

# 心理学統計実習

Exercises in Psychological Statistics with R/RStudio

Koji Kosugi



# Table of contents

はじめに	7
ライセンス等	7
<b>第 1 章    はじめよう R/RStudio</b>	<b>9</b>
1.1    環境の準備	9
1.1.1    R のインストール	9
1.1.2    RStudio のインストール	10
1.1.3    環境の準備に関する導入サイト	10
1.2    RStudio の基礎（4つのペイン）	11
1.2.1    領域 1；エディタ・ペイン	12
1.2.2    領域 2；コンソール・ペイン	13
1.2.3    領域 3；環境ペイン	13
1.2.4    領域 4；ファイルペイン	13
1.2.5    そのほかのタブ	14
1.3    R のパッケージ	14
1.4    RStudio のプロジェクト	15
1.5    課題	16
1.6    References	17
<b>第 2 章    R の基礎</b>	<b>19</b>
2.1    R で計算	19
2.2    オブジェクト	20
2.3    関数	21
2.4    変数の種類	22
2.5    オブジェクトの型	22
2.5.1    ベクトル	23
2.5.2    行列	24
2.5.3    リスト型	25
2.5.4    データフレーム型	26
2.6    外部ファイルの読み込み	28
2.7    おまけ；スクリプトの清書	30
2.8    課題	30

2.9	References . . . . .	31
<b>第 3 章</b>	<b>R によるデータハンドリング</b>	<b>33</b>
3.1	tidyverse の導入 . . . . .	33
3.2	パイプ演算子 . . . . .	34
3.3	課題 1 . . . . .	35
3.4	列選択と行選択 . . . . .	36
3.4.1	列選択 . . . . .	36
3.4.2	行選択 . . . . .	38
3.5	変数を作る・再割り当てする . . . . .	39
3.6	課題 2 . . . . .	40
3.7	ロング型とワイド型 . . . . .	41
3.8	グループ化と要約統計量 . . . . .	44
3.9	データ整形課題 . . . . .	46
3.10	References . . . . .	46
<b>第 4 章</b>	<b>R によるレポートの作成</b>	<b>47</b>
4.1	Rmd/Quarto の使い方 . . . . .	47
4.1.1	概略 . . . . .	47
4.1.2	ファイルの作成と knitr . . . . .	48
4.1.3	マークダウンの記法 . . . . .	52
4.2	プロットによる基本的な描画 . . . . .	53
4.3	ggplot による描画 . . . . .	54
4.4	幾何学的オブジェクト geom . . . . .	56
4.5	描画 tips . . . . .	59
4.5.1	ggplot オブジェクトを並べる . . . . .	60
4.5.2	ggplot オブジェクトの保存 . . . . .	62
4.5.3	テーマの変更（レポートに合わせる） . . . . .	62
4.6	マークダウンと描画の課題 . . . . .	64
4.7	References . . . . .	65
<b>第 5 章</b>	<b>R でプログラミング</b>	<b>67</b>
5.1	代入 . . . . .	67
5.2	反復 . . . . .	67
5.3	条件分岐 . . . . .	67
5.4	自作関数 . . . . .	67
5.5	さまざまな乱数 . . . . .	67
5.6	確率分布の関数 . . . . .	67
5.7	References . . . . .	67
<b>第 6 章</b>	<b>確率とシミュレーション</b>	<b>69</b>
6.1	サンプリング関数 . . . . .	69

6.2	期待値と分散のシミュレーション	69
6.3	母集団と標本	69
6.4	一致性	69
6.5	不偏性	69
6.6	有効性	69
6.7	References	69
<b>第 7 章</b>	<b>統計的仮説検定の論理とエラー</b>	<b>71</b>
7.1	帰無仮説検定の論理	71
7.2	相関係数の検定	71
7.3	標本相関係数の分布	71
7.4	2 種類の検定のエラー確率	71
7.5	References	71
<b>第 8 章</b>	<b>平均値差の検定</b>	<b>73</b>
8.1	一標本検定	73
8.2	二標本検定	73
8.3	二標本検定（ウェルチの補正）	73
8.4	対応のある二標本検定	73
8.5	レポートを書くような課題	73
8.6	References	73
<b>第 9 章</b>	<b>多群の平均値差の検定</b>	<b>75</b>
9.1	分散分析の基礎	75
9.2	検定の多重性	75
9.3	ANOVA 君を使う	75
9.4	Between デザイン	75
9.5	Within デザイン	75
9.6	References	75
<b>第 10 章</b>	<b>帰無仮説検定のシミュレーション</b>	<b>77</b>
10.1	統計的検定と QRP	77
10.2	タイプ 2 エラー確率のコントロールとサンプルサイズ設計	77
10.3	サンプルサイズ設計の実践	77
10.3.1	一標本 $t$ 検定	77
10.3.2	二標本 $t$ 検定	77
10.3.3	相関係数のサンプルサイズ設計	77
10.4	References	77
<b>第 11 章</b>	<b>回帰分析</b>	<b>79</b>
11.1	回帰分析の基礎	79
11.2	重回帰分析の場合	79

11.3	回帰分析のいくつかの特徴 . . . . .	79
11.4	シミュレーションとパラメタリカバリ . . . . .	79
11.5	係数の標準誤差 . . . . .	79
11.6	係数の検定 . . . . .	79
11.7	サンプルサイズ設計 . . . . .	79
11.8	References . . . . .	79
<b>第 12 章</b>	<b>線型モデルの展開</b>	<b>81</b>
12.1	一般線型モデル . . . . .	81
12.2	一般化線型モデル . . . . .	81
12.3	階層線型モデル . . . . .	81
12.4	References . . . . .	81
<b>第 13 章</b>	<b>多変量解析の入り口</b>	<b>83</b>
13.1	因子分析 . . . . .	83
13.2	構造方程式モデリング . . . . .	83
13.3	References . . . . .	83

# はじめに

この資料は、授業「心理学統計演習」についてのものです。演習という授業名にあるように、理論的な解説で「理解して進む」ことよりも、「手を動かして理解する」ことを目的にしています。

この資料を活用する人は、理論的な（いわゆる座学の）心理学統計を履修済みであることを前提にしています。また、資料集という位置付けですので、行間の説明が省略されていることが多くあります。その点は講義時間中の講話で補完していくつもりですので、不明な点があれば授業時間中に質問してください。

## ライセンス等

この資料は Creative Commons BY-SA(CC BY-SA) ライセンス Version 4.0 に基づいて提供されています。著者に適切なクレジットを与える限り、この本を再利用、再編集、保持、改訂、再頒布（商用利用を含む）をすることができます。もし再編集したり、このオープンなテキストを変更したい場合、すべてのバージョンにわたってこれと同じライセンス、CC BY-SA を適用しなければなりません。

This article is published under a Creative Commons BY-SA license (CC BY-SA) version 4.0. This means that this book can be reused, remixed, retained, revised and redistributed (including commercially) as long as appropriate credit is given to the authors. If you remix, or modify the original version of this open textbook, you must redistribute all versions of this open textbook under the same license - CC BY-SA.





## 第 1 章

# はじめよう R/RStudio

「R」。この一文字で表現されるがゆえに、検索しにくいことこの上ないそれは、統計に特化したプログラミング言語であり、心理学はもちろん統計に関する学問領域で多岐にわたって利用されているものである。フリーソフトウェア、すなわち自由で開かれているソフトウェアであるから、ソースコードに至るまで公開されており、誰でも無償で利用できる。無償すなわち無料ではない。補償がないので無償なのだが、逆に金銭で計算をはじめ科学的真実性が保証されるわけではない、という至極まともな考え方は理解できるだろう。科学はもちろん、ソフトウェアも人類の共有財産として、オープンに育んでいこう。

R はコミュニティ活動も盛んで、Tokyo.R を中心に日本の各地で R ユーザからなる自主的な勉強会が開催されている<sup>\*1</sup>。また R 自体がインターネットを通じて公開されているように、導入から応用までさまざまな資料がオンラインで活用できる。以下では導入から解説していくが、頻繁にアップデートされるものでもあるので、必要に応じて検索し、なるべく時系列的に近い情報を吟味して活用することを薦める。

### 1.1 環境の準備

#### 1.1.1 R のインストール

R のインストールに関して、初心者でも利用可能な資料がオンラインで公開されている。

R は The [Comprehensive R Archive Network](#), 通称 CRAN<sup>\*2</sup> というネットワークで公開されている。CRAN のトップページにはダウンロードリンクが用意されており、自分のプラットフォームにあった最新版をダウンロードしよう<sup>\*3</sup>。

---

<sup>\*1</sup> 2024 年 1 月現在で、Tokyo だけでなく Fukuoka, Sapporo, Yamaguchi, Iruma など地方コミュニティがあり、参加者みんなで楽しまれている。

<sup>\*2</sup> CRAN は「しーらん」、あるいは「くらん」と発音される。筆者はしーらん派。

<sup>\*3</sup> この授業のために自身の PC に R をインストールしたとして、次に使うときに半年以上間隔が空いたのなら、改めて最新版をチェックし、バージョンが上がっていたら旧版をアンインストールして最新版をインストールするところから始めた方がよい。R で利用するパッケージなどが新しい版にしか対応していないことなどもある。R と量は新しい方がよい。

### 1.1.2 RStudio のインストール

R のインストールが終われば、次は RStudio をインストールしよう。RStudio は総合開発環境 (IDE) と呼ばれるものである。R は単体で、統計の分析や関数の描画など、専門的な利用に耐えうる分析機能を有している。その本質はもちろん計算機能であって、計算を実行する命令文 (スクリプト) を与えれば、必要な返答をあたえてくれる。このように分析の本質が計算機能であったとしても、実際の分析活動に際しては、スクリプトの下書きと清書、入出力データや描画ファイルの生成・管理、パッケージ (後述) の管理など、分析にまつわるさまざまな周辺活動が含まれる。喩えるなら料理の本質が包丁・まな板・コンロによる加工であったとしても、実際の調理に際しては、広い調理スペースや使いやすいシンク、ボウルやタッパーなどの補助的な調理器具があった方がスムーズにことが進む。いわば、R 単体で分析をするのは飯盒炊爨のような必要最低限かつワイルドな調理法であり、RStudio は総合的な調理環境を提供してくれるものなのである。

繰り返しになるが、本質的には R 単体で作業が可能である。なるべく単純な環境を維持したいというのであれば R 単体での利用を否定するものではないが、RStudio はエディタや文書作成ソフトとしても有用であるので、本授業では RStudio を使うことを前提とする<sup>\*4</sup>。

### 1.1.3 環境の準備に関する導入サイト

以下に執筆時点 (2024 年 1 月) で参照可能な、導入に関する Web 教材を挙げておく。自分に合ったものを適宜参照し、R と RStudio を自身の PC 環境に導入してほしい。もちろん自身で「R RStudio インストール」などとして検索しても良いし、chatGPT に相談しても良い。

#### 1.1.3.1 For Windows

- 東京大学・大学院農学生命科学研究科アグリバイオインフォマティクス教育研究プログラムによる [PDF 資料](#)
- [初心者向け R のインストールガイド](#)
- 関西学院大学商学部土方ゼミ [資料](#)
- 多摩大学情報社会研究所・応用統計学室 [資料](#)
- 奥村晴彦先生の [ページ](#)

#### 1.1.3.2 For Macintosh

- 東京大学・大学院農学生命科学研究科アグリバイオインフォマティクス教育研究プログラムによる [PDF 資料](#)
- note の [記事](#)
- いちばんやさしい、医療統計 [記事](#)

---

<sup>\*4</sup> VSCode のようなエディタから使うことも可能であるし、Jupyter Notebook の計算エンジンを R にすることも可能。最近では分析ソフトウェアを個々人で準備せず、環境として提供することも一般的になってきており、例えば [Google Colaboratory](#) のエンジンを R にすることもできるようになっている。ローカル PC に自前の環境を作るということが、時代遅れになる日も近いかもしれない。

なお、Mac の場合は Homebrew などのパッケージ管理ソフトを使って導入することもできる（し、そのほうがいい）。その場合は以下の資料を参照。

- 群馬大学大学院医学系研究科機能形態学の[記事](#)
- コアラさばお氏の note [記事](#)
- Ryu Takahashi 氏の Qiita [記事](#)
- Yuhki Yano 氏の Qiita [記事](#)

## 1.2 RStudio の基礎（4つのペイン）

ここまでで、R および RStudio を利用する準備が整っているものとする。

さて、RStudio を起動すると大きくわけて 4 つの領域に分かれた画面が出てくる。この領域のことをペインと呼ぶ。図中の「領域 1」がないように見えるときもあるが、下のペインが最大化され折りたたまれているだけなので、ペイン上部のサイズ変更ボタンを操作することで出てくるだろう。



Figure1.1: RStudio の初期画面

このペインのレイアウトは、メニューの Tools > Global Options... > Pane Layout から変更することもできる。基本的に 4 分割であることに変わりはないが、自分が利用しやすい位置にレイアウトを変更するとよい。



Figure1.2: レイアウト変更画面。このほかにも背景色などを変えることもできる

以下、各ペイン (領域) が何をするとところかを簡単に解説する。

### 1.2.1 領域 1 ; エディタ・ペイン

エディタ領域。R のスクリプトはもちろん、レポートの文章など、基本的に入力するときはこのペインに書く。ここで作業するファイルの種類は、File > New File から見ると明らかなように、R 言語だけでなく C 言語、Python 言語などのスクリプトや、Rmd, md,Qmd,HTML などのマークアップ言語、Stan や SQL など特殊な言語など

にも対応している。ペインの右下に現在開かれているファイルの種類が表示されているのを確認しておこう。

R 言語でスクリプトを書く例で解説しよう。R は命令を逐次実行していくインタプリタ形式であり、ここに記述された R コードを、右上の Run ボタンでコンソールに送って計算を実行するように使う。一回の命令をコマンド、コマンドが積み重ねられた全体をスクリプト、あるいはプログラムと呼ぶ。複数のコマンドを実行したい場合は、エディタ領域で複数行選択して Run ボタンを、スクリプトファイル全体を実行したいときは Run ボタンのとなりにある Source を押す。CTRL+Enter (Mac の場合はコマンド +Enter) で Run ボタンのショートカットになる。

### 1.2.2 領域 2；コンソール・ペイン

R 単体で利用する場合は、このペインだけを利用するようなものである。すなわち、ここに示されているのが R 本体というか、R の計算機能そのものである。ここに「>」の記号が表示されているところをプロンプトといい、プロンプトが表示されているときは R が入力待ちの状態である。

R は逐次的に計算を行うので、プロンプトのある状態でコマンドを入力すると計算結果が返される。ここに直接コマンドを書いて行っても良いが、書き間違えたりすることもあるし、コマンドが複数行に渡ることが一般的になってくるので、エディタ領域に清書するつもりで記述していったほうがよい。ごくたまに、一時的に確認したいことがある時だけ、直接コンソールを触るようにすると良い。

なお、コンソールを綺麗にしたいときは右上の箒ボタンをおすとよい。

### 1.2.3 領域 3；環境ペイン

基本的にこのペインと次の領域 4 のペインは複数のタブが含まれる。Pane Layout でどちらにどのタブを含めるかを自分好みにカスタマイズすることもできる。ここでは代表的な 2 つのタブについてのみ言及する。

**Environment** タブは、R の実行メモリ内に保管されている変数や関数などが表示されている。「変数や関数など」をまとめて**オブジェクト**というが、ここで内容や構造を GUI で確認することができる。

**History** タブは履歴である。これまでコンソールに送られてきたコマンドが順に記録されている。History タブからエディタ、コンソールにコマンドを送ることも可能であり、「さっきの命令をもう一度実行したい」といったときに参照すると良い。

### 1.2.4 領域 4；ファイルペイン

ここでも代表的なタブについてのみ解説する。

**Files** タブは Mac でいう Finder、Windows でいうエクスプローラーのような、ファイル操作画面である。フォルダの作成、ファイルの削除、リネーム、コピーなどの操作が可能である。

**Plot** タブは R コマンドで描画命令が出された時の結果がここに表示される。RStudio の利点の一つは、この Plot から図をファイルに Export することが可能であり、その際にファイルサイズやファイル形式を指定できるところにある。

**Packages** タブは読み込まれているパッケージ、(読み込まれていないが) 保管しているパッケージのリストが表示されている。新しくパッケージを導入するときも、ここの `install` ボタンから可能であり、保管しているパッケージのアップデートもボタンひとつで可能である。なお、パッケージについては後ほど言及する。

**Help** タブは R コマンドでヘルプを表示する命令 (`help` 関数) が実行された時の結果が表示される領域である。ヘルプを使うことで関数の引数、戻り値、使用例などを参照できる。

### 1.2.5 そのほかのタブ

そのほか、表示の有無もオプションになっているようないくつかのタブについて、簡単に解説しておく。

**Connections** タブは R を外部データベースなどに繋げるときに参照する。大規模データをローカルにすべて取り込むことなく、SQL で必要なテーブルだけ取り出すといった操作をする際には必要になってくるだろう。

**Git** タブは R、とくに R プロジェクト (後述) のバージョンを管理するときを利用する。Git とは複数のプログラマによって同時並行的にプログラムを作っていく時の管理システムである。時系列的な差分の記録を得意とするシステムなので、レポートの作成時などに応用すればラポノートの記録としても利用できる。

**Build** タブは R パッケージや Web サイトを構築するときを利用する。なおこの資料も RStudio を利用して作られており、資料を生成 (原稿から HTML や PDF にする) ときにはこのタブを利用している。

**Tutorial** タブはチュートリアルツアーを楽しむ時のタブである。

**Viewer** タブは RStudio で作られた HTML や PDF などを見るためのタブである。

**Presentation** タブは RStudio で作られたプレゼンテーションを見るためのタブである。

**Terminal** タブは Windows/Mac でいう Terminal, Linux でいう端末についてのタブであり、R に限らず、コマンドラインを通じて OS に命令するときを使う。

**Background Jobs** タブはその名の通りバックグラウンドで作業をさせるときに利用する。R は基本的にシングルコアで計算が実行されるが、このタブを使ってスクリプトファイルをバックグラウンドで実行することで並列的に作業が可能になる。

## 1.3 R のパッケージ

R は単体でも線型モデルなどの基本的な分析は可能であるが、より進んだ統計モデルを利用したい場合は専門の **パッケージ**を導入することになる。パッケージとは関数群のことであり、これも CRAN や Github などインターネットを介して提供されている。ちなみに提供されているパッケージは、CRAN で公開されているものだけで 344,607 件あり<sup>\*5</sup>、Github<sup>\*6</sup>で公開されているものなど、CRAN を介さないパッケージも少なくない。

