目次

1 Rによるデータ操作

第 5 回講義の**到達目標**は、

- 様々なデータを R で読み込むことができる。
- 読み込んだデータを確認できる。
- 基本的なデータ操作ができる。
- 整然データの構造を理解し、データを必要な形に成形できる。
- 複数のデータを結合できる。
- 作成したデータを保存できる。

第 5 回講義の**到達度検証のための課題**は、以下の通りです。

1. CSV ファイル、Excel ファイルを読み込んで、中身を確認する。
2. 必要なデータの抽出、変数の追加、変数の選択を行い、分析に適した形に持っていける。(`filter()`、`mutate()`,`select()`, `arrange()`, `pivot_longer()`, `pivot_wider()`)
3. データ結合の種類を理解し、複数のデータを結合して、1 つのデータフレームを作成する。(`bind_rows()`, `bind_cols()`, `left_join()`, `right_join()`, `inner_join()`, `full_join()`)

この章では、R を用いたデータ操作の基本的な方法を学びます。この章は何度も読み返し、繰り返し練習してください。ここでは、なぜ R を使うと便利なのかを分かってもらうために、Excel の操作と比較する形で、R のデータ操作の基本を学びます。

1.1 データの読み込み

1.1.1 CSV ファイルの読み込み

多くのプログラミング言語で、読み込むデータとして最も多いのが、CSV 形式のファイルです。ファイルの拡張子は.csv です。CSV とは、Comma Separated Values の略で、カンマで区切られたデータのことです。次のような形をしています。

```
企業 ID, 決算年月, 売上高
13,2020/03,1000
13,2021/03,1200
13,2022/03,1500
24,2020/03,2000
24,2021/03,2200
24,2022/03,2500
33,2020/03,3000
33,2021/03,3200
33,2022/03,3500
```

このように、値とコンマ, のみで構成されたファイルのため、余計な情報が入っておらず、またファイルサイズも小さく、加工が簡単なので、データのやり取りによく使われます。

さっそくファイルを読み込んでみましょう。ここでは、松浦のウェブサイトにあるデータ keshohin_2023.csv を読み込んでみます。R の場合は、read.csv という関数を使って、URL を直接指定して読み込むことができます。読み込んだデータを df という変数に代入しています。

Excel の場合は、インターネット上のデータを直接取り込むことは難しいので、いったんパソコンの中に保存してから、ファイルを開くとします。

> **R の場合**
>
> R で csv ファイルを読み込む最もシンプルな方法は、基本関数 read.csv() を用いて、ファイル名やファイルを参照する URL を直接指定することです。

```
df <- read.csv("https://so-ichi.com/kesho_2023.csv")
```

> **MS Excel の場合**
>
> 1. URLhttps://so-ichi.com/kesho_2023.csv をブラウザに入力してファイルをダウンロードし、任意の場所に保存
> 2. 「ファイル」から「開く...」をクリックして、保存した CSV ファイルを選択し「開く」をクリック

1.1.2 Excel ファイルの読み込み

MS Excel のファイルは拡張子が.xlsx、古い MS Excel だと.xls です。R で Excel ファイルを読み込むときは、read_excel という関数を使います。Excel ファイルを用意するのが面倒なので、ここではこうやれば読み込めるよ、というコードだけ説明します。ファイル名は hoge.xlsx とします。

> **!** R の場合
>
> R で MS Excel のファイルを読み込むには、readxl パッケージの read_excel() 関数を用います。
>
> ```
> dfx <- readxl::read_excel("hoge.xlsx")
> ```

> MS Excel の場合
>
> 1. 「ファイル」から「開く...」をクリックし、保存してある Excel ファイルを選択し「開く」をクリック

MS Excel の問題点は、目的のデータがどの Excel ファイルに入っていて、それがどこに保存されているのかを覚えておかないと、いちいちファイルを開いて探さないといけないことです。

R だとソースコードを残すことができますので、どこにあるファイルを読み込んで、そこに何が入っているのかをコメントで残しておくことができます。

1.2 読み込んだデータの確認

MS Excel は読み込んだデータが画面上に表として表示されていますが、R では変数に代入しただけでは、画面には何も表示されません。そこでデータの中身を確認する関数として、次のようなものがあります。

- head()：最初の数行を表示させる基本関数
- str()：データの構造を表示させる基本関数
- glimpse()：データの構造を表示させる dplyr パッケージの関数
- names()：変数名を表示させる基本関数

これらを使って、データの中身を確認し、データの形に適した処理方法を学ぶ必要があります。以下では、head() 関数を使って、データの最初の数行を表示させてから、str() 関数でデータの中の変数とその型を確認します。

