R で読む Excel ファイル（仮）

やわらかクジラ @matsuchiy

目次

はじめに

本書を書こうと思ったのは「R と RStudio を使いたい！と思う人がもっと増えればいいのに」という願いからです。使う人が多くなれば、新しい知識に出会いやすくなりますし、仕事でも使う機会が増える可能性があります。

使う人を増やすためにはよい入門書や web サイトが必要ですが、それは巷にあふれていて無料でアクセスできるものも多いです。

例えば

- R for Data Science（英語）[*1]
- 日本社会心理学会　第 5 回春の方法論セミナー　R と Rstudio 入門 [*2]

そこで本書では目的を絞って、R（実際はすべて RStudio から使います）を使いたいと思わせる部分を解説することを目指します。R でどんな便利なことができるか、入門書などでもあまり深く解説されてない部分にフォーカスして紹介します。

本書の特徴

便利なことといってもいろいろあるので、その中でも、つまづくと嫌になってしまうことの多そうな、**「手元の excel ファイルを読み込む」**所に着目しました。このトピックだけを R のいわゆるモダンな方法を使って重点的に解説した資料は、筆者の知る範囲では見つけられてないので、本書の最も際立った特徴といえます。また、関連するファイル形式として、csv ファイルの読み書きも少しだけ触れています。こちらはつまづくことの多いであろう文字コードについても解説しました。

[*1] https://r4ds.had.co.nz/
[*2] https://kazutan.github.io/JSSP2018_spring/index.html

解析したいデータは山ほどあれど、世の中綺麗なデータばかりとは限りません。たとえば、会社の部署ごとに分かれたデータなど、解析に持っていくまでに大量のファイルを読みこまなければいけない場合もあります。その読み込みの際にいかに楽をできるかという点を意識しています。データさえスムーズに読み込めれば、後はすぐれた解説がネット上でもたくさんあり、やりたいことが可能になる環境が整うからです。

excel ファイルをただ読みこむといってもいろんなバリエーションが考えられます。その単なる読みこみプロセスを通じて、R を使う上で便利な様々な関数や手続きを学ぶこともできるでしょう。**戦いの中で自然に強くなった的な効果**も見込めるかもしれません。

本書の内容は、github レポジトリの https://github.com/izunyan/excel_r ですべて公開しています。コードやサンプルデータはこちらのレポジトリをダウンロードしてお試しください。本書の発行後 6 か月をめどに、html 版を読めるように準備予定です。自力でできる方は、`Build Book` ですぐにでも読むことができます。

想定読者

色々な excel ファイルを読み込んで分析する機会があるのであれば、全く R のことを知らない方から、少し R の経験があるけど複数のファイルを一度に読みこんだことはないというレベルの方ぐらいまでが対象となるでしょう。

本書の到達目標は、R での excel ファイルの読み書きレベルをある程度高める、という所に定めました。その先は是非好きなように可視化なり解析なり進めていただければと思います。その一助として、特別付録として可視化のためのggplot2 の辞書も紹介しています。

本書の構成

まず1章では、RStudio でファイルを読み書きする際に、最低限知っておいた方がよい知識について解説しておきます。とっつきにくいかもしれませんが、知っておいてよかったと後になって実感する類のものなので、使って慣れていきましょう。

2章から4章は本書のメインである excel ファイルの読み込みについて解説します。一つのファイルの読み込みから、複数シート、複数ファイルの読み込みまで、様々なシーンに対応しました。また、読みこんだファイルを一つのデータフレームにまとめる方法についても触れています。

5章と6章は excel ファイルの保存についてです。ここでも、一つのファイルの保存から、複数ファイルの保存まで解説します。ここまでの内容が理解できれば、大量ファイルの読み書きにまつわる単純な繰り返し作業とはさよならできるでしょう。

7章は関連知識として csv ファイルの読み込み、8章は csv ファイルの保存について解説します。windows ユーザーは文字コードの違いによる文字化けという深い闇と対峙することになり、初心者はここで脱落していくことが多いのではないかと思います。そのために、サバイバルスキルとして知っておくことが有用だと思い書いておきました。自分が相当苦しんだので…

執筆環境

- 本書はbookdownにて執筆しました [*3]
- R および RStudio、パッケージのバージョン
  - R version 3.6.1
  - RStudio version 1.3.1073
  - readxl version 1.3.1
  - tidyverse version 1.3.0

注意事項など

- 本書の内容はすべて windows 環境を想定しています。
- この本に書いてある内容は、筆者が学習したことをまとめているものにすぎないため、正常な動作の保証はできません。使用する際は、自己責任でお願いします。

[*3] https://bookdown.org/

第 1 章

前提：プロジェクトの設定

R の基本的な使い方は他の情報源にゆだねていますが、ここだけは避けて通れないので解説しておきます。

1.1　どのフォルダのファイルかの指定

excel ファイルに限らず、ファイルを R に読み込む際は、どのフォルダから読むのか、位置を正確に指定する必要があります。

そこで重要となる概念が、「作業フォルダ」というものです。

コンソールに `getwd()` と打って出てくるフォルダが現在の作業フォルダになります。

1.2　プロジェクトとは

RStudio の「プロジェクト」とは、作業フォルダにまつわる面倒な設定を意識しないですむ非常に便利な機能です。

ざっくり説明すると、データを加工して解析する際に、1 つのフォルダ（サブフォルダも含む）の中に関連するデータやコード、出力をまとめておき、そのフォルダをプロジェクトとして設定するのです。これにより、ファイルの読み書きの際の場所指定をいちいち意識しないで作業できるようになります。

