応用プログラミング3

第7回　tidyverseによるデータハンドリング（1）

専修大学ネットワーク情報学部
田中健太

1．課題の振り返り

1

2. tidyverseの全体像

2

今回と次回、2回にわたり、tidyverseパッケージ群について、概要を紹介します。
2週かけても概要しか紹介できないくらい、パッケージの種類も、提供される関数も膨大なので、本講義で紹介した内容を踏まえ、さらに情報収集、実践をしていただきたいと思います。

2.1　tidyverseの特徴

- https://www.tidyverse.org/
- tidyverseは、「モダンなRプログラミング」を実現するためのパッケージ群
- 2020年代は、最初からtidyverseを使う前提でRを学んだ方が良いのでは
- "tidy data" を作成、処理するための機能が提供されている

3

Rの基本として紹介したように、Rの基本的な関数や「古典的」なパッケージを使えば、大抵のデータ分析を行うことができます。しかし、関数やパッケージによって書式や出力が異なるため、プログラムの可読性（見やすさ）や開発効率が低下する、という課題がありました。

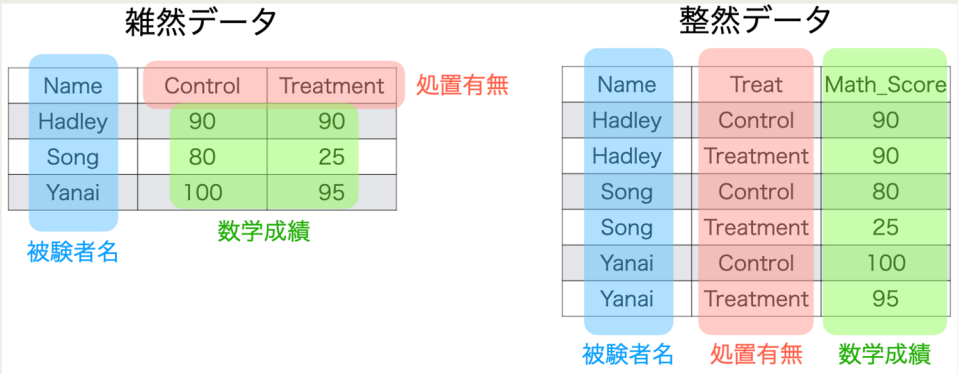
そこで、近年はより統一されたインターフェースで、見通しの良いプログラミングをするために、tidyverseというパッケージ群が開発され、世界的に広く使われています。

また、tidyverseパッケージ群を使用することを前提に、関連するパッケージも多数開発されており、現在ではRプログラミングにおいて、tidyverseを使うことが事実上の標準（デファクトスタンダード）になっています。

これからR言語について学習する上では、tidyverseを使うことを前提とした方が良いでしょう。

2.2　tidy dataとは

- 「整然データ」と訳されることもある
- 「コンピューターにとって処理しやすい」データ構造
  - 1つの変数は1つの列に
  - 1つの観測結果（レコード）は1行に
  - 個々のデータは1つのセルに



15. 整然データ構造 | 私たちのR: ベストプラクティスの探究
https://www.jaysong.net/RBook/tidydata.html

4

tidyverseを活用するうえで基本となる考え方に、"tidy data" があります。以前にも紹介しましたが、「コンピューターにとって処理しやすい、縦に長いデータ構造」です。
この考え方自体は、当然古くからさまざまなコンピューター科学者が提唱してきましたが、2014年にHadley Wickhamが発表した論文で体系化されました。

- 参考1: Wickham, H. . (2014). Tidy Data. J. Statistical Software, 59(10), 1-23. https://doi.org/10.18637/jss.v059.i10

ここでtidy dataの考え方について詳しく述べることはしませんが、日本語による優れた解説が、西原史暁氏によって公開されています。

- 参考2: 整然データとは何か | Colorless Green Ideas
https://id.fnshr.info/2017/01/09/tidy-data-intro/

私たちがRやその他のソフトウェアでデータを処理するとき、多くの場合列単位での操作を行います。その列の中に、異なる属性が混在しないように構造を整理することで、高速な検索や正確な抽出、適切な加工がしやすくなります。コンピューターが処理しやすいデータ構造にすることは、結果的に私たちの作業効率を高め、生産性を向上させます。以前にも述べましたが、これからデータを収集、記録する際にはtidyな構造になるよう意識しましょう。

2.3　tidyverseを構成するパッケージ

- ggplot2
- dplyr: データ操作、抽出、加工のためのパッケージ
- tidyr: "tidy data"を作成、操作するためのパッケージ
- readr: ファイルを柔軟に読み込むためのパッケージ
- purrr: 関数型プログラミングを実現するためのパッケージ
- tibble: データ構造tibbleを作成、操作するためのパッケージ
- stringr: 文字列を容易に、柔軟に操作するためのパッケージ
- forcats: factor型のデータを柔軟に操作するためのパッケージ
- magrittr: パイプ（%>%）演算子の機能を提供するパッケージ
- readxl: Excel形式のファイルを読み込むためのパッケージ