Excel は目視が中心ですが、見ただけでは、文字列なのか数なのかが分からないので、やはりデータの型は確認する必要があります。

R の場合

```
head(df)
```

```
  code   name    term shubetsu ren  sales netincome month
1  641 資生堂 1985/11       10   1 371040     14526    12
2  641 資生堂 1986/11       10   1 375294     13632    12
3  641 資生堂 1987/11       10   1 378977      9014    12
4  641 資生堂 1988/11       10   1 401311      9515    12
5  641 資生堂 1989/03       10   1 130654      4265     4
6  641 資生堂 1990/03       10   1 456352     11362    12
```

```
str(df)
```

```
'data.frame':   130 obs. of  8 variables:
 $ code     : int  641 641 641 641 641 641 641 641 641 641 ...
 $ name     : chr  "資生堂" "資生堂" "資生堂" "資生堂" ...
 $ term     : chr  "1985/11" "1986/11" "1987/11" "1988/11" ...
 $ shubetsu : int  10 10 10 10 10 10 10 10 10 10 ...
 $ ren      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ sales    : int  371040 375294 378977 401311 130654 456352 517252 553299 561549 549178 ...
 $ netincome: int  14526 13632 9014 9515 4265 11362 15850 16011 13290 14668 ...
 $ month    : int  12 12 12 12 4 12 12 12 12 12 ...
```

MS Excel の場合

画面を見て確認する。

このデータには,

- code : 企業コード (文字列)
- name : 企業名 (文字列)
- term : 決算年月 (文字列)
- shubetsu : 会計基準の種類 (数値)
- ren : 連結か単体 (数値)
- sales : 売上高 (数値)
- netincome : 当期純利益 (数値)
- month : 決算月数 (数値)

が入っています。

1.3 データの整形

1.3.1 データ操作の基礎

さあ面白くなってきました。次はデータを操作していきます。R によるデータ操作では、tidyverse パッケージ群の dplyr パッケージが大活躍します。

dplyr パッケージの関数の中でもよく使うものに次のようなものがあります。

- select()：変数を選択する
- filter()：データを抽出する
- mutate()：変数を追加する
- arrange()：データを並び替える
- summarise()：データを集計する
- group_by()：データをグループ化する

1.3.2 パイプ演算子

R でソースコードを書く際に, 理解しやすく, 読みやすいコードにするために非常に便利なのが, パイプ演算子%>%です。パイプ演算子%>%は, 左側のオブジェクトを右側の関数の第一引数に渡すという処理を行います。たとえば,

```
(1 + 2) %>% sqrt()
```

```
[1] 1.732051
```

と書くと，sqrt(1 + 2) と同じ意味になります。たとえば，rnorm() 関数を使って平均 0, 分散 1 の標準正規分から 100 個のデータを作りたいとします。rnorm() 関数は 3 つの引数を取ります。

1. データの個数
2. 平均
3. 標準偏差

したがって，rnorm(100, 0, 1) と書くと, 平均 0, 分散 1 の標準正規分布から 100 個のデータを取り出すことができます。パイプ演算子を使うと,

```
100 %>% rnorm(mean = 0, sd = 1)
```

```
  [1]  0.281488930  2.334217654  0.667765877 -1.625444757  0.747930299
  [6] -0.167272634  0.250156579 -0.760214929 -1.029153086  0.221490247
 [11]  1.407880025  0.002725538 -0.197906652 -1.446391215  0.389440078
 [16]  0.707242115 -1.835972314 -0.462262016  0.189745792  0.359925971
```

```
 [21] -1.758538944 -0.660750003 -0.211715027 -1.527516138 -0.976026364
 [26]  0.028368618 -1.908937317  2.065818795 -0.383380902  0.918549464
 [31] -0.161017274 -0.853737897 -0.732956494  1.358232966  0.467783006
 [36]  1.919650459  1.280445931  0.657965097  0.510670392 -0.429442943
 [41] -2.073221876  0.901943032  0.450876300  0.196228002 -0.418926810
 [46] -0.029697614 -0.563684138 -0.264410244 -0.705116870  1.782092068
 [51]  1.435200704  0.520580485 -0.485007200  0.935502451  0.346518698
 [56] -1.406444186 -0.043501604  2.366550949  0.473035650 -0.327943569
 [61]  1.001747557  1.049112747  0.729533629  0.395605203 -1.043718055
 [66]  0.792910344  0.795056588 -0.278431389 -0.493694649 -1.486674955
 [71]  0.874922480  0.678283398  3.036084754 -0.161230773 -0.503172131
 [76] -0.140662670  0.765777052  1.088178955  1.681413109  0.738623448
 [81]  1.659309348  0.107318470  0.048612634 -0.153977014 -0.255634563
 [86] -0.544356201  0.134187622  0.256127264 -1.545381603  0.878093864
 [91] -0.137135588 -0.220185585  0.124012591 -1.096021072  0.041501739
 [96]  1.078336419 -0.065415226 -0.700745339  2.071237140 -0.515809755
```