第 2 章

一つの excel ファイルを読み込む

- excel ファイルを読みこむには、`readxl` パッケージを使います。
- data フォルダ（`data/`で表現）に入っている「ペンギン.xlsx」を開きます。
- `read_xlsx()` 関数を使います。したがって、以下すべてファイル形式は`.xlsx` を想定します。
  - `read_xls()` や `read_excel()` もあるので、用途によって使い分けられます。ファイル形式が混ざっている時は `read_excel()` が有用かもしれません。

```
library(readxl)

df <-
 read_xlsx("data/ペンギン.xlsx") # excel ファイルの読み込み

df # データの表示
```

```
## # A tibble: 344 x 9
##   species 種類  island bill_length_mm bill_depth_mm
##   <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 Adelie  アデリー~ Torge~           39.1          18.7
## 2 Adelie  アデリー~ Torge~           39.5          17.4
## 3 Adelie  アデリー~ Torge~           40.3          18
## 4 Adelie  アデリー~ Torge~           NA            NA
## 5 Adelie  アデリー~ Torge~           36.7          19.3
## # ... with 339 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
```

```
## #   year <dbl>
```

上記コードを実行すると、RStudio の右上（デフォルトの配置であれば）の Environment タブに、

```
df  344 obs. of 9 variables
```

という表示が出ると思います。つまり、344 行のデータと 9 列の変数が入っているデータを、オブジェクト df として R に読みこんだ、ということを示しています。df <-の部分がその作業に該当します。

オブジェクト名である df と打つことで、デフォルトでは最初の 10 行分のデータが表示されます。ここでは紙面の都合で設定を変えているので 5 つだけにしています。表示された最初の行にも、A tibble: 344 x 9 と、行数 x 列数の情報が出ています。表示しきれなかった行は、with 339 more rows と省略され、表示しきれなかった列は、body_mass_g <dbl>, sex <chr>, year <dbl> と、名前とが表示されます。

なお、読みこんだファイルの保存については @ref(write_one_excel) 章で解説します。

2.1　最高な機能だよ！パスの自動補完

- read_xlsx("") と打った後に、" " の中にカーソルを置いて、tab キーを押すと、プロジェクトの中身が一覧で表示されるので、選んでいくだけで目的のファイルがキーボードを打つことなしに選べます！
  - RStudio は" " と打てばどこでもこの補完が可能です
  - ただし、R version 4.0 で文字コード関連の挙動が変わったみたいで、R 4.0.2 ではエラーになりました…（未解決）
- 上の階層のフォルダに行きたいときは、" " の中に../と打てば可能です。その後に tab キーを押せば上の階層のフォルダが選べます。

2.2　列名（変数名）がひどい場合の読み込み

```
read_xlsx("data/ペンギン（ひどい列名）ver.xlsx")
```

```
## # A tibble: 344 x 9
##   Ｓｐｅｃｉｅｓ `種　類` `*島の名前` `クチバシ　長さ（ｍｍ）`~
```

```
##   <chr>          <chr>    <chr>                    <dbl>
## 1 Adelie         アデリー Torgersen                 39.1
## 2 Adelie         アデリー Torgersen                 39.5
## 3 Adelie         アデリー Torgersen                 40.3
## 4 Adelie         アデリー Torgersen                 NA
## 5 Adelie         アデリー Torgersen                 36.7
## # ... with 339 more rows, and 5 more variables:
## #   `ｸﾁﾊﾞｼ__大きさ（ｍｍ）` <dbl>, `翼：長さ(mm)` <dbl>,
## #   `体重　単位はｇ` <dbl>, `<U+329B><U+329A>` <chr>,
## #   `２００７～２００９` <dbl>
```

読めることは読めますが、今後のデータ処理を進めるうえで不安が残ります。

2.2.1 スペースや記号などを自動的に変換してくれる関数できれいに

janitor パッケージの clean_names() 関数を使って、列名に入り込んでいるスペースや記号などを安全な記号に変換します。

なお、日本語の列名では、引数（ひきすう）に case = "old_janitor" をつけないと読みにくい結果になります。

```
library(tidyverse)
library(janitor)

read_xlsx("data/ペンギン（ひどい列名）ver.xlsx") %>%
  clean_names(case = "old_janitor")
```

```
## # A tibble: 344 x 9
##   ｓｐｅｃｉｅｓ 種_類 x_島の名前 x_クチバシ_長さ_ｍｍ~
##   <chr>          <chr> <chr>                    <dbl>
## 1 Adelie         アデリー~ Torgersen                 39.1
## 2 Adelie         アデリー~ Torgersen                 39.5
## 3 Adelie         アデリー~ Torgersen                 40.3
## 4 Adelie         アデリー~ Torgersen                 NA
## 5 Adelie         アデリー~ Torgersen                 36.7
## # ... with 339 more rows, and 5 more variables:
```

```
## #   x_ｸﾁﾊﾞｼ_大きさ_ｍｍ <dbl>, 翼_長さ_mm <dbl>,
## #   x_体重_単位はｇ <dbl>, x_u_329b_u_329a <chr>,
## #   ２００７_２００９ <dbl>
```

