

応用プログラミング3
第8回 tidyverseによる
データハンドリング (2)

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

2. lubridateパッケージによる日時データの操作

2

前回に引き続き、tidyverseに含まれる、あるいは周辺のパッケージ群を紹介していきます。

まず紹介するのは、日付や時刻などの日時データを効率的に扱うことができる、lubridateパッケージです。

2.1 Rにおける日時データ

クラス	用途	関数	注意点
Date	年月日を管理する	as.Date()	時刻の情報は持たない
POSIXct	年月日日時を管理する	as.POSIXct()	<ul style="list-style-type: none"> 内部ではUNIX Timeの秒数で管理されている タイムゾーンに注意
POSIXlt	年月日日時を管理する	as.POSIXlt()	要素がリストで格納されている
Period	年月日を持たない時刻や、日時の差分を管理する	period()	lubridateパッケージが提供

3

はじめに、lubridateパッケージに限らない、Rにおける日時データの型についてまとめます。Rには、標準で4つの日時データ型があります。

Date型は、YYYY-mm-dd の形式で年月日を管理します。時刻の情報は持ちません。そして、日時を管理する型として、POSIXct型とPOSIXlt型があります。内部的には、どちらも1970-01-01 00:00:00からの経過秒数であるUNIX Timeでデータを保持しています。大まかには、POSIXct型はベクトルで、POSIXlt型はリストで値を持っています。基本的にはPOSIXct型で扱う、と考えておけばよいでしょう。

なお、いずれの型についても、タイムゾーンの扱いには注意が必要です。特に、Posit Cloudは、裏で動いているサーバーのタイムゾーンがUTC (協定標準時)なので、日本時間とは9時間の時差があります。例えば時系列データを可視化する際に、手元のコンピューターとPosit Cloudで、同じプログラムでも結果が変わる(値の表示位置が9時間分ずれる)ことがありますので、普段から明示的に `tz = "Asia/Tokyo"` とタイムゾーンを記述するとよいでしょう。

また、昨日と今日、去年と今年など、2つの時点の差などを扱うPeriod型もあります。これらの型については、次のページ以降でプログラムを交えて説明します。

- 参考: 時系列データ <https://okumuralab.org/~okumura/stat/timeseries.html>

2.2 標準の日時データ操作関数（1）Date型

- `as.Date("YYYY-mm-DD")` でDate型のオブジェクトを作成できる
- 内部では1970年1月1日からの経過日数で管理される
- 整数の加減算で日付を増減できる
- `Sys.Date()` 関数で今日の日付を取得できる

```
today <- Sys.Date()
today
## [1] "2022-11-07"

today + 1
## [1] "2022-11-08"
```

4

前述のように、Date型は年月日を保持します。内部では、1970年1月1日からの経過日数として管理しています。Date型のオブジェクトは、文字列を `as.Date()` 関数で変換して作成できます。また、今日の日付を、`Sys.Date()` 関数で取得できます。この場合、タイムゾーンは使用しているコンピューターに設定されているものとなり、指定できません。

Date型のオブジェクトは、+1 で1日後、+7 で1週間後、といったように加減の演算ができます。

2.3 標準の日時データ操作関数 (2) POSIXct型

- `as.POSIXct("YYYY-mm-DD HH:MM:SS")` でPOSIXct型のオブジェクトを作成できる
- 内部では1970年1月1日0時からの経過秒数で管理される
- 整数の加減算で秒単位で時刻を増減できる
- `Sys.time()` 関数で現在の日時を取得できる

```
now <- Sys.time() # この教材を作っていた日時
now

## [1] "2022-11-07 13:57:51 JST"

now + 600 # 10分後

## [1] "2022-11-07 14:07:51 JST"
```

5

POSIXct型は、1970-01-01 00:00:00から任意の時刻までの経過秒数を保持します。文字列を `as.POSIXct()` 関数に与えて作成します。こちらも、`Sys.time()` 関数で、関数を実行した瞬間の日時を取得できます。

POSIXct型も、+1 や -60 など加減算ができます。ただし、Date型と違い秒単位での計算ですので注意してください。

なお、レアパターンですが、日本に戦後の一時期だけ存在した「夏時間」により、一部の日時が処理できないことがあります。以下の参考URLによると、「1948年は、5月2日0:00時に1時間時刻を進めて、9月の第2土曜日の深夜24:00に1時間時計の針を戻した」ことで、「5月2日は1時から始まり、0時が存在しない」ようになっています。夏時間は、1948年から1951年まで実施されたそうです。

- 参考1: C++20でサマータイムを謳歌しよう - Qiita
<https://qiita.com/MitsutakaTakeda/items/e7bb08da3260f8205607>
- 参考2: 時系列データ <https://okumuralab.org/~okumura/stat/timeseries.html>

経済データなどは長期間に渡る変化を分析することも多いため、まれにこの事象に遭遇するかもしれません。その時に思い出してください。

2.4 標準の日時データ操作関数 (3) POSIXlt型

- `as.POSIXlt("YYYY-mm-DD HH:MM:SS")` でPOSIXlt型のオブジェクトを作成できる
- 内部では1970年1月1日0時からの経過秒数で管理される
- 年、月、日、時、分、秒がリストの要素として格納される

```
now <- Sys.time()                ##
now <- as.POSIXlt(now)           ## $class
now                               ## [1] "POSIXlt" "POSIXt"
## [1] "2022-11-07 13:57:51 JST"      ##
attributes(now)                  ## $tzone
## $names                        ## [1] ""      "JST" "JDT"
## [1] "sec" "min" "hour"            now$hour
"mday" "mon" "year" "yday"      ## [1] 13
"yday" "isdst" "zone" "gmtoff" ↵
```

