心理学統計実習

Exercises in Psychological Statistics with R/RStudio

Koji Kosugi

Table of contents

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

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[*1]。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[*2]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[*3]。

[*1] 2024 年 1 月現在で，Tokyo だけでなく Fukuoka，Sapporo，Yamaguchi，Iruma などで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。

[*2] 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。

[*3] 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良い。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 と畳は新しい方が良い。

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[*4]。

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

[*4] 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。

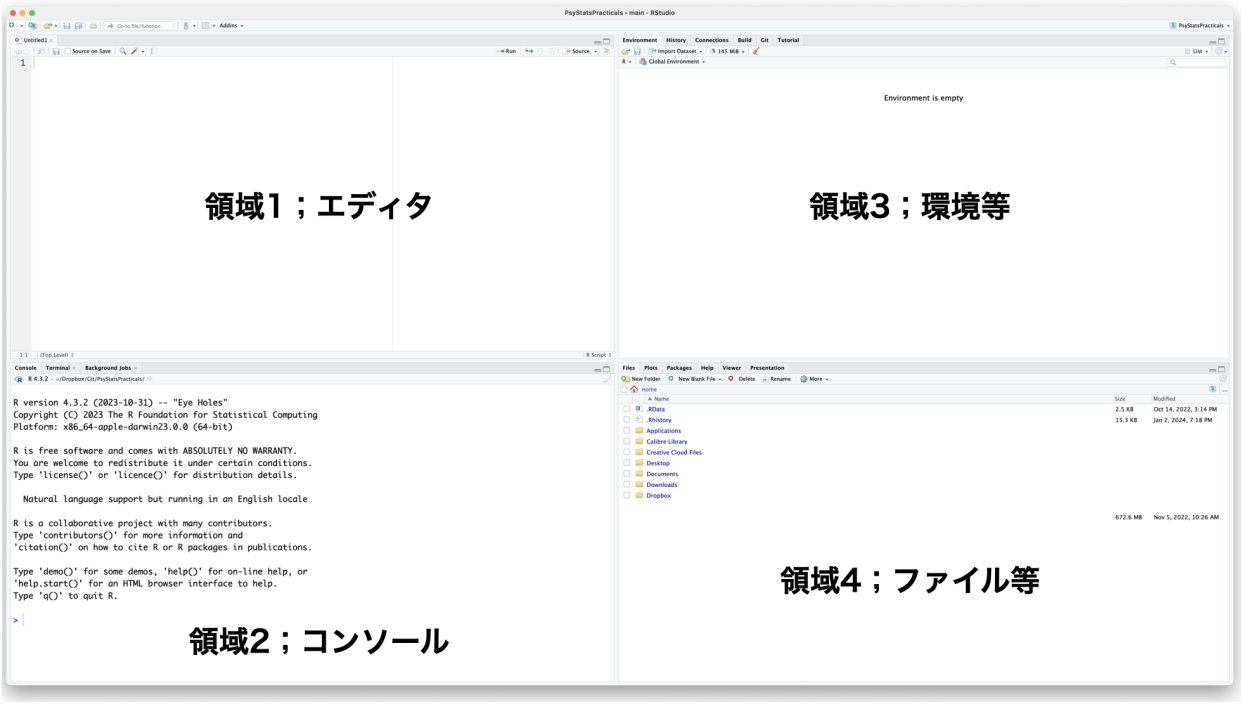


Figure1.1: RStudio の初期画面

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。

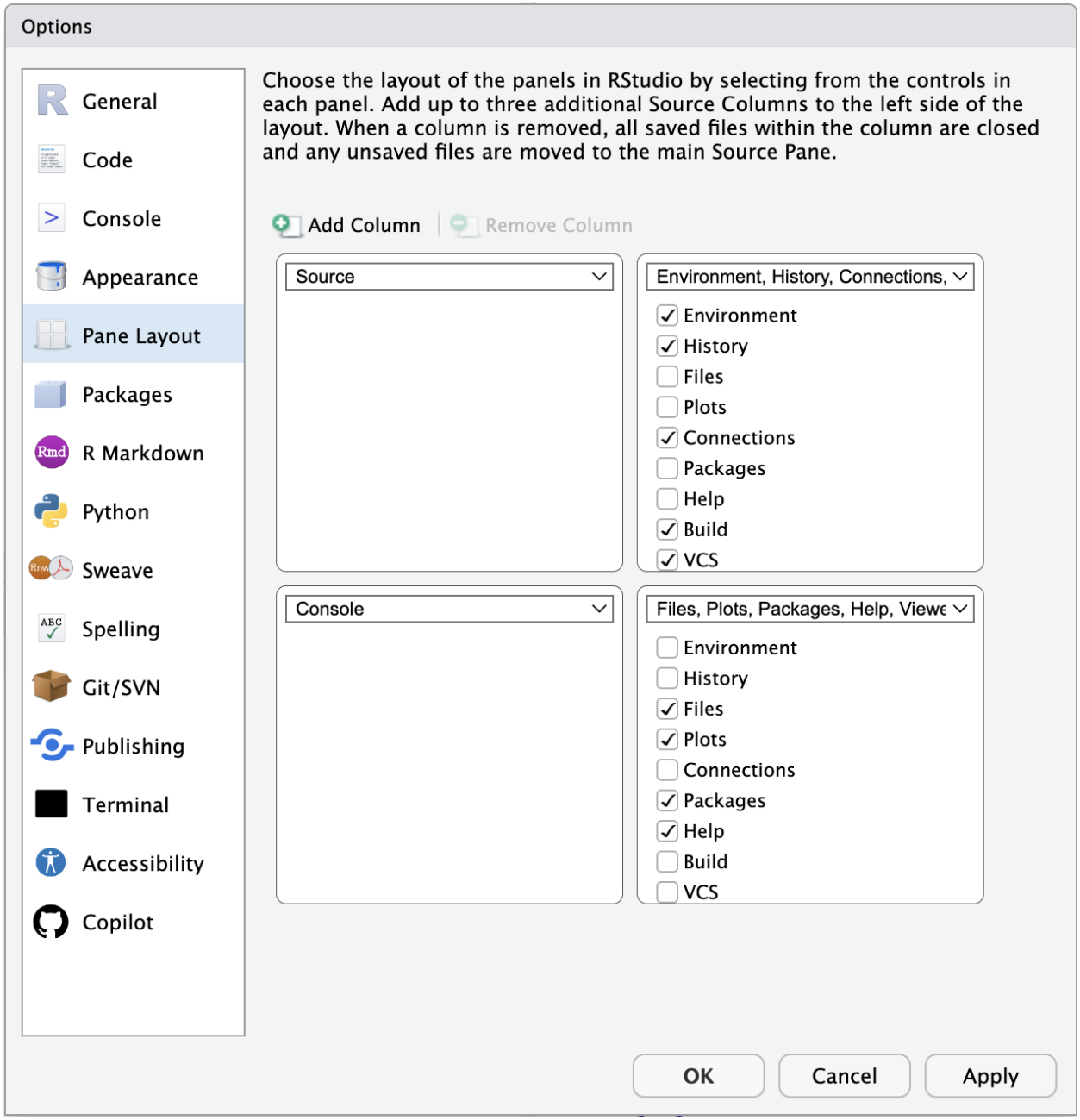


Figure1.2: レイアウト変更画面。このほかにも背景色などを変えることもできる

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に利用する。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ボ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に利用する。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 に命令するときに使う。

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[*5]，Github[*6]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。

[*5] 2024 年 01 月 18 日調べ

[*6] 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 とも連携しており，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，最近は 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 パッケージ導入以後のことである。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[*7]。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

[*7] 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)[*8]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。
- RStido 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。

[*8] 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。)。

第 2 章

R の基礎

ここから実際に R/RStudio をつかった演習に入る。前回すでに言及したように，この講義ようの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と良い (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) を含み[*1]，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。

[*1] 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 型にしておくと，カテゴリとして使えて便利である。

次に示すのは順序つき 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。たとえば先ほどのおオブジェクト `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

解析の実際では，データセットを手入力することはなく，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[*2]。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
```

[*2] 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 などで開くと文字化けし，以下の手続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 などで開くことなく，直接 R から読み込むようにされたし。

```
1 2011 年度 永川　勝浩 Carp  12000       O 型    188     97          20      投手
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 `Tbl` 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 `Tbl` 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)
- `Tbl` オブジェクトの人口 (Pop) 変数の平均を計算してください。
- `Tbl` 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 などで整形してください。

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 をつか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     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
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，forecats 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，特段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，苦労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. パイプ演算子

- `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ましょう。どこを見れば良いでしょうか。`filter`,`lag` 関数はどうでしょうか。
- `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りました。`dplyr` パッケージの `filter` 関数をヘルプで見てみましょう。
- 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てみましょう。
- 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ましょう。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ものです。また絶対値を計算する R 関数は `abs` です。

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

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

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な関数がある。

- 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。たとえば連続変数をある値を境に「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だす」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上書きされるため，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.select, filter, mutate

- `Baseball.csv` を読み込んで、データフレーム `df` に代入しましょう。
- `df` には複数の変数が含まれています。変数名の一覧は `names` 関数で確認できます。`df` オブジェクトに含まれる変数名を確認しましょう。
- `df` には多くの変数がありますが、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) だけです。これらの変数だけを選択して、`df2` オブジェクトを作成しましょう。
- `df2` には数年分のデータが含まれています。2020 **年度**のデータだけを分析したいので、選別してみましょう。
- 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ましょう。
- 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でしょうか。
- 選手の身体的特徴を表す BMI 変数を作成しましょう。なお、BMI は体重 (kg) を身長 (m) の二乗で除したものです。変数 `height` の単位が cm であることに注意しましょう。
- 投手と野手を区別する新しい変数 `position2` を作成しましょう。これは Factor 型にします。なお、野手は投手でないもの、すなわち内野手、外野手、捕手のいずれかです。
- 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) に分かれています。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す。`df2` を加工して、所属するリーグの変数 `League` を作成しましょう。この変数も Factor 型にしておきましょう。
- 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います。実際に使うときは不便なので、「年度」という文字を除外し、数値型変数に変換しましょう。

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: ワイド型データ

	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と「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[*2]。

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
```

[*2] 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。

```
# 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),
    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)[*3]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) %>%
```

[*3] 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。

```
  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 (tilda,~) 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>
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 なども利用できる。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 課題 3. データの整形

- 上で作った df2 オブジェクトを利用します。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きましょう。
- 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)、平均年俸を見てみましょう。
- 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)、平均年俸を見てみましょう。
- 続いて、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、ワイド型データを作りたいと思います。pivot_wider を使って上のオブジェクトをワイド型にしてみましょう。
- 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ましょう。

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こらない。これを解決するのが R markdon や Quarto という書式・ソフトウェアなのである。

Rmarkdown の `markdown` とは書式の一種である。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、詳しくは高橋 (2018) 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ゥが開き，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。

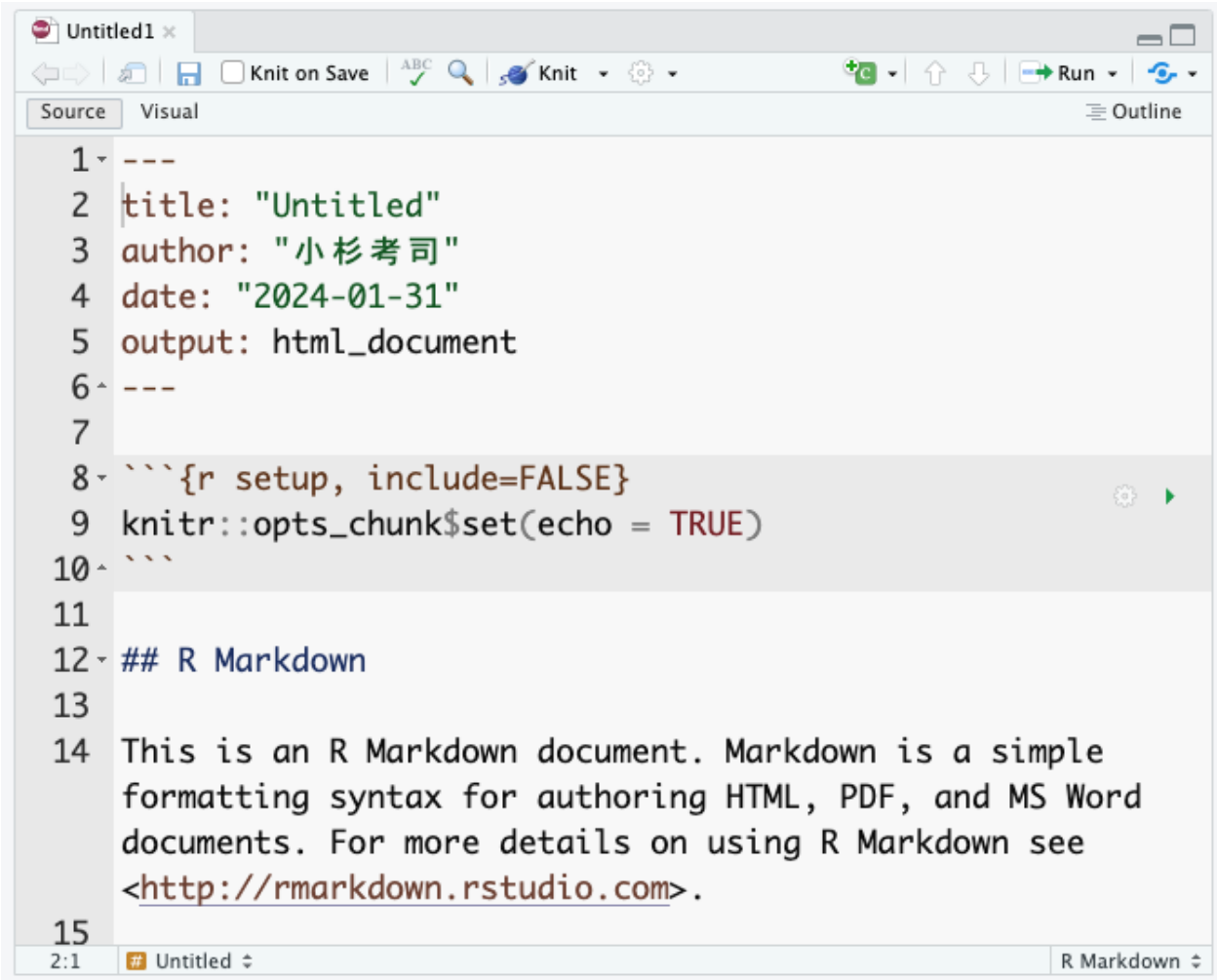


Figure4.1: Rmarkdown ファイルのサンプル

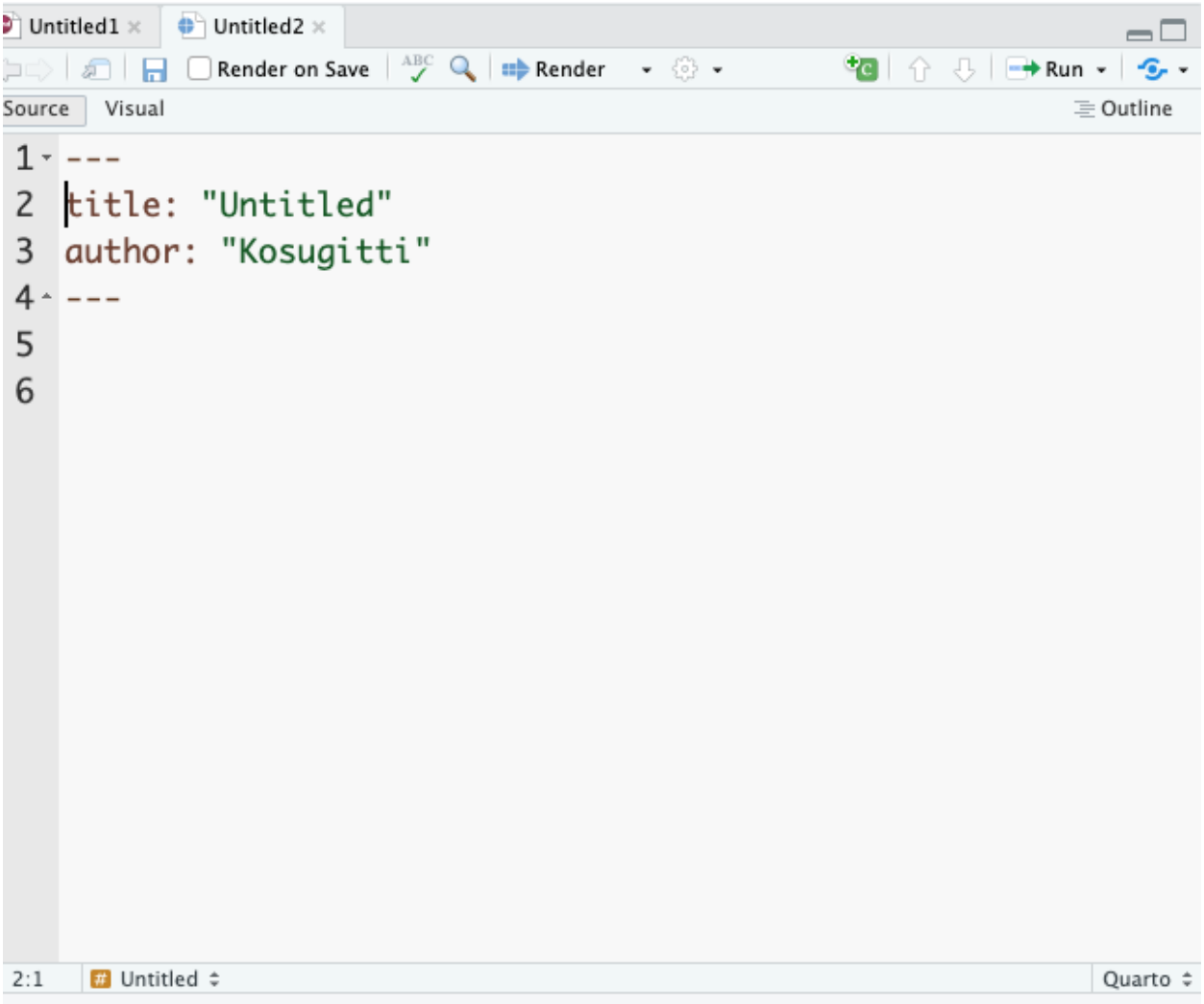


Figure4.2: Quarto ファイルのサンプル

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。[*1]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