さて、ここで使われている %>% は非常に大事なので解説しておきます。

2.2.1.1 %>% とは？

「パイプ」と読みます。処理を重ねてコードに書いていきたい際に重宝し、現代の tidyverse を使った R のコードに欠かせないものです。

たとえば、df の species 列を選択する、という処理の

```
select(df, species)
```

は

```
df %>% select(species)
```

と書けます。%>% の左側にあるものを右側の最初の部分（第 1 引数）に渡すという働きです。パイプの利点は、いくつもつないで書いていけることです。たとえば、種類別にクチバシの長さの平均値を出したいときには次のようにできます。

```
df %>%
  group_by(species) %>%
  summarise(平均値 = mean(bill_length_mm, na.rm = TRUE))
```

```
## # A tibble: 3 x 2
##   species   平均値
##   <chr>      <dbl>
## 1 Adelie      38.8
## 2 Chinstrap   48.8
## 3 Gentoo      47.5
```

以下では%>% を多用していきます。

なお、ショートカット ctrl + shit + M（Mac だと Cmd + Shift + M）で出せます。

2.2.2 全角 ←→ 半角を自動で

stringi パッケージの stri_trans_nfkc() 関数を使って、変数名で全角-半角のばらつきを統一させます。

ここでは、変数名をリネームするのに dplyr 1.0.0 で登場した rename_with() 関数を使いました。すべての変数に対し、全角文字を含んでいたら半角に直すというコードになります。

```
library(stringi)
read_xlsx("data/ペンギン（ひどい列名）ver.xlsx") %>%
  rename_with(~stri_trans_nfkc(.),
              everything())
```

```
## # A tibble: 344 x 9
##   Species `種　類` `*島の名前` `1クチバシ　長さ(mm)`~ `2クチバシ_大きさ(mm)`~
##   <chr>   <chr>   <chr>                        <dbl>             <dbl>
## 1 Adelie  アデリー~ Torgersen                    39.1              18.7
## 2 Adelie  アデリー~ Torgersen                    39.5              17.4
## 3 Adelie  アデリー~ Torgersen                    40.3              18
## 4 Adelie  アデリー~ Torgersen                    NA                NA
## 5 Adelie  アデリー~ Torgersen                    36.7              19.3
## # ... with 339 more rows, and 4 more variables:
## #   `翼:長さ(mm)` <dbl>, `体重　単位はg` <dbl>, 女男 <chr>,
## #   `2007~2009` <dbl>
```

2.2.3 上記の合わせ技

%>% でつなぎ合わせて 1 つの実行で合わせてしまうこともできます。

```
read_xlsx("data/ペンギン（ひどい列名）ver.xlsx") %>%
  rename_with(~stri_trans_nfkc(.),
              everything()) %>%
    clean_names(case = "old_janitor")
```

```
## # A tibble: 344 x 9
##   species 種_類 x_島の名前 x1クチバシ_長さ_mm~ x2クチバシ_大きさ_mm~
##   <chr>   <chr> <chr>                  <dbl>            <dbl>
## 1 Adelie  アデリー~ Torgersen              39.1             18.7
## 2 Adelie  アデリー~ Torgersen              39.5             17.4
## 3 Adelie  アデリー~ Torgersen              40.3             18
## 4 Adelie  アデリー~ Torgersen              NA               NA
## 5 Adelie  アデリー~ Torgersen              36.7             19.3
## # ... with 339 more rows, and 4 more variables:
## #   翼_長さ_mm <dbl>, x_体重_単位はg <dbl>, 女男 <chr>,
## #   x2007_2009 <dbl>
```

2.3　開始行を指定して読みこむ

理想的なデータは 1 行目に列名が入力されている形ですが、最初の数行が空だったり、文字の説明が入っていたりすることも多いです。そうした場合は、以下のような読み込み結果になります。

```
read_xlsx("data/ペンギン（3 行空き）.xlsx")
```

```
## # A tibble: 346 x 9
##   ...1  ここに説明とか書いてあるファイ~ ...3  ...4  ...5  ...6  ...7  ...8
##   <chr> <chr>            <chr> <chr> <chr> <chr> <chr> <chr>
## 1 <NA>  <NA>             <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
## 2 spec~ island           bill~ bill~ flip~ body~ sex   year
## 3 Adel~ Torgersen        39.1  18.7  181   3750  male  2007
## 4 Adel~ Torgersen        39.5  17.4  186   3800  fema~ 2007
## 5 Adel~ Torgersen        40.3  18    195   3250  fema~ 2007
## # ... with 341 more rows, and 1 more variable: ...9 <chr>
```

列名が、セルに内容が入っている行から始まり、他の列名が...1, ...3 といったものになりました。

これを、指定した行から読み込むには、引数 skip = にとばしたい行の数を指定します。

```
read_xlsx("data/ペンギン（3 行空き）.xlsx", skip = 3)
```

```
## # A tibble: 344 x 9
##   species island bill_length_mm bill_depth_mm flipper_length_~
##   <chr>   <chr>           <dbl>         <dbl>            <dbl>
## 1 Adelie  Torge~           39.1          18.7              181
## 2 Adelie  Torge~           39.5          17.4              186
## 3 Adelie  Torge~           40.3          18                195
## 4 Adelie  Torge~           NA            NA                 NA
## 5 Adelie  Torge~           36.7          19.3              193
## # ... with 339 more rows, and 4 more variables:
## #   body_mass_g <dbl>, sex <chr>, year <dbl>, 種類 <chr>
```

このように、ちゃんと読むこむことができました。

2.4 セルを指定して読みこむ

引数 range = にセル範囲を指定すれば、そのセル範囲だけを読みこむこともできます。

```
read_xlsx("data/ペンギン.xlsx", range = "A1:D5")
```

```
## # A tibble: 4 x 4
##   species 種類     island    bill_length_mm
##   <chr>   <chr>    <chr>              <dbl>
## 1 Adelie  アデリー Torgersen           39.1
## 2 Adelie  アデリー Torgersen           39.5
## 3 Adelie  アデリー Torgersen           40.3
## 4 Adelie  アデリー Torgersen           NA
```

