# タイトル

著者名

# 目次

# はじめに

本書『**R で読む Excel ファイル**』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

解析したいデータが、綺麗な一つにまとまった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。pdf 版が読みたい方は、以下のページで無料で入手可能です。自力でできる方は、`Build Book` でも作成できます。

- 技術書典マーケットの販売ページ
  - https://techbookfest.org/product/4794168259903488?productVariantID=5913872206659584

## 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。とはいえ、そこでお好きなように！と言われても路頭に迷う方もいるかもしれないので、データの内容把握に関して、要約値や欠損値の一覧、簡単な可視化、相関の一覧についても少しだけ解説しました。その一助として、特別付録として本書と並行してまとめた、可視化のためのggplot2 の辞書（特別付録について参照）もあります。

なお、データをきれいにする過程（例：前処理、データクリーニング、データクレンジング、データラングリングなど）については多くの説明を要するため、本書の範囲を超えます。これはまた別機会にまとめられたらなと思っています。

## 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[*3]にまとめる方法についても触れています。

3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。

4章は関連知識として csv ファイルの読み込みと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つらみと対峙することになり、初学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…

5章は、読み込んだファイルの特徴をざっと把握する方法について解説しました。ここまでやれば、（きれいなデータであれば！）きっとデータ解析に入っていくことができるでしょう。

## 執筆環境

- 本書はbookdownにて執筆しました [*4]
- 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

[*3] 変数（列）とオブザベーション（行）が碁盤の目のようになった集まりの形のデータ。Excel であれば通常 1 行目に列名が入り、2 行目以降が個別のデータを表す。データ解析において便利で分かりやすいため、本書ではデータフレームの形で説明していく

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

## 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。

## 特別付録について

本書の執筆に先駆けて、順序が違う気がしますがまず付録の作成からはじめました。特別付録は以下でアクセス可能なオンライン付録となります。

- ggplot2 の辞書
  - https://izunyan.github.io/practice_ggplot2/

Twitter で応援してもらったら項目が増えていく仕様にしているので、もっと読みたい方はこちらのツイートに何らかのレスポンスください！

# 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。RStudio を開いて右上のところに設定アイコンがあります（Figure**??**）。

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。

したがって、本書に出てくる内容をご自身の PC 上で再現するには、Github レポジトリ https://github.com/izunyan/excel_r をダウンロードして、その中にある `excel_r.Rproj` ファイルを実行すればお手軽です。

詳しくは、先述の R for Data Science での解説が参考になります。[*1]

[*1] https://r4ds.had.co.nz/workflow-projects.html

# 第 2 章

# Excel ファイルの読み込み

## 2.1 一つの Excel ファイルを読み込む

### 2.1.1 ファイルの準備と読み込み

Excel ファイルを R に読み込むには、`readxl` パッケージが便利です。[*1]セクション1.2で述べた通り、ここからは、github レポジトリの https://github.com/izunyan/excel_r をダウンロードしてプロジェクトを開いて進めてみてください。data フォルダ（`data/`で表現）に入っている「ペンギン.xlsx」[*2]を読み込んでみましょう。

読み込みには `read_xlsx()` 関数を使います。したがって、以下すべて Excel ファイル形式は`.xlsx` を想定します。`read_xls()` や `read_excel()` という関数もあるので、ファイル形式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
```

[*1] 以下、まだインストールしていないパッケージがあれば、`install.packages("")` の"" 中にパッケージ名を入れて実行するか、RStudio の `Packages` タブの `Install` からインストールしてください

[*2] Horst AM, Hill AP, Gorman KB (2020). palmerpenguins: Palmer Archipelago (Antarctica) penguin data. R package version 0.1.0. https://allisonhorst.github.io/palmerpenguins/

```
## # 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 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
## # ... with 341 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、df という名前のもの（オブジェクト）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。<-の右側の内容を左側のオブジェクトに格納するという意味です。ここで読み込まれた形がデータフレーム（はじめに > 本書の構成の注参照）です。