[*1] 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。

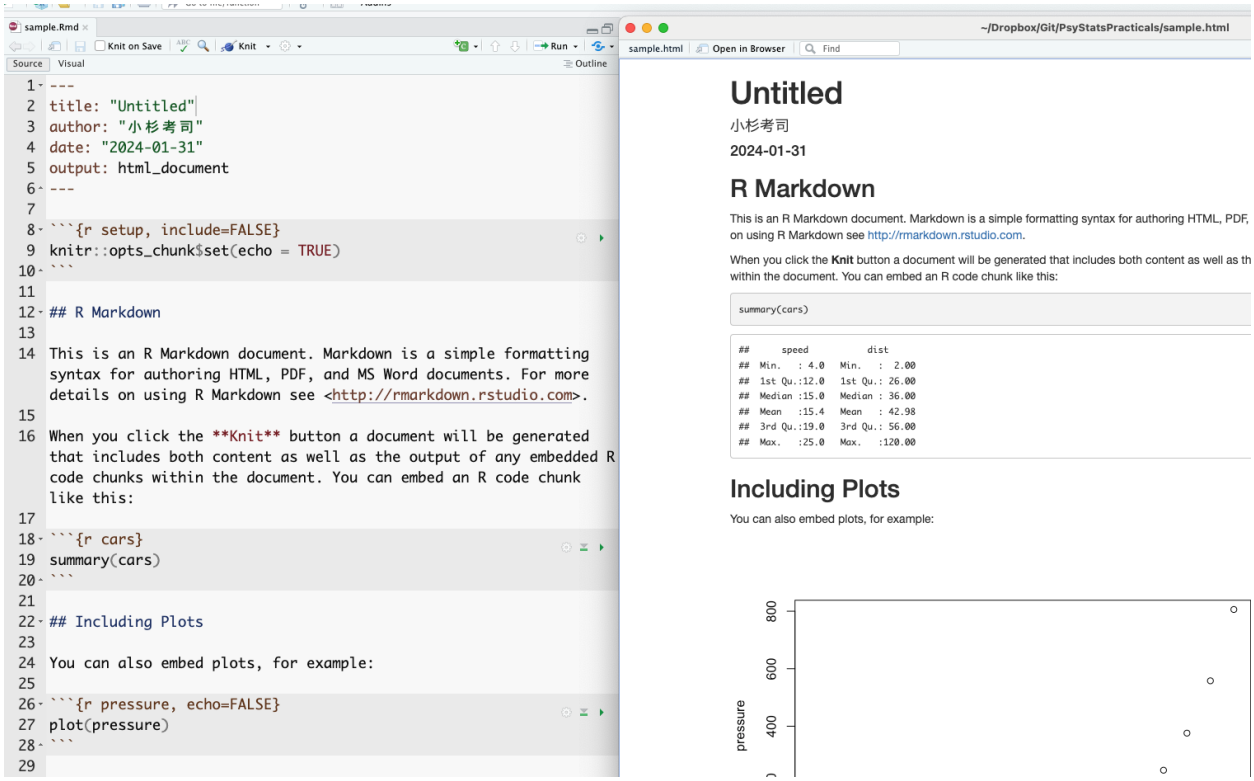


Figure4.3: Rmd ファイルと出力結果の対応

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，同じ加工をしていれば，環境が違っても追跡可能である。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が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。不便に感じるところ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。[*2]

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。

[*2] 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 Juilia 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) “‘

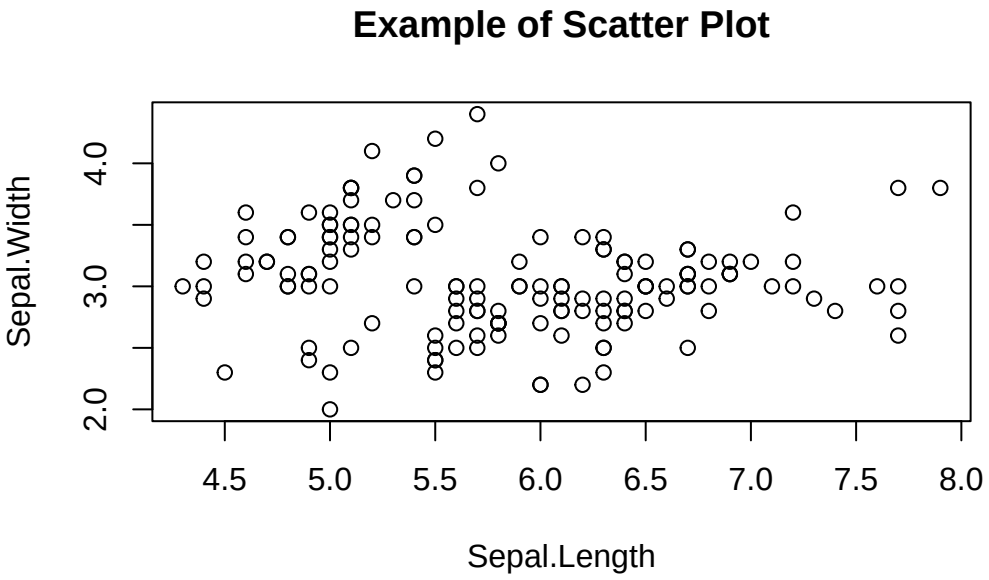
4.2 プロットによる基本的な描画

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 Healy (2018 瓜生他訳 2021) 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

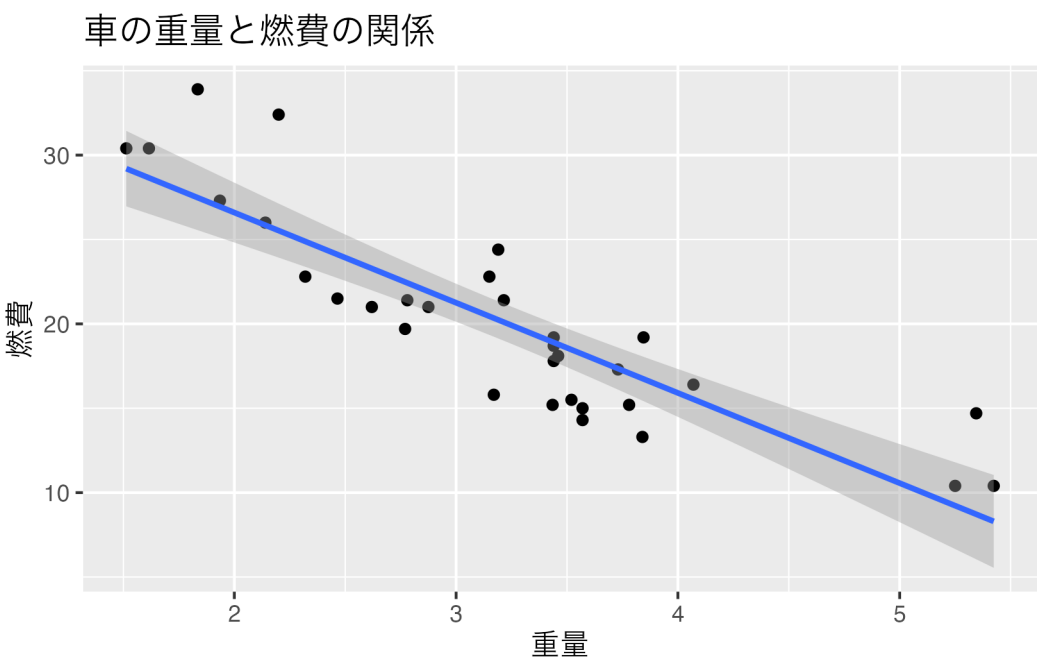
ここでは，`tidyverse` に含まれる描画専用のパッケージである，`ggplot2` パッケージを用いた描画を学ぶ。R の基本描画関数でもかなりのことができるのだが，この `ggplot2` パッケージをもちいた図の方が美しく，直観的に操作できる。というのも `ggplot` の `gg` とは `The Grammar of Graphics`(描画の文法) のことであり，このことが示すようにロジカルに図版を制御できるからである。`ggplot2` の形で記述された図版のスクリプトは可読性が高く，視覚的にも美しいため，多くの文献で利用されている。

`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，すぐにで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")
```



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。

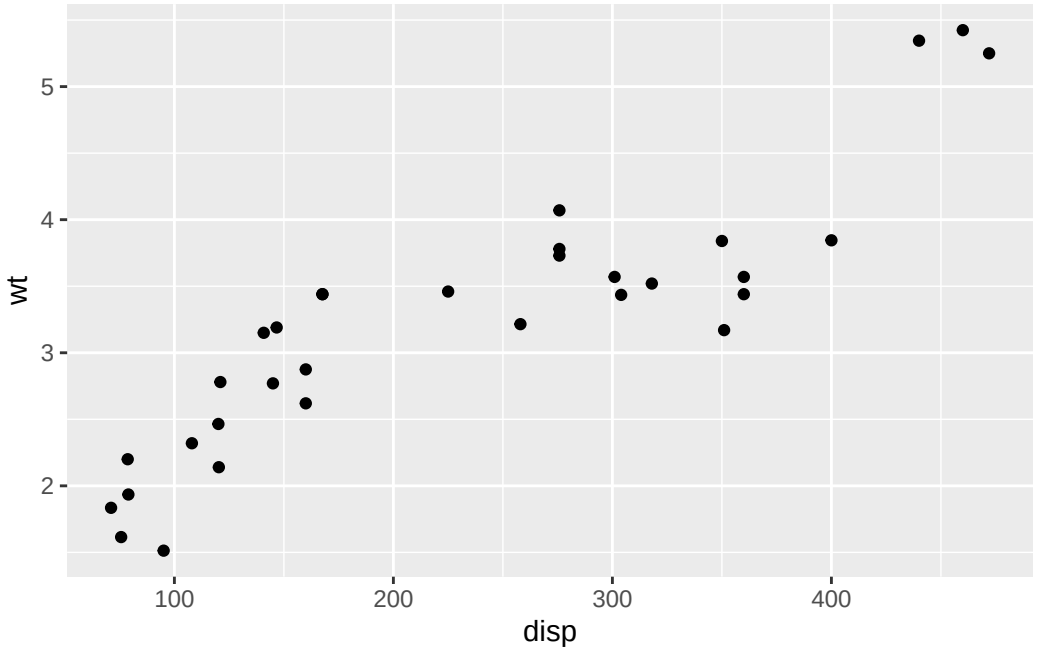
4.4　幾何学的オブジェクト geom

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。

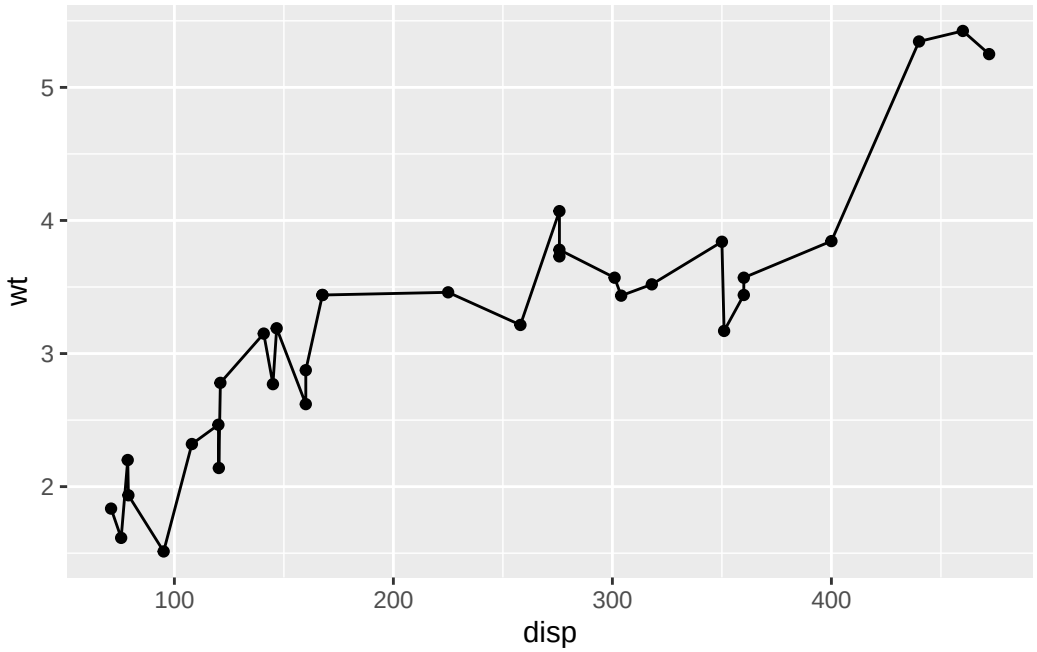
```
ggplot() +
  geom_point(data = mtcars, mapping = aes(x = disp, y = wt))
```



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。

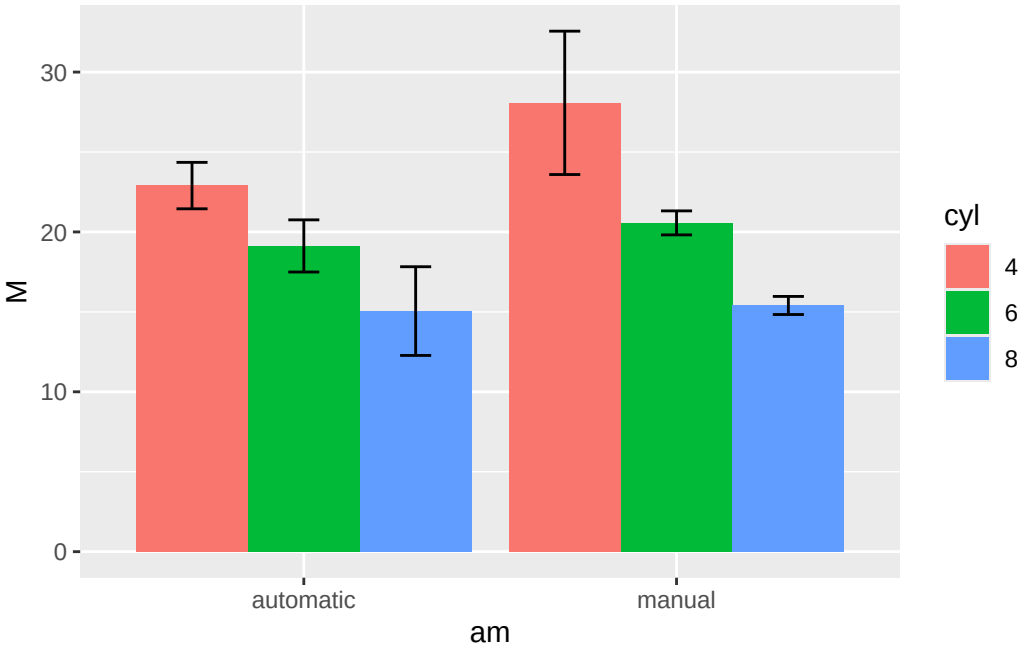
```
mtcars %>%
  ggplot(mapping = aes(x = disp, y = wt)) +
  geom_point() +
  geom_line()
```

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。[*3]

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 に尋ねることでも良いが，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

[*3] 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。

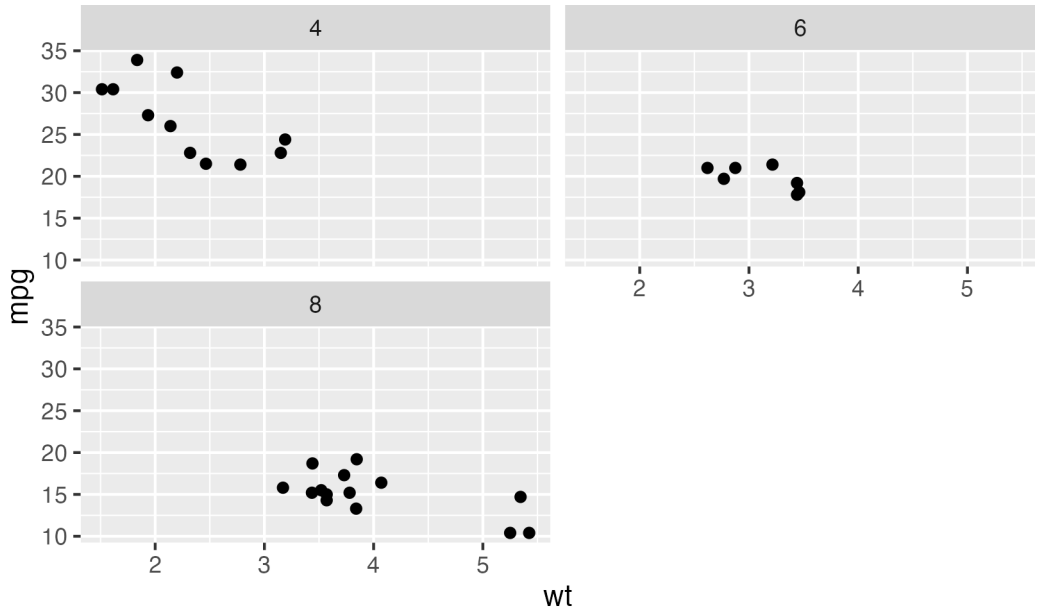
画について詳しくは松村他 (2021) の 4 章を参照すると良い。

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。

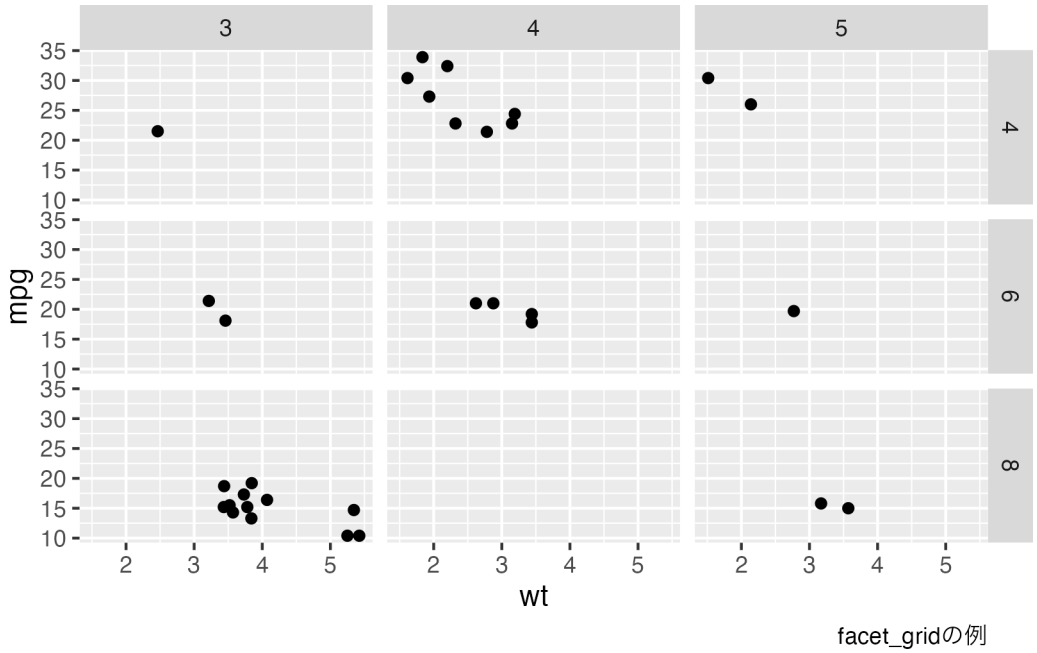
```
mtcars %>%
  # 重さ wt と燃費 mpg の散布図
  ggplot(aes(x = wt, y = mpg)) +
  geom_point() +
  # シリンダ数 cyl で分割
  facet_wrap(~cyl, nrow = 2) +
  # タイトルをつける
  labs(caption = "facet_wrap の例")
```



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
```

