R による探索的データ分析入門 - 変換

発電基盤開発課　高津一誠

2018 年 *11* 月 *28* 日

1　変換とは

前回は可視化について勉強しましたが、データの各変数（列）の値をそのまま使用しました。これは実験結果の生データがそのまま評価対象だった場合の処理と考えることができます。これに対して、生データから変換したデータが評価対象になることも多くあります。例えば、直接計測が難しいものを間接的に計測する場合や、空間的なバラツキが大きいときに複数の計測点の平均値を使いたい場合など、様々な場合があります。今回はこのような場合に使用する変換の方法を学びます。ただしデータを変換する場合には、誤差の扱いに注意が必要です。（統計の勉強会で既に扱った不確かさの伝播の問題ですが、ここでは触れません。）

2　様々な変換処理

R でのデータ変換処理は、tidyverse パッケージに含まれる dplyr パッケージを使用します。データ変換処理の一覧は、メニュー [Help -> Cheetsheets -> Data Transformation with dplyr] で確認することができますが、今回はこの中で最もよく使うものを紹介します。

表1: よく使う変換処理

処理	説明
`filter()`	フィルタ：条件に合った行を抽出する。
`arrange()`	並び替え：指定した変数（列）で並び替える。
`select()`	選択：変数（列）を絞り込む。
`mutate()`	変換：新しい変数（列）を作ったり、いまある変数を書き換えたりする。
`summarise()`	要約：変数（列）のすべての値（行）を 1 つの値に要約する。
`group_by()`	グループ化：変数（列）の値でデータをグループ化する。

今回使用するデータは、パッケージ `nycflights13` に入っているデータ `flights` で、ニューヨークの空港の 2013 年の離発着データです。パッケージをインストールしていない方は、まずインストールしてください。

```
flights <- nycflights13::flights
flights
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
```

```
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

2.1 フィルタ

フィルタは指定した条件に合った行を抽出する処理で、Excel でも似た機能（下表）があり、オートフィルタはよく使っていると思います。しかし、オートフィルタでは列ごとに指定した条件が AND 条件（すべての条件が満たされた行が残る）にしかならない点が、フィルタオプションでは若干使いにくい点が難点です。そしてどちらの機能でも、様々なフィルタ条件での結果をすべて残したいなら、シートごとコピーするくらいしか方法がありません。

表2: Excel のフィルタ機能

名前	説明
オートフィルタ	ヘッダ行に追加されたドロップダウンメニューで、フィルタ条件を指定できる（AND 条件のみ）
フィルタオプション	列ごとのフィルタ条件を、データと別のセルに書く（行内の条件は AND、行間の条件は OR）

R ではもっとスマートにフィルタをかけることができます。`filter()` で、フィルタの条件式を指定します。条件式は変数（列）名と比較演算子を使って記述し、複数の条件式を論理演算子を使って組み合わせることができます。結果を他の処理にも使いたい場合は、名前をつけて保存することができます。

例 1：出発遅延時間 `dep_delay` が 120 分より長いものを抽出し、可視化により確認する。

フィルタをかけた結果に `result` という名前をつけて保存し、その結果を可視化[*1]しています。結果を一つの処理にしか使わない場合は、直接パイプ演算子`%>%` でフィルタ処理 `filter()` を別の処理（この場合は可視化

[*1] 確認しやすいようグラフにいくつかの工夫をしています。`geom_histogram()` で指定している `boundary` は階級の境界値です。`scale_x_continuous()` で指定している `breaks` は軸目盛りで、1〜11 の連続する整数の 120 倍を目盛りに指定しています。

処理 ggplot()）とつなげば OK です。