5

　これまでも述べているように、tidyverseは多くのパッケージからなるパッケージ群です。Hadley WickhamやRStudio社が中心となって、tidy dataの考え方にもとづく、モダンなRプログラミング環境を整備するために開発されています。

　tidyverseを構成するパッケージも、随時増加しています。また、現在の多くのパッケージがtidyverseを前提として開発されているため、「何がtidyverseに含まれていて、何が含まれていないか」という議論はあまり意味がありません。ここでは、tidyverseのWebサイトに掲載されている "Core Tidyverse" を中心に列挙しています。

　以降のページで、これらのパッケージについて紹介していきます。

2.4　パイプによる処理の効率化

- tidyverseの特徴はパイプ（%>%）
- パイプで処理をつなげていくことで、効率的なプログラミングができる

```
library(tidyverse)
# irisデータからPetal.Length, Species列を選択し、Speciesでグループ化、
# ラベル（品種）ごとに平均を算出してデータフレームに格納する
df <- iris %>% select(Petal.Length, Species) %>%
               group_by(Species) %>%
               summarise(avg = mean(Petal.Length))
df
```

```
## # A tibble: 3 × 2
##   Species      avg
##   <fct>      <dbl>
## 1 setosa      1.46
## 2 versicolor  4.26
## 3 virginica   5.55
```

```
# Native Pipeの使用例
# tidyverseを読み込まず、個別に関数を参照している
df <- iris |> dplyr::select(Petal.Length, Species) |>
              dplyr::group_by(Species) |>
              dplyr::summarise(avg = mean(Petal.Length))
```

6

tidyverseパッケージ群の特徴は多岐にわたりますが、代表的なものとして、パイプ (%>%) を使った処理の連結があります。実際には、tidyverseに含まれるmagrittrパッケージの機能ですが、パイプ演算子の左辺で実行した処理の結果が、右辺に記述した次の処理の引数として渡されます。そのため、複雑な処理を次々にパイプでつなげて実現できます。

ある関数の実行結果をオブジェクトに代入し、そのオブジェクトを引数に別の関数を実行し、といったかたちではプログラムの行数が多くなりますし、関数(関数(関数(データ))) といった記述は可読性が下がります。関数(データ) %>% 関数() %>% 関数() とパイプを活用して書けば、見通しの良いプログラムが作成できます。

tidyverseパッケージ群で提供される関数は、ほぼすべてがパイプ演算子に対応しているので、効率的なプログラミングが可能です。

なお、2021年5月にリリースされたR 4.1.0から、パッケージを必要としない標準機能として、"Native Pipe" が実装されました。これは、|> と記述することで、magrittrパッケージを（裏で）読み込まなくても、ほぼ同じ機能が利用できます。なお、細かい部分では %>% と |> には違いがあります。よくまとまった記事が以下のURLにありますので、参考にしてください。例えば、「前の関数で処理したデータ」というような意味の "プレースホルダー" と呼ばれるものが、tidyverseのパイプでは .（ドット）ですが、Native Pipeでは _（アンダースコア）になります。その他、Native Pipeはまだ開発中の機能なので、今後動作、仕様が変わる可能性も高いため、最新の情報にアクセスするようにしてください。

- 参考: R 4.2.0のリリースでにわかに盛り上がる Base Pipe "|>" とは何なのか? %>%との違いを調べました | エクセルとRでデータ分析 https://excel2rlang.com/base-pipe-r420/

3. tidyverseにおける ファイルの読み書き

7

ここからは、目的・用途別にパッケージを紹介していきます。まず、テキストファイルを読み書きするためのパッケージを紹介します。

3.1 readrパッケージによるデータの読み込み

- read_*() 関数で表構造データをより効率的に読み込める
- read_csv(), read_tsv, read_csv2() (;) などが使用できる
- 列ごとに型を柔軟に指定できる
- UTF-8以外の文字コードは指定がやや面倒
- locale = locale(encoding = "CP932") などとする

```
library(tidyverse) # library(readr)

df <- read_csv("day.csv", col_types = cols(instant = "-",
   season = col_factor(), yr = col_factor(), mnth = col_factor(),
   holiday = col_factor(),
   weekday = col_factor(levels = as.character(0:6)),
   workingday = col_factor(),
   weathersit = col_factor(levels = as.character(1:4))))
```

8

テキストファイルは、もちろんRの組み込み関数 read.csv() などで読み込めます。ただ、ファイルサイズが大きい場合や、列ごとの型を柔軟に指定したい場合などに、Core Tidyverseのreadrパッケージなどを使うと便利です。readrパッケージが提供する関数のうち、ここでは read_csv() 関数と cols() 関数を紹介します。

read_csv() 関数は、ファイル名を指定すれば、それを読み込みます。また、col_types オプションで各列の型などを指定できます。この際、cols(列名 = col_型名()) や cols(.default = col_デフォルトの型名()) というように cols() 関数を使います。型の指定には、以下のような関数を使います (一部抜粋)。短縮文字を使うこともできます。

- col_character(), c: 文字列
- col_number(), n: 数値
- col_date(), D: 年月日
- col_datetime(), T: 年月日時分
- col_factor(), f: factor型
- col_skip(), _-: 列を読み込まない

また、skip オプションを指定し、先頭から指定した行数を読み込まないこともできます。詳しくは、関数のドキュメントなどを参照してください。

- 参考1: Read a delimited file (including CSV and TSV) into a tibble — read_delim • readr https://readr.tidyverse.org/reference/read_delim.html
- 参考2: データの読み込みは{readr}にお任せを - 医療職からデータサイエンティストへ https://www.medi-08-data-06.work/entry/how_to_use_readr0224