---

<sup>\*5</sup> 2024 年 01 月 18 日調べ

<sup>\*6</sup> Git はバージョン管理システムであるが、これをインターネット上のサーバ (レポジトリ) で行うものを Github という。RStudio は Github と連携しており、プロジェクトを Github と紐づけることで簡単にバージョン管理ができる。しかもここで言及しているように、Github 上でパッケージを公開することもできるので、最近 CRAN の校閲を待たずに公開できる Github が好まれている側面もある。

パッケージを利用する際は、まずローカルにパッケージファイルをインストールしなければならない。その上で、R を起動するごとに (セッションごとに)、関数 `library` でパッケージを呼び出して利用する。インストールを毎回行う必要はないことに注意。

インストールは R のコマンドでも可能だが、RStudio の Packages ペインを使って導入するのが簡単だろう。以下に、一部の有名かつ有用なパッケージ名とその簡単な説明を挙げる。本講義の中で使うものもあるので、事前に準備しておくことが望ましい。

- *tidyverse* パッケージ (Wickham et al., 2019) ; R が飛躍的に使いやすくなったのは、この tidyverse パッケージ導入後のことである。開発者の Hadley Wickham は R 業界で神と崇められており、R 業界に与えたインパクトは大きい。このパッケージは「パッケージ群」「パッケージのパッケージ」であり、tidyverse とは tidy な (整然とした)verse(世界) というような意味合いである。このパッケージは統計分析モデルを提供するものではなく、その前のデータの**前処理**に関する便利な関数を提供する<sup>\*7</sup>。このパッケージをインストールすると、関連する依存パッケージが次々取り込まれるので、少々時間がかかる。
- *psych* パッケージ (Revelle, 2021) ; 名前の通り、心理学統計に関する統計モデルの多くが含まれている。特に特殊な相関係数や、因子分析モデルなどは非常に便利なので、インストールしておいて間違いない。
- *GPArotation* パッケージ (Bernaards & Jennrich, 2005) ; 因子分析における因子軸の回転に使うパッケージ。
- *styler* パッケージ ; スタイルを整えてくれるパッケージ。スクリプトの清書に便利。
- *lavaan* パッケージ (Rosseel, 2012) ; 潜在変数を含んだモデル (LAtent VArIable ANalysis) の分析、要するに構造方程式モデリング (Structural Equation Modeling;SEM, 共分散構造分析ともいう) を実行するパッケージ。
- *ctv* パッケージ (Zeileis, 2005); CRAN Task Views の略で、膨大に膨れ上がった CRAN から必要なパッケージを見つけ出すのは困難であることから、ある程度のジャンルごとに関連しそうなパッケージをまとめて導入してくれるのがこのパッケージ。例えば、このパッケージをインストールした後で、`install.views("Psychometrics")` とすると、心理統計関係の多くのパッケージを次々導入してくれる。
- *cmdstanr* パッケージ (Gabry et al., 2023) ; 複雑な統計モデルで利用される、確率的プログラミング言語 stan を R から使うことができるようになるパッケージ。導入にはこのパッケージの他にも stan やコンパイル環境の準備が必要なので、[公式の導入サイト](#)も参考にしてほしい。

## 1.4 RStudio のプロジェクト

実際に R を使っていく前に、最後の準備として RStudio におけるプロジェクトについて解説しておく。

みなさんも、PC をつかって文書を作ったり保管したりするときに、フォルダにまとめて入れておくことがあるだろう。フォルダは例えば「文書」>「心理学」>「心理学統計演習」のように階層的に整理することが一般的で、そうしておくことで必要なファイルをすぐに取り出すことができる。

逆に言えば、こうしたフォルダ管理をしておかなければファイルが PC のなかで散乱してしまい、必要な情報を

---

<sup>\*7</sup> 実は統計データの解析にかかる時間のほとんどが、解析に適切な形にデータを整形する「前処理」に費やされる。前処理、別名データハンドリングをいかに上手く、素早く、直感的にできるかは、その後の分析にも影響するほど重要な手順であるため、tidyverse パッケージの登場はありがたかった。これを使ったデータハンドリングだけの専門書 (松村他, 2021) が重宝されるほどである。



得るために逐一 PC の中身を検索しなければならないだろう。

R/RStudio をつかった分析実践の場合も同様で、一回のテーマについて複数のファイル (スクリプトファイル、データファイル、画像ファイル、レポートなど文書ファイル等々) があり、シーンに合わせて (例えば「授業」「卒論」など) フォルダで管理することになる。

さらに、PC 環境には作業フォルダ (Working Directory)<sup>\*8</sup> という概念がある。たとえば R/RStudio を起動・実行しているときに、R が「今どこで」実行されているか、どこを管理場所としているか、を表す概念である。例えばこの作業フォルダの中に `sample.csv` というファイルがあって、それをスクリプト上から読み込みたい、というコマンドを実行するのであれば、そのままファイル名を書けば良い。しかし別の場所にそのファイルが保存されているのなら、作業フォルダから見た相対的な位置を含めて指示してやるか (相対パス)、あるいは PC 環境全体からみた絶対的な位置を含めて (絶対パス) 指示してやる必要がある。相対・絶対パスの違いは、「ここから二つ目の角を右」のように指示するか、住所で指示するかの違いであると考えれば良い。

ともあれ、この作業フォルダがどこに設定されているかは、実行するときに常に気にしていなければならない。ちなみにこの作業フォルダは、RStudio のファイルペイン・Files タブでひらいているところとは限らないことに注意してほしい。GUI 上でエクスプローラ/Finder で開いたからといって、作業フォルダが自動的に切り替わるようにはなっていない。

そこで RStudio のプロジェクトである。RStudio には「プロジェクト」という概念があり、作業フォルダや環境の設定などをそこで管理することができる。新しくプロジェクトを始めるときは `File > New Project`、すでに一度プロジェクトを作っているときは `File > Open Project` としてプロジェクトファイル (拡張子が `.proj` のファイル) を開くようにする。そうすると、作業フォルダが当該フォルダに設定される。プロジェクトを Git に連携しておくとバージョン管理などもフォルダ単位で行える。

以後、本講義で外部ファイルを参照する場合、プロジェクトフォルダの中にそのファイルがあるものとして (パスを必要としない形で) 論じるので注意されたし。

## 1.5 課題

- R の最新版を CRAN からダウンロードし、自分の PC にインストールしてください。
- RStudio の Desktop 版を [Posit 社のサイト](#) からダウンロードし、自分の PC にインストールしてください。
- RStudio を起動し、ペインレイアウトをデフォルトではない状態に並べ直してみてください。ソースペインを 3 列にするのも良いでしょう。
- コンソールペインに書かれている文字を全て消去してみてください。
- ファイルペインにある Files タブをつかって、色々なフォルダを開けてみたり、不要なファイルを削除したり、ファイル名を変更したりしてみてください。
- ファイルペインにある Files タブを開き、More のところから `Go To Working Directory` を選択・実行してください。何か起こったでしょうか。

---

<sup>\*8</sup> ここでは、フォルダとディレクトリは同じ意味であると思ってもらって良い。一般に、CUI ではディレクトリ、GUI ではフォルダという用語が好まれる。語幹 `direct` にあるように、ファイルやアクセス先など具体的な指し示す先を強調しているのがディレクトリであり、それにファイル群などまとまった容れもの、という意味を付加したのがフォルダである。フォルダの方が言葉としてわかりやすいし。



- この授業のために、新しいプロジェクトを作成してください。プロジェクトは新しいフォルダでも、既存のフォルダでも構いません。
- プロジェクトが開いた状態のとき、RStudio のウィンドウ・タブのどこかに「プロジェクト名」が表示されているはずです。確認してください。
- またファイルペインの Files タブから、色々なファイル操作をした上で、改めて Go To Working Directory をしてください。プロジェクトフォルダの中に戻ってこれたら成功です。
- 新しい R スクリプトファイルを開き、空白のままで結構ですからファイル名をつけて保存してください。
- RStudio を終了あるいは最小化させ、OS のエクスプローラ/Finder から、プロジェクトフォルダに移動してください。先ほど作ったファイルが保存されていることを確認してください。
- プロジェクトフォルダには、プロジェクト名 + .proj というファイルが存在するはずです。これを開いて、RStudio のプロジェクトを開いてください。
- RStudio の File > Close Project からプロジェクトを閉じてください。画面の細部でどこが変わったか、確認してください。
- RStudio を終了し、再び RStudio を起動してください。起動の方法はプロジェクトファイルからでも、アプリケーションの起動でも構いません。起動後に、プロジェクトを開いてください (あるいはプロジェクトが開かれていることを確認してください。)

## 1.6 References

- Bernaards, C. A. & Jennrich, R. I. (2005). Gradient Projection Algorithms and Software for Arbitrary Rotation Criteria in Factor Analysis. *Educational and Psychological Measurement*, 65, 676–696. <https://doi.org/10.1177/0013164404272507>
- Gabry, J., Češnovar, R., & Johnson, A. (2023). *cmdstanr: R Interface to 'CmdStan'*. <https://mc-stan.org/cmdstanr/>, <https://discourse.mc-stan.org>.
- Hadley, W. (2014). Tidy Data. *Journal of Statistical Software*, 59, 1–23. <https://doi.org/10.18637/jss.v059.i10>
- 松村 優哉・湯谷 啓明・紀ノ定 保礼・前田 和寛 (2021). 改訂 2 版 R ユーザのための RStudio[実践] 入門——tidyverse によるモダンな分析フローの世界—— 技術評論社
- Revelle, W. (2021). *psych: Procedures for Psychological, Psychometric, and Personality Research*. R package version 2.1.3. Northwestern University. Evanston, Illinois. from <https://CRAN.R-project.org/package=psych>
- Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software*, 48 (2), 1–36. <https://doi.org/10.18637/jss.v048.i02>
- Stevens, S. S. (1946). On the theory of scales of measurement. *Science*, 103 (2684), 677–680.
- 高橋 康介 (2018). 再現可能性のすゝめ 石田 基広 (編) Wonderful R 3 共立出版
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4 (43), 1686. <https://doi.org/10.21105/joss.01686>
- Zeileis, A. (2005). CRAN Task Views. *R News*, 5 (1), 39–40.
- 総務省 (2020). 統計表における機械判別可能なデータ作成に関する表記方法.



## 第 2 章

# R の基礎

ここから実際に R/RStudio を使った演習に入る。前回すでに言及したように、この講義のようなプロジェクトを準備し、RStudio はプロジェクトが開かれた状態であることを前提に話を進める。

### 2.1 R で計算

まずは R を使った計算である。R スクリプトファイルを開き、最初の行に次の 4 行を入力してみよう。各行を実行 (Run ボタン, あるいは ctrl+enter) し、コンソールの結果を確認しよう。

```
1 + 2
```

```
[1] 3
```

```
2 - 3
```

```
[1] -1
```

```
3 * 4
```

```
[1] 12
```

```
6 / 3
```

```
[1] 2
```

それぞれ加減乗除の計算結果が正しく出ていることを確認してほしい。なお、出力のところに [1] とあるのは、R がベクトルを演算の基本としているからで、回答ベクトルの第 1 要素を返していることを意味する。

四則演算の他に、次のような演算も可能である。

```
# 整数の割り算
```

```
8 %/% 3
```

```
[1] 2
```

```
# 余り
7 %% 3
```

```
[1] 1
```

```
# 冪乗
2^3
```

```
[1] 8
```

ここで、#から始まる行は**コメントアウト**されたものとして、実際にコンソールに送られても計算されないことに注意しよう。スクリプトが単純なものである場合はコメントをつける必要はないが、複雑な計算になったり、他者と共有するときは「今どのような演算をしているか」を逐一解説するようにすると便利である。

実践上のテクニックとして、複数行を一括でコメントアウトしたり、アンコメント (コメントアウトを解除する) したりすることがある。スクリプトを複数行選択した上で、Code メニューから **Comment/Uncomment Lines** を押すとコメント/アンコメントを切り替えられるので試してみよう。また、ショートカットキーも確認し、キーからコメント/アンコメントができるように慣れておくの良い (Ctrl+ ↑ +C/Cmd+ ↑ +C)。

One more tips. コメントではなく、大きな段落的な区切り (セクション区切り) が欲しいこともあるかもしれない。Code メニューの一番上に「Insert Section」があるのでこれを選んでみよう。ショートカットキーから入力しても良い (Ctrl+ ↑ +R/Cmd+ ↑ +R)。セクション名を入力するボックスに適当な命名をすると、スクリプトにセクションが挿入される。次に示すのがセクションの例である。

```
# 計算 -----
```

これはもちろん実行に影響を与えないが、ソースが長くなった場合はこのセクション単位で移動したり (スクリプトペインの左下)、アウトラインを確認したり (スクリプトペインの右上にある横三本線) できるので、活用して欲しい。

## 2.2 オブジェクト

R では変数、関数などあらゆるものを**オブジェクト**としてあつかう。オブジェクトには任意の名前をつけることができる (数字から始まる名前は不可)。オブジェクトを作り、そこにある値を**代入**する例は次の通りである。

```
a <- 1
b <- 2
A <- 3
a + b # 1 + 2 におなじ
```

```
[1] 3
```

```
A + b # 3 + 2 におなじ
```

```
[1] 5
```

ここでは数字をオブジェクトに保管し、オブジェクトを使って計算をしている。大文字と小文字が区別されているため、計算結果が異なることに注意。

代入に使った記号<-は「小なり」と「ハイフン」であるが、左矢印のイメージである。次のように、=や->を使うこともできる。

```
B = 5
7 -> A
```

ここで、二行目に 7 -> A を行った。先ほど A <- 3 としたが、その後に A には 7 を代入し直したので、値は上書きされる。

```
A + b # 7 + 2 におなじ
```

```
[1] 9
```

このように、オブジェクトに代入を重ねると、警告などなしに上書きされることに注意して欲しい。似たようなオブジェクト名を使い回していると、本来意図していたものと違う値・状態を保管していることになりかねないからである。

ちなみに、オブジェクトの中身を確認するためには、そのままオブジェクト名を入力すれば良い。より丁寧には、print 関数を使う。

```
a
```

```
[1] 1
```

```
print(A)
```

```
[1] 7
```

あるいは、RStudio の Environment タブをみると、現在 R が保持しているオブジェクトが確認でき、単一の値の場合は Value セクションにオブジェクト名と値を見ることができる。

注意点として、オブジェクト名として、次の名前は使うことができない。> break, else, for, if, in, next, function, repeat, return, while, TRUE, FALSE.

これらは R で特別な意味を持つ**予約語**と呼ぶ。特に TRUE と FALSE は真・偽を表すもので、大文字の T,F でも代用できるため、この一文字だけをオブジェクト名にするのは避けた方が良い。

## 2.3 関数

関数は一般に  $y = f(x)$  と表されるが、要するに  $x$  を与えると  $y$  に形が変わる作用のことを指す。プログラミング言語では一般に、 $x$  を**引数** (ひきすう,argument),  $y$  を**戻り値** (もどりち,value) という。以下、関数の使用例を挙げる。

```
sqrt(16)
```

[1] 4

```
help("sqrt")
```

最初の例は平方根 square root を取る関数 `sqrt` であり、引数として数字を与えるとその平方根が返される。第二の例は関数の説明を表示させる関数 `help` であり、これを実行するとヘルプペインに関数の説明が表示される。

## 2.4 変数の種類

先ほどの `help` 関数に与えた引数 `"sqrt"` は文字列である。文字列であることを明示するためにダブルクォーテーション (") で囲っている (シングルクォーテーションで囲っても良い)。このように、R が扱う変数は数字だけではない。変数の種類は数値型 (numeric)、文字型 (character)、論理値 (logical) の3種類がある。

```
obj1 <- 1.5
obj2 <- "Hello"
obj3 <- TRUE
```

数値型は整数 (integer)、実数 (double) を含む<sup>\*1</sup>、そのほか、複素数型 (complex)、欠損値を表す `NA`、非数値を表す `NaN` (Not a Number)、無限大を表す `Inf` などがある。

文字型はすでに説明した通りで、対になるクォーテーションが必要であることに注意してほしい。終わりを表すクォーテーションがなければ、R は続く数字や文字も含めた「語」として処理する。この場合、`enter` キーを押しても文字入力が閉じられていないため、コンソールには「+」の表示が出る (この記号は前の行から入力が続いており、プロンプト状態ではないことを表している)。

また、文字型は当然のことながら四則演算の対象にならない。ただし、論理型の `TRUE/FALSE` はそれぞれ 1,0 に対応しているため、計算結果が表示される。次のコードを実行してこのことを確認しよう。

```
obj1 + obj2
obj1 + obj3
```

## 2.5 オブジェクトの型

ここまでみてきたように、数値や文字など (まとめてリテラルという) にも種類があるが、これをストックしておくものは全てオブジェクトである。オブジェクトとは変数のこと、と理解しても良いが、関数もオブジェクトに含まれる。

---

<sup>\*1</sup> 実数は real number じゃないのか、という指摘もあろうかとおもう。ここでは電子計算機上の数値の分類である、倍精度浮動小数点数 (double-precision floating-point number) の意味である。倍精度とは単精度の倍を意味しており、単精度は 32 ビットを、倍精度は 64 ビットを単位として一つの数字を表す仕組みのことである。

### 2.5.1 ベクトル

R のオブジェクトは単一の値しか持たないものではない。むしろ、複数の要素をセットで持つことができるのが特徴である。次に示すのは、ベクトルオブジェクトの例である。

```
vec1 <- c(2, 4, 5)
vec2 <- 1:3
vec3 <- 7:5
vec4 <- seq(from = 1, to = 7, by = 2)
vec5 <- c(vec2, vec3)
```

それぞれのオブジェクトの中身を確認しよう。最初の `c()` は結合 `combine` 関数である。また、コロン (`:`) は連続する数値を与える。`seq` 関数は複数の引数を取るが、初期値、終了値、その間隔を指定した連続的なベクトルを生成する関数である。

ベクトルの計算は要素ごとに行われる。次のコードを実行し、どのように振る舞うか確認しよう。

```
vec1 + vec2
```

```
[1] 3 6 8
```

```
vec3 * 2
```

```
[1] 14 12 10
```

```
vec1 + vec5
```

```
[1] 3 6 8 9 10 10
```