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 3 行だけにしています。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 341 more rows と省略され、表示しきれなかった列は、flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>, year <dbl> と、名前 < データの型名 > 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3.1章で解説します。

#### 2.1.1.1 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。[*3]

[*3] 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。

### 2.1.2 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
## # ... with 341 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。

#### 2.1.2.1 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、clean_names() 関数に case = "old_janitor" をつけないと読みにくい結果になります。このように、関数の中に追加する情報のことを引数（ひきすう）と言います。引数によってさまざまな条件を変更することが可能になります。

```
library(tidyverse)
    # ↑は普通まず最初に読み込むパッケージですが、本書ではここで初めて使います
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
## # ... with 341 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。

#### 2.1.2.2 %>% 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、データフレーム 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、ペンギンの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。

#### 2.1.2.3 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。everything() という引数を含めているので、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(~ `2クチバシ_大き~
##   <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
## # ... with 341 more rows, and 4 more variables:
## #   翼:長さ(mm) <dbl>, 体重 単位はg <dbl>, 女男 <chr>,
## #   2007~2009 <dbl>
```

#### 2.1.2.4 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~
##   <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
## # ... with 341 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>
```

### 2.1.3 開始行を指定して読み込む

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") %>%
  select(1:3) # 紙面の都合のため最初の 3 列に制限
```

```
## # A tibble: 346 x 3
##   ...1    ここに説明とか書いてあるファイル読むのつら~ ...3
##   <chr>   <chr>                                       <chr>
## 1 <NA>    <NA>                                        <NA>
## 2 species island                                      bill_lengt~
## 3 Adelie  Torgersen                                   39.1
```

```
## # ... with 343 more rows
```

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。そしてデータがうまく読み込めていません。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  Torgersen           39.1          18.7               181
## 2 Adelie  Torgersen           39.5          17.4               186
## 3 Adelie  Torgersen           40.3          18                 195
## # ... with 341 more rows, and 4 more variables:
## #   body_mass_g <dbl>, sex <chr>, year <dbl>, 種類 <chr>
```

このように、ちゃんと読む込むことができました。

### 2.1.4 セルを指定して読み込む

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込むこともできます。

```
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 [*4]を参照してください。

## 2.2 シートを指定して読み込む

### 2.2.1 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"
```

### 2.2.2 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 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
## # ... with 149 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。

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

### 2.2.3 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  ジェンツー Biscoe           46.1          13.2
## 2 Gentoo  ジェンツー Biscoe           50            16.3
## 3 Gentoo  ジェンツー Biscoe           48.7          14.1
## # ... with 121 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
```

### 2.2.4 すべてのシートから一気に読み込み

今まで身につけた知識を使うと、すべてのシートからデータを読みたいときは、単純に

```
df_Adelie <-
  read_excel("data/ペンギン（シート別）.xlsx", sheet = "アデリー")
df_Gentoo <-
  read_excel("data/ペンギン（シート別）.xlsx", sheet = "ジェンツー")
df_Chinstrap <-
  read_excel("data/ペンギン（シート別）.xlsx", sheet = "ヒゲ")
```

と 1 つずつ読み込めばよいとわかります。しかし、これが 100 シート分あったら読むだけで多くの時間がかかり、うんざりしてしまうでしょう。また、疲労によりミスも起こるかもしれません。コードも長くて読みにくくなってしまいます。そんな時に**便利で正確で短くコードが書ける**素敵な方法があるのです。おそらくこれを知って使えるようになることが、初心者から中級者への第一歩になるのではないでしょうか。

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 ファイルを読み込む

便利な部分（その用途に限らず、コード書いてコンピュータに働いてもらうの最高！ってなる部分）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 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
## # ... with 149 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  ジェンツー Biscoe           46.1          13.2
## 2 Gentoo  ジェンツー Biscoe           50            16.3
## 3 Gentoo  ジェンツー Biscoe           48.7          14.1
## # ... with 121 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 Chinstrap ヒゲ  Dream            46.5          17.9
## 2 Chinstrap ヒゲ  Dream            50            19.5
## 3 Chinstrap ヒゲ  Dream            51.3          19.2
## # ... with 65 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 にリストと呼ばれる形式に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  ジェンツー Biscoe           46.1          13.2
## 2 Gentoo  ジェンツー Biscoe           50            16.3
## 3 Gentoo  ジェンツー Biscoe           48.7          14.1
## # ... with 121 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。繰り返し似たような作業をするときに、この map() 関数が非常に便利です。