3.1　readrパッケージによるデータの読み込み

```
df %>% head(3)
## # A tibble: 3 × 15
##   dteday     season yr    mnth  holiday weekday workingday weathersit  temp
##   <date>     <fct>  <fct> <fct> <fct>   <fct>   <fct>      <fct>       <dbl>
## 1 2011-01-01 1      0     1     0       6       0          2           0.344
## 2 2011-01-02 1      0     1     0       0       0          2           0.363
## 3 2011-01-03 1      0     1     0       1       1          1           0.196
##   atemp   hum windspeed casual registered   cnt
##   <dbl> <dbl>     <dbl>  <dbl>      <dbl> <dbl>
## 1 0.364 0.806     0.160    331        654   985
## 2 0.354 0.696     0.249    131        670   801
## 3 0.189 0.437     0.248    120       1229  1349
```

9

3.2 readxlパッケージによるExcelファイルの読み込み

- tidyverseの一部として開発されているが、別途インストールが必要
- Excelの有無によらず、Excel形式のファイルを柔軟に開ける
- `read_excel()`, `excel_sheets()` 関数などを使う
- `range = "セル指定"` オプションで読み込む範囲を指定できる

```
library(tidyverse)
library(readxl)

df <- read_excel("customer_data.xlsx", col_types = c("text", "text",
"text", "numeric", "text", "text", "text"))
```

10

Rは標準では、Excel形式のファイル（.xlsx, .xls など）に対応していません。しかし、ビジネスデータのファイル形式としては、圧倒的にExcelが使用されています（残念ながら）。そこで、Excel形式のファイルを読み込むためのパッケージが以前から存在します。ただ、それらはJava言語の実行環境を必要としたり、しばらく更新されていなかったりと不便な面があります。readxlパッケージは、tidyverseの一部（Coreではない）として活発に開発が続いており、また外部プログラムを必要としません。そのため現在Excelファイルを読み込む用途における、スタンダードになっています。

readxlパッケージでは、`excel_sheets()` 関数で、ファイルの中のシートの情報を確認できます。読み込むシートが決まったら、`read_excel()` 関数で読み込みます。シートは、シート名と左のシートから順に1, 2, 3といった整数で指定できます。

シートを読み込む際には、`col_types` オプションで列の型指定などができます。ただし、前述の `read_csv()` 関数とは異なり、Excelのデータ型に対応した `"text"`, `"numeric"`, `"date"` といった指定をします。また、`"skip"` とすることで、列を読み込まない指定もできます。また、セルの番地を用いて、読み込む範囲も指定できます。例えば、`range = "B3:D6"` といったようにします。他にも、柔軟な指定ができるので、ドキュメントを参照してください。

- 参考: Read Excel Files • readxl https://readxl.tidyverse.org/index.html

3.2 readxlパッケージによるExcelファイルの読み込み

```
df %>% head(3)
## # A tibble: 3 × 7
##   顧客番号 姓    名    年齢 居住都道府県 電話番号
##   <chr>    <chr> <chr> <dbl> <chr>        <chr>
## 1 1        佐藤  大介     22 愛媛県      08039425914
## 2 2        中村  貴之     33 北海道      09094044092
## 3 3        伊沢  朋花     20 宮崎県      09066235552
##   メールアドレス
##   <chr>
## 1 daisuke.sato@wtayyx.oeyo.bim
## 2 nakamura.takayuki@atfaw.xq
## 3 tomoka.izawa@ltgauck.qn.ij
```

3.3　writexlパッケージによるExcelファイルへの書き出し

- 分析結果のデータフレームをExcelファイルに書き出すには、writexlパッケージを使う
- writexlパッケージはtidyverseではなく、別のパッケージ群であるrOpenSciの一部
- write_xlsx() 関数で、Officeのインストールされていない環境でも出力できる

```
library(tidyverse)
library(writexl)

df <- iris %>% select(Petal.Length, Species) %>%
               group_by(Species) %>%
               summarise(avg = mean(Petal.Length))
write_xlsx(df, "iris_modified.xlsx")
```

12

次に、Rのデータフレームを Excelファイルとして出力するためのwritexlパッケージを紹介します。このパッケージはtidyverseの一部ではなく、別途さまざまなパッケージを開発・公開しているrOpenSciプロジェクトによるものです。

- 参考1: rOpenSci - open tools for open science https://ropensci.org/

使い方はシンプルで、write_xlsx() 関数にデータフレームと出力先のファイル名を指定します。他に、データフレームにExcelの数式を埋め込める xl_formula() 関数などが提供されています。ビジネスにおいては、前述のようにExcelが標準的なフォーマットとなっているので、データ分析者がRでひな形を作り、社内での展開、運用はExcelで行いたい、という場合に活用できるかもしれません。

- 参考2: Export Data Frames to Excel xlsx Format • writexl https://docs.ropensci.org/writexl/

4. dplyrパッケージによるデータハンドリング

13

ここからは、tidyverseの中でggplot2と並んで中心的な役割を担う、dplyr（ディープライヤー）パッケージの使い方を紹介していきます。

4.1　dplyrパッケージの概要

- https://dplyr.tidyverse.org/
- データ操作のための文法
  "grammar of data manipulation"
- ファイルなどから読み込んだデータを分析しやすい
  かたちに加工するための各種の関数を提供する
- magrittrパッケージが提供するパイプ %>% と
  組み合わせて、効率的で可読性の高いプログラムを
  作成できる