最後の計算でエラーが出なかったことに注目しよう。たとえば `vec1 + vec4` はエラーになるが、ここでは計算結果が示されている (=エラーにはなっていない)。数学的には、長さの違うベクトルは計算が定義されていないのだが、`vec1` の長さは 3、`vec5` の長さは 6 であった。**R はベクトルを再利用する**ので、長いベクトルが短いベクトルの定数倍になるときは反復して利用される。すなわち、ここでは

$$(2, 4, 5, 2, 4, 5) + (1, 2, 3, 7, 6, 5) = (3, 6, 8, 9, 10, 10)$$

の計算がなされた。この R の仕様については、意図せぬ挙動にならぬよう注意しよう。

ベクトルの要素にアクセスするときは大括弧 (`[ ]`) を利用する。特に第二・第三行目のコードの使い方を確認しておこう。大括弧の中は、要素番号でも良いし、真/偽の判断でも良いのである。この真偽判断による指定の方法は、条件節 (`if` 文) をつかって要素を指定できるため、有用である。

```
vec1[2]
```

```
[1] 4
```

```
vec2[c(1, 3)]
```

```
[1] 1 3
```

```
vec2[c(TRUE, FALSE, TRUE)]
```

```
[1] 1 3
```

ここまで、ベクトルの要素は数値で説明してきたが、文字列などもベクトルとして利用できる。

```
words1 <- c("Hello!", "Mr.", "Monkey", "Magic", "Orchestra")
words1[3]
```

```
[1] "Monkey"
```

```
words2 <- LETTERS[1:10]
words2[8]
```

```
[1] "H"
```

ここで `LETTERS` はアルファベット 26 文字が含まれている予約語ベクトルである。

ベクトルを引数に取る関数も多い。たとえば記述統計量である、平均、分散、標準偏差、合計などは、次のようにして計算する。

```
dat <- c(12,18,23,35,22)
mean(dat) # 平均
```

```
[1] 22
```

```
var(dat) # 分散
```

```
[1] 71.5
```

```
sd(dat) # 標準偏差
```

```
[1] 8.455767
```

```
sum(dat) # 合計
```

```
[1] 110
```

他にも最大値 `max` や最小値 `min`、中央値 `median` などの関数が利用可能である。

## 2.5.2 行列

数学では線形代数でベクトルを扱うが、同時にベクトルが複数並んだ二次元の行列も扱うだろう。R でも行列のように配置したオブジェクトを利用できる。

次のコードで作られる行列  $A, B$  がどのようなものか確認しよう。

```
A <- matrix(1:6, ncol = 2)
B <- matrix(1:6, ncol = 2, byrow = T)
```



行列を作る関数 `matrix` は、引数として要素、列数 (`ncol`)、行数 (`nrow`)、要素配列を行ごとにするかどうかの指定 (`byrow`) をとる。ここでは要素を 1:6 としており、1 から 6 までの連続する整数をあたえている。`ncol` で 2 列であることを明示しているので、`nrow` で行数を指定してやる必要はない。`byrow` の有無でどのように数字が変わっているかは表示させれば一目瞭然であろう。

与える要素が行数 × 列数に一致しておらず、ベクトルの再利用も不可能な場合はエラーが返ってくる。

また、ベクトルの要素指定のように、行列も大括弧を使って要素を指定することができる。行、列の順に指定し、行だけ、列だけの指定も可能である。

```
A[2, 2]
```

```
[1] 5
```

```
A[1, ]
```

```
[1] 1 4
```

```
A[, 2]
```

```
[1] 4 5 6
```

### 2.5.3 リスト型

行列はサイズの等しいベクトルのセットであるが、サイズの異なる要素をまとめて一つのオブジェクトとして保管しておきたいときはリスト型をつかう。

```
Obj1 <- list(1:4, matrix(1:6, ncol = 2), 3)
```

このオブジェクトの第一要素 (`[[1]]`) はベクトル、第二要素は行列、第三要素は要素 1 つのベクトル (スカラー) である。オブジェクトの要素の要素 (ex. 第二要素の行列の 2 行 3 列目の要素) にどのようにアクセスすれば良いか、考えてみよう。

このリストは要素へのアクセスの際に `[[1]]` など数字が必要だが、要素に名前をつけることで利便性が増す。

```
Obj2 <- list(  
  vec1 = 1:5,  
  mat1 = matrix(1:10, nrow = 5),  
  char1 = "YMO"  
)
```

この名前付きリストの要素にアクセスするときは、`$`記号を用いることができる。

```
Obj2$vec1
```

```
[1] 1 2 3 4 5
```

これを踏まえて、名前付きリストの要素の要素にアクセスするにはどうすれば良いか、考えてみよう。

リスト型はこのように、要素のサイズ・長さを問わないため、いろいろなものを保管しておくことができる。統計関数の結果はリスト型で得られることが多く、そのような場合、リストの要素も長くなりがちである。リストがどのような構造を持っているかを見るために、`str` 関数が利用できる。

```
str(Obj2)
```

```
List of 3
```

```
$ vec1 : int [1:5] 1 2 3 4 5
$ mat1 : int [1:5, 1:2] 1 2 3 4 5 6 7 8 9 10
$ char1: chr "YMO"
```

`str` 関数の返す結果と同じものが、RStudio の Environment タブからオブジェクトを見ることでも得られる。また、リストの要素としてリストを持つ、すなわち階層的になることもある。そのような場合、必要としている要素にどのようにアクセスすれば良いか、確認しておこう。

```
Obj3 <- list(Obj1, Second = Obj2)
str(Obj3)
```

```
List of 2
```

```
$      :List of 3
..$ : int [1:4] 1 2 3 4
..$ : int [1:3, 1:2] 1 2 3 4 5 6
..$ : num 3
$ Second:List of 3
..$ vec1 : int [1:5] 1 2 3 4 5
..$ mat1 : int [1:5, 1:2] 1 2 3 4 5 6 7 8 9 10
..$ char1: chr "YMO"
```

#### 2.5.4 データフレーム型

リスト型は要素のサイズを問わないことはすでに述べた。しかしデータ解析を行うときは得てして、2次元スプレッドシートのような形式である。すなわち一行に1オブザベーション、各列は変数を表すといった具合である。このように矩形かつ、列に変数名を持たせることができる特殊なリスト型を**データフレーム型**という。以下はそのようなオブジェクトの例である。

```
df <- data.frame(
  name = c("Ishino", "Pierre", "Marin"),
  origin = c("Shizuoka", "Shizuoka", "Hokkaido"),
  height = c(170, 180, 160),
  salary = c(1000, 20, 800)
)
# 内容を表示させる
df
```

```

      name   origin height salary
1 Ishino Shizuoka   170   1000
2 Pierre Shizuoka   180     20
3 Marin  Hokkaido   160    800

```

```
# 構造を確認する
```

```
str(df)
```

```

'data.frame':   3 obs. of  4 variables:
 $ name   : chr  "Ishino" "Pierre" "Marin"
 $ origin: chr  "Shizuoka" "Shizuoka" "Hokkaido"
 $ height: num   170 180 160
 $ salary: num  1000 20 800

```

ところで、心理統計の初歩として Stevens の尺度水準 (Stevens, 1946) について学んだことと思う。そこでは数値が、その値に許される演算のレベルをもとに、名義、順序、間隔、比率尺度水準という 4 つの段階に分類される。間隔・比率尺度水準の数値は数学的な計算を施しても良いが、順序尺度水準や名義尺度水準の数字はそのような計算が許されない (ex.2 番目に好きな人と 3 番目に好きな人が一緒になっても、1 番好きな人に敵わない。)

R には、こうした尺度水準に対応した数値型がある。間隔・比率尺度水準は計算可能なので `numeric` 型でよいが、名義尺度水準は `factor` 型 (要因型, 因子型とも呼ばれる), 順序尺度水準は `ordered.factor` 型と呼ばれるものである。

`factor` 型の変数の例を挙げる。すでに文字型として入っているものを `factor` 型として扱うよう変換するためには、`as.factor` 関数が利用できる。

```

df$origin <- as.factor(df$origin)
df$origin

```

```

[1] Shizuoka Shizuoka Hokkaido
Levels: Hokkaido Shizuoka

```

要素を表示させて見ると明らかなように、値としては `Shizuoka,Shizuoka,Hokkaido` の 3 つあるが、レベル (水準) は `Shizuoka,Hokkaido` の 2 つである。このように `factor` 型にしておく、カテゴリとして使えて便利である。

次に示すのは順序付き `factor` 型変数の例である。

```

# 順序付き要因型の例
ratings <- factor(c("低い", "高い", "中程度", "高い", "低い"),
                  levels = c("低い", "中程度", "高い"),
                  ordered = TRUE)
# ratings の内容と型を確認
print(ratings)

```

```

[1] 低い   高い   中程度 高い   低い
Levels: 低い < 中程度 < 高い

```

集計の際などは factor 型と違わないため、使用例は少ないかもしれない。しかし R は統計モデルを適用する時に、尺度水準に対応した振る舞いをするものがあるので、データの尺度水準を丁寧に設定しておくのも良いだろう。

データフレームの要素へのアクセスは、基本的に変数名を介してのものになるだろう。たとえば先ほどのオブジェクト `df` の数値変数に統計処理をしたい場合は、次のようにすると良い。

```
mean(df$height)
```

```
[1] 170
```

```
sum(df$salary)
```

```
[1] 1820
```

また、データフレームオブジェクトを一括で要約する関数もある。

```
summary(df)
```

name	origin	height	salary
Length:3	Hokkaido:1	Min. :160	Min. : 20.0
Class :character	Shizuoka:2	1st Qu.:165	1st Qu.: 410.0
Mode :character		Median :170	Median : 800.0
		Mean :170	Mean : 606.7
		3rd Qu.:175	3rd Qu.: 900.0
		Max. :180	Max. :1000.0

## 2.6 外部ファイルの読み込み

解析の実際には、データセットを手入力することではなく、データベースから取り出してくるか、別ファイルから読み込むことが一般的であろう。

統計パッケージの多くは独自のファイル形式を持っており、R にはそれぞれに対応した読み込み関数も用意されているが、ここでは最もプレーンな形でのデータである CSV 形式からの読み込み例を示す。

提供されたサンプルデータ、`Baseball.csv` を読み込むことを考える。なおこのデータは UTF-8 形式で保存されている<sup>\*2</sup>。これを読み込むには、R がデフォルトで持っている関数 `read.csv` が使える。

```
dat <- read.csv("Baseball.csv")
head(dat)
```

	Year	Name	team	salary	bloodType	height	weight	UniformNum	position
1	2011 年度	永川 勝浩	Carp	12000	O 型	188	97	20	投手

<sup>\*2</sup> UTF-8 というのは文字コードの一種で、0 と 1 からなる機械のデータを人間語に翻訳するためのコードであり、世界的にもっとも一般的な文字コードである。しかし WindowsOS はいまだにデフォルトで Shift-JIS というローカルな文字コードにしているため、このファイルを一度 Windows 機の Excel などと開くと文字化けし、以下の手順が正常に作用しなくなることがよくある。本講義で使う場合は、ダウンロード後に Excel などと開くことなく、直接 R から読み込むようにされたし。

2	2011 年度	前田 健太	Carp	12000	A 型	182	73	18	投手
3	2011 年度	栗原 健太	Carp	12000	O 型	183	95	5	内野手
4	2011 年度	東出 輝裕	Carp	10000	A 型	171	73	2	内野手
5	2011 年度	シュルツ	Carp	9000	不明	201	100	70	投手
6	2011 年度	大竹 寛	Carp	8000	B 型	183	90	17	投手

	Games	AtBats	Hit	HR	Win	Lose	Save	Hold
1	19	NA	NA	NA	1	2	0	0
2	31	NA	NA	NA	10	12	0	0
3	144	536	157	17	NA	NA	NA	NA
4	137	543	151	0	NA	NA	NA	NA
5	19	NA	NA	NA	0	0	0	9
6	6	NA	NA	NA	1	1	0	0

```
str(dat)
```

```
'data.frame': 7944 obs. of 17 variables:
 $ Year      : chr  "2011 年度" "2011 年度" "2011 年度" "2011 年度" ...
 $ Name      : chr  "永川 勝浩" "前田 健太" "栗原 健太" "東出 輝裕" ...
 $ team      : chr  "Carp" "Carp" "Carp" "Carp" ...
 $ salary    : int  12000 12000 12000 10000 9000 8000 8000 7500 7000 6600 ...
 $ bloodType : chr  "O 型" "A 型" "O 型" "A 型" ...
 $ height    : int  188 182 183 171 201 183 177 173 176 188 ...
 $ weight    : int  97 73 95 73 100 90 82 73 80 97 ...
 $ UniformNum: int  20 18 5 2 70 17 31 6 1 43 ...
 $ position  : chr  "投手" "投手" "内野手" "内野手" ...
 $ Games     : int  19 31 144 137 19 6 110 52 52 40 ...
 $ AtBats    : int  NA NA 536 543 NA NA 299 192 44 149 ...
 $ Hit       : int  NA NA 157 151 NA NA 60 41 11 35 ...
 $ HR        : int  NA NA 17 0 NA NA 4 2 0 1 ...
 $ Win       : int  1 10 NA NA 0 1 NA NA NA NA ...
 $ Lose      : int  2 12 NA NA 0 1 NA NA NA NA ...
 $ Save      : int  0 0 NA NA 0 0 NA NA NA NA ...
 $ Hold      : int  0 0 NA NA 9 0 NA NA NA NA ...
```

ここで `head` 関数はデータフレームなどオブジェクトの冒頭部分 (デフォルトでは 6 行分) を表示させるものである。また, `str` 関数の結果から明らかなように, 読み込んだファイルが自動的にデータフレーム型になっている。

ちなみに, サンプルデータにおいて欠損値に該当する箇所には NA の文字が入っていた。 `read.csv` 関数では, 欠損値はデフォルトで文字列 "NA" としている。しかし, 実際は別の文字 (ex. ピリオド) や, 特定の値 (ex. 9999) の場合もあるだろう。その際は, オプション `na.strings` で「欠損値として扱う値」を指示すれば良い。

## 2.7 おまけ；スクリプトの清書

さて、ここまでスクリプトを書いてきたことで、そこそこ長いスクリプトファイルができたことと思う。スクリプトの記述については、もちろん「動けばいい」という考え方もあるが、美しくかけていたほうがなお良いだろう。「美しい」をどのように定義するかは異論あるだろうが、一般に「コード規約」と呼ばれる清書方法がある。ここでは細部まで言及しないが、RStudioのCodeメニューからReformat Codeを実行してみよう。スクリプトファイルが綺麗に整ったように見えないだろうか？

美しいコードはデバッグにも役立つ。時折Reformatすることを心がけよう。

## 2.8 課題

- Rを起動し、新しいスクリプトファイルを作成してください。そのファイル内で、2つの整数を宣言し、足し算を行い、結果をコンソールに表示してください。
- スクリプトに次の計算を書き、実行してください。

$-\frac{5}{6} + \frac{1}{3}$   
 $-9.6 \div 4$   
 $-2.3 + \frac{1}{2}$   
 $-3 \times (2.2 + \frac{4}{5})$   
 $-(-2)^4$   
 $-2\sqrt{2} \times \sqrt{3}$   
 $-2\log_e 25$

- Rのスクリプトファイル内で、ベクトルを作成してください。ベクトルには1から10までの整数を格納してください。その後、ベクトルの要素の合計と平均を計算してください。ベクトルを合計する関数はsum、平均はmeanです。
- 次の表をリスト型オブジェクトTb1にしてください。

Name	Pop	Area	Density
Tokyo	1,403	2,194	6,397
Beijing	2,170	16,410	1,323
Seoul	949	605	15,688

- 先ほど作ったTb1オブジェクトの、東京(Tokyo)の面積(Area)の値を表示させてください(リスト要素へのアクセス)
- Tb1オブジェクトの人口(Pop)変数の平均を計算してください。
- Tb1オブジェクトをデータフレーム型オブジェクトdf2に変換してください。新たに作り直しても良いですし、as.data.frame関数を使っても良い。

- R のスクリプトを使用して、Baseball2022.csv ファイルを読み込み、データフレーム `dat` に格納してください。ただし、このファイルの欠損値は 999 という数値になっています。
- 読み込んだ `dat` の冒頭の 10 行を表示してみてください。
- 読み込んだ `dat` に `summary` 関数を適用してください。
- このデータセットの変数 `team` は名義尺度水準です。Factor 型にしてください。他にも Factor 型にすべき変数が 2 つありますので、それらも同様に型を変換してください。
- このデータセットの変数の中で、数値データに対して平均、分散、標準偏差、最大値、最小値、中央値をそれぞれ算出してください。
- 課題を記述したスクリプトファイルに対して、Reformat など整形してください。

## 2.9 References





## 第3章

# Rによるデータハンドリング

心理学を始め、データを扱うサイエンスでは、データ収集の計画、実行と、データに基づいた解析結果、それを踏まえてのコミュニケーションとの間に、「データをわかりやすい形に加工し、可視化し、分析する」という手順がある。このデータの加工を**データハンドリング**という。統計といえば「分析」に注目されがちだが、実際にはデータハンドリングと可視化のステップが最も時間を必要とし、重要なプロセスである。

### 3.1 tidyverse の導入

本講義では tidyverse を使ったデータハンドリングを扱う。tidyverse は、データに対する統一的な設計方針を表す概念でもあり、具体的にはそれを実装したパッケージ名でもある。まずは tidyverse パッケージをインストール (ダウンロード) し、次のコードで R に読み込んでおく。

```
library(tidyverse)

-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.3      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.1
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.3      v tidyr      1.3.0
v purrr      1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

Attaching core tidyverse packages, と表示され、複数のパッケージ名にチェックマークが入っていたものが表示されただろう。tidyverse パッケージはこれらの下位パッケージを含むパッケージ群である。これに含まれる dplyr, tidyr パッケージはデータの整形に、readr はファイルの読み込みに、forcats は Factor 型変数の操作に、stringr は文字型変数の操作に、lubridate は日付型変数の操作に、tibble はデータフレーム型オブジェクトの操作に、purrr はデータに適用する関数に、ggplot2 は可視化に特化したパッケージである。

続いて Conflicts についての言及がある。tidyverse パッケージに限らず、パッケージを読み込むと表示されることのあるこの警告は、「関数名の衝突」を意味している。ここまで、R を起動するだけで、sqrt, mean などの関数が利用できた。これは R の基本関数であるが、具体的には base パッケージに含まれた関数である。R は起動時に base などいくつかのパッケージを自動的に読み込んでいるのである。これに別途パッケージを読み込むとき、あとで読み込まれたパッケージが同名の関数を使っていることがある。このとき、関数名は後から読み込んだもので上書きされる。そのことについての警告が表示されているのである。具体的にみると、dplyr::filter() masks stats::filter() とあるのは、最初に読み込んでいた stats パッケージの filter 関数は、(tidyverse パッケージに含まれる)dplyr パッケージのもつ同名の関数で上書きされ、今後はこちらが優先的に利用されるよ、ということを示している。

このような同音異字関数は、関数を特定するときに混乱を招くかもしれない。あるパッケージの関数であることを明示したい場合は、この警告文にあるように、パッケージ名::関数名、という書き方にすると良い。

## 3.2 パイプ演算子

続いてパイプ演算子について解説する。パイプ演算子は tidyverse パッケージに含まれていた magrittr パッケージで導入されたもので、これによってデータハンドリングの利便性が一気に向上した。そこで R も ver 4.2 からこの演算子を導入し、特設パッケージのインストールを必要としなくとも使えるようになった。この R 本体のパイプ演算子のことを、tidyverse のそれと区別して、ナイーブパイプと呼ぶこともある。

ともあれこのパイプ演算子がいかに優れたものであるかを解説しよう。次のスクリプトは、あるデータセットの標準偏差を計算するものである\*1。数式で表現すると次の通り。ここで  $\bar{x}$  はデータベクトル  $x$  の算術平均。

$$v = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

