日本医療検査科学会第52回大会
第2回医療情報技術セミナー
「知っておきたい臨床検査データ解析のテクニック」

2020-09-23_ver.5 (final)

臨床検査データ解析の実際

～Rを用いた解析の基本・応用・発展～

九州大学病院　検査部
瀬戸山大樹

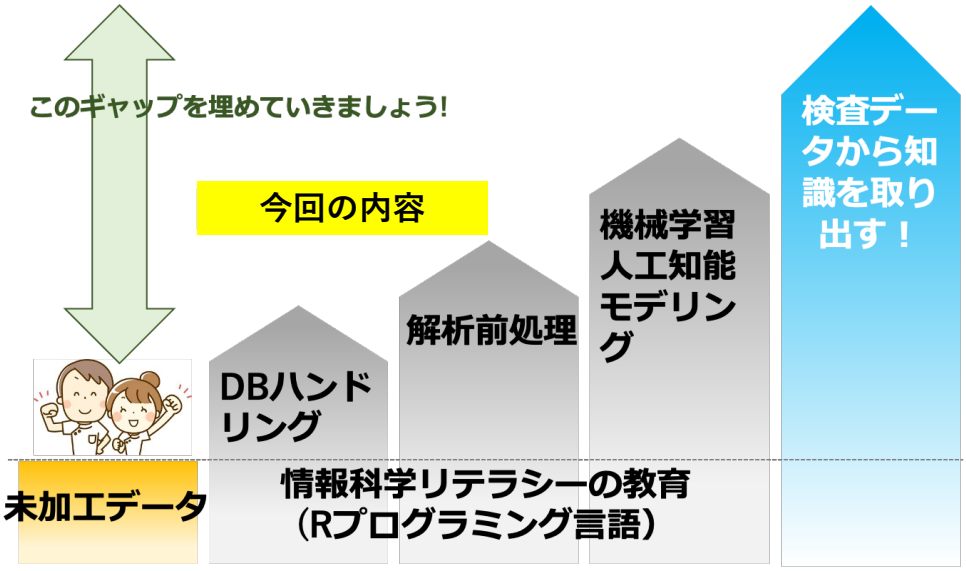
一般社団法人 日本医療検査科学会
The Japan Association for Clinical Laboratory Science

一般社団法人日本医療検査科学会
COI（利益相反）開示
筆頭発表者名：　瀬戸山 大樹
発表責任者名：　同上

演題発表に関連し，開示すべきCOI関係にある企業等はありません。

セミナーのねらい

- ✓ Rの基本操作に慣れる（コードを確認しながら進めます）
- ✓ リアルの病院検査データを眺めて特徴を捉える
- ✓ 便利なRパッケージの使い方に慣れる
- ✓ これなら自分でできるかもと、更に一歩踏み出す勇気を持つ



本セミナーの内容

セクション１：Rの基本操作（仮想臨床検査データ）

- ファイルの読み込み、書き出し
- 変数の扱い、挿入、条件抽出など

セクション２：Rの応用操作（データの連結と縦横変換）

- 列方向の連結
- 横方向の連結
- 縦持ちと横持ち変換

セクション３：Rの応用操作（リストとループ処理）

- リストの扱い
- forループ処理について

セクション４：病院データの解析前処理（リアル病院検査値データ）

- データの統合、縦持ち・横持ち変換
- 欠損値の少ない検査項目の選別

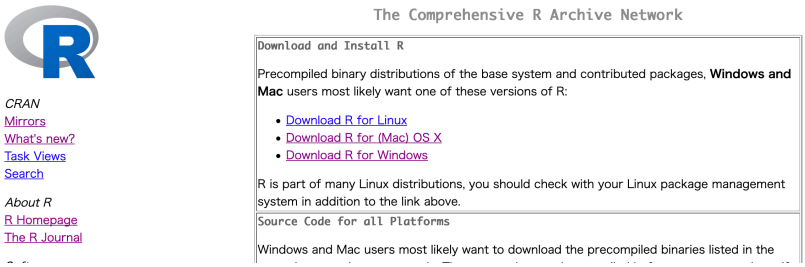
セクション５：臨床検査データの正規化（病院検査値+基準範囲データ）

セクション６：検診データの偏差値表示（検診+基準範囲データ）

必要な環境整備。。。

必ずインストールして下さい

1. Rのダウンロード(現在version **4.0.2** from 2020-06-22)
-> **https://cran.r-project.org/**



2. RStudioのダウンロード(現在version **1.3.1093** from 2020-09-18)
-> **https://www.rstudio.com/**

本セミナーで使用するRパッケージ

パッケージとはある機能に特化した関数の集合体のことです

現在**16,356**個のパッケージがCRANに登録されています(2020年9月23日)

- **readxl**: エクセルファイルを読み込むためのパッケージ(ver. **1.3.1**)
- **data.table**: 大規模データを高速で読み書きするパッケージ(ver. **1.13.0**)
- **tidyr**: データの縦持ち・横持ち形式を変換するパッケージ(ver. **1.1.2**)
- **dplyr**: 数多くの前処理を簡単に行えるパッケージ(ver**. 1.0.2**)
- **ggplot2**: データを見栄え良く可視化するためのパッケージ(ver. **3.3.2**)
- **scales**: グラフィックパラメータ調節のためのパッケージ(ver.**1.1.1**)

スライドでは、(パッケージ名)::(関数名)と表示します

例 readxlパッケージのread_excel関数を使用する場合
- readxl::read_excel()

Rパッケージのダウンロード

```
#インストールしたいパッケージ名を” ”で囲む
install.packages(”XXXXX")
#エクセルファイル読み込み用: readxl
#データ前処理用: data.table; tidyr; dplyr
#可視化ツール: ggplot2; scales
```

事前にダウンロードして下さい

```
#ダウンロードしたパッケージを実装する
library(readxl); library(data.table);
library(tidyr); library(ggplot2); library(scales)
```

libraryフォルダをUSBに入れ、別のPCのlibraryフォルダに置き換えることで、Rパッケージを共有することができます!

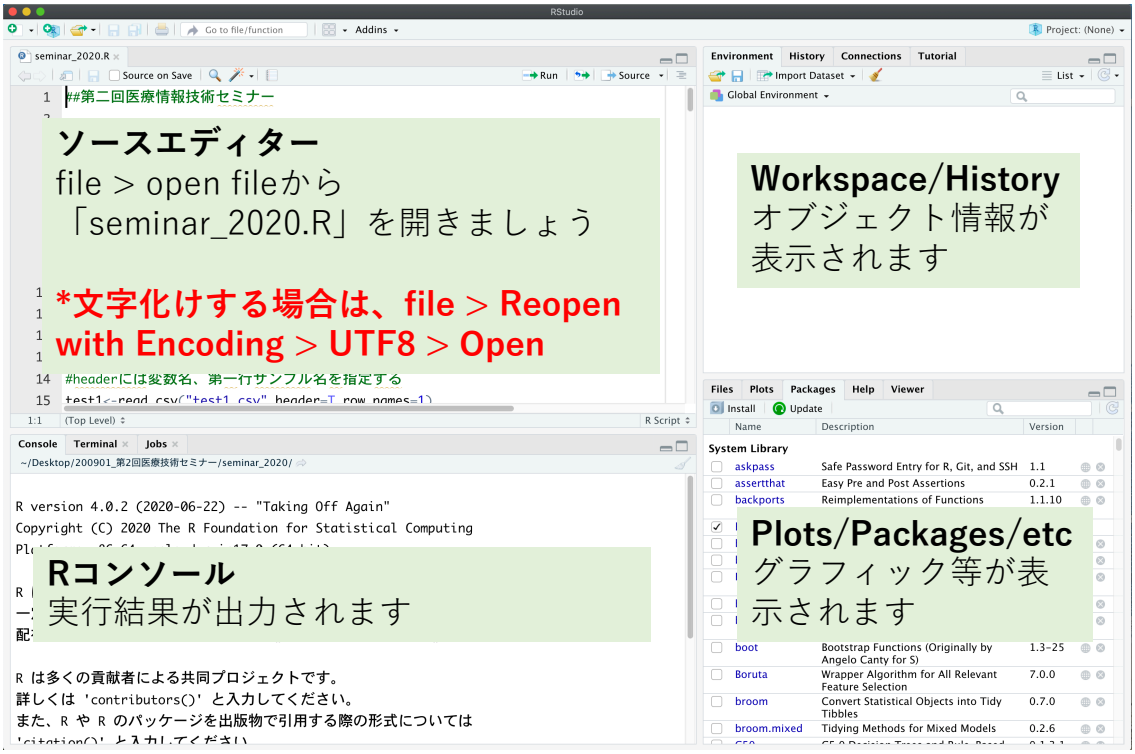
```
#tidyrパッケージファイルの場所を確認する
path.package(“tidyr”)
```

"/Library/Frameworks/R.framework/Versions/**4.0**/Resources/**library**/tidyr"

セクション1：Rの基本

- ファイルの読み込み、書き出し
- 変数の扱い、挿入、条件抽出など

RStudioの立ち上げ



作業ディレクトリを指定しよう

データの読み込み先や保存場所を予め指定しておきます

```
#現在のディレクトリはgetwd()で把握できます
#macとwindowsでは少し標記が異なります
#デスクトップ上に「seminar_2020」フォルダを置きます
setwd("/users/xxx/desktop/seminar_2020/koushukai/")

#windowsの場合
setwd("C:/users/xxx/desktop/seminar_2020/koushukai/")
```

セクション1,2,3で扱うデータセット

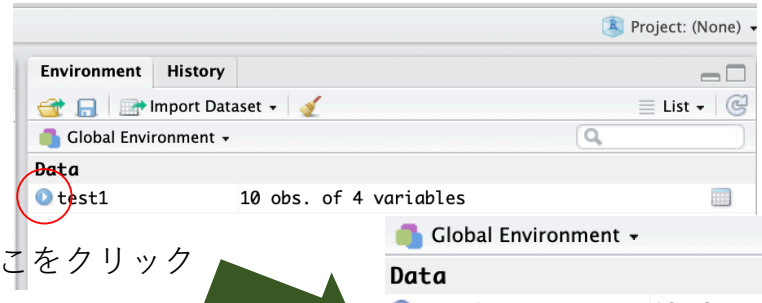
✓ **患者の検査値データ(test1.csv, test2.csv, test3.csv, test4.csv)**

- 患者のイニシャル、性別sex、各検査値情報を含む仮想データです
- これらのデータを使ってRの基本操作を学びます

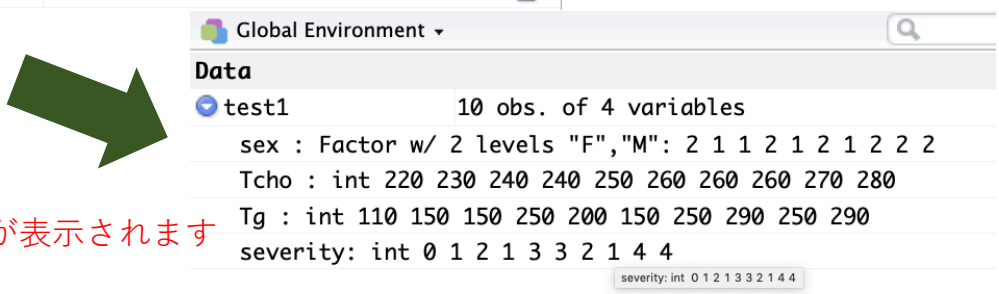
csvファイルの読み込み

```
#csvファイルの読み込み
#read.csv関数を使います(標準関数)
#第一行目の変数名をheaderとし、第一列目をサンプル名に指定します
test1 <- read.csv("test1.csv", header=TRUE, row.names=1)
```

Rstudioの右上のパネルの下記のように表示されます



ここをクリック

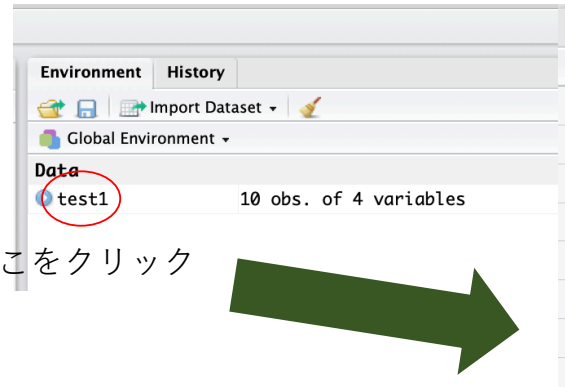


データの概要が表示されます

csvファイルの読み込み

```
#csvファイルの読み込み
#headerには変数名、第一行サンプル名を指定する
#read.csv関数を使います(標準関数)
test1<-read.csv("test1.csv", header=TRUE, row.names=1)
```

Rstudioの右上のパネルの下記のように表示されます



ここをクリック

左上のパネルに
データが表示されます

190918_実習用スクリプト.R　test1

Filter

	sex	Tcho	Tg	severity
DS	M	220	110	0
TH	F	230	150	1
MS	F	240	150	2
MA	M	240	250	1
TN	F	250	200	3
KG	M	260	150	3
YI	F	260	250	2
KI	M	260	290	1
KD	M	270	250	4
YA	M	280	290	4

データの表示

```
#データを眺めてみましょう
test1<-read.csv("test1.csv", header=TRUE, row.names=1)
test1
```

変数

変数名に下記の禁則文字は使わないこと
" $ % ^ * + - ( ) [ ] # ! ? < >

名前はユニーク(一意)でなければいけません

```
   sex Tcho  Tg severity
DS   M  220 110        0
TH   F  230 150        1
MS   F  240 150        2
MA   M  240 250        1
TN   F  250 200        3
KG   M  260 150        3
YI   F  260 250        2
KI   M  260 290        1
KD   M  270 250        4
YA   M  280 290        4
```

観測値
サンプル名

データの構造を把握する

```
#str関数（標準関数）を使って構造をみます
str(test1)
```

```
'data.frame':   10 obs. of  4 variables:
 $ sex     : Factor w/ 2 levels "F","M": 2 1 1 2 1 2 1 2 2 2
 $ Tcho    : int  220 230 240 240 250 260 260 260 270 280
 $ Tg      : int  110 150 150 250 200 150 250 290 250 290
 $ severity: int  0 1 2 1 3 3 2 1 4 4
```

- test1オブジェクトはデータフレームです
- 10個の観測値(observation)と4つの変数(variables)を含みます
- 変数sexは因子型(factor)でFとMの２つの水準を含みます
- 変数Tcho, Tg, severityは整数型(integer)です

- **データフレーム(data frame)は、複数の変数型を持つ2次元配列です**
- **行列(matrix)は、数値型のみの変数を持つ2次元配列です**

データ型(class)について

データの種類	型(class)	変数	備考
スカラ	scalar	数値のみ	
ベクター	vector	単一の型	
行列	matrix	数値のみ	2次元配列
データフレーム	data.frame	複数種	2次元配列
リスト	list	複数（ベクター、行列、データフレーム）	セクション3で説明します
	tibble	複数(上記に加え、リストも可)	tidyverseパッケージで扱うデータ型の一種

```
#class(data)で確認できます
class(test1)
```

```
> class(test1)
[1] "data.frame"
```

変数の「型(mode)」について

```
'data.frame':   10 obs. of  4 variables:
 $ sex     : Factor w/ 2 levels "F","M": 2 1 1 2 1 2 1 2 2 2
 $ Tcho    : int  220 230 240 240 250 260 260 260 270 280
 $ Tg      : int  110 150 150 250 200 150 250 290 250 290
 $ severity: int  0 1 2 1 3 3 2 1 4 4
```