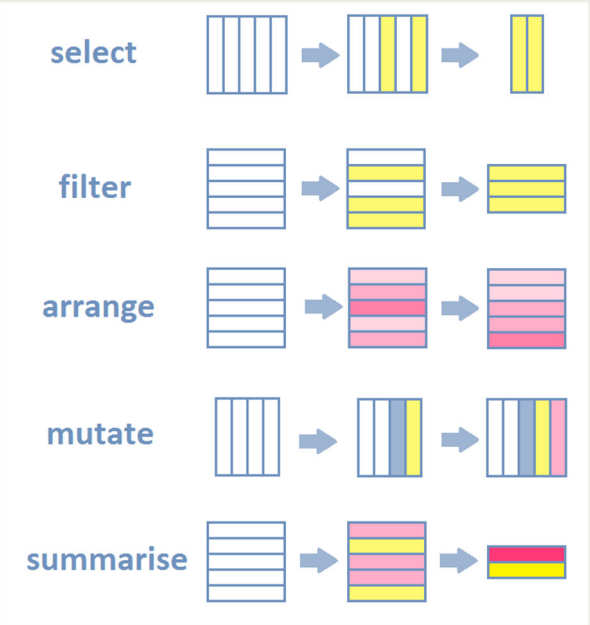
14

dplyrパッケージは、主にデータの選択（列）、抽出（行）、加工・集計などを担うパッケージです。授業や書籍などで提供される、学習のための「きれいな」データと違い、研究やビジネスの現場で扱うデータは、多くの場合、そのままでは分析できない構造・品質です。そのため、"前処理" や "Data Preprocessing"、"Data Wrangling" と呼ばれる作業を行い、「分析できる」「精度が出やすい」かたちに加工することが必要です。dplyrパッケージは、前処理における典型的な操作を、tidy dataに対して適用するための関数を提供します。

dplyrパッケージのプロトタイプとして、plyrパッケージ
(https://github.com/hadley/plyr) があり、古い書籍やWeb上の記事では、そちらを使ったプログラムが示されていることがあります。plyrパッケージ自体は現在も公開されていますが、あえて古いものを使う必要はありません。情報収集する際には、その情報がいつ作成・公開されたのかをしっかりと確認しましょう。

4.2　dplyrパッケージの主要な関数

- select(): 列を選択する
- filter(): 行を抽出する
- mutate(): 列を追加する
- summarise(): 要約する
- group_by(): グループ化する

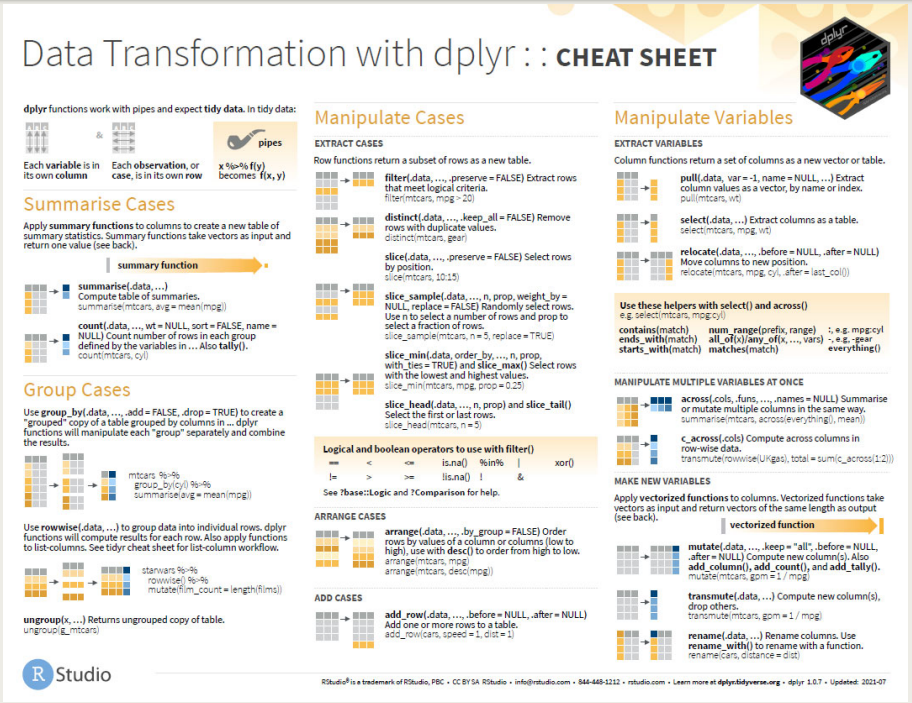


dplyr: package magique pour manipuler ses tableaux de donn es - R-atique
http://perso.ens-lyon.fr/lise.vaudor/dplyr/

15

dplyrパッケージは数多くの関数を提供していますが、ここで広く使われるものを紹介します。それぞれの関数については、次のページ以降で解説していきます。

4.2　dplyrパッケージの主要な関数



"Data transformation with dplyr cheatsheet"
https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-transformation.pdf