#### 2.2.4.1 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")
```

#### 2.2.4.2 作成したデータフレームの確認

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。dplyr パッケージの 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 Torgers~           39.1          18.7
## 2 アデリー Adelie  アデリー Torgers~           39.5          17.4
## 3 アデリー Adelie  アデリー Torgers~           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  Chinstrap ヒゲ  Dream            49.6          18.2
## 2 ヒゲ  Chinstrap ヒゲ  Dream            50.8          19
## 3 ヒゲ  Chinstrap ヒゲ  Dream            50.2          18.7
## # ... with 4 more variables: flipper_length_mm <dbl>,
## #   body_mass_g <dbl>, sex <chr>, year <dbl>
```

一気にやるには slice() 関数が便利です。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     Chinstrap ヒゲ     Dream            49.6          18.2
## 5 ヒゲ     Chinstrap ヒゲ     Dream            50.8          19
## 6 ヒゲ     Chinstrap ヒゲ     Dream            50.2          18.7
## # ... with 4 more variables: flipper_length_mm <dbl>,
## #   body_mass_g <dbl>, sex <chr>, year <dbl>
```

## 2.3 複数の Excel ファイルを読み込む

それでは、さらに 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。やり方としては、2.2.4で使った方法とほぼ同じです。

ただし、ここでは読み込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に、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。

おそらく、単純に思いつく方法は、

```
df_Adelie    <- read_xlsx("data/複数/アデリー.xlsx")
df_Gentoo    <- read_xlsx("data/複数/ジェンツー.xlsx")
df_Chinstrap <- read_xlsx("data/複数/ヒゲ.xlsx")
```

と 1 つずつ読んでいく方法ですが、これも大量にあったら泣きたくなる作業です。1 つ 1 つファイル名を入力やコピペする間にいくらでもミスが生じます。2.1.1.1で紹介した自動補完を使っても、楽しいのは最初だけでしょう。そこで、ファイルを指定するところから極力人の手を介さず進めていく方法を次に解説していきます。

### 2.3.1 読み込むファイル名の一覧のオブジェクト作成

まず、読み込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"       "data/複数/フォルダ1"
## [5] "data/複数/フォルダ2"       "data/複数/フォルダ3"
```

`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。

このうち、使用するのは xlsx ファイルだけなので、文字列で該当するデータを取得する str_subset() を用い、以下のように限定します。

```
files <-
  files %>% str_subset("xlsx")

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"
```

### 2.3.2 ファイルを一括で読み込む

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

ldata
```

```
## [[1]]
## # 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 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
## # ... with 149 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
##
## [[2]]
## # A tibble: 124 x 9
##   species 種類       island bill_length_mm bill_depth_mm
```

```
##   <chr>   <chr>      <chr>           <dbl>         <dbl>
## 1 Gentoo  ジェンツー Biscoe           46.1          13.2
## 2 Gentoo  ジェンツー Biscoe           50            16.3
## 3 Gentoo  ジェンツー Biscoe           48.7          14.1
## # ... with 121 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
##
## [[3]]
## # A tibble: 68 x 9
##   species   種類  island bill_length_mm bill_depth_mm
##   <chr>     <chr> <chr>           <dbl>         <dbl>
## 1 Chinstrap ヒゲ  Dream            46.5          17.9
## 2 Chinstrap ヒゲ  Dream            50            19.5
## 3 Chinstrap ヒゲ  Dream            51.3          19.2
## # ... with 65 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
```

ここでできた ldata は、2.2.4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。各要素の上の部分にある名前が [[1]], [[3]], [[3]] と表示されています（つまり、1, 2, 3 の数値が割り当てられている）。要素名が入っていないのは不便なので、以下で要素名を改めてつけます。

#### 2.3.2.1 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"
```