変数の種類	型(mode)	サブタイプ(type)	備考
カテゴリカル	factor		
数値型	numeric	integer	整数値
		double	実数値
文字型	character		
日付型	date		
論理型	boolean		TRUE/FALSE

データの部分表示

```
#head関数（標準関数）を使って上位5行のみを表示させましょう
#デフォルトは上位6行です
head(test1, 5)
```

```
> head(test1,5)
   sex Tcho  Tg severity
DS   M  220 110        0
TH   F  230 150        1
MS   F  240 150        2
MA   M  240 250        1
TN   F  250 200        3
```

```
#tail関数（標準関数）を使って下位5行のみを表示させましょう
#デフォルトは上位6行です
tail(test1, 5)
```

変数の表示法

```
#names関数（標準関数）を使って変数名を表示させます
names(test1)
```

```
> names(test1)
[1] "sex"      "Tcho"     "Tg"       "severity"
```


変数のアクセス法

```
#中性脂肪Tgを取り出す方法
#$を使用する方法 *「$」は「～の」の意味だと思ってください
test1$Tg
#[行, 列]を使用する方法
#指定無しの場合は「すべてを取り出す」の意味です
test1[ , 3]
#直接名前を記入する方法
test1[ ,"Tg"]
```

“$”や”[“は演算機能を有する記号=演算子（オペレーター）といいます

```
> test1$Tg
 [1] 110 150 150 250 200 150 250 290 250 290
> test1[,3]
 [1] 110 150 150 250 200 150 250 290 250 290
> test1[,"Tg"]
 [1] 110 150 150 250 200 150 250 290 250 290
```

括弧の使い分け

◆ 丸括弧（parenthesis, “()“）と角括弧(square blankets, “[]”)の違い

✓関数の後は丸括弧
✓オブジェクトの後は角括弧

例1

head (data1, 10)

関数　対象データ　条件

引数、パラメータ

data1に対してhead関数を適用し、上位10行を表示させなさい

例2

data1[3, 2]

対象データ 行 列

data1の3行2列目を表示させなさい

複数の変数を取り出す方法　その1

```
#総コレステロールTcho、中性脂肪Tg、重症度severityの変数を取り出す
#第2列から第4列までを連続して取り出す
#「：」は連続値を取り出す演算子です
test1[ , 2:4]
```

```
> test1[,2:4]
   Tcho  Tg severity
DS  220 110        0
TH  230 150        1
MS  240 150        2
MA  240 250        1
TN  250 200        3
KG  260 150        3
YI  260 250        2
KI  260 290        1
KD  270 250        4
YA  280 290        4
```

複数の変数を取り出す方法　その２

```
#飛び飛びの変数はどうでしょうか？Tchoとseverityを抽出します
#「c」はcombine（結合する）関数です
test1[ , c("Tcho", "severity")]
#列数指定でもできます
test1[ , c(2,4)]
```

結果はどちらも一緒です

```
> test1[,c("Tcho","severity")]
   Tcho severity
DS  220        0
TH  230        1
MS  240        2
MA  240        1
TN  250        3
KG  260        3
YI  260        2
KI  260        1
KD  270        4
YA  280        4
```

```
> test1[,c(2,4)]
   Tcho severity
DS  220        0
TH  230        1
MS  240        2
MA  240        1
TN  250        3
KG  260        3
YI  260        2
KI  260        1
KD  270        4
YA  280        4
```


条件にあうデータ（行）を抽出する方法

```
#severityが3の患者だけを抽出する場合
#真偽式を利用します
test1$severity==3
```

```
> test1$severity==3
 [1] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

```
#真偽式がTRUEの行（患者）を抽出します
test1[test1$severity==3, ]
```

```
> test1[test1$severity==3,]
   sex Tcho  Tg severity
TN   F  250 200        3
KG   M  260 150        3
```

```
#dplyrパッケージのfilter()関数を使うと簡潔にコードできます
library(dplyr) #dplyrの実装
dplyr::filter(test1, severity==3)
```

条件抽出で使用するオペレーター(演算子)

Operator	Definition	意味
&	And	かつ
|	Or	あるいは
==	Equal to	等しい
!	Not	~ではない
!=	Not equal to	等しくない
>	Bigger than	より大きい
>=	Equal to or bigger than	等しい、もしくは、より大きい

新たな変数をデータフレームに挿入する方法

```
#新しい変数sumを作りましょう (sum=Tcho+Tg)
sum <- test1$Tcho+test1$Tg
sum
```

```
> sum<-test1$Tcho+test1$Tg
> sum
 [1] 330 380 390 490 450 410 510 550 520 570
```

```
#test1にsumを加えたデータフレームtest2を作りましょう
#まずtest1をtest2にコピーします
test2<-test1
#test2にtest1がコピーされているか確認しましょう
Test2

#dplyrパッケージのmutate()関数を使うと簡潔にコードできます
test2<-dplyr::mutate(test1, sum=sum)
```

新たな変数をデータフレームに挿入する方法

```
#次に、test2に新しい変数sumを作って、sumを代入しましょう
test2$sum<-sum
#test2を確認しましょう
test2
```

```
> test2<-test1
> test2
   sex Tcho  Tg severity
DS   M  220 110        0
TH   F  230 150        1
MS   F  240 150        2
MA   M  240 250        1
TN   F  250 200        3
KG   M  260 150        3
YI   F  260 250        2
KI   M  260 290        1
KD   M  270 250        4
YA   M  280 290        4
```

```
> test2$sum<-sum
> test2
   sex Tcho  Tg severity sum
DS   M  220 110        0 330
TH   F  230 150        1 380
MS   F  240 150        2 390
MA   M  240 250        1 490
TN   F  250 200        3 450
KG   M  260 150        3 410
YI   F  260 250        2 510
KI   M  260 290        1 550
KD   M  270 250        4 520
YA   M  280 290        4 570
```

新たな変数

データフレームから特定の変数を除去する方法

```
#test2のseverity(4列目)を除きましょう
#いくつかのやり方があります
test2[ ,-4]  #4列目を除去するという意味です
test2[ ,c(1,2,3,5)]  #4列目以外を選択するという意味です

#いずれも列数がわからない場合や変数が多い場合は使いにくいですね
#先程のオペレーターを使ってみましょう
#否定演算子「！」を使うと下記のようになります
test2[ ,!names(test2)==“severity”]

#dplyrパッケージのselect()関数を使うと簡潔にコードできます
dplyr::select(test2, -severity)
```

条件に合ったラベルを挿入する(ifelse文)

```
#ある変数に条件をつけて、複数のカテゴリーに分類しましょう
#ifelse関数を使い、ifelse(条件式, 真, 偽）のように使用します
#「grade」という新しい変数を作り、Tgが160未満であれば0を、160以上であれば1をつけましょう
test2$grade<-ifelse(test2$Tg<160, 0, 1)
```

```
#より複雑な条件分岐
#Tgが160未満であれば0を、200未満であれば1を、それ以上であれば2をつけましょう
test2$grade<-ifelse(test2$Tg<160, 0,
                    ifelse(test2$Tg<200, 1, 2))

#dplyrパッケージのmutate()関数を使うと簡潔にコードできます
test2<-dplyr::mutate(test2, grade=ifelse(Tg<160, 0,
                                         ifelse(Tg<200, 1, 2)))
```

データフレームをcsvファイルに保存する

```
#データフレームを作業ディレクトリに保存しよう
#write.csv関数(標準関数）を使います
write.csv(test2, file="test1_modified.csv")
```

セクション1のまとめ

Rの基本操作

- csvファイルの読み込み、書き出しができる
- データの型(クラス)、変数の型(モード)を理解した
- データの構造をつかむことができる
- データフレームの特定の変数を抽出できる
- データフレームに新たな変数を挿入できる
- データフレームの特定の変数を除去できる
- 条件分岐により、観察値にラベルを付与することができる

dplyrパッケージの関数	機能
filter()	特定の観測値（行）を抽出する
select()	特定の変数（列）抽出する
mutate()	新たな変数（列）を追加する

セクション2：Rの応用操作
データの連結と縦横変換

- 縦方向の連結
- 横方向の連結
- 縦持ち横持ち変換

縦方向（行）の連結: dplyr::bind_rows()

```
#このセクション2では複数のデータを連結する操作について学びます
#主にdplyrパッケージの関数を使用します
#SQLに精通している方にとってはおなじみの操作ですね
#test2.csvファイルを読み込みましょう
#test1とほぼ同じフォーマットですが、別の患者のデータが含まれています
test3<-read.csv("test2.csv", header=T, row.names=1)
```

```
> test3
    sex Tcho  Tg
KK    M  230 110
KN    F  210 150
KT    F  220 150
MS    M  240 270
KS    F  250 200
TN    M  260 150
AY    F  210 230
JW    F  260 290
NS    M  280 270
TK    M  300 280
```

```
> test1
    sex Tcho  Tg severity
DS    M  220 110        0
TH    F  230 150        1
MS    F  240 150        2
MA    M  240 250        1
TN    F  250 200        3
KG    M  260 150        3
YI    F  260 250        2
KI    M  260 290        1
KD    M  270 250        4
YA    M  280 290        4
```


縦方向（行）の連結: dplyr::bind_rows()

```
#データを縦方向に連結する場合はdplyr::bind_rows()関数を使います
test13<-dplyr::bind_rows(test1, test3)
test13
```


```
> test13
         sex Tcho  Tg severity
DS         M  220 110        0
TH         F  230 150        1
MS...3     F  240 150        2
MA         M  240 250        1
TN...5     F  250 200        3
KG         M  260 150        3
YI         F  260 250        2
KI         M  260 290        1
KD         M  270 250        4
YA         M  280 290        4
KK         M  230 110       NA
KN         F  210 150       NA
KT         F  220 150       NA
MS...14    M  240 270       NA
KS         F  250 200       NA
TN...16    M  260 150       NA
AY         F  210 230       NA
JW         F  260 290       NA
NS         M  280 270       NA
TK         M  300 280       NA
```

test1

test2

連結するデータの変数が一部異なる場合は、新たな変数が追加され、値を持たない部分はNA（欠損値）表示されます

横方向（列）の連結

```
#次に横方向の連結について学びます
#まずtest3.csvファイルを読み込みましょう
test4<- read.csv("test3.csv", header=T, row.names=1)

#test1と比べてみましょう
test1
test4
```

```
> test1
   sex Tcho  Tg severity
DS   M  220 110        0
TH   F  230 150        1
MS   F  240 150        2
MA   M  240 250        1
TN   F  250 200        3
KG   M  260 150        3
YI   F  260 250        2
KI   M  260 290        1
KD   M  270 250        4
YA   M  280 290        4
```

```
> test4
    AST ALT GGT
DS   25  22  25
TH   26  21  25
MS   22  22  39
MA   27  23  30
TN   33  40  60
KG   29  22  32
YI   30  26  40
KI   29  25  41
```

test4には、test1の一部の患者の別の検査データが含まれています

横方向（列）の連結を紐付けるプライマリーキー

#2つのデータを横方向に連結させるとき、お互いを紐付ける変数（プライマリーキーまたは主キーという）を指定します
#プライマリーキーは必ず"一意（ユニークである）"でなければいけません

df.x

ID	age	sex	ast	alt
A				
B				
C				
D				

×

df.y

ID	alb	HbA1c	glucose
A			
B			
C			
D			

「ID」をプライマリーキーとした連結

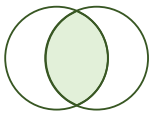
df.z

ID	age	sex	ast	alt	alb	HbA1c	glucose
A							
B							
C							
D							

横方向（列）の連結: dplyr::join ()関数群

#join()関数には様々なバリエーションがあります
#プライマリーキーはby=“ ”で特定の変数を指定して連結します

(1) inner_join(a, b, by=“x1”)



a

x1	x2
A	1
B	2
C	3
D	4

＋

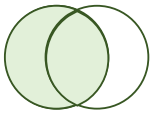
b

x1	x3
A	5
B	6
E	7
F	8

＝

x1	x2	x3
A	1	5
B	2	6

(2) left_join(a, b, by=“x1”)



x1	x2
A	1
B	2
C	3
D	4

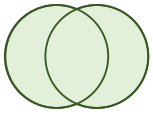
＋

x1	x3
A	5
B	6
E	7
F	8

＝

x1	x2	x3
A	1	5
B	2	6
C	3	NA
D	4	NA

(3) full_join(a, b, by=“x1”)



x1	x2
A	1
B	2
C	3
D	4

＋

x1	x3
A	5
B	6
E	7
F	8

＝

x1	x2	x3
A	1	5
B	2	6
C	3	NA
D	4	NA
E	NA	7
F	NA	8

実際の連結

#test1とtest4を横に連結させます
#しかし、今のままではイニシャルがサンプル名として読み込まれているため、プライマリーキーとして使うことができません
#改めてデータを読み込む必要があります
test1<-read.csv(“test1.csv”, header=T)
test4<-read.csv(“test3.csv”, header=T)

「row.names=1」のオプションを外し、IDを変数として読み込ませました

test14.inner<-dplyr::inner_join(test1, test4, by=“ID”)
test14.inner

```
> test14.inner
  ID sex Tcho  Tg severity AST ALT GGT
1 DS   M  220 110        0  25  22  25
2 TH   F  230 150        1  26  21  25
3 MS   F  240 150        2  22  22  39
4 MA   M  240 250        1  27  23  30
5 TN   F  250 200        3  33  40  60
6 KG   M  260 150        3  29  22  32
7 YI   F  260 250        2  30  26  40
8 KI   M  260 290        1  29  25  41
```

test1とtest4で共通する患者のみの検査データを連結しました

複合プライマリーキーについて

#プライマリーキーは1つの変数である必要はありません
#複数の変数の組み合わせが「一意（ユニークである）」であれば、それを「複合プライマリーキー」として使い、データを紐付ける場合が多いです

#test5にtest4.csvを読み込み、test1と比べてみましょう
test5<-read.csv("test4.csv", header=T)

```
> test5
  ID sex AST ALT GGT
1 DS   M  25  22  25
2 TH   F  26  21  25
3 MS   F  22  22  39
4 MA   M  27  23  30
5 TN   F  33  40  60
6 MA   F  29  22  32
7 YI   F  30  26  40
8 KI   M  29  25  41
9 MS   M  24  22  65
```

```
> test1
  ID sex Tcho  Tg severity
1 DS   M  220 110        0
2 TH   F  230 150        1
3 MS   F  240 150        2
4 MA   M  240 250        1
5 TN   F  250 200        3
6 KG   M  260 150        3
7 YI   F  260 250        2
8 KI   M  260 290        1
9 KD   M  270 250        4
10 YA  M  280 290        4
```

test5ではMSさんとMAさんが重複していますので、IDをプライマリーキーとして使うことはできません

IDと性別(sex)を組み合わせるとユニークになります

複合プライマリーキーを使った連結

#プライマリーキーをIDに指定した場合とID+sexを指定した場合で比較してみましょう
test15<-dplyr::full_join(test1, test5, by="ID")
test15.unique<-dplyr::full_join(test1, test5, by=c("ID","sex"))

```
> test15
   ID sex.x Tcho  Tg severity sex.y AST ALT GGT
1  DS     M  220 110        0     M  25  22  25
2  TH     F  230 150        1     F  26  21  25
3  MS     F  240 150        2     F  22  22  39
4  MS     F  240 150        2     M  24  22  65
5  MA     M  240 250        1     M  27  23  30
6  MA     M  240 250        1     F  29  22  32
7  TN     F  250 200        3     F  33  40  60
8  KG     M  260 150        3  <NA>  NA  NA  NA
9  YI     F  260 250        2     F  30  26  40
10 KI     M  260 290        1     M  29  25  41
11 KD     M  270 250        4  <NA>  NA  NA  NA
12 YA     M  280 290        4  <NA>  NA  NA  NA
```