6

POSIXlt型は、POSIXct型と同様の日時情報を、リストとして保持する型です。オブジェクト名\$hour などとして、個別に要素を取り出すことができます。

ただ、そのような機能も、次に紹介するtidyverseのlubridateパッケージがカバーしているため、意図的にPOSIXlt型を使う機会はないでしょう。古いパッケージの関数が、POSIXlt型で結果を返してくることはあるかもしれません。

2.5 lubridateパッケージ

- <https://lubridate.tidyverse.org/>
- tidyverseの一部を構成する
- 標準の関数よりも柔軟に日時データを操作できる
- "2023年11月14日" など日本語にも対応している



7

lubridateは、tidyverseの一部として、効率的な日時データの利用を実現するパッケージです。さまざまなフォーマットの文字列を日時型に変換したり、日時型のオブジェクトから必要な情報を容易に取り出すことができます。日本語を含む文字列も、特に設定せずに日時型に変換できます。

2.6 lubridateパッケージの日時データ操作関数

(1) オブジェクトの作成

- 年月日、日時などさまざまなフォーマットの文字列をDate型やPOSIXct型に変換できる
- `ymd()`, `ydm()`, `mdy()`, `myd()`, `dmy()`, `dym()`, `yq()`: 年月日に対応
- `ymd_hms()`, `ymd_hm()`, `ymd_h()`, `dmy_hms()`, `dmy_hm()`, `dmy_h()`, `mdy_hms()`, `mdy_hm()`, `mdy_h()`, `ydm_hms()`, `ydm_hm()`, `ydm_h()`: 年月日時刻に対応
- タイムゾーンはデフォルトでUTCなので
`tz = "Asia/Tokyo"` と指定する
- `parse_date_time()`(文字列, フォーマット): 極めて柔軟な解釈で日時データを取り扱う

8

lubridateパッケージが提供する、文字列を日時型に変換する関数は多岐にわたります。対象となる文字列のフォーマットに応じて、スライドに列挙したさまざまな関数を使い分けます。経済データやビジネスデータで使われる「四半期」についても、`yq("2022Q3")` といった記述で変換できます。この場合、期のはじめの日 (3Qであれば7月1日) に変換されます。

なお、lubridateパッケージは、標準でタイムゾーンをUTCとして扱います。そのため、日本時間であることを明示したい場合は、`tz = "Asia/Tokyo"` と指定します。

また、特殊なフォーマットで記録されている文字列は、`parse_date_time()` 関数で直接フォーマットを指定して処理できます。ひとつのファイル (データフレーム) の中でフォーマットが異なっているような場合も、フォーマットを複数記述して、当てはまるものを適用できます。

2.6 lubridateパッケージの日時データ操作関数

(1) オブジェクトの作成

```
today <- "2021-08-25"
ymd(today)

## [1] "2021-08-25"

now <- "2021-08-25 23:48:36"
ymd_hms(now, tz = "Asia/Tokyo")

## [1] "2021-08-25 23:48:36 JST"

now2 <- "25日,8月 23時53分, 21年"
parse_date_time(now2, "dmHMy")

## [1] "2021-08-25 23:53:00 UTC"
```

2.7 lubridateパッケージの日時データ操作関数 (2) 日時情報の抽出と変更

- Date型、POSIXct型のオブジェクトから任意の要素を抽出できる
- `month()` 関数ではロケール（言語）ごとの別名も取得できる
- `year(オブジェクト) <- 2022` など、値を変更することもできる

```
now <- ymd_hms(Sys.time(), tz = "Asia/Tokyo")
year(now)

## [1] 2022

month(now, label = TRUE)

## [1] 11

## Levels: 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < 11 < 12
```

10

lubridateパッケージには、日時型のオブジェクトから、必要な情報を取り出すための関数も用意されています。`year()` 関数や `month()` 関数で、年だけ、月だけなどを取得できます。`month()` や `wday()` (曜日) 関数では、`label = TRUE` オプションと `locale = "ロケール"` オプションを組み合わせることで、それぞれの言語特有の月名、曜日名を得ることもできます。この時のロケールは、OSがサポートしているものに限りです。

また、`year(オブジェクト) <- 2022` などとすることで、既存の日時型オブジェクトの情報を変更することもできます。

2.7 lubridateパッケージの日時データ操作関数 (2) 日時情報の抽出と変更

```
wday(now, label = TRUE, locale = "Japanese_Japan.utf8")
```

```
## [1] 月
```

```
## Levels: 日 < 月 < 火 < 水 < 木 < 金 < 土
```

```
wday(now, label = TRUE, locale = "Chinese_China.utf8")
```

```
## [1] 周一
```

```
## Levels: 周日 < 周一 < 周二 < 周三 < 周四 < 周五 < 周六
```

```
year(now) <- 2022
```

```
hour(now) <- hour(now) + 1
```

```
now
```

```
## [1] "2022-11-07 14:57:51 JST"
```

