができます。

範囲は、データの最大値、最小値の間の距離です。

標準化

では、標準化について見ていきましょう。

以下のように打ち込んだファイルを作成し、実行してみましょう。

3.r

```
data1 <- read.csv("example.csv",header = FALSE)
data2 <- data1[,1]
```

```
mean_data <- mean(data2)
dev <- sqrt(mean((data2-mean_data)^2))
z_score <- (data2-mean_data)/dev

> dev
[1] 5.821512

> z_score
 [1] -0.1545990 -0.8417058 -1.1852591 -0.8417058  1.7349445  1.3913911
.....
```

dev はデータそのものの標準偏差、z_score はデータを「平均が 0、分散が 1 になうように」標準化したのもです。

標準化をするメリットを見ていきましょう。

以下の表を見てみてください。

	国語	数学	英語
A君	84	84	65
B君	75	39	54
C君	59	34	42
D君	45	69	81
E君	50	77	46

左の図は、5 人のとある試験での点数の表です。

この 5 人の中で、一番いい成績を取っている人は誰かを考えたいと思います。

この表における最高点は A 君の国語で 84 点です。

では、本当に A 君の国語が一番良い成績なのでしょうか。

	平均	標準偏差
国語	60.57	23.87855
数学	60.63	24.78413
英語	54	26.28726

左の図は、受験者全員の成績から計算した各教科の平均点と標準偏差です。

これを用いると、5 人以外のデータも使用した指標で 5 人の点数を見比べることができます。

標準化後	国語	数学	英語
A君	0.981	0.943	0.418
B君	0.604	-0.873	0
C君	-0.066	-0.107	-0.456
D君	-0.652	-0.338	1.027
E君	-0.443	0.066	-0.304

左の図は、標準化後の z スコアです。

z スコアは平均値や標準偏差などから算出される値で、これを見比べることでより詳しい比較が可能になります。

この結果、一番 z スコアが高いのは D 君の英語の為、一番いい成績は D 君の英語ということになります。

では、z スコアが一体何かを見ていきましょう。

zスコアとは、前述のとおり、「平均が 0、分散が 1 になうように」標準化した後の各値の事を指しま

す。

z スコアは、以下のようにして求められます。

$$z = \frac{x - \bar{x}}{s}$$

x:各値、$\bar{x}$:平均値、s:標準偏差

偏差値

偏差値についても見ていきましょう。

偏差値とは、「平均が 50、標準偏差が 10 になうように」標準化したものです。

先ほど作成したファイルに以下のように追記して実行してみましょう。

```
dev_50 <- 10*z_score+50


> dev_50
 [1] 48.45401 41.58294 38.14741 41.58294 67.34945 63.91391 53.60731
……
```

この、dev_50 が、偏差値です。

偏差値は、以下のように求められます。

$$SS = 10 \times z + 50$$

SS:偏差値、z:z スコア

このようにして求められた偏差値の平均は 50 に、標準偏差は 10 になっています。

6.3 最頻値、歪度、尖度

では、ここからは最頻値、歪度、尖度について見ていきます。

これらは全てヒストグラムにした時の山の頂点に関するものになっています。

最頻値

ではまず、最頻値について見ていきましょう。

最頻値は、データのうち、最も高い頻度で観測される値の事を指します。

以下のリンクから csv ファイルをダウンロードし、作業ディレクトリに移動させてください。

https://kimshun0213kr.github.io/R_tutorial_for_RB/sample/6/example2.csv

以下のように打ち込んだファイルを作成し、実行してみましょう。

4.r

```
data1 <- read.csv("example2.csv",header = TRUE,row.names = 1)
data_jap <- data1$jap


source("https://raw.githubusercontent.com/kimshun0213kr/R_tutorial_for
_RB/main/example_codes/ex5/source.r")
```

```
kekka <- table_percent(data_jap,5,1,100,1)

mode1 <- kekka[which.max(kekka[,3]),2]

> mode1
[1] 63
```