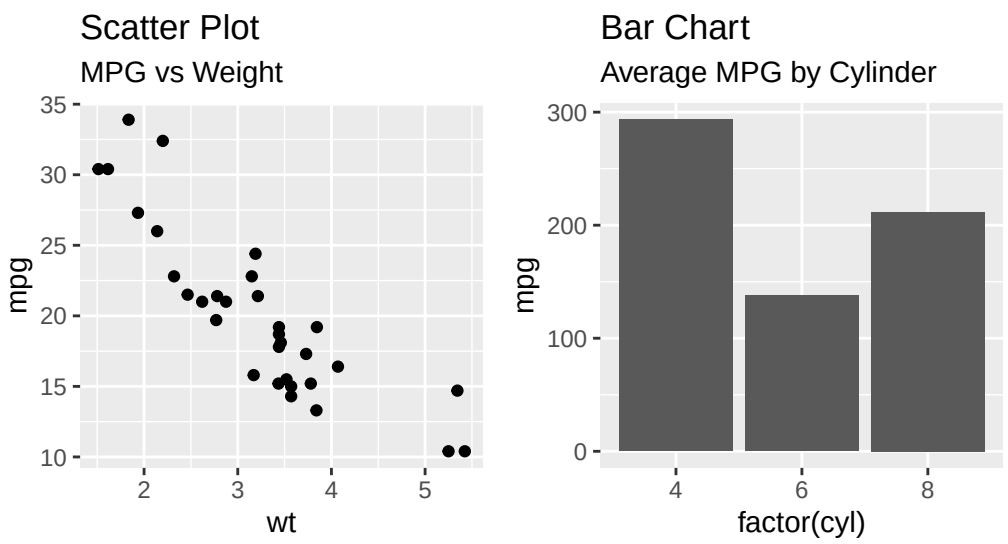
```
  labs(caption = "facet_grid の例")
```



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)
```



4.5.2　ggplot オブジェクトの保存

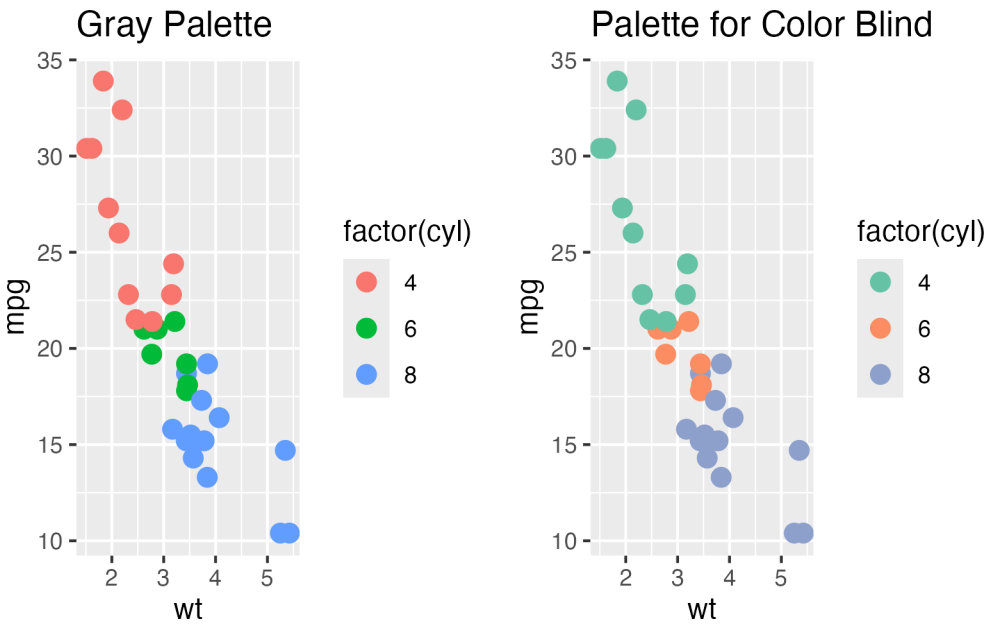
Rmd や Quarto で文書を作るときは，図が自動的に生成されるので問題ないが，図だけ別のファイルとして利用したい，保存したいということがあるかもしれない。その時は `ggsave` 関数で `ggplot` オブジェクトを保存するとよい。

```
# 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）

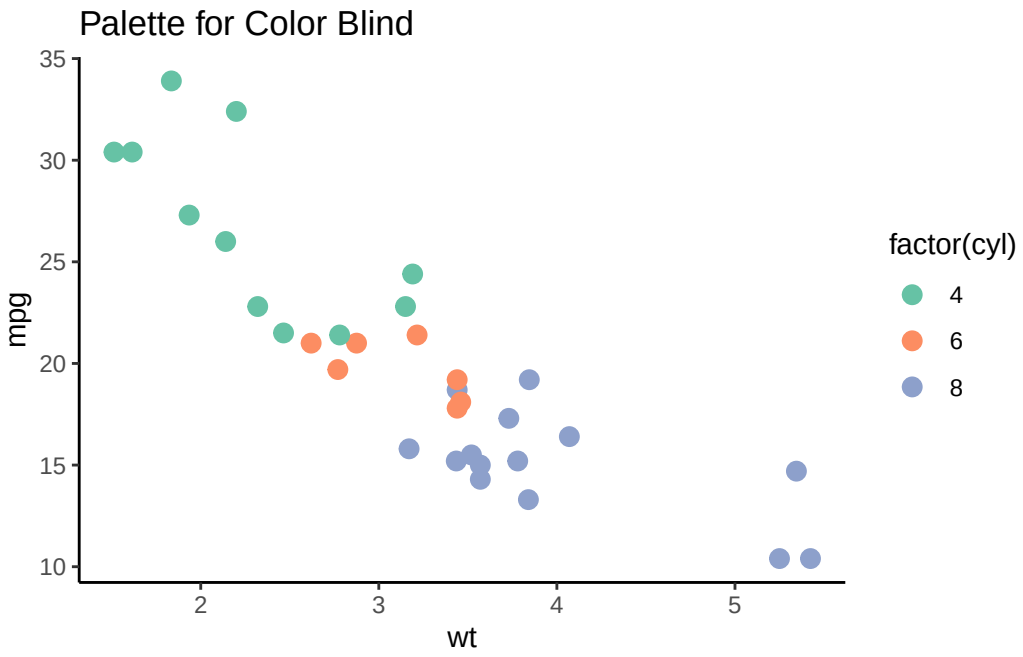
レポートや論文などの提出次の条件として，図版をモノクロで表現しなければならないことがあるかもしれない。ggplot では自動的に配色されるが，その背後ではデフォルトの絵の具セット (**パレット**という) が選択されているからである。このセットを変更すると，同じプロットでも異なる配色で出力される。モノクロ (グレイスケール) で出力したい時のパレットは Grays である。

```
# 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。

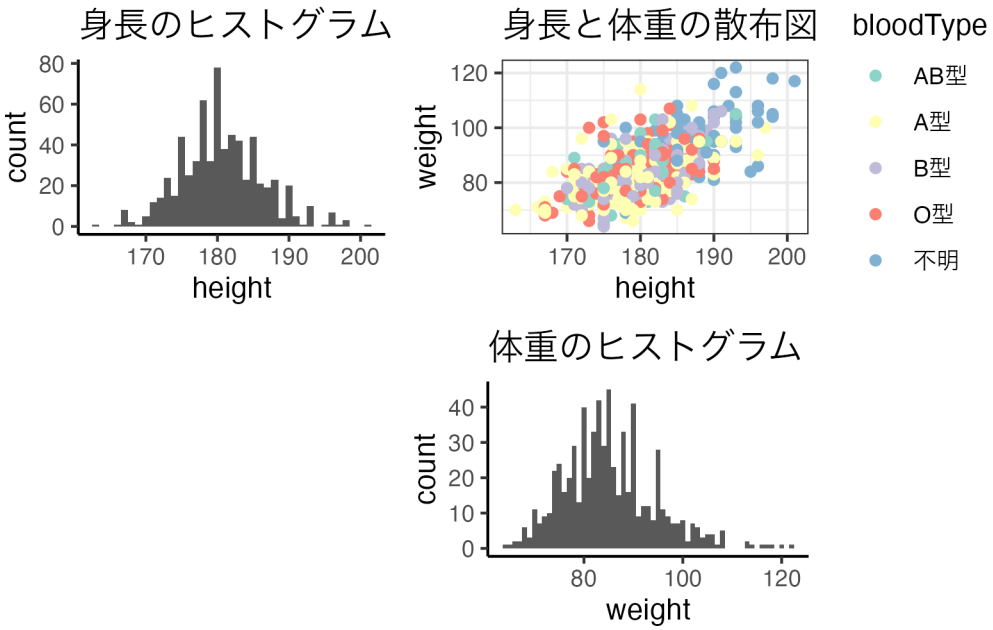
4.6　課題

以下にデスマス調に整えた文章を示します。

- 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、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済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使って下の図を描き、`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。



第 5 章

R でプログラミング

ここではプログラミング言語としての R について解説する。なお副読本として小杉他 (2023) を挙げておく。また，プログラミングのより専門的な理解のために，Lander (2017 高柳他訳 2018), Ren (2016 株式会社ホクソエム監訳 2017), Wickham (2015 石田他訳 2016) なども参考にすると良い。

プログラミング言語は，古くは C や Java，最近では Python や Julia などがよく用いられている。R も統計パッケージというよりプログラミング言語として考えるのが適切かもしれない。R は他のプログラミング言語に比べて，変数の型宣言を事前にしなくても良いことや，インデントなど書式についておおらかなところは，初心者にとって使いやすいところだろう。一方で，ベクトルの再利用のところで注意したように (Section 2.5.1)，不足分を補うために先回りして補填されたり，この後解説する関数の作成時に明示的な指定がなければ環境変数を参照する点など，親切心が空回りするところがある。より厳格な他言語になれていると，こうした点はかえって不便に思えるところもあるかもしれない。総じて，R 言語は初心者向けであるといえるだろう。

さて，世にプログラミング言語は多くあれど[*1]，その全てに精通する必要はないし，不可能である。それよりも，プログラミング言語一般に通底する基本的概念を知り，あとは各言語による「方言」がある，と考えた方が生産的である。その基本的概念を 3 つ挙げるとすれば，「代入」「反復」「条件分岐」になるだろう。

5.1 代入

代入は，言い換えればオブジェクト (メモリ) に保管することを指す。これについては既に Chapter 2 で触れた通りであり，ここでは言及しない。オブジェクトや変数の型，常に上書きされる性質に注意しておけば十分だろう。

一点だけ追加で説明しておくと，次のような表現がなされることがある。

```
a <- 0
a <- a + 1
print(a)
```

```
[1] 1
```

[*1] シ (2016) には 117 種もの計算機言語が紹介されている。

ここではあえて，代入記号として=を使った。2 行目に a = a + 1 とあるが，これを見て数式のように解釈しようとすると混乱する。数学的には明らかにおかしな表現だが，これは上書きと代入というプログラミング言語の特徴を使ったもので，「(いま保持している)a の値に 1 を加えたものを，(新しく同じ名前のオブジェクト)a に代入する (=上書きする)」という意味である。この方法で，a をカウンタ変数として用いることがある。誤読の可能性を下げるため，この授業においては代入記号を<-としている。

このオブジェクトを上書きするという特徴は多くの言語に共通したものであり，間違いを避けるためには，オブジェクトを作る時に初期値を設定することが望ましい。先の例では，代入の直上で a <- 0 としており，オブジェクト a に 0 を初期値として与えている。この変数の初期化作業がないと，以前に使っていた値を引き継いでしまう可能性があるので，今から新しく使う変数を作りたいというときは，このように明示しておくといいだろう。

なお，変数をメモリから明示的に削除する場合は，remove 関数を使う。

```
remove(a)
```

これを実行すると，RStudio の Environment タブからオブジェクト a が消えたことがわかるだろう。メモリの一斉除去は，同じく RStudio の Environment タブにある箒マークをクリックするか，remove(list=ls()) とすると良い[*2]。

5.2　反復

5.2.1　for 文

電子計算機の特徴は，電源等のハードウェア的問題がなければ疲労することなく計算を続けられるところにある。人間は反復によって疲労が溜まったり，集中力が欠如するなどして単純ミスを生成するが，電子計算機にそういったところはない。

このように反復計算は電子計算機の中心的特徴であり，細々した計算作業を指示した期間反復させ続けることができる。反復の代表的なコマンドは for であり，for ループなどと呼ばれる。for ループはプログラミングの基本的な制御構造であり，R 言語の for ループの基本的な構文は次のようになる：

```
for (value in sequence) {
    # 実行するコード
}
```

ここの value は各反復で sequence の次の要素を取る反復インデックス変数である。。sequence は一般にベクトルやリストなどの配列型のデータであり，「#実行するコード」はループ体内で実行される一連の命令になる。

以下は for 文の例である。

```
for (i in 1:5) {
  cat("現在の値は", i, "です。\n")
}
```

[*2] ls() 関数は list objects の意味で，メモリにあるオブジェクトのリストを作る関数

```
現在の値は 1 です。
現在の値は 2 です。
現在の値は 3 です。
現在の値は 4 です。
現在の値は 5 です。
```

for 文は続く小括弧のなかである変数を宣言し (ここでは i)，それがどのように変化するか (ここでは 1:5，すなわち 1,2,3,4,5) を指定する。続く中括弧の中で，反復したい操作を記入する。今回は cat 文によるコンソールへの文字力の出力を行っている。ここでのコマンドは複数あってもよく，中括弧が閉じられるまで各行のコマンドが実行される。

次に示すは，sequence にあるベクトルが指定されているので，反復インデックス変数が連続的に変化しない例である。

```
for (i in c(2, 4, 12, 3, -6)) {
  cat("現在の値は", i, "です。\n")
}
```

```
現在の値は 2 です。
現在の値は 4 です。
現在の値は 12 です。
現在の値は 3 です。
現在の値は -6 です。
```

また，反復はネスト (入れ子) になることもできる。次の例を見てみよう。

```
# 2 次元の行列を定義
A <- matrix(1:9, nrow = 3)

# 行ごとにループ
for (i in 1:nrow(A)) {
  # 列ごとにループ
  for (j in 1:ncol(A)) {
    cat("要素 [", i, ", ", j, "] は ", A[i, j], "\n")
  }
}
```

```
要素 [ 1 ,  1 ] は  1
要素 [ 1 ,  2 ] は  4
要素 [ 1 ,  3 ] は  7
要素 [ 2 ,  1 ] は  2
要素 [ 2 ,  2 ] は  5
要素 [ 2 ,  3 ] は  8
要素 [ 3 ,  1 ] は  3
```

```
要素 [ 3 ,  2 ] は  6
要素 [ 3 ,  3 ] は  9
```

ここで，反復インデックス変数が i と j というように異なる名称になっていることに注意しよう。例えば今回，ここで両者を i にしてしまうと，行変数なのか列変数なのかわからなくなってしまう。また少し専門的になるが，R 言語は for 文で宣言されるたびに，内部で反復インデックス変数を新しく生成している (異なるメモリを割り当てる) ためにエラーにならないが，他言語の場合は同じ名前のオブジェクトと判断されることが一般的であり，その際は値が終了値に到達せず計算が終わらないといったバグを引き起こす。反復に使う汎用的な変数名として i,j,k がよく用いられるため，自身のスクリプトの中でオブジェクト名として単純な一文字にすることは避けた方がいいだろう。

5.2.2　while 文

while ループはプログラミングの基本構造であり，特定の条件が真（True）である間，繰り返し一連の命令を実行する。「“while”（〜する間)」という名前から直感的に理解できるだろう。

R 言語の while ループの基本的な構文は次のようになる：

```
while (condition) {
    # 実行するコード
}
```

ここで，「condition」はループが終了するための条件である。「# 実行するコード」はループ体内で実行される一連の指示である。たとえば，1 から 10 までの値を出力する while ループは以下のように書くことができる：

```
i <- 1
while (i <= 5) {
  print(i)
  i <- i + 1
}
```

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

このコードでは，「i」が 5 以下である限りループが続く。「print(i)」で「i」の値が表示され，「i <- i + 1」で「i」の値が 1 ずつ増加する。これにより，「i」の値が 10 を超えると条件が偽 (False) となり，ループが終了する。

while ループを使用する際の一般的な注意点は，無限ループ（終わらないループ）を避けることである。これは，condition が常に真 (True) である場合に発生する。そのような状況を避けるためには，ループ内部で何らかの形で condition が最終的に偽 (False) となるようにコードを記述することが必要である。

また，R 言語は他の多くのプログラミング言語と異なり，ベクトル化された計算を効率的に行う設計がされてい

る。したがって，可能な限り `for` ループや `while` ループを使わずに，ベクトル化した表現を利用すれば計算速度を上げることができる。

5.3 条件分岐

条件分岐はプログラム内で特定の条件を指定し，その条件が満たされるかどうかによって異なる処理を行うための制御構造である。R 言語では `if-else` を用いて条件分岐を表現する。

5.3.1 `if` 文の基本的な構文

以下が `if` 文の基本的な構文になる：

```
if (条件) {
    # 条件が真である場合に実行するコード
}
```

`if` の後の小括弧内に条件を指定する。この条件が真 (TRU) であれば，その後の中括弧`{}` 内のコードが実行される。さらに，`else` を使用して，条件が偽（FALSE）の場合の処理を追加することもできる：

```
if (条件) {
    # 条件が真である場合に実行するコード
} else {
    # 条件が偽である場合に実行するコード
}
```

以下に具体的な使用例を示そう：

```
x <- 10

if (x > 0) {
  print("x is positive")
} else {
  print("x is not positive")
}
```