```
dat <- c(10, 13, 15, 12, 14) # データ
M <- mean(dat) # 平均
dev <- dat - M # 平均偏差
pow <- dev^2 # 平均偏差の2乗
variance <- mean(pow) # 平均偏差の2乗の平均が分散
standardDev <- sqrt(variance) # 分散の正の平方根が標準偏差
```

ここでは、標準偏差オブジェクト standardDev を作るまでに平均オブジェクト M、平均偏差ベクトル dev、その2乗したもの pow、分散 variance と4つものオブジェクトを作って答えに到達している。また、作られるオブジェクトが左側にあり、その右側にどのような演算をしているかが記述されているため、頭の中では「オブジェクトを作る、次の計算で」と読んでいったことだろう。

パイプ演算子はこの思考の流れをそのまま具現化する。パイプ演算子は %>% と書き、左側の演算結果をパイプ演算子の右側に来る関数の第一引数として右側に渡す役目をする。これを踏まえて上のスクリプトを書き直してみよう。ちなみにパイプ演算子はショートカット Ctrl(Cmd)+Shift+M で入力できる。

\*1 もちろん sd(dat) の一行で済む話だが、ここでは説明のために各ステップを書き下している。もっとも、sd 関数で計算されるのは  $n-1$  で割った不偏分散の平方根であり、標本標準偏差とは異なるものである。

```
dat <- c(10, 13, 15, 12, 14)
standardDev <- dat %>%
  {. - mean(.)} %>%
  {.^2} %>%
  mean() %>%
  sqrt()
```

ここでピリオド (.) は、前の関数から引き継いだもの (プレイスホルダー) であり、二行目は `{dat - mean(dat)}`、すなわち平均偏差の計算を意味している。それを次のパイプで二乗し、平均し、平方根を取っている。平均や平方根を取るときにプレイスホルダーが明示されていないのは、引き受けた引数がどこに入るかが明らかなので省略しているからである。

この例に見るように、パイプ演算子を使うと、データ → 平均偏差 → 2乗 → 平均 → 平方根、という計算の流れと、スクリプトの流れが一致しているため、理解しやすくなったのではないだろうか。

また、ここでの計算は、次のように書くこともできる。

```
standardDev <- sqrt(mean((dat - mean(dat))^2))
```

この書き方は、関数の中に関数がある入れ子状態になっており、 $y = h(g(f(x)))$  のような形式である。これも対応するカッコの内側から読み解いていく必要があり、思考の流れと逆転しているため理解が難しい。パイプ演算子を使うと、`x %>% f() %>% g() %>% h() -> y` のように記述できるため、苦勞せずに読むことができる。

以下はこのパイプ演算子を使った記述で進めていくので、この表記法 (およびショートカット) に慣れていこう。

### 3.3 課題 1

- `sqrt`, `mean` 関数が `base` パッケージに含まれることをヘルプで確認してみよう。どこを見れば良いだろうか? `filter`, `lag` 関数はどうだろうか?
- `tidyverse` パッケージを読み込んだことで、`filter` 関数は `dplyr` パッケージのものが優先されることになった。 `dplyr` パッケージの `filter` 関数をヘルプで見よう。
- 上書きされる前の `stats` パッケージの `filter` 関数に関するヘルプを見よう。
- 先ほどのデータを使って、平均値絶対偏差 (MeanAD) および中央絶対偏差 (MAD) をパイプ演算子を使って算出してみよう。なお平均値絶対偏差、中央値絶対偏差は次のように定義される。また絶対値を計算する R 関数は `abs` である。

$$MeanAD = \frac{1}{n} \sum_{i=1}^n |x_i - \bar{x}|$$

$$MAD = median(|x_1 - median(x)|, \dots, |x_n - median(x)|)$$

## 3.4 列選択と行選択

ここからは tidyverse を使ったより具体的なデータハンドリングについて言及する。まずは特定の列および行だけを抜き出すことを考える。データの一部にのみ処理を加えたい場合に重宝する。

### 3.4.1 列選択

列選択は `select` 関数である。これは tidyverse パッケージ内の dplyr パッケージに含まれている。select 関数は MASS パッケージなど、他のパッケージに同名の関数が含まれることが多いので注意が必要である。

例示のために、R がデフォルトで持つサンプルデータ、iris を用いる。なお、iris データは 150 行あるので、以下ではデータセットの冒頭を表示する head 関数を用いているが、演習の際には head を用いなくても良い。

```
# iris データの確認
```

```
iris %>% head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

```
# 一部の変数を抜き出す
```

```
iris %>% select(Sepal.Length, Species) %>% head()
```

	Sepal.Length	Species
1	5.1	setosa
2	4.9	setosa
3	4.7	setosa
4	4.6	setosa
5	5.0	setosa
6	5.4	setosa

逆に、一部の変数を除外したい場合はマイナスをつける。

```
iris %>% select(-Species) %>% head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
3	4.7	3.2	1.3	0.2

4	4.6	3.1	1.5	0.2
5	5.0	3.6	1.4	0.2
6	5.4	3.9	1.7	0.4

# 複数変数の除外

```
iris %>% select(-c(Petal.Length, Petal.Width)) %>% head()
```

	Sepal.Length	Sepal.Width	Species
1	5.1	3.5	setosa
2	4.9	3.0	setosa
3	4.7	3.2	setosa
4	4.6	3.1	setosa
5	5.0	3.6	setosa
6	5.4	3.9	setosa

これだけでも便利だが、`select` 関数は適用時に抜き出す条件を指定してやればよく、そのために便利な以下のよう関数がある。

- `starts_with()`
- `ends_with()`
- `contains()`
- `matches()`

使用例を以下に挙げる。

# `starts_with` で特定の文字から始まる変数を抜き出す

```
iris %>% select(starts_with("Petal")) %>% head()
```

	Petal.Length	Petal.Width
1	1.4	0.2
2	1.4	0.2
3	1.3	0.2
4	1.5	0.2
5	1.4	0.2
6	1.7	0.4

# `ends_with` で特定の文字で終わる変数を抜き出す

```
iris %>% select(ends_with("Length")) %>% head()
```

	Sepal.Length	Petal.Length
1	5.1	1.4
2	4.9	1.4
3	4.7	1.3
4	4.6	1.5
5	5.0	1.4

6            5.4            1.7

```
# contains で部分一致する変数を取り出す
iris %>% select(contains("etal")) %>% head()
```

	Petal.Length	Petal.Width
1	1.4	0.2
2	1.4	0.2
3	1.3	0.2
4	1.5	0.2
5	1.4	0.2
6	1.7	0.4

```
# matches で正規表現による選択をする
iris %>% select(matches(".t.")) %>% head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
3	4.7	3.2	1.3	0.2
4	4.6	3.1	1.5	0.2
5	5.0	3.6	1.4	0.2
6	5.4	3.9	1.7	0.4

ここで触れた**正規表現**とは、文字列を特定するためのパターンを指定する表記ルールであり、R言語に限らずプログラミング言語一般で用いられるものである。書誌検索などでも用いられることがあり、任意の文字列や先頭・末尾の語などを記号 (メタ文字) を使って表現するものである。詳しくは正規表現で検索すると良い (たとえば[こちらのサイト](#)などがわかりやすい。)

### 3.4.2 行選択

一般にデータフレームは列に変数が並んでいるので、`select` 関数による列選択とは変数選択とも言える。これに対し、行方向にはオブザベーションが並んでいるので、行選択とはオブザベーション (ケース、個体) の選択である。行選択には `dplyr` の `filter` 関数を使う。

```
# Sepal.Length 変数が 6 以上のケースを抜き出す
iris %>% filter(Sepal.Length > 6) %>% head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	7.0	3.2	4.7	1.4	versicolor
2	6.4	3.2	4.5	1.5	versicolor
3	6.9	3.1	4.9	1.5	versicolor
4	6.5	2.8	4.6	1.5	versicolor
5	6.3	3.3	4.7	1.6	versicolor

```
6          6.6          2.9          4.6          1.3 versicolor
```

```
# 特定の種別だけ抜き出す
```

```
iris %>% filter(Species == "versicolor") %>% head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	7.0	3.2	4.7	1.4	versicolor
2	6.4	3.2	4.5	1.5	versicolor
3	6.9	3.1	4.9	1.5	versicolor
4	5.5	2.3	4.0	1.3	versicolor
5	6.5	2.8	4.6	1.5	versicolor
6	5.7	2.8	4.5	1.3	versicolor

```
# 複数指定の例
```

```
iris %>% filter(Species != "versicolor", Sepal.Length > 6) %>% head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	6.3	3.3	6.0	2.5	virginica
2	7.1	3.0	5.9	2.1	virginica
3	6.3	2.9	5.6	1.8	virginica
4	6.5	3.0	5.8	2.2	virginica
5	7.6	3.0	6.6	2.1	virginica
6	7.3	2.9	6.3	1.8	virginica

ここで==とあるのは一致しているかどうかの判別をするための演算子である。=ひとつだと「オブジェクトへの代入」と同じになるので、判別条件の時には重ねて表記する。同様に、!=とあるのは not equal、つまり不一致のとき真になる演算子である。

## 3.5 変数を作る・再割り当てする

既存の変数から別の変数を作る、あるいは値の再割り当ては、データハンドリング時に最もよく行う操作のひとつである。たとえば連続変数がある値を境に「高群・低群」というカテゴリカルな変数に作り変えたり、単位を変換するために線形変換したりすることがあるだろう。このように、変数を操作するときに「既存の変数を加工して特徴量を作り出す」というときの操作は、基本的に dplyr の mutate 関数を用いる。次の例をみてみよう。

```
mutate(iris, Twice = Sepal.Length * 2) %>% head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Twice
1	5.1	3.5	1.4	0.2	setosa	10.2
2	4.9	3.0	1.4	0.2	setosa	9.8
3	4.7	3.2	1.3	0.2	setosa	9.4
4	4.6	3.1	1.5	0.2	setosa	9.2
5	5.0	3.6	1.4	0.2	setosa	10.0

```
6          5.4          3.9          1.7          0.4 setosa 10.8
```

新しく `Twice` 変数ができたのが確認できるだろう。この関数はパイプ演算子の中で使うことができる (というかその方が主な使い方である)。次の例は、`Sepal.Length` 変数を高群と低群の2群に分けるものである。

```
iris %>%
  select(Sepal.Length) %>%
  mutate(Sepal.HL = ifelse(Sepal.Length > mean(Sepal.Length), 1, 2)) %>%
  mutate(Sepal.HL = factor(Sepal.HL, label = c("High", "Low"))) %>%
  head()
```

	Sepal.Length	Sepal.HL
1	5.1	Low
2	4.9	Low
3	4.7	Low
4	4.6	Low
5	5.0	Low
6	5.4	Low

ここでもちいた `ifelse` 関数は、`if(条件判断, 真のときの処理, 偽のときの処理)` という形でもちいる条件分岐関数であり、ここでは平均より大きければ1、そうでなければ2を返すようになっている。`mutate` 関数でこの結果を `Sepal.HL` 変数に代入 (生成) し、次の `mutate` 関数では今作った `Sepal.HL` 変数を Factor 型に変換して、その結果をまた `Sepal.HL` 変数に代入 (上書き) している。このように、変数の生成先を生成元と同じにしておくとお書きされるため、たとえば変数の型変換 (文字型から数値型へ、数値型から Factor 型へ、など) にも用いることができる。

## 3.6 課題2

- `Baseball.csv` を読み込んで、データフレーム `df` に代入しよう。
- `df` には複数の変数が含まれている。変数名の一覧は `names` 関数で行う。`df` オブジェクトに含まれる変数名を確認しよう。
- `df` には多くの変数があるが、必要なのは年度 (Year)、選手名 (Name)、所属球団 (team)、身長 (height)、体重 (weight)、年俸 (salary)、守備位置 (position) だけである。変数選択をして、これらの変数だけからなる `df2` オブジェクトを作ろう。
- `df2` に含まれるデータは数年分のデータが含まれる。2020 年度のデータだけ分析したいので、選別してみよう。
- 同じく 2020 年度の阪神タイガースに関するデータだけを選別してみよう。
- 同じく 2020 年度の阪神タイガース以外のデータセットはどのようにして選別できるだろうか。
- 選手の身体的特徴を表す BMI 変数をつくろう。なお、BMI は体重 (kg) を身長 (m) の二乗で除したものである。変数 `height` の単位が cm であることに注意しよう。
- 投手と野手を区別する、新しい変数 `position2` を作ってみよう。これは Factor 型にしよう。なお、野手は投手でないもの、すなわち内野手、外野手、捕手のいずれかである。



- 日本プロ野球界は大きく分けてセリーグ (Central League) とパリーグ (Pacific League) にわかれている。セリーグに所属する球団は Giants, Carp, Tigers, Swallows, Dragons, DeNA であり、パ・リーグはそれ以外である。df2 を加工して、所属するリーグの変数 `League` をつくってみよう。この変数も Factor 型にしておこう。
- 変数 `Year` は語尾に「年度」という文字が入っているため文字列型になっており、実際に使うときは不便である。「年度」という文字を除外した、数値型変数に変換しよう。

## 3.7 ロング型とワイド型

ここまでみてきたデータは行列の 2 次元に、ケース × 変数の形で格納されていた。この形式は、人間が見て管理するときにわかりやすい形式をしているが、計算機にとっては必ずしもそうではない。たとえば「神エクセル」と揶揄されることがあるように、稀に表計算ソフトを方眼紙ソフトあるいは原稿用紙ソフトと勘違いしたかのような使い方がなされる場合がある。人間にとってはわかりやすい (見て把握しやすい) かもしれないが、計算機にとって構造が把握できないため、データ解析に不向きである。巷には、こうした分析しにくい電子データがまだまだたくさん存在する。

これをうけて 2020 年 12 月、総務省により機械判読可能なデータの表記方法の統一ルールが策定された (総務省, 2020)。それには次のようなチェック項目が含まれている。

- ファイル形式は Excel か CSV となっているか
- 1 セル 1 データとなっているか
- 数値データは数値属性とし、文字列を含まないこと
- セルの結合をしていないか
- スペースや改行等で体裁を整えていないか
- 項目名を省略していないか
- 数式を使用している場合は、数値データに修正しているか
- オブジェクトを使用していないか
- データの単位を記載しているか
- 機種依存文字を使用していないか
- データが分断されていないか
- 1 シートに複数の表が掲載されていないか

データの入力の基本は、1 行に 1 ケースの情報が入っている、過不足のない 1 つのデータセットを作ることといえるだろう。

同様に、計算機にとって分析しやすいデータの形について、Hadley (2014) が提唱したのが**整然データ (Tidy Data)**という考え方である。整然データとは、次の 4 つの特徴を持ったデータ形式のことを指す。

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

この形式のデータであれば、計算機が変数と値の対応構造を把握しやすく、分析しやすいデータになる。データハンドリングの目的は、混乱している雑多なデータを、利用しやすい整然データの形に整えることであると言っても過言ではない。さて、ここでよく考えてみると、変数名も一つの変数だと考えることに気づく。一般に、行列型のデータは次のような書式になっている。

Table3.1: ロング型データ

	午前	午後	夕方	深夜
東京	晴	晴	雨	雨
大阪	晴	曇	晴	晴
福岡	晴	曇	曇	雨

ここで、たとえば大阪の夕方の天気を見ようとする「晴れ」であることは明らかだが、この時の視線の動きは大阪行の、夕方列、という参照の仕方である。言い方を変えると、大阪・夕方の「晴れ」を参照するときに、行と列の両方のラベルを参照する必要がある。

ここで同じデータを次のように並べ替えてみよう。

Table3.2: ロング型データ

地域	時間帯	天候
東京	午前	晴
東京	午後	晴
東京	夕方	雨
東京	深夜	雨
大阪	午前	晴
大阪	午後	曇
大阪	夕方	晴
大阪	深夜	晴
福岡	午前	晴
福岡	午後	曇
福岡	夕方	曇
福岡	深夜	雨

このデータが表す情報は同じだが、大阪・夕方の条件を絞り込むことは行選択だけでよく、計算機にとって使いやすい。この形式をロング型データ、あるいは「縦持ち」データという。これに対して前者の形式をワイド型データ、あるいは「横持ち」データという。

ロング型データにする利点のひとつは、欠損値の扱いである。ワイド型データで欠損値が含まれる場合、その行あるいは列全体を削除するのは無駄が多く、かと言って行・列両方を特定するのは技術的にも面倒である。これに対しロング型データの場合は、当該行を絞り込んで削除するだけで良い。

tidyverse には (正確には tidyr には), このようなロング型データ, ワイド型データの変換関数が用意されている。実例とともに見てみよう。まずはワイド型データをロング型に変換する `pivot_longer` である。

```
iris %>% pivot_longer(-Species)
```

```
# A tibble: 600 x 3
  Species name      value
  <fct>    <chr>    <dbl>
1 setosa Sepal.Length  5.1
2 setosa Sepal.Width   3.5
3 setosa Petal.Length  1.4
4 setosa Petal.Width   0.2
5 setosa Sepal.Length  4.9
6 setosa Sepal.Width   3
7 setosa Petal.Length  1.4
8 setosa Petal.Width   0.2
9 setosa Sepal.Length  4.7
10 setosa Sepal.Width  3.2
# i 590 more rows
```