2.8 lubridateパッケージの日時データ操作関数

(3) 日時の演算

- オブジェクト + months(1) で1月後、などの演算ができる
- 日付を月初または月末に揃えたい、秒を四捨五入したいといったニーズがある
- floor_date(unit = "単位"): 時刻を指定した単位 (minute, hour, month 等) で切り下げる
- ceiling_date(unit = "単位"): 時刻を指定した単位 (minute, hour, month 等) で切り上げる
- round_date(unit = "単位"): 時刻を指定した単位 (minute, hour, month 等) で丸める

12

lubridateでは、日時の演算を容易にする関数も提供されています。years() や months()、days() などを使うことで、任意の単位で日時を加減算できます。

また、日時データを処理する際に、細かすぎる情報を、日単位、月単位、年単位にまとめたいことがあります。year() 関数と month() 関数などを組み合わせて、paste(year(オブジェクト), month(オブジェクト), sep = "-") といったような操作を行い、文字列として扱うこともできますが、lubridateパッケージには、より柔軟に日時を揃える・丸めるための関数があります。

最も使う機会が多いと思われるのは、floor_date() 関数です。日時データを、切りの良い直前の値に変換します。つまり、unit = "month" と指定すると、毎月1日の0時ちょうどに切り下げられます。unit オプションには、"3 month" などを指定することも可能です。

逆に、月末や年末に切り上げるのが ceiling_date() 関数です。使い方は同じです。また、round_date() 関数は期間の中ほどに丸めますが、コンピューターにおける丸めは単純な四捨五入ではなく、やや直感的ではない処理なので、あまり積極的には使わないほうがよいでしょう。

- 参考1: Rのround()は四捨五入をするわけではない - 盆暗の学習記録
<https://nigimitama.hatenablog.jp/entry/2018/11/30/131543>
- 参考2: lubridate::floor_dateで任意の曜日スタートで切り下げ - 今週も特にありません
<https://masaqol.hatenablog.com/entry/2019/03/31/223722>

2.8 lubridateパッケージの日時データ操作関数

(3) 日時の演算

```
now <- ymd_hms(Sys.time(), tz = "Asia/Tokyo")
now
## [1] "2022-11-07 13:57:51 JST"
round_date(now, unit = "hour")
## [1] "2022-11-07 14:00:00 JST"
floor_date(now, unit = "hour")
## [1] "2022-11-07 13:00:00 JST"
ceiling_date(now, unit = "hour")
## [1] "2022-11-07 14:00:00 JST"
ceiling_date(now, unit = "month")
## [1] "2022-12-01 JST"
ceiling_date(now, unit = "month") - days(1) # 月末にするには1日引く
## [1] "2022-11-30 JST"
```

2.9 zipanguパッケージによる休日の処理

- <https://uribo.github.io/zipangu/>
- 経済活動などを分析する際には休日効果を考慮する必要がある
- zipanguは日本のRユーザーによる、日本に特有の住所、元号、休日などを処理する関数を提供するパッケージ
- `is_jholiday()` 関数で休日の判定ができる

```
library(zipangu)

is_jholiday("2021-07-23") # 2021年のみ「スポーツの日」
## [1] TRUE

jholiday_spec(2021, "スポーツの日", lang = "jp")
## [1] "2021-07-23"

jholiday_spec(2019, "体育の日", lang = "jp") # 2019年以前は「体育の日」
## [1] "2019-10-14"
```

14

日本特有の休祝日などを処理するために、日本人が作成したパッケージにzipanguがあります。それ以前に、Nipponというパッケージがありましたが、更新が停止し、CRANからも削除されました(※)。

そこで、代替となるパッケージとして開発されたのがzipanguです。現在のところ、祝日などの変更に合わせて随時更新されているので、しばらくは利用できるでしょう。日時の処理以外にも、日本語特有の半角・全角の変換や、住所の切り分けなど、豊富な機能を提供しています。

- 参考: 住所や年号、漢数字のデータ操作を楽にするRパッケージをCRANに登録しました - cucumber flesh <https://uribo.hatenablog.com/entry/2019/12/02/163114>

※ GitHubのCRANアーカイブには残っているので、無理やりインストールすることはできますが、あえてやることはありません。

3. stringrパッケージによる文字列データの操作

15

ここからは、文字列処理を柔軟に、効率的に行うためのstringrパッケージを紹介します。

3.1 文字列と正規表現

- 数値だけでなくアンケートの自由記述など、文字列（テキスト）を扱うことも多い
- 大量のテキストから条件を満たすものを抽出する際に、正規表現が使われる
- 対象文字列をできるだけ抽象化し、シンプルなパターンで記述することを心掛ける
- 何でもかんでもRでやる必要はない（エディターの機能や別のプログラムで処理したほうが早いこともある）

The diagram illustrates several regular expressions and their matches for the text "スモモも桃も桃のうち 桃もスモモも桃のうち".

