dplyr の基礎

Toshiki SHIBANO

2021-02-15

目次

dplyr とは

dplyr: A Grammar of Data Manipulation
データフレームを効率よく簡単に操作することを可能にするパッケージです．内部は C++ で書かれているため，高速に動作します．%>% (パイプ演算子) という特殊な演算子を用いることで，可読性 (読み易い) に優れたコードを書くことが出来ます．

以下のコマンドを実行して dplyr をインストールしましょう

```r
# インストールする時は install.package() の文章をコメントアウトして下さい.
# dplyr パッケージのインストール
#install.packages("dplyr")
# もしくは tidyverse パッケージ (dplyr や ggplot2 など) をインストール
#install.packages("tidyverse")

# 一度インストールした後は library() 関数を使用する
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.1     v stringr 1.4.0
## v tidyr   1.1.2     v forcats 0.5.0
## v readr   1.3.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

よく使う dplyr パッケージの関数

チートシートはこちらの URL[https://github.com/rstudio/cheatsheets/blob/master/data-transformation.pdf] にあります．RStudio の GitHub にアクセスします．そこに dplyr 以外にも沢山のチートシートがあります．日本語訳も translations フォルダにあります．探してみて下さい．そしてぜひ手元に置いてください．

例を元にして，よく使う関数の紹介をします．実際に実行していくことをお勧めします

データの準備

みんな大好き iris データを用います．iris データは，それぞれの Species(setosa, versicolo, virginica) に対して，4 つの変数 (Sepal.Lentgh, Sepal.Width, Petal.Length, Petal.Width) を記録しています．
実際に見ていきましょう

```
# iris データの確認
# iris データは R にもともと入っている
# iris データフレームの上から 10 個のデータを確認
head(iris, n = 10)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa

```
# str 関数を使うことでデータの概略が分かる
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

select()

select() 関数: 指定した列 (column) のみを抽出する

```r
iris %>%
  select(Sepal.Length) %>%
  head(n = 5) # 上から 5 行を表示する
```

Sepal.Length
5.1
4.9
4.7
4.6
5.0

```r
# パイプ演算子を使わなかった場合はこうなります
head(select(iris, Sepal.Length), n = 5)
```

Sepal.Length
5.1
4.9
4.7
4.6
5.0

%>% (パイプ演算子) について
- a %>% function(b) - function(a, b)
この二つは同じことを表しています．すなわち，パイプ演算子を使うことで，a という変数に対して，引数 b の関数を実行するということを示しています．上の select 関数については，iris 変数を引数 Sepal.Length の select 関数に渡して実行する，ということになります．
初めはパイプ演算子に慣れないかもしれませんが，書いていくうちに分かるようになります．

mutate()

mutate() 関数: データフレームに列として結合させる

```r
iris %>%
  filter(Species == "versicolor") %>%
  mutate(Sepal.Length_Sepal.Width = Sepal.Length*Sepal.Width) %>%
  # Sepal.Length と Sepal.Width の掛け算した値を
  # 新たな変数 (Sepal.Length_Sepal.Width) として追加
```

```
  head(n = 5)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Sepal.Length_Sepal.Width
7.0	3.2	4.7	1.4	versicolor	22.40
6.4	3.2	4.5	1.5	versicolor	20.48
6.9	3.1	4.9	1.5	versicolor	21.39
5.5	2.3	4.0	1.3	versicolor	12.65
6.5	2.8	4.6	1.5	versicolor	18.20

filter()

filter() 関数: 指定した行 (row) を抽出する

```
iris %>%
  filter(Species == "setosa") %>%
  head(n = 5)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa

arrange()

arrange() 関数: 行を並び替える

```
iris %>%
  filter(Species == "setosa") %>%  # 種が setosa の行を抽出
  arrange(Sepal.Length) %>%  # Sepal.Length を昇順でソート
  head(n = 6)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
4.3	3.0	1.1	0.1	setosa
4.4	2.9	1.4	0.2	setosa
4.4	3.0	1.3	0.2	setosa
4.4	3.2	1.3	0.2	setosa
4.5	2.3	1.3	0.3	setosa
4.6	3.1	1.5	0.2	setosa

```
iris %>%
  filter(Species == "virginica") %>%
  arrange(desc(Sepal.Length)) %>% # Sepal.Length を降順でソート
  head(n = 6)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
7.9	3.8	6.4	2.0	virginica
7.7	3.8	6.7	2.2	virginica
7.7	2.6	6.9	2.3	virginica
7.7	2.8	6.7	2.0	virginica
7.7	3.0	6.1	2.3	virginica
7.6	3.0	6.6	2.1	virginica

summarise()

summarize() 関数: データを関数に基づいて集計する

```
# iris データの全ての個数および全ての平均を求める
iris %>%
  summarise(N = n(),   # n() 関数で個数を求める
            mena_Sepal_Length = mean(Sepal.Length) # mean() 関数で平均値を求める
            )
```

N	mena_Sepal_Length
150	5.843333

group_by()

group_by(): グルーピングする．
summarize() 関数と組み合わせることで威力を発揮します．例えば，それぞれ種ごとに平均値や標準偏差を出したい場面があると思います．それは次のように書くことが出来ます．結果を確認してください．