他にも、cell_cols = で読みたい列の指定、cell_rows = で読みたい行の指定も行えます。

詳細は、?readxl と打ち込んでヘルプを見るか、readxl のweb サイト [*1]を参照してください。

[*1] https://readxl.tidyverse.org/

第 3 章

シートを指定して読みこむ

3.1 シート名の確認

複雑な excel ファイルとなると、たくさんのシートが含まれていて、その全容を知るのも一苦労です。R では、`readxl` パッケージの `excel_sheets()` 関数でシート名の一覧を簡単に取得できます。

```
excel_sheets("data/ペンギン（シート別）.xlsx")
```

```
## [1] "アデリー"   "ジェンツー" "ヒゲ"
```

3.2 普通の読み込み

シートが分かれている excel ファイルをそのまま読みこんでみます。

```
read_xlsx("data/ペンギン（シート別）.xlsx")
```

```
## # A tibble: 152 x 9
##   species 種類  island bill_length_mm bill_depth_mm
##   <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 Adelie  アデリー~ Torge~           39.1          18.7
## 2 Adelie  アデリー~ Torge~           39.5          17.4
## 3 Adelie  アデリー~ Torge~           40.3          18
## 4 Adelie  アデリー~ Torge~           NA            NA
```

```
## 5 Adelie  アデリー~ Torge~           36.7          19.3
## # ... with 147 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
```

デフォルトでは一番最初のシートのデータが読みこまれます。ここでは、シート「アデリー」が読み込まれました。

3.3　シートを指定した読み込み

引数の sheet = にシート名を指定することで読み込めます。

```
read_excel("data/ペンギン（シート別）.xlsx", sheet = "ジェンツー" )
```

```
## # A tibble: 124 x 9
##   species 種類  island bill_length_mm bill_depth_mm
##   <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 Gentoo  ジェンツ~ Biscoe           46.1          13.2
## 2 Gentoo  ジェンツ~ Biscoe           50            16.3
## 3 Gentoo  ジェンツ~ Biscoe           48.7          14.1
## 4 Gentoo  ジェンツ~ Biscoe           50            15.2
## 5 Gentoo  ジェンツ~ Biscoe           47.6          14.5
## # ... with 119 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
```

3.4　すべてのシートから読み込み

ここで**一気にレベルが上がり**ますが、これこそが R を使って excel ファイルを読みこむ便利な部分なので、その魅力をみていきましょう。

```
path_name <- "data/ペンギン（シート別）.xlsx" # データのパスを格納

# シート名を取得しそれぞれから読み込んでリストにまとめる
```

```
df_list <-
  excel_sheets(path_name) %>%
  set_names() %>%           # 名前付きベクトルにする
  map(read_excel, path =  path_name)

df_list # 作成したリストの表示
```

```
## $アデリー
## # A tibble: 152 x 9
##   species 種類  island bill_length_mm bill_depth_mm
##   <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 Adelie  アデリー~ Torge~           39.1          18.7
## 2 Adelie  アデリー~ Torge~           39.5          17.4
## 3 Adelie  アデリー~ Torge~           40.3          18
## 4 Adelie  アデリー~ Torge~           NA            NA
## 5 Adelie  アデリー~ Torge~           36.7          19.3
## # ... with 147 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
##
## $ジェンツー
## # A tibble: 124 x 9
##   species 種類  island bill_length_mm bill_depth_mm
##   <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 Gentoo  ジェンツ~ Biscoe           46.1          13.2
## 2 Gentoo  ジェンツ~ Biscoe           50            16.3
## 3 Gentoo  ジェンツ~ Biscoe           48.7          14.1
## 4 Gentoo  ジェンツ~ Biscoe           50            15.2
## 5 Gentoo  ジェンツ~ Biscoe           47.6          14.5
## # ... with 119 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
##
## $ヒゲ
## # A tibble: 68 x 9
```

```
##   species 種類  island bill_length_mm bill_depth_mm
##   <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 Chinst~ ヒゲ  Dream            46.5          17.9
## 2 Chinst~ ヒゲ  Dream            50            19.5
## 3 Chinst~ ヒゲ  Dream            51.3          19.2
## 4 Chinst~ ヒゲ  Dream            45.4          18.7
## 5 Chinst~ ヒゲ  Dream            52.7          19.8
## # ... with 63 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
```

それぞれの excel シートから読みこまれた 3 つのデータ（アデリー、ジェンツー、ヒゲ）はデータフレームとして、df_list にリストとしてまとめて格納されています。リストは最初は理解が難しいですが、慣れるとなんでもリストにしたくなるくらい便利なものです。リストの中身を個別に取り出してみてみましょう

```
df_list$ジェンツー
```

```
## # A tibble: 124 x 9
##   species 種類  island bill_length_mm bill_depth_mm
##   <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 Gentoo  ジェンツ~ Biscoe           46.1          13.2
## 2 Gentoo  ジェンツ~ Biscoe           50            16.3
## 3 Gentoo  ジェンツ~ Biscoe           48.7          14.1
## 4 Gentoo  ジェンツ~ Biscoe           50            15.2
## 5 Gentoo  ジェンツ~ Biscoe           47.6          14.5
## # ... with 119 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
```

これは、df_list というリストの中の、ジェンツーという要素（ここではデータフレーム）を取り出す、というコードです。$ が「(左側にくるオブジェクト）の中の」という意味を表しています。自分でコードを打つと、df_list$ と打った時点で、中の要素の一覧が表示されるはずなので、そこからクリックして選ぶこともできます。