```
result <- flights %>% filter(dep_delay > 120)
result %>%
  ggplot(aes(x = dep_delay)) +
  geom_histogram(binwidth = 40, boundary = 120) +
  scale_x_continuous(breaks = 120 * 1:11)
```

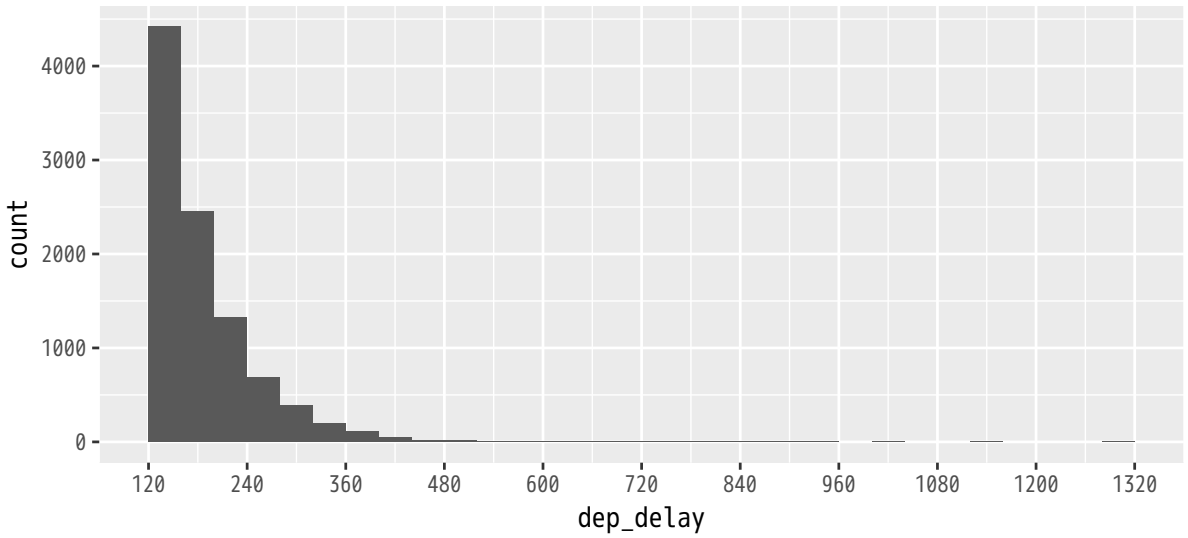

図1　フィルタの例 1

表3: 比較演算子

演算子	説明
>	大きい
>=	以上
<	小さい
<=	以下
==	等しい
!=	等しくない
%in%	含む
%between%	以上、以下

表4: 論理演算子

演算子	説明
&	AND

演算子	説明
|	OR
!	NOT

複数の条件式を組合せた例をいくつか紹介します。

例 2：出発遅延時間が 120 分以上 280 分以下のデータを抽出する。

条件式を以下のように組合せれば、目的とするフィルタ条件を作成することができます。
また、「以上 AND 以下」の組合せの場合、`data.table` パッケージを読み込むと使える`%between%` 演算子を使うと簡単に書けます。

```
flights %>% filter(dep_delay >= 120 & dep_delay <= 280) %>%
  ggplot(aes(x = dep_delay)) +
  geom_histogram(binwidth = 40, boundary = 0) +
  scale_x_continuous(breaks = 40 * 3:7)
```

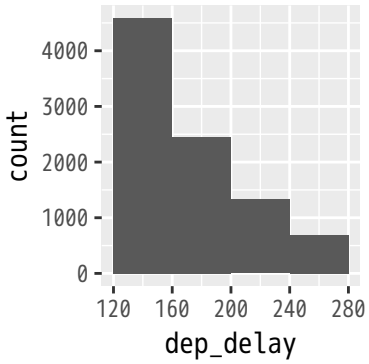

図2　フィルタの例 2

```
library(data.table)
flights %>% filter(dep_delay %between% c(120, 280)) %>%
  ggplot(aes(x = dep_delay)) +
  geom_histogram(binwidth = 40, boundary = 0) +
  scale_x_continuous(breaks = 40 * 3:7)
```

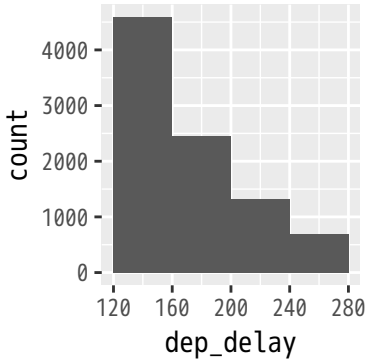

図3　フィルタの例 2

例 3 : 航空会社がアメリカン航空、デルタ航空、ユナイテッド航空のいずれかのデータを抽出する。

条件式を以下のように組合せれば、目的とするフィルタ条件を作成することができます。
また、この場合は、`%in%` 演算子を使うと簡単に書けます。