16

ここではdplyrパッケージに関する、Posit社が作成したチートシートを示しています。

- 参考: Posit Cheatsheets https://posit.co/resources/cheatsheets/

4.3 select() 関数による列の選択

- select(データフレーム, 列名1, 列名2,...) または データフレーム %>% select(列名1, 列名2,...) とする
- 列を削除（非選択）したい場合は -列名 とする
- 列名の指定は starts_with(), ends_with(), contains(), matches(正規表現) などで柔軟に指定できる

```
library(tidyverse) # library(dplyr)
df <- read_csv("tokyo_weather_2020.csv")
df[["年月日"]] <- lubridate::ymd(df[["年月日"]])
df %>% select(年月日, 平均気温)
## # A tibble: 366 × 2
##    年月日     平均気温
##    <date>        <dbl>
##  1 2020-01-01      5.5
##  2 2020-01-02      6.2
...
```

17

はじめに、select() 関数について紹介します。select() 関数は、条件を指定し、合致する列を選択します。シンプルな例としては、列名を指定すればその列だけ選択できます。また、選択しない列を "-列名" と明示することもできます。複数の列は "-c(列名1, 列名2)" として指定できます。

また、ヘルパー関数と呼ばれる関数を使い、柔軟に条件を指定することもできます。スライドに例示したように、starts_with(), ends_with(), contains(), matches(正規表現) などの関数で、ある文字から始まる / 終わる列、ある文字を含む列などを選択できます。他に、すべての列をあらわす everything() 関数もあります。

- 参考1: Objects exported from other packages — reexports • dplyr
  https://dplyr.tidyverse.org/reference/reexports.html
- 参考2: dplyr Tutorial => Helper Functions
  https://riptutorial.com/dplyr/example/26251/helper-functions

なお、select() 関数ではあくまで列名を指定して選択します。各列の値について条件を指定して抽出するには、次の filter() 関数を使います。

4.3 select() 関数による列の選択

```
df %>% select(-contains("気温"))
## # A tibble: 366 × 5
##    年月日      降水量の合計 日照時間 天気概況_夜          天気概況_昼
##    <date>             <dbl>    <dbl> <chr>               <chr>
##  1 2020-01-01           0        7.4 快晴                晴
##  2 2020-01-02           0        5.7 晴                  晴一時曇
##  3 2020-01-03           0        8.8 快晴                快晴
...
```

4.4 filter() 関数による行の抽出

- filter(データフレーム, 条件) または
  データフレーム %>% filter(条件) とする
- 条件は演算子（==, >, >=, <, <=, &, |, !）や
  between() 関数で指定する
- 組み込み関数にも filter() があるので注意する

```
df %>% filter(最高気温 >= 35)
## # A tibble: 12 × 8
##    年月日      平均気温 最高気温 最低気温 降水量の合計 日照時間
##    <date>         <dbl>    <dbl>    <dbl>        <dbl>    <dbl>
##  1 2020-08-07     29.5     35.4     26             0        8.8
##  2 2020-08-10     30.5     35.2     26.9           0        8.8
##  3 2020-08-11     31.7     37.3     27             0       12.5
...
```

19

filter() 関数は、データフレームから条件に合致する行を抽出する関数です。条件として、演算子を用いた比較や、between() 関数による範囲の指定などが使用できます。
between() 関数は、between(対象列, 下限, 上限) というように使用します。この時、下限、上限に指定した値は含まれます。

- 参考: Subset rows using column values — filter • dplyr
  https://dplyr.tidyverse.org/reference/filter.html

4.4 filter() 関数による行の抽出

```
df %>% filter(年月日 >= "2020-12-01" & 平均気温 >= 11)
## # A tibble: 2 × 8
##   年月日      平均気温 最高気温 最低気温 降水量の合計 日照時間 天気概況_夜
##   <date>         <dbl>    <dbl>    <dbl>         <dbl>    <dbl> <chr>
## 1 2020-12-08     11.4     17.2      5.8             0      6.7 曇
## 2 2020-12-12     11.4     15        8.3             0      2.3 晴時々曇
...

df %>% filter(between(平均気温, 20, 24))
## # A tibble: 63 × 8
##    年月日      平均気温 最高気温 最低気温 降水量の合計 日照時間 天気概況_夜
##    <date>         <dbl>    <dbl>    <dbl>         <dbl>    <dbl> <chr>
##  1 2020-05-02     20.6     26.9     15.3             0     12.2 晴後一時薄曇
##  2 2020-05-03     20.7     25.8     16.5             0      7.1 曇時々雨
##  3 2020-05-05     21.3     28.4     16.4             0      7.7 曇後時々雨
...
```

4.5 mutate() 関数による列の追加

- 既存の列の値をもとに、加工した結果を新しい列として追加したり、既存の列に上書きできる
- mutate(列名 = 処理) と記述する
- 処理は演算子や組み込み関数（cumsum() など）、dplyrの関数（lead(), lag(), if_else() など）を使って指定する

```
df <- df %>% mutate(気温差 = 最高気温 - 最低気温)
df <- df %>% mutate(日照時間 = 日照時間 * 60) # 分に変換
```