ここでは元の iris データについて, Species セルを軸として, それ以外の変数名と値を name,value に割り当てて縦持ちにしている。

逆に, ロング型のデータをワイド型に持ち替えるには, `pivot_wider` を使う。実例は以下の通りである。

```
iris %>%
  select(-Species) %>%
  rowid_to_column("ID") %>%
  pivot_longer(-ID) %>%
  pivot_wider(id_cols = ID, names_from = name, values_from = value)
```

```
# A tibble: 150 x 5
  ID Sepal.Length Sepal.Width Petal.Length Petal.Width
  <int>         <dbl>         <dbl>         <dbl>         <dbl>
1     1         5.1         3.5         1.4         0.2
2     2         4.9         3         1.4         0.2
3     3         4.7         3.2         1.3         0.2
4     4         4.6         3.1         1.5         0.2
5     5         5         3.6         1.4         0.2
6     6         5.4         3.9         1.7         0.4
7     7         4.6         3.4         1.4         0.3
8     8         5         3.4         1.5         0.2
9     9         4.4         2.9         1.4         0.2
```

```
10      10      4.9      3.1      1.5      0.1
# i 140 more rows
```

今回は `Species` 変数を除外し、別途 ID 変数として行番号を変数に付与した。この行番号をキーに、変数名は `names` 列から、その値は `value` 列から持ってくることでロング型をワイド型に変えている<sup>\*2</sup>。

### 3.8 グループ化と要約統計量

データをロング型にすることで、変数やケースの絞り込みが容易になる。その上で、ある群ごとに要約した統計量を算出したい場合は、`group_by` 変数によるグループ化と、`summarise` あるいは `reframe` がある。実例を通して確認しよう。

```
iris %>% group_by(Species)

# A tibble: 150 x 5
# Groups:   Species [3]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
      <dbl>      <dbl>      <dbl>      <dbl> <fct>
1         5.1         3.5         1.4         0.2 setosa
2         4.9         3         1.4         0.2 setosa
3         4.7         3.2         1.3         0.2 setosa
4         4.6         3.1         1.5         0.2 setosa
5          5         3.6         1.4         0.2 setosa
6         5.4         3.9         1.7         0.4 setosa
7         4.6         3.4         1.4         0.3 setosa
8          5         3.4         1.5         0.2 setosa
9         4.4         2.9         1.4         0.2 setosa
10        4.9         3.1         1.5         0.1 setosa
# i 140 more rows
```

上のコードでは、一見したところ表示されたデータに違いがないように見えるが、出力時に `Species[3]` と表示されていることがわかる。ここで、`Species` 変数の 3 水準で群分けされていることが示されている。これを踏まえて、`summarise` してみよう。

```
iris %>%
  group_by(Species) %>%
  summarise(n = n(),
            Mean = mean(Sepal.Length),
            Max = max(Sepal.Length),
```

<sup>\*2</sup> `Species` 変数を除外したのは、これをキーにしたロング型をワイド型に変えることができない (`Species` は 3 水準しかない) からで、個体を識別する ID が別途必要だったからである。`Species` 情報が欠落することになったが、これはロング型データの `value` 列が `char` 型と `double` 型の両方を同時に持てないからである。この問題を回避するためには、`Factor` 型のデータを `as.numeric()` 関数で数値化することなどが考えられる。

```
IQR = IQR(Sepal.Length))
```

```
# A tibble: 3 x 5
  Species      n Mean   Max   IQR
  <fct>    <int> <dbl> <dbl> <dbl>
1 setosa     50  5.01   5.8 0.400
2 versicolor 50  5.94    7  0.7
3 virginica  50  6.59   7.9 0.675
```

ここではケース数 (n), 平均 (mean), 最大値 (max), 四分位範囲 (IQR)<sup>\*3</sup>を算出した。

また, ここでは Sepal.Length についてのみ算出したが, 他の数値型変数に対しても同様の計算がしたい場合は, across 関数を使うことができる。

```
iris %>%
  group_by(Species) %>%
  summarise(across(c(Sepal.Length, Sepal.Width, Petal.Length),
    ~mean(.x)))
```

```
# A tibble: 3 x 4
  Species    Sepal.Length Sepal.Width Petal.Length
  <fct>          <dbl>         <dbl>         <dbl>
1 setosa          5.01           3.43           1.46
2 versicolor      5.94           2.77           4.26
3 virginica       6.59           2.97           5.55
```

ここで, `~mean(.x)` の書き方について言及しておく。チルダ (tilde, ~) で始まるこの式を, R では特にラムダ関数とかラムダ式と呼ぶ。これはこの場で使う即席関数の作り方である。別の方法として, 正式に関数を作る関数 `function` を使って次のように書くこともできる。

```
iris %>%
  group_by(Species) %>%
  summarise(across(
    c(Sepal.Length, Sepal.Width, Petal.Length),
    function(x) {
      mean(x)
    }
  ))
```

```
# A tibble: 3 x 4
  Species    Sepal.Length Sepal.Width Petal.Length
  <fct>          <dbl>         <dbl>         <dbl>
```

<sup>\*3</sup> 四分位範囲 (Inter Quantaile Range) とは, データを値の順に 4 分割した時の上位 1/4 の値から, 上位 3/4 の値を引いた範囲である。

1	setosa	5.01	3.43	1.46
2	versicolor	5.94	2.77	4.26
3	virginica	6.59	2.97	5.55

ラムダ関数や自作関数の作り方については、後ほどあらためて触れるとして、ここでは複数の変数に関数をあてがう方法を確認して置いて欲しい。across 関数で変数を選ぶ際は、select 関数の時に紹介した starts\_with など利用できる。次に示す例は、複数の変数を選択し、かつ、複数の関数を適用する例である。複数の関数を適用するために、ラムダ関数をリストで与えることができる。

```
iris %>%
  group_by(Species) %>%
  summarise(across(starts_with("Sepal"),
    .fns = list(M = ~mean(.x),
                 Q1 = ~quantile(.x,0.25),
                 Q3 = ~quantile(.x,0.75))
  ))
```

```
# A tibble: 3 x 7
  Species    Sepal.Length_M Sepal.Length_Q1 Sepal.Length_Q3 Sepal.Width_M
  <fct>          <dbl>          <dbl>          <dbl>          <dbl>
1 setosa         5.01            4.8            5.2            3.43
2 versicolor     5.94            5.6            6.3            2.77
3 virginica      6.59            6.22           6.9            2.97
# i 2 more variables: Sepal.Width_Q1 <dbl>, Sepal.Width_Q3 <dbl>
```

### 3.9 データ整形課題

- 上で作った df2 オブジェクトを利用する。環境に df2 オブジェクトが残っていない場合は、もう一度上の課題に戻って作り直しておこう。
- 年度 (Year) でグルーピングし、年度ごとの登録選手数 (データの数)、平均年俸を見てみよう。
- 年度 (Year) とチーム (team) でグルーピングし、同じく年度ごとの登録選手数 (データの数)、平均年俸を見てみよう。
- つづいて、一行に1年度分、列に各チームと変数の組み合わせが入った、ワイド型データをつくりたい。pivot\_wider にして上のオブジェクトをワイド型にしてみよう。
- ワイド型になったデータを、Year 変数をキーにして pivot\_longer でロング型データに変えてみよう。

### 3.10 References

## 第 4 章

# R によるレポートの作成

### 4.1 Rmd/Quarto の使い方

#### 4.1.1 概略

今回は RStudio を使った文書作成法について解説する。皆さんは、これまで作成と言えば、基本的に Microsoft Word のような文書作成ソフトを使ってきたものと思う。また、統計解析といえば R(やその他のソフト)、図表の作成は Excel といったように、用途ごとに異なるアプリケーションを活用するのが一般的であろう。

このやり方は、統計解析の数値結果を表計算ソフトに、そこで作った図表を文書作成ソフトにコピー＆ペーストする、という転記作業が何度も発生する。ここで転記ミス・貼り付け間違いが生じると、当然ながら出来上がる文書は間違っただけのものになる。こうした転記ミスのことを「コピペ汚染」と呼ぶこともある。

問題はこの環境をまたぐ作業にあるわけで、計算・作図・文書が一つの環境で済めばこうした問題が起らない。これを解決するのが R markdown や Quarto という書式・ソフトウェアなのである。

Rmarkdown の `markdown` とは書式の一つである。マークアップ言語と呼ばれる書き方の一種で、なかでも R との連携に特化したのが Rmarkdown である。マークアップ言語とは、言語の中に専門の記号を埋め込み、その書式に対応した読み込みアプリで、表示の際に書式を整える方式のことを指す。有名なマークアップ言語としては、数式に特化した LaTeX や、インターネットウェブサイトで用いられている HTML などがある。

Rmarkdown は `markdown` の書式を踏襲しつつ、R での実行結果を文中に埋め込むコマンドを有している。R のコマンドで計算したり図表を作成しつつ、その結果を埋め込む場所をマークアップ言語で指定する。最終的に文書を閲覧する場合は、マークアップ言語を出力ファイルに変換する (コンパイルする、ニットするという) 必要があり、その時 R での計算が実行される。コンパイルのたびに計算されるので、同じコードでも乱数を使ったコードを書いていたたり、一部読み込みファイルを変更するだけで、出力される結果は変わる。しかしコピペ汚染のように、間違っただけの値・図表が含まれるものではないので、研究の再現性にも一役買うことになる。再現可能な文書の作成について、詳しくは @Takahashi201805 を参考にすると良い。

Quarto は Rmarkdown をさらに拡張させたもので、RStudio を提供している Posit 社が今もっとも注力しているソフトのひとつである。Rmarkdown は R と `markdown` の連携であったが、Quarto は R だけでなく、Python や Julia といった他の言語にも対応しているし、これら複数の計算言語の混在も許す。すなわち、一部は R で計算し、

その結果を Python で検算して Julia で描画する、といったことを一枚のファイルで書き込むことも可能である。

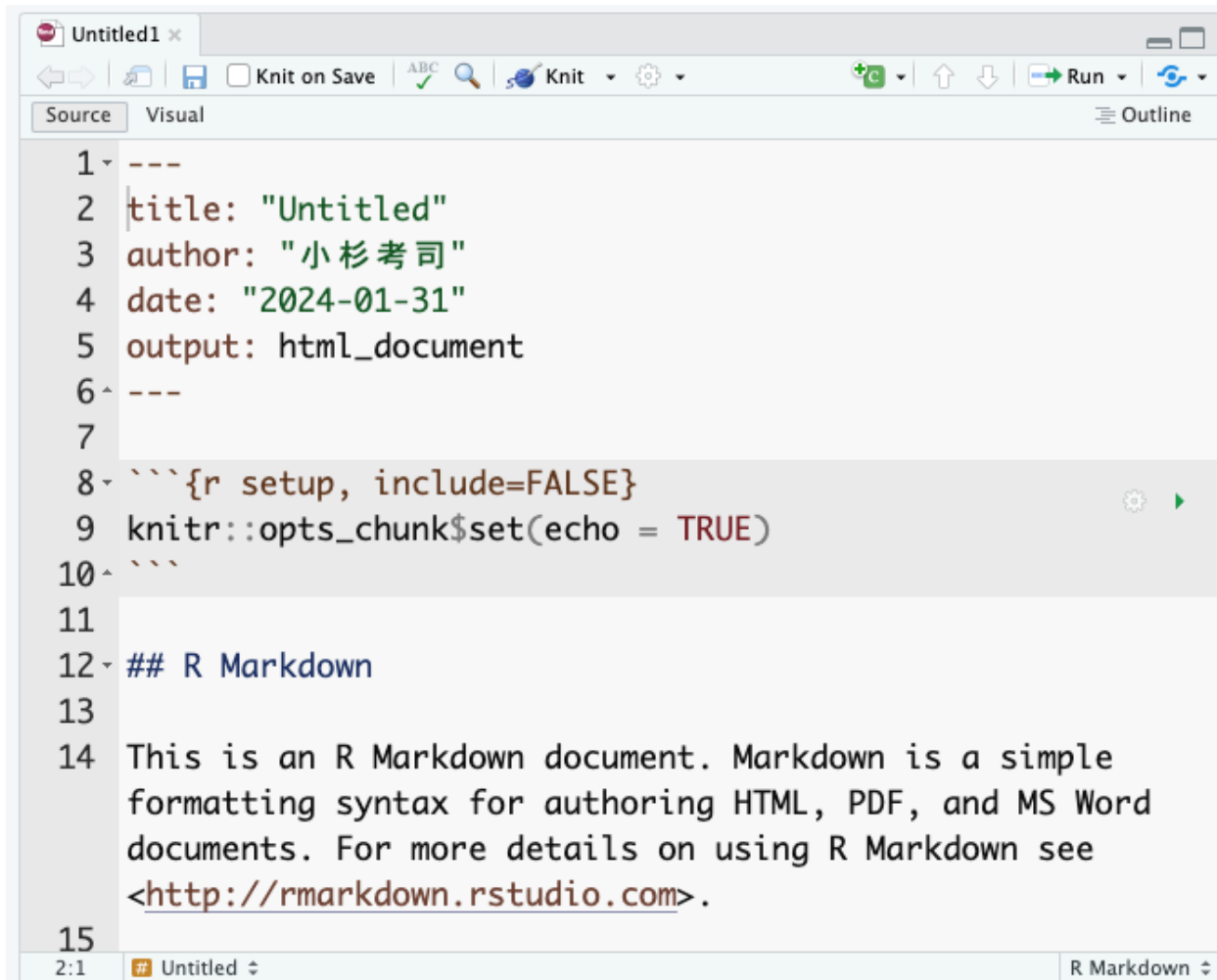
なおこの授業資料も Quarto で作成している。このように Quarto はプレゼンテーション資料やウェブサイトも作成できるし、出力形式もウェブサイトだけでなく PDF や ePUB(電子書籍の形式) にすることが可能である。なおこの授業の資料もウェブサイトと同時に [PDF 形式](#) と、[ePUB 形式](#) で出力されている。Quarto について専門の解説書はまだないが、インターネットに充実したドキュメントがあるので検索するといいだろう。まだ新しい技術なので、[公式](#)を第一に参照すると良い。

#### 4.1.2 ファイルの作成と knit

Rmarkdown は RStudio との相性がよく、RStudio の File > New File から **R Markdown** を選ぶと Rmarkdown ファイルがサンプルとともに作成される。作成時に文書のタイトル、著者名、作成日時や出力フォーマットが指定できるサブウィンドウが開き、作成するとサンプルコードが含まれた R markdown ファイルが表示されるだろう。

Quarto も同様に、RStudio の File > New File から **Quarto Document** を選ぶことで新しいファイル画面が開く。なお Rmarkdown ファイルの拡張子は `Rmd` とすることが一般的であり、Quarto は `Qmd` とすることが一般的である。もっとも、Quarto は RStudio 以外のエディタから利用することも考えられていて、例えば VS Code などの一般的なエディタで作成し、コマンドライン経由でコンパイルすることも可能である。





```
1 ---
2 title: "Untitled"
3 author: "小杉 考司"
4 date: "2024-01-31"
5 output: html_document
6 ---
7
8 ```{r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = TRUE)
10 ```
11
12 ## R Markdown
13
14 This is an R Markdown document. Markdown is a simple
15 formatting syntax for authoring HTML, PDF, and MS Word
16 documents. For more details on using R Markdown see
17 <http://rmarkdown.rstudio.com>.
18
```

Figure4.1: Rmarkdown ファイルのサンプル

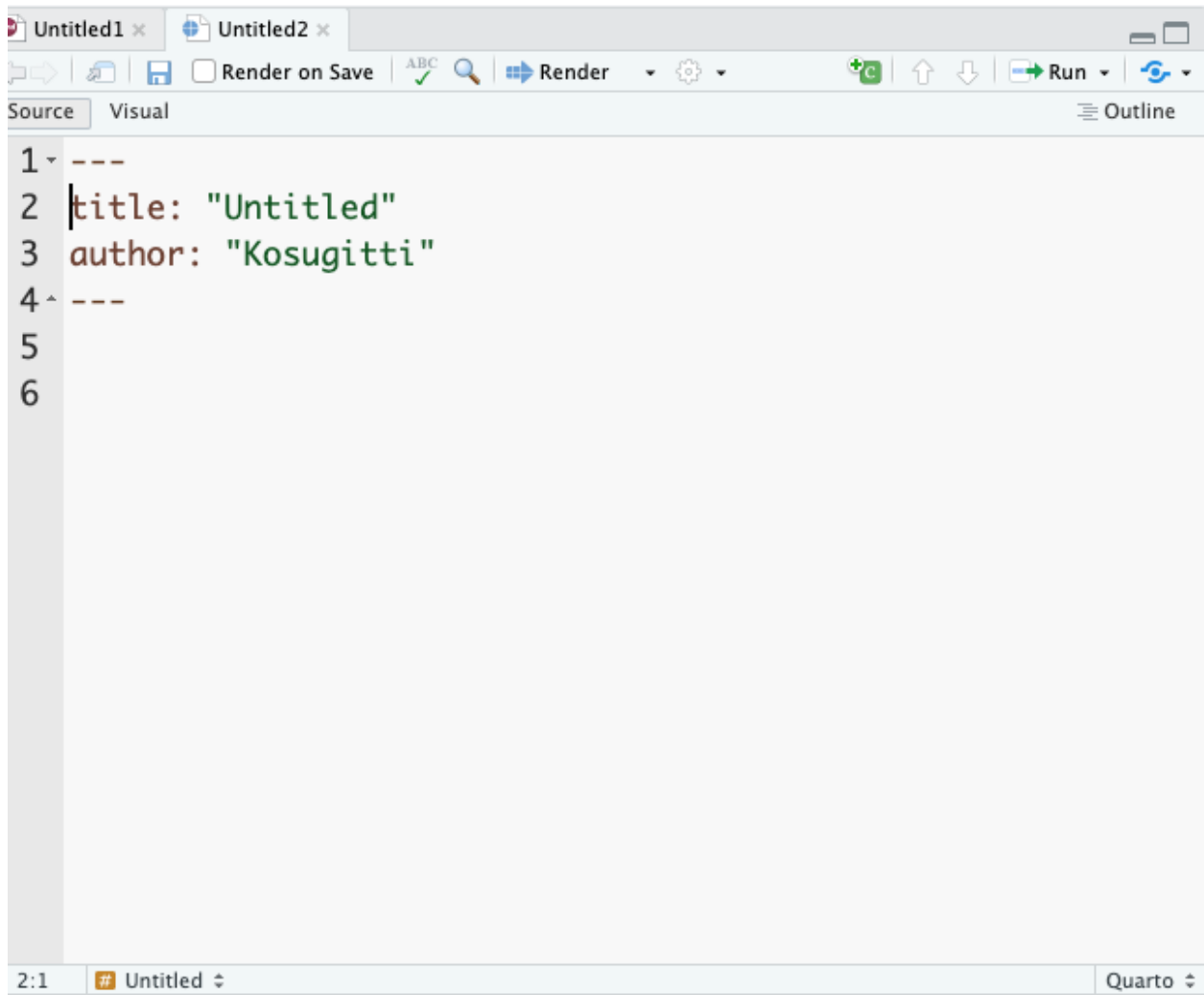


Figure4.2: Quarto ファイルのサンプル

Rmarkdown, Quarto とともに、ファイルの冒頭に 4 つのハイフンで囲まれた領域が見て取れるだろう。これは YAML ヘッダ (YAML は Yet Another Markup Language の略。ここはまだマークアップ領域じゃないよということ) と呼ばれる、文書全体に対する設定をする領域のことである。