```
flights %>% filter(carrier == "AA" | carrier == "DL" | carrier == "UA") %>%
  ggplot(aes(x = carrier)) +
  geom_bar()
```

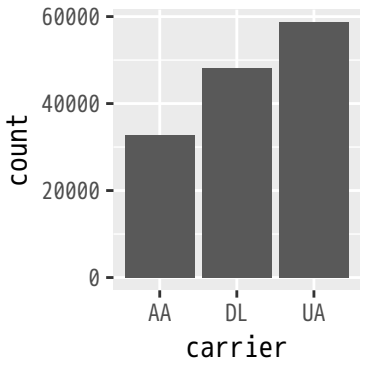

図4　フィルタの例 3

```
flights %>% filter(carrier %in% c("AA", "DL", 'UA')) %>%
  ggplot(aes(x = carrier)) +
  geom_bar()
```

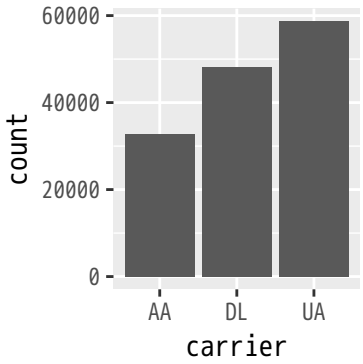

図5　フィルタの例 3

2.2　並べ替え

並び替えは指定した変数（列）でデータを並び替える処理で、Excel でも同等の機能があるので分かりやすいと思います。R では `arrange()` で、並び替えに使用したいキー変数名を指定するだけです。複数の変数を指定するときはカンマで区切ります。また降順にしたいときは、変数に `desc()`（降順：descending order の略）を適用した結果を指定します。

例：出発遅延時間と到着遅延時間で並び替える。

まず出発遅延時間の昇順で並び替え、出発遅延時間が同じものは到着遅延時間の降順で並び替えます。また、データ数が多いので、あらかじめフィルターをかけ、出発遅延時間が 380〜410 分のデータに絞り込んでいます。可視化による確認は、以下の 2 つの点で確認できるように工夫しています。

1. `geom_path()` はデータの並び順に線を引くので、線が横軸の昇順と縦軸の降順に並んでいれば OK です。
2. 並び替えた後に行番号をつけ、行番号を色に割当てているので、点の色が横軸の正順と縦軸の逆順のグラデーションになっていれば OK です。