#### 2.3.2.2 リストの要素名にファイル名を付与

2.2.4で使った set_names() 関数は、リストの要素名を付ける時にも使えます。リスト ldata の 3 つの要素に、file_name の中身を割り当てます。

```
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 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
## # ... with 149 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  ジェンツー Biscoe           46.1          13.2
## 2 Gentoo  ジェンツー Biscoe           50            16.3
## 3 Gentoo  ジェンツー Biscoe           48.7          14.1
## # ... with 121 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 Chinstrap ヒゲ  Dream            46.5          17.9
## 2 Chinstrap ヒゲ  Dream            50            19.5
## 3 Chinstrap ヒゲ  Dream            51.3          19.2
## # ... with 65 more rows, and 4 more variables:
## #   flipper_length_mm <dbl>, body_mass_g <dbl>, sex <chr>,
## #   year <dbl>
```

このように、ちゃんと要素に名前が付きました。これらを 1 つのデータフレームにまとめるには、2.2.4.1と同じ手順でできます。

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

### 2.3.3 読み込むファイルが複数フォルダにある場合

data/複数フォルダの中に、さらにフォルダ 1〜フォルダ 3 がありました。この中にもそれぞれ xlsx ファイルが入っていて、それぞれ読み込みたいとします。その場合は、先ほど使った list.files() 関数の引数、recursive = TRUE を追加します。これによって、フォルダの深い階層までもすべて読み込むことが可能になります。

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

files
```

```
## [1] "data/複数//フォルダ1/アデリー.xlsx"
## [2] "data/複数//フォルダ2/ジェンツー.xlsx"
## [3] "data/複数//フォルダ3/ヒゲ.xlsx"
## [4] "data/複数/アデリー.xlsx"
## [5] "data/複数/ジェンツー.xlsx"
## [6] "data/複数/ヒゲ.xlsx"
```

このうち、使用するのはフォルダ 1〜3 に入っている xlsx ファイルだけなので、str_subset() 関数で以下のように絞り込みます。ここの"" の中に入る文字列がどうな

るかが難しいところかもしれませんが、ここは正規表現という本書の範囲を超える世界なので、深入りはしません。

```
files <-
  files %>% str_subset(" フォルダ")
```

あとは2.3.2と同じ手順で読み込めます。

# 第 3 章

# Excel ファイルの保存

## 3.1 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。

### 3.1.1 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),  # 平均値計算
                        n = ~sum(!is.na(.))))        # 欠損のない人数計算
            )

# ここは結果の整形部分
df_group_mean <-
```

```
  df_group_mean %>%
  select(-bill_depth_mm_n) %>%  # n が同じなので列削除
  rename(n = bill_length_mm_n)  # 列名を n にリネーム


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

| 種類 | bill_length_mm_m | n | bill_depth_mm_m |
|---|---|---|---|
| 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 |

なお、ここで knitr::kable() という書き方が出てきますが、これは knitr パッケージの kable() 関数という意味です。関数を 1 回しか使わない場合などに library() で呼び出すのは無駄が多かったりするので、時々こういった記述が出てきます。

### 3.1.2 Excel ファイルの保存

```
library(writexl)
```

前節で作成したカテゴリ別平均値のデータフレーム df_group_mean を Excel ファイルとして保存します。保存は write_xlsx() 関数の中に、データフレームのオブジェクトと出力先のパスと保存ファイル名を入れるだけです。