```
> test15.unique
   ID sex Tcho  Tg severity AST ALT GGT
1  DS   M  220 110        0  25  22  25
2  TH   F  230 150        1  26  21  25
3  MS   F  240 150        2  22  22  39
4  MA   M  240 250        1  27  23  30
5  TN   F  250 200        3  33  40  60
6  KG   M  260 150        3  NA  NA  NA
7  YI   F  260 250        2  30  26  40
8  KI   M  260 290        1  29  25  41
9  KD   M  270 250        4  NA  NA  NA
10 YA   M  280 290        4  NA  NA  NA
11 MA   F   NA  NA       NA  29  22  32
12 MS   M   NA  NA       NA  24  22  65
```

test15ではMSさんとMAさんの連結がおかしいです
test15.uniqueでは正しく連結されています

データの縦持ちと横持ちについて

#ここではデータの表示形式について学びます
#test1のデータ表示方法には2通りあります

◆ 1行に1つの検査値
◆ 縦持ち, 行持ち, long-formという

ID	sex	item	score
DS	M	Tcho	220
TH	F	Tcho	230
MS	F	Tcho	240
MA	M	Tcho	240
TN	F	Tcho	250
KG	M	Tcho	260
YI	F	Tcho	260
KI	M	Tcho	260
DS	M	Tg	110
TH	F	Tg	150

データベース管理にはこちらの方が便利です（保存用）

同一情報

◆ 1列に1つの検査項目
◆ 横持ち, 列持ち, wide-formという

item

ID	sex	Tcho	Tg
DS	M	220	110
TH	F	230	150
MS	F	240	150
MA	M	240	250
TN	F	250	200
KG	M	260	150
YI	F	260	250
KI	M	260	290

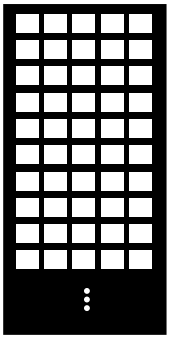
score

データ解析にはこちらの方が便利です（解析用）

tidyrのspread()関数とgather()関数

#縦持ちを横持ちに変換する時はtidyr::spread()関数
#横持ちを縦持ちに変換する時はtidyr::gather()関数を使います

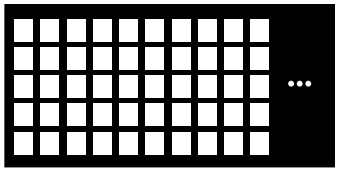
検査値ごとの縦持ちデータ



tidyr::spread

tidyr::gather

患者ごとの横持ちデータ



```
spread(data, key="item", value="score ")
gather(data, key="item", value="score", Tcho, Tg)
```

値を"score"に格納する

変数名を"item"に格納する

実際の縦持ち変換

```
#test1のデータは横持ち形式です
#test1を縦持ちに変換しましょう
#test1のIDは変数名として読み込んでおきます
test1<-read.csv("test1.csv", header=T)
library(tidyr) # tidyrパッケージを実装します
test1.long<- tidyr::gather(test1, key="item", value="score",
                                                    Tcho, Tg, severity)
str(test1.long)
test1.long
```

```
> str(test1.long)
'data.frame':   30 obs. of  4 variables:
 $ ID   : chr  "DS" "TH" "MS" "MA" ...
 $ sex  : chr  "M" "F" "F" "M" ...
 $ item : chr  "Tcho" "Tcho" "Tcho" "Tcho" ...
 $ score: int  220 230 240 240 250 260 260 260 270 280
> test1.long
   ID sex     item score
1  DS   M     Tcho   220
2  TH   F     Tcho   230
3  MS   F     Tcho   240
4  MA   M     Tcho   240
```


実際の横持ち変換

```
#今度は逆にtest1.longを横持ち変換しましょう
test1.wide<-tidyr::spread(test1.long, key="item", value="score")
test1.wide
```

```
> test1.wide
   ID sex severity Tcho  Tg
1  DS   M        0  220 110
2  KD   M        4  270 250
3  KG   M        3  260 150
4  KI   M        1  260 290
5  MA   M        1  240 250
6  MS   F        2  240 150
7  TH   F        1  230 150
8  TN   F        3  250 200
9  YA   M        4  280 290
10 YI   F        2  260 250
```

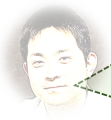
元のtest1に戻りました

解析の目的に応じて、横持ち形式と縦持ち形式を使い分けて、作業の効率を上げましょう

セクション2のまとめ

Rの応用操作　データの連結と縦横変換

- dplyrパッケージのbind_rows()関数を使って2つのデータを縦方向に連結できる
- 2つのデータを横方向に連結するためには一意変数であるプライマリーキーで紐付けることを理解した
- dplyrパッケージには様々なjoin()関数があり、目的に応じて使い分けることを理解した
- 複数の変数を組み合わた複合プライマリーキーを使って2つのデータを連結できる
- tidyrパッケージのgather()関数とspread()関数を使ってデータの縦持ち・横持ち変換ができる



データ解析において、飛車角級の働きをする重要なパッケージです。ぜひマスターしましょう！

セクション3：Rの応用

- リストの扱い
- forループ処理について

リストについて

リストは異なるデータを複数格納できるデータ型(class)のことです
行列(matrix)、データフレーム、ベクトルなど何でも入れ込めます

```
#データフレームtest1とtest2をリストに代入しましょう
#list関数（標準関数)を使います
test.list<-list(test1,test2)
test.list
```

```
> test.list
[[1]]
   sex Tcho  Tg severity
DS   M  220 110        0
TH   F  230 150        1
MS   F  240 150        2
```

test.listリストの要素1にtest1データフレーム、要素2にtest2データフレームが格納されています

```
[[2]]
   sex Tcho  Tg severity sum
DS   M  220 110        0 330
TH   F  230 150        1 380
MS   F  240 150        2 390
MA   M  240 250        1 490
```


リストの要素（の要素）を取り出す

```
#リストの要素は二重角括弧[[]]を使って取り出します
#リストの要素の要素は[[]][]を使って取り出します
#test2の3-5列目Tcho, severity, sumを取り出しましょう
test.list[[2]][,3:5]
```

```
> test.list[[2]][,3:5]
    Tg severity sum
DS 110        0 330
TH 150        1 380
MS 150        2 390
MA 250        1 490
TN 200        3 450
KG 150        3 410
YI 250        2 510
KI 290        1 550
KD 250        4 520
YA 290        4 570
```

基本統計量を表示する

#summary関数（標準関数）を使ってデータの基本統計量を表示させます
summary(test1)

```
sex        Tcho           Tg         severity
F:4   Min.   :220   Min.   :110   Min.   :0.0
M:6   1st Qu.:240   1st Qu.:150   1st Qu.:1.0
      Median :255   Median :225   Median :2.0
      Mean   :251   Mean   :209   Mean   :2.1
      3rd Qu.:260   3rd Qu.:250   3rd Qu.:3.0
      Max.   :280   Max.   :290   Max.   :4.0
```

表記	意味
Min.	最小値
1st Qu.	第一四分位
Median	中央値
Mean	平均値
3rd Qu.	第三四分位
Max.	最大値

lapply関数を使ったリストの操作

#lapply関数（標準関数）を使って、リストの要素を一括処理しましょう
#リストの要素の基本統計量をsummary関数を使って算出します
lapply(test.list, summary)

```
> lapply(test.list,summary)
[[1]]
 sex        Tcho           Tg         severity
 F:4   Min.   :220   Min.   :110   Min.   :0.0
 M:6   1st Qu.:240   1st Qu.:150   1st Qu.:1.0
       Median :255   Median :225   Median :2.0
       Mean   :251   Mean   :209   Mean   :2.1
       3rd Qu.:260   3rd Qu.:250   3rd Qu.:3.0
       Max.   :280   Max.   :290   Max.   :4.0

[[2]]
 sex        Tcho           Tg         severity        sum
 F:4   Min.   :220   Min.   :110   Min.   :0.0   Min.   :330.0
 M:6   1st Qu.:240   1st Qu.:150   1st Qu.:1.0   1st Qu.:395.0
       Median :255   Median :225   Median :2.0   Median :470.0
       Mean   :251   Mean   :209   Mean   :2.1   Mean   :460.0
       3rd Qu.:260   3rd Qu.:250   3rd Qu.:3.0   3rd Qu.:517.5
       Max.   :280   Max.   :290   Max.   :4.0   Max.   :570.0
```

#次に、リストの要素の要素(Tchoとseverity)を取り出しましょう
lapply(test.list, "[", c(2,4))

各要素から同じ要素が取り出されていますね
この操作は、**フォーマットが共通している要素に対して一括処理する際に絶大な威力を発揮します（詳細はセクション4; スライド61）**

forループについて

ここではforループについて学びます。
ループは、他のいろいろなプログラミング言語でも見られる概念で、繰り返し（反復）処理のことです。しかし実際、Rではforループは計算速度が遅いなどの理由から敬遠される傾向にあり、代わりに非常に高速に動く関数群が用意されています。しかし、どうしてもループを使わざるを得ない場合があり、その時には絶大な威力を発揮します。後のセクション5,6ではそのような事例を取り上げます。

```
#まず、colSums関数（標準関数）を使って、test2の一部の要素を集計しましょう
#colSums関数はcol(列)に対してSums(合計）値を算出する関数です
result1<-colSums(test2[,2:4])
```

```
> result1<-colSums(test2[,2:4])
> result1
    Tcho       Tg severity
    2510     2090       21
```


colSums関数の処理を解読する

```
result1<-colSums(test2[,2:4])
```

```
> test2[,2:4]
   Tcho  Tg severity
DS  220 110        0
TH  230 150        1
MS  240 150        2
MA  240 250        1
TN  250 [illegible] 3
KG  260 150        3
YI  260 250        2
KI  260 290        1
KD  270 250        4
YA  280 290        4
```

test2の2列目から4列目までの各行の合計値を求める処理

1. 処理1: 2列目の合計値を求め、第1要素に代入する
2. 処理2: 3列目の合計値を求め、第2要素に代入する
3. 処理3: 4列目の合計値を求め、第3要素に代入する
4. 変数名を代入する

apply関数の処理を解読する

```
#Rにはapply関数群と呼ばれる一括演算に特化した関数が用意されています
#apply関数はapply(matrix, a, function)として使用します
#「各行」に作用させる場合はa=1,「各列」に作用させる場合はa=2を代入します
#先程のcolSums関数はapply(matrix, 2, sum)と同義です
result2<-apply(test2[,2:4], 2, sum)
```


colSums()処理をforループで置き換える

ループの開始　　*i* を2から4まで変化させつつ
test2の *i* 列の合計値を算出し
result3のオブジェクトの *i*-1 番目の要素に代入する
ループの終了

```
#まず空のベクターオブジェクトresult3を用意します
result3<- vector()

#次にforループ処理を行いresult3に代入します
for (i in 2:4){
        result3[i-1]<- sum(test2[ ,i])
}

#最後にresult3に名前を代入します
names(result3)<-names(test2[,2:4])
```

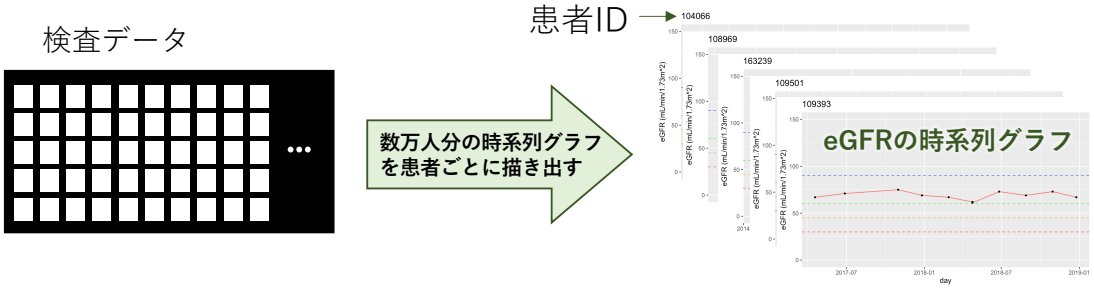
```
for (i in start:end)
{
        処理
        処理
        処理
 }
```