となります。これは rnorm() 関数の第 1 引数がデータの個数なので, そこに 100 を渡しています。ここで平均に値を渡したい場合を考えます。mean 引数は第 2 引数なので, パイプ演算子では自動で渡してくれません。そこで . を使って渡す場所を指定してあげます。

```
100 %>% rnorm(100, mean =. , sd = 1)
```

```
  [1]  99.13849  99.57009  97.63354  97.93992 100.74842  98.99645  99.87391
  [8]  98.40504  99.00836 103.08457 100.39964  97.83821 101.39189  99.67665
 [15] 101.57587 100.88880 101.43428  99.11924  98.19301 100.19221  99.35759
 [22] 101.57248 100.59761 100.49504  99.46449  99.40095  99.47328 100.70865
 [29]  99.54094 101.45892 100.41077 100.40363  98.42490 100.48687 101.14750
 [36] 101.46394 100.96344  99.81582 100.64071  97.57490 100.74032 101.34378
 [43]  99.87838  98.90590 100.90028 101.29781 102.46943 100.33046  99.34633
 [50] 100.01545 100.43407 100.80851  99.63716 101.22431 100.15020 100.11720
 [57] 100.38748 100.66382  99.53273 100.96054  99.07339 101.35509 101.32225
 [64] 102.61873  99.86789 101.31107 101.92803 102.88668 100.18561  98.82133
 [71] 100.76885  99.16287  98.95447  99.48445  98.45482  99.60013  98.78961
 [78] 100.48115  99.46917  99.84121 101.48909  99.74357  99.58458  98.86173
 [85]  98.05519  99.68692  99.17234 100.55380  99.63725 100.19245  98.51727
 [92] 100.07958 100.61006 100.51796 101.11872 101.04054  98.30348  99.21155
 [99]  99.71528  99.66745
```

これで平均 100, 標準偏差 1 の正規分布から 100 個のデータを取り出せました。

これだけだとパイプ演算子%>%の便利さが伝わらないので, たとえば次のような処理を考えてみましょう。

1. 2020 年のデータを抜き出し,
2. 売上高当期純利益率を計算し,
3. 産業グループごとに平均を計算する
4. 利益率が高い順番に並び替える

をパイプ演算子を使って書くと,

```
df <- df %>%
    filter(term == "2020") %>% # 2020 年のみ
    mutate( # 新しい変数を作成
        ratio = netincome / sales # 売上高利益率
        ) %>%
    group_by(sangyo) %>% # 産業グループごとに
    summarise( # 平均を計算
        mean_ratio = mean(ratio) # 利益率の平均
        ) %>%
    arrange(desc(mean_ratio)) # 利益率の高い順に並び替え
```

のように, 上から順番に処理を実行し, 次に渡す, というプロセスが分かりやすく, 読みやすいコードができました。コメントも残しておけば, 後から見返したときにも分かりやすいですし, 他人によんでもらうときも親切ですね。したがって, 以下ではパイプ演算子を駆使して, データ操作を行っていきます。

新しい変数を作成する `mutate`

新しい変数を作成するには, `dplyr` パッケージの `mutate()` 関数を使います。先ほど読みこんだデータから, 当期純利益を売上高で除して売上高当期純利益率を計算して, `ratio` という変数を作ってみましょう。

R の場合

```
df <- df %>%
    mutate( # 新しい変数を作成
        ratio = netincome / sales # 売上高利益率
        )
```

MS Excel の場合

I1 のセルに変数名を表す `ratio` と入力する。F 列の `sale` と G 列の `netincome` を使って, I2 のセルに
`= G2 / F2`

とし，I2 セルの右下の四角をダブルクリックすると，自動で下のセルにも同じ計算がコピーされる。

次に，ある変数の値に応じて異なる値をとる変数を作るには，mutate() 関数と ifelse() 関数を同時に使います。ifelse() 関数は次のような引数を取ります。

```
ifelse(条件，条件が真のときの値，条件が偽のときの値)
```

先ほど計算した売上高当期純利益率が 5%以上ならば「高い」，そうでなければ「低い」という変数 highlow を作ってみましょう。

R の場合

```
df <- df %>%
    mutate( # 新しい変数を作成
        highlow = ifelse(ratio >= 0.05, "高い", "低い") # 売上高利益率
        )
```

MS Excel の場合

J1 セルに highlow と入力する。J2 セルに
= if(I2 >= 0.05, "高い", "低い")
と入力し，J2 セルの右下の四角をダブルクリックすると，自動で下のセルにも同じ計算がコピーされる。

Excel だとセルの移動や変数名の入力，計算式の入力，セルのコピーといった作業で，キーボードとマウスを行ったり来たりする必要があり，若干面倒です。

ついでに，mutate() 関数を使って，長すぎる企業名を短くしてみます。ここでは「ポーラ・オルビスホールディングス」を「ポーラ」と略してみます。mutate() と ifelse を使って，name 変数の値が「ポーラ・オルビスホールディング」ならば「ポーラ」という値をとる変数 name 上書きします。を作ってみましょう。

R の場合

```
df <- df %>%
    mutate( # 新しい変数を作成
        name = ifelse(
            name == "ポーラ・オルビスホールディング", "ポーラ", name) # 企業名
        )
```