```
[1] "x is positive"
```

このコードでは，変数 `x` が正の場合とそうでない場合で異なるメッセージを出力する。

条件は論理式（例：`x > 0`, `y == 1`）や論理値（TRUE/FALSE）を返す関数・操作（例：`is.numeric(x)`）などで指定する。また，複数の条件を組み合わせる際には論理演算子 (`&&`, `||`) を使用する。

この例では，`x` が正と `y` が負の場合に特定のメッセージを出力する。それ以外の場合は，「Other case」と出力される。`x` や `y` の値を色々変えて，試してみて欲しい。

```
x <- 10
y <- -3

if (x > 0 && y < 0) {
  print("x is positive and y is negative")
} else {
  print("Other case")
}
```

```
[1] "x is positive and y is negative"
```

5.4　反復と条件分岐に関する練習問題

1. 1 から 20 までの数字で，偶数だけをプリントするプログラムを書いてください。
2. 1 から 40 までの数値をプリントするプログラムを書いてください。ただしその数値に 3 がつく (1 か 10 の位の値が 3 である) か，3 の倍数の時だけ，数字の後ろに「サァン！」という文字列をつけて出力してください。
3. ベクトル `c(1, -2, 3, -4, 5)` の各要素について，正なら “positive”，負なら “negative” をプリントするプログラムを書いてください。
4. 次の行列 A と B の掛け算を計算するプログラムを書いてください。なお，R で行列の積は `%*%` という演算子を使いますが，ここでは `for` 文を使ったプログラムにしてください。出来上がる行列の i 行 j 列目の要素 c_{ij} は，行列 A の第 i 行の各要素と，行列 B の第 j 列目の各要素の積和，すなわち

$$c_{ij} = \sum_k a_{ik} b_{kj}$$

   になります。検算用のコードを下に示します。

```
A <- matrix(1:6, nrow = 3)
B <- matrix(3:10, nrow = 2)
## 課題になる行列
print(A)
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```
print(B)
```

```
     [,1] [,2] [,3] [,4]
[1,]    3    5    7    9
[2,]    4    6    8   10
```

```
## 求めるべき答え
C <- A %*% B
print(C)
```

```
     [,1] [,2] [,3] [,4]
[1,]   19   29   39   49
[2,]   26   40   54   68
[3,]   33   51   69   87
```

5.5 関数を作る

複雑なプログラムも，ここまでの代入、反復，条件分岐の組み合わせからなる。回帰分析や因子分析のような統計モデルを実行するときに，統計パッケージのユーザとしては，統計モデルを実現してくれる関数にデータを与えて答えを受け取るだけであるが，そのアルゴリズムはこれらプログラミングのピースを紡いでつくられているのである。

ここでは関数を自分で作ることを考える。といっても身構える必要はない。表計算ソフトウェアで同じような操作を繰り返すときにマクロに記録するように，R 上で同じようなコードを何度も書く機会があるならば，それを関数という名のパッケージにしておこう，ということである。関数化しておくことで手続きをまとめることができ，小単位に分割できるため並列して開発したり，バグを見つけやすくなるという利点がある。

5.5.1 基本的な関数の作り方

関数が受け取る値のことを**引数** (ひきすう，argument) といい，また関数が返す値のことを**戻り値** (もどりち,value) という。$y = f(x)$ という式は，引数が x で戻り値が y な関数 f，と言い換えることができるだろう。

R の関数を書く基本的な構文は以下のようになる。

```
function_name <- function(argument) {
   # function body
   return(value)
}
```

ここで function body とあるのは計算本体である。例えば与えられた数字に 3 を足して返す関数，add3 を作ってみよう。プログラムは以下のようになる。

```
add3 <- function(x) {
  x <- x + 3
  return(x)
}
# 実行例
add3(5)
```

```
[1] 8
```

また，2 つの値を足し合わせる関数は次のようになる。

```
add_numbers <- function(a, b) {
  sum <- a + b
  return(sum)
}
# 実行例
add_numbers(2, 5)
```

```
[1] 7
```

ここで示したように，引数は複数取ることも可能である。また，既定値 default value を設定することも可能である。次の例を見てみよう。

```
add_numbers2 <- function(a, b = 1) {
  sum <- a + b
  return(sum)
}
# 実行例
add_numbers2(2, 5)
```

```
[1] 7
```

```
add_numbers2(4)
```

```
[1] 5
```

関数を作るときに，(a,b=1) としているのは，b に既定値として 1 を与えていて，特に指定がなければこの値を使うよう指示しているということである。実行例において，引数が 2 つ与えられている場合はそれらを使った計算をし (2+5)，1 つしか与えられていない場合は第一引数 a に与えられた値を，第二引数 b は既定値を使った計算をする (4+1)，という挙動になる。

ここから推察できるように，われわれユーザが使う統計パッケージの関数にも実は多くの引数があり，既定値が与えられているということだ。これらは選択的に，あるいは能動的に与えることができるものであるが，これらの引数は選択的に指定することができるのだが，通常は一般的に使われる値や計算の細かな設定に関するものであり，開発者がユーザの手間を省くために提供しているものである。関数のヘルプを見ると指定可能な引数の一覧が表示されるので，ぜひ興味を持って見てもらいたい。

5.5.2　複数の戻り値

R での戻り値は 1 つのオブジェクトでなければならない。しかし，複数の値を返したいということがあるだろう。そのような場合は，返すオブジェクトを list などでひとまとめにして作成すると良い。以下に簡単な例を示す。

```r
calculate_values <- function(a, b) {
  sum <- a + b
  diff <- a - b
  # 戻り値として名前付きリストを作成
  result <- list("sum" = sum, "diff" = diff)
  return(result)
}
# 実行例
result <- calculate_values(10, 5)
# 結果を表示
print(result)
```

```
$sum
[1] 15

$diff
[1] 5
```

5.6 課題

1. ある値を与えたとき，正の値なら”positive”，負の値なら”negative”，0 のときは”Zero” と表示する関数を書いてください。
2. ある 2 組の数字を与えた時，和，差，積，商を返す関数を書いてください。
3. あるベクトルを与えた時，算術平均，中央値，最大値，最小値，範囲を返す関数を書いてください。
4. あるベクトルを与えた時，標本分散を返す関数を書いてください。なお R の分散を返す関数 `var` は不偏分散 $\hat{\sigma}$ を返しており，標本分散 v とは計算式が異なります。念のため，計算式を以下に示します。

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

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

第 6 章

確率とシミュレーション

6.1 確率の考え方と使い所

統計と確率は密接な関係がある。まずデータをたくさん集めると，個々のケースでは見られない全体的な傾向が見られるようになり，それを表現するのに確率の考え方を使う，というのがひとつ。次にデータがそれほどたくさんなくとも，大きな全体の中から一部を取り出した標本 Sample と考えられるとき，標本は全体の性質をどのように反映しているかを考えることになる。ここで全体の傾向から一部を取り出した偶然性を表現するときに確率の考え方を使うことになる。最後に，理論的・原理的に挙動がわかっている機械のようなものでも，現実的・実践的には系統だったズレが生じたり，偶然としか考えられない誤差が紛れ込むことがある。前者は機械の調整で対応できるが，後者は偶然が従う確率を考える必要がある。

心理学は人間を対象に研究を行うが，あらゆる人間を一度に調べるわけにはいかないので，サンプルを取り出して調査したり実験したりする (第 2 のケース)。データサイエンスでは何万レコードというおおきなデータセットになるが，心理学の場合は数件から数十件しかないことも多い。また，心理学的傾向を理論立ててモデル化できたとしても，実際の行動には誤差が含まれている可能性が高い (第 3 のケース)。このことから，心理学で得られるデータは確率変数として考えられ，小標本から母集団の性質を推測する**推測統計**と共に利用される。

厳密に数学的な意味での**確率**は，集合，積分，測度といった緻密な概念の積み重ねから定義される[*1]。ここではその詳細に分け入らず，単に「特定の結果が生じる可能性について，0 から 1 の間の実数でその大小を表現したもの」とだけ理解しておいて欲しい。この定義からは，「全ての可能な組み合わせのうち当該事象の成立する割合」という解釈も成り立つし，「主観的に重みづけた真実味の強さに関する信念の度合い」という解釈も成り立つ。[*2]これまで学んできた確率は順列・組み合わせを全て書き出す退屈なもの，と思っていたかもしれないが，「十中八九まちがいないね (80-90% ほど確からしいと考えている)」という数字も確率の一種として扱えるので，非常に身近で適用範囲の広い概念である。理解を進めるポイントの 1 つとして，確率を面積として考えると良いかもしれない。ありうる状況の全体の空間に対して，事象の成立する程度がどの程度の面積がどの程度の割合であるかを表現したのが確率という量である，と考えるのである (平岡・堀 (2009) は書籍の中で一貫して面積で説明している。

[*1] 詳しくは吉田 (2021) , 河野 (1999), 佐藤 (1994) などを参照のこと。

[*2] 前者の解釈は高校までの数学で学ぶ確率であり，頻度主義的確率と呼ばれることがある。一方後者の解釈は，降水確率 X% のように日常でも使うものであり，主観確率と呼ばれることがある。こうした解釈の違いを，主義主張の対立であって数学的ではない，と批判する向きもあるが，実際コルモゴロフの公理はどちらの立場でも成立するように整えられており，筆者個人的にはユーザが理解しやすく計算できればどちらでも良いと考えている。

この説明だと，条件付き確率などの理解がしやすい。)。

ただし注意して区別しておいて欲しいのが，確率変数とその実現値の違いである。データセットやスプレッドシートに含まれる値は，あくまでも**確率変数の実現値**というのであって，**確率変数**はその不確実な状態を有した変数そのものを指す言葉である。サイコロは確率変数だが，サイコロの出目は確率変数の実現値である。心理変数は確率変数だが，手に入れたデータはその実現値である。実現値を通じて変数の特徴を知り，全体を推測するという流れである。

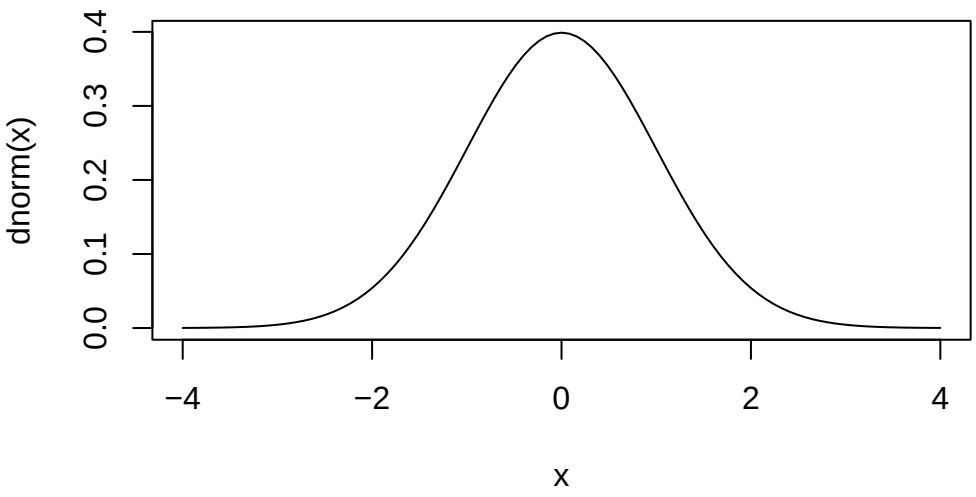
目の前のデータを超えて，抽象的な実体で議論を進めることが難しく感じられるかもしれない。実は誰しもそうなのであって，確率の正確な理解は非常に難易度が高い。しかし R など計算機言語に実装されている関数を通じて，より具体的に，操作しながら理解することで徐々に理解していこう。

6.2　確率分布の関数

確率変数の実現値は，**確率分布**に従う。確率分布とは，その実現値がどの程度生じやすいかを全て表した総覧であり，一般的に関数で表現される。実現値が連続的か離散的かによって名称が異なるが，連続的な確率分布関数は**確率密度関数 (Probability Density Function)**，離散的な確率分布関数は**確率質量関数 (Probability Mass Function)** という。

R には最初から確率に関する関数がいくつか準備されている。最も有名な確率分布である**正規分布**について，次のような関数がある。

```
# 標準のプロット関数，curve
curve(dnorm(x), from = -4, to = 4)
```

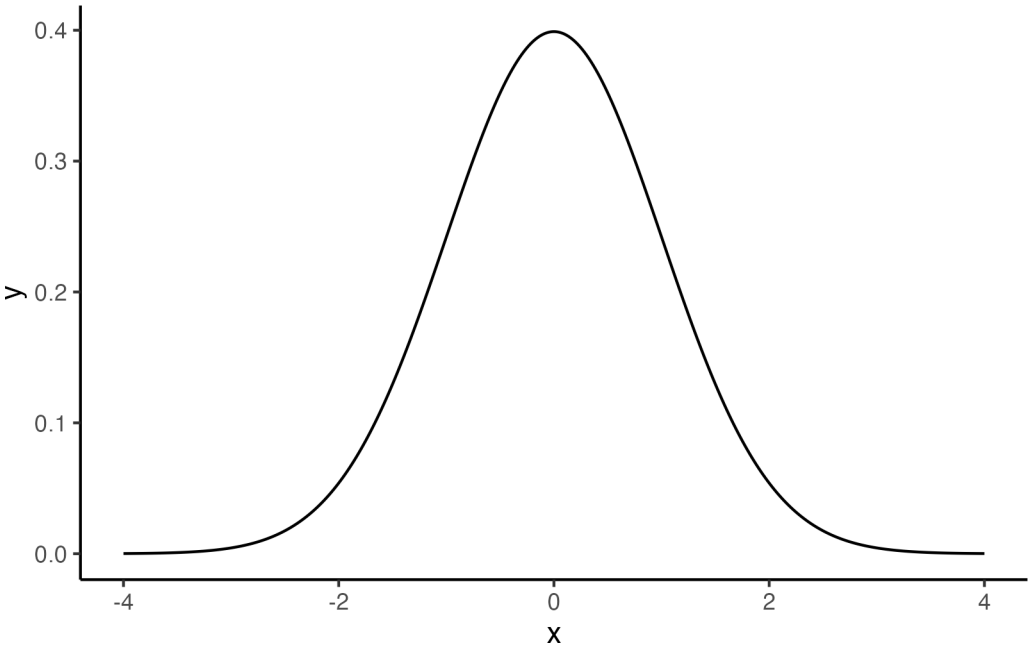



```
# ggplot2 を使ってカッコよく
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
```

```
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
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
```

```
data.frame(x = seq(-4, 4, by = 0.01)) %>%
  mutate(y = dnorm(x)) %>%
  ggplot(aes(x = x, y = y)) +
  geom_line() +
  theme_classic()
```

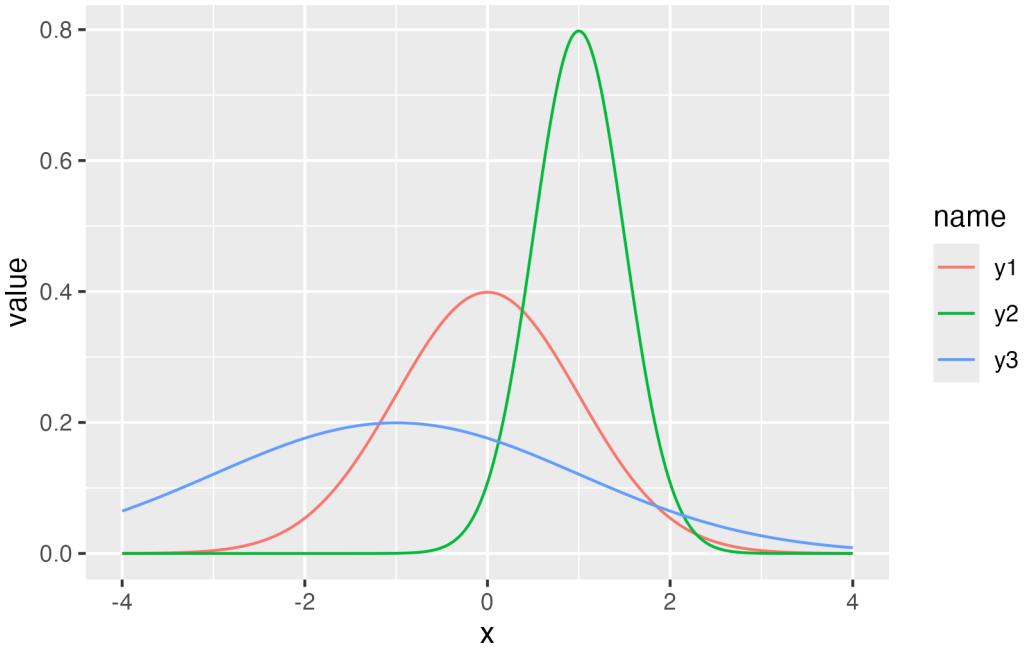


ここで dnorm という関数を使っているが，d は Density(確率密度) の頭文字であり，norm は Normal Distribution(正規分布) の一部である。このように，R では確率分布の名前を表す名称 (ここでは norm) と，それに接頭文字ひとつ (d) で関数を構成する。この接頭文字は他に p,q,r があり，dpois(ポアソン分布 poisson distribution の確率密度関数)，pnorm(正規分布 normal distribution の累積分布関数),rbinom(二項分布 binomial distribution からの乱数生成) のように使う。

ここでは正規分布を例に説明を続けよう。正規分布は平均 μ と標準偏差 σ でその形状が特徴づけられる。これらの確率分布の特徴を表す数字のことを**母数 parameter** という。たとえば，次の 3 つの曲線はパラメータが異なる正規分布である。

```
data.frame(x = seq(-4, 4, by = 0.01)) %>%
  mutate(
    y1 = dnorm(x, mean = 0, sd = 1),
```

```
    y2 = dnorm(x, mean = 1, sd = 0.5),
    y3 = dnorm(x, mean = -1, sd = 2)
  ) %>%
  pivot_longer(-x) %>%
  ggplot(aes(x = x, y = value, color = name)) +
  geom_line()
```



平均は位置母数，標準偏差はスケール母数とも呼ばれ，分布の位置と幅を変えていることがわかる。言い換えると，データになるべく当てはまるように正規分布の母数を定めることもできるわけで，左右対称で単峰の分布という特徴があれば，正規分布でかなり様々なパターンを表せる。

さて，上の例で用いた関数はいずれも d を頭に持つ dnorm であり，確率分布の密度の高さを表現していた。では p や q が表すのは何であろうか。数値と図の例を示すので，その対応関係を確認してもらいたい。