```
#result3とresult1, result2を比べてみましょう
```

「臨床検査データ解析」におけるforループ

これまで、私自身が臨床検査データの解析を行ってきた中で、forループを使わざるを得なかった4つの事例について紹介します

事例1: 検査データから、患者ごとに検査値の時系列グラフを作成する場合



事例2: 検査データから、検査値ごとのヒストグラムを一括表示させる場合

事例3: 検査データを検査項目ごとの基準範囲に従って標準化する場合

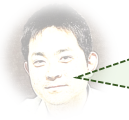
セクション4,5で説明します

事例4: 検査データの機械学習を自動化する場合

セクション3のまとめ

Rの応用

- ☐ 複数のデータフレームをリスト化できる
- ☐ リストの要素を取り出すことができる
- ☐ リストの各要素を一括処理できる
- ☐ forループで行われている処理を理解できる
- ☐ forループを使わない関数群を知っている



リストとforループは、実はとても相性の良いコンビなのです。実際に臨床検査データを解析する際、機械学習による予測モデルの作成や性能評価する時にこの２つを組み合わせて使う場合が多いです。

セクション4：病院検査値データ解析の実際

講習会で扱うデータセット

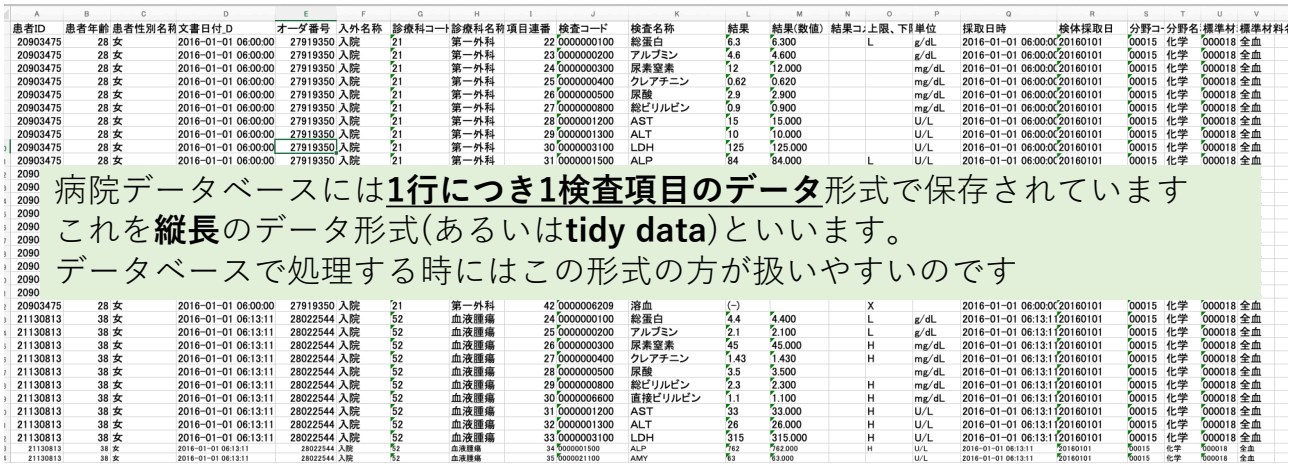
✓ **病院データベースから抽出したリアルデータ(エクセルファイル)**

- 「koushukai」フォルダの「data」に入っています
- 2016年に九州大学病院検査部で測定した生化学検査値のデータです
- 月ごとにエクセルファイル(.xlsx)にまとめられています
- **患者IDは変換済みです**
- 主病名に「糖尿病」を含む患者のデータになります
- **169,158件**の検査オーダーで**3,990,925項目**の検査値を含みます
- 患者**21,729人分**のデータになります
- 患者ID以外はほぼ原型をとどめています
- **本講習会以外の用途で使用することを禁じます**

生データのままでは解析できないので、
解析できるフォーマットに変換します

リアルな病院検査データ

#リアル病院データを使いましょう！
#扱うデータは、九州大学病院の2016年の生化学検査データ
#まずは2016年１月のエクセルファイルを開いてみましょう



病院データベースには**1行につき1検査項目のデータ**形式で保存されています
これを**縦長**のデータ形式(あるいは**tidy data**)といいます。
データベースで処理する時にはこの形式の方が扱いやすいのです

立ち上がり遅い　同じ患者IDが多い
日付が細かい　診療科名いろいろ
結果もいろいろ　単位もいろいろ。。

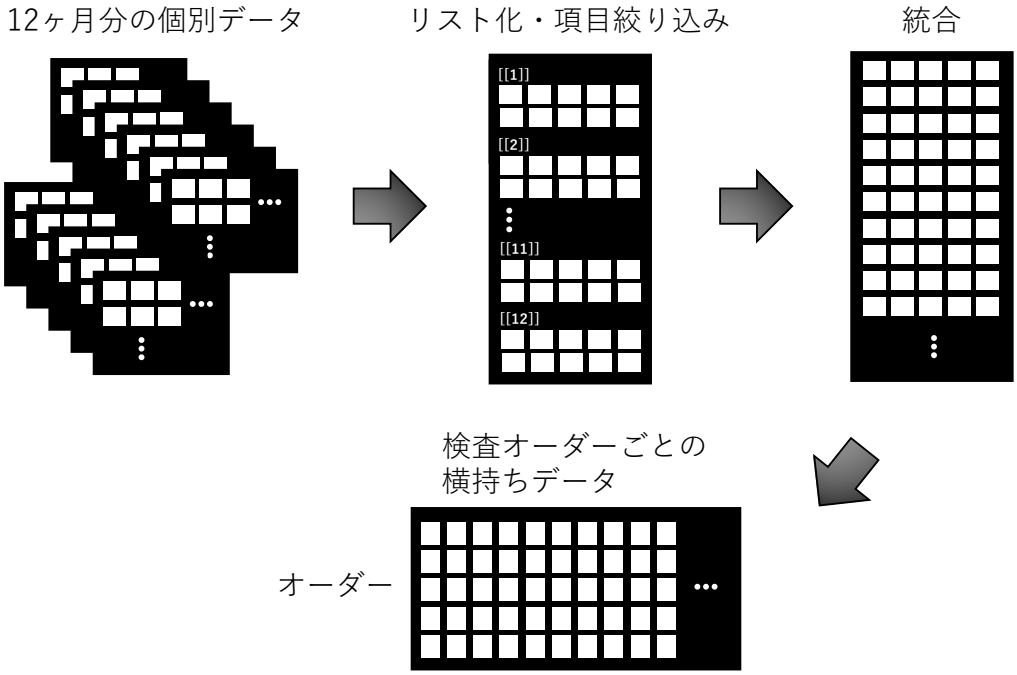
リアルな病院検査データ

本実習では右の項目を使って解析を進めます

- **患者ID（第1列目）**
- **性別（第３列目）**
- **日付（第4列目）**
- **オーダー番号（第5列目）**
- **検査コード（第10列目）**
- **結果（数値）（第12列目）**

病院データの解析前処理

#これから処理するワークフローのイメージです



パッケージの実装、関数のヘルプ参照

```
#必要なパッケージを実装しましょう
#エクセルファイル読み込み用
library(readxl)
#データ前処理用
library(data.table);library(tidyr);library(dplyr)
#可視化ツール
library(ggplot2)
```

```
#関数の使い方はヘルプを参照して下さい
#?関数名で右下の画面に表示されます
#各関数がもつ引数やパラメーター設定など確認して下さい
```

ファイルパスの一括取得

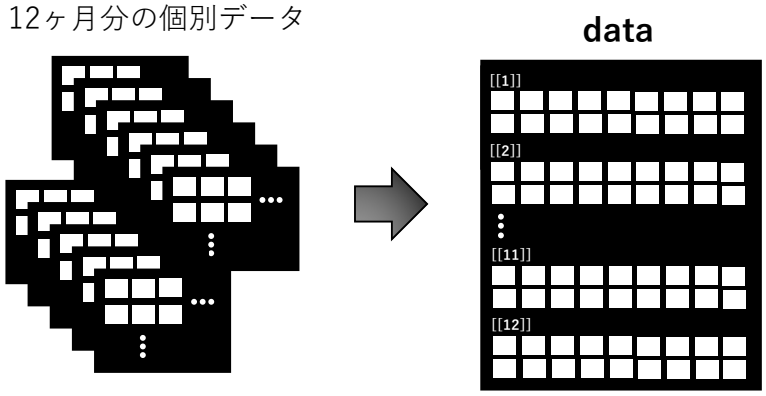
```
#毎月の検査値エクセルデータ12ヶ月分がdataフォルダ内にあります
#今からdataフォルダのエクセルファイルを一括で読み込みます
#まず、現作業フォルダの下にdataフォルダを指定します
path <- "./data/"
#次にdataフォルダのファイル名を全て入手します
#list.files関数(標準関数）を使います
fl<-list.files(path, full.names=T)
#確認しましょう
fl
```

```
> fl
 [1] "./data//201601_chemistry.xlsx" "./data//201602_chemistry.xlsx"
 [3] "./data//201603_chemistry.xlsx" "./data//201604_chemistry.xlsx"
 [5] "./data//201605_chemistry.xlsx" "./data//201606_chemistry.xlsx"
 [7] "./data//201607_chemistry.xlsx" "./data//201608_chemistry.xlsx"
 [9] "./data//201609_chemistry.xlsx" "./data//201610_chemistry.xlsx"
[11] "./data//201611_chemistry.xlsx" "./data//201612_chemistry.xlsx"
```


エクセルファイルを一括読み込み

```
#apply関数のリスト版であるlapply関数を使います
#readxl::read_excel関数と組み合わせることで各月の検査値データをリストの
要素として読み込ませます
data <- lapply(fl, readxl::read_excel)。　。　。
#RStudio右上のオブジェクトパネルを確認してください
```

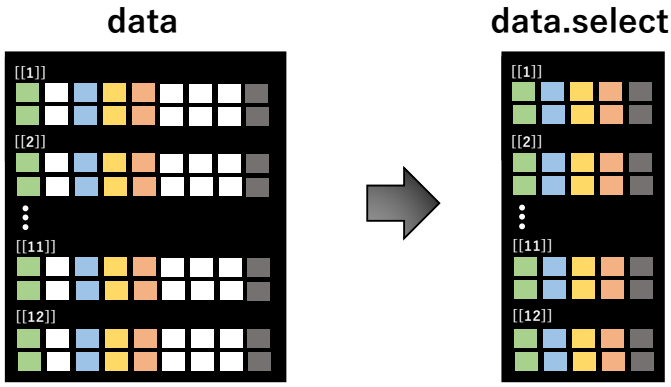
最も時間のかかる処理



項目の絞り込み

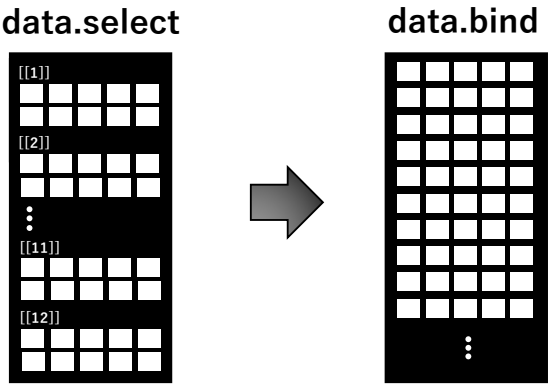
```
#必要な項目のみを抽出します
#患者ID（第1列目）
#性別(第３列目)
#検査オーダー番号（第４列目）
#オーダー日（第５列目）
#検査コード（第１０列目）
#検査値(第１２列目)
data.select<-lapply(data, "[", c(1,3,4,5,10,12))
```

リストの要素の要素を抽出する操作：スライド46を参照してください



リストの各要素を縦方向に連結する処理

```
#リストに含まれるデータファイルを縦方向に連結します
#全ての要素のフォーマットが全く同じであることが必要です
#dplyrパッケージのbind_rows関数を使います
data.bind<-dplyr::bind_rows(data.select)
```



日本語標記名の確認と変換

```
#名前を見ましょう
names(data.bind)
```

```
> names(data.bind)
[1] "患者ＩＤ"       "患者性別名称" "文書日付_D"    "オーダ番号"    "検査コード"    "結果"
```

```
#ちゃんと抽出できていますが、日本語表記は扱いづらいので、英語表記に変換します
names(data.bind)<-c("id","sexmale", "day","order","item","score")
#確認して下さい
```


変数の型(mode)を確認する

```
#このデータフレームの各変数のデータ型を確認しましょう
str(data.bind)
```

```
> str(data.bind)
tibble [3,990,925 × 6] (S3: tbl_df/tbl/data.frame)
 $ id     : num [1:3990925] 20903475 20903475 20903475 20903475 20903475 ...
 $ sexmale: chr [1:3990925] "女" "女" "女" "女" ...
 $ day    : chr [1:3990925] "2016-01-01 06:00:00" "2016-01-01 06:00:00" "2016-01-01 06:00:00" "2016-01-01 06:00:00" ...
 $ order  : num [1:3990925] 27919350 27919350 27919350 27919350 27919350 ...
 $ item   : chr [1:3990925] "0000000100" "0000000200" "0000000300" "0000000400" ...
 $ score  : chr [1:3990925] "6.3" "4.6" "12" "0.62" ...
```

変数の型を変更しましょう

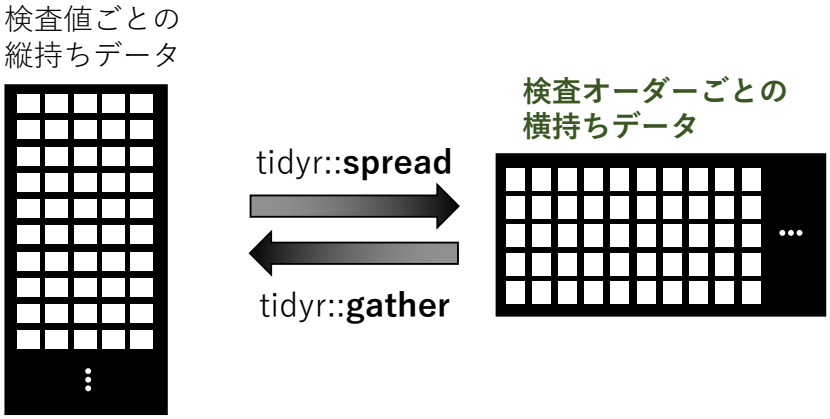
ID: 数値型 性別:文字型 日付：文字型 オーダー：数値型 検査項目：文字化 検査値：文字化	**ID: 文字化** **性別:因子化** **日付：Date型** **オーダー：文字型** **検査項目：数値化** **検査値：数値化**

変数の型を変換する

```
#idと検査オーダーを文字化(character)します
#オーダー日を日付型化(Date)します（* 年月日に短縮されます）
#検査コードと検査値は数値化(numeric)します
data.bind$id<-as.character(data.bind$id)
data.bind$sexmale<-factor(data.bind$sexmale,levels=c("女","男"),labels=c(0,1))
data.bind$order<-as.character(data.bind$order)
data.bind$item<-as.numeric(data.bind$item)
data.bind$day<-as.Date(data.bind$day)
data.bind$score<-as.numeric(data.bind$score)
#確認しましょう
str(data.bind)
```


縦持ち->横持ち変換

```
#縦持ち(long型)から横持ち(wide型）に変換します
#検査オーダー(ID,日付）ごとのデータに変形します
spread.data<-tidyr::spread(data.bind, key=item, value=score)
spread.data<-as.data.frame(spread.data)
```




```
#右上のパネルでspread.dataオブジェクトを確認してください
```

リアル検査データの観察

```
#spread.dataの中身を観察してみましょう
str(spread.data)
```

```
> str(spread.data)
'data.frame':   169158 obs. of  153 variables:
 $ id     : chr  "20903475" "21130813" "13279003" "8906411" ...
 $ sexmale: Factor w/ 2 levels "0","1": 1 1 2 2 1 2 2 1 1 1 ...
 $ day    : Date, format: "2016-01-01" "2016-01-01" "2016-01-01" ...
 $ order  : chr  "27919350" "28022544" "28019770" "28011398" ...
 $ 100    : num  6.3 4.4 4.6 4.5 6.2 8.7 NA 4.2 3.3 4 ...
 $ 200    : num  4.6 2.1 2.3 2.7 3.4 1.9 NA 2.9 2.6 1.2 ...

 $ 2900   : num  NA NA NA NA NA NA NA NA NA NA ...
 $ 3100   : num  125 315 383 130 133 304 NA 218 207 227 ...
 $ 3300   : num  NA NA NA 38 NA NA NA NA NA NA ...
 $ 3900   : num  NA NA NA NA NA NA NA NA NA NA ...
 $ 3901   : num  NA NA NA NA NA NA NA NA NA NA ...
 $ 3902   : num  NA NA NA NA NA NA NA NA NA NA ...
```

169,158の観測値(検査オーダー）があり、153種類の検査項目があります

「NA」は欠損値です
検査項目によってはほとんどが欠損値です

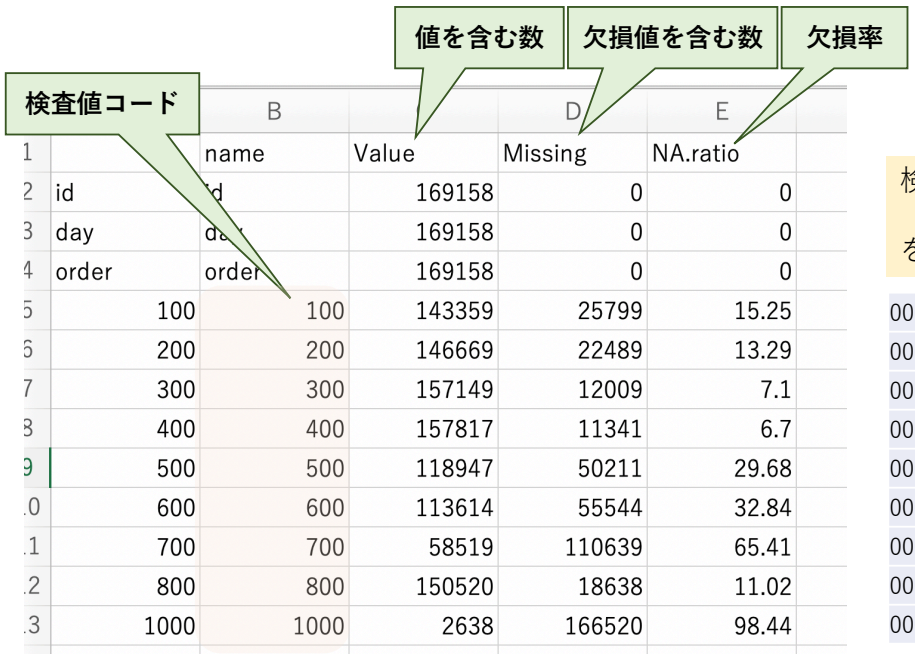
検査項目ごとに、どれだけ欠損値を含むのかを調べる必要があります

検査値の欠損率の算出