mode1 の値が、最頻値です。

mode1 <- kekka[which.max(kekka[,3]),2]の行以外は第 5 章で作成したファイルと読み込むファイル名以外同じです。

mode1 には、単に kekka というデータファイルの行と列を指定して値を代入しています。

行数は、階級値が格納されている 2 列目を指定しています。

列数は、少し複雑な方法で代入しています。

max()で、最大値を表示できるのは、先ほど見た通りです。

これを、whitch.max()とすると、最大値の行番号を表示させることができます。

これを用いて、行番号には、度数が格納されている 3 列目の最大値の行番号を代入しています。

この方法を応用すると、どのようなデータでも最頻値を求めることができるようになります。

歪度

次に、歪度(わいど)について見ていきましょう。

歪度とは、分布をヒストグラムにした時に「山の頂上が中心からどれくらいの位置にあるか」を数値化したものです。

歪度は、以下の式で求めることができます。

$$a_3 = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s}\right)^3$$

s:標準偏差、x_i:各データ、$\bar{x}$:平均

では、先ほどのコードに以下のように追記して実行してみましょう。

```
data2 <- data_jap-mean(data_jap)
data3 <- sum(data2^2)
data4 <- sqrt(data3/length(data_jap))

skew <- sum(((data2)/data4)^3)/length(data_jap)

> skew
[1] -0.1216414
```

skew に、歪度が代入されています。

歪度が正の値なら、山の頂点が中央より左に寄っていることを、歪度が 0 なら、左右対称の分布になっていることを、歪度が負の値なら山の頂点が右に寄っていることを指します。

ここでは、歪度が負の値であるため、分布が右に寄っていることが分かります。

尖度

では、尖度(せんど)を見ていきましょう。

尖度とは、分布のとがり具合を数値で表したものです。

尖度は、以下の式で求めることができます。

$$a_4 = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s}\right)^4 - 3$$

s:標準偏差、x_i:各データ、$\bar{x}$:平均

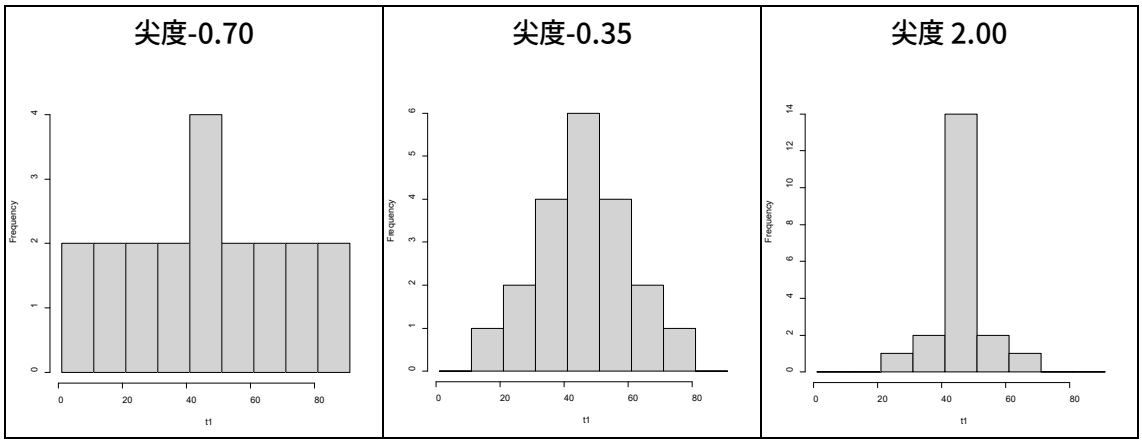
では、先ほどのコードに以下のように追記して実行してみましょう。

```
kurt <- sum(((data2)/data4)^4)/length(data_jap)-3


> kurt
[1] -0.9816638
```