それでは、先ほど実行した読み込みコードの解説をします。

```
path_name <- "data/ペンギン（シート別）.xlsx"
```

これは、単にファイルの場所を path_name に格納しただけです。自分のデータで試してみたいときは、基本的にここのパス名を変えるだけで実行できるはずです。

```
df_list <-
  excel_sheets(path_name) %>%
  set_names() %>%
  map(read_excel, path =  path_name)
```

excel_sheets() は上で実行したのと同じです。実行結果はベクトルとして保存されています。set_names() は、ベクトルを名前付きベクトルにする働きをします。なので、ここでできるのは、

```
excel_sheets(path_name) %>%
  set_names()
```

```
##      アデリー     ジェンツー          ヒゲ
##   "アデリー" "ジェンツー"       "ヒゲ"
```

です。それぞれについて purrr パッケージの map() 関数を使って read_excel() を 1 つ 1 つのシート（ここでは作成した名前付きベクトルの要素）に適用していき、データを読み込みデータフレームにし、1 つのリストにまとめるという作業をします。

3.4.1 一つのデータフレームにする

bind_rows() は、データフレームを縦に連結します。データフレームがリストになったものが引数にくると、それらをすべて縦につなげてくれます。引数.id = で、リストの要素名を変数の値として入れることができるので、どのデータフレームから来たのか識別することが可能になります。ここでは group という名前にしています。

```
df_all <-
bind_rows(df_list, .id = "group")
```

3.4.1.1 作成したデータフレームの確認

dplyr パッケージの slice() 関数を使って、最初の 3 行と最後の 3 行だけを表示してどんなものができたか確認します。1:3 は 1 行目から 3 行目、(n()-2):n() は、列数（ただし現在の group 内）を表す n() とそれから-2 行した (n()-2) で表されています。

```
df_all %>%
  slice(1:3, (n()-2):n())
```

```
## # A tibble: 6 x 10
##   group species 種類  island bill_length_mm bill_depth_mm
##   <chr> <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 アデリー~ Adelie  アデリー~ Torge~           39.1          18.7
## 2 アデリー~ Adelie  アデリー~ Torge~           39.5          17.4
## 3 アデリー~ Adelie  アデリー~ Torge~           40.3          18
## 4 ヒゲ  Chinst~ ヒゲ  Dream            49.6          18.2
## 5 ヒゲ  Chinst~ ヒゲ  Dream            50.8          19
## 6 ヒゲ  Chinst~ ヒゲ  Dream            50.2          18.7
## # ... with 4 more variables: flipper_length_mm <dbl>,
## #   body_mass_g <dbl>, sex <chr>, year <dbl>
```

それぞれ、別々に出したほうが簡単かもしれません。それぞれ slice_head(), slice_tail() 関数を使って、引数 n = に表示したい行数を指定することで、最初および最後の数行を取得できます。

```
# 最初の 3 行
df_all %>% slice_head(n = 3)
```

```
## # A tibble: 3 x 10
##   group species 種類  island bill_length_mm bill_depth_mm
##   <chr> <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 アデリー~ Adelie  アデリー~ Torge~           39.1          18.7
## 2 アデリー~ Adelie  アデリー~ Torge~           39.5          17.4
## 3 アデリー~ Adelie  アデリー~ Torge~           40.3          18
## # ... with 4 more variables: flipper_length_mm <dbl>,
```

```
## #   body_mass_g <dbl>, sex <chr>, year <dbl>
```

```
# 最後の 3 行
df_all %>% slice_tail(n = 3)
```

```
## # A tibble: 3 x 10
##   group species 種類  island bill_length_mm bill_depth_mm
##   <chr> <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 ヒゲ  Chinst~ ヒゲ  Dream            49.6          18.2
## 2 ヒゲ  Chinst~ ヒゲ  Dream            50.8          19
## 3 ヒゲ  Chinst~ ヒゲ  Dream            50.2          18.7
## # ... with 4 more variables: flipper_length_mm <dbl>,
## #   body_mass_g <dbl>, sex <chr>, year <dbl>
```

第 4 章

複数の excel ファイルを読み込む

それでは、いよいよ R の恩恵を深く実感できる部分に入ります。読みこみたい excel ファイルが大量にある場合です。これも実務上よく遭遇します。

ただし、ここでは読みこむファイルの構造がすべて同様の場合に限ります。残念ながら、それがかなわない状況現実ではたくさん遭遇します。いつか本書の応用編を書くことがあったら、そちらで解説することとして、今回は構造が単純な場合に限って解説します。

4.1 読み込むファイル名の一覧のオブジェクト作成

まず、読みこみたいファイルが格納されているフォルダのファイル名、およびパス名の一覧を取得します。

```
files <-
    list.files(path = "data/複数/", full.names = TRUE)

files
```

```
## [1] "data/複数/アデリー.xlsx"   "data/複数/ジェンツー.xlsx"
## [3] "data/複数/ヒゲ.xlsx"
```

list.files() 関数は、path = で指定したフォルダ内の情報を取得します。full.names = TRUE でパスも含めます。これをつけないと、ファイル名と拡張子だけの取得になります。

4.2　ファイルを一括で読み込む

```
ldata <-
    map(files, ~read_xlsx(.))
```

ここでできた ldata は、3.4で作成した df_list と同じ構造です。違いはそれぞれのデータフレームの要素名（アデリー、ジェンツー、ヒゲ）が入っていない点です。これは不便なので、以下で要素名を改めてつけます。