```
# 累積分布関数
pnorm(1.96, mean = 0, sd = 1)
```

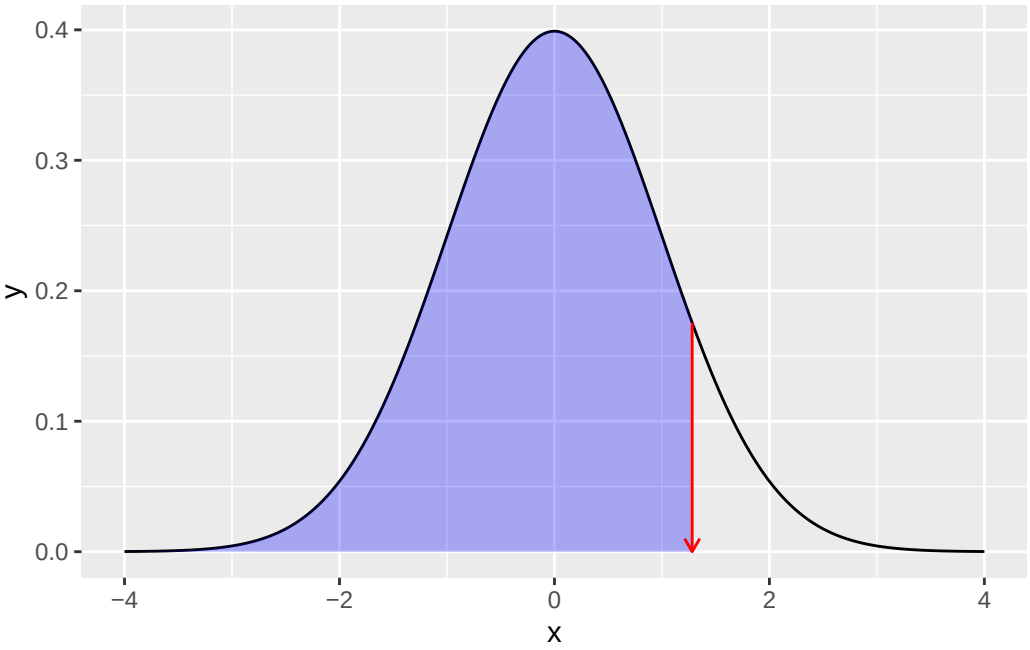
```
[1] 0.9750021
```

```
# 累積分布の逆関数
qnorm(0.975, mean = 0, sd = 1)
```

```
[1] 1.959964
```

数値で直感的にわかりにくい場合，次の図を見て確認しよう。pnorm 関数は x 座標の値を与えると，そこまでの面積 (以下のコードで描かれる色付きの領域) すなわち確率を返す。qnorm 関数は確率 (=面積) を与えると，確率密度関数のカーブの下領域を積分してその値になるときの x 座標の値を返す。

```r
# 描画
prob <- 0.9
## 全体の正規分布カーブ
df1 <- data.frame(x = seq(from = -4, 4, by = 0.01)) %>%
  mutate(y = dnorm(x, mean = 0, sd = 1))
## qnorm(0.975) までのデータ
df2 <- data.frame(x = seq(from = -4, qnorm(prob), by = 0.01)) %>%
  mutate(y = dnorm(x, mean = 0, sd = 1))
## データセットの違いに注意
ggplot() +
  geom_line(data = df1, aes(x = x, y = y)) +
  geom_ribbon(data = df2, aes(x = x, y = y, ymin = 0, ymax = y), fill = "blue", alpha = 0.3) +
  ## 以下装飾
  geom_segment(
    aes(x = qnorm(prob), y = dnorm(qnorm(prob)), xend = qnorm(prob), yend = 0),
    arrow = arrow(length = unit(0.2, "cm")), color = "red"
  )
```



d,p,q,r といった頭の文字は，他の確率分布関数にも付く。では次に r について説明しよう。

6.3 乱数

乱数とは何であるかを説明するのは，「ランダムである (確率変数である) とは如何なることか」を説明するのと同じように難しい。カンタンに説明するなら，規則性のない数列という意味である。しかし計算機はアルゴリズムに沿って正しく数値を計算するものだから，ランダムに，規則性がない数字を示すということは厳密にはあり得な

い。計算機が出す乱数は，乱数生成アルゴリズムに沿って出される数字であり，ランダムに見えて実は規則性があるので，疑似乱数というのが正しい。

とはいえ，人間が適当な数字を思いつきで誦じていく[*3]よりは，よほど規則性がない数列を出すので，疑似的とはいえ十分に役に立つ。たとえばアプリなどで「ガチャ」を引くというのは，内部で乱数によって数値を出し，それに基づいてあたり・ハズレ等の判定をしている。他にも，RPG などで攻撃する時に一定の確率で失敗するとか，一定の確率で「会心の一撃」を出すというのも同様である。ここで大事なのは，そうしたゲームへの実装において規則性のない数字に基づくプログラムにしたとしても，その統計的な性質，すなわち実現値の出現確率はある程度制御したいのである。

そこで，ある確率分布に基づく乱数を生成したい，ということになる。幸いにして，一様乱数 (全ての実現値が等しい確率で生じる) を関数で変換することで，正規分布ほか様々な確率分布に従う乱数を作ることができる。R にはその基本関数として幾つかの確率分布に従う乱数が実装されている。たとえば次のコードは，平均 50，SD10 の正規分布に従う乱数を 10 個出現させるものである。

```
rnorm(n = 10, mean = 50, sd = 10)
```

```
 [1] 52.38166 68.04342 44.56098 45.47789 60.88762 55.99496 63.58422 50.42189
 [9] 35.60008 76.97508
```

たとえば諸君が心理統計の練習問題を作ろうとして，適当な数列が欲しければこのようにすれば良いかもしれない。しかし，同じ問題をもう一度作ろうとすると，乱数なのでまた違う数字が出てしまう。

```
rnorm(n = 10, mean = 50, sd = 10)
```

```
 [1] 59.58609 47.64540 73.11588 57.74211 49.67149 40.22341 49.26134 37.06255
 [9] 55.75034 37.95547
```

疑似乱数に過ぎないのだから，再現性のある乱数を生じさせたいと思うかもしれない。そのような場合は，`set.seed` 関数を使う。疑似乱数は内部の乱数生成の種 (seed) から計算して作られているため，その数字を固定してやると同じ乱数が再現できる。

```
# seed を指定
set.seed(12345)
rnorm(n = 3)
```

```
[1]  0.5855288  0.7094660 -0.1093033
```

```
# 同じ seed を再設定
set.seed(12345)
rnorm(n = 3)
```

```
[1]  0.5855288  0.7094660 -0.1093033
```

[*3] 厳密なエビデンスは示せないが，俗に「嘘のゴサンパチ」というように人間が適当に数字を述べると 5,3,8 が使われる率がチャンスレベルより高いと言われている。

6.3.1 乱数のつかいかた

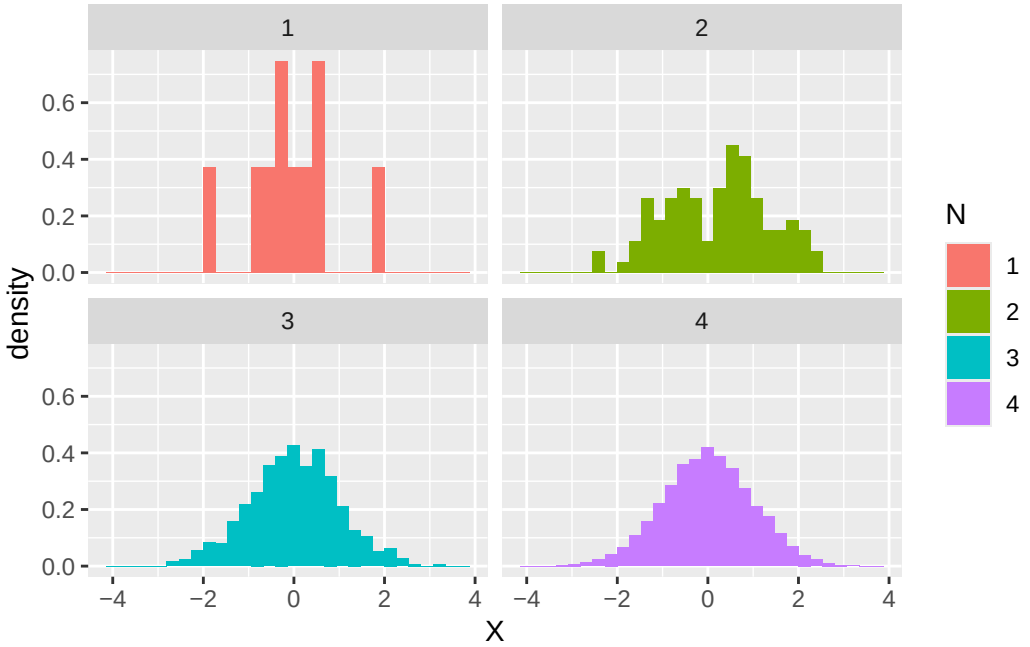
乱数の使い方のひとつは，先に述べたように，プログラムが偶然による振る舞いをしているように仕掛けたいとき，ということだろう。

実は他にも使い道がある。それは確率分布を具体的に知りたいときである。次に示すのは，標準正規分布から $n = 10, 100, 1000, 10000$ とした時のヒストグラムである。

```
rN10 <- rnorm(10)
rN100 <- rnorm(100)
rN1000 <- rnorm(1000)
rN10000 <- rnorm(10000)

data.frame(
  N = c(
    rep(1, 10), rep(2, 100),
    rep(3, 1000), rep(4, 10000)
  ),
  X = c(rN10, rN100, rN1000, rN10000)
) %>%
  mutate(N = as.factor(N)) %>%
  ggplot(aes(x = X, fill = N)) +
  # 縦軸を相対頻度に
  geom_histogram(aes(y = ..density..)) +
  facet_wrap(~N)
```

```
Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(density)` instead.
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



これを見ると，最初の 10 個程度のヒストグラムは不規則な分布に見えるが，100,1000 と増えるに従って徐々に正規分布の理論的形状に近似していくところがみて取れる。

R にはポアソン分布や二項分布などに加え，統計に馴染みの深い t 分布や F 分布，χ^2 分布などの確率分布関数も実装されている。これらの分布はパラメタの値を聞いてもイメージしにくいところがあるかもしれないが，そのような時はパラメタを指定した上で乱数を大量に生成し，そのヒストグラムを描けば確率分布関数の形が眼に見えてくるため，より具体的に理解できるだろう。

実際，ベイズ統計学が昨今隆盛している一つの理由は，計算機科学の貢献によるところが大きい。**マルコフ連鎖モンテカルロ法** (MCMC 法) と呼ばれる乱数発生技術は，明確な名前を持たないモデルによって作られる事後分布からでも，乱数を生成できる技術である。この分布は解析的に示すことは困難であるが，そこから乱数を生成し，そのヒストグラムを見ることで，形状を可視化できるのである。

また，この乱数利用法の利点は可視化だけではない。標準正規分布において，ある範囲の面積 (=確率) が知りたいとする。たとえば，確率点-1.5 から +1.5 までの範囲の面積を求めたいとしよう。正規分布の数式はわかっているので，次のようにすればその面積は求められる。

$$p = \int_{-1.5}^{+1.5} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$$

もちろん我々は pnorm 関数を知っているので，次のようにして数値解を得ることができる。

```
pnorm(+1.5, mean = 0, sd = 1) - pnorm(-1.5, mean = 0, sd = 1)
```

```
[1] 0.8663856
```

同様のことは乱数を使って，次のように近似解を得ることができる。

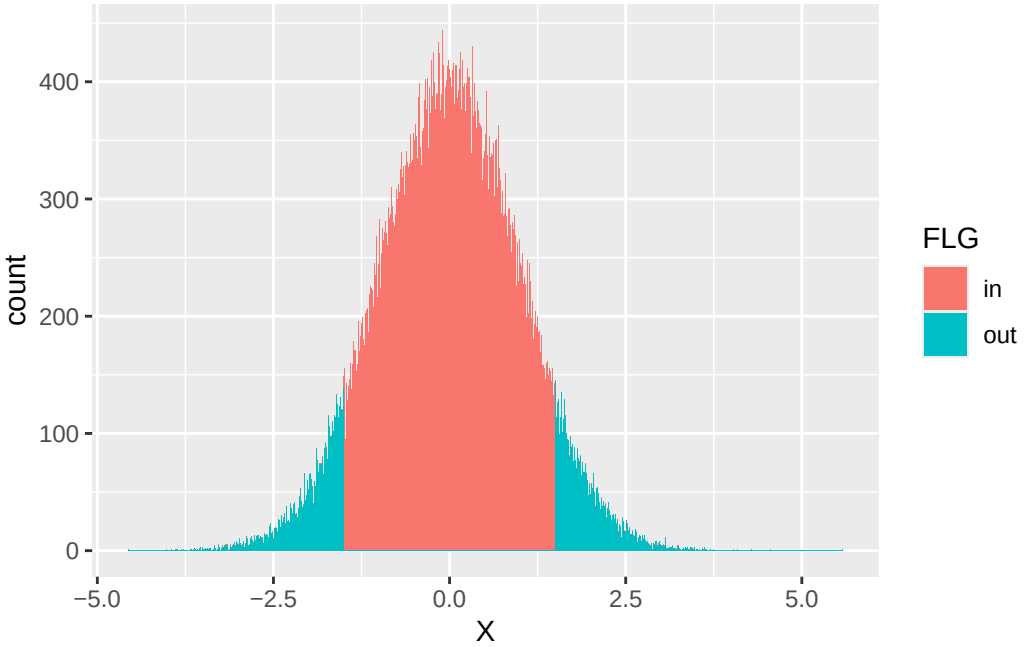
```r
x <- rnorm(100000, mean = 0, sd = 1)
df <- data.frame(X = x) %>%
  # 該当する範囲かどうかを判定する変数を作る
  mutate(FLG = ifelse(X > -1.5 & X < 1.5, 1, 2)) %>%
  mutate(FLG = factor(FLG, labels = c("in", "out")))
## 計算
df %>%
  group_by(FLG) %>%
  summarise(n = n()) %>%
  mutate(prob = n / 100000)
```

```
# A tibble: 2 x 3
  FLG       n  prob
  <fct> <int> <dbl>
1 in    86642 0.866
2 out   13358 0.134
```

ここでは乱数を 10,000 個生成し，指定の範囲内に入るかどうか (入れば 1, 入らなければ 2) を示す factor 型変数 FLG を作った。この変数ごとに群分けして数を数え，総数で割ることで相対度数にする。確率は全体の中に占める相対的な面積の割合であり，今回当該領域の値が 0.866 と pnorm 関数で算出した解とほぼ同等の値変えられている。

なお，次のようにすれば範囲の可視化も容易い。

```r
## 可視化
df %>%
  ggplot(aes(x = X, fill = FLG)) +
  geom_histogram(binwidth = 0.01)
```



繰り返すが，確率分布の形がイメージできなかったり，解析的にその式を書き表すことが困難であった場合でも，具体的な数値にすることでヒストグラムで可視化でき，また近似的に確率計算ができている。

あくまでも近似に過ぎないのでその精度が信用できない，というひとは生成する乱数の数を 10 倍，100 倍にすれば良い。昨今の計算機の計算能力において，その程度の増加はさほど計算料の負担にならない。複雑な積分計算が記述統計量 (数え上げ) の問題になる点で，具体的に理解できるという利点は大きい。

さらに思いを馳せてほしいのだが，心理学者は心理学実験や調査によって，データを得る。しかしそれらは個人差や誤差を考え，確率変数だとされている。目の前の数件から数十件のデータであっても，正規分布に従うと仮定して統計的処理をおこなう。これは「乱数によって生成したデータ」に対して行うとしても本質的には同じである。すなわち，調査実験を行う前に，乱数によってシミュレーションしておくことができるのである。調査実験の本番一発勝負をする前に，自分の取ろうとしているデータがどのような性質を持ちうるかを具体的に確かめておくことは重要な試みであろう。

6.4　練習問題；乱数を用いて

正規乱数を用いて，次の値を近似計算してみよう。なお設定や解析的に算出した「真の値」と少数以下 2 位までの精度が得られるように工夫しよう。

1. 平均 100, 標準偏差 8 の正規分布の期待値。なお連続確率変数の期待値は次の式で表されます。

$$E[X] = \int_{-\infty}^{\infty} x f(x)\, dx$$

ここで x は確率変数を表し，$f(x)$ は確率密度関数であり，確率密度関数の全定義域を積分することで得られます。正規分布の期待値は，平均パラメータに一致しますので，今回の真値は設定した 100 になります。

2. 平均 100, 標準偏差 3 の正規分布の分散を計算してみよう。なお連続確率変数の分散は次の式で表されます。

$$\sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)\, dx$$

ここで μ は確率変数の期待値であり，正規分布の分散は，標準偏差パラメータの二乗に一致しますので，今回の真値は $3^2 = 9$ です。

3. 平均 65，標準偏差 10 の正規分布に従う確率変数 X の，$90 < X < 110$ の面積。解析的に計算した結果は次の通りです。

```
pnorm(110, mean = 65, sd = 10) - pnorm(90, mean = 65, sd = 10)
```

```
[1] 0.006206268
```

4. 平均 10，標準偏差 10 の正規分布において，実現値が 7 以上になる確率。解析的に計算した結果は次の通りです。

```
1 - pnorm(7, mean = 10, sd = 10)
```

```
[1] 0.6179114
```

5. 確率変数 X, Yがあります。X は平均 10,SD10 の正規分布，Yは平均 5，SD8 の正規分布に従うものとします。ここで，X と Yが独立であるとしたとき，和 $Z = X + Y$の平均と分散が，もとの X, Yの平均の和，分散の和になっていることを，乱数を使って確認してください。

6.5 母集団と標本

ここまで確率分布の性質を見るために乱数を利用する方法を見てきた。ここからは，推測統計学における確率分布の利用を考える。推測統計では，知りたい集団全体のことを**母集団 population**，そこから得られた一部のデータを**標本 sample** と呼ぶのであった。標本の統計量を使って，母集団の性質を推論するのが推測統計/統計的推測である。母集団の特徴を表す統計量は**母数 parameter** と呼ばれ，母平均，母分散など「母」の字をつけて母集団の情報であることを示す。同様に，標本の平均や分散も計算できるが，この時は標本平均，標本分散など「標本」をつけて明示的に違いを強調することもある。