```
write_xlsx(df_group_mean, "out/種類別平均値.xlsx")
```

out/の部分が出力先のフォルダを示しています。

#### 3.1.2.1 【応用】ファイル名に自動で本日の日付を入れる

これはちょっと応用技ですが、便利なので紹介しておきます。lubridate パッケージの today() 関数で、今日の日付を表示できるので、それを保存名を入れる時に組み込む（文字列なので区切って str_c() でくっつける）と、日付入りファイル名が作成できます。

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

## 3.2 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。

### 3.2.1 データフレームをカテゴリ別に分割してリストにする

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3.1.1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 4
##   種類     bill_length_mm_m     n bill_depth_mm_m
##   <chr>               <dbl> <int>           <dbl>
## 1 アデリー             38.8   151            18.3
##
## $ジェンツー
## # A tibble: 1 x 4
##   種類       bill_length_mm_m     n bill_depth_mm_m
##   <chr>                 <dbl> <int>           <dbl>
## 1 ジェンツー             47.5   123            15.0
##
```

```
## $ヒゲ
## # A tibble: 1 x 4
##   種類  bill_length_mm_m     n bill_depth_mm_m
##   <chr>            <dbl> <int>           <dbl>
## 1 ヒゲ              48.8    68            18.4
```

### 3.2.2 リストの各要素を個別で 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")))
```

#### 3.2.2.1 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")))
```

### 3.2.3 一つのファイルの複数シートに保存する

3.2.1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 4 章

# csv ファイルの読み込みと保存

windows 環境で過ごしていて日本語を使う 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つらみ（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。

## 4.1 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 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…

### 4.1.1 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
## # ... with 341 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>
```

### 4.1.2 【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。

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。

### 4.1.3 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
## # ... with 341 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>
```

### 4.1.4 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
```

```
## # 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
## # ... with 341 more rows, and 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.. <chr>, ２００７.２００９ <int>
```

UTF-8 を読む場合は引数 `encoding =` で指定します。が、列名がひどすぎたせいかちゃんと読めないので出力はスキップします…

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

### 4.1.5 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
## # ... with 341 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。

## 4.2 csv ファイルの保存

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。

### 4.2.1 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")
```

#### 4.2.1.1 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")
```

### 4.2.2 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")
```

# 第 5 章

# データ解析に向けて

ここまでの解説で、Excel ファイルと csv ファイルの読み込みについて一通り解説してきました。この段階では、まだデータがきれいでなくて、解析に入れないことも多いと思います。

## 5.1 要約値や欠損データの確認

まずどんなデータがどのように入っているか、欠損値（NA）はどれくらい発生しているか確認することは重要なプロセスです。変数の一覧を要約して確認することが簡単にできる `skim` パッケージの `skimr()` 関数を使って確認してみましょう。データは2.1.1で読み込んだ `df` を使います。

```
library(skimr) # version 2.1.2
```

```
skim(df)
```

データの型が数値である変数（ここでは `Variable type:numeric`）については、`hist` 列に簡単なヒストグラムが表示されます（図**??**）。しかしデータが大量な場合、動作が遅くなります。その場合は、ヒストグラムを描かない以下の関数が使えます。

```
skim_without_charts(df)
```

### 5.1.1 結果を Excel ファイルに出力する

as_tibble() 関数を使うことで、結果を一つのデータフレームにまとめることができます。

```
res_skim_df <-
skim(df) %>%
  as_tibble()

res_skim_df %>%
  head()        # 最初の 6 行を表示
```

```
## # A tibble: 6 x 17
##   skim_type skim_variable  n_missing complete_rate character.min
##   <chr>     <chr>              <int>         <dbl>         <int>
## 1 character species                0         1                 6
## 2 character 種類                   0         1                 2
## 3 character island                 0         1                 5
## 4 character sex                   11         0.968             4
## 5 numeric   bill_length_mm         2         0.994            NA
## 6 numeric   bill_depth_mm         2         0.994            NA
## # ... with 12 more variables: character.max <int>,
## #   character.empty <int>, character.n_unique <int>,
## #   character.whitespace <int>, numeric.mean <dbl>,
## #   numeric.sd <dbl>, numeric.p0 <dbl>, numeric.p25 <dbl>,
## #   numeric.p50 <dbl>, numeric.p75 <dbl>, numeric.p100 <dbl>,
## #   numeric.hist <chr>
```