```
flights %>%
  filter(dep_delay %between% c(380, 410)) %>%
  arrange(dep_delay, arr_delay %>% desc) %>%
  rowid_to_column("id") %>%
  ggplot(aes(x = dep_delay, y = arr_delay, color = id)) +
  geom_path() +
  geom_point() +
  scale_color_viridis_c()
```

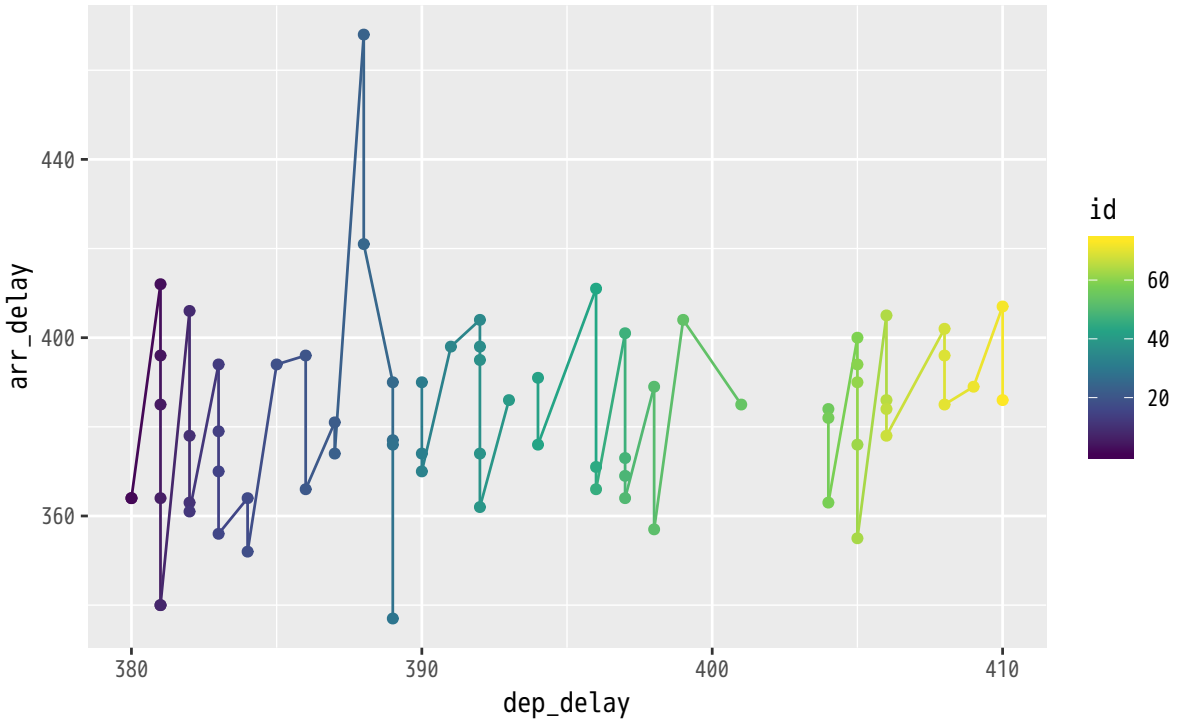

図6　並び替えの例

2.3　選択

選択はデータ分析に必要な変数（列）のみ残して他を削除する処理で、Excel なら列を削除する操作にあたります。R では `select()` で、選択したい変数名を指定するだけです。また削除したい変数名の前にマイナス記号をつけて指定し、その他を残すこともできます。ヘルパー関数と呼ばれる関数を使うと、変数名の指定をさらに楽にすることができます。

表5: ヘルパー関数

ヘルパー関数	説明
`starts_with()`	指定した文字列から始まる変数名
`ends_with()`	指定した文字列で終わる変数名
`contains()`	指定した文字列を含む変数名
`matches()`	指定した正規表現がマッチする変数名
`num_range()`	指定した文字列と数字の組合せの変数名
`one_of()`	指定した一覧に含まれる変数名
`everything()`	すべての変数名

例 1: 航空会社と遅延時間だけを選択する。

選択したい変数名を指定しただけの簡単な例です。

```
flights %>% select(carrier, dep_delay, arr_delay)
```

```
## # A tibble: 336,776 x 3
##    carrier dep_delay arr_delay
##    <chr>       <dbl>     <dbl>
##  1 UA              2        11
##  2 UA              4        20
##  3 AA              2        33
##  4 B6             -1       -18
##  5 DL             -6       -25
##  6 UA             -4        12
##  7 B6             -5        19
##  8 EV             -3       -14
##  9 B6             -3        -8
## 10 AA             -2         8
## # ... with 336,766 more rows
```

例 2 : 以下のように変数を選択する少し複雑な例です。

- 航空会社
- 変数名に dep_ を含む変数
- （ここまで選択された変数のうち）変数名が sched_ から始まらない変数
- 変数名が一覧 colnames_datetime に含まれる変数

```
colnames_datetime <- c("year", "month", "day", "hour", "minute")

flights %>%
  select(carrier, contains("dep_"), -starts_with("sched_"), one_of(colnames_datetime))
```

```
## # A tibble: 336,776 x 8
##    carrier dep_time dep_delay  year month   day  hour minute
##    <chr>      <int>     <dbl> <int> <int> <int> <dbl>  <dbl>
##  1 UA           517         2  2013     1     1     5     15
##  2 UA           533         4  2013     1     1     5     29
##  3 AA           542         2  2013     1     1     5     40
##  4 B6           544        -1  2013     1     1     5     45
##  5 DL           554        -6  2013     1     1     6      0
##  6 UA           554        -4  2013     1     1     5     58
##  7 B6           555        -5  2013     1     1     6      0
```