21

mutate() 関数は、データフレームの列に対して何らかの処理を適用し、結果を新しい列として追加したり、既存の列に上書きする機能を提供します。関数の処理自体はシンプルで、df[["列名"]] <- 処理 といったコードと同等ですが、tidyverseのパイプ処理の流れの中では mutate() 関数を使います。引数に、さまざまな関数を指定します。例えば、累積和を求める組み込み関数 cumsum() を指定すると、前の行までの累積和が代入されます。

また、前後の値との差分を求めたい場合、dplyrパッケージが提供する lead() (次の値)、lag() (前の値) 関数が使えます。列名 = 列名 - lag(列名) といったようにすると、前の値との差分が得られます。n オプションで、いくつ先 / 前の値を参照するかを指定できます。

他に、「1週間の移動平均」といった、区間 (ウィンドウ) をずらしながら集計する用途では、RcppRollパッケージやsliderパッケージの各種関数を組み合わせることができます。

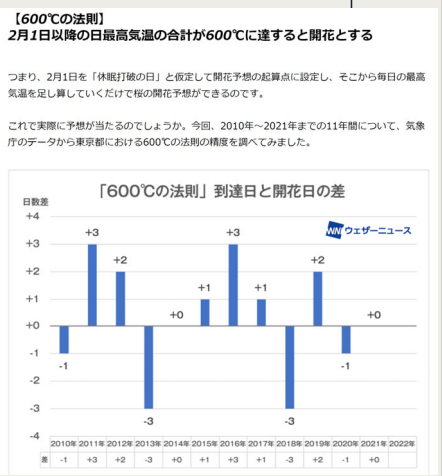
- 参考1: Create, modify, and delete columns — mutate • dplyr https://dplyr.tidyverse.org/reference/mutate.html
- 参考2: dplyr 1.0.0 時代の時系列データ処理 — 特に移動集計 — https://rstudio-pubs-static.s3.amazonaws.com/640203_28880d5e47b44029b9279d8e24385fba.html
- 参考3: dplyrを使いこなす！Window関数編 - Qiita https://qiita.com/matsuou1/items/db6e8c48fcfd791dd876
- 参考4: slider 0.1.0 https://www.tidyverse.org/blog/2020/02/slider-0-1-0/

4.5 mutate() 関数による列の追加

```
# 桜は2月1日からの累積気温が600度に達すると開花するらしい
df %>% filter(年月日 >= "2020-02-01") %>% mutate(累積気温 = cumsum(最高気温))

## # A tibble: 335 × 9
##    年月日      ... 天気概況_夜  天気概況_昼 累積気温
##    <date>          <chr>        <chr>          <dbl>
##  1 2020-02-01      晴           快晴            13.4
##  2 2020-02-02      曇後一時雨   晴              27.2
##  3 2020-02-03      晴時々薄曇   晴時々曇        42.4
...
```

開花の時期の目安がわかる　東京の桜「600℃の法則」とは - ウェザーニュース
https://weathernews.jp/s/topics/202203/190205/



4.6 summarise() 関数による要約

- summarise(列名 = 集計処理) と記述する
- 後述の group_by() と組み合わせることが多い

```
df %>% summarise(年平均気温 = mean(平均気温))

# 最頻値を算出する関数
Mode = function(x){...} # スライドでは省略

df %>% summarise(年平均気温 = mean(平均気温),
   天気概況_昼_最頻値 = Mode(天気概況_昼))
## # A tibble: 1 × 2
##   年平均気温 天気概況_昼_最頻値
##        <dbl> <chr>
## 1       16.6 晴
```

23

summarise() または summarize() 関数は、名前の通り、データを集計・要約する関数です。sとzは、前回触れたようにイギリス英語とアメリカ英語の違いで、どちらでも構いません。引数に、結果を格納する列名 = 集計処理 と記述します。集計・要約をするため、元のデータとは構造 (行・列) が変わります。

集計処理は、既存の関数や自作の関数を指定できます。欠損値の扱いなどは、集計処理の中で定義しておく必要があります。データ全体の要約をしたい場合は、直接 summarise() 関数を使えばよいですが、多くの場合、グループ (条件) ごとの集計結果を見たいので、この後紹介する group_by() 関数と組み合わせて使います。

- 参考: Summarise each group to fewer rows — summarise • dplyr
  https://dplyr.tidyverse.org/reference/summarise.html

4.7 across(), if_any(), if_all() 関数

- 複数列にまたがる関数の適用や、条件指定による抽出を行う
- across() 関数は複数列を直接指定したり、starts_with() などで指定した条件に合致する列に関数を適用する
- if_any() はいずれか1列でも、if_all() はすべての列が条件に合致する場合に TRUE を返す
- 以前は {select, filter, mutate, summarise}_{if, at, all}() があったが、across() に統一された

24

データの前処理を行う際、複数の列に対して同じ処理を適用したいことがあります。以前のdplyrパッケージでは、select(), filter(), mutate() などの関数にそれぞれ、*_if(), *_at(), *_all() などのバリエーションが用意されていました。しかし、近年ではそれらが整理され、select() などの関数はそのまま、条件 (行や列) を指定する部分で、across() や if_any()、if_all() という関数を使うようになりました。

across() 関数に指定する列は、Sepal.Length:Petal.Width といったように直接名前を指定したり、前述の starts_with() などの関数で指定できます。if_any()、if_all() 関数については、列と条件を合わせて指定します。条件指定については、~ (チルダ) や . (ドット) の使い方がポイントになります。これについては、purrrパッケージが提供する機能なので、次回あらためて紹介します。