これで好きなように加工して Excel ファイルとして出力することが可能になります。例えば、数値変数だけに絞る場合は

```
res_skim_df %>%
  filter(skim_type  == "numeric") %>%
  select(skim_type, skim_variable,
         n_missing, numeric.mean, numeric.sd)
```

```
## # A tibble: 5 x 5
##   skim_type skim_variable     n_missing numeric.mean numeric.sd
##   <chr>     <chr>                 <int>        <dbl>      <dbl>
## 1 numeric   bill_length_mm            2         43.9       5.46
## 2 numeric   bill_depth_mm             2         17.2       1.97
## 3 numeric   flipper_length_mm         2        201.       14.1
## 4 numeric   body_mass_g               2       4202.      802.
## 5 numeric   year                      0       2008.        0.818
```

また、group_by() を使ってグループ別に上記結果を出すことも可能です。

```
res_df_num_g <-
df %>%
  group_by(種類) %>%
  skim() %>%
  as_tibble() %>%
  filter(skim_type  == "numeric") %>%
  select(skim_type, skim_variable, 種類,
         n_missing, numeric.mean, numeric.sd)

res_df_num_g
```

```
## # A tibble: 15 x 6
##    skim_type skim_variable     種類       n_missing numeric.mean
##    <chr>     <chr>             <chr>          <int>        <dbl>
##  1 numeric   bill_length_mm    アデリー           1         38.8
##  2 numeric   bill_length_mm    ジェンツー         1         47.5
##  3 numeric   bill_length_mm    ヒゲ               0         48.8
##  4 numeric   bill_depth_mm     アデリー           1         18.3
##  5 numeric   bill_depth_mm     ジェンツー         1         15.0
```

```
##  6 numeric   bill_depth_mm      ヒゲ              0         18.4
##  7 numeric   flipper_length_mm アデリー          1        190.
##  8 numeric   flipper_length_mm ジェンツー        1        217.
##  9 numeric   flipper_length_mm ヒゲ              0        196.
## 10 numeric   body_mass_g        アデリー          1       3701.
## 11 numeric   body_mass_g        ジェンツー        1       5076.
## 12 numeric   body_mass_g        ヒゲ              0       3733.
## 13 numeric   year               アデリー          0       2008.
## 14 numeric   year               ジェンツー        0       2008.
## 15 numeric   year               ヒゲ              0       2008.
## # ... with 1 more variable: numeric.sd <dbl>
```

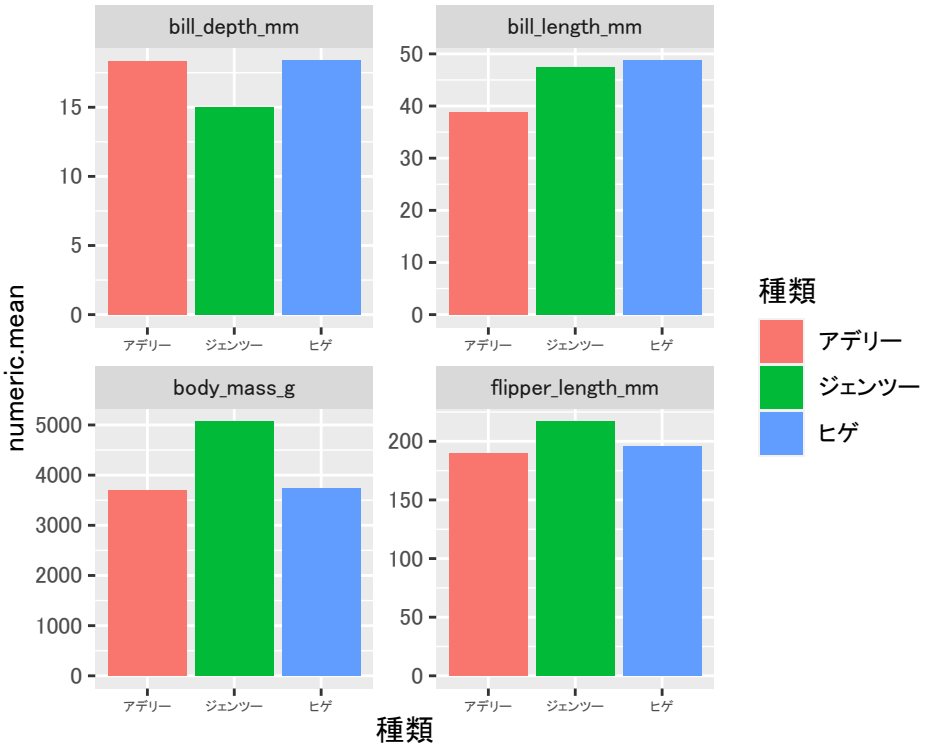
あとは、以下のように出力するだけです。

```
write_xlsx(res_df_num_g, "out/種類別平均値（全数値型変数）.xlsx")
```