```
##  8 EV           557        -3  2013     1     1     6      0
##  9 B6           557        -3  2013     1     1     6      0
## 10 AA           558        -2  2013     1     1     6      0
## # ... with 336,766 more rows
```

2.4 変換

変換は新しく変数（列）を作ったり、今ある列を書き換えたりする処理で、Excel では新しく列を作って値を計算する操作にあたります。R では `mutate()` で、値を算出する式と変数名を等号で組合せることで指定します。

表6: 指定できる式

式の種類	説明
固定値	文字列や数値、真偽値などの固定値を指定することができる。
演算式	各種の演算子や関数を使って演算した結果を指定することができる。
条件式	`if_else()` などを使って、条件ごとに式を変えることができる。

表7: 演算子や関数の一部

種類	一覧
四則演算、べき乗	+,-,*,/,^
モジュラー演算（商、余）	%/%,%%
初等関数（三角関数）	sin(),cos(),tan(),asin(),acos(),atan()[*2]
初等関数（指数関数）	exp()
初等関数（対数関数）	log()[*3],log2(),log10()
初等関数（平方根）	sqrt()
その他（絶対値）	abs()
その他（丸めなど）	round(),floor(),ceiling()
比較演算	>,>=,<,<=,==,!=
論理演算	&,|,!

例 1：出発遅延時間を自分で計算してみる。

出発時刻から出発予定時刻を引いた値を持つ、新しい変数 `dep_delay_calc` を作っています。可視化による確認は、元のデータにある出発遅延時間と比較するために散布図を描いています。ただし、この計算には問題があるので修正が必要です。

[*2] y 値と x 値を指定できる `atan2()` もあります。
[*3] `base` を指定すると任意の底の対数を求めることができます。

```
flights %>%
  mutate(dep_delay_calc = dep_time - sched_dep_time) %>%
  ggplot(aes(x = dep_delay, y = dep_delay_calc)) +
  geom_point() +
  coord_fixed()
```

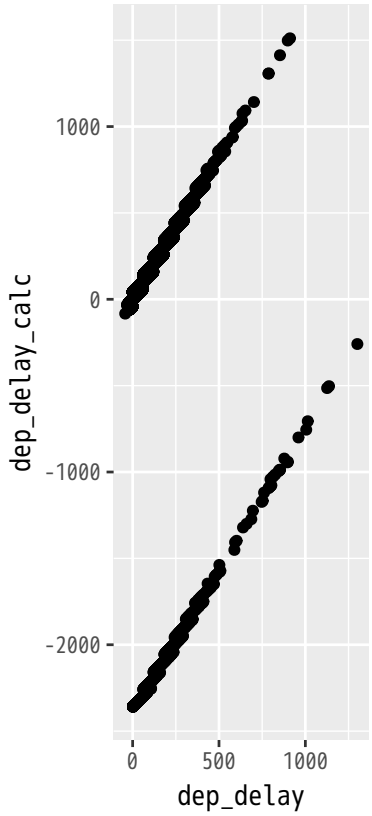

図7　変換の例 1

例 2：両方の遅延時間が 60 分より大きい場合に “YES”、そうでない場合に “NO” とする。

条件式 `if_else()` を使って固定値を切り替えています。ただし、この場合には、結果を真偽値としていいなら、条件式をそのまま指定した方がより簡単です。

```
flights %>%
  mutate(large_delay = if_else((dep_delay > 60 & arr_delay > 60),
                               "YES", "NO")) %>%
  ggplot(aes(x = dep_delay)) + geom_histogram(binwidth = 50) +
  facet_wrap(~ large_delay)
```

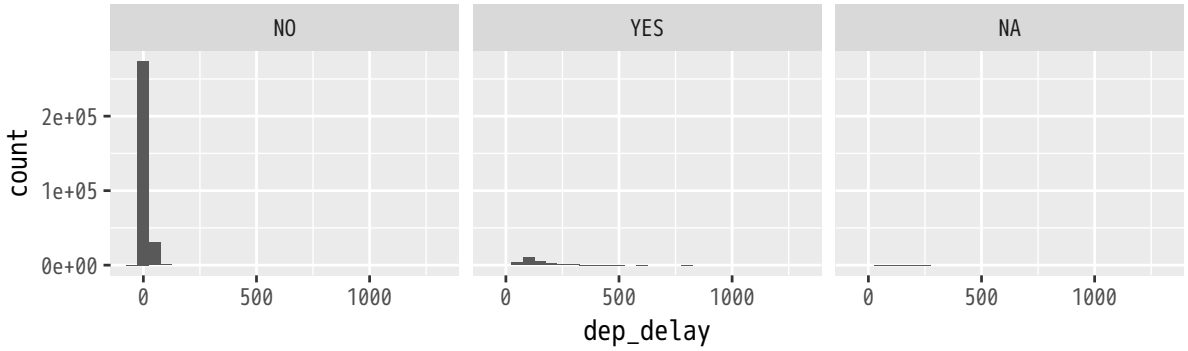

図8　変換の例 2