kurt に尖度が代入されています。

尖度について、以下の図を見てみましょう。



山のとがり具合が鋭いほど、尖度が大きくなっていることが分かります。

正規分岐の尖度は 0 になり、これを中尖と言います。

6.4 章末問題

問題①

以下のリンクから csv ファイルをダウンロードし、現在の作業ディレクトリに移動させてください。

https://kimshun0213kr.github.io/R_tutorial_for_RB/sample/6/example3.csv

これに header はありません。

このファイルを正しく読み込み、1 列目のデータの標本分散、不偏分散、標準偏差を求めてください。

問題②

既にダウンロードしてある example2.csv の mat 列に格納されている数学の点数一覧を使用して、一人一人の数学の偏差値を求めなさい。

問題③

example2.csv の理科の得点(sci 列)の歪度、尖度をそれぞれ求めなさい。

7 Rと要約統計量

さて、統計に手を出してしまったら引くに引けません。
筆者も理解しきれているのか怪しい所です。
頑張りましょう。

7.1 要約統計量とは？

この章を始める前に、要約統計量について説明します。
要約統計量とは、標本の分布の特徴を数値に着目して記述し、要約する値の事です。
これらの値は、標本の分布の特徴を示します。
そのため、この値を見ると、標本全体を見なくても、偏りや分布の様子が分かります。

7.2 中心を示す統計量

では、中心を表す統計量のうち、今までに紹介していなかったものを紹介していきます。

幾何平均

ではまず、幾何平均についてです。
これは等比級数に従う観測値に適用します。これの対数は、それぞれの値の対数の平均に一致します。
免疫分野で使用します。
今まで使用していた平均値は、「算術平均」と呼びます。
幾何平均は、以下の式で求めることができます。

$$G = \sqrt[n]{x_1 \times x_2 \times x_3 \times \cdots \times x_n} = \sqrt[n]{\prod_{i=1}^{n} x_i}$$

n:データの個数

以下のように打ち込んだファイルを作成し、実行してみましょう。

1.r

```
func1 <- function(masterdata){
    tmp <- 1
    for(i in masterdata){
        tmp <- tmp*i
    }
    return(tmp)
}
```

```
data <- c(1:10)
g_mean <- func1(data)^(1/length(data))

> g_mean
[1] 4.528729
```

ここでは、関数を作成して、ベクトル内の数字を全てかけるという計算をしています。

g_mean が、幾何平均の値です。

ベクトル内の数字を全てかけたものを 1/length(data)乗することで、$\sqrt[n]{\ }$の計算をしています。

算術平均の値とは異なるので、注意が必要です。

調和平均

次に、調和平均について見ていきましょう。

これは、「各データの逆数の平均」の逆数です。速度などを扱うのに便利です。

調和平均は、以下の 2 通りの式で求めることができます。

$$H = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

$$\frac{n}{H} = \sum_{i=1}^{n} \frac{1}{x_i}$$

以下のように打ち込んだファイルを作成し、実行してみましょう。

2.r

```
data2 <- c(1:10)^-1
h_tmp <- sum(data2)/length(data2)
h_mean <- h_tmp^-1

> h_mean
[1] 3.414172
```

h_mean が、調和平均の値です。

データを-1 乗することで、データすべての逆数を出しています。

中心を示す統計量まとめ

中心を示す統計量をまとめます。

名称	説明	定義式
平均値	一般的に使う平均	$\bar{X} = \sum_{i=1}^{n} x_i / n$
中央値	データの中心を示す	$x_{(n+1)/2}$ (n が奇数) $(x_{n/2} + x_{(n/2)+1})/2$ (n が偶数)

最頻値	最も高い頻度で観測される値	—
幾何平均	データの総乗のデータ数乗根の値	$G = \sqrt[n]{x_1 \times x_2 \times x_3 \times \cdots \times x_n} = \sqrt[n]{\prod_{i=1}^{n} x_i}$
調和平均	「逆数の平均値」の逆数	$H = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$ $\frac{n}{H} = \sum_{i=1}^{n} \frac{1}{x_i}$

7.3 変動を示す統計量

では次に、変動を示す統計量のうち、今までに紹介していなかったものについて紹介していきます。

偏差平方和

ではまず、偏差平方和についてです。

偏差平方和は、名前の通り「偏差(平均からの各データのずれ)の平方(2 乗)の和」です。

これを使用することで、符号による影響のない偏差のばらつき具合を知ることができます。

偏差平方和は以下の式で求めることができます。

$$SS_x = (x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2 = \sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=1}^{n} {x_i}^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2$$