### 5.1.2 可視化

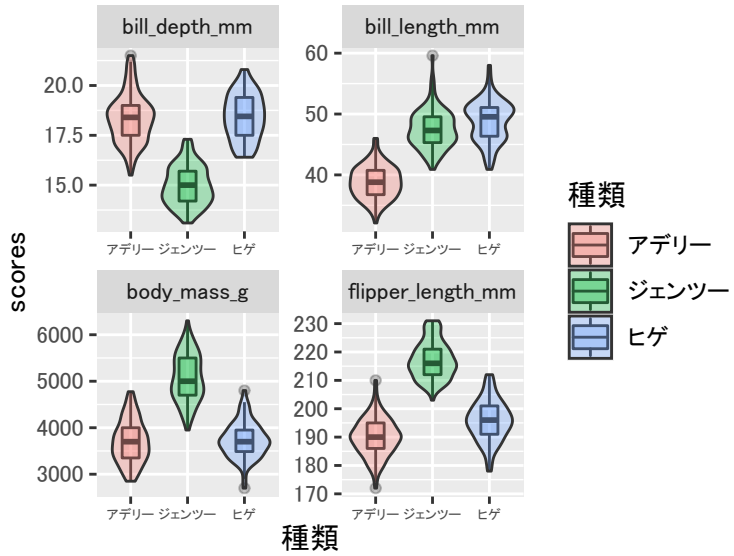
さらに ggplot2 を使って可視化することも可能になります。

```
res_df_num_g %>%
  filter(skim_variable %in% c("bill_length_mm", "bill_depth_mm",
                              "flipper_length_mm", "body_mass_g")) %>%
  ggplot(aes(x = 種類, y = numeric.mean, fill = 種類)) +
   geom_col() +
    theme(axis.text.x = element_text(size = 5)) +
   facet_wrap(vars(skim_variable), scale = "free")
```




ただし、ここの場合もっと潤沢な可視化グラフは元データから作成できます。例えば

```
df %>%
  select(種類, bill_length_mm, bill_depth_mm, flipper_length_mm,
         body_mass_g) %>%
  pivot_longer(-種類,                    # wide データから long データに変換
               names_to = "variables",
               values_to = "scores") %>%
  ggplot(aes(x = 種類, y = scores, fill = 種類)) +
   geom_boxplot(alpha = 0.3, width = 0.3) + # 箱ひげ図
   geom_violin(alpha = 0.3) +               # バイオリンプロット
    theme(axis.text.x = element_text(size = 5)) +
   facet_wrap(vars(variables), scales = "free")
```




詳しくは特別付録のggplot2 の辞書を参照ください。

## 5.2 相関の確認

変数同士の相関関係を見たいという要望はビジネス、アカデミックを問わず多く発生すると思います。簡単な相関行列の出し方やその可視化について解説します。ここで便利なパッケージが `corrr` です。

### 5.2.1 相関行列を出す

```
library(corrr)
```

まずペンギンデータの中の数値変数だけにしぼります。そして、year はここでは不要なので落とします。そうして作ったデータフレーム `cor_df` を、`correlate()` 関数に入れるだけです。

```
cor_df <-
df %>%
  select(where(is.numeric)) %>% # 数値変数だけにしぼる
  select(-year)                 # 不要なので落とす
```

```
correlate(cor_df)
```

```
## # A tibble: 4 x 5
##   term  bill_length_mm bill_depth_mm flipper_length_~ body_mass_g
##   <chr>          <dbl>         <dbl>             <dbl>       <dbl>
## 1 bill~         NA            -0.235             0.656       0.595
## 2 bill~         -0.235        NA                -0.584      -0.472
## 3 flip~          0.656        -0.584            NA           0.871
## 4 body~          0.595        -0.472             0.871      NA
```