- ^ス** (行マッチの場合): スモモも桃も桃のうち, 桃もスモモも桃のうち
- ち\$** (行マッチの場合): スモモも桃も桃のうち, 桃もスモモも桃のうち
- ^...も** (行マッチの場合): スモモも桃も桃のうち, 桃もスモモも桃のうち
- .*もス.*** (行マッチの場合): スモモも桃も桃のうち, 桃もスモモも桃のうち
- も.*** (行マッチの場合): スモモも桃も桃のうち, 桃もスモモも桃のうち
- .*桃.*{2,}** (行マッチの場合): スモモも桃も桃のうち, 桃もスモモも桃のうち

Central text: スモモも桃も桃のうち
桃もスモモも桃のうち

16

アンケートやSNSの書き込みなどを分析する場合、文字列の処理が必要になります。ある条件に合致するテキストだけ抽出したい、前処理の一環として特定の文字列を削除・置換したいなどがニーズとしてあります。条件が複雑な場合、単純なマッチングではなく、正規表現を使います。

正規表現そのものをここで説明することはしませんが、ソフトウェアによって「方言」があるので、自分が使用するソフトウェアがどのような正規表現に対応しているかを確認する必要があります。

また、これからstringrパッケージを使って文字列処理を行う方法を紹介しますが、そもそもRではない別のツールを使ったほうが効率的かもしれません。目的に合わせて、最適な手段を選びましょう。

3.2 標準の文字列データ操作関数

- 組み込み関数として `grep()`, `grepL()`, `gsub()` などがある
- `grep("パターン", オブジェクト)` はマッチした項目のインデックスを、`grepL("パターン", オブジェクト)` は各要素についてマッチしたかどうかの `TRUE` / `FALSE` を返す
- `gsub("対象", "置換後文字列", オブジェクト)` はマッチした文字列をすべて置換する

```
vec <- c("日本", "Japan", "ニホン", "にほん", "Japan")
grep("Japan", vec)
## [1] 2 5

grepL("日本", vec)
## [1] TRUE FALSE FALSE FALSE FALSE

gsub("Japan", "Japón", vec)
## [1] "日本" "Japón" "ニホン" "にほん" "Japón"
```

17

Rには、標準（組み込み）で文字列データを操作するための関数が多く実装されています。それぞれ、正規表現に対応し、条件に合致するベクトルの要素を抽出したり、文字列を置換できます。

ただし、`tidyverse`のパイプの中で使用するのが難しかったり、書式や返り値がばらばらだったり、あまり便利ではありません。そこで、豊富な機能と統一されたインターフェースを有する`stringi`（ストリングーと発音するようです）というパッケージが開発されました。さらに、`tidyverse`の文脈でより使いやすいように、`stringi`のラッパー（wrapper）として振る舞う`stringr`パッケージが使われています。

今回は、`stringr`パッケージの機能を紹介していきます。

- 参考1: R における正規表現 - RjpWiki
<http://www.okadajp.org/RWiki/?R+%E3%81%AB%E3%81%8A%E3%81%91%E3%82%8B%E6%AD%A3%E8%A6%8F%E8%A1%A8%E7%8F%BE>
- 参考2: `stringi`: Fast and Portable Character String Processing in R — `stringi`
<https://stringi.gagolewski.com/>
- 参考3: Simple, Consistent Wrappers for Common String Operations • `stringr`
<https://stringr.tidyverse.org/>

3.3 stringrパッケージ

- <https://stringr.tidyverse.org/>
- tidyverseの一部を構成する、文字列を統一的な文法で扱うためのパッケージ
- tidyverseのパイプや `map()` 系関数と組み合わせて使いやすい



18






stringrパッケージはtidyverseの一部で、前述のようにstringiパッケージのラッパーとして機能します。パイプや、後述するpurrrパッケージの `map()` 系関数と組み合わせて使いやすい関数を提供しています。

3.4 stringrパッケージの文字列データ操作関数

(1) 検索

- `str_count("文字列", "パターン")`: 要素ごとにマッチした回数を出力する
- `str_detect("文字列", "パターン")`: 要素ごとにマッチしたかどうかTRUE / FALSEを返す
negate = TRUE で結果を反転する
- `str_which("文字列", "パターン")`: マッチした要素のインデックスを返す

Detect Matches

	TRUE TRUE FALSE TRUE	str_detect(string, pattern, negate = FALSE) Detect the presence of a pattern match in a string. Also str_like() . <code>str_detect(fruit, "a")</code>
	TRUE TRUE FALSE TRUE	str_starts(string, pattern, negate = FALSE) Detect the presence of a pattern match at the beginning of a string. Also str_ends() . <code>str_starts(fruit, "a")</code>
	1 2 4	str_which(string, pattern, negate = FALSE) Find the indexes of strings that contain a pattern match. <code>str_which(fruit, "a")</code>
	start end 2 4 4 7 NA NA 3 4	str_locate(string, pattern) Locate the positions of pattern matches in a string. Also str_locate_all() . <code>str_locate(fruit, "a")</code>
	0 3 1 2	str_count(string, pattern) Count the number of matches in a string. <code>str_count(fruit, "a")</code>

Stringr Cheat Sheet - Posit

<https://posit.co/wp-content/uploads/2022/10/strings-1.pdf>

19

stringrパッケージが提供する関数は、書式が統一されています (当たり前ですが)。関数名("対象文字列", "パターン") が基本で、さらに置換の場合、その後ろに置換後の文字列を指定します。

文字列を検索するための関数として、`str_count()`、`str_detect()`、`str_which()` があります。それぞれ、条件にマッチした回数やベクトルの要素ごとの TRUE / FALSE、マッチした要素番号を返します。

なお、このスライドのプログラム例では、Web上のExcelファイルを直接読み込むために、`rio`パッケージ (<https://github.com/leeper/rio>) の `import()` 関数を使っています。前回紹介した、`readxl`パッケージの `read_excel()` 関数は、ローカルのファイルしか読み込めないためです。