以下のように打ち込んだファイルを作成し、実行してみましょう。

3.r

```
data <- c(1:10)
me1 <- mean(data)

ss1 <- sum((data-me1)^2)
ss2 <- sum((data)^2)-((sum(data)^2)/length(data))

> ss1
[1] 82.5
> ss2
[1] 82.5
```

ss1、ss2 の値は、どちらも偏差平方和です。

ss1 は、各データと平均の差を 2 乗して全て足す方法で、ss2 は各データの 2 乗の合計から各データの

2 乗の合計をデータの長さで割ったものを引く方法でそれぞれ偏差平方和を求めています。

変動係数

次に、変動係数についてです。

変動係数は、不偏分散の平方根を標本平均で割った値で、データのばらつきを表す係数です。

これを用いることで、異なるデータの間でもデータのばらつきを比較することができます。

以下の式で求めることができます。

$$CV = \frac{\sqrt{V_x}}{\bar{x}}$$

以下のように打ち込んだファイルを作成し、実行してみましょう。

4.r

```
data1 <- c(seq(1,20,2))
data2 <- c(5:14)

me1 <- mean(data1)
me2 <- mean(data2)

v2_1 <- sum((data1-me1)^2)/(length(data1)-1)
v2_2 <- sum((data2-me2)^2)/(length(data2)-1)

cv_1 <- sqrt(v2_1)/me1
cv_2 <- sqrt(v2_2)/me2

> v2_1
[1] 36.66667
> v2_2
[1] 9.166667
> cv_1
[1] 0.6055301
> cv_2
[1] 0.3187
```

cv_1 に data1 の変動係数が、cv_2 に data2 の変動係数が代入されています。

途中で導出している v2_1、v2_2 はそれぞれ data1、data2 の不偏分散です。

このようにしてみると、cv_1 より cv_2 の値のほうが小さいため、data2 のほうがデータのばらつきが小さいということが分かります。

標準誤差

標準誤差について見ていきましょう。

標準誤差とは、母集団から摘出された標本から標本平均を求めたときに、標本英金の値が母集団の平均に対してどの程度ばらついているかを表す値です。

これは、母集団に含まれるデータの数が大きくなると小さくなります。

標準誤差は、以下の式で求めることができます。

$$SE = \sqrt{\frac{V_x}{n}}$$

以下のように打ち込んだファイルを作成し、実行してみましょう。

5.r

```
data1 <- c(seq(1,91,10))
data2 <- c(1:91)
me1 <- mean(data1)
me2 <- mean(data2)
v2_1 <- sum((data1-me1)^2)/(length(data1)-1)
v2_2 <- sum((data2-me2)^2)/(length(data2)-1)

se_1 <- sqrt(v2_1/length(data1))
se_2 <- sqrt(v2_2/length(data2))

> me1
[1] 46
> me2
[1] 46
> se_1
[1] 9.574271
> se_2
[1] 2.768875
```

se_1 が data1 の標準誤差、se_2 が data2 の標準誤差です。

このコードは、最後の 2 行以外、先ほど変動係数を求めたコードと変わりません。

data1、data2 のどちらも平均値は 46 で変わりありません。

しかしながら、data1 の標本数は 10、data2 の標本数は 91 と、圧倒的に data21 の標本数が多くなっています。

このため、標準誤差も、data1 の標準誤差 se_1 は 9.574271、data2 の標準誤差 se_2 は

2.768875 と、data2 の標準誤差のほうが小さくなっています。

変動を示す統計量まとめ

中心を示す統計量をまとめます。

名称	説明	定義式
偏差平方和	各データの平均からのずれの平方和	$SS_x = \sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i{}^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2$
標本分散	偏差平方和をデータ数で割った値 標本のみを考える	$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$
不偏分散	偏差平方和をデータ数-1で割った値 母集団の推測にも使える	$S^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$
標準偏差	不偏分散の平方根	$SD = \sqrt{V_x}$
変動係数	不偏分散の平方根(標準偏差)を標本の平均で割った値	$CV = \frac{\sqrt{V_x}}{\bar{x}}$
標準誤差	不偏分散をデータ数で割った値の平方根	$SE = \sqrt{\frac{V_x}{n}}$