```
#各検査値の欠損値を把握しましょう
#各検査値の欠損値を算出する関数を作りました
#下記スクリプトをコピペして実行して下さい
na.summary<-function(data){
  name<-names(data) #nameは各検査値の名前（検査コード）
  Value<-colSums(!is.na(data)) #Nは値を含むデータ数
  Missing<-colSums(is.na(data)) #Missingは欠損値を含むデータ数
  NA.ratio<-round((Missing/nrow(data))*100,2) #NA.ratioは欠損率
  df<-as.data.frame(cbind(name,Value,Missing,NA.ratio))
  write.csv(df,file="summary of NA.csv")
}

#使い方はna.summary(データフレーム)です
#作業フォルダに結果が保存されます
na.summary(spread.data)
```

病院リアルデータの特徴把握

```
#summary of NA.csvファイルをエクセルで開きましょう
```



	A	B	C	D	E
1		name	Value	Missing	NA.ratio
2	id	id	169158	0	0
3	day	day	169158	0	0
4	order	order	169158	0	0
5	100	100	143359	25799	15.25
6	200	200	146669	22489	13.29
7	300	300	157149	12009	7.1
8	400	400	157817	11341	6.7
9	500	500	118947	50211	29.68
10	600	600	113614	55544	32.84
11	700	700	58519	110639	65.41
12	800	800	150520	18638	11.02
13	1000	1000	2638	166520	98.44

検査値コード対応表（エクセルファイル）を参照して下さい

0000000100	総蛋白
0000000200	アルブミン
0000000300	尿素窒素
0000000400	クレアチニン
0000000500	尿酸
0000000600	カルシウム
0000000700	無機リン
0000000800	総ビリルビン
0000001000	ＴＴＴ

検査項目ごとに欠損値が異なる

検査項目の絞り込み

```
#欠損率を多く含む検査値を除外します
#充足率が70%以上の検査値だけをspread.dataから抽出します
select.70<-
  spread.data[ ,colSums(!is.na(spread.data))>nrow(spread.data)*0.7]
```

真偽式になっています

翻訳すると。。

spread.dataの中から、欠損値ではない(!is.na(spread.data))数を各検査項目ごとにカウントして(colSums)、それがデータ数70%(nrow(spread.data)*0.7)より大きい(>)場合を判定し、TRUEの検査項目(列) を抽出します

```
names(select.70)
```

```
> names(select.70)
 [1] "id"    "day"   "order" "100"   "200"   "300"   "400"   "500"   "800"   "1200"  "1300"
[12] "1500"  "1600"  "2000"  "2100"  "2200"  "2300"  "2800"  "3100"  "6207"  "6400"
```

これらの21種類の変数（検査項目）が該当するそうです

検査値名への変換

```
#正式な検査値名(略称)に変換しましょう
#name70.csvファイルに21種類の名前が入っています
#このファイルを読み込んで、select.70の名前を置換しましょう
name70<-read.csv("name70.csv",header=T)
names(select.70)<-name70$name
names(select.70)
```

ようやくおなじみの名前が出てきました。。

```
> names(select.70)
 [1] "id"    "day"   "order" "TP"    "ALB"   "BUN"   "CRE"   "UA"    "Tbil"  "AST"   "ALT"
[12] "ALP"   "GGT"   "GLU"   "Na"    "K"     "Cl"    "CRP"   "LDH"   "A/G"   "eGFR"
```

1	TP	総蛋白	10	GGT	gamma-GT
2	ALB	アルブミン	11	GLU	グルコース
3	BUN	尿素窒素	12	Na	Na+
4	CRE	クレアチニン	13	K	K+
5	UA	尿酸	14	Cl	Cl-
6	Tbil	総ビリルビン	15	CRP	CRP
7	AST	AST	16	LDH	LDH
8	ALT	ALT	17	A/G	AG比
9	ALP	ALP	18	eGFR	推算糸球体濾過量

データの確認

```
#データの構造を確認しておきましょう
str(select.70)
```

```
> str(select.70)
'data.frame':   169158 obs. of  22 variables:
 $ id     : chr  "20903475" "21130813" "13279003" "8906411" ...
 $ sexmale: Factor w/ 2 levels "0","1": 1 1 2 2 1 2 2 1 1 1 ...
 $ day    : Date, format: "2016-01-01" "2016-01-01" "2016-01-01" ...
 $ order  : chr  "27919350" "28022544" "28019770" "28011398" ...
 $ TP     : num  6.3 4.4 4.6 4.5 6.2 8.7 NA 4.2 3.3 4 ...
 $ ALB    : num  4.6 2.1 2.3 2.7 3.4 1.9 NA 2.9 2.6 1.2 ...
 $ BUN    : num  12 45 35 7 24 22 NA 27 35 7 ...
 $ CRE    : num  0.62 1.43 2.21 0.58 0.59 3.09 NA 0.45 0.7 0.54 ...
 $ UA     : num  2.9 3.5 5 2.3 NA 7.1 NA 4.7 3.4 2.4 ...
 $ Tbil   : num  0.9 2.3 1.1 1.8 0.7 2.1 NA 3.5 1.5 0.8 ...
 $ AST    : num  15 33 52 148 17 61 NA 28 46 32 ...
 $ ALT    : num  10 26 26 561 25 24 NA 36 75 12 ...
 $ ALP    : num  84 762 242 189 207 375 NA 259 65 577 ...
 $ GGT    : num  NA NA NA 232 NA NA 233 NA NA NA ...
 $ GLU    : num  95 118 195 120 NA 119 NA 143 125 105 ...
 $ Na     : num  141 139 135 143 134 142 NA 147 135 138 ...
 $ K      : num  3.5 3.2 3.5 3.9 4.2 3.4 NA 4.9 4.5 4.4 ...
 $ Cl     : num  107 105 102 106 101 93 NA 118 108 105 ...
 $ CRP    : num  0.05 1.86 21.98 0.1 0.03 ...
 $ LDH    : num  125 315 383 130 133 304 NA 218 207 227 ...
 $ AG     : num  2.71 0.91 1 1.5 1.21 0.28 NA 2.23 3.71 0.43 ...
 $ eGFR   : num  93 34 23 115 75 18 NA 103 66 86 ...
```

- 169,158の観測値(18種類の検査値プロファイル）に絞り込むことができました。
- このデータセットを使って、検査値の相関解析やプロファイル比較、あるいは時系列変化等を解析することができます。
- また、患者の層別化解析なども可能です。
- 患者の病歴や投薬情報等を付与することができれば、より病態に即した検査値変化の解析が可能になります。

患者数の把握

```
#患者数は何人ですか？
#unique関数(標準関数）を使って重複のないidを求めます
#更に、length関数(標準関数)を使って数をカウントします
#つまり、「重複無しのIDの数=患者数」です
length(unique(select.70$id))
```

```
> length(unique(select.70$id))
[1] 21789
```


セクション4のまとめ

病院検査値データ解析の実際

- ☐ 病院から取り出した検査値データの構造を理解できる
- ☐ 共通したフォーマットファイルをリストとして読み込ませることができる
- ☐ リスト要素の連結ができる
- ☐ データフレームの変数の型を変換できる
- ☐ 縦持ち、横持ち変換ができる
- ☐ 変数ごとの欠損率を把握することができる
- ☐ 欠損率（またはデータの充足率）に従って変数を絞り込むことができる

セクション5：病院検査値データの正規化

このセクションの目的と免責事項

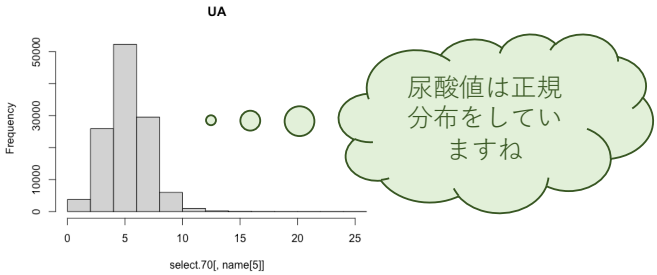
- これ以降のセクションでは、「検査値の基準範囲(reference interval」を使って**検査値データを正規化する方法**を解説します。
- ただし、本手法は、私が独自に考案したもので、本学会、本委員会および臨床検査学・医療情報学の分野で**公認されているものではありません**。
- 本手法に関するご質問・ご意見はいつでも歓迎しますが、本手法を用いて生じた結果については、私は責任を負いかねますのでご了承ください。
- あくまでも、「リファレンス情報を使って変数（検査値）ごとに値を正規化する技術を習得する」という位置づけで臨んでください。

各検査値を可視化する

```
#各検査値の分布をヒストグラムを描いて眺めましょう
#hist()関数を使い、hist(data)で可視化します
#まず、変数名をnameオブジェクトに取り出しましょう
#select.70の第5-22列目に検査値が含まれています
name<-names(select.70[,5:22])
name
```

```
> name
 [1] "TP"   "ALB"  "BUN"  "CRE"  "UA"   "Tbil" "AST"  "ALT"  "ALP"  "GGT"  "GLU"  "Na"   "K"    "Cl"
[15] "CRP"  "LDH"  "AG"   "eGFR"
```

```
#尿酸値(UA)のヒストグラムは下記のように描けます
hist(select.70[,name[5]], main=name[5])
#mainはタイトルを表示させるオプションです
```



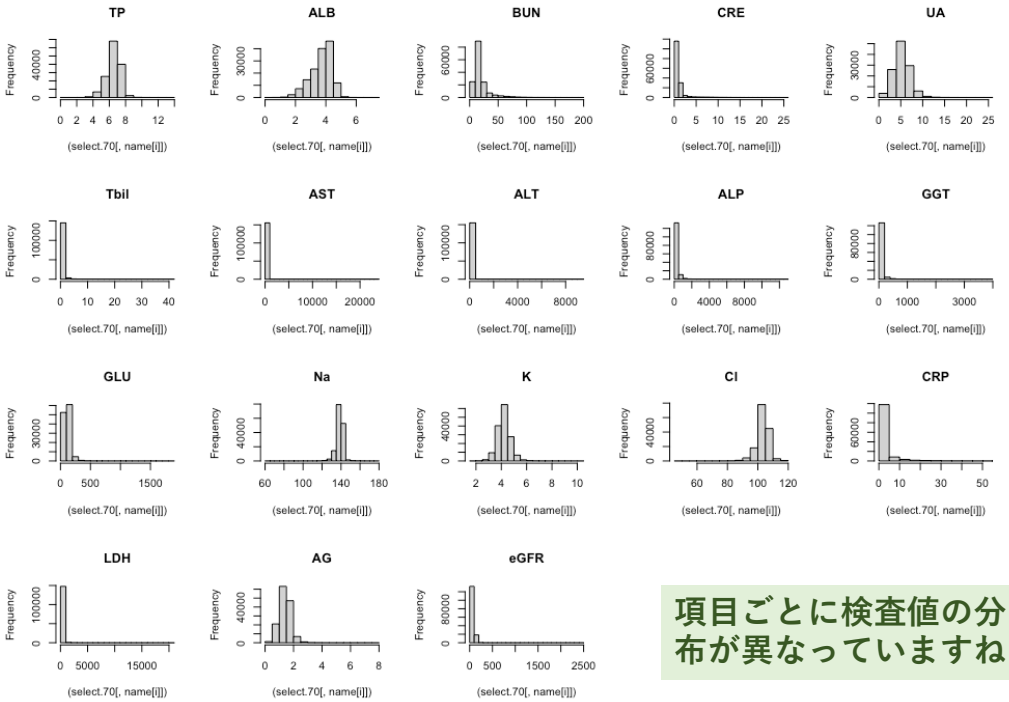
各検査値を可視化する

```
#forループ関数を使ってヒストグラムを一括表示させましょう
#検査値は18種類ありますので、18枚の図を同じ画面に表示させます
#par()関数のmfrowオプションで指定します

par(mfrow=c(4,5))  #図を4行5列に配置する設定です

#次にforループ処理を行います
for(i in 1:18){
  hist((select.70[, name[i]]), main=name[i])
}

#描画が終わりましたらpar()オプションを初期化しておきましょう
par(mfrow=c(1,1))
```

mfrowとはmultiframe rowwise layout の略です

各検査値を可視化する

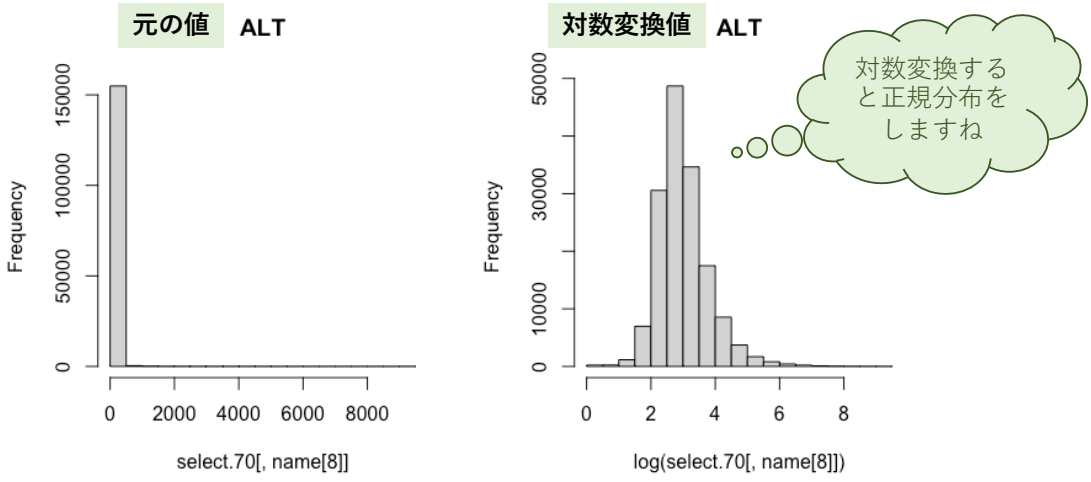
- 縦軸はデータ数/横軸は各検査値の値



項目ごとに検査値の分布が異なっていますね

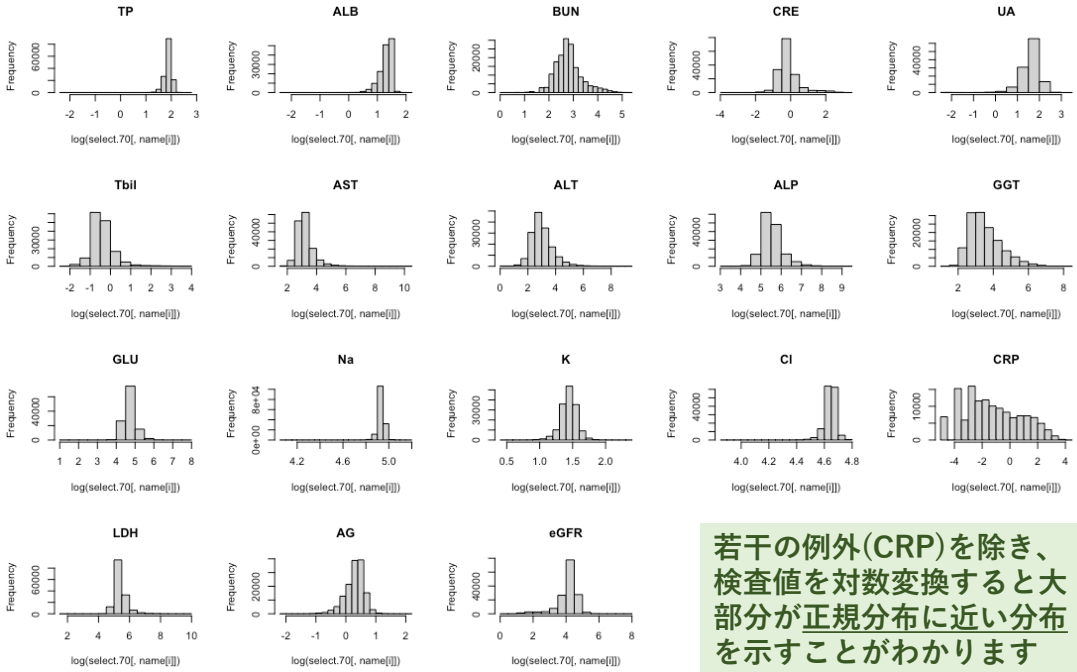
検査値の対数変換値を可視化する