`rio`パッケージは、"A Swiss-Army Knife for Data I/O" と自称し、さまざまなフォーマットのデータを柔軟に読み込むための関数を提供します。

- 参考1: r - Read Excel file from a URL using the readxl package - Stack Overflow
<https://stackoverflow.com/a/63910298>
- 参考2: Rでウェブ解析：いろいろなファイルを読み込み、書き出しができます。「rio」パッケージ <https://www.karada-good.net/analyticsr/r-82/>

3.4 stringrパッケージの文字列データ操作関数

(1) 検索

```
library(tidyverse) # library(stringr)
library(rio)
df <- import(file = "https://cio.go.jp/.../r2_survey_comments.xlsx",
             sheet = 5, skip = 5)
df %>% mutate(結果1 = str_count(内容, "コロナ"),
             結果2 = str_detect(内容, "データ")) %>% select(starts_with("結果"))

##   結果1 結果2
## 1     1  TRUE
## 2     1 FALSE
...

df %>% summarise(結果3 = str_which(内容, "アプリ"))

##   結果3
## 1     11
## 2     12
...
```

3.5 stringrパッケージの文字列データ操作関数 (2) 抽出

- `str_extract("文字列", "パターン")`: マッチした部分文字列を抽出する。マッチしない要素は NA が返る
- `str_subset("文字列", "パターン")`: マッチした要素全体を返す
- `str_match_all("文字列", "パターン")`: 後方参照((¥1) など) が使用可能で、結果は行列で返る

```
df %>% mutate(結果 = str_extract(内容, "^..."))
  %>% select(結果) # 最初の3文字を抽出

df %>% summarise(結果 = str_subset(内容, "アプリ"))

df %>% mutate(結果 = str_match_all(内容, "アプリ")) %>% select(結果)

str_match_all(df[["内容"]], "アプリ")
```

21

文字列から、条件にマッチする箇所を抽出する関数も複数提供されています。
`str_extract()` 関数はマッチした部分のみ、`str_subset()` 関数はマッチしたベクトルの要素を返します。

`str_match_all()` 関数はやや複雑な動作をしますが、結果はリストに格納された行列 (ベクトルの1要素につき1つの行列) で出力され、マッチした要素が複数ある場合は、行方向に個別に出力されます。…何が何だかイメージできないと思いますが、ハンズオンのプログラム例をぜひ実行してみてください。なお、`mutate()` 関数などでデータフレームの列として出力した場合は、列の中にリストが格納されます。データフレームは柔軟な構造なので、列に入れ子構造でデータフレームや行列、リストを格納できます。なお、`str_match()` 関数は、マッチした最初の1つだけ抽出します。

3.6 stringrパッケージの文字列データ操作関数

(3) 置換

- `str_replace_all("文字列", "パターン", "置換後文字列")`: マッチした文字列を置換する
- `str_to_upper("文字列")`, `str_to_lower()`: 大文字小文字の変換
- `str_pad("文字列", 文字数, side, "パディング文字")`: 文字列の端を指定した文字で埋める。数値をゼロ埋めするなどで使用する
- `str_trim("文字列", side)`: 先頭と末尾の空白文字を除去する

```
df %>% mutate(内容 = str_replace_all(内容,
  "(新型)*コロナ(ウイルス)*(感染症)*", "COVID-19")) %>% select(内容)

df %>% mutate(地方公共団体コード = str_pad(地方公共団体コード, width = 8,
  side = "left", pad = 0)) %>% select(地方公共団体コード)

str_trim(" あいうえお ")
```

22

次に、文字列の置換を行うための関数を紹介します。全般的な置換機能を提供するのが、`str_replace_all()` 関数です。all を付けない場合、マッチした最初の1つだけ処理します。

また、用途別に特化した関数もいくつかあります。`str_to_upper()` および `str_to_lower()` 関数は、名前の通りアルファベットを大文字または小文字に変換します。`str_pad()` 関数は、文字列の前や後ろの端を、指定した桁数まで任意の文字で埋めるパディングの機能を提供します。`side` オプションに "left", "right", "both" のいずれかを指定します。また、`str_trim()` 関数で、文字列の前後にある空白文字を除去できます。こちらも、"left", "right", "both" のいずれかを指定します。

4. purrrパッケージによる 繰り返し処理

23

ここでは、tidyverseにおいて、繰り返し (ループ) 処理を簡潔に記述するための関数を提供するpurrrパッケージを紹介します。"purrr" は、「パーア」などと発音し、「フランス語で猫がのどを鳴らす様子をあらわす擬音」だそうです。

従来、繰り返し処理にはfor文やwhile文を使っていましたが、tidyverseの思想では、繰り返し処理を実行する部分と、処理の内容を切り分けて、よりシンプルなプログラムになることを目指します。つまり、毎回自分でカウンター変数を定義し、行数や列数を取得して、処理の後カウンターを1つ増やすプログラムを書くのではなく、「すべての要素に同じ処理を適用する」機能を持った関数に、処理だけ渡すほうがより洗練されていて良い、ということです。(このあたりはHadley Wickhamの書籍 "R for Data Science" を読んだ筆者の解釈です)

purrrパッケージは、上記のような「すべての要素に同じ処理を適用する」といった機能を持つ関数を提供します。

- 参考1: 21 Iteration | R for Data Science <https://r4ds.had.co.nz/iteration.html#for-loops-vs.-functionals>
- 参考2: Lessons and Examples <https://jennybc.github.io/purrr-tutorial/index.html>