乱数を使って具体的な例で見てみよう。ここに 100 人から構成される村があったとする。この村の人々の身長を測ってデータにしたとしよう。100 個の適当な数字を考えるのは面倒なので，乱数で生成してこれに代える。

```
set.seed(12345)
# 100 人分の身長データをつくる。小数点以下 2 桁を丸めた
Po <- rnorm(100, mean = 150, sd = 10) %>% round(2)
print(Po)
```

```
  [1] 155.86 157.09 148.91 145.47 156.06 131.82 156.30 147.24 147.16 140.81
 [11] 148.84 168.17 153.71 155.20 142.49 158.17 141.14 146.68 161.21 152.99
 [21] 157.80 164.56 143.56 134.47 134.02 168.05 145.18 156.20 156.12 148.38
 [31] 158.12 171.97 170.49 166.32 152.54 154.91 146.76 133.38 167.68 150.26
```

```
 [41] 161.29 126.20 139.40 159.37 158.54 164.61 135.87 155.67 155.83 136.93
 [51] 144.60 169.48 150.54 153.52 143.29 152.78 156.91 158.24 171.45 126.53
 [61] 151.50 136.57 155.53 165.90 144.13 131.68 158.88 165.93 155.17 137.04
 [71] 150.55 142.15 139.51 173.31 164.03 159.43 158.26 141.88 154.76 160.21
 [81] 156.45 160.43 146.96 174.77 159.71 168.67 156.72 146.92 155.37 158.25
 [91] 140.36 141.45 168.87 146.08 140.19 156.87 144.95 171.58 144.00 143.05
```

この 100 人の村が母集団なので，母平均や母分散は次のようにして計算できる。

```
M <- mean(Po)
V <- mean((Po - M)^2)
# 母平均
print(M)
```

```
[1] 152.4521
```

```
# 母分散
print(V)
```

```
[1] 123.0206
```

さて，この村からランダムに 10 人の標本を得たとしよう。ベクトルの前から 10 人でも良いが，R にはサンプリングをする関数 sample があるのでこれを活用する。

```
s1 <- sample(Po, size = 10)
s1
```

```
 [1] 164.61 155.86 136.93 143.29 160.43 168.87 151.50 155.17 153.71 135.87
```

この s1 が手元のデータである。心理学の実験でデータを得る，というのはこのように全体に対してごく一部だけ取り出したものになる。このサンプルの平均や分散は標本平均，標本分散である。

```
m1 <- mean(s1)
v1 <- mean((s1 - mean(s1))^2)
# 標本平均
print(m1)
```

```
[1] 152.624
```

```
# 標本分散
print(v1)
```

```
[1] 110.2049
```

今回，母平均は 152.4521 で標本平均は 152.624 である。実際に知りうる値は標本の値だけなので，標本平均 152.624 を得たら，母平均も 152.624 に近い値だろうな，と推測するのはおかしなことではないだろう。しかし標本平均は，標本の取り方によって毎回変わるものである。試しにもう一つ，標本をとったとしよう。

```
s2 <- sample(Po, size = 10)
s2
```

```
 [1] 154.76 135.87 143.05 171.45 136.57 170.49 156.87 158.25 155.17 155.20
```

```
m2 <- mean(s2)
v2 <- mean((s2 - mean(s2))^2)
# 標本平均その 2
print(m2)
```

```
[1] 153.768
```

今回の標本平均は 153.768 になった。このデータが得られたら，諸君は母平均が「153.768 に近い値だろうな」と推測するに違いない。標本 1 の 152.624 と標本 2 の 153.768 を比べると，前者の方が正解 152.4521 に近い (その差はそれぞれ-0.1719 と-1.3159 である)。つまり，標本の取り方によっては当たり外れがあるということである。データをとって研究していても，仮説を支持する結果なのかそうでないのかは，こうした確率的揺らぎの下にある。

つまり，**標本は確率変数であり，標本統計量も確率的に変わりうる**ものである。標本統計量でもって母数を推定するときは，標本統計量の性質や標本統計量が従う確率分布を知っておく必要がある。以下では母数の推定に望ましい性質を持つ推定量の望ましい性質をみていこう。

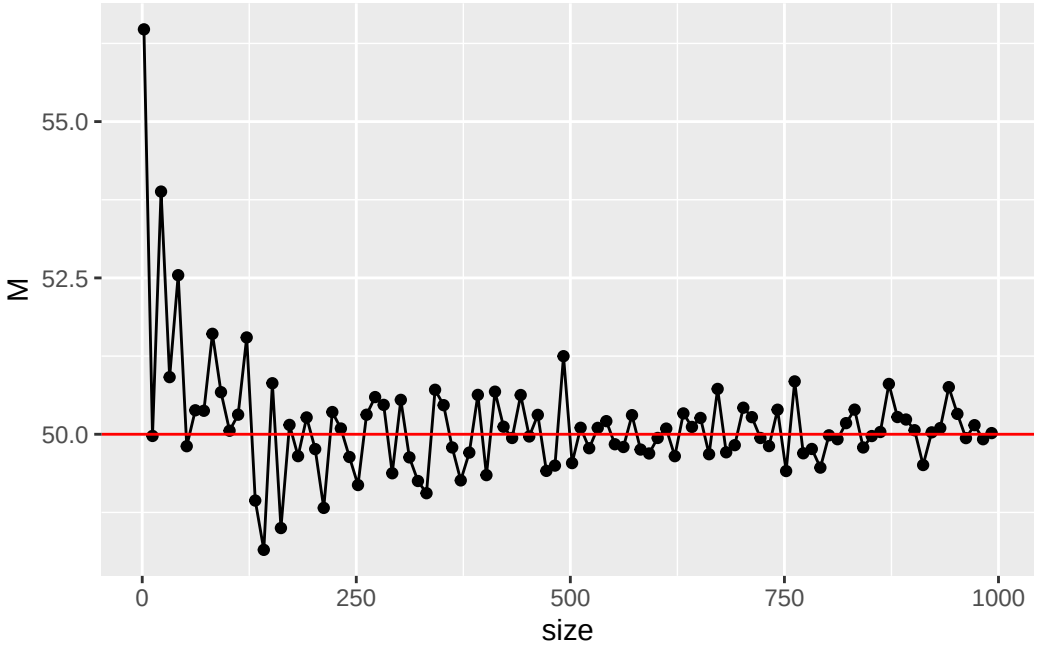
6.6 一致性

最も単純には，標本統計量が母数に近ければ近いほど，できれば一致してくれれば喜ばしい。先ほどの例では 100 人の村から 10 人しか取り出さなかったが，もし 20 人，30 人とサンプルサイズが大きくなると母数に近づいていくことが予想できる。この性質のことを**一致性** consistency といい，推定量が持っていてほしい性質のひとつである。幸い，標本平均は母平均に対して一致性を持っている。

このことを確認してみよう。サンプルサイズを様々に変えて計算してみれば良い。例として，平均 50,SD10 の正規分布からサンプルサイズを 2 から 1000 まで増やしていくことにしよう。サンプルを取り出すことを，乱数生成に置き換えてその平均を計算していくこととする。

```
set.seed(12345)
sample_size <- seq(from = 2, to = 1000, by = 10)
# 平均値を格納するオブジェクトを初期化
sample_mean <- rep(0, length(sample_size))
# 反復
for (i in 1:length(sample_size)) {
  sample_mean[i] <- rnorm(sample_size[i], mean = 50, sd = 10) %>%
    mean()
}
```

```
# 可視化
data.frame(size = sample_size, M = sample_mean) %>%
  ggplot(aes(x = size, y = M)) +
  geom_point() +
  geom_line() +
  geom_hline(yintercept = 50, color = "red")
```



このようにサンプルサイズが増えていくにつれて，真値の 50 に近づいていくことが見て取れる。母集団分布の形状やパラメータ，サンプルサイズなどを変えて確認してみよう。

6.7　不偏性

推定量は確率変数であり，確率分布でその性質を記述することができる。標本統計量の従う確率分布のことを**標本分布**と呼ぶが，標本分布の確率密度関数がわかっているなら，その期待値や分散も計算できるだろう。推定量の期待値 (平均) が母数に一致することも，推定量の望ましい性質の一つであり，この性質のことを**不偏性** unbiasedness という。

心理統計を学ぶ時に初学者を苛立たせるステップの一つとして，分散の計算の時にサンプルサイズ n ではなく $n-1$ で割る，という操作がある。これは不偏分散といって標本分散とは違うのだが，前者が不偏性を持っているのに対し，後者がそうでないからである。これを乱数を使って確認してみよう。

平均 50，SD10(母分散 $10^2 = 100$) の母集団から，サンプルサイズ $n = 20$ の標本を繰り返し得る。これはサイズ 20 の乱数生成で行う。各標本に対して標本分散と不偏分散を計算し，その平均 (標本統計量の期待値) を計算してみよう。

```
iter <- 5000
vars <- rep(0, iter)
unbiased_vars <- rep(0, iter)

## 乱数の生成と計算
set.seed(12345)
for (i in 1:iter) {
  sample <- rnorm(n = 20, mean = 50, sd = 10)
  vars[i] <- mean((sample - mean(sample))^2)
  unbiased_vars[i] <- var(sample)
}

## 期待値
mean(vars)
```

```
[1] 95.08531
```

```
mean(unbiased_vars)
```

```
[1] 100.0898
```

標本分散を計算したオブジェクト vars の平均すなわち期待値は 95.0853144 であり，設定した値 (真値) の 100 からは幾分はなれている。これに対して，R の埋め込み関数である var をつかった不偏分散の平均すなわち期待値は 100.0898047 であり，母分散の推定量としてはこちらの方が好ましいことがわかる。このように標本分散にはバイアスが生じることがわかっているので，あらかじめバイアスを補正するために元の計算式を修正していたのである。この説明で，苛立ちを感じていた人の溜飲が下がればよいのだが。

他にも推定量の望ましい性質として有効性 efficacy があるが，詳細は小杉他 (2023) を参照してほしい。この本には正規分布以外の例や，相関係数など他の標本統計量の例なども載っているが，いずれも乱数生成による近似で理解を進めるものである。諸君も数理統計的な説明に疲れたなら，ぜひ参考にしてもらいたい。

6.8 信頼区間

標本統計量は確率変数であり，標本を取るたびに変わる。標本を取るときに入る確率的ゆらぎによるからで，標本平均は一致性，不偏性という望ましい性質を持ってはいるが，標本平均 = 母平均とはならない。

標本平均という確率変数の実現値一点でもって，母平均を推測することは，母平均を推測する上ではほぼ確実に外れるギャンブルである。そこで母数に対してある幅でもって推定することを考えよう。

たとえば平均 50，標準偏差 10 の標準正規分布を母集団分布とし，サンプルサイズ 10 の標本をとり，その標本平均を母平均の推定値としよう (点推定)。同時に，その推定値に少し幅を持たせ，たとえば標本平均 ± 5 の**区間推定**をしたとする。この時，真値 0 を正しく推測できる確率を，反復乱数生成のシミュレーションで確かめてみよう。

```r
iter <- 10000
n <- 10
mu <- 50
SD <- 10

# 平均値を格納しておくオブジェクト
m <- rep(0, iter)

set.seed(12345)
for (i in 1:iter) {
  # サンプリングし，標本統計量を保存
  sample <- rnorm(n, mean = mu, sd = SD)
  m[i] <- mean(sample)
}

result.df <- data.frame(m = m) %>%
  # 推定が一致すると TRUE，外れると FALSE になる変数を作る
  mutate(
    point_estimation = ifelse(m == mu, TRUE, FALSE),
    interval_estimation = ifelse(m - 5 <= mu & mu <= m + 5, TRUE, FALSE)
  ) %>%
  summarise(
    n1 = sum(point_estimation),
    n2 = sum(interval_estimation),
    prob1 = mean(point_estimation),
    prob2 = mean(interval_estimation)
  ) %>%
  print()
```

```
  n1   n2 prob1 prob2
1  0 8880     0 0.888
```

結果からわかるように，点推定値は一度も正しく母数を当てていない。これは当然で，実数でやる以上小数点以下どこかでズレてしまうことがあるからで，精度を無視すると一致することはあり得ないのである。これに対して幅を持った予測の場合は，10^4 回の試行のうち 8880 回はその区間内に真値を含んでおり，その正答率は 88.8% である。

区間推定において正答率を 100% にするためには，その区間を無限に広げなければならない (母平均の推定の場合)。これは実質的に何も推定していないことに等しいので，5% 程度の失敗を認めよう，95% の正答率で区間推定しようというのが習わしになっている。この区間のことを 95% の**信頼区間** confidence interval という。

6.8.1 正規母集団分布の母分散が明らかな場合の信頼区間

上のシミュレーションを応用して，区間推定が正当する確率が 95% になるまで区間を調整して行ってもよいが，さすがにそれは面倒なので，推測統計学によって明らかになっている性質を紹介しよう。

母集団が正規分布に従い，その母平均が μ，母分散が σ^2 であることがわかっている場合，標本平均の従う分布は平均 μ, 分散 $\frac{\sigma^2}{n}$(標準偏差 $\frac{\sigma}{\sqrt{n}}$) の正規分布であることがわかっている。

標準正規分布の 95% 区間は，次の通り約 ± 1.96 である。

```
# 両端から 2.5% ずつ取り除くと
qnorm(0.025)
```

```
[1] -1.959964
```

```
qnorm(0.975)
```

```
[1] 1.959964
```

これらを合わせると，標本平均が $\bar{X}$ であったとき，95% 信頼区間は標準偏差を 1.96 倍して，次のようになる。

$$\bar{X} - 1.96\frac{\sigma}{\sqrt{n}} \leq \mu \leq \bar{X} + 1.96\frac{\sigma}{\sqrt{n}}$$

先ほどの数値例を応用して，これを確かめてみよう。95 %ちかい割合で，区間内に真値が含んでいることがわかる。

```
interval <- 1.96 * SD / sqrt(n)
result.df2 <- data.frame(m = m) %>%
  # 推定が一致すると TRUE, 外れると FALSE になる変数を作る
  mutate(
    interval_estimation = ifelse(m - interval <= mu & mu <= m + interval, TRUE, FALSE)
  ) %>%
  summarise(
    prob = mean(interval_estimation)
  ) %>%
  print()
```

```
    prob
1 0.9498
```

6.8.2　正規母集団分布の母分散が不明な場合の信頼区間

先ほどの例では母分散がわかっている場合の例であったが，母平均や母分散がわかっていれば推測する必要はないわけで，実践的には母分散がわからない場合の推定が必要になってくる。幸いにしてそのような場合，すなわち母分散を不偏分散 (標本統計量) で置き換えた場合は，標本平均が自由度 $n-1$ の t 分布に従うことがわかっている。(詳細は小杉他 (2023) を参照) ただその場合，標準正規分布のように 95% 区間が ± 1.96 に限らず，サンプルサイズに応じて t 分布の形が変わるから，それを考慮して以下の式で信頼区間を算出する。

$$\bar{X} + T_{0.025}\frac{U}{\sqrt{n}} \leq \mu \leq \bar{X} + T_{0.975}\frac{U}{\sqrt{n}}$$

ここで $T_{0.025}$ は t 分布の 2.5 パーセンタイル，$T_{0.975}$ は 97.5 パーセンタイルを指す。t 分布は (平均が 0 であれば) 左右対称なので，$T_{0.025} = -T_{0.975}$ と考えても良い。また U^2 は不偏分散である (U はその平方根)。

これも乱数による近似計算で確認しておこう。同じく 95 %ちかい割合で，区間内に真値が含まれていることがわかる。

```
# シミュレーションの設定
iter <- 10000
n <- 10
mu <- 50
SD <- 10

# 平均値を格納しておくオブジェクト
m <- rep(0, iter)
interval <- rep(0, iter)

set.seed(12345)
for (i in 1:iter) {
  # サンプリングし，標本統計量を保存
  sample <- rnorm(n, mean = mu, sd = SD)
  m[i] <- mean(sample)
  U <- sqrt(var(sample)) # sd(sample) でも同じ
  interval[i] <- qt(p = 0.975, df = n - 1) * U / sqrt(n)
}

result.df <- data.frame(m = m, interval = interval) %>%
  # 推定が一致すると TRUE，外れると FALSE になる変数を作る
  mutate(
    interval_estimation = ifelse(m - interval <= mu & mu <= m + interval, TRUE, FALSE)
  ) %>%
  summarise(
```

```
    prob = mean(interval_estimation)
  ) %>%
  print()
```

```
    prob
1 0.9482
```

6.9 課題

1. 算術平均 $M = \frac{1}{n}\sum x_i$ が一致推定量であることが示されましたが，調和平均 $HM = \frac{n}{\sum \frac{1}{x_i}}$ や幾何平均 $GM = (\prod x_i)^{\frac{1}{n}} = \exp(\frac{1}{n}\sum \log(x_i)))$ はどうでしょうか。シミュレーションで確かめてみましょう。
2. サンプルサイズ n が大きくなるほど，標本平均が母平均に近づくという性質は正規分布以外でも成立するでしょうか。自由度 $\nu = 3$ の t 分布を使って，シミュレーションで確認してみましょう。なお t 分布の乱数は `rt()` で生成でき，非心度パラメータ `ncp` を指定しなければその平均は 0 です。
3. t 分布の自由度 ν が極めて大きい時は，標準正規分布に一致することがわかっています。`rt()` 関数を使って自由度が 10,50,100 のときの乱数を 1000 個生成し，ヒストグラムを書いてその形状を確認しましょう。また乱数の平均と標本標準偏差を計算し，標準正規分布に近づくことを確認しましょう。
4. 平均が 50，標準偏差が 10 の正規分布からサンプルサイズ 20 の乱数を 10000 個生成し，quantile 関数をつかって 95 %信頼区間をシミュレートしてください。理論値と比較して確認してください。
5. 平均が 100，標準偏差が 15 の正規分布から抽出された標本について，サンプルサイズを 10，100，1000 と変えたときの標本平均の 95% 信頼区間の幅を比較してください。

第 7 章

統計的仮説検定 (Null Hypothesis Statistical Testing)

帰無仮説検定は，心理学における統計の利用シーンの代表的なものだろう。その手順は形式化されており，統計パッケージによってはデータの種類を指定するだけで自動的に結果の記述までしてくれるものもあるほどである。誰がやっても同じ結果になり，また，機械的に手続きを自動化できることは大きな利点ではある。欠点は，初学者がそのメカニズムを十分に理解せずに誤った結果を得たり，悪意のある利用者が自分に都合の良い数字を出させたりすることにある。科学的営みは悪意をもった実践者を想定しておらず，もしそのような悪例が露見した場合には事後的に摘発・対処するしかない。しかし残念なことながら，初学者の浅慮や意図せぬ悪用も多くみられる。

心理学において，先行研究の結果が再現しないことを再現性問題というが，そのひとつは統計的手法の誤った使い方にあるとされる (池田・平石, 2016)。改めて，丁寧に帰無仮説検定の手続きやロジックを見ていくことにしよう。