データを抽出する filter

データを抽出するには，dplyr パッケージの filter() 関数を使います。filter() 関数は，次のような引数を取ります。

```
filter(データ, 条件)
```

先ほど作成した ratio2 が「高い」企業だけを抽出してみましょう。filter() 関数の中の条件は，==を使って，"高い"という文字列と一致するかどうかを確認しています。ここでは，highlow 変数の値が"高い"と一致する企業だけを抽出し，df_high という変数に代入しています。

> ❗ R の場合

```
df_high <- df %>%
    filter(highlow == "高い") # 条件
```

> 💡 MS Excel の場合
>
> highlow 変数のある J 列をクリックして枠を移動させ，上の「ホーム」メニューから「並び替えとフィルター」をクリックし，「フィルター」をクリックする。すると，変数名 highlow のヨコに漏斗のようなマークが出るので，それをクリックすると，記録されたデータの種類が出てくるので，「高い」だけにチェックが入った状態にする。

Excel のクリック回数が増えてきましたね。

filter() 関数の中で指定する条件は，

- ==：一致する
- !=：一致しない
- >=や<=：以上や以下
- >や<：より大きいや小さい
- %in%：いずれかに一致する

などがあります。またこれらの条件を組み合わせることもできます。その場合は，以下のように&や|を使います。

- &：かつ
- |：または

たとえば，資生堂と花王を抽出したり，売上高当期純利益率が 5%以上かつ売上高が 1000 億円以上の企業を抽出するには，次のように書きます。

R の場合

```
df_shiseido_kao <- df %>%
    filter(name %in% c("資生堂", "花王")) # 2 社だけ抽出
df_high2 <- df %>%
    filter(ratio >= 0.05 & sales >= 1000) # 2 条件を同時に満たす
```

変数を選択する select

データの中から必要な変数だけを選択するには，dplyr パッケージの select() 関数を使います。たとえば，先ほど作成した df から，企業コード，企業名，売上高当期純利益率の 3 つの変数だけを選択してみましょう。

R の場合

```
df3 <- df %>%
    select(code, name, ratio) # 3 つの変数だけ選択
```

MS Excel の場合

オリジナルのデータをコピーして，下のタブから別のシートを選択し，そこに貼り付ける。貼り付けたデータから code と name と ratio 以外の列を削除する。

MS Excel だと，不要なデータを削除するのが怖い作業で，必要になったときにまた元のデータを読み込まないといけないので，面倒ですし，ミスのもとです。

select() 関数の中で使えるものには，以下のようなものがあります。とても便利なので，覚えておくとよいでしょう。

- - : 除外する (-ratio とかくと ratio 以外を選択)
- : : 連続する変数を選択 (code:ren と書くと code から ren までを選択)
- starts_with() : ある文字列で始まる変数を選択
- ends_with() : ある文字列で終わる変数を選択

たとえば，mutate() で新しい変数を作る場合に，変数名に法則性をつけておけば，starts_with() を使って一気に変数を選択することができます。たとえば，比率を表す変数は ratio で始まるように統一しておく，基準化した変数には_K を最後に付けておく，などです。

データを並び替える arrange

データを並び替えるには，dplyr パッケージの arrange() 関数を使います。たとえば，先ほど作成した df から，売上高当期純利益率を並び替えてみましょう。

❗ R の場合

```
df %>%
    select(name, ratio) %>% # 2 つの変数だけ選択
    arrange(ratio) %>%
    head()
```

```
  name       ratio
1 ポーラ -0.43495809
2 資生堂 -0.07576384
3 資生堂 -0.03859062
4 資生堂 -0.02166802
5 資生堂 -0.01384122
6 資生堂 -0.01266169
```

小さい順に並び替えられました。大きい順にするには，desc() 関数を使います。ついでに knitr パッケージの kabble() 関数で表を見やすく加工してみます。

❗ R の場合

```
df %>%
    select(name, ratio) %>% # 2 つの変数だけ選択
    arrange(desc(ratio)) %>%
    head(10) %>% # 先頭の 10 行
    knitr::kable(booktabs = TRUE) # 表をきれいに表示
```

name	ratio
ポーラ	0.1110647
花王	0.1019213
花王	0.0987028
花王	0.0986613
ユニ・チャーム	0.0929384
花王	0.0912752
ポーラ	0.0895507
ユニ・チャーム	0.0891383

ユニ・チャーム　0.0890311
ユニ・チャーム　0.0869777

これでどの企業のどの年度の売上高当期純利益率が大きいのかが一目瞭然になりました。

MS Excel だと,

MS Excel の場合

「ホーム」メニューから「並び替えとフィルター」をクリックし,「昇順」をクリックする。
必要なデータだけ選択してコピペすれば, 表が完成します。

となります。簡単ですが, MS Excel の並び替えは注意が必要で, 並び替えた後にデータを追加すると, 並び替えが解除されてしまい, 元に戻せなくなったり, 空列があると並び替えがうまくいかなかったりします。