7.4 章末問題

問題①

1~100 の連続した値を代入したベクトルを作成し、このベクトルの算術平均・幾何平均・調和平均をそれぞれ求めなさい。

問題②

以下のリンクから csv ファイルをダウンロードし、作業ディレクトリに移動させてください。

https://kimshun0213kr.github.io/R_tutorial_for_RB/sample/7/example.csv

この csv ファイルに、header はありません。

このファイルを正しく読み込み、1 列目のデータの偏差平方和、変動係数、標準誤差をそれぞれ求めなさい。

最後に

さて、なんやかんやノリで始めたこの資料も結構多めのボリュームになってしまいました。(とは言いつつも50 ページくらいしかないんですよ。驚き。)

こんなに熱中するとは私自身思いませんでした。

この資料が皆さんの R の助けになれば幸いです。

何か質問などあれば気軽に質問してください。

きむしゅん

1 章末問題の回答

A1.1 第二章

問題①

行列、配列

問題②

Q2.r

```
Vec_Q2 <- c(1:10)
```

問題③

Q3.r

```
Mat_Q3 <- matrix(c(1:24),nrow = 4 , ncol = 6 , byrow = FALSE)
```

問題④

Q4.r

```
Arr_Q4 <- array(1:36,c(3,4,3))
```

A1.2 第三章

問題①

Q1.r

```
Mat_Q1_1 <- matrix(c(1:9),ncol = 3,nrow = 3,byrow = FALSE)
Mat_Q1_2 <- Mat_Q1*3
```

問題②

Q2.r

```
Mat_Q2_1 <- matrix(c(1:9),ncol = 3,nrow = 3,byrow = TRUE)
Mat_Q2_2 <- matrix(c(10:18),ncol = 3,nrow = 3,byrow = FALSE)
Mat_Q2_3 <- Mat_Q2_1 %*% Mat_Q2_2
```

問題③

Q3.r

```
Mat_Q3_1 <- matrix(c(13:16),nrow = 2,ncol = 2,byrow = TRUE)
Mat_Q3_2 <- solve(Mat_Q3_1)
```

問題④

Q4.r

```
Mat_Q4_1 <- matrix(c(11:19),ncol = 3,nrow = 3,byrow = FALSE)
Mat_Q4_2 <- det(Mat_Q4_1)
```

A1.3 第四章

問題①

データファイル

問題②

Q2.r

```
Q2_csv <- read.csv("example.csv",header = TRUE,row.names = 1)
Q2_csv70 <- subset(Q2_csv,Q2_csv$mat>=70&Q2_csv$eng>=70)
```

問題③

Q3.r

```
Q3_txt <- read.table("Q3.txt",header = FALSE,sep = "\t",skip=12)
```

A1.4 第五章

問題①

Q1.r

```
data1 <- read.table("Q3.txt",header = FALSE,sep = "\t",skip=12)
data2 <- data1[,2]

hist(data2,breaks=seq(0,100,5),main="テスト結果",xlab="点数",ylab="人数")
```

問題②

Q2.r

```
data1 <- read.table("Q3.txt",header = FALSE,sep = "\t",skip=12)

summ <- summary(data1[,3])
```

問題③

Q3.r

```
Q3_txt <- read.table("Q3.txt",header = FALSE,sep = "\t",skip=12)
```

A1.5 第六章

問題①

Q1.r

```
data <- read.csv("example3.csv",header = FALSE)
data2 <- data[,1]

data_mean <- mean(data2)

dev1 <- data2 - data_mean
dev2 <- dev1^2
sum_dev <- sum(dev2)

data_length <- length(data2)

disp <- sum_dev/data_length
var1 <- var(data2)
sd1 <- sd(data2)
```

問題②

Q2.r

```
data1 <- read.csv("example2.csv",header = TRUE,row.names = 1)
data_mat <- data1$mat
```

```
mean_data <- mean(data_mat)
dev <- sqrt(mean((data_mat-mean_data)^2))
z_score <- (data_mat-mean_data)/dev

dev_50 <- 10*z_score+50
```