4.2.1　ファイル名抽出

先ほど作成した files から、ファイル名部分だけに加工します。str_replace() は、stringr パッケージの、文字の置換をする関数です。ここでは、拡張子とパス名をそれぞれ""、つまり空白に置換しています。

```
file_name <-
  str_replace(files, ".xlsx", "") %>%
  str_replace("data/複数/", "")

file_name
```

```
## [1] "アデリー"   "ジェンツー" "ヒゲ"
```

4.2.2　リストの要素名にファイル名を付与

```
ldata <-
    set_names(ldata, file_name)

ldata
```

```
## $アデリー
## # A tibble: 152 x 9
```

```
##   species 種類  island bill_length_mm bill_depth_mm
##   <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 Adelie  アデリー~ Torge~           39.1          18.7
## 2 Adelie  アデリー~ Torge~           39.5          17.4
## 3 Adelie  アデリー~ Torge~           40.3          18
## 4 Adelie  アデリー~ Torge~           NA            NA
## 5 Adelie  アデリー~ Torge~           36.7          19.3
## # ... with 147 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
##
## $ジェンツー
## # A tibble: 124 x 9
##   species 種類  island bill_length_mm bill_depth_mm
##   <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 Gentoo  ジェンツ~ Biscoe           46.1          13.2
## 2 Gentoo  ジェンツ~ Biscoe           50            16.3
## 3 Gentoo  ジェンツ~ Biscoe           48.7          14.1
## 4 Gentoo  ジェンツ~ Biscoe           50            15.2
## 5 Gentoo  ジェンツ~ Biscoe           47.6          14.5
## # ... with 119 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
##
## $ヒゲ
## # A tibble: 68 x 9
##   species 種類  island bill_length_mm bill_depth_mm
##   <chr>   <chr> <chr>           <dbl>         <dbl>
## 1 Chinst~ ヒゲ  Dream            46.5          17.9
## 2 Chinst~ ヒゲ  Dream            50            19.5
## 3 Chinst~ ヒゲ  Dream            51.3          19.2
## 4 Chinst~ ヒゲ  Dream            45.4          18.7
## 5 Chinst~ ヒゲ  Dream            52.7          19.8
## # ... with 63 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
```

これらを 1 つのデータフレームにまとめるには、3.4.1と同じ手順でできます。

```
bind_rows(ldata, .id = "group")
```

第 5 章

一つの excel ファイルを保存する

writexl パッケージの write_xlsx() 関数で、直接 excel ファイルとして出力ができます。ここでは、新しく作成したデータフレームを excel ファイルとして保存してみます。

5.1 カテゴリ別平均値の作成

クチバシの長さと大きさを表す変数である、bill_length_mm、bill_depth_mm について、平均値と欠損を抜いた n（ペンギン数）を種類別に計算します。

group_by() でカテゴリ別にしたい変数を指定し、summarise() で平均値と n を計算するコードが以下になります。

```
df_group_mean <-
  df %>%
  group_by(種類) %>%
  summarise(across(
                    c(bill_length_mm, bill_depth_mm), # ここに変数
                    list(m = ~mean(., na.rm = TRUE),
                         n = ~sum(!is.na(.))))
            )

knitr::kable(df_group_mean) # きれいな出力にするコード
```

種類	bill_length_mm_m	bill_length_mm_n	bill_depth_mm_m	bill_depth_mm
アデリー	38.79139	151	18.34636	
ジェンツー	47.50488	123	14.98211	
ヒゲ	48.83382	68	18.42059	

5.2　excel ファイルの保存

```
library(writexl)
```

```
write_xlsx(df_group_mean, "out/df_group_mean.xlsx")
```

out/の部分が出力先のフォルダを示しています。

5.2.1　ファイル名に自動で本日の日付を入れる

```
write_xlsx(df_group_mean,
           str_c("out/df_group_mean", "_", lubridate::today(), ".xlsx"))
```

第 6 章

複数のファイルを一度に保存する

これが活躍する場面としては、たとえばカテゴリ別（例：ペンギンの種類別、会社の部署別など）に集計した要約値をそのカテゴリ別に個々の excel ファイルにするといった状況が思いつきますので、それをやってみます。

6.1 データフレームをカテゴリ別に分割する

split() 関数を使うことで、カテゴリ別にデータフレームを分割し、リストにまとめた結果を作成できます。データは5.1で作成した df_group_mean を使います。

分割に使う変数は、df_group_mean$species のように、データフレーム名の後に $ をつけてその後に指定します。この変数の中身が、そのままリストの要素名になるので、後の処理がとても楽になります。

```
df_gmean_list <-
  split(df_group_mean, df_group_mean$種類)

df_gmean_list
```

```
## $アデリー
## # A tibble: 1 x 5
##   種類  bill_length_mm_m bill_length_mm_n bill_depth_mm_m
##   <chr>            <dbl>            <int>           <dbl>
## 1 アデリー~             38.8              151            18.3
## # ... with 1 more variable: bill_depth_mm_n <int>
```

```
##
## $ジェンツー
## # A tibble: 1 x 5
##   種類  bill_length_mm_m bill_length_mm_n bill_depth_mm_m
##   <chr>            <dbl>            <int>           <dbl>
## 1 ジェンツ~             47.5              123            15.0
## # ... with 1 more variable: bill_depth_mm_n <int>
##
## $ヒゲ
## # A tibble: 1 x 5
##   種類  bill_length_mm_m bill_length_mm_n bill_depth_mm_m
##   <chr>            <dbl>            <int>           <dbl>
## 1 ヒゲ              48.8               68            18.4
## # ... with 1 more variable: bill_depth_mm_n <int>
```