この領域を一瞥すると、タイトルや著者名、出力形式などが記載されていることが見て取れる。YAML ヘッダはインデントに敏感で、また正しくない記述が含まれているとエラーになって出力ファイルが作られないことが少なくないため、ここを手動で書き換えるときは注意が必要である。とはいえ、ここを自在に書き換えることができるようになると、様々な応用が効くので興味があるものは調べて色々トライしてもらいたい。

さて、Rmd/Qmd のファイル上部に Knit あるいは Render と書かれたボタンがあるだろう。これをクリックすると、表示用ファイルへの変換が実行される。<sup>\*1</sup>Rmarkdown の場合は、すでにサンプルコードが含まれているので、数値および図表のはいった HTML ドキュメントが表示されるだろう。以下はこのサンプルコードを例に説明する

<sup>\*1</sup> もし新しく開いているファイルに名前がつけられていないのなら (Untitled のままになっているようであれば)、ファイル名の指定画面が開く。また環境によっては、初回実行時にコンパイルに必要な関連パッケージのダウンロードが求められることがある。

ため、Rmarkdown とそのコンパイル (knit)) を一度試してもらいたい。その上で、元の Rmd ファイルと出来上がったファイルとの対応関係を確認してみよう。

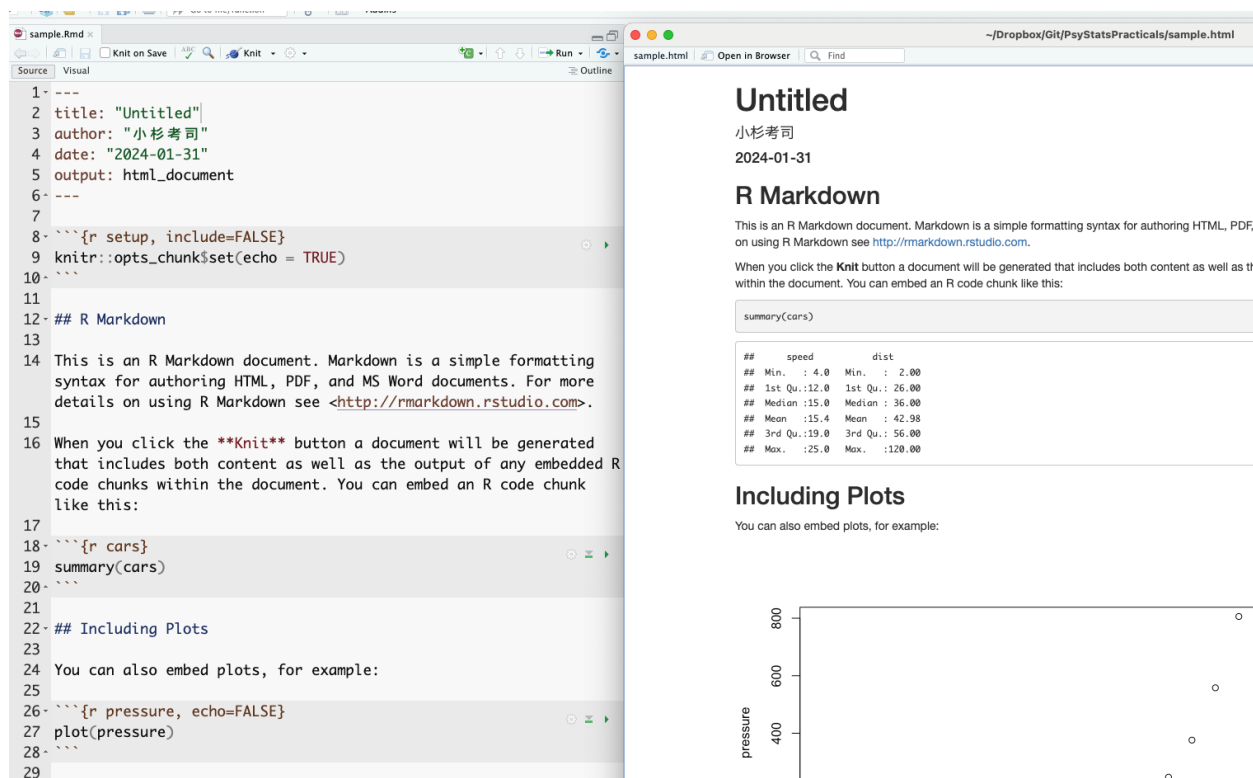


Figure4.3: Rmd ファイルと出力結果の対応

おおよそ、何がどのように変換されているかの対応が推察できるだろう。出力ファイルの冒頭には、YAML で設定したタイトル、著者名、日付などが表示されているし、#の印がついていた一行は見出しとして強調されている。

特に注目したいのが、元のファイルで3つのクォーテーションで囲まれた灰色の領域である。この領域のことを特に**チャンク**といい、ここに書かれた R スクリプトが変換時に実行され、結果として出力される。出力ファイルをみると、`summary(cars)` というチャンクで指定された命令文があり、その結果 (cars というデータセットの要約) が出力されているのが見て取れる。繰り返しになるが、ポイントは原稿ファイルには計算を指示するスクリプトが書かれているだけで、出力結果を書いていないことにある。原稿は指示だけなのである。こうすることで、コピー&ペーストのミスがなくなるし、同じ Rmd/Qmd 原稿とデータを持っていれば、ことなる PC 上でも同じ出力が得られる。環境を統合することで、ミスの防止と再現可能性に貢献しているのがわかるだろう。

今回は `cars` という R がデフォルトでもっているサンプルデータの例なので、どの環境でも同じ結果が出力されている。しかしもちろん、個別のデータファイルであっても、同じファイルで同じ読み込み方、同じ加工をしている場合、環境が違って追跡可能である。注意して欲しいのは、コンパイルするときは新しい環境から行われるという点がある。すなわち、**原稿ファイルにないオブジェクトの利用はできない**のである。これは再現性を担保するという意味では当然のことで、「事前に別途処理しておいたデータ」から分析を始められても、その事前処理が適切だったかどうかチェックできないからである。Rmd/Qmd ファイルと、CSV ファイルなどの素データが共有されていれば再現できる、という利点を活かすため、データハンドリングを含めた前処理も全てチャンクに書き込

み、新しい環境で最初からトレースできるようにする必要がある。不便に感じることもあるかもしれないが、科学的営みとして重要な手続きであることを理解してもらいたい。<sup>\*2</sup>

RStudi では、ビジュアルモードやアウトライン表示、チャンク挿入ボタンやチャンクごとの実行・設定など Rmd/Qmd ファイルの編集に便利な機能も複数用意されているので、高橋 (2018) などを参考にいろいろ試してみるといいだろう。

### 4.1.3 マークダウンの記法

以下では、マークダウン記法について基本的な利用法を解説する。

#### 4.1.3.1 見出しと強調

すでに見たように、マークダウンでは#記号で見出しを作ることができる。#の数が見出しレベルに対応し、#はトップレベル、本でいうところの「章,chapter」、HTML でいうところの H1 に相当する。#記号の後ろに半角スペースが必要なことに注意されたし。以下、##で「節,section」あるいは H2、###で小節 (subsection,H3)、####で小小節 (subsubsection,H4)...と続く。

心理学を始め、科学論文の書き方としての「パラグラフライティング」を既にみしっていることだろう。文章をセクション、サブセクション、パラグラフ、センテンスのように階層的に分割し、それぞれの区分が4つの下位区分を含むような文章構造である。心理学の場合は特に「問題、方法、結果、考察」の4セクションで一論文が構成されるのが基本である。こうしたアウトラインを意識した書き方は読み手にも優しく、マークダウンの記法ではそれが自然と実装できるようになっている。

これとは別に、一部を太字や斜体で強調したいこともあるだろう。そのような場合はアスタリスクを1つ、あるいは2つつけて**強調**したり**強調**したりできる。

#### 4.1.3.2 図表とリンク

文中に図表を挿入したいこともあるだろう。表の挿入は、マークダウン独自の記法があり、縦棒|やハイフン-を駆使して以下のように表記する。

```
| Header 1 | Header 2 | Header 3 |
| ----- | ----- | ----- |
| Row 1    | Data 1    | Data 2    |
| Row 2    | Data 3    | Data 4    |
```

R のコードの中には分析結果をマークダウン形式で出力してくれる関数もあるし、表計算ソフトなどでできた表があるなら、chatGPT など生成 AI を利用するとすぐに書式変換してくれるので、そういったツールを活用すると良い。

<sup>\*2</sup> もっとも、R のバージョンやパッケージのバージョンによっては同じ計算結果が出ない可能性がある。より本質的な計算過程に違いがあるかもしれないのである。そのため、R 本体やパッケージのバージョンごとパッキングして共有する工夫も考えられている。Docker と呼ばれるシステムは、解析環境ごと保全し共有するシステムの例である。

図の挿入は、マークダウンでは図のファイルへのリンクと考えると良い。次のように、大括弧で括った文字がキャプション、つづく小括弧で括ったものが図へのリンクとなる。実際に表示されるときは図が示される。

! [図のキャプション] (図へのリンク)

同様に、ウェブサイトへのリンクなども、[表示名] (リンク先) の書式で対応できる。

#### 4.1.3.3 リスト

並列的に箇条書きを示したい場合は、プラスあるいはマイナスでリストアップする。注意すべきは、リストの前後に改行を入れておくべきことである。

ここまで前の文

```
+ list item 1
+ list item 2
+ list item 3
  - sub item 1
  - sub item 2
```

ここから次の文

#### 4.1.3.4 チャンク

既に述べたように、チャンク (chunk) と呼ばれる領域は実行されるコードを記載するところである。チャンクはまず、バックスラッシュ 3 つ繋げることでコードブロックであることを示し、次に `r` と書くことで計算エンジンが R であることを明示する。ここに Julia や Python など他の計算エンジンを指定することも可能である。

可能であれば、チャンク名をつけておくと良い。次の例は、チャンク名として「chunksample」を与えたものである。チャンク名をつけておくと、RStudio では見出しジャンプをつかって移動することもできるので、編集時に便利である。

```
“{r chunksample,echo = FALSE} summary(cars) “
```

さらに、`echo = FALSE` のようにチャンクオプションを指定することができる。`echo=FALSE` は入力したスクリプトを表示せず、結果だけにするオプションである。そのほか「計算結果を含めない」「表示せずに計算は実行する」等様々な指定が可能である。

なお Quarto ではこのチャンクオプションを次のように書くこともできる。

```
“{r} #| echo: FALSE #| include: FALSE summary(cars) “
```

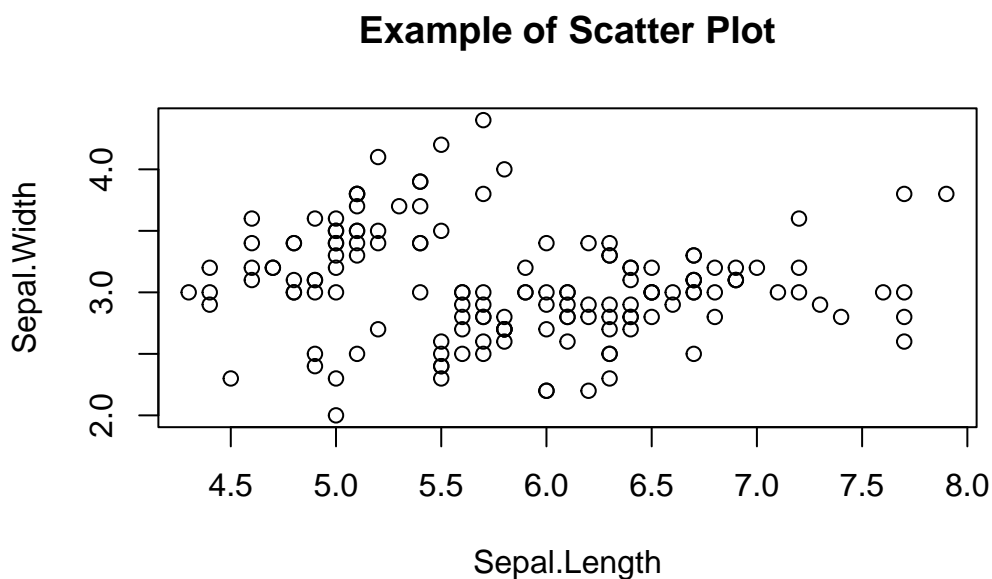
## 4.2 プロットによる基本的な描画

再現可能な文書という観点から、図表もスクリプトによる記述で表現することは重要である。

データはまず可視化するものと心がけよ。可視化は、数値の羅列あるいはまとめられた統計量では把握しきれない多くの情報を提供し、潜在的な関係性を直観的に見つけ出せる可能性がある。なので、取得したあらゆるデータはまず可視化するもの、と思っておいて間違いはない。大事なことなので二度言いました。可視化の重要性については心理学の知見にも触れている@Kieran2018 も参考にしてほしい。

さて、R には基本的な作図環境も整っており、plot という関数に引数として、x 軸、y 軸に相当する変数を与えるだけで、簡単に散布図を書いてくれる。

```
plot(iris$Sepal.Length, iris$Sepal.Width,  
     main = "Example of Scatter Plot",  
     xlab = "Sepal.Length",  
     ylab = "Sepal.Width"  
)
```



この関数のオプションとして、タイトルを与えたり、軸に名前を与えたりできる。またプロットされるピンの形、描画色、背景色など様々な操作が可能である。特段のパッケージを必要とせずとも、基本的な描画機能は備えていると言えるだろう。

### 4.3 ggplot による描画

ここでは、tidyverse に含まれる描画専用のパッケージである、ggplot2 パッケージを用いた描画を学ぶ。R の基本描画関数でもかなりのことができるのだが、この ggplot2 パッケージをもちいた図の方が美しく、直観的に操作できる。というのも ggplot の gg とは The Grammar of Graphics(描画の文法) のことであり、このことが示すようにロジカルに図版を制御できるからである。ggplot2 の形で記述された図版のスク립トは可読性が高く、視覚的にも美しいため、多くの文献で利用されている。

ggplot2 パッケージの提供する描画環境の特徴は、レイア (Layer) の概念である。図版は複数のレイアの積み重ねとして表現される。まず土台となるキャンバスがあり、そこにデータセット、幾何学的オブジェクト (点、線、

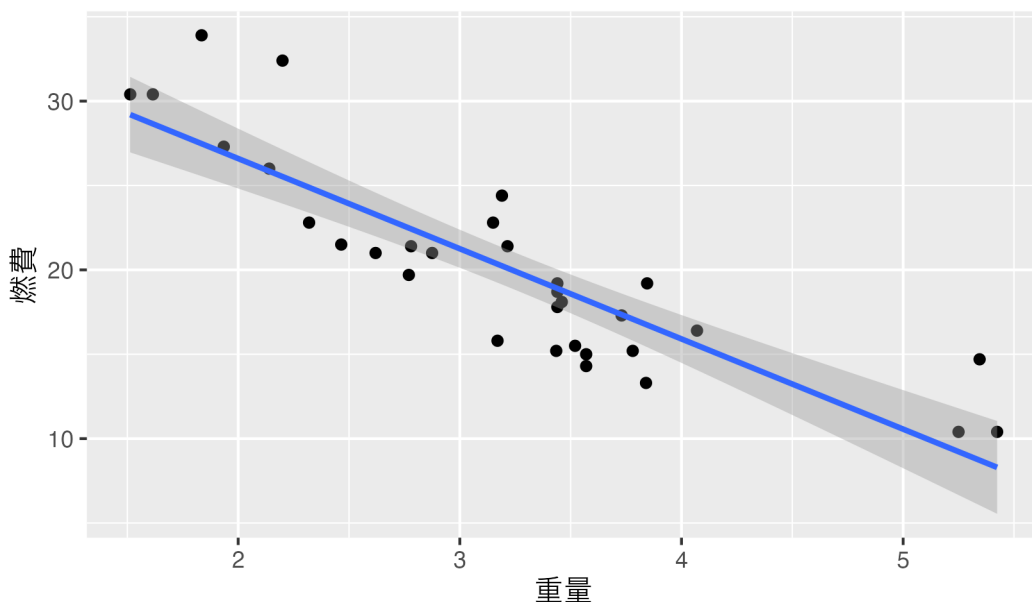
バーなど), エステティックマッピング (色, 形, サイズなど), 凡例やキャプションを重ねていく, という発想である。そして図版全体を通したテーマを手強することで, カラーパレットの統一などの仕上げをすれば, すぐにも論文投稿可能なレベルの図版を描くことができる。

以下に ggplot2 における描画のサンプルを示す。サンプルデータ `mtcars` を用いた。

```
library(ggplot2)

ggplot(data = mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_smooth(method = "lm", formula = "y ~ x") +
  labs(title = "車の重量と燃費の関係", x = "重量", y = "燃費")
```