問題③

Q3.r

```
data1 <- read.csv("example2.csv",header = TRUE,row.names = 1)
data_sci <- data1$sci

data2 <- data_sci-mean(data_sci)
data3 <- sum(data2^2)
data4 <- sqrt(data3/length(data_sci))

skew <- sum(((data2)/data4)^3)/length(data_sci)
kurt <- sum(((data2)/data4)^4)/length(data_sci)-3
```

A1.3 第七章

問題①

Q1.r

```
G_mean <- function(masterdata){
    tmp <- 1
    for(i in masterdata){
        tmp <- tmp*i
    }
    return(tmp)
}

data <- c(1:100)

me <- mean(data)
g_me <- (G_mean(data))^(1/length(data))
data2 <- data^-1
h_me <- (sum(data2)/length(data))^-1
```

問題②

Q2.r

```
data <- read.csv("example.csv",header = FALSE)

data1 <- data[,1]

me <- mean(data1)

ss <- sum((data1-me)^2)
v2 <- sum((data1-me)^2)/(length(data1)-1)
cv <- sqrt(v2)/me
se <- sqrt(v2/length(data1))
```

2 Rと最小二乗法

付録ではありますが、一番使いそうなことを説明します。

A2.1 最小二乗法の振り返り

まず初めに、最小二乗法の振り返りをしましょう。

最小二乗法とは、誤差を伴う測定値の処理において、その誤差の二乗の和を最小にするようにし、最も確からしい関係式を求める方法です。

化学実験で使用したことを覚えている人もいるかもしれませんね。

A2.2 単回帰分析

単回帰分析とは、関数をデータに当てはめ、変数 y の変動を別の変数 x の変動によって説明するための手法です。

この時の変数 y を目的変数、それを予測するための変数 x を説明変数と言います。

単回帰分析は、この「説明変数」が 1 つの回帰モデルの事を言います。

説明関数が 1 つなので、関数は y=ax+b の形であらわすことができ、この形を仮定して目的変数を予測します。

A2.3 R での最小二乗法

では、R で最小二乗法を用いて測定値のグラフを作ってみましょう。

ここでは、以下のデータの検量線を、最小二乗法を用いて作成してみましょう。

濃度(%)	吸光度
0.002	0.11
0.004	0.22
0.006	0.32
0.008	0.42
0.010	0.51
0.012	0.61
0.014	0.69