- 参考1: Apply a function (or functions) across multiple columns — across • dplyr https://dplyr.tidyverse.org/reference/across.html
- 参考2: Across関数 | R入門コース https://r-online-course.netlify.app/post/across-function/
- 参考3: Why I love dplyr's across - Will Hipson https://willhipson.netlify.app/post/dplyr_across/dplyr_across/

4.7 across(), if_any(), if_all() 関数

```
# 「気温」を含む列の値を華氏に変換する
df %>% mutate(across(contains("気温"), ~ . * 1.8 + 32,
   .names = "{col}_華氏")) %>%
    select(contains("気温")) # 結果確認のため
```

```
## # A tibble: 366 × 6
##    平均気温 最高気温 最低気温 平均気温_華氏 最高気温_華氏 最低気温_華氏
##       <dbl>    <dbl>    <dbl>         <dbl>         <dbl>         <dbl>
##  1      5.5     10.2      3.2          41.9          50.4          37.8
##  2      6.2     11.3      1.9          43.2          52.3          35.4
##  3      6.1     12        1.4          43.0          53.6          34.5
...
```

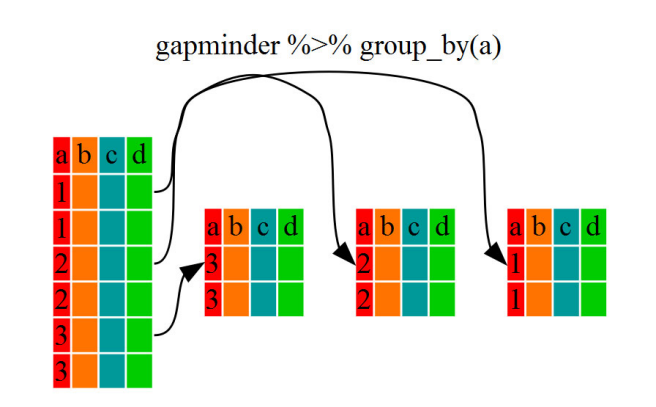
```
iris %>%
    filter(if_any(ends_with("Width"), ~ . > 4))
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.7         4.4          1.5         0.4  setosa
## 2          5.2         4.1          1.5         0.1  setosa
## 3          5.5         4.2          1.4         0.2  setosa
```

25

4.8 group_by() 関数によるグループ化

- 定性データ列の値（ラベル）に基づいて他の列をグループ化する
- group_by(グループ化変数1, グループ化変数2,...)と記述する
- グループ化するだけではあまり意味がなく、summarise() 関数などと組み合わせて使う



R for Reproducible Scientific Analysis:
dplyrによるデータフレームの操作
https://swcarpentry-ja.github.io/r-novice-gapminder/ja/_episodes/13-dplyr/index.html

26

データが何らかのグループ（条件）ごとに測定されている場合、そのグループごとに集計し、特徴を観察したいことがあります。dplyrパッケージでは、group_by() 関数を使うことで、ある列の値（定性データ）ごとにデータをグループ化し、その後の処理につなげることができます。グループ化変数は複数指定できます。

なお、何らかの理由があって、後の処理でグループ化を解除したい場合があります。例えば、グループ化して集計した値を、「全体の平均」で割りたい、という場合、グループ化されたままでは、グループ内の平均が算出され、使われます。意図した結果を得るには、グループ化を解除する必要があります。mutate() 関数については、先に ungroup() 関数を使い、解除します。summarise() 関数については、オプションとして .groups = "drop" と指定することで、すべてのグループ化を解除できます。その他、細かい制御については以下の参考URLを参照してください。

- 参考1: Group by one or more variables — group_by • dplyr
  https://dplyr.tidyverse.org/reference/group_by.html
- 参考2: dplyr 1.0.0 を使ってみる: summarise() - Technically, technophobic.
  https://notchained.hatenablog.com/entry/2020/06/28/134109

4.8 group_by() 関数によるグループ化

```
# day.csvを使用
df %>% group_by(yr, mnth) %>%
   summarise(平均気温 = mean(temp), 平均利用者数 = mean(cnt)) %>%
   mutate(前月比 = 平均利用者数 / lag(平均利用者数))

## # A tibble: 24 × 5
## # Groups:   yr [2] # summarise() 関数は最後のグループ化を解除する
##    yr    mnth  平均気温 平均利用者数 前月比
##    <fct> <fct>    <dbl>        <dbl>  <dbl>
##  1 0     1        0.198        1232. NA
##  2 0     2        0.283        1722.  1.40
##  3 0     3        0.332        2066.  1.20
...
```