```
#検査値を対数変換(eを底とする）してヒストグラムを描いてみましょう
#log()関数を使って変換します
#ALTを例にとり、元の検査値と対数値のヒストグラムを比較しましょう
par(mfrow=c(1,2))
hist(select.70[,name[8]], main=name[8]) #元の検査値のグラフ
hist(log(select.70[,name[8]]), main=name[8]) #対数変換値のグラフ
```



検査値の対数変換値を可視化する

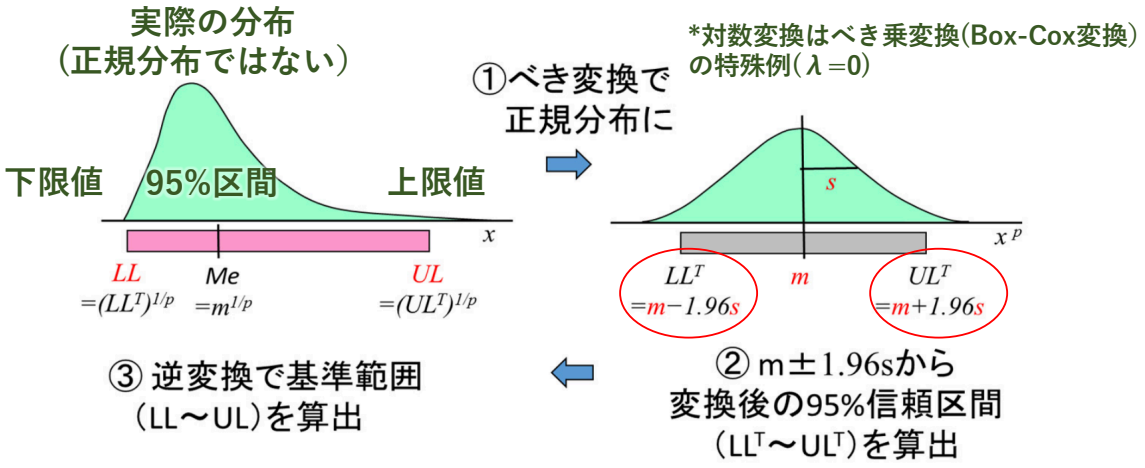
- 縦軸はデータ数/横軸は各検査値の対数変換値



若干の例外(CRP)を除き、検査値を対数変換すると大部分が正規分布に近い分布を示すことがわかります

臨床検査値の基準範囲の算出法

日本における主要な臨床検査項目の共用基準範囲-解析と利用の手引-
2019/01/25修正版
日本臨床検査標準協議会　基準範囲共用化委員会 編



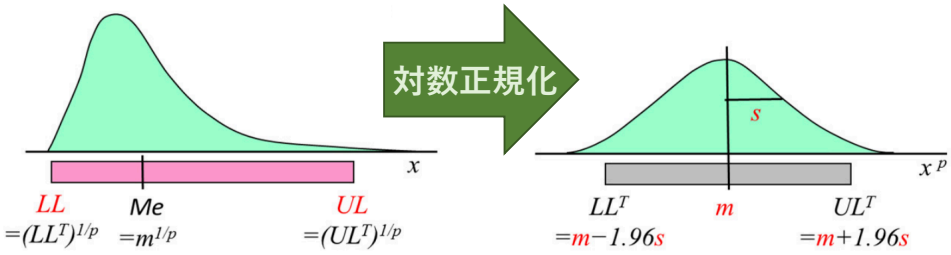
基準範囲の上限値・下限値の（対数）変換値は、
(対数)正規分布の「m±1.96s」の近似値である

検査データの正規化の意味と算出法

リアル検査値
- 値の分布や単位が異なる
- 基準範囲は年齢や性別によって異なる場合がある

正規化検査値
- 左の制約が無くなる
- 全ての検査値を同じ尺度（Zスコア）で表示させる

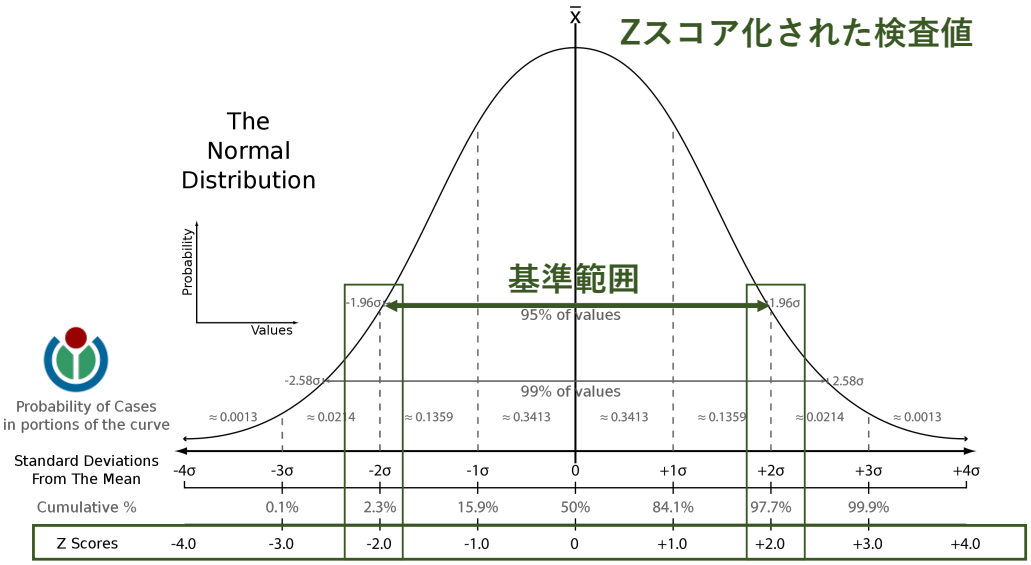
- 対数変換は全ての検査値を、厳密な意味において、正規分布化させるわけではない点にご留意ください。あくまで、正規分布に「近似」させて処理しているに過ぎません



- 正規化検査値　＝ (ln(x)-m)/s
- 正規化平均値m＝ (ln(下限値) + ln(上限値))/2
- 正規化標準偏差s ＝ (ln(上限値) - ln(下限値))/3.92

*ln: eを底とするlogのこと

正規化データ(Zスコア）の解釈



Xスコアの値	解釈
Z <- 2	異常低値(極端値）です
-2 <= Z <= +2	基準範囲内です
Z > +2	異常高値（極端値）です

検査値データの正規化 Normalization

✓ **検査値の正規化を行うロジック**

1. 検査値の基準範囲(reference interval)は、基準個体の基準値95%区間の上限・下限値で定義される
2. 一部の検査値の基準範囲は性別や年齢によって異なる
3. 検査値は固有の分布を持つが、対数変換（またはべき乗変換）すると正規分布（対数正規分布）に近似できる（経験則）
4. 正規分布では平均値±2SD値の範囲に全体の約95%が含まれる

↓

対数変換した基準範囲の上・下限値を使った対数正規分布に対し、対数変換した検査値を当てはめることで、基準範囲に基づく正規化された値(Zスコア）を算出できる

↓

- 検査値同士をZスコアで比較することができる
- 検査値を性別や基準範囲に照合させることなく、正常高・低値または異常値の判断が可能となる
- 正規化データを使った様々な解析(機械学習など)が可能となる

性別で基準範囲が異なる検査項目

検査項目	略称	単位	男性	女性
Serum chemistry				
Serum creatinine	CRE	mg/dL	0.65～1.07	0.46～0.79
Uric acid	UA	mg/dL	3.7～7.8	2.6～5.5
Triglyceride	TG	mg/dL	40～234	30～117
High-density lipoprotein	HDL	mg/dL	38～90	48～103
Alanine aminotransferase	ALT	U/L	10～42	7～23
gamma-Glutamyl transferase	GGT	U/L	13～64	9～32
Creatine kinase	CK	U/L	59～248	41～153
Complete blood count (CBC)				
Red blood cell	RBC	$\times 10^6/\mu L$	4.35～5.55	3.86～4.92
Hemoglobin	Hb	g/dL	13.7～16.8	11.6～14.8
Hematocrit	Ht	%	40.7～50.1	35.1～44.4

JCCLSの共用基準範囲による

このセクションで使用するデータ

男女別の検査値の共用基準範囲データ

- 「reference interval.m.csv」
- 「reference interval.f.csv」
- 全43項目の検査値名(description)、略称(item)、単位(unit)、基準範囲の下限値(min)、上限値(max)、由来(source)に関する情報をまとめています
- セクション4で作成したデータセット「select.70」の18種類の検査項目の基準範囲も含まれています

desctiption	item	unit	min	max	source
Total protein	TP	g/dL	6.6	8.1	JCCLS
Albumin	ALB	g/dL	4.1	5.1	JCCLS
Albumin/IgG ratio	AG	nd	1.32	2.23	JCCLS
Blood urea nitrogen	BUN	mg/dL	8	20	JCCLS
Serum creatinine	CRE	mg/dL	0.65	1.07	JCCLS
Uric acid	UA	mg/dL	3.7	7.8	JCCLS
Na	Na	mmol/L	138	145	JCCLS
K	K	mmol/L	3.6	4.8	JCCLS
Cl	Cl	mmol/L	101	108	JCCLS
⋮	⋮	⋮	⋮	⋮	⋮

検査値データの正規化手順

STEP1: 検査値データ（セクション3）を男女別に分割する

▼

STEP2: 男女別の基準範囲データを読み込む

▼

STEP3: 対数正規分布のパラメータを逆算する

▼

STEP4: forループを使って検査データを正規化する

▼

STEP5: 男女分割データを統合し、元の検査値データに並び替える

STEP1: 検査値データを男女別に分割する

```
##Step1: 検査データを男女別に分割する
data.m<-select.70[select.70$sexmale==1,]
data.f<-select.70[select.70$sexmale==0,]
```


STEP2: 男女別の基準範囲データを読み込む

```
#Step2: 基準範囲データの読み込み
#男女別の検査値ごとの基準範囲（95%区間の上限値、下限値）を記述
ri.m<-read.csv("reference interval.m.csv",header=T) #男性の基準範囲
ri.f<-read.csv("reference interval.f.csv",header=T) #女性の基準範囲

#中身を確認しましょう
#検査値表記, 略称, 単位, 下限値, 上限値, 参照値が含まれています
str(ri.m)
```

```
> str(ri.m)
'data.frame':   43 obs. of  6 variables:
 $ desctiption: chr  "Total protein" "Albumin" "Albumin/IgG ratio" "Blood urea nitrogen"
 $ item       : chr  "TP" "ALB" "AG" "BUN" ...
 $ unit       : chr  "g/dL" "g/dL" "nd" "mg/dL" ...
 $ min        : num  6.6 4.1 1.32 8 0.65 3.7 138 3.6 101 8.8 ...
 $ max        : num  8.1 5.1 2.23 20 1.07 7.8 145 4.8 108 10.1 ...
 $ source     : chr  "JCCLS" "JCCLS" "JCCLS" "JCCLS" ...
```

基準範囲の下限値(min)と上限値(max)が含まれています

STEP3: 対数正規分布のパラメータを逆算する

```
#Step3: 対数正規分布のパラメータを逆算する
#対数正規分布の平均値とSD値を求め、新たな変数として格納する
#男性の基準範囲データを変換する
ri.m$avr<-(log(ri.m$min)+log(ri.m$max))/2)  #log()は自然対数に変換します
ri.m$sd<-(log(ri.m$max)-log(ri.m$min))/3.92)
#女性の基準範囲データを変換する
ri.f$avr<-(log(ri.f$min)+log(ri.f$max))/2)
ri.f$sd<-(log(ri.f$max)-log(ri.f$min))/3.92)
str(ri.m)
```

```
> str(ri.m)
'data.frame':   43 obs. of  8 variables:
 $ desctiption: chr  "Total protein" "Albumin" "Albumin/IgG ratio" "Blood urea nitrogen"
 $ item       : chr  "TP" "ALB" "AG" "BUN" ...
 $ unit       : chr  "g/dL" "g/dL" "nd" "mg/dL" ...
 $ min        : num  6.6 4.1 1.32 8 0.65 3.7 138 3.6 101 8.8 ...
 $ max        : num  8.1 5.1 2.23 20 1.07 7.8 145 4.8 108 10.1 ...
 $ source     : chr  "JCCLS" "JCCLS" "JCCLS" "JCCLS" ...
 $ avr        : num  1.989 1.52 0.54 2.538 -0.182 ...
 $ sd         : num  0.0512 0.0546 0.1311 0.2291 0.1246 ...
```

対数正規分布の平均値avr
対数正規分布の標準偏差sd

forループ処理のイメージ

検査値データ **data**

		AST	ALT			
		25	22			
		26	21			
		22	22			
		27	23			

data[,"AST"]

基準範囲データ **ri**

Item			avr	sd
AST			2.99	0.35
ALT			2.71	0.56

ASTのavr=**ri**[item=="AST", "Avr"]
ASTのsd=**ri**[item=="AST", "SD"]

$$\text{Z score}=\frac{\log(\textbf{data}[\ ,\text{"AST"}])-\textbf{ri}[\text{item}=\text{"AST"},\text{"avr"}]}{\textbf{ri}[\text{item}=\text{"AST"},\text{"sd"}]}$$

"AST"の部分を name[*i*]を使って書き換えるとforループで処理できる

STEP4: forループを使って検査データを正規化する

```
##Step4: 基準範囲による正規化
#新しい男女別データフレームを用意しましょう
std.m<-data.m
std.f<-data.f

#forループで検査値ごとの対数正規分布に従って正規化します
for(i in 1:length(name)){

#男性データの変換
std.m[,name[i]]<-
((log(std.m[,name[i]])-ri.m[ri.m$item==name[i],"avr"])/ri.m[ri.m$item==name[i],"sd"]))

#女性データの変換
std.f[,name[i]]<-
((log(std.f[,name[i]])-ri.f[ri.f$item==name[i],"avr"])/ri.f[ri.f$item==name[i],"sd"]))

}

#解読できましたか？
#今まで学んだことを総動員したら理解できるはずです！
#検査データの検査値名と基準範囲データのitemの名前は完全一致させる必要がありますので、注意してください
```


STEP5: データを統合し、並び替える

```
##Step5
#dplyr::bind_rows関数を使い、男女別の標準化データを統合する
std <- dplyr::bind_rows(std.m, std.f)

#dplyr::arrange関数を使い、日付の昇順に並び替える
test <- dplyr::arrange(std, day)
#以上の操作により元のデータを検査値のみ標準化したデータセットを得ることができる
```

結果の観察と考察

元の検査値データ