6.2　個別で Excel ファイルに保存する

purrr パッケージの imap() 関数を使って、リスト内の各データフレームに、それぞれの要素名をファイル名として、excel ファイルに出力します。

ここでは、リスト内の各要素を示すのが.x、要素名（位置）に当たるのは.y です。次々に代わるファイル名を作るのに、str_c() 関数で文字列を結合しています。

```
imap(df_gmean_list, ~write_xlsx(.x, path = str_c("out/", .y , ".xlsx")))
```

6.2.1　サンプルデータセット作成コード

ちなみに data/複数/フォルダにあるサンプルデータセットは以下のコードで作りました。

```
imap(df_list, ~write_xlsx(.x, path = str_c("data/複数/",.y , ".xlsx")))
```

6.3　一ファイルの複数シートに保存する

6.1で作成した、ペンギンの種類別クチバシの長さと大きさ平均値のデータを、個別のファイルでなく、一ファイルの複数シートに保存したいときは、とてもシンプルなコードで可能になります。

要素名のついたデータフレームのリストが作成されていれば、それを単純に `write_xlsx()` で出力するだけで完成します。

```
write_xlsx(df_gmean_list, "data/平均値（複数シート）.xlsx")
```

第 7 章

csv ファイルを読み込む

windows 環境で過ごしている日本語を使う R ユーザーにおいては、csv ファイルを扱う際に文字コードの違いという深い闇（いわゆる文字化け）に遭遇することが少なくありません。

出会う可能性が高い文字コードには、大きく `cp932`（`Shift-JIS`）と `UTF-8` という形式があり、一般的に業務で読みこもうとするファイルは前者であることが多いことを知っておくと役に立ちます。

7.1 一つの csv ファイルを読み込む

- csv ファイルを読みこむには、`readr` パッケージの `read_csv()` 関数を使います。`tidyverse` を読み込んだら一緒に読み込まれます。
  - `readr` では基本的に `utf8` の読み書きが想定されています。RStudio のメニューで以下の部分をを UTF-8 に変えておかないと、色々つらい思いをします。
    - `Tools > Global Options > Code > Saving > Default text encoding:`
    - `Tools > Project Options > Code Editing > text encoding:`
- `data > csv` フォルダ（ここでは > は階層関係を示し、コードでは `data/csv/`で表現）に入っている「ペンギン（ひどい列名）ver_utf8.csv」を開きます。
  - ㊛㊚列のみ、環境依存文字のため、csv にする時点で文字化けています…

7.1.1 UTF-8 でエンコードされた csv ファイル

```
df_csv <-
 read_csv("data/csv/ペンギン（ひどい列名）ver_utf8.csv")

df_csv
```

```
## # A tibble: 344 x 9
##   Ｓｐｅｃｉｅｓ `種　類` `*島の名前` `クチバシ　長さ（ｍｍ）`~
##   <chr>         <chr>    <chr>                    <dbl>
## 1 Adelie        アデリー Torgersen                 39.1
## 2 Adelie        アデリー Torgersen                 39.5
## 3 Adelie        アデリー Torgersen                 40.3
## 4 Adelie        アデリー Torgersen                 NA
## 5 Adelie        アデリー Torgersen                 36.7
## # ... with 339 more rows, and 5 more variables:
## #   `ｸﾁﾊﾞｼ＿大きさ（ｍｍ）` <dbl>, `翼：長さ(mm)` <dbl>,
## #   `体重　単位はｇ` <dbl>, `<U+329B><U+329A>` <chr>,
## #   `２００７～２００９` <dbl>
```

7.1.2 【文字化けの例】Shift-JIS でエンコードされた csv ファイル

正確には Shift-JIS の拡張版である cp932 でエンコードされたファイルです。変数名も文字化けして読みこみ自体できなくなるので、`clean_names()` で読める形式に変換しています。

```
read_csv("data/csv/ペンギン（ひどい列名）ver_cp932.csv") %>%
  clean_names(case = "old_janitor") %>%
  select(1:3) # 紙面の都合で最初の 3 つの変数に限定
```

```
## # A tibble: 344 x 3
##   s_u_0082_u_0090_u_008~ x_u_008e_ed_u_008~ x_u_0081_u_00a6_u_00~
##   <chr>                  <chr>              <chr>
```

```
## 1 Adelie                  "\x83A\x83f\x83\x~ Torgersen
## 2 Adelie                  "\x83A\x83f\x83\x~ Torgersen
## 3 Adelie                  "\x83A\x83f\x83\x~ Torgersen
## 4 Adelie                  "\x83A\x83f\x83\x~ Torgersen
## 5 Adelie                  "\x83A\x83f\x83\x~ Torgersen
## # ... with 339 more rows
```

日本語の変数名と、2 列目の日本語の値が文字化けします。

7.1.3 Shift-JIS でエンコードされた csv ファイル

これを読むためには，引数 `locale = locale(encoding = )` で Shift-JIS のファイルであることを指定する必要があります。

```
read_csv("data/csv/ペンギン（ひどい列名）ver_cp932.csv"
         , locale = locale(encoding = "cp932"))
```

```
## # A tibble: 344 x 9
##   Ｓｐｅｃｉｅｓ `種　類` `*島の名前` ` クチバシ　長さ（ｍｍ）`~
##   <chr>         <chr>    <chr>                  <dbl>
## 1 Adelie        アデリー Torgersen               39.1
## 2 Adelie        アデリー Torgersen               39.5
## 3 Adelie        アデリー Torgersen               40.3
## 4 Adelie        アデリー Torgersen               NA
## 5 Adelie        アデリー Torgersen               36.7
## # ... with 339 more rows, and 5 more variables:
## #   `ｸﾁﾊﾞｼ＿大きさ（ｍｍ）` <dbl>, `翼：長さ(mm)` <dbl>,
## #   `体重　単位はｇ` <dbl>, `??` <chr>,
## #   `２００７～２００９` <dbl>
```