7.1 帰無仮説検定の理屈と手続き

7.1.1 帰無仮説検定の目的

帰無仮説検定は，実験や調査で得たデータから得られた知見が意味のあるものかどうか，母集団の性質として一般化可能かどうかを判定するための枠組みである。手法と判断基準が明確なゲームの一種だと考えたよう。というのも，帰無仮説検定は**有意水準**という基準を設けて，**帰無仮説**と**対立仮説**という 2 つの考え方 (モデル) を対決させ，勝敗を決するものだからである。勝敗を決するとしたのは，帰無仮説と対立仮説は排他的な関係にあるからであり，どちらも正しいとかどちらも間違っているという結末にはならないからである。ただし，あくまでも推測統計的なロジックに基づく判定であるから，判定結果にも確率的な要素が含まれる。本当は帰無仮説が正しい時に，間違って「対立仮説が正しい」と判定してしまう確率はゼロではない。逆に帰無仮説が正しくない時に，間違って「対立仮説が正しくない (帰無仮説が正しい)」と判定してしまう可能性もある。前者を**タイプ 1 エラー**，後者を**タイプ 2 エラー**という。どちらの確率もゼロであってほしいが，そうはならないので，前者を α，後者を β としたときに，それぞれを一定の水準以下に抑えたい。この目的のために手順を整え，一般化したのが帰無仮説検定である。なお，先に述べた有意水準は，この α の許容される水準であり，心理学では一般に 5% に設定する。

このように帰無仮説検定という考え方は，エラーの統制が本来の狙いであるから，「有意になるように工夫する」

という発想は根本的に間違っている。また，統計的推定という数学的手続きに，人間が納得しやすい判定を下すという人為的手続きが組み合わさったものであるから，帰無仮説検定の結果に過剰な意味を持たせたり一喜一憂したりすることがないように注意しよう。

7.1.2　帰無仮説検定の手続き

帰無仮説検定の手続きを一般化すれば，次のようになる。

1. 帰無仮説と対立仮説を設定する。
2. 検定統計量を選択する。
3. 判定基準を決定する。
4. 検定統計量を計算する。
5. 判定する。

帰無仮説検定は，群間の平均値に差があるかどうか，相関係数に統計的な意味があるかどうかといった事例に対して適用される。当然のことながら，これは標本から母集団を推定するという文脈における話で，物理学的な真偽を理論的に判断するとか，全数調査のように母集団全体の情報が手に入る場合といった場合の話ではない。また，標本のサンプルサイズが小さく，標本統計量の信頼区間が大きいことから，枠組みなしには判定できないという背景があることも再確認しておこう。

母集団の状態がわからないので，仮説を設定する。帰無仮説 Null Hypothesis は空っぽの仮説という意味で，母平均差がない (差がゼロ，$\mu_1 - \mu_2 = 0$) とか，母相関がゼロ ($\rho = 0$) である，とされる。対立仮説 Alternative Hypothesis は帰無仮説と排他的な関係にある仮説としてつくられるから，「差が無くはない ($\mu_1 - \mu_2 \neq 0$)」「相関がゼロではない ($\rho \neq 0$)」という表現になる。なぜ帰無仮説がゼロであることから始められるかといえば，ふたつの排他的な仮説を考えた時にゼロでない状態というのは無数にあり得るので，仮説として特定できないからである (差が 1 のとき，1.1 のとき，1.11 のとき・・・と延々と検定し続けるわけにもいくまい)。

検定統計量の選択は，二群の平均値差のときは t，三群以上の時は F，相関係数の検定も t，と天下り的に示されることが一般的である。もちろんこれらの統計量が選ばれるのは，数理統計的な論拠に基づいている。判定基準は 5% 水準とすることが一般的だし，検定統計量の計算はアルゴリズムに沿って機械的に可能である。判定は客観的な指標に基づいて行われるから，「どの状況でどのような帰無仮説をおくか」が類型化できれば，この手続き全体が自動的に進められる。

しかしここでは改めて，丁寧に手順を追いながらみてみよう。

7.2　相関係数の検定

ここでは相関係数の検定を例に取り上げる。俗に「無相関検定」と呼ばれるように，相関がどれほど大きいとかどれほど意味があるということをチェックするのでは無く，無相関ではない，ということをチェックする。もちろん標本相関は計算してゼロでなければ，それは無相関ではない。ここで考えたいのは，母相関がゼロではないということである。言い換えると，母相関がゼロの状態であっても，標本相関がゼロでないことは，小標本のサンプリングという背景のもとでは当然のことである。

確認してみよう。まず，無相関なデータセットを作ることを考える。R の MASS パッケージを使い，多変量正規分布の確率分布関数から乱数を生成しよう。

```
library(MASS)
set.seed(12345)
N <- 100000
X <- mvrnorm(N,
  mu = c(0, 0),
  Sigma = matrix(c(1, 0, 0, 1), ncol = 2),
  empirical = TRUE
)
head(X)
```

```
           [,1]        [,2]
[1,] -0.4070308 -0.72271139
[2,] -0.5774631 -0.57075167
[3,]  0.2312929 -0.42458994
[4,]  0.6242499 -0.55522146
[5,] -0.7791585  0.55004824
[6,]  1.8995860 -0.04899946
```

ここでは 10^5 個の乱数を生成した。つくられたオブジェクト X は表示されているように，2 変数からなる。ここでは相関のある 2 変数を想定しており，各変数がそれぞれ標準正規分布に従っているという設定である。rnorm 関数を 2 つ使って 2 変数をつくっても良いのだが，2 変数セットで取り出すことを考えると多変量正規分布をかんがえることになる。多変量正規分布は，ひとつひとつの変数については正規分布として平均と SD をもち，かつ，変数の組み合わせとして共分散をもつものである。mvrnorm の引数をみると，mu は平均ベクトルであり，Sigma が分散共分散行列である。分散共分散行列とは，ここでは 2×2 の正方行列であり，対角項に分散を，非対角項に共分散をもつ行列である。共分散は標準偏差と相関係数の積で表される。

分散

$$s_x^2 = \frac{1}{n}\sum (x_i - \bar{x})^2 = \frac{1}{n}\sum (x_i - \bar{x})(x_i - \bar{x})$$

標準偏差

$$s_x = \sqrt{s_x^2} = \sqrt{\frac{1}{n}\sum (x_i - \bar{x})^2}$$

共分散

$$s_{xy} = \frac{1}{n}\sum (x_i - \bar{x})(y_i - \bar{y})$$

相関係数

$$r_{xy} = \frac{s_{xy}}{s_x s_y} = \frac{\frac{1}{n}\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\frac{1}{n}\sum(x_i - \bar{x})^2}\sqrt{\frac{1}{n}\sum(y_i - \bar{y})^2}}$$

分散共分散行列

$$\Sigma = \begin{pmatrix} s_x^2 & s_{xy} \\ s_{yx} & s_y^2 \end{pmatrix} = \begin{pmatrix} s_x^2 & r_{xy}s_x s_y \\ r_{xy}s_x s_y & s_y^2 \end{pmatrix}$$

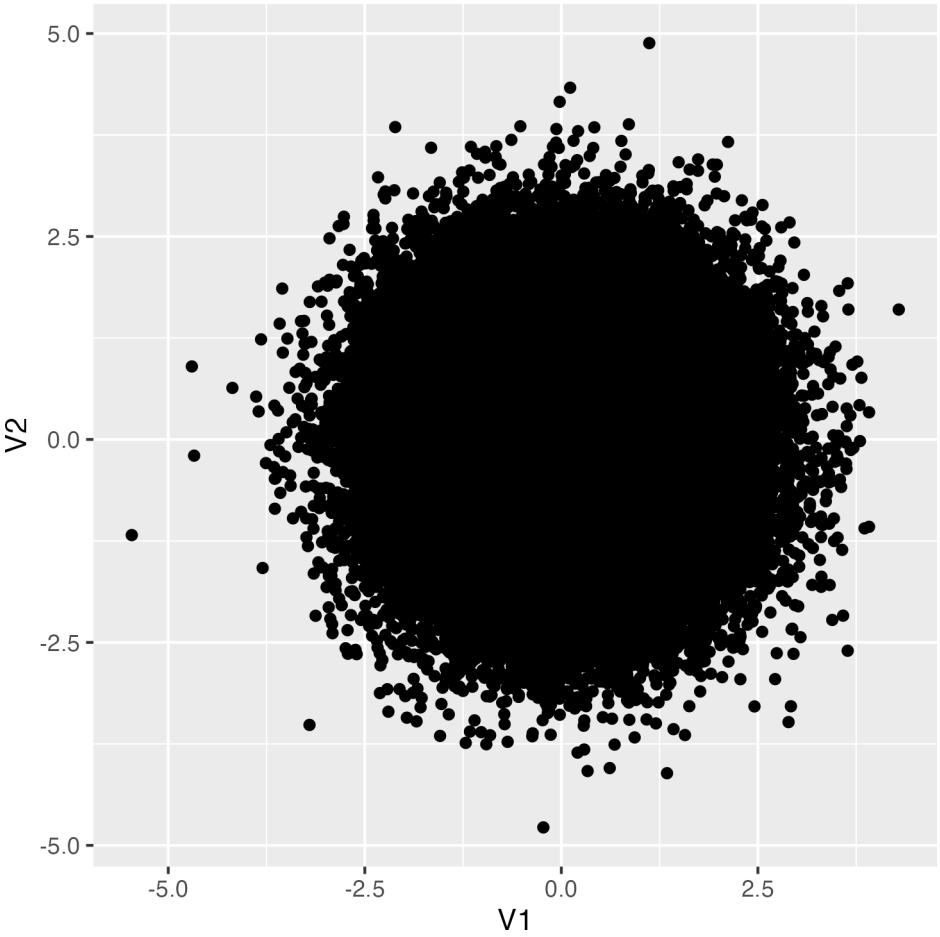
今回 `Sigma = matrix(c(1,0,0,1),ncol = 2)` としたのは，この 2 変数が無相関であること (SD はそれぞれ 1 であること) を指定している。ちなみに `empirical = TRUE` のオプションは，生成された乱数が設定した分散共分散行列のもつ相関係数と一致するように補正することを意味している。

可視化しておこう。つくられた乱数が無相関であることを，散布図を使って確認する。

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
x dplyr::select() masks MASS::select()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
X %>%
  as.data.frame() %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_point()
```



数値的にも確認しておこう。

```
cor(X) %>% round(5)
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

つくられた乱数が無相関であることが確認できた。さてこれが母集団であったとして，ここからたとえば n = 20 のサンプルをとったとする。この時の相関はどうなるだろうか。R で計算してみよう。sample 関数をつかって抜き出す行を決めて，該当する行だけ s1 オブジェクトに代入する。その上で相関係数を計算してみよう。

```
selected_row <- sample(1:N, 20)
print(selected_row)
```

```
 [1]  9647 80702 57543 93179 99032 82624 32672 53670 69698 42383 23801 69303
[13]  9816 61803 69464 23107 76958 44447    10 27292
```

```
s1 <- X[selected_row, ]
cor(s1)
```

```
           [,1]       [,2]
```

```
[1,] 1.0000000 0.1431698
[2,] 0.1431698 1.0000000
```

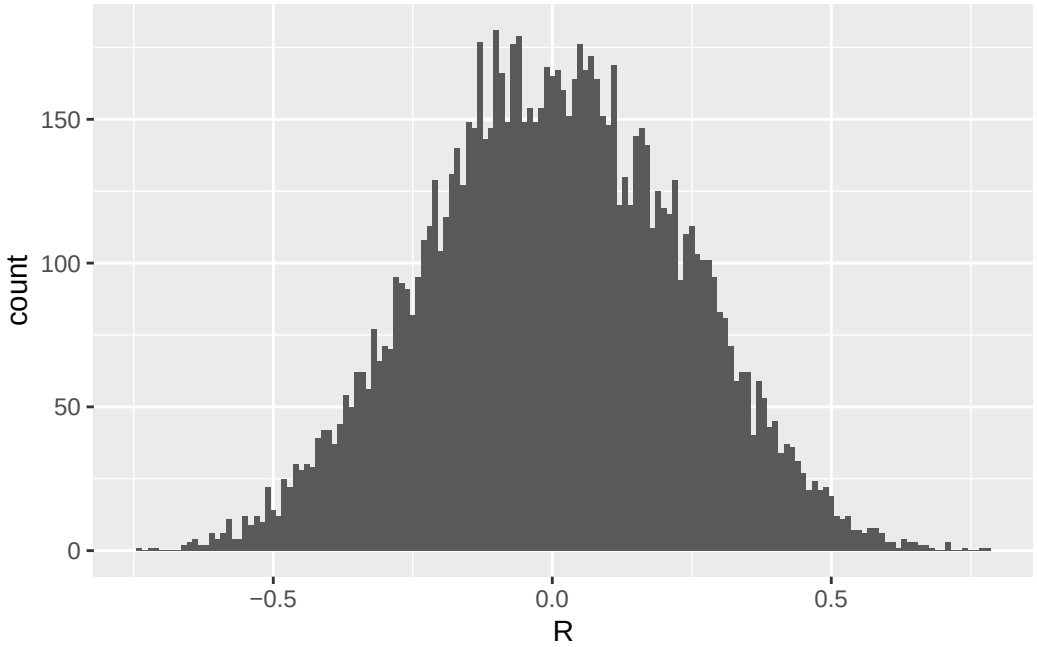
今回の相関係数は 0.1431698 となった。母集団の相関係数が 0 であっても，適当に抜き出した 20 点が相関係数を持ってしまう (0 でない) ことはあり得ることなのである。問題は，これがどの程度あり得ることなのか，である。いいかえると，研究者が $n = 20$ のサンプルをとって相関を得た時，それが $r = 0.14$ であったとしても，母相関 $\rho = 0.0$ からのサンプルである可能性がどれぐらいあるか，ということである。

7.3　標本相関係数の分布と検定

標本相関係数は確率変数なので，毎回標本を取る度に値が変わるし，どの実現値がどの程度出現するかは標本分布で表現できる。ではどのような標本分布に従うのだろうか。先ほどのサンプリングを繰り返して，乱数によって近似してみよう[*1]。

```
iter <- 10000
samples <- c()
for (i in 1:iter) {
  selected_row <- sample(1:N, 20)
  s_i <- X[selected_row, ]
  cor_i <- cor(s_i)[1, 2]
  samples <- c(samples, cor_i)
}
df <- data.frame(R = samples)
# ヒストグラムの描画
g <- df %>%
  ggplot(aes(x = R)) +
  geom_histogram(binwidth = 0.01)
print(g)
```

[*1] このような二度手間を取らず，mvrnorm からサンプルサイズ 20 の乱数を反復生成しても良い。母集団を具体的なものとしてイメージするために，母相関が 0 の母集団からサンプリングを繰り返す方法をとった。

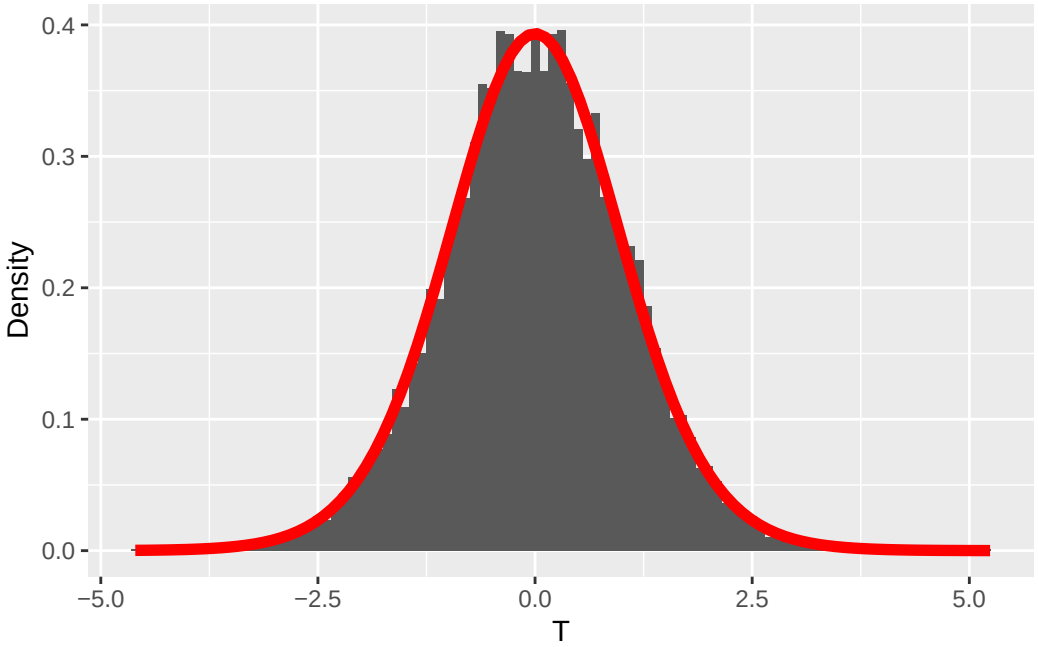


ヒストグラムを見ると，サンプルサイズが 20 の場合，母相関係数 $\rho = 0.0$ であっても $r = 0.3$ や $r = 0.4$ 程度の標本相関が出現することはある程度みられることである。

また，標本分布は左右対称の何らかの理論分布に従っていそうだ。数理統計学の知見から，相関係数の場合，標本相関係数を次の式によって変換することで，自由度が $n - 2$ の t 分布に従うことが知られている。

$$t = \frac{r\sqrt{n-2}}{\sqrt{1-r^2}}$$

```
df %>%
  mutate(T = R * sqrt(18) / sqrt(1 - R^2)) %>%
  ggplot(aes(x = T)) +
  geom_histogram(aes(y = after_stat(density)), binwidth = 0.1) +
  # 自由度 18 の t 分布の確率密度関数を追加
  stat_function(fun = dt, args = list(df = 18), color = "red", linewidth = 2) +
  # Y 軸のラベルを変更
  ylab("Density")
```



これを利用して相関係数の検定が行われる。以下，サンプルサイズ 20 で標本相関係数が $r = 0.5$ だったとして，手順に沿って解説する。

1. 帰無仮説は母相関 $\rho = 0.0$ とする。対立仮説は $\rho \neq 0.0$ である。
2. 検定統計量は相関係数 r を変換した t とする。
3. 判定基準として，$\alpha = 0.05$ とする。すなわち，母相関が 0 であるという仮説を棄却して間違える確率を 5% 以下に制御したい。
4. 検定統計量を計算する。$n = 20, r = 0.5$ より，

$$t = \frac{0.5 \times (\sqrt{18})}{\sqrt{1 - 0.5^2}} = 2.449$$

5. 標本相関係数の**絶対値が** 0.5 **を超える**確率は，t 分布の理論値から，次のように計算できる。あるいは，t 分布の両端 5% を切り出す**臨界値**を次のように計算できる。

```
(1 - pt(0.5 * sqrt(18) / sqrt(1 - 0.5^2), df = 18)) * 2
```

```
[1] 0.02476956
```

```
qt(0.975, df = 18)
```

```
[1] 2.100922
```