```
> head(select.70[,5:22],6)
  TP ALB BUN  CRE  UA Tbil AST ALT ALP GGT GLU  Na   K  Cl   CRP LDH   AG eGFR
1 6.3 4.6  12 0.62 2.9  0.9  15  10  84  NA  95 141 3.5 107  0.05 125 2.71   93
2 4.4 2.1  45 1.43 3.5  2.3  33  26 762  NA 118 139 3.2 105  1.86 315 0.91   34
3 4.6 2.3  35 2.21 5.0  1.1  52  26 242  NA 195 135 3.5 102 21.98 383 1.00   23
4 4.5 2.7   7 0.58 2.3  1.8 148 561 189 232 120 143 3.9 106  0.10 130 1.50  115
5 6.2 3.4  24 0.59  NA  0.7  17  25 207  NA  NA 134 4.2 101  0.03 133 1.21   75
6 8.7 1.9  22 3.09 7.1  2.1  61  24 375  NA 119 142 3.4  93  7.11 304 0.28   18
```

正規化データ

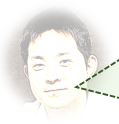
```
> head(std.select.70[,5:22],6)
     TP   ALB   BUN   CRE    UA Tbil  AST   ALT  ALP  GGT  GLU    Na     K    Cl    CRP  LDH    AG  eGFR
1 -0.25 -0.35  1.35  1.46  0.17 0.47 1.30  0.38 0.42   NA 1.12 -0.02 -0.02  0.01 157.00 1.17 -0.14 -0.52
2 -0.26 -0.27 -0.05 -0.07 -0.18 0.93 4.50 13.12 0.26 3.42 0.43  0.03  0.06  0.05   0.71 0.03  0.08  0.50
3  0.26 -0.43  0.70  2.28  0.44 1.13 1.60  0.33 0.84   NA 0.42  0.03 -0.04 -0.07  50.79 0.81 -0.47 -0.58
4    NA    NA    NA    NA    NA   NA   NA    NA   NA 3.44   NA    NA    NA    NA     NA   NA    NA    NA
5 -0.05 -0.24  0.50  0.42  0.17 0.07 0.77  0.36   NA   NA   NA  0.01  0.19  0.05  79.29 1.40 -0.20 -0.19
6 -0.10 -0.24  0.15  0.16  0.13 0.47 0.27 -0.02 1.00 0.73   NA -0.02  0.15  0.01  39.79 1.64 -0.14  0.08
```

元の検査値データは、基準範囲を見比べないと異常値かどうか判断できません。それに対して、正規化データでは、＋2よりも離れた値が異常高値ですので、ひと目見てでわかります。と同時に、異常の程度もわかるはずです。

セクション5のまとめ

病院検査値データの正規化

- ☐ 個々の検査値の分布は正規分布している場合とそうでない場合があることを理解した
- ☐ 検査値をべき乗（対数）変換すると正規分布することを理解した
- ☐ 検査値の基準範囲（reference interval)の成り立ちを理解した
- ☐ 基準範囲の上限値・下限値に基づき、検査値を正規化する手順について理解した
- ☐ 正規化した検査値の解釈を理解した



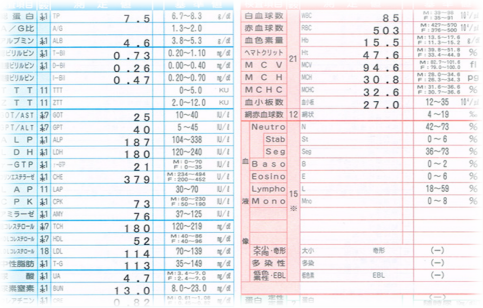
このセクションは、将来、検査値を使って機械学習やディープラーニング等の解析を行う人を対象に解説しました。実際、機械学習等のアルゴリズムは、全ての変数が同じスケールになっていることを前提としています。このため、元の値をそのまま使うよりもスケールを揃えた上で実行した方が上手く動作するそうです。

セクション6：検診データの偏差値化

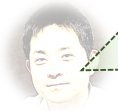
このセクションの目的

このセクションでは、検診データの検査値を偏差値を使って表示させる方法を解説します

健康診断の結果



偏差値？

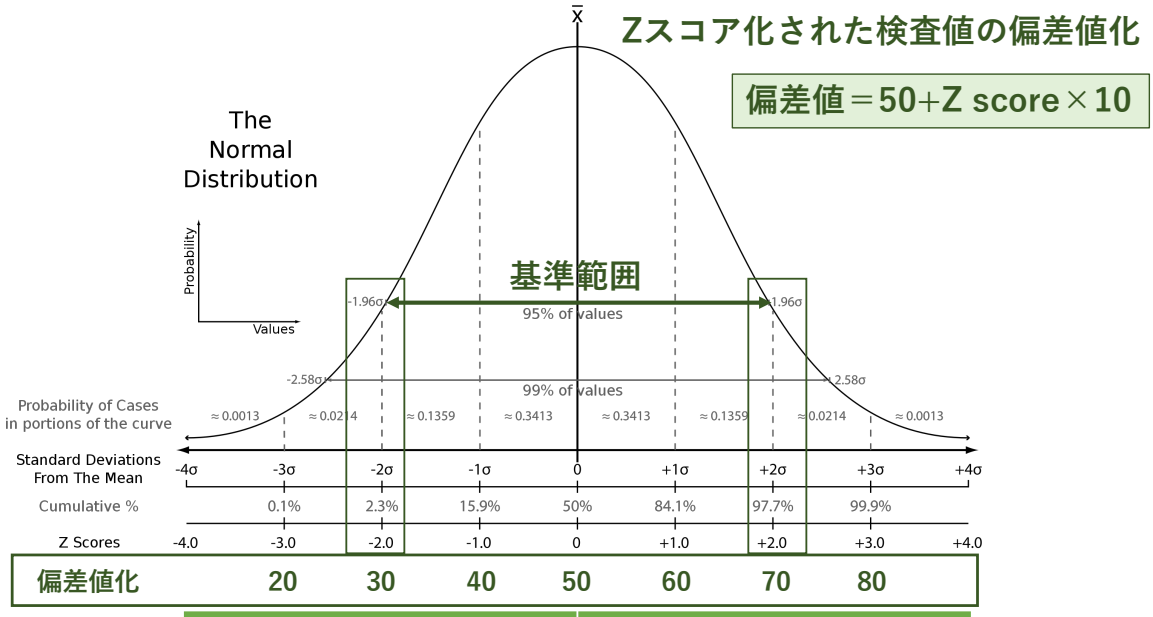


個人的な感想で恐縮ですが、、
健康診断の結果をみて、自分の検査結果が「正常」なのか？「異常」なのか？「正常」でも範囲ギリギリの値なのか？そうでないのか？は、一つずつ検査結果とその「正常範囲」をじっくり眺めないとわかりません。この時、ひと目で見てパットわかるように結果を返してくれたらいいな、と思ったこと有りませんか？

検査結果の偏差値化

Zスコア化された検査値の偏差値化

偏差値＝50+Z score×10



偏差値	解釈
<30	異常低値(極端値）です
30 <= 偏差値 <= 70	基準範囲内です
Z > 70	異常高値（極端値）です

このセクションで使用するデータ

1. ある男性の2011年から2020年までの健康診断結果（生化学検査と血球検査のみを抜粋）
   - 「laboratory test.csv」ファイル
   - 空白期間がありますが、基本的に年2回の健康診断結果です
   - 19項目の臨床検査項目を含みます
   - 空白は欠損値（未測定のため）

2. 検診結果に記載されている基準範囲？のデータ（一部改変）

   ***健診の「基準範囲」は臨床検査分野で使われている「共用基準範囲」とは少し異なりますので、注意してください。**
   - 「reference interval.kenko zaidan.csv」ファイル
   - セクション5で使用した男性用基準範囲データの一部を検診データに置き換えています(source=kenko zaidan)
   - CRNの下限値はJCCLSの下限値に変更しました
   - eGFRの基準範囲は70-90に設定しました（便宜上）
   - UAの基準範囲は7.0以下ですが、下限値をJCCLS値に設定しました

検診データの偏差値化手順

STEP1: 検診データと基準範囲データを読み込む

▼

STEP2:対数正規分布のパラメータを逆算する

▼

STEP3: forループを使って検診データを偏差値化する

▼

STEP4: 値に応じてグレードを付与する

▼

STEP5: グラフ化

STEP1: 検診データと基準範囲データを読み込む

```
#Step1：検診データと基準範囲データの読み込み
data<-read.csv("laboratory test.csv", header=T)
ri<-read.csv("reference interval.kenko zaidan.csv", header=T)
head(data, 6)
head(ri, 6)
```

```
> data
        id  order      date sexmale age RBC   Hb   Ht   WBC  PLT AST ALT GGT Tbil ALP  TG HDL
1 61748341 202082 2011/3/10       1  32 524 14.9 47.3 4.710   NA  25  22  25   NA  NA  78  61
2 61748341 106065 2012/6/19       1  33 525 15.6 47.7 5.000   NA  26  21  25   NA  NA  79  53
3 61748341 153980 2012/12/5       1  33 540 16.1 49.1 4.340   NA  22  22  39   NA  NA 110  59
4 61748341 478382 2013/6/12       1  34 503 15.1 46.5 5.480   NA  27  23  30   NA  NA 122  51
5 61748341 482283 2013/12/2       1  34 523 15.6 47.5 5.650   NA  33  40  60   NA  NA 104  58
6 61748341 714751 2016/6/17       1  37 516 15.4 47.8 6.340 19.9  29  22  32   NA  NA  84  58
> ri
                              desctiption  item class              unit    min    max       source
1                           Total protein    TP     1              g/dL   6.60   8.10        JCCLS
2                                 Albumin   ALB     2              g/dL   4.10   5.10        JCCLS
3                           Albumin/IgG ratio    AG     3                nd   1.32   2.23        JCCLS
4                 Aspartate aminotransferase   AST     4               U/L  10.00  40.00 kenko zaidan
5                   Alanine aminotransferase   ALT     5               U/L   5.00  45.00 kenko zaidan
6                 gamma-Glutamyl transferase   GGT     6               U/L   5.00  79.00 kenko zaidan
```

STEP2:対数正規分布のパラメータを逆算する

```
#Step2: 対数正規分布のmとSDを逆算する（スライド95と同様です）
ri$avr<-((log(ri$min)+log(ri$max))/2)
ri$sd<-((log(ri$max)-log(ri$min))/3.92)
str(ri)
```

```
> str(ri)
'data.frame':   43 obs. of  9 variables:
 $ desctiption: chr  "Total protein" "Albumin" "Albumin/IgG ratio" "Aspartate aminotransferase" ...
 $ item       : chr  "TP" "ALB" "AG" "AST" ...
 $ class      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ unit       : chr  "g/dL" "g/dL" "nd" "U/L" ...
 $ min        : num  6.6 4.1 1.32 10 5 5 0.4 106 8 3.7 ...
 $ max        : num  8.1 5.1 2.23 40 45 79 1.5 322 20 7 ...
 $ source     : chr  "JCCLS" "JCCLS" "JCCLS" "kenko zaidan" ...
 $ avr        : num  1.99 1.52 0.54 3 2.71 ...
 $ sd         : num  0.0522 0.0557 0.1338 0.3536 0.5605 ...
```

計算結果が挿入されています

STEP3: forループ処理で検診データを偏差値化する

```
##Step3: 基準範囲による正規化
#検診データの検査項目をnameに代入します
name<-names(data[, 6:ncol(data)])

#検診データを検査項目ごとの基準範囲にしたがって偏差値化します
std<-data

for(i in 1:length(name)){

  std[, name[i]]<-
((log(std[ , name[i]])-ri[ri$item==name[i],"avr"])/ri[ri$item==name[i], "sd"])*10+50

}
```

ここで偏差値化していますね

結果の観察

```
#偏差値化された検診データを観察しましょう
head(std, 6)
```

```
> head(std,6)
        id  order      date sexmale age      RBC       Hb       Ht      WBC    PLT      AST      ALT      GGT Tbil /
1 61748341 202082 2011/3/10       1  32 55.89672 41.76920 55.07604 41.81836     NA 56.30979 56.83285 53.25860   NA
2 61748341 106065 2012/6/19       1  33 56.16788 47.98553 56.39389 44.11607     NA 57.41883 56.00290 53.25860   NA
3 61748341 153980 2012/12/5       1  33 60.17443 52.25729 60.92089 38.67219     NA 52.69507 56.83285 59.57441   NA
4 61748341 478382 2013/6/12       1  34 50.07959 43.57461 52.40657 47.64117     NA 58.48600 57.62590 55.84809   NA
5 61748341 482283 2013/12/2       1  34 55.62505 47.98553 55.73635 48.81600     NA 64.16033 67.49867 65.69275   NA
6 61748341 714751 2016/6/17       1  37 53.70863 46.23837 56.72162 53.24696 44.242 60.50664 56.83285 56.76472   NA
```

```
#偏差値の小数部分はあまり意味がないので、丸めましょう
#apply関数を使い、偏差値部分にround関数を作用させます
std[, 6:24]<-apply (std[, 6:24], 2, round)
head(std)
```

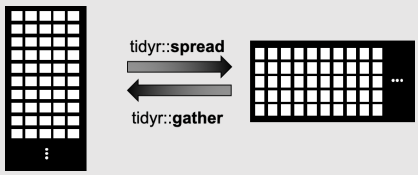
```
> head(std,6)
        id  order      date sexmale age RBC Hb Ht WBC PLT AST ALT GGT Tbil ALP TG HDL LDL TC GLU HbA1c CRE eGFR UA
1 61748341 202082 2011/3/10       1  32  56 42 55  42  NA  56  57  53   NA  NA 46  54  42 39  NA    41  NA   NA 66
2 61748341 106065 2012/6/19       1  33  56 48 56  44  NA  57  56  53   NA  NA 47  46  39 32  NA    44  NA   NA 51
3 61748341 153980 2012/12/5       1  33  60 52 61  39  NA  53  57  60   NA  NA 59  52  47 49  NA    47  NA   NA 70
4 61748341 478382 2013/6/12       1  34  50 44 52  48  NA  58  58  56   NA  NA 62  44  43 34  NA    49  NA   NA 65
5 61748341 482283 2013/12/2       1  34  56 48 56  49  NA  64  67  66   NA  NA 57  51  58 54  NA    47  NA   NA 70
6 61748341 714751 2016/6/17       1  37  54 46 57  53  44  61  57  57   NA  NA 49  51  55 50  NA    54  NA   NA 55
```


STEP4: 値に応じてグレードを付与する

```
#偏差値の値に応じて下記のようにグレードを付与します
```

偏差値の範囲	グレード	割合(%)
<30	Low	2.3
30 <= 偏差値 < 35	Normal Low	4.4
35 <= 偏差値 =< 65	Normal	86.6
65 < 偏差値 =< 70	Normal High	4.4
>70	High	2.3

スライド42, 70




```
#このため、横持ちから縦持ちデータに変換する必要があります
library(tidyr)
std.ts<-tidyr::gather(std, key="item", value="score", name)
std.ts$date<-as.Date(std.ts$date)
head(std.ts,6)
```

nameには全ての検査項目名が含まれています（スライド108）

```
> head(std.ts)
        id  order       date sexmale age item score
1 61748341 202082 2011-03-10       1  32  RBC    56
2 61748341 106065 2012-06-19       1  33  RBC    56
3 61748341 153980 2012-12-05       1  33  RBC    60
4 61748341 478382 2013-06-12       1  34  RBC    50
5 61748341 482283 2013-12-02       1  34  RBC    56
6 61748341 714751 2016-06-17       1  37  RBC    54
```

この偏差値に対してグレードを付与します

STEP4: 値に応じてグレードを付与する

```
#ifelse関数を使ってラベル化します（スライド29を参照）
#dplyrパッケージのmutate関数を使って新たな変数gradeを挿入します
library(dplyr)
std.ts<-dplyr::mutate(std.ts, grade=ifelse(score<30, "Low (<30)",
                    ifelse(score<35, "Normal Low (30-35)",
                        ifelse(score<65, "Normal (35-65)",
                            ifelse(score<70, "Normal High (65-70)", "High (>70)")))))
head(std.ts, 6)
```