```
# 各種の個数，最大値，平均値，分散を求める
iris %>%
  group_by(Species) %>%
  summarise(N = n(), # n() 関数で個数を求める
            max_Sepal_Length = max(Sepal.Length), # max() 関数で最大値求める
            mean_Sepal_Length = mean(Sepal.Length, na.rm = TRUE),  # mean() 関数で平均値を求める
            sd_Sepal_Length = sd(Sepal.Length, na.rm = TRUE), # sd() 関数で標準偏差を求める
            .groups = "drop" # グループ化した後の処理．後ほど述べます．書かない方が良い時もあります
            )
```

Species	N	max_Sepal_Length	mean_Sepal_Length	sd_Sepal_Length
setosa	50	5.8	5.006	0.3524897
versicolor	50	7.0	5.936	0.5161711
virginica	50	7.9	6.588	0.6358796
補足				
mean() や sd() 関	数は R	に既に入っている関数	です．	
summarise() 関	数では	既存の関数以外にも自	作関数を使うことも出	来ます．

補足: summarise() 関数の.groups = “drop” について

.groups = “drop” を書かないと
`summarise()` ungrouping output (override with `.groups` argument)
というメッセージがコンソール画面に出力されます．これについて述べます．
gourp_by() 関数は 2 つ以上の条件でグルーピングすることも可能です．group_by() 関数でグルーピングした後に，summarise() 関数を用いて要約すると，最後にグルーピングした条件のみが解除され，それ以外のグルーピング条件は**継続したまま**となります．場合によっては，グルーピングは全て解除したい場合もあります．そこで，.groups = “drop” と書くことでグルーピング条件を全て解除することが可能です．実際に，実行してみます．

```r
# 使用するデータの作成
value <- rnorm(n = 50, mean = 0, sd = 1)
variety <- c(rep("A", 20), rep("B", 20), rep("C", 10))
treatment <- c(rep(c(rep("X", 10), rep("Y", 10)), 2), c(rep("X", 5)), c(rep("Y", 5)))
df <- data.frame(value = value,
                 variety = variety,
                 treatment = treatment)
df
```

value	variety	treatment
0.7715428	A	X
0.2071038	A	X
-0.9420244	A	X
0.1563849	A	X
0.6425689	A	X
0.2728034	A	X
0.0463920	A	X
0.0896131	A	X
-0.1888194	A	X
-0.5795880	A	X
0.6574117	A	Y
0.2014096	A	Y
1.4333835	A	Y
0.1339437	A	Y
0.0284005	A	Y
0.5682067	A	Y
0.4675586	A	Y
-0.6952491	A	Y
-0.2235065	A	Y
0.2321855	A	Y
0.5354605	B	X
-0.2844522	B	X
-0.3554012	B	X
-0.7999926	B	X
0.1845348	B	X
0.2300341	B	X
-0.5314899	B	X
-1.7560927	B	X
0.1382223	B	X

value	variety	treatment
0.1466555	B	X
-0.8442631	B	Y
-0.8262901	B	Y
-1.4069148	B	Y
-0.9020950	B	Y
0.1766304	B	Y
-0.3995852	B	Y
1.0438013	B	Y
0.7838034	B	Y
-0.7243677	B	Y
0.8263820	B	Y
1.4258338	C	X
0.0267035	C	X
-0.0292830	C	X
1.2913371	C	X
-1.3849581	C	X
0.0211683	C	Y
-1.1880092	C	Y
0.9023986	C	Y
-2.8770413	C	Y
1.7372381	C	Y

```
# .groups = "drop"を指定する
df %>%
  group_by(variety, treatment) %>%  # まずvarietyでグルーピングして，次にtreatmentでグルーピング
  summarise(N = n(),
            .groups = "drop") %>%  # グルーピングを全て解除
  mutate(prop = N/sum(N))
```

variety	treatment	N	prop
A	X	10	0.2
A	Y	10	0.2
B	X	10	0.2
B	Y	10	0.2
C	X	5	0.1
C	Y	5	0.1

```r
# ungroup() 関数でもでもグルーピングを解除できるが，警告文? は出る
df %>%
  group_by(variety, treatment) %>%
  summarize(N = n()) %>%
  ungroup() %>% # グルーピングを解除する関数
  mutate(prop = N/sum(N))
```

```
## `summarise()` regrouping output by 'variety' (override with `.groups` argument)
```

variety	treatment	N	prop
A	X	10	0.2
A	Y	10	0.2
B	X	10	0.2
B	Y	10	0.2
C	X	5	0.1
C	Y	5	0.1

```r
# .groups = "drop" を指定しない
df %>%
  group_by(variety, treatment) %>%
  summarise(N = n()) %>%
  mutate(prop = N/sum(N))
```

```
## `summarise()` regrouping output by 'variety' (override with `.groups` argument)
```

variety	treatment	N	prop
A	X	10	0.5
A	Y	10	0.5
B	X	10	0.5
B	Y	10	0.5
C	X	5	0.5
C	Y	5	0.5

各場合の割合を求めます．グルーピングを解除した場合，全体に対する割合を求めることになります．一方で解除しなかった場合，上の例では，まだ variety に対するグルーピングが残っているので，variety グループ内で割合を求めます．よって以上のような違いを確認することができます．このことに注意をしておかないと間違った集計をしてしまう可能性があります．気をつけましょう．

その他の機能

dplyr の機能はたくさんあります．しかし一度に全てを覚える必要はありません．行いたい作業を考えた時に，まずそれを効率化する方法はないか？ということを考えて，機能を調べるのが一番良いと思います．多くの場合ですでに便利な関数が実装されています．

参考文献

ウェブサイトをあげておきます．自分も目下勉強中です．

- https://dplyr.tidyverse.org
- https://dplyr.tidyverse.org/articles/dplyr.html (上のサイトで Get started 押したサイト)
- https://qiita.com/matsuou1/items/e995da273e3108e2338e
- https://www.jaysong.net/dplyr_intro/