1.3.3 long 形式と wide 形式

人間には読みやすいけれどパソコンは読みにくい, というデータの形式があります。例えば下の表を見てみましょう。

地点	6 時	12 時	18 時
札幌	12 ℃	15 ℃	13 ℃
大阪	20 ℃	24 ℃	22 ℃
福岡	23 ℃	25 ℃	25 ℃

このような形のデータをワイド形式 (wide) といいます。天気予報で見かけそうなこの表は, 人間にとっては分かりやすいですが, 実はコンピュータにとっては, 分かりにくいものです。コンピュータが理解しやすいデータとして表すなら, 次のような表になります。

地点	時間	気温 (℃)
札幌	6 時	12
札幌	12 時	15
札幌	18 時	13
大阪	6 時	20
大阪	12 時	24
大阪	18 時	22
福岡	6 時	23
福岡	12 時	25
福岡	18 時	25

このような形式のデータをロング型 (long) といいます。このロング型のうち, 一定のルールに従って作成されたデータを**整然データ (tidy data)** といい, R では, この整然データを扱うことが多いです。

R 神 Hadley Wickham 氏は, データの型を理解することを, データ分析の第一歩とし, その一貫として整然データという考え方を提唱しています。整然データとは, 次のような原則に従って構築されたデータのことです (Wickham, 2014) 参考 https://id.fnshr.info/2017/01/09/tidy-data-intro/。

1. 個々の変数 (variable) が 1 つの列 (column) をなす。
2. 個々の観測 (observation) が 1 つの行 (row) をなす。
3. 個々の観測の構成単位の類型 (type of observational unit) が 1 つの表 (table) をなす。
4. 個々の値 (value) が 1 つのセル (cell) をなす

上の表は, 地点, 時間, 天気, 気温の 4 つの変数があり 1 つの列をつくっています (ルール 1)。大阪 12 時の天気は雨, 気温は 12 ℃といったように 1 つの行が 1 つの観測を表しています (ルール 2)。このデータには種類の異なる観測はない (ルール 3)。また, 各セルには 1 つの値が入っています (ルール 4)。よって, これが整然データとなります。

上のロング型の天気データを使って, ロングからワイド, ワイドからロングの操作を学びましょう。

まずデータを作ります。

```
df_weather <- data.frame(
    place = c("札幌","札幌","札幌","大阪","大阪","大阪","福岡","福岡","福岡"), # 各地を 3 個ず
    time = rep(c("6 時", "12 時", "18 時"),3),
    temp = c(12,15,13,20,24,22,23,25,25)
)
print(df_weather)
```

```
  place time temp
1  札幌  6 時   12
2  札幌 12 時   15
3  札幌 18 時   13
4  大阪  6 時   20
5  大阪 12 時   24
6  大阪 18 時   22
7  福岡  6 時   23
8  福岡 12 時   25
9  福岡 18 時   25
```

これはロング型の整然データとなります。

ロングからワイド pivot_wider

R で使うならこのままでよいのですが, あえてこれをワイド型に変えてみましょう。

教科書で使用されている spread() は「根本的に設計ミスってた」と公式で発表されているので, R 神が作った pivot_wider() を使います。wider という名前の通り, ワイド型に変換する関数です。

pivot_wider() の引数は，names_from と values_from です。names_from は, ワイド型に変換するときに, どの変数を列にするかを指定します。values_from は, ワイド型に変換するときに, どの変数の値を使うかを指定します。

以下のコードでは，time 変数の値を列に，temp 変数の値を値にして，df_wide という変数に代入しています。

```
df_wide <- df_weather %>%
    pivot_wider(names_from = time, values_from = temp)
print(df_wide)
```

```
# A tibble: 3 x 4
  place `6 時` `12 時` `18 時`
  <chr>  <dbl>   <dbl>   <dbl>
1 札幌      12      15      13
2 大阪      20      24      22
3 福岡      23      25      25
```

これでワイド型に変換できました。

ワイドからロング pivot_longer

次に, このワイド型のデータをロング型に変換してみます。教科書では，tidyr の gather() を使っていますが, これも wider() と同じ問題を持っているので，R 神による pivot_longer() を使います。

pivot_longer() の引数は，cols と names_to と values_to です。

- cols は, ロング型に変換するときに, どの変数を行にするかを指定
- names_to は, ロング型に変換するときに, どの変数の値を使うかを指定
- values_to は, ロング型に変換するときに, どの変数の値を使うかを指定

以下のコードでは，6 時，12 時，18 時の 3 つの変数を行に，time という変数の値を列に，temp という変数の値を値にして，df_long という変数に代入しています。

```
df_long <- df_wide %>%
    pivot_longer(
```

```
        cols = c("6時", "12時", "18時"), # 縦にする変数
        names_to = "time", # 縦にした変数名
        values_to = "temp") # 値
print(df_long)
```

```
# A tibble: 9 x 3
  place time   temp
  <chr> <chr> <dbl>
1 札幌  6時       12
2 札幌  12時      15
3 札幌  18時      13
4 大阪  6時       20
5 大阪  12時      24
6 大阪  18時      22
7 福岡  6時       23
8 福岡  12時      25
9 福岡  18時      25
```