7.1.4 read.csv() を使う場合

従来の csv を読む関数 `read.csv()` を使えば，デフォルトで Shift-JIS のファイルは読めます。

```
read.csv("data/csv/ペンギン（ひどい列名）ver_cp932.csv") %>%
    as_tibble() %>% # データフレームを tibble 型にし見やすい出力に
    head()
```

```
## # A tibble: 6 x 9
##   Ｓｐｅｃｉｅｓ 種.類 X.島の名前 X.クチバシ.長さ.ｍｍ.~
##   <fct>         <fct> <fct>                  <dbl>
## 1 Adelie        アデリー~ Torgersen              39.1
## 2 Adelie        アデリー~ Torgersen              39.5
## 3 Adelie        アデリー~ Torgersen              40.3
## 4 Adelie        アデリー~ Torgersen              NA
## 5 Adelie        アデリー~ Torgersen              36.7
## 6 Adelie        アデリー~ Torgersen              39.3
## # ... with 5 more variables: X.ｸﾁﾊﾞｼ.大きさ.ｍｍ. <dbl>,
## #   翼.長さ.mm. <int>, X.体重.単位はｇ <int>, X.. <fct>,
## #   ２００７.２００９ <int>
```

UTF-8 を読む場合は引数 encoding = で指定します。

```
read.csv("data/csv/ペンギン（ひどい列名）ver_utf8.csv",
         encoding = "UTF-8") %>%
    as_tibble() %>%
    head()
```

```
## # A tibble: 6 x 12
##   Ｓｐｅｃｉｅｓ.種.類..島の~ X.クチバシ.長さ.ｍｍ...~ X.体重.単位はｇ..U.32~
##   <fct>           <fct>           <fct>
## 1 Adelie          アデリー        Torgersen
## 2 Adelie          アデリー        Torgersen
## 3 Adelie          アデリー        Torgersen
## 4 Adelie          アデリー        Torgersen
## 5 Adelie          アデリー        Torgersen
## 6 Adelie          アデリー        Torgersen
## # ... with 9 more variables: ２００７.２００９.Adelie <dbl>,
## #   アデリー <dbl>, Torgersen <int>, X39.1 <int>, X18.7 <fct>,
```

```
## #   X181 <int>, X3750 <lgl>, male <lgl>, X2007 <lgl>
```

7.1.5 大きいデータなら fread()

これまで紹介した csv ファイルを読みこむための関数は、小規模なデータならそんなに時間はかかりませんが、データが数万行 × 数百列と大きくなってくると、時間がかかるようになります。

そこで大きく時間を短縮できるのが、data.table パッケージの fread() 関数です。実は筆者が一番使ってるのはこの関数です。

```
data.table::fread("data/csv/ペンギン（ひどい列名）ver_cp932.csv") %>%
  as_tibble()
```

```
## # A tibble: 344 x 9
##   Ｓｐｅｃｉｅｓ `種　類` `*島の名前` ` クチバシ　長さ（ｍｍ）`~
##   <chr>          <chr>    <chr>                  <dbl>
## 1 Adelie         アデリー Torgersen               39.1
## 2 Adelie         アデリー Torgersen               39.5
## 3 Adelie         アデリー Torgersen               40.3
## 4 Adelie         アデリー Torgersen               NA
## 5 Adelie         アデリー Torgersen               36.7
## # ... with 339 more rows, and 5 more variables:
## #   `ｸﾁﾊﾞｼ__大きさ（ｍｍ）` <dbl>, `翼：長さ(mm)` <int>,
## #   `体重　単位はｇ` <int>, `??` <chr>,
## #   `２００７～２００９` <int>
```

なお、UTF-8 でエンコーディングされた csv ファイルの場合は、引数に encoding = "UTF-8" を加えることで読み込めます。

第 8 章

csv ファイルを保存する

- csv ファイルを保存するには、`readr` パッケージの `write_csv()` 関数を使います。
  - ただし、出力された csv ファイルを Excel で開くとたぶん文字化けします。LibreOffice の Calc であれば、最初にダイアログボックスが開いて読む文字コードを選べます。

8.1 write_csv() を使う

先ほど読みこんだ `df_csv` と、保存先を`""` 中に指定します。

```
write_csv(df_csv, "out/df_csv_utf8.csv")
```

8.1.1 Excel で開いても読めるように

Excel で開いても読める、BOM（byte order mark, バイトオーダーマーク）付きファイルとして出力する関数です。

```
write_excel_csv(df_csv, "out/df_csv_utf8_forxl.csv")
```

8.2　write.csv() を使う

書き込みが遅いですが、文字化け回避の最終手段として、なぜかうまくいく時があるので、write.csv() も役に立ちます。R でデフォルトの文字エンコードが cp932 で、そちらを採用するからと思われます。

```
write.csv(df_csv, "out/df_csv_cp932.csv")
```

あとがき

これから書きます

著者：やわらかクジラ
発行： 2020 年 9 月 12 日
サークル名：ヤサイゼリー
twitter： @matsuchiy
印刷：印刷所名