```
> head(std.ts,6)
        id  order       date sexmale age item score          grade
1 61748341 202082 2011-03-10       1  32  RBC    56 Normal (35-65)
2 61748341 106065 2012-06-19       1  33  RBC    56 Normal (35-65)
3 61748341 153980 2012-12-05       1  33  RBC    60 Normal (35-65)
4 61748341 478382 2013-06-12       1  34  RBC    50 Normal (35-65)
5 61748341 482283 2013-12-02       1  34  RBC    56 Normal (35-65)
6 61748341 714751 2016-06-17       1  37  RBC    54 Normal (35-65)
```

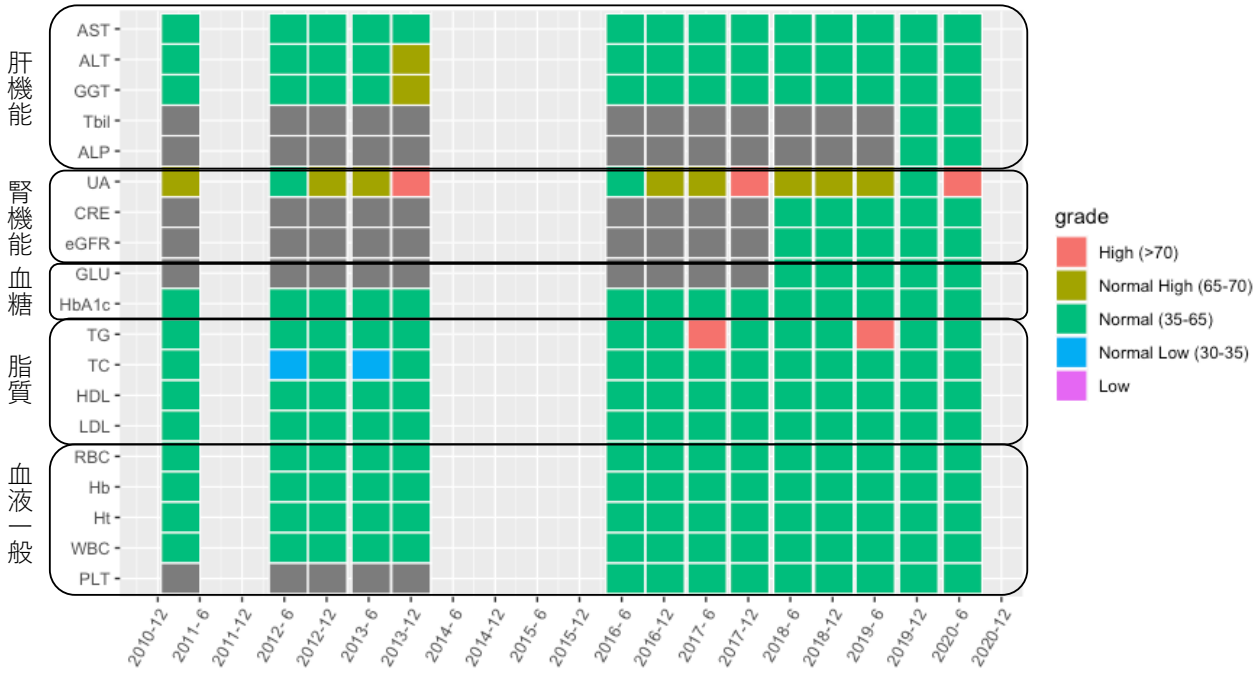
ちゃんと付与されています

STEP5: グラフ化

```
#最後はggplot2を使ったグラフ化ですが、詳細は省略します
  *ggplot2を使ったグラフ化はスライド118の参考文献（特に下２冊）をご参照ください

#コメントのみ残しておきます
#検査値のグルーピングに関するラベルを読み込む
ri.class<-ri[, 2:3]
#縦持ちの偏差値データと連結する
std.join<-dplyr::left_join(std.ts, ri.class, by="item")

#ggplot2のgeom_tile関数でヒートマップ化する
library(ggplot2); library(scales)

graph<-ggplot(std.join, aes(x=date, y=reorder(item, -class), fill=grade))
graph<-graph+
 scale_x_date(date_breaks="6 month", labels=date_format("%Y-%b"))+
 geom_tile(color="white", size=.4)+
 theme(axis.text.x=element_text(angle=60, hjust=1, vjust=1))+
 scale_fill_discrete(limits=c("High (>70)", "Normal High (65-70)", "Normal (35-65)",
                    "Normal Low (30-35)", "Low"))

graph
```

検診データの偏差値表示

パッと見て、正常高値、正常低値、異常値の判別ができる！
”患者フレンドリー“な検査結果の表現方法といえる！

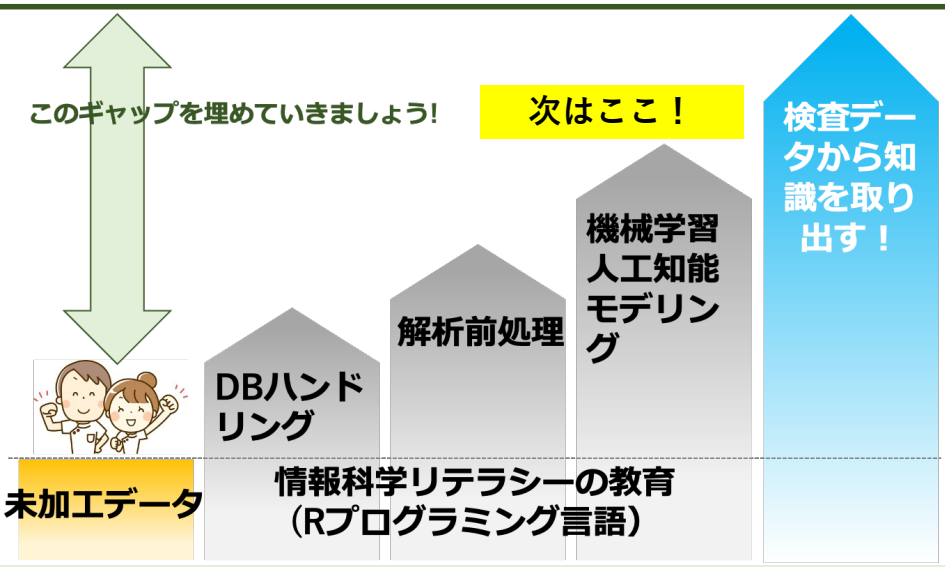


セクション6のまとめ

検診データの偏差値化

- ☐ 検診結果をよりわかりやすい形に変換したいという講演者の意図を理解した
- ☐ 検査値を正規化し、偏差値化することができる
- ☐ 偏差値結果を縦持ちに変換し、それぞれの値にグレードを付与することができる

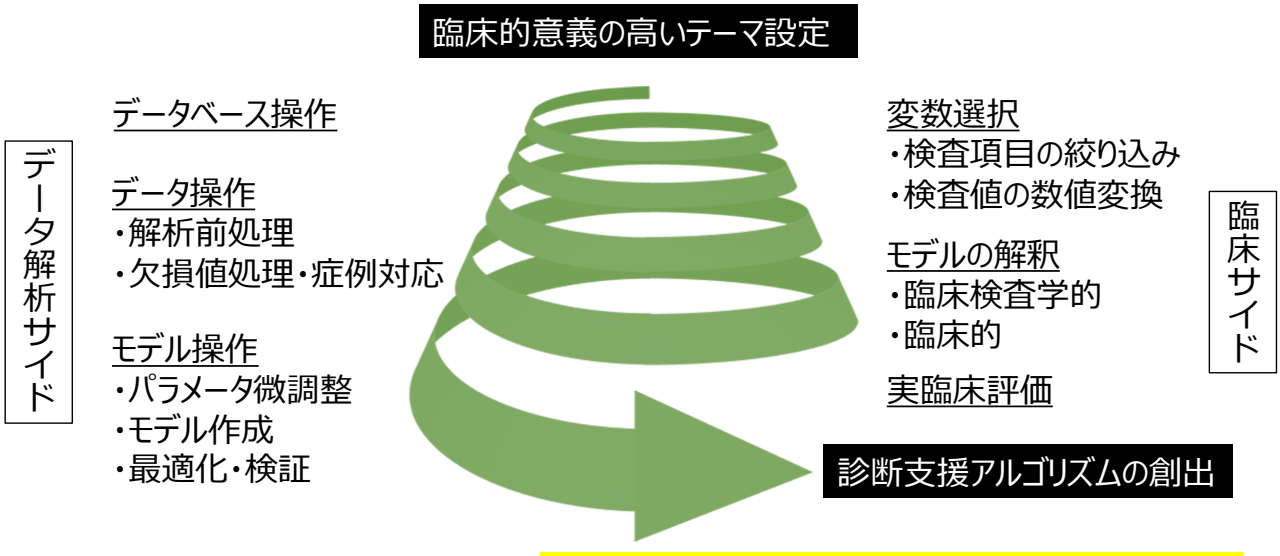
本格的な解析にむけて



- ✓ さらなる前処理法
  - 患者IDの変換法・定性値(1+, -+など）の変換法
  - 患者の除外・選別・層別化方法（病名・投薬情報の付与）
  - 欠損値処理・ケースコントロール（症例対応法）
- ✓ 機械学習の技法
  - 様々なアルゴリズムによるモデル作成・最適化・性能評価法

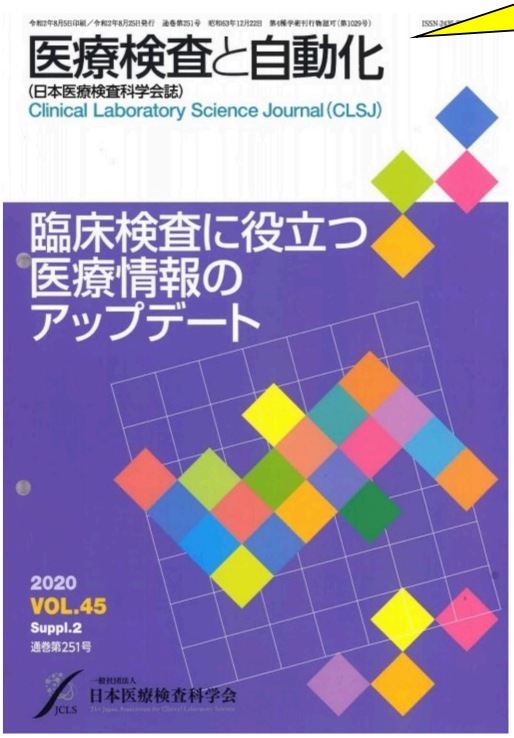
臨床検査データ解析を成功させるための3箇条

1. 臨床的意義の高いテーマを設定すること
2. 検査データの品質を高めること
3. データ解析-臨床双方のコミュニケーションを大事にすること



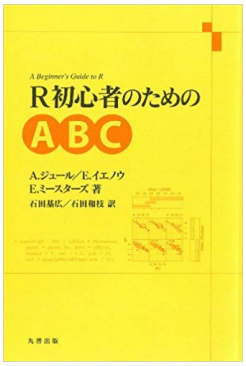
日本医療検査科学会誌　「臨床検査に役立つ医療情報のアップデート」 III-5より

8月25日に発刊されました！！



本委員会の活動成果です!
ぜひ御覧ください！

もっと知りたい人のために



9年前に読んだR本です。
著者は「**Rはあなた方が考えつくことは何でも実現できます**」と述べています。
私も同感です。

辞書代わりに重宝します

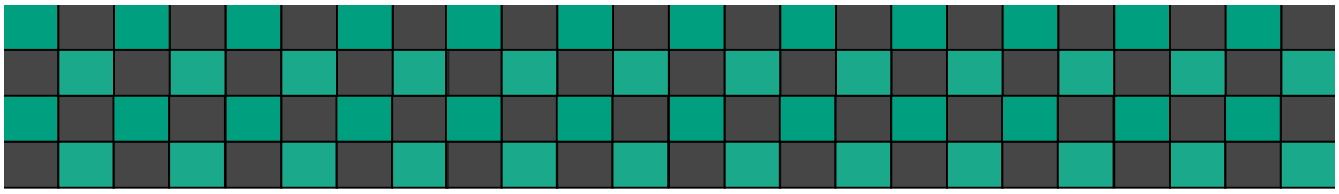
1,3章は本講習会にピッタリ
グラフ化は4章を参照してください

ggplot2を使う際、
辞書代わりに重宝します

本セミナーで使用した関数一覧（A-Z）

関数	由来	用途	出現スライド
!is.na()	標準	欠損値ではない(!=notの意)判定	74
apply	標準	並列演算処理	53,109
arrange()	dplyr	並び替える	98
as.character()	標準	文字化	69
as.data.frame()	標準	データフレーム化	70,72
as.Date()	標準	日付化（年月日）	69,110
as.numeric()	標準	数値化	69
bind_rows()	dplyr	リストの要素を縦方向に連結する	34,66,98
class()	標準	オブジェクトの型を確認する	17
colSums()	標準	列の総和を算出	51,52,72,74
factor()	標準	因子化	55
filter()	dplyr	特定の行を抽出する	24
for()	標準	ループ処理	54,82,97,108
full_join()	dplyr	データの連結	40
gather()	tidyr	縦持ち変換	42,43,110
getwd()	標準	現在の作業ディレクトリを表示	11
ggplot()	ggplot2	データの可視化	112
head()	標準	データの上位一部表示	18,99,106,109,110,111
hist()	標準	ヒストグラムを描画	81,82,84
ifelse()	標準	条件分岐	29,111
inner_join()	dplyr	データの連結	38
install.packages()	標準	パッケージのダウンロード	7
is.na()	標準	欠損値の判定	47
lapply()	標準	関数を複数のオブジェクトに適用、結果をリストで返す	50,64,65
left_join()	dplyr	データの連結	112
length()	標準	ベクトルの長さを求める	77,97,108
library()	標準	パッケージの実装	7,35,58
list.files()	標準	ファイル名の取得	63
list()	標準	リストの作成	47
log()	標準	自然対数を算出する	84,95,97,107,108
mutate()	dplyr	列の挿入	26,29,111
na.summary()	自作	変数の欠損率を算出	72
names()	標準	データの変数名を表示	19,54,67,72,74,75,81,108
nrow()	標準	行数をカウントする	74
par()	標準	グラフィック環境調整パラメータ	82,84
path.packages()	標準	パッケージの保存場所の確認	7
read_excel()	readxl	エクセルファイルの読み込み	64
read.csv()	標準	csvファイルの読み込み	12,33,35,38,43,75,94,106
round()	標準	四捨五入する	72,109
select()	dplyr	変数の選択、削除	28
setwd()	標準	作業ディレクトリの指定	11
spread()	tidyr	縦持ちデータを横持ち化	42,44,70
str()	標準	データ構造を表示	15,43,68,69,71,76,94,95,107
sum()	標準	合計値の算出	53,54
summary()	標準	データの基本統計量を表示	49,50
tail()	標準	データの下位一部表示	18
unique()	標準	ユニーク（一意）の値を返す	77
write.csv()	標準	csvファイルの書き出し	30,72

#関数の使い方は”**?関数名**”で調べてください
#tidyr, dplyr, ggplot2などはRStudioのHPに**Cheat Sheet**があります。
非常に便利ですので、ぜひ参考にしてください
https://www.rstudio.com/resources/cheatsheets/



Enjoy !

ご質問等有りましたらいつでもどうぞ

dseto@cclm.med.kyushu-u.ac.jp