```
flights %>%
  mutate(large_delay = dep_delay > 60 & arr_delay > 60) %>%
  ggplot(aes(x = dep_delay)) + geom_histogram(binwidth = 50) +
  facet_wrap(~ large_delay)
```

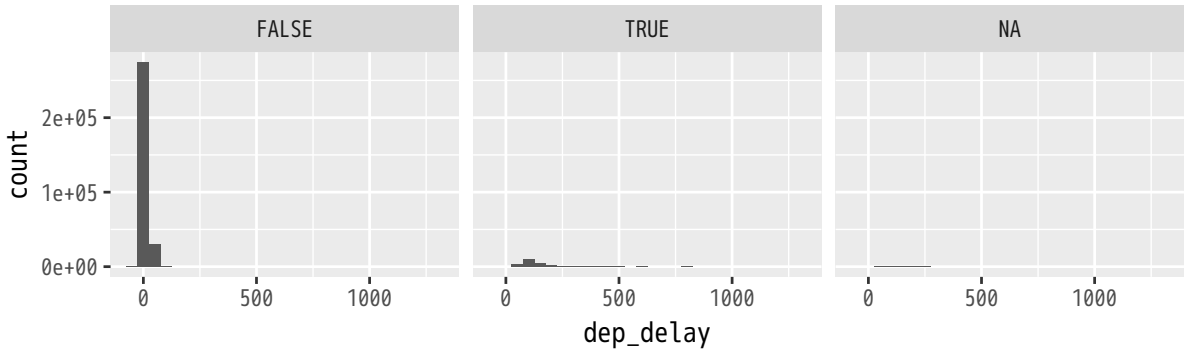

図9　変換の例 2

2.5　要約

要約はすべての行の値を 1 つの値に変換する処理で、Excel では範囲選択して集計関数を使う操作にあたります。R では `summarise()` で、要約値の名前と要約値を求める式とを等号で組合せることで指定します。また `summarise_at()` で、要約対象の変数名と要約関数の一覧とを指定する方法もあります。複数種類の要約値を求める場合は、`summarise_at()` を使った方が便利です。

例 1：到着遅延時間の平均値を `summarise()` で求める。

到着遅延時間の平均値を求めて、`mean_arr_delay` と名前をつけています。[*4]

[*4] あとで説明しますが、到着遅延時間には欠損値が含まれているため、そのままでは平均値を求めることができません。

```
flights %>% summarise(mean_arr_delay = mean(arr_delay))
```

```
## # A tibble: 1 x 1
##   mean_arr_delay
##            <dbl>
## 1             NA
```

例 2：到着遅延時間の平均値を summarise_at() で求める。

要約対象と要約関数を指定しています。[*5]

```
flights %>% summarise_at("arr_delay", funs(mean))
```

```
## # A tibble: 1 x 1
##   arr_delay
##       <dbl>
## 1        NA
```

2.5.1 欠損値の扱い

R では、データが欠損した場合に特別な値 NA[*6]を使います。そして、NA を含む計算結果は NA となったり、NA を含むデータを可視化すると警告メッセージが表示されたりします。これは、欠損値に対して適切な処理を行うことをユーザに要求していると考えてください。例えば、欠損したデータは前後の値から補間できるかもしれませんし、除去すべきかもしれません。ここでは、欠損値を処理する例をいくつか見てみましょう。

- 欠損値を含む行をすべて除去する

  すべての変数に欠損がない場合しか、データを分析する意味がない場合に使います。

  ```
  data <- tibble(time = 1:100,
                 a = c(NA, NA, rnorm(98)),
                 b = c(1:50, NA, 52:100) * 0.1 + 20)

  data %>% drop_na() %>% summarise_at("a", funs(mean, sd))
  ```

  ```
  ## # A tibble: 1 x 2
  ##     mean    sd
  ##    <dbl> <dbl>
  ## 1 0.0826 0.981
  ```

[*5] あとで説明しますが、到着遅延時間には欠損値が含まれているため、そのままでは平均値を求めることができません。

[*6] 計測器には、データが測れなかったときにレンジ外の値を使ったり、数値の代わりに文字列を指定したりするものがあります。こうした値をとることは、その計測器だけで意味を持つ方法で統一的に使える方法ではありません。（逆に言うと、そうした計測器から R にデータを読み込むときには、個別のルールに基づいて欠損値 NA として読み込むべきで、後で勉強しますが簡単な方法で実現できます。）

- 対象変数に含まれる欠損のみを除去する

  個別の変数の集約値を求めるなど、欠損を個別に処理した方がよい場合に使います。集約関数の中で欠損値を除去するように指定する方法と、drop_na() で変数を指定する方法があります。