4.1 map() 系関数による繰り返し処理

- **map(対象, 関数)**: 引数に指定した対象の各要素に関数を適用し、結果をリストで返す
- **map2(対象1, 対象2, 関数)**: 引数が2つあるパターン
- **map_dbl(対象, 関数)** / **map_int(対象, 関数)** / **map_chr(対象, 関数)**: 結果を指定した型のベクトルで返す
- **map_dfr(対象, 関数) (row)** / **map_dfc(対象, 関数) (column)**: 結果をデータフレームで返す
- **map_if(対象, 条件, 関数)**: 条件に合致する要素にだけ処理を適用する

```
iris %>% select(-Species) %>% map(mean)

iris %>% map_if(is.numeric, mean)
```

24

purrrパッケージが提供する主要な機能は、**map() 系関数**です。基本的な動作は共通して、「対象のすべての要素に同じ処理を適用する」ものです。注意点は、**結果がリストで返ってくる** (ものが多い) ということです。個別の処理結果にアクセスするには、リスト名[[要素番号]][要素番号] といったように記述する必要があります。やや不便ですが、リストにすることで、さらにその中での summarise() などにつなげていくことができます。

なお、**map_dbl()**, **map_int()**, **map_chr()**, **map_lgl()** (Logical: TRUE / FALSE) の各関数は、結果をベクトルで返します。そのため、用途としては前段に他の **map() 系関数**による処理があり、さらにその結果に個別に処理を適用したい場合などに使います。

また、**map_if()** 関数や **map_at()** 関数など、条件を指定し、合致した要素にだけ処理を適用するものもあります。

4.2 modify() 系関数による繰り返し処理

- `modify(対象, 関数)`: 引数に指定した対象の各要素に関数を適用し、結果を元のオブジェクトと同じ型で返す
- `modify2(対象1, 対象2, 関数)`: 引数が2つあるパターン
- `modify_if(対象, 条件, 関数)`: 条件に合致する要素にだけ処理を適用する

```
iris %>% select(-Species) %>% modify(scale)

##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1      -0.89767      1.01560      -1.33575      -1.3110521
## 2      -1.13920      -0.13154      -1.33575      -1.3110521
...

iris %>% modify_if(is.numeric, scale)

##      Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1      -0.89767      1.01560      -1.33575      -1.3110521 setosa
## 2      -1.13920      -0.13154      -1.33575      -1.3110521 setosa
```

25

`map()` 系関数は、基本的に結果をリストで返しますが、`modify()` 系関数は、結果を処理対象のオブジェクトと同じ型で返します。つまり、データフレームを与えると、結果もデータフレームになります。こちらのほうが、Rプログラミングに詳しくない(大多数の)ユーザーにとっては直感的かもしれません。使い方は、`map()` 系関数と同じです。

4.3 map() 系関数と並列処理

- map() 系関数は並列処理ではない
- 記述がシンプルになるだけで処理は順次行われる
- マルチコア処理をするには、furrrパッケージやparallelパッケージを使う
- Posit Cloud無料版はシングルコアなので機能しない



26

purrrパッケージが提供する map() 系関数は、複数の要素 (列) に同じ処理を「同時に」適用しているように見えますが、実際には、処理自体は並列では行われていません。…というか、R自体が標準ではシングルコアしか使用しません。そのため、データ量が多い場合、CPUリソースは余っているのに、処理に時間がかかってしまいます。

Rでマルチコアを用いた並列処理を行うには、追加でパッケージを利用する必要があります。ここで紹介している map() 系の処理には、専用のパッケージとしてfurrrがあります。

- 参考1: GitHub - DavisVaughan/furrr: Apply Mapping Functions in Parallel using Futures <https://github.com/DavisVaughan/furrr>
- 参考2: furrr パッケージで R で簡単並列処理 | Atusy's blog <https://blog.atusy.net/2018/12/06/furrr/>

(次ページへ続く)

また、map()に限らない、一般的な処理の並列化については、以前(2010年代)は多数のパッケージが乱立していましたが、それらを統合したparallelパッケージが標準で提供されるようになりました。ただ、通常のプログラムに、使用するCPU数を指定したり、どの処理を並列実行させるかを追加で記述する必要があり、それほど手軽に利用できるものではありません。

parallelパッケージにさらに使いやすいインターフェース(関数)を被せたdoParallelパッケージなどがあり、それらを使うと多少はプログラミングが容易になります。

- 参考3: 並列計算 | R の parallel パッケージを使用して並列計算を行う方法
<https://stats.biopapyrus.jp/r/devel/parallel.html>
- 参考4: 【R初心者人向け】doParallelパッケージを用いた並列化の方法 ~NMDSを例に~ - Qiita <https://qiita.com/yasainiki/items/bafd93505e6b4b81b8e9>

また、dplyrによる処理を並列化するための、multidplyrパッケージも、tidyverseの一部として開発されています。

- 参考5: GitHub - tidyverse/multidplyr: A dplyr backend that partitions a data frame over multiple processes <https://github.com/tidyverse/multidplyr>

上記のような、「大きな計算を、より速く」行うための技術領域を、「HPC」(High Performance Computing)と言います。HPCの領域では、複数台のコンピューターからなるクラスターを構築し、より多くのCPU、メモリを使った大規模な計算を行うために、Apache Sparkというソフトウェアが広く使われています。RからSparkクラスターに接続するためのパッケージとして、Posit社が開発しているsparklyrがあります。