元のロング型に戻りました。

1.3.4 データの結合

別々のデータを結合させて使いたいことはよくあります。例えば, 次のようなデータを結合させる場合を考えてみましょう。

1.3.4.1 * 表 A

name	term	sale
トヨタ	2020	1000
トヨタ	2021	900
トヨタ	2022	1400
ホンダ	2020	800
ホンダ	2021	700
ホンダ	2022	900

```
df_A <- data.frame(
    name = c("トヨタ", "トヨタ", "トヨタ", "ホンダ", "ホンダ", "ホンダ"),
    term = c(2020, 2021, 2022, 2020, 2021, 2022),
    sale = c(1000, 900, 1400, 800, 700, 900)
```

```
)
```

1.3.4.2 * 表 B

name	term	sale
日産	2020	400
日産	2021	500
日産	2022	900
マツダ	2020	300
マツダ	2021	400
マツダ	2022	200

```
df_B <- data.frame(
    name = c("日産", "日産", "日産", "マツダ", "マツダ", "マツダ"),
    term = c(2020, 2021, 2022, 2020, 2021, 2022),
    sale = c(400, 500, 900, 300, 400, 200)
)
```

1.3.4.3 * 表 C

name	term	netincome
トヨタ	2020	100
トヨタ	2021	90
トヨタ	2022	150
ホンダ	2020	140
ホンダ	2021	100
ホンダ	2022	90
スバル	2020	30
スバル	2021	35
スバル	2022	50

```
df_C <- data.frame(
    name = c("トヨタ", "トヨタ", "トヨタ", "ホンダ", "ホンダ", "ホンダ", "スバル", "スバル", "
    term = c(2020, 2021, 2022, 2020, 2021, 2022, 2020, 2021, 2022),
    netincome = c(100, 90, 150, 140, 100, 90, 30, 35, 50)
)
```

この 3 つのデータを結合させる場合を考えます。まず表 A と表 B は同じ変数をもつデータなので, これらを結合させるには, 縦につなげる必要があります。このような結合を**縦結合**とか連結といいます。縦結合は，dplyr パッケージの bind_rows() 関数を使います。

```
df_AB <- bind_rows(df_A, df_B)
print(df_AB)
```

```
     name term sale
1   トヨタ 2020 1000
2   トヨタ 2021  900
3   トヨタ 2022 1400
4   ホンダ 2020  800
5   ホンダ 2021  700
6   ホンダ 2022  900
7     日産 2020  400
8     日産 2021  500
9     日産 2022  900
10 マツダ 2020  300
11 マツダ 2021  400
12 マツダ 2022  200
```

縦に結合できたので, トヨタ, ホンダ, 日産, マツダのデータが入ったデータベース df_AB ができました。

次に, この df_AB と df_C を結合させます。df_C は netincome という df_AB にはない変数があり, 異なる変数をもつデータ同士の結合となります。これらを結合させるには, 横につなげる必要があります。このような結合を**結合**といいます。

結合には,

- **内部結合** (inner join)
- **外部結合** (outer join)

があり, 外部結合には,

- **完全結合** (full join)
- **左結合** (left join)
- **右結合** (right join)

があります。

内部結合は**両方のデータベースに存在する観測値のみを保持**するため, 多くのデータが欠落することになりますが，**外部結合**は、少なくとも 1 つのテーブルに存在する観測値を保持するので, 大部分のデータが欠落することにはなりません。

3 つの外部結合の特徴は次の通りです。

- **完全結合**は、x と y のすべての観測値を保持します。
- **左結合**は、x のすべての観測値を保持します。
- **右結合**は、y のすべての観測値を保持します。

R 神の神書籍 R for Data Science (2e) の図がわかりやすいので, ここで紹介します。

内部結合と 3 つの外部結合をベン図で表すとこうなります。

最もよく使われる結合は**左結合**です。元データに他のデータを結合する場合, 元データに含まれるデータのみ保持したい場合が多いので, 追加データを調べるときはいつもこれを使います。左結合はデフォルトの結合であるべきで、他の結合を選択する強い理由がない限り、これを使用します。

では，`df_AB` と `df_C` を左結合してみましょう。結合する際にキーとなる変数を指定する必要があります。ここでは `name` と `term` の 2 つの変数をキーとして指定します。こうすることで，`name` と `term` が一致する観測値を結合します。

```
df_left <- df_AB %>%
    left_join(df_C, by = c("name", "term"))
print(df_left)
```

```
   name term sale netincome
1  トヨタ 2020 1000       100
2  トヨタ 2021  900        90
3  トヨタ 2022 1400       150
4  ホンダ 2020  800       140
5  ホンダ 2021  700       100
6  ホンダ 2022  900        90
7    日産 2020  400        NA
8    日産 2021  500        NA
9    日産 2022  900        NA
10 マツダ 2020  300        NA
11 マツダ 2021  400        NA
12 マツダ 2022  200        NA
```