```
data %>% summarise_at("a", funs(mean(., na.rm = TRUE), sd(., na.rm = TRUE)))
```

```
## # A tibble: 1 x 2
##      mean    sd
##     <dbl> <dbl>
## 1 0.0813 0.976
```

```
data %>% drop_na(a) %>% summarise_at("a", funs(mean, sd))
```

```
## # A tibble: 1 x 2
##      mean    sd
##     <dbl> <dbl>
## 1 0.0813 0.976
```

- 欠損値を線形補間する

  欠損値の前後の値を使って線形補間するのが妥当な場合に使います。線形補間処理 na.approx() を使うには、zoo パッケージをインストールし、ロードする必要があります。

```
library(zoo)
data %>% ggplot(aes(x = time, y = b)) + geom_path() + geom_point()
```

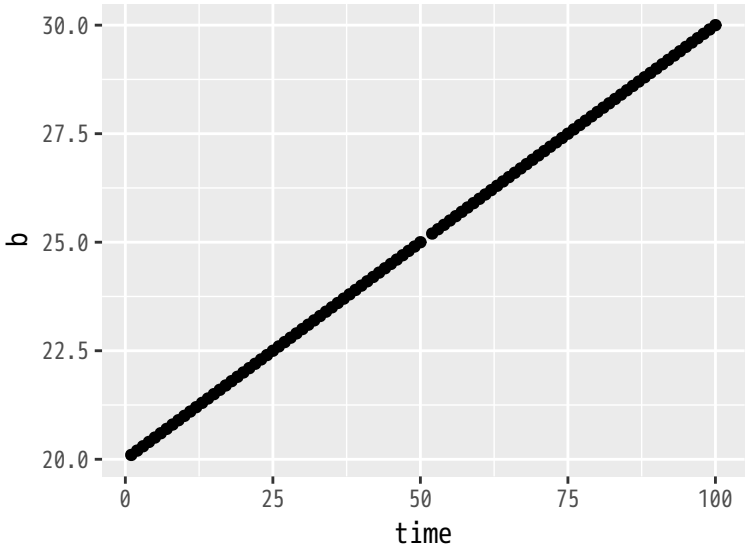

図10　欠損値の扱いの例 3

```
data %>% mutate(b = b %>% na.approx(time)) %>%
  ggplot(aes(x = time, y = b)) + geom_path() + geom_point()
```

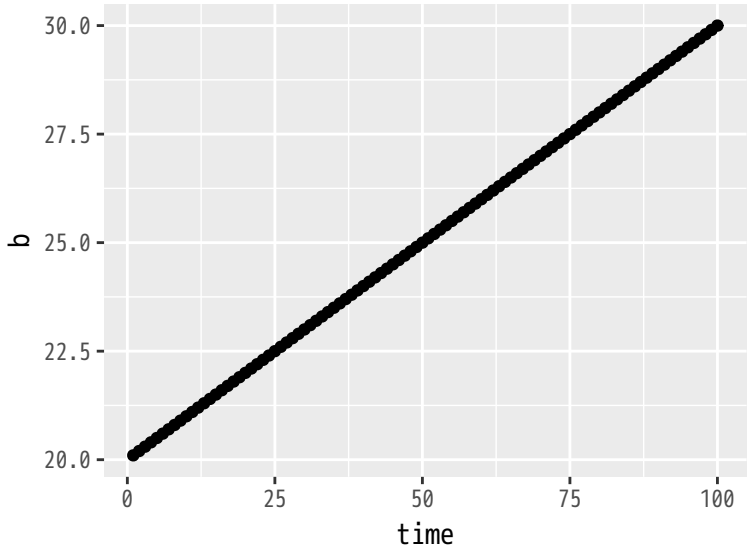

図11　欠損値の扱いの例 3

2.6　グループ化

グループ化は指定した変数（列）の値ごとのグループにデータをまとめる処理で、グループ化されたデータを使って処理を行うと、グループごとに処理が実行されます。可視化でサブグラフの階層を追加すると 1 つのグラフが変数の組み合わせごとのグラフに変化するのと似ていて、グループ化を使うと効率的に処理を記述することができます。Excel では、この機能の一部をピボットテーブルを使った集計で実現できますが、すべての機能ではありませんし使いにくい点[*7]もあります。R では `group_by()` で、グループ化したい変数名を指定するだけです。

2.6.1　グループ化＋要約

グループ化した後に要約処理を行うと、要約値をグループごとに求めることができます。

例：航空会社ごとに、出発遅延時間の平均、中央値、標準偏差を求める。