- 参考6: sparklyr <https://spark.rstudio.com/>

RStudioは、sparklyrを使ってクラスターに接続する機能が組み込まれており、(デフォルトで)右上のペインの"Connections"タブから接続できます。

5. tidyverseを中心としたエコシステム

28

ここからは、tidyverseそのものではありませんが、tidyverseを補完する、あるいはtidyverseを念頭に開発された、有用なパッケージを紹介していきます。

5.1 DBIパッケージによるデータベースアクセス

- <https://dbi.r-dbi.org/>
- さまざまなRDBMSに対する統一的なインターフェースを提供するパッケージ
- DBへの接続には別途個別のパッケージが必要

```
library(DBI)
library(RSQLite)
con <- dbConnect(SQLite(), "stationery.db")
dbListFields(con, "店舗マスター")

## [1] "店舗番号" "店舗名" "開店時間" "閉店時間" "所在都道府県" "電話番号"

res <- dbSendQuery(con, "SELECT * FROM 店舗マスター")
dbFetch(res)

##   店舗番号  店舗名  開店時間  閉店時間  所在都道府県  電話番号
## 1         1   鯖江店         9        16       福井県 0770138207
## 2         2   銀座店        10        21       東京都 0382038460
```

29

DBIパッケージは、DataBase Interface、つまりRDBMSに接続するための機能を提供します。実際には、DBIパッケージは抽象化（統合）されたインターフェースを提供するもので、個別のRDBMSに接続するためのパッケージも合わせて必要です。従来、MySQLにはRMariaDB（※）、PostgreSQLにはRPostgresなど、それぞれパッケージがあり、使用法が異なっていましたが、DBIはそれを統一し、「どのDBを使っても、Rユーザーの書くプログラムは同じ」環境を実現しています。

基本的には、`dbConnect()` 関数でDBに接続し、`dbSendQuery()` 関数でSQLを発行します。また、取得した結果（プログラム例では `res`）を、tidyverseのパイプで続けて処理していくこともできます。

なお、DBIパッケージはSQLでの操作が前提ですが、tidyverseの文法で、SQLを意識せずにDBを操作できるdbplyrパッケージなどもあります。

- 参考: A dplyr backend for databases • dbplyr <https://dbplyr.tidyverse.org/>

※オープンソースであったMySQLをOracle社が買収し、商用化したことに反発したユーザーが分岐（fork）したのがMariaDBです。 <https://mariadb.org/>

5.2 jsonliteパッケージによるJSONデータの操作

- <https://github.com/jeroen/jsonlite>
- JSON形式のデータを扱うためのパッケージ

```
library(jsonlite)

url <- "https://sysbird.jp/toriko/api/?apikey=guest&
      format=json&keyword=ポテトチップス&max=10"

dat <- fromJSON(url)

head(dat[["item"]], 3)
```

30

jsonliteパッケージは、JSON (Javascript Object Notation) 形式のデータを扱うためのパッケージです。近年 (といってもここ10年以上ですが) Web APIなど、さまざまな場面でJSONデータを扱う機会が増えています。RでJSONデータを扱うパッケージも複数ありますが、現在ではjsonliteパッケージが最も広く使われています。

基本的には、`fromJSON()` 関数の引数にファイルやURLを指定すれば、JSONの構造を解析して、データフレームとして出力してくれる、というだけです。ただ、JSONがネスト (入れ子) している場合は、リストとして返ってくるので、前述のpurrrパッケージの `map()` 系関数と組み合わせて処理することになります。あるいは、リストの各要素の型判定をjsonliteパッケージに任せてよい、というのであれば、`fromJSON(..., simplifyVector = TRUE)` とすると、パッケージ側で最大限フラットなデータフレームになるよう解釈してくれます。

5.3 xmlconvertパッケージによるXMLデータの操作

- <https://github.com/jsugarelli/xmlconvert/>
- XML形式のデータを扱うためのパッケージ
- 一般的にはxml2パッケージが使われているが、こちらも便利

```
library(xmlconvert)
url <- "https://sysbird.jp/toriko/api/?apikey=guest&format=xml&
      keyword=ポテトチップス&max=10"
dat <- xml_to_df(url, records.tags = "item")
head(dat, 3)
```

##	id	name kana	maker	price	type
## 1	10992	いぶりがっこポテトチップス	NA 三真	100	snack
## 2	10974	ドラゴンポテトサッポロポテトバーベQ味	NA 三真	135	snack
## 3	10908	ポテトチップスやみつきフレンチサラダ	NA カルビー	147	snack
...					