`df_AB` にはトヨタ, ホンダ, 日産, マツダのデータがありますが，`df_C` には日産とマツダのデータがなく, スバルのデータがあります。そのため左結合すると, 日産とマツダの `netincome` には `NA` が入り, スバルは欠落します。

`df_AB` と `df_C` を右結合してみましょう。

```
df_right <- df_AB %>%
    right_join(df_C, by = c("name", "term"))
print(df_right)
```

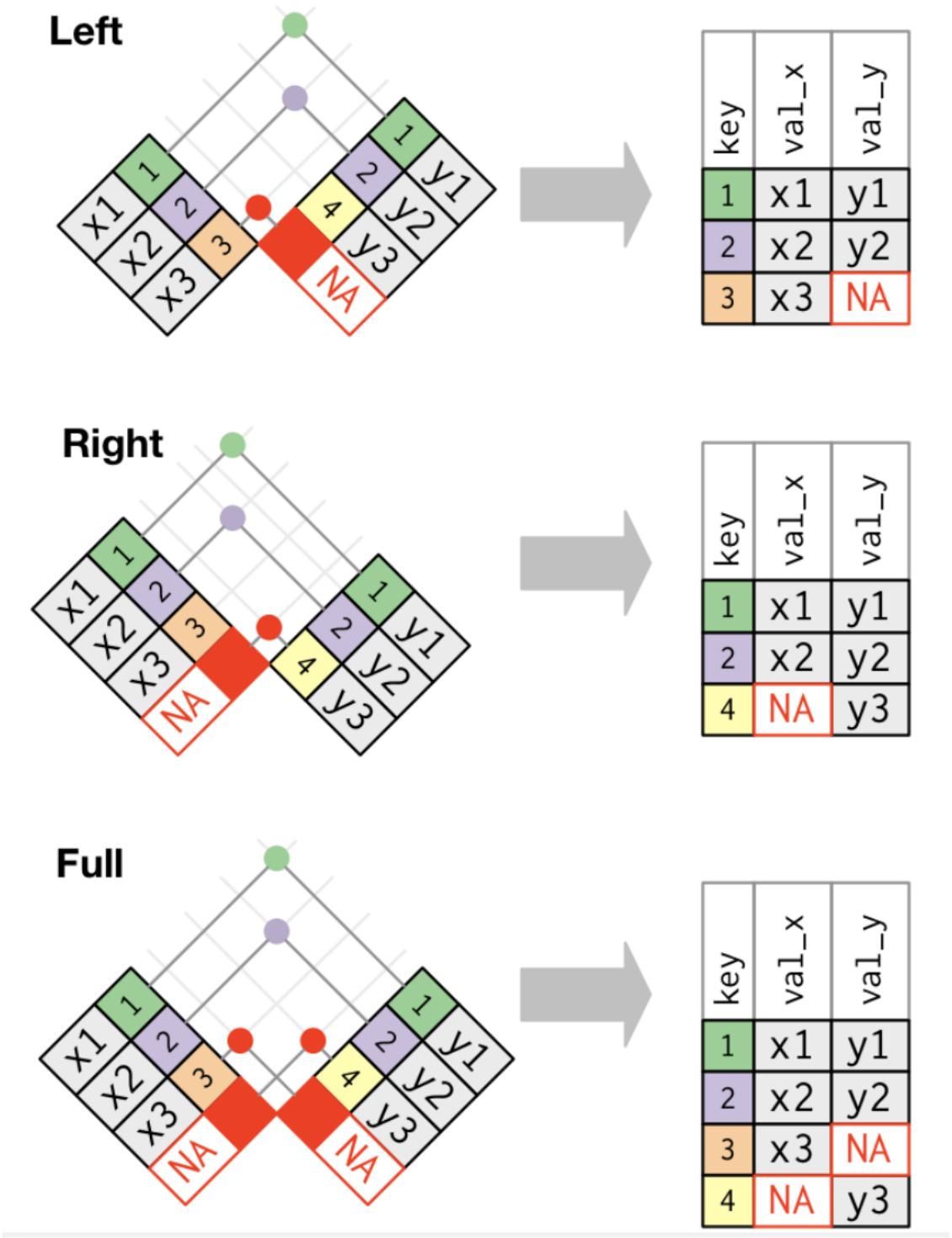
Left
x1
x2
x3
y1
y2
y3
NA
key
val_x
val_y
1 x1 y1
2 x2 y2
3 x3 NA
Right
x1
x2
x3
NA
y1
y2
y3
key
val_x
val_y
1 x1 y1
2 x2 y2
4 NA y3
Full
x1
x2
x3
NA
NA
y1
y2
y3
key
val_x
val_y
1 x1 y1
2 x2 y2
3 x3 NA
4 NA y3