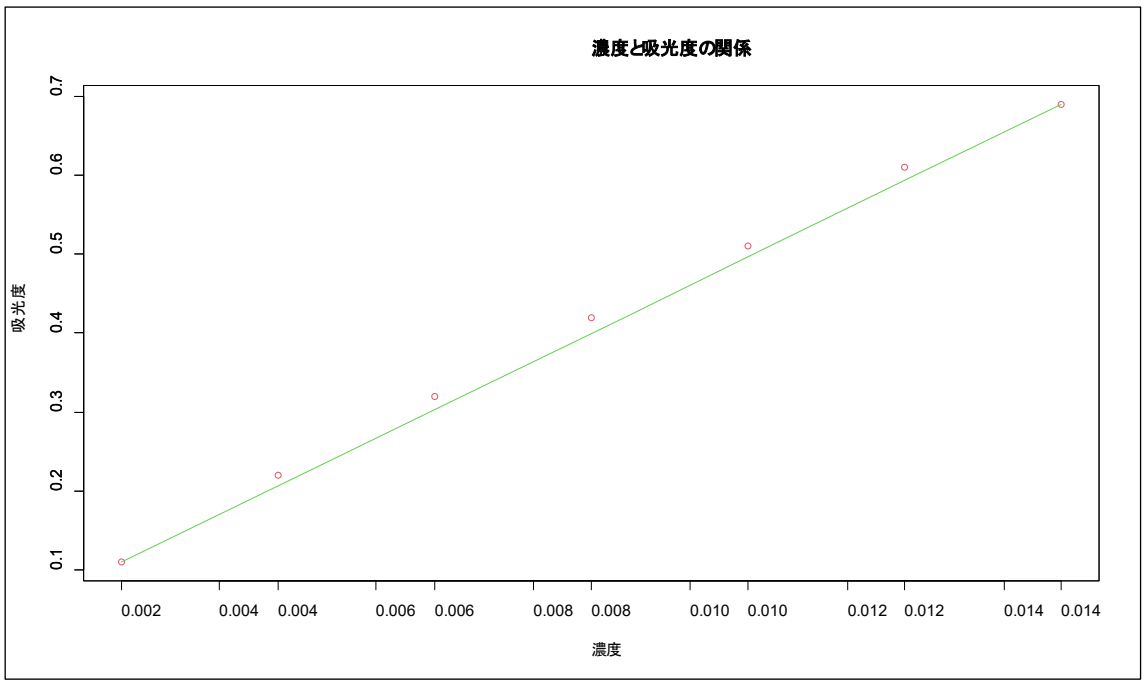
以下のように打ち込んだファイルを作成し、実行してみましょう。

1.r

```
y <- c(0.002,0.004,0.006,0.008,0.010,0.012,0.014)
x <- c(0.11,0.22,0.32,0.42,0.51,0.61,0.69)

ans <- lsfit(x,y)
con <- ans$coefficients
x_con <- x*con[2]-con[1]
plot(y,x,col=2,ylab="",xlab="")
par(new=T)
plot(x_con,x,type="l",col=3,main="濃度と吸光度の関係",ylab="吸光度
```

実行して表示されたグラフを見てみましょう。



このように、最小二乗法を使用した検量線を作成することができます。

y は濃度を、x は吸光度をそれぞれ代入したベクトルです。

lsfit を使用すると、最小二乗法を使用した分析の結果を調べることができます。

lsfit の内容を見てみましょう。

```
> ans
$coefficients
    Intercept             X
-0.0004890146  0.0206330215
```

```
$residuals
[1]  2.193822e-04 -5.025016e-05 -1.135523e-04 -1.768545e-04 -
3.382641e-05
[6] -9.712856e-05  2.522297e-04

$intercept
[1] TRUE

$qr
$qt
[1] -0.0211660105  0.0105750166 -0.0001721720 -0.0002532480 -
0.0001262164
[6] -0.0002072924  0.0001278468

$qr
      Intercept           X
[1,] -2.6457513 -1.08853768
[2,]  0.3779645  0.51252874
[3,]  0.3779645  0.01707057
[4,]  0.3779645 -0.17804043
[5,]  0.3779645 -0.35364034
[6,]  0.3779645 -0.54875135
[7,]  0.3779645 -0.70484015

$qraux
[1] 1.377964 1.212182

$rank
[1] 2

$pivot
[1] 1 2

$tol
[1] 1e-07

attr(,"class")
[1] "qr"
```

このうち、coefficients の列に格納されている intercept の値が y=ax+b の切片、X の値がグラフの傾きになっています。

この、coefficients の値だけを見てみましょう。

```
> con
    Intercept             X
-0.0004890146  0.0206330215
```

このようになっていることが分かります。

よって、このデータの検量線のグラフの式は、

$$y = 0.0206330215x - 0.0004890146$$

であることが分かります。

con[1]を参照することで intercept の値を、con[2]を参照することで X の値を使用することができます。

これを使用して、x_con に各データに X の値をかけ、intercept の値を引いたものを代入しています。

その後、測量値をプロットしたグラフを plot(x,y,col=2,ylab="",xlab="")の行で表示させています。

Plot(x,y)で、単純に測量値をプロットしたグラフを表示させることが可能ですが、col=2 とすることでプロットさせる点の色をオレンジ色に、ylab=""、xlab=""とすることで、x 軸、y 軸の表示をさせないようにしています。

また、par(new=T)とすることで、グラフを重ねての表示をさせるようにしています。

そして、最後に plot(x_con,x,type="l",col=3,main="濃度と吸光度の関係",ylab="吸光度",xlab="濃度")として、x_con と x の値を使用したグラフを表示させています。

Type="l"とすることで折れ線グラフを表示させ、col=3 とすることで、線の色を緑に指定しています。

また、main="濃度と吸光度の関係",ylab="吸光度",xlab="濃度"とすることで、グラフの題名を「濃度と吸光度の関係」に、y 軸の表示を「吸光度」に、x 軸の表示を「濃度」にしています。