車の重量と燃費の関係

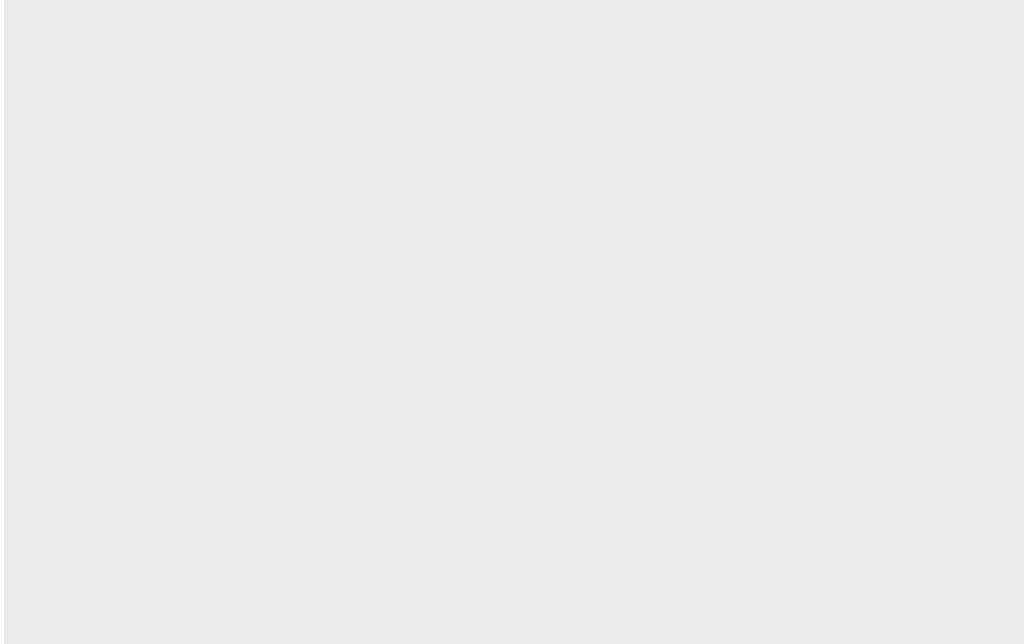


まずは出来上がる図版の美しさと, コードのイメージを把握してもらいたい。最初の `library(ggplot2)` はパッケージを読み込んでいるところである。今回は明示的に `ggplot2` を読み込んでいるが, `tidyverse` パッケージを読み込むと同時に読み込まれているので, R のスクリプトの冒頭に `library(tidyverse)` と書く癖をつけておけば必要ない。

続いて `ggplot` の関数が 4 行にわたって書いてあるが, それぞれが `+` の記号で繋がれていることがわかるだろう。これがレイアを重ねるという作業に相当する。まずは, 図を書くためのキャンバスを用意し, その上にいろいろ重ねていくのである。

次のコードは, キャンバスだけを描画した例である。

```
g <- ggplot()
print(g)
```



ここでは `g` というオブジェクトを `ggplot` 関数でつくり、それを表示させた。最初はこのようにプレーンなキャンバスだが、ここに次々と上書きしていくことになる。

## 4.4 幾何学的オブジェクト geom

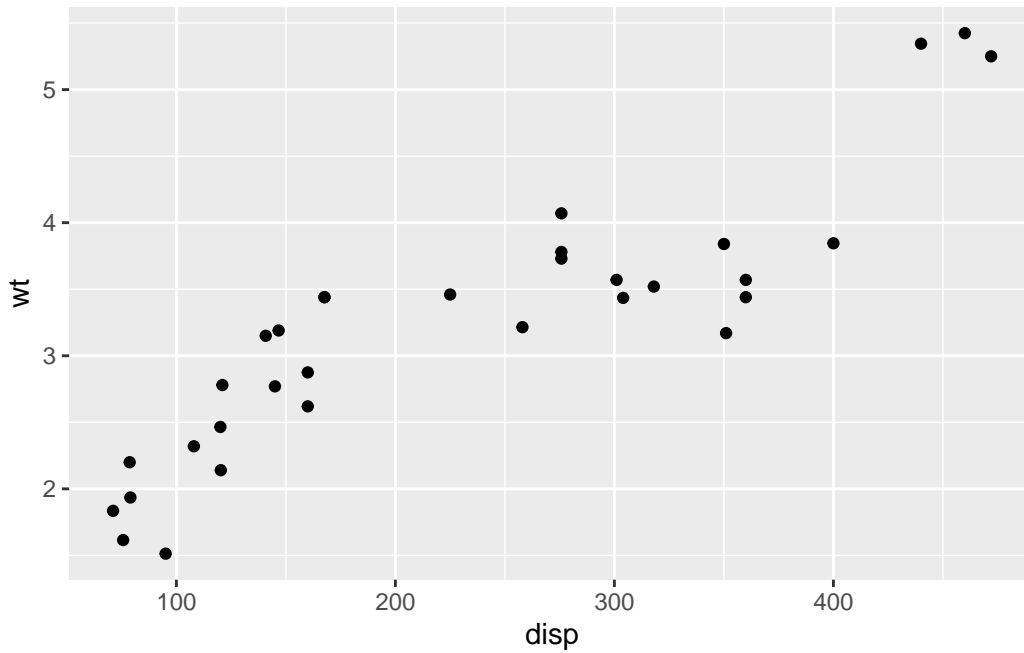
幾何学的オブジェクト (geometric object) とは、データの表現方法の指定であり、`ggplot` には様々なパターンが用意されている。以下に一例を挙げる。

- `geom_point()`: 散布図で使用され、データ点を個々の点としてプロットする。
- `geom_line()`: 折れ線グラフで使用され、データ点を線で結んでプロットする。時系列データなどによく使われる。
- `geom_bar()`: 棒グラフで使用され、カテゴリごとの量を棒で表示する。データの集計（カウントや合計など）に適している。
- `geom_histogram()`: ヒストグラムで使用され、連続データの分布を棒で表示する。データの分布を理解するのに役立つ。
- `geom_boxplot()`: 箱ひげ図で使用され、データの分布（中央値、四分位数、外れ値など）を要約して表示する。
- `geom_smooth()`: 平滑化曲線を追加し、データのトレンドやパターンを可視化する。線形回帰やローパスフィルタなどの方法が使われる。

これらの幾何学的オブジェクトに、データおよび軸との対応を指定するなどして描画する。次に挙げるのは `geom_point` による点描画、つまり散布図である。

```
ggplot() +  
  geom_point(data = mtcars, mapping = aes(x = disp, y = wt))
```

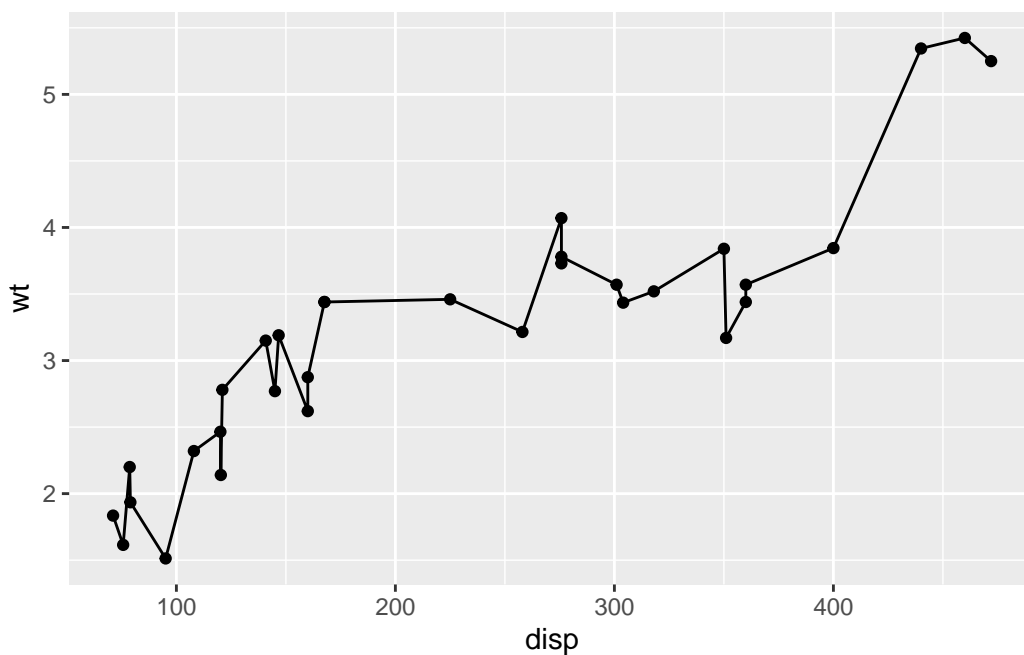




一行目でキャンバスを用意し，そこに `geom_point` で点を打つようにしている。このとき，データは `mtcars` であり，x 軸に変数 `disp` を，y 軸に変数 `wt` をマッピングしている。マッピング関数の `aes` は aesthetic mappings の意味で，データによって変わる値 (x 座標, y 座標, 色, サイズ, 透明度など) を指定することができる。

レイアは次々と重ねることができる。以下の例を見てみよう。

```
g <- ggplot()
g1 <- g + geom_point(data = mtcars, mapping = aes(x = disp, y = wt))
g2 <- g1 + geom_line(data = mtcars, mapping = aes(x = disp, y = wt))
print(g2)
```



重ねることを強調するために、g オブジェクトを次々作るようにしたが、もちろん1つのオブジェクトでまとめて書いてもいいし、g オブジェクトとして保管せずとも、最初の例のように直接出力することもできる。また、ここでは点描画オブジェクトに線描画オブジェクトを重ねているが、データやマッピングは全く同じである。異なるデータを一枚のキャンバスに書く場合は、このように幾何学オブジェクトごとの指定が可能であるが、図版は得てして一枚のキャンバスに一種類のデータになりがちである。そのような場合は、以下に示すようにキャンバスの段階から基本となるデータセットとマッピングを与えることが可能である。

```
ggplot(data = mtcars, mapping = aes(x = disp, y = wt)) +
  geom_point() +
  geom_line()
```

また、この用例の場合 ggplot 関数の第一引数はデータセットなので、パイプ演算子で渡すことができる。

```
mtcars %>%
  ggplot(mapping = aes(x = disp, y = wt)) +
  geom_point() +
  geom_line()
```

パイプ演算子を使うことで、素データをハンドリングし、必要な形に整えて可視化する、という流れがスクリプト上で読むように表現できるようになる。慣れてくると、データセットから可視化したい要素を特定し、最終的にどのように成形すれば ggplot に渡しやすくなるかを想像して加工していくようになる。そのためには到達目標となる図版のイメージを頭に描き、その図の x 軸、y 軸は何で、どのような幾何学オブジェクトが上に乗っているのか、といったように図版のリバースエンジニアリング、あるいは図版の作成手順の書き下しができる必要がある。たとえるなら、食べたい料理に必要な材料を集め、大まかな手順(下ごしらえからの調理)を組み立てられるかどうか、である。実際にレシピに書き起こす際は生成 AI の力を借りると良いが、その際も最終的な目標と、全体的な設計方針から指示し、微調整を追加していくように指示すると効率的である。

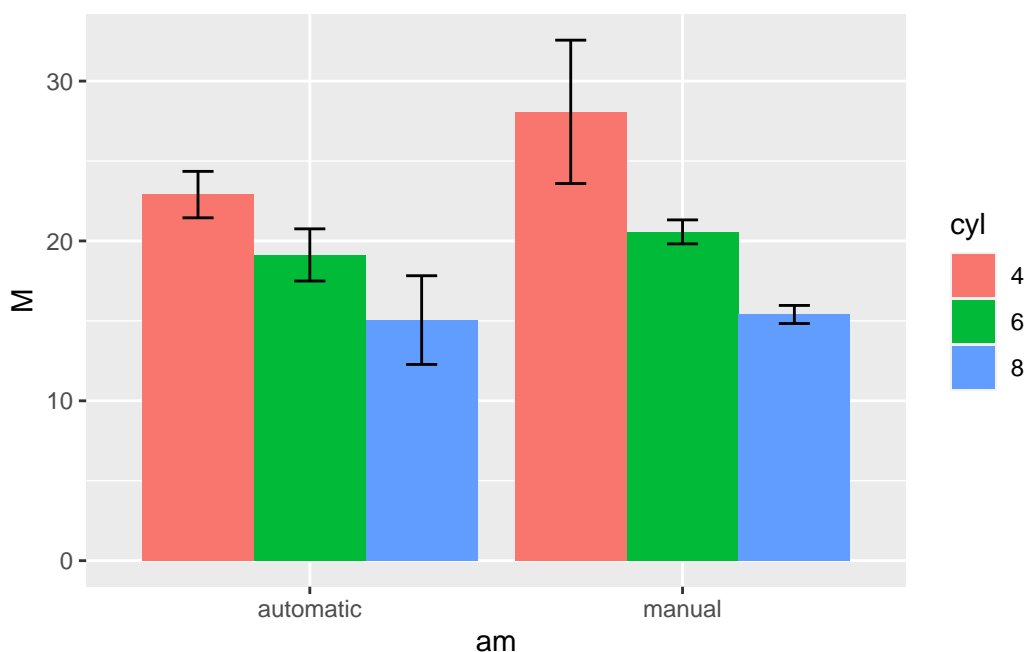
以下に、データハンドリングと描画の一例を示す。各ステップにコメントをつけたので、文章を読むように加工と描画の流れを確認し、出力結果と照らし合わせてみよう。

```
# mtcars データセットを使用
mtcars %>%
  # 変数選択
  select(mpg, cyl, wt, am) %>%
  mutate(
    # 変数 am, cyl を Factor 型に変換
    am = factor(am, labels = c("automatic", "manual")),
    cyl = factor(cyl)
  ) %>%
  # 水準ごとにグループ化
  group_by(am, cyl) %>%
  summarise(
    M = mean(mpg), # 各グループの平均燃費 (M) を計算
```

```

SD = sd(mpg), # 各グループの燃費の標準偏差 (SD) を計算
.groups = "drop" # summarise 後の自動的なグルーピングを解除
) %>%
# x 軸にトランスミッションの種類、y 軸に平均燃費、塗りつぶしの色は cyl
ggplot(aes(x = am, y = M, fill = cyl)) +
# 横並びの棒グラフ
geom_bar(stat = "identity", position = "dodge") +
# ±1SD のエラーバーを追加
geom_errorbar(
  # エラーバーのマッピング
  aes(ymin = M - SD, ymax = M + SD),
  # エラーバーの位置を棒グラフに合わせる
  position = position_dodge(width = 0.9),
  width = 0.25 # エラーバーの幅を設定
)

```



繰り返しになるが、このコードは慣れてくるまでいきなり書けるものではない。重要なのは「出力結果をイメージ」することと、それを「要素に分解」、「手順に沿って並べる」ことができるかどうかである。<sup>\*3</sup>

## 4.5 描画 tips

最後に、いくつかの描画テクニックを述べておく。これらについては、必要な時に随時ウェブ上で検索したり、生成 AI に尋ねることも良いが、このような方法がある、という基礎知識を持っておくことも重要だろう。なお描

<sup>\*3</sup> 実際コードは chatGPTver4 に指示して生成した。いきなり全体像を描くのではなく、徐々に追記していくと効果的である。

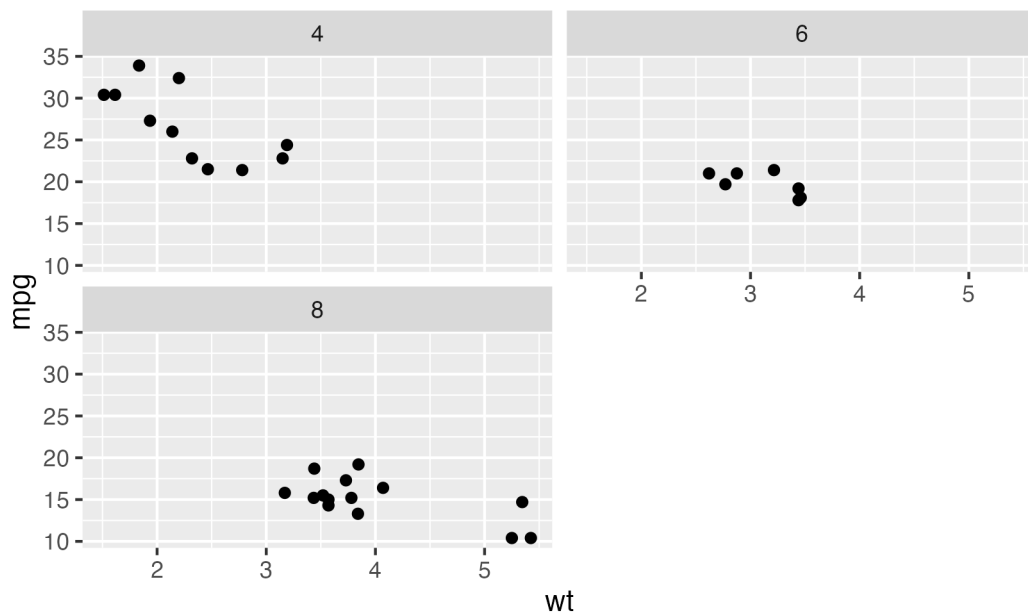
画について詳しくは@Kinosady2021 の4章を参照すると良い。

### 4.5.1 ggplot オブジェクトを並べる

複数のプロットを一枚のパネルに配置したい、ということがあるかもしれない。先ほどの `mtcars` データの例でいえば、`am` 変数にオートマチック車かマニュアル車かの2水準があるが、このようなサブグループごとに図を分割したいという場合である。

このような時には、`facet_wrap` や `facet_grid` という関数が便利である。前者はある変数について、後者は2つの変数について図を分割する。

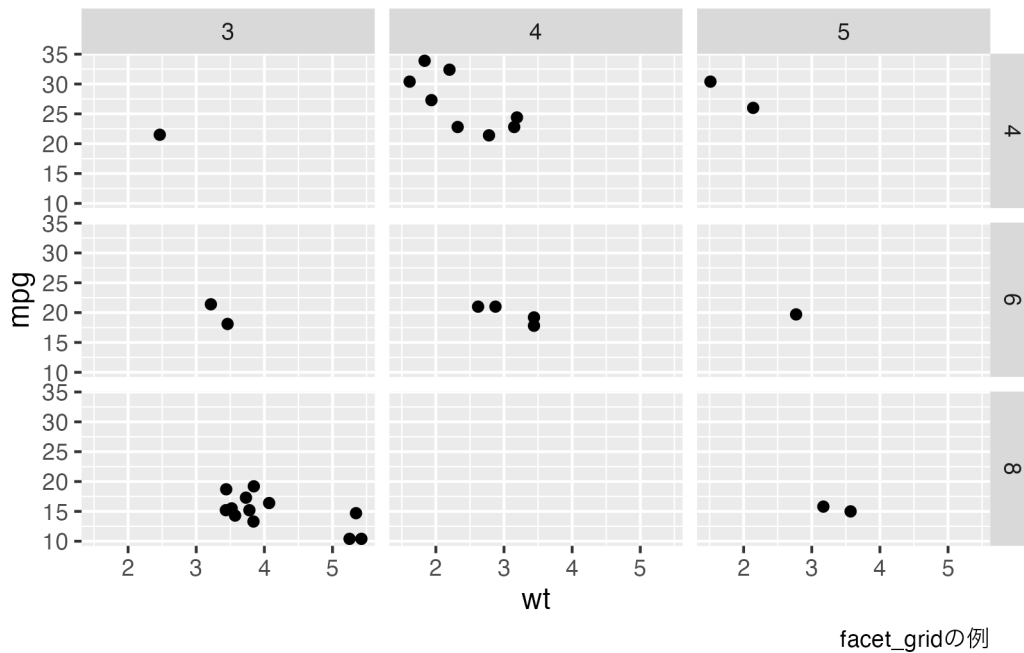
```
mtcars %>%
  # 重さ wt と燃費 mpg の散布図
  ggplot(aes(x=wt, y=mpg)) +
  geom_point() +
  # シリンダ数 cyl で分割
  facet_wrap(~ cyl, nrow=2) +
  # タイトルをつける
  labs(caption = "facet_wrap の例")
```



facet\_wrapの例

```
mtcars %>%
  ggplot(aes(x=wt, y=mpg)) +
  geom_point() +
  # シリンダ数 cyl とギア数 gear で分割
  facet_grid(cyl ~ gear) +
  # キャプションをつける
```

```
labs(caption = "facet_grid の例")
```