上側と下型で相関係数が重複しているので、片側だけを残したい場合があります。その際は shave() 関数で簡単に重複部分をなくせます。引数に upper = FALSE と入れれば、上側だけにすることもできます。

相関係数の表示したい桁の指定は、fashion() 関数で、引数に decimals = で桁数を指定することで可能になります。

```
cormat <-
correlate(cor_df) %>%
  shave() %>%
  fashion(decimals = 1)

cormat
```

```
##                term bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
## 1    bill_length_mm
## 2     bill_depth_mm            -.2
## 3 flipper_length_mm             .7           -.6
## 4       body_mass_g             .6           -.5                .9
```

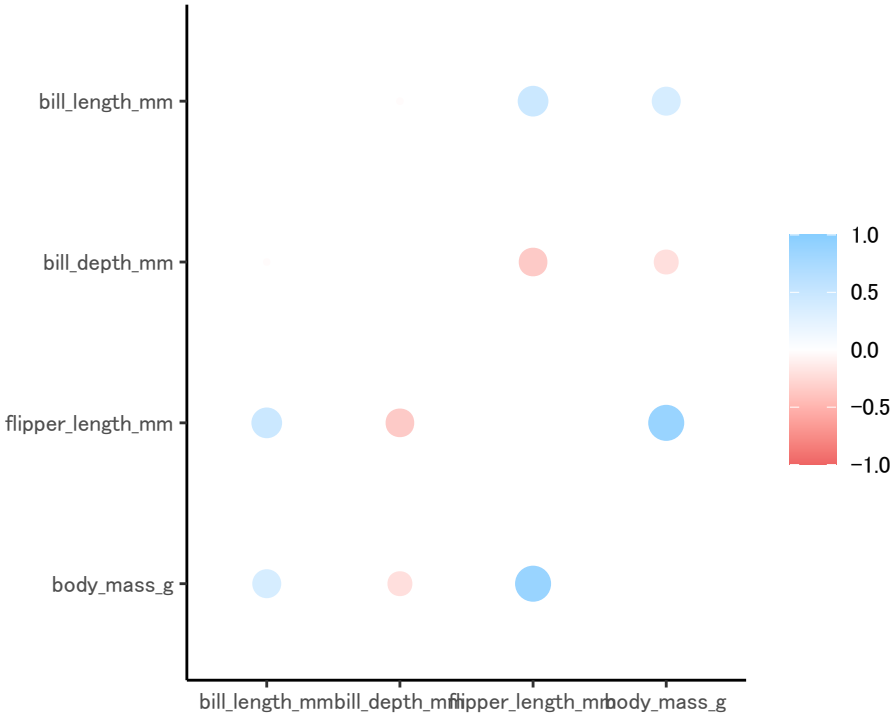
あとは、以下のように出力するだけです。

```
write_xlsx(cormat, "out/相関行列.xlsx")
```

### 5.2.2 相関行列の可視化

corrr パッケージの関数で相関行列の可視化も簡単にできます。

```
correlate(cor_df) %>%
  rplot()
```



# あとがき

初めての同人誌執筆が終わろうとしています。執筆に用いた Bookdown について前より理解が深まったと同時に、全然わからないことも増えました。。執筆途中で、メインで使っていたノート PC の充電ができなくなるというアクシデントにも見舞われました。こうした様々な学びを活かして、次回作につなげられればと思います。

本書の執筆にあたり、同人誌制作の先輩である天川榎 `@EnokiAmakawa` 氏から背中押し＆多くの助言をいただきました。この場を借りてお礼申し上げます。

この本の内容が、なるべく多くの方に届いて、みんなの仕事の効率化につながればうれしいです。そして、R の可能性がさらに多くの方に認識され、R を使ってお仕事できる環境が一層整っていくことを願ってやみません。

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：電子出版のみ