図 1: 外部結合の例

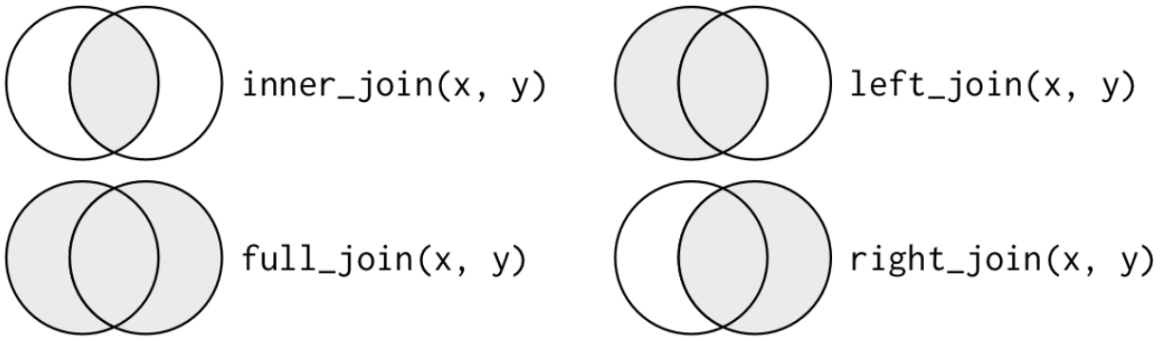

図 2: 外部結合のベン図

```
    name term sale netincome
1 トヨタ 2020 1000       100
2 トヨタ 2021  900        90
3 トヨタ 2022 1400       150
4 ホンダ 2020  800       140
5 ホンダ 2021  700       100
6 ホンダ 2022  900        90
7 スバル 2020   NA        30
8 スバル 2021   NA        35
9 スバル 2022   NA        50
```

df_C には日産とマツダのデータがなく, トヨタとホンダとスバルのデータがあります。そのため右結合すると日産とマツダのデータが欠落し，df_C に含まれていたトヨタ, ホンダ, スバルのデータが残ります。しかしスバルの sale には NA が入ります。

最後に，df_AB と df_C を完全結合してみましょう。

```r
df_full <- df_AB %>%
    full_join(df_C, by = c("name", "term"))
print(df_full)
```

```
    name term sale netincome
1  トヨタ 2020 1000       100
2  トヨタ 2021  900        90
3  トヨタ 2022 1400       150
4  ホンダ 2020  800       140
5  ホンダ 2021  700       100
6  ホンダ 2022  900        90
7    日産 2020  400        NA
8    日産 2021  500        NA
```

```
9     日産 2022   900         NA
10 マツダ 2020   300         NA
11 マツダ 2021   400         NA
12 マツダ 2022   200         NA
13 スバル 2020    NA         30
14 スバル 2021    NA         35
15 スバル 2022    NA         50
```

df_AB にはトヨタ, ホンダ, 日産, マツダのデータがありますが, df_C にはトヨタ, ホンダ, スバルのデータがあるため, 完全結合した df_full にはすべての企業のデータが入ります。しかし, 日産とマツダの netincome には NA が入り, スバルの sale にも NA が入ります。

このように, 結合するデータによって, 結合したデータに含まれるデータが変わるので, 自分が望む結合後のデータの形を考えて, どの結合を使うかを選ぶ必要があります。

ついでに内部結合もやってみましょう。

```r
df_inner <- df_AB %>%
    inner_join(df_C, by = c("name", "term"))
print(df_inner)
```

```
  name term sale netincome
1 トヨタ 2020 1000       100
2 トヨタ 2021  900        90
3 トヨタ 2022 1400       150
4 ホンダ 2020  800       140
5 ホンダ 2021  700       100
6 ホンダ 2022  900        90
```

予想どおり, 両方のデータに含まれているトヨタとホンダだけが残り, 片方のデータにしか含まれていない日産, マツダ, スバルのデータは欠落してしまいました。このように内部結合は, 両方のデータに存在する観測値のみを保持するため, 多くのデータが欠落することになり, 利用する機会があまりないです。

1.4 データの保存

前処理が終わったデータは, ファイルとして保存しておくとよいでしょう。たとえば, df_left を df_left.csv というファイル名で保存するには, readr パッケージの write_csv() 関数を使います。

write_csv() 関数の第 1 引数は保存したいオブジェクト (ここでは df_left) で, あとの主要な引数は,

- file

- na = "NA"
- append = FALSE

となります。file は保存するファイル名を指定します。na は欠損値をどうするかを指定します。デフォルトでは NA となっています。append は, 既存のファイルに追記するかどうかを指定します。基本は上書きなので, FALSE にしておきます。

```
write_csv(df_left, file = "df_left.csv")
```

これで, 作業ディレクトリに df_left.csv が保存されました。分析を進める際は, このようにして保存したデータを読み込んで使います。