一枚の図をサブグループに分けるのではなく、異なる図を一枚の図として赤痛いこともあるかもしれない。そのような場合は、`patchwork` パッケージを使うと便利である。

```
library(patchwork)

# 散布図の作成
g1 <- ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point() +
  # 散布図のタイトルとサブタイトル
  ggtitle("Scatter Plot", "MPG vs Weight")

# 棒グラフの作成
g2 <- ggplot(mtcars, aes(x=factor(cyl), y=mpg)) +
  geom_bar(stat="identity") +
  # 棒グラフのタイトルとサブタイトル
  ggtitle("Bar Chart", "Average MPG by Cylinder")

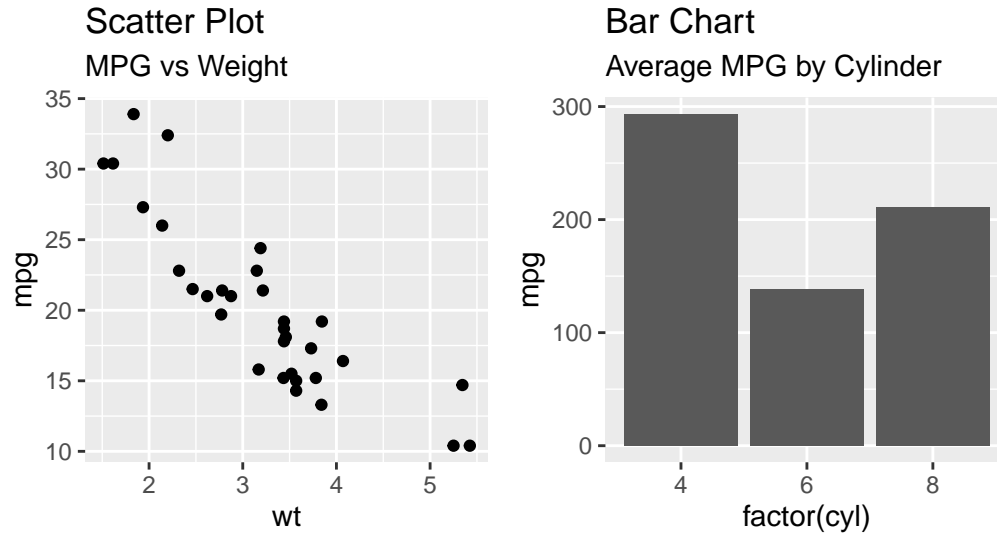
# patchwork を使用して 2 つのグラフを組み合わせる
combined_plot <- g1 + g2 +
  plot_annotation(title = "Combined Plots",
                  subtitle = "Scatter and Bar Charts")

# プロットを表示
```

```
print(combined_plot)
```

## Combined Plots

### Scatter and Bar Charts



### 4.5.2 ggplot オブジェクトの保存

Rmd や Quarto で文書を作るときは、図が自動的に生成されるので問題ないが、図だけ別のファイルとして利用したい、保存したいということがあるかもしれない。その時は `ggsave` 関数で `ggplot` オブジェクトを保存するとよい。

```
# 散布図を作成
p <- ggplot(mtcars, aes(x=wt, y=mpg)) + geom_point()
ggsave(
  filename = "my_plot.png", # 保存するファイル名。
  plot = p,                 # 保存するプロットオブジェクト。
  device = "png",           # 保存するファイル形式。
  path = "path/to/directory", # ファイルを保存するディレクトリのパス
  scale = 1,                # グラフィックスの拡大縮小比率
  width = 5,                # 保存するプロットの幅 (インチ)
  height = 5,               # 保存するプロットの高さ (インチ)
  dpi = 300,                # 解像度 (DPI: dots per inch)
)
```

### 4.5.3 テーマの変更 (レポートに合わせる)

レポートや論文などの提出次の条件として、図版をモノクロで表現しなければならないことがあるかもしれない。`ggplot` では自動的に配色されるが、その背後ではデフォルトの絵の具セット (パレットという) が選択されてい

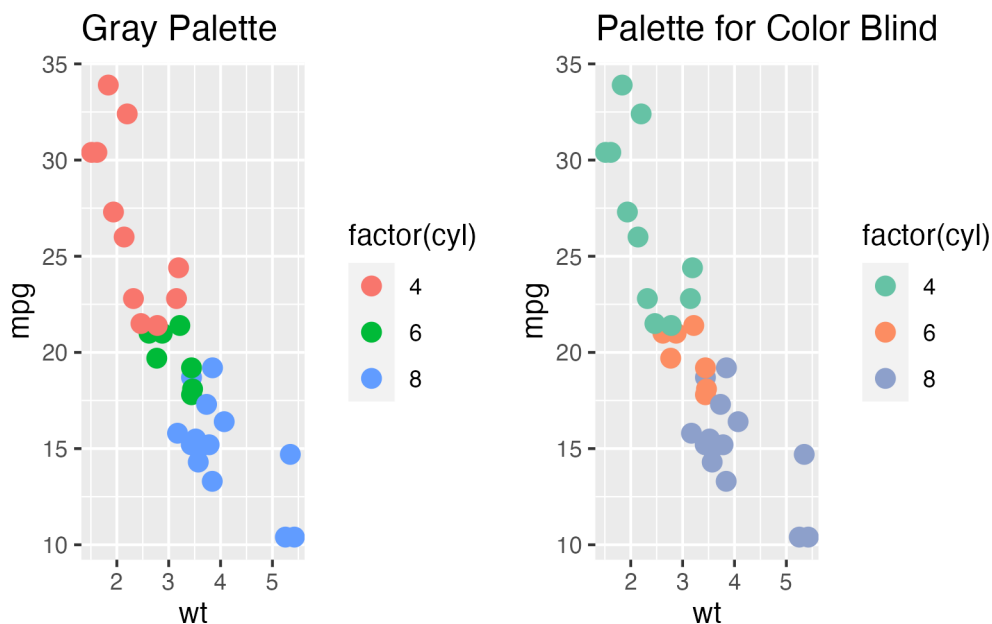
るからである。このセットを変更すると、同じプロットでも異なる配色で出力される。モノクロ (グレイスケール) で出力したい時のパレットは `Greys` である。

```
# グレースケールのプロット
p1 <- ggplot(mtcars, aes(x=wt, y=mpg, color=factor(cyl))) +
  geom_point(size=3) +
  scale_fill_brewer(palette = "Greys") +
  ggtitle("Gray Palette")

# カラーパレットが多く含まれているパッケージの利用
library(RColorBrewer)

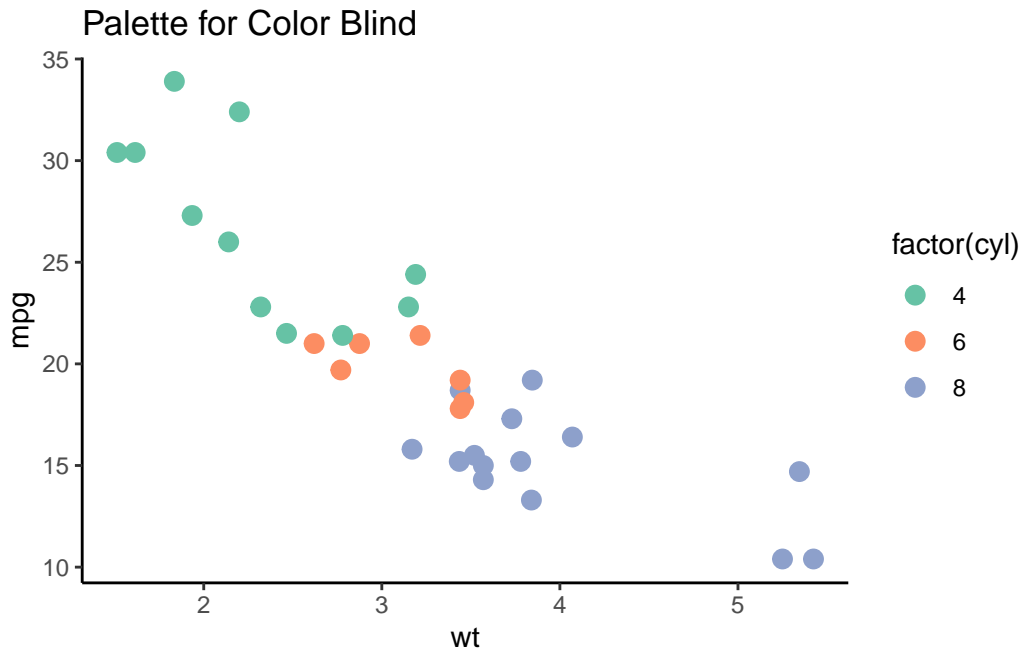
# 色覚特性を考慮したカラーパレット
p2 <- ggplot(mtcars, aes(x=wt, y=mpg, color=factor(cyl))) +
  geom_point(size=3) +
  scale_color_brewer(palette="Set2") + # 色覚特性を考慮したカラーパレット
  ggtitle("Palette for Color Blind")

# 両方のプロットを並べて表示
combined_plot <- p1 + p2 + plot_layout(ncol = 2)
print(combined_plot)
```



また、`ggplot2` のデフォルト設定では、背景色が灰色になっている。これは全体のテーマとして `theme_gray()` が設定されているからである。しかし日本心理学会の執筆・投稿の手引きに記載されているグラフの例を見ると、背景は白色とされている。このような設定に変更するためには、`theme_classic()` や `theme_bw()` を用いる。

```
p2 + theme_classic()
```



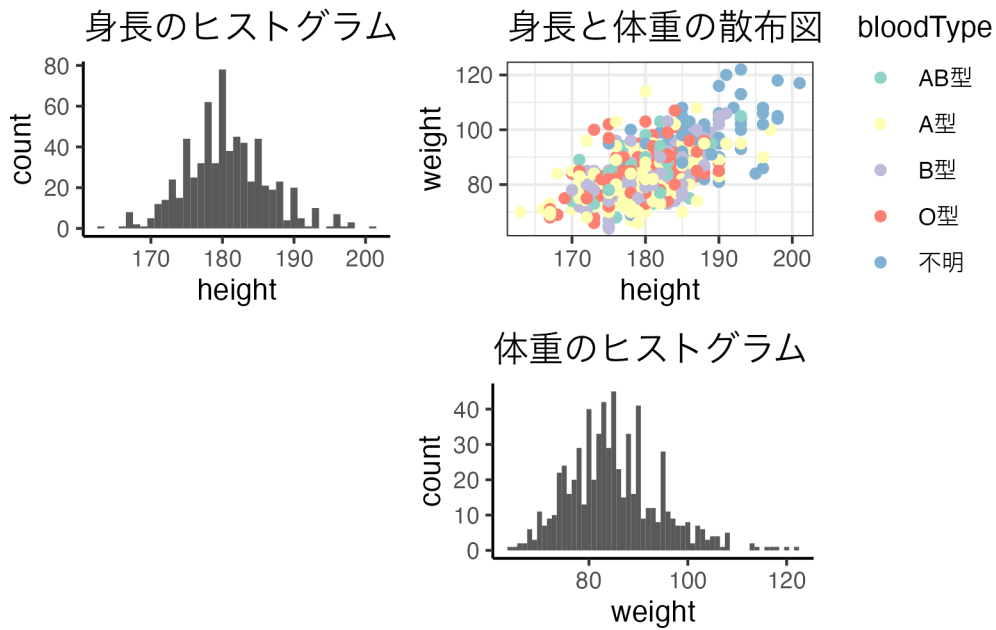
このほかにも、様々な描画上の工夫は考えられる。目標となる図版のレシピを書き起こせるように、要素に分解ができれば、殆どのケースにおいて問題を解決することができるだろう。

## 4.6 マークダウンと描画の課題

- 今日の課題は Rmarkdown で記述してください。著者名に学籍番号と名前を含め、適宜見出しをつくり、かつ、平文で以下に挙げる課題を記載することでどの課題に対する回答のコード (チャンク) であるかわかるようにしてください。
1. Baseball.csv を読み込み、2020 年度のデータセットに限定し、以下の操作に必要であれば変数の変換をすませたデータセット、dat.tb を用意してください。
  2. dat.tb の身長変数を使って、ヒストグラムを描いてください。この時、テーマを theme\_classic にしてください。
  3. dat.tb の身長変数と体重変数を使って、散布図を描いてください。この時、テーマを theme\_bw にしてください。
  4. (承前) 散布図の各点を血液型で塗り分けてください。この時、カラーパレットを Set3 に変えてください。
  5. (承前) 散布図の点の形を血液型で変えてください。
  6. dat.tb の身長と体重についての散布図を、チームごとに分割してください。
  7. (承前) geom\_smooth() でスムーズな線を引いてください。特に method を指定する必要はありません。
  8. (承前) geom\_smooth() で直線関数を引いてください。method="lm" と指定するといいいでしょう。
  9. x 軸は身長、y 軸は体重の平均値をプロットしてください。方法はいろいろありますが、要約統計量を計算した別のデータセット dat.tb2 を作るか、幾何学オブジェクトの中で、次のように関数を適用することもできます。ヒント: geom\_point(stat="summary", fun=mean)。



10. 課題 2,4 および体重のヒストグラムをつかって下の図を描き, `ggsave` 関数を使って保存するコードを書いてください。ファイル名やその他オプションは任意です。



## 4.7 References



## 第 5 章

# R でプログラミング

5.1 代入

5.2 反復

5.3 条件分岐

5.4 自作関数

5.5 さまざまな乱数

5.6 確率分布の関数

5.7 References



## 第 6 章

# 確率とシミュレーション

6.1 サンプル関数

6.2 期待値と分散のシミュレーション

6.3 母集団と標本

6.4 一様性

6.5 不偏性

6.6 有効性

6.7 References



## 第 7 章

# 統計的仮説検定の論理とエラー

- 7.1 帰無仮説検定の論理
- 7.2 相関係数の検定
- 7.3 標本相関係数の分布
- 7.4 2 種類の検定のエラー確率
- 7.5 References





## 第 8 章

# 平均値差の検定

- 8.1 一標本検定
- 8.2 二標本検定
- 8.3 二標本検定（ウェルチの補正）
- 8.4 対応のある二標本検定
- 8.5 レポートを書くような課題
- 8.6 References



## 第 9 章

# 多群の平均値差の検定

9.1 分散分析の基礎

9.2 検定の多重性

9.3 ANOVA 君を使う

9.4 Between デザイン

9.5 Within デザイン

9.6 References



## 第 10 章

# 帰無仮説検定のシミュレーション

10.1 統計的検定と QRP<sub>s</sub>

10.2 タイプ 2 エラー確率のコントロールとサンプルサイズ設計

10.3 サンプルサイズ設計の実践

10.3.1 一標本  $t$  検定

10.3.2 二標本  $t$  検定

10.3.3 相関係数のサンプルサイズ設計

10.4 References



## 第 11 章

# 回帰分析

- 11.1 回帰分析の基礎
- 11.2 重回帰分析の場合
- 11.3 回帰分析のいくつかの特徴
- 11.4 シミュレーションとパラメタリカバリ
- 11.5 係数の標準誤差
- 11.6 係数の検定
- 11.7 サンプルサイズ設計
- 11.8 References





## 第 12 章

# 線型モデルの展開

- 12.1 一般線型モデル
- 12.2 一般化線型モデル
- 12.3 階層線型モデル
- 12.4 References



## 第 13 章

# 多変量解析の入り口

13.1 因子分析

13.2 構造方程式モデリング

13.3 References



# 引用文献

- Bernaards, C. A. & Jennrich, R. I. (2005). Gradient Projection Algorithms and Software for Arbitrary Rotation Criteria in Factor Analysis. *Educational and Psychological Measurement*, 65, 676–696. <https://doi.org/10.1177/0013164404272507>
- Gabry, J., Češnovar, R., & Johnson, A. (2023). *cmdstanr: R Interface to 'CmdStan'*. <https://mc-stan.org/cmdstanr/>, <https://discourse.mc-stan.org>.
- Hadley, W. (2014). Tidy Data. *Journal of Statistical Software*, 59, 1–23. <https://doi.org/10.18637/jss.v059.i10>
- 松村 優哉・湯谷 啓明・紀ノ定 保礼・前田 和寛 (2021). 改訂 2 版 R ユーザのための RStudio[実践] 入門——tidyverse によるモダンな分析フローの世界—— 技術評論社
- Revelle, W. (2021). *psych: Procedures for Psychological, Psychometric, and Personality Research*. R package version 2.1.3. Northwestern University. Evanston, Illinois. from <https://CRAN.R-project.org/package=psych>
- Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software*, 48 (2), 1–36. <https://doi.org/10.18637/jss.v048.i02>
- Stevens, S. S. (1946). On the theory of scales of measurement. *Science*, 103 (2684), 677–680.
- 高橋 康介 (2018). 再現可能性のすゝめ 石田 基広 (編) Wonderful R 3 共立出版
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4 (43), 1686. <https://doi.org/10.21105/joss.01686>
- Zeileis, A. (2005). CRAN Task Views. *R News*, 5 (1), 39–40.
- 総務省 (2020). 統計表における機械判別可能なデータ作成に関する表記方法.