```
flights %>% drop_na() %>%
  group_by(carrier) %>%
  summarise_at("arr_delay", funs(mean, median, sd))
```

[*7] 処理が遅い、データが変更（追加・変更・削除）された時の更新が面倒、ピボットテーブルから更に集計することができない、などが使いにくい点です。

```
## # A tibble: 16 x 4
##    carrier    mean median    sd
##    <chr>     <dbl>  <dbl> <dbl>
##  1 9E        7.38      -7  50.1
##  2 AA        0.364     -9  42.5
##  3 AS       -9.93     -17  36.5
##  4 B6        9.46      -3  42.8
##  5 DL        1.64      -8  44.4
##  6 EV       15.8       -1  49.9
##  7 F9       21.9        6  61.6
##  8 FL       20.1        5  54.1
##  9 HA       -6.92     -13  75.1
## 10 MQ       10.8       -1  43.2
## 11 OO       11.9       -7  48.6
## 12 UA        3.56      -6  41.0
## 13 US        2.13      -6  33.1
## 14 VX        1.76      -9  50.0
## 15 WN        9.65      -3  46.9
## 16 YV       15.6       -2  52.9
```

2.6.2 グループ化＋フィルタ

グループ化した後にフィルタ処理を行うと、フィルタ条件の式がグループごとに評価されます。

例：航空会社ごとに、出発が最も遅れた日付を求める。

```
flights %>% drop_na() %>%
  group_by(carrier) %>%
  filter(dep_delay == max(dep_delay)) %>%
  select(carrier, year, month, day, dep_delay)
```

```
## # A tibble: 16 x 5
## # Groups:   carrier [16]
##    carrier  year month   day dep_delay
##    <chr>   <int> <int> <int>     <dbl>
##  1 HA       2013     1     9      1301
##  2 B6       2013     1    16       502
##  3 YV       2013    10    22       387
##  4 EV       2013    12     5       548
##  5 F9       2013     2    10       853
##  6 9E       2013     2    16       747
```

```
##  7 DL       2013     4    10       960
##  8 WN       2013     5    11       471
##  9 AS       2013     5    23       225
## 10 MQ       2013     6    15      1137
## 11 US       2013     6    28       500
## 12 VX       2013     7     7       653
## 13 UA       2013     7    26       483
## 14 OO       2013     8    28       154
## 15 FL       2013     9    12       602
## 16 AA       2013     9    20      1014
```

2.6.3 グループ化＋変換

グループ化した後に変換処理を行うと、値を算出する式がグループごとに評価されます。

例：航空会社ごとに、平均遅延時間からのオフセット値を求める。

```
flights %>% drop_na() %>%
  arrange(year, month, day, dep_time) %>%
  group_by(carrier) %>%
  mutate(dep_delay_offset = dep_delay - mean(dep_delay)) %>%
  select(carrier, year, month, day, dep_delay, dep_delay_offset)
```

```
## # A tibble: 327,346 x 6
## # Groups:   carrier [16]
##    carrier  year month   day dep_delay dep_delay_offset
##    <chr>   <int> <int> <int>     <dbl>            <dbl>
##  1 UA       2013     1     1         2           -10.0
##  2 UA       2013     1     1         4            -8.02
##  3 AA       2013     1     1         2            -6.57
##  4 B6       2013     1     1        -1           -14.0
##  5 DL       2013     1     1        -6           -15.2
##  6 UA       2013     1     1        -4           -16.0
##  7 B6       2013     1     1        -5           -18.0
##  8 EV       2013     1     1        -3           -22.8
##  9 B6       2013     1     1        -3           -16.0
## 10 AA       2013     1     1        -2           -10.6
## # ... with 327,336 more rows
```