31

xmlconvertパッケージは、XML (eXtensible Markup Language) を処理するためのパッケージです。近年はJSONが優勢ですが、Web APIの出力として、XMLが返ってくるケースはそれなりにあります。また、地図や空間情報などの分野では、KMLという標準が策定されているため、地理空間情報システム (GIS) ではXMLを扱う機会が多いでしょう。

RでXMLを扱う際に広く使われているのはXML2パッケージです。XMLの構造やタグなどの詳細な指定ができます。一方、ここで紹介するxmlconvertパッケージは、「XMLをデータフレームに変換する」ことに特化したパッケージです。処理したいXMLが典型的な表構造に落とし込めそうであれば、xmlconvertパッケージを使う方が簡単でしょう。

5.4 forcatsパッケージによる factor型データの操作

- <https://forcats.tidyverse.org/>
- カテゴリー変数 (factor型) のデータを効率的に扱うためのパッケージ
- `fct_reorder`(変数名, 基準, 関数): 基準列の集計結果に応じて順序を並べ替える
- `fct_infreq`(変数名): ラベルの出現頻度に応じて順序を並べ替える
- `fct_lump`(変数名, n=ラベル数, other_level="列名"): 上位n個のラベルを残し、他を「その他」にまとめる

32

forcatsパッケージは、Core Tidyverseの一部です。主にfactor型のデータを加工する際に使います。factor型は、表示上は「男性」「女性」といったラベルですが、内部では "男性: 2", "女性: 1" (文字コード順) というように数値で管理されており、順序 (水準, levels) があります。このfactor型の仕組みは、特にggplot2などを使ってグラフィックスを作成する際に、グラフ (棒や折れ線) が意図した順番で並ばない、というトラブルにつながります (実際にはRは内部の順序通りに並べているだけで、正しい挙動ですが)。

factor型のlevelsを入れ替えるには、組み込みの `factor(..., levels = c("水準1", "水準2", ...))` 関数なども使えます。ただ、forcatsパッケージではデータの出現頻度に応じた並べ替えなど、より柔軟な機能を提供します。

- 参考1: 【R前処理講座29】 {forcats}: ファクター処理 【tidyverse】 - データサイエンスの道標 <https://datasciencemore.com/forcats/>

なお、このプログラム例では、dplyrパッケージの `count()` 関数を使っています。データの件数をカウントするには、`group_by(グループ化変数) %>% summarise(count = n())` といった記述もできますが、`count()` 関数はよりシンプルな記述で集計結果を得られます。

- 参考2: Count observations by group — count • dplyr
<https://dplyr.tidyverse.org/reference/count.html>

5.4 forcatsパッケージによる factor型データの操作

```
library(tidyverse) #library(forcats)

starwars %>% filter(!is.na(species)) %>%
  mutate(new_species = fct_lump(species, n = 3, other_level = "その他")) %>%
  count(new_species)

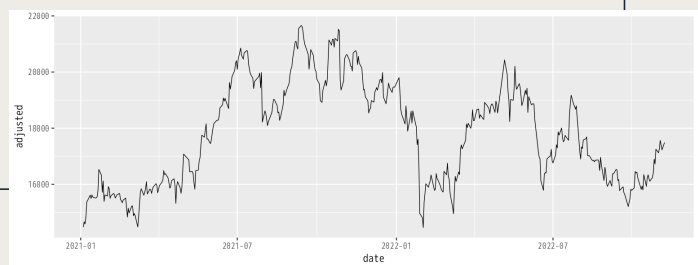
## # A tibble: 4 × 2
##   new_species      n
##   <fct>         <int>
## 1 Droid           6
## 2 Gungan          3
## 3 Human          35
## 4 その他         39
```

5.5 tidyquantパッケージによる 株価データの取得

- <https://github.com/business-science/tidyquant>
- 金融系データの取得・分析を行うためのパッケージ
- 日本の株価データも取得できる
- `tq_get("銘柄コード.T", get = "stock.prices", from = "取得開始日", to = "取得終了日")`

```
library(tidyquant)
# 富士通（株）の株価を取得
fj_stock <- tq_get("6702.T", get = "stock.prices", from = "2022-01-01")
```

```
library(ggplot2)
ggplot(fj_stock, aes(x = date,
  y = adjusted)) + geom_line()
```



34

tidyquantパッケージは、株価や為替など、金融系のデータを取得し、分析するためのパッケージです。名前の通り、tidyverseを意識した設計になっています。大規模なパッケージなので、詳しくは取り上げませんが、`tq_get()` 関数に `get = "stock.prices"` オプションを指定することで、米Yahoo! Financeから、企業の株価を範囲を指定して取得できます。日本の市場もカバーしているので、「当該企業がY! Finance上で何というコードで管理されているか」さえ確認すれば、取得できます。例えば、このページのプログラム例では、富士通株式会社の株価を取得しています。日本では個別の株は銘柄コードで管理されており、富士通の銘柄コードは6702です。そして、Y! Financeでは東京市場の、という意味でTが付与され管理されています。そこで、引数に 6702.T と指定しています。アメリカでは、企業名などを短縮したティッカーシンボルという文字列が使われています。例えばAppleは、AAPLとなっています。

他に、(tidyverse関連、という本節の意図からはずれませんが) `yahoofinancer` という、名前の通りY! Financeからデータを取得することに特化したパッケージも使われています。

- 参考: GitHub - rsquaredacademy/yahoofinancer: Obtain historical and near real time data from Yahoo Finance <https://github.com/rsquaredacademy/yahoofinancer>

6. まとめ

6.1 今日の内容

- lubridateパッケージによる日時データの操作
- stringrパッケージによる文字列データの操作
- tidyverseを中心としたエコシステム
 - DBIパッケージによるデータベースアクセス
 - jsonliteパッケージによるJSONデータの操作
 - xmlconvertパッケージによるXMLデータの操作
 - forcatsパッケージによるfactor型データの操作
 - tidyquantパッケージによる株価データの取得

6.2 次回までの課題

- Posit Cloudプロジェクト内の
`08_tidyverse_02_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: 2023-11-21 23:59