27

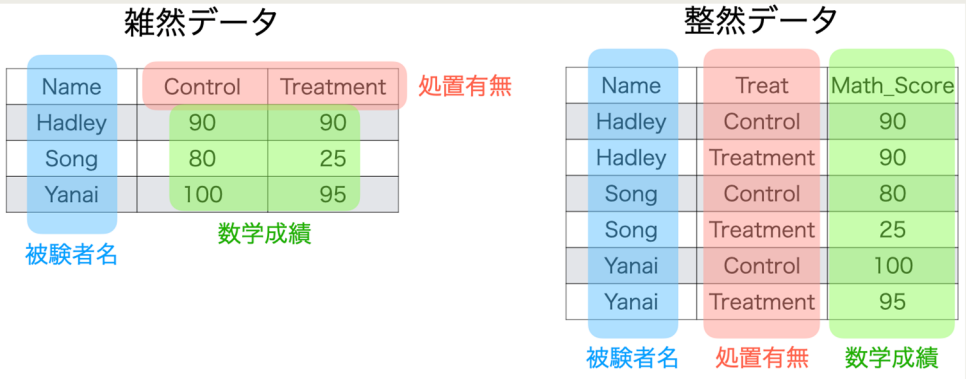
5. tidyrパッケージによるデータ構造の変換

28

次に、tidyrパッケージの機能を紹介します。tidyrパッケージは、人間にとって自然な横に長いデータ構造を、tidy dataに変換する機能を担います。

5.1　横持ちと縦持ち

- 人間にとって自然な集計済みデータが「横持ち」
- 一方で、コンピューターにとってはグループ化、集計などがしにくい
- 前述の "tidy data" の原則に沿った
  「1変数1列」のデータが「縦持ち」



tidyr入門 https://www.jaysong.net/tutorial/R/tidyr_intro.html

29

私たちは、一般的に「行」でデータを認識します。例えば、顧客情報が1行につき1人で記録され、その属性として名前、年齢、性別、過去の購入金額などが各列の値として横に並ぶような構造です。このようなデータを、(マーケティング界隈では?)「横持ちデータ」と呼びます。

(何度目かの繰り返しになりますが) 一方、Rに限らずコンピューターは極力列の少ない、縦に長いデータの扱いを得意とします。これを、「縦持ちデータ」と呼びます。

これからデータを収集するのであれば、はじめから縦持ちのtidy dataを想定してフォーマットを設計すればよいですが、すでにある横持ちデータを分析することになった場合、tidyrパッケージの機能を使い、データ構造を変換することができます。

5.2 pivot_longer() 関数による縦持ちデータへの変換

- 横持ちのデータを縦持ちに変換する
- pivot_longer(データフレーム, 変数化する列, names_to = "変数の列名", values_to = "値の列名") と記述する

```
long_iris <- iris %>% rowid_to_column("id") %>%
pivot_longer(Sepal.Length:Petal.Width,names_to="type",values_to="value")
long_iris
## # A tibble: 600 × 4
##       id Species type         value
##    <int> <fct>   <chr>        <dbl>
##  1     1 setosa  Sepal.Length   5.1
##  2     1 setosa  Sepal.Width    3.5
##  3     1 setosa  Petal.Length   1.4
...
```

30

横持ちのデータを縦持ちに変換するには、pivot_longer() 関数を使います。パイプで前の処理からデータフレームを受け取る場合、引数にははじめに縦持ちに変換したい列（複数可）を指定します。そして、names_to 引数には個々の属性を表現する列名、values_to 引数には値を記録する列名を指定します。なお、引数に指定しなかった列は、除かれるのではなくそのまま出力されます。

以前のtidyrパッケージでは、同様の機能を gather() 関数として提供していましたが、これは現在では廃止になっています（警告が出るだけで、使えなくはないですが）。古い書籍や記事を参照する際には注意しましょう。

5.3 pivot_wider() 関数による横持ちデータへの変換

- 縦持ちのデータを横持ちに変換する
- pivot_wider(id_cols = 1行にまとめるID,
  names_from = 列名にする変数名,
  values_from = 値にする変数名)

```
long_iris %>%
   pivot_wider(id_cols = "id", names_from = "type", values_from = "value")
## # A tibble: 150 × 5
##       id Sepal.Length Sepal.Width Petal.Length Petal.Width
##    <int>        <dbl>       <dbl>        <dbl>       <dbl>
##  1     1          5.1         3.5          1.4         0.2
##  2     2          4.9         3            1.4         0.2
##  3     3          4.7         3.2          1.3         0.2
...
```

31

次に、tidy dataを列方向に展開し、横持ちのデータに変換する pivot_wider() 関数を紹介します。はじめに、id_cols 引数に縦に並んだデータをひとまとめにするためのIDを指定します。次に、names_from 引数に個々の列名となるラベルが格納された列を指定します。そして、values_from 引数に値が格納された列を指定します。tidy dataの構造はシンプルなので、指定に迷うことはないでしょう。

以前のtidyrパッケージでは、同様の機能を spread() 関数として提供していましたが、これは現在では廃止になっています (警告が出るだけで、使えなくはないですが)。古い書籍や記事を参照する際には注意しましょう。

6. まとめ

32

6.1　今日の内容

- tidyverseの基礎
    - tidy dataの考え方
    - パイプによる処理の効率化
    - tidyverseを構成するパッケージ群
- readrパッケージによるデータの読み込み
- dplyrパッケージによるデータハンドリング
- tidyrパッケージによるデータ構造の変換

33

6.2　次回までの課題

- RStudio Cloudプロジェクト内の
  `07_tidyverse_01_exercise.R` について、
  指示に従ってプログラムを作成してください
- 編集したファイルは、ファイル一覧でチェックを
  入れ、[more] メニューから [Export] を選択し、
  [Download] ボタンを押してダウンロードして
  ください
- ダウンロードしたファイルを、Classroomの
  課題ページから提出してください
- **提出期限: 2022-11-07 23:59:59**

34