ここで注意してほしい点は，今回の検定の目的が「母相関が 0 であるという帰無仮説を棄却できるかどうか」であり，相関係数の符号については関心がなく絶対値で考える点である。pt 関数は，ある確率点までの累積面積であるから，1 から引くことでその確率点以上の値がでる確率が示される。t 分布は左右対称の分布なので，これを 2 倍した値が絶対値で考えた時の出現確率である。これが 5% よりも小さければ，有意であると判断できる。今回は，統計的に有意であるといって良い。

なお，表現上の細かい注意点になるが，この確率は今回の実現値「以上」のより極端な値が出る確率であり，この

実現値が出る確率という言い方はしない。確率は面積であり，点に対する面積はないからである。

qt 関数で示されるのは確率点なので，これ以上の値を今回の実現値が出していたら，統計的に有意であると判断できる。今回の実現値から算出した値は $t(18) = 2.449$ であり，臨界値の 2.100 よりも大きな値なので，有意であると判断できる。

7.4　2 種類の検定のエラー確率

上では丁寧に計算過程をみてきたが，実践場面ではサンプルはひとつであり，標本統計量もひとつ算出されるだけである。自分の大切なデータであるから，標本分布から得られた特定のケースにすぎないことが直感的にわかりにくいかもしれない。

相関係数の検定をするときは，R の関数 cor.test を使って次のように行う。ここでは mvrnorm 関数を使って，相関係数 0.5 の仮想データを作っている。

```
set.seed(17)
n <- 20
sampleData <- mvrnorm(n,
  mu = c(0, 0),
  Sigma = matrix(c(1, 0.5, 0.5, 1), ncol = 2),
  empirical = TRUE
)
cor.test(sampleData[, 1], sampleData[, 2])
```

```
    Pearson's product-moment correlation

data:  sampleData[, 1] and sampleData[, 2]
t = 2.4495, df = 18, p-value = 0.02477
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.07381057 0.77176071
sample estimates:
cor
0.5
```

結果として示されている，t の値や自由度，p 値が先ほど示した例と対応していることを確認できる。さらに，相関係数の信頼区間や標本相関係数そのものも示されている。この信頼区間が 0 を跨いでいないことからも，帰無仮説が棄却されることが見て取れるだろう。

われわれは既に，母相関が 0 のデータセットの一部を取り出すと，その相関係数が 0 ではなく 0.5 のような数字になることも知っている。もちろん母相関が 0 であれば標本相関も 0 近い値が出やすいとしても，である。つまり標本から得られた値をあまり大事に考えすぎない方が良い (もちろん一般化を念頭においている時は，である)。

また，帰無仮説は「母相関が 0 である」なので，これが**棄却**されたとしても「母相関が 0 であるとは言えない」のに過ぎない。ここから，母相関も $r = 0.5$ 付近にあるはずだとか，p 値が 2.4% なので 5% よりもずいぶん低いのは証拠の重要さを物語っているのだ，と論じるのは適切ではない。母相関が 0 という仮想的な状況のもとでの話であって，母相関が実際にどの程度なのかを検討しているわけではない。この点が誤解されやすいので特に注意してほしい。

ここに来るとタイプ 1 エラー，タイプ 2 エラーがより具体的に理解できるようになってきたではないだろうか。タイプ 1 エラーはこの帰無仮説が正しい時に，標本相関から計算した統計量で判断する確率であるから，上の手続きで見たことそのものである。

別の角度で見てみよう。`cor.test` をつかうと標本統計量の信頼区間が算出できる。この信頼区間が母相関 – ここでは帰無仮説である $\rho = 0$ を「正しく」含んでいる割合を見てみよう。`cor.test` 関数が返すオブジェクトには，`conf.int` という名前のものがあり，デフォルトではここで 95% の信頼区間が含まれている。シミュレーションに先立って，結果を格納する 2 列のデータフレームを作っておき，シミュレーション後に `ifelse` 関数で母相関が含まれているかどうかの判定をした。

```
set.seed(42)
iter <- 10000
intervals <- data.frame(matrix(NA, nrow = iter, ncol = 2))
names(intervals) <- c("Lower", "Upper")
for (i in 1:iter) {
  selected_row <- sample(1:N, 20)
  s_i <- X[selected_row, ]
  cor_i <- cor.test(s_i[, 1], s_i[, 2])
  intervals[i, ] <- cor_i$conf.int[1:2]
}
#
df <- intervals %>%
  mutate(FLG = ifelse(Lower <= 0 & Upper >= 0, 1, 0)) %>%
  summarise(type1error = mean(FLG)) %>%
  print()
```

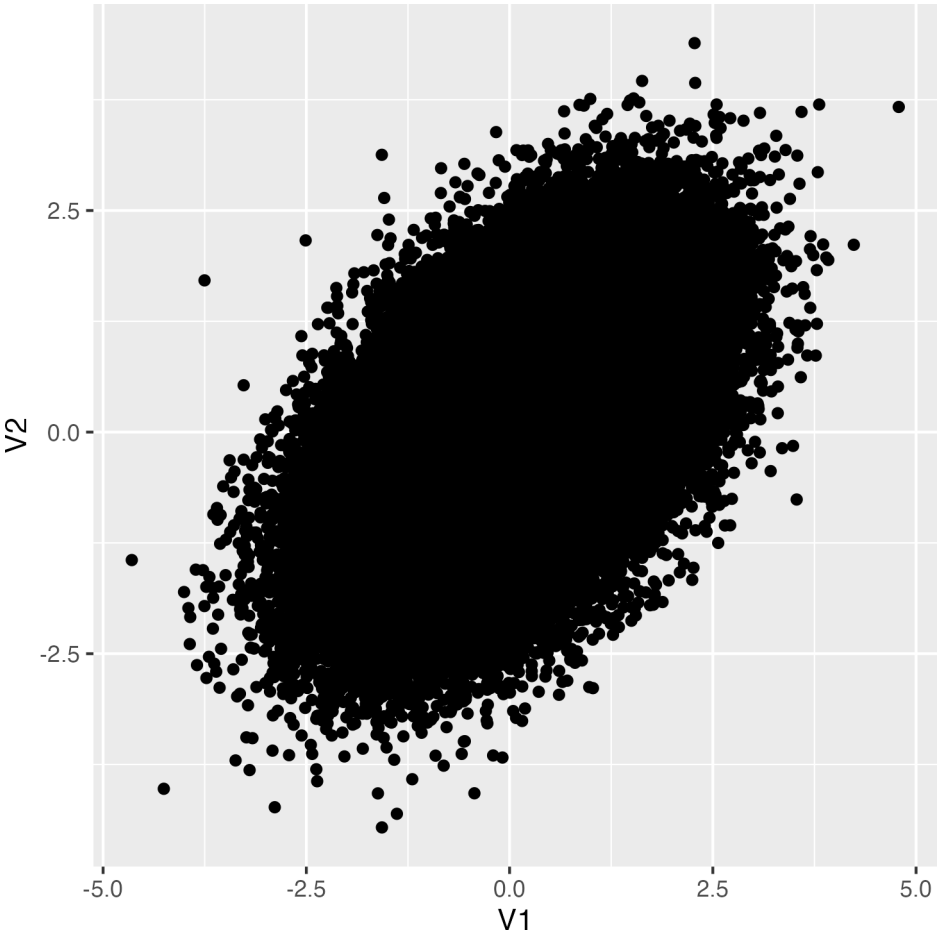
```
  type1error
1       0.95
```

今回の例では，95% の割合で正しく判断できていた。言い換えると，エラーが生じる割合は 5% だったので，タイプ 1 エラー確率を 5% 以下にするという目的はしっかり達成できていたことが確認できた。

同様に，タイプ 2 エラーは，帰無仮説が正しくないときに帰無仮説を採択する確率だから，シミュレーションするなら次のようになる。まず母相関が 0 でない状況を作り出そう。今回は母相関が 0.5 であるとして，母集団分布を描いてみよう。

```
set.seed(12345)
N <- 100000
```

```
X <- mvrnorm(N,
  mu = c(0, 0),
  Sigma = matrix(c(1, 0.5, 0.5, 1), ncol = 2),
  empirical = TRUE
)

X %>%
  as.data.frame() %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_point()
```



今度は，ここからサンプルサイズ 20 のデータセットを取り出し，検定することにしよう。検定の結果，有意になれば 1，ならなければ 0 というオブジェクトを作って，判定の正しさを考えてみることにする。

```
iter <- 10000
judges <- c()
for (i in 1:iter) {
  selected_row <- sample(1:N, 20)
  s_i <- X[selected_row, ]
```

```
  cor_i <- cor.test(s_i[, 1], s_i[, 2])
  judges <- c(judges, cor_i$p.value)
}
df <- data.frame(p = judges) %>%
  mutate(FLG = ifelse(p <= 0.05, 1, 0)) %>%
  summarise(
    sig = sum(FLG == 1),
    non.sig = sum(FLG == 0),
    type2error = non.sig / iter
  ) %>%
  print()
```

```
   sig non.sig type2error
1 6442    3558     0.3558
```

今回は母相関が 0.5 であり，帰無仮説は棄却されて然るべきなのだが，有意でないと判断された割合が 35.58% あったことになる。心理学の研究などでは，この確率 β が 0.2 未満，逆にいうと検出が 0.8 以上あることが望ましいとされているので，今回のこの事例では十分な件出力がなかった，と言えるだろう。

もちろん実際には，母相関がどれぐらいなのかわからない。0.3 なのかもしれないし，-0.5 であるかもしれない。つまりタイプ 2 エラーは研究者が制御できるところではなく，せいぜい大きな相関が見込めそうな変数について標本を取ろうと心がけるだけである。

タイプ 1,2 エラーの確率は，サンプルサイズや効果量 (ここでは母相関の大きさ) の関数である。サンプルサイズは研究者が決定することができるので，効果を見積もり，制御したいエラー確率の基準を決めて，合理的にサンプルサイズを決めるべきである。

7.5 課題

1. 母相関が 0 の母集団から，サンプルサイズ 10 の標本を取り出して標本相関を見た時の標本分布を，乱数のヒストグラムで近似してみましょう。
2. 同じく，サンプルサイズ 50 の標本を取り出して標本相関を見た時の標本分布を，乱数のヒストグラムで近似してみましょう。サンプルサイズが 20 や 10 の時と比べてどういう違いがあるでしょうか。
3. サンプルサイズ 50 の標本相関が $r = -0.3$ のとき，統計的に有意と言えるでしょうか。`cor.test` をつかって検定し，検定結果と判断結果を記述してください。
4. 標本相関が $r = -0.3$ だとします。サンプルサイズが 10,20,50,1000 のとき，統計的に有意と言えるでしょうか。`cor.test` を使って検定し，検定結果を一覧にしてみましょう。ここから何がわかるでしょうか。
5. 母相関が $\rho = -0.3$ だったとします。サンプルサイズ 20 のとき，どの程度の検出力があると見込めるでしょうか。シミュレーションで近似してください。

第 8 章

平均値差の検定

平均値差の検定は，実験計画の結論を出すために用いられる手段である。無作為割り当てによって個人差や背景要因が相殺され，平均的な因果効果を検証することができるからである。その結果を一般化するためには，やはり推測統計学の知見が必要であり，サンプルサイズやタイプ 1,2 エラーが関わってくることに変わりはない。

8.1 一標本検定

まず配置標本検定の例から始める。母平均がわかっている，あるいは理論的に仮定される特定の値に対して，標本平均が統計的に有意に異なっていると言って良いかどうかの判断をするときに用いる。たとえば 7 件法のデータを取ったときに，ある項目の平均が中点 4 より有意に離れていると言って良いかどうか，といった判定をするときに用いる。かりに，サンプルサイズ 10 で 7 件法のデータが得られたとしよう。ここでは平均 4,SD1 の正規乱数を 10 件生成することで表現する。実際にはこの値を，人に対する尺度カテゴリへの反応として得ているはずである。

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
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
```

```
set.seed(17)
n <- 10
mu <- 4
X <- rnorm(n, mean = mu, sd = 1)
```

```
print(X)
```

```
 [1] 2.984991 3.920363 3.767013 3.182732 4.772091 3.834388 4.972874 5.716534
 [9] 4.255237 4.366581
```

今回，標本平均は 4.177 であり，これより極端な値が $\mu = 4$ の母集団から得られるかどうかを検定する。帰無仮説検定の手順にそって進めていくと，以下のようになる。

1. 帰無仮説は母平均が理論的な値 (ここでは尺度の中点 4) であること，すなわち $\mu = 4$ であり，対立仮説は $\mu \neq 4$ である。
2. 検定統計量は，正規母集団から得られる標本平均が従う標本分布であり，母分散が未知の場合の区間推定に用いた T 統計量になる。
3. 判断基準は心理学の慣例に沿って 5% とする。

このあと，検定統計量の計算と判定である。これを R は `t.test` 関数で一気に処理できる。

```
result <- t.test(X, mu = mu)
print(result)
```

```

    One Sample t-test

data:  X
t = 0.6776, df = 9, p-value = 0.5151
alternative hypothesis: true mean is not equal to 4
95 percent confidence interval:
 3.585430 4.769131
sample estimates:
mean of x
 4.177281
```

結果として，今回の検定統計量の実現値は 0.678 であり，自由度 9 の t 分布からこれ以上の値が出てくる確率は，0.515 であることがわかる。これは 5% 水準と見比べてより大きいので，レアケースではないと判断できる。つまり，母平均 4 の正規母集団から，4.177 の標本平均が得られることはそれほど珍しいものではなく，統計的に有意に異なっていると判断するには及ばない，ということである。

レポートなどに記載するときは，これら実現値や p 値を踏まえて「$t(9) = 0.66776, p = 0.5151, n.s.$」などとする。ここで n.s. は not significant の略である。

さてこの例では，母平均 4 の正規乱数を生成し，その平均が 4 と異なるとはいえない，と結論づけた。これは一見，当たり前のことのようであり，無意味な行為におもえるかもしれない。しかし次の例を見てみよう。

```
n <- 3
mu <- 4
X <- rnorm(n, mean = mu, sd = 1)
```

```
mean(X) %>%
  round(3) %>%
  print()
```

```
[1] 5.04
```

```
result <- t.test(X, mu = mu)
print(result)
```

```

    One Sample t-test

data:  X
t = 5.1723, df = 2, p-value = 0.03541
alternative hypothesis: true mean is not equal to 4
95 percent confidence interval:
 4.174825 5.904710
sample estimates:
mean of x
 5.039768
```

ここではサンプルサイズ $n = 3$ であり，標本平均が 5.04 であった。このとき t 値は 5% 臨界値を上回っており，「母平均 4 のところから得られる値にしては極端」であるから，統計的に有意に異なる，と判断することになる。乱数生成時は平均を確かに 4 に設定したが，母平均から取り出したごく一部が，そこから大きく離れてしまうことはあり得るのである。

8.2 二標本検定

続いて二標本の検定について考えよう。実験群と統制群のように，無作為割り当てをすることで平均因果効果をみる際に行われるのが，この検定である。帰無仮説は「群間差はない」であり，対立仮説はその否定である。また，正規母集団からの標本を仮定するので，検定統計量はここでも t 分布に従う値になる。帰無仮説検定の手順に沿って，改めて確認しておこう。

1. 帰無仮説は「二群の母平均に差がない」である。二群の母平均をそれぞれ μ_1, μ_2 とすると，帰無仮説は $\mu_1 = \mu_2$，あるいは $\mu_1 - \mu_2 = 0$ と表される。対立仮説は $\mu_1 \neq \mu_2$ あるいは $\mu_1 - \mu_2 \neq 0$ である。
2. 検定統計量は，正規母集団から得られる標本平均が従う標本分布であり，母分散が未知の場合の区間推定に用いた T 統計量になる。
3. 判断基準は心理学の慣例に沿って 5% とする。

これを検証するために，サンプルデータを乱数で生成しよう。まず，各群のサンプルサイズを `n1,n2` とする。ここでは話を簡単にするため，サンプルサイズは両群ともに 10 とした。つぎに両群の母平均だが，群 1 の母平均を μ_1，群 2 の母平均を $\mu_2 = \mu_1 + \delta$ で表現した。この δ は差分であり，これが $\delta = 0$ であれば母平均が等しいこと，$\delta \neq 0$ であれば母平均が異なることになる。最後に両群の母 SD を設定した。

ここでの検定は，この差分 d が母平均 0 の母集団から得られたと判断して良いかどうか，という形で行われる。検定統計量 T は，次式で算出されるものである。

$$T = \frac{d - \mu_0}{\sqrt{U_p^2 / \frac{n_1 n_2}{n_1 + n_2}}}$$

ここで d は二群の標本平均の差であり，U_p^2 はプールされた不偏分散と呼ばれ，二群を合わせて計算された全体の母分散推定量である。各群の標本分散をそれぞれ S_1^2, S_2^2 とすると，次式で算出される。

$$U_p^2 = \frac{n_1 S_1^2 + n_2 S_2^2}{n_1 + n_2 - 2}$$

これらの式はつまり，サンプルサイズの違いを考慮するため，一旦両群の標本分散に各サンプルサイズを掛け合わせ，プールした全体のサンプルサイズから各々 -1 をすることで全体として不偏分散にしている。

これを踏まえて，具体的な数字で見ていこう。その上で乱数でデータを生成し，その標本平均を確認した上で，`t.test` 関数によって検定を行っている。

```
n1 <- 10
n2 <- 10
mu1 <- 4
sigma <- 1
delta <- 1
mu2 <- mu1 + (sigma * delta)

set.seed(42)
X1 <- rnorm(n1, mean = mu1, sd = sigma)
X2 <- rnorm(n2, mean = mu2, sd = sigma)

X1 %>%
  mean() %>%
  round(3) %>%
  print()
```

```
[1] 4.547
```

```
X2 %>%
  mean() %>%
  round(3) %>%
  print()
```

```
[1] 4.837
```

```
result <- t.test(X1, X2, var.equal = TRUE)
print(result)
```

```
    Two Sample t-test

data:  X1 and X2
t = -0.49924, df = 18, p-value = 0.6237
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.506473  0.927980
sample estimates:
mean of x mean of y
 4.547297  4.836543
```

今回の母平均は $\mu_1 = 4, \mu_2 = 4 + 1$ にしているが，標本平均は 4.547 と 4.837 であり，標本上では大きな差が見られなかった。結果として，t 値は 0.4992369 であり，自由度 18 のもとでの p 値は 0.6236593 である。5% 水準を上回る値であるから，結論としては対立仮説を採択するには至らない，差があるとはいえない，である。

今回の設定では母平均に差があるはず $(4 \neq 4 + 1)$ なのだから，これは誤った判断で，タイプ 2 エラーが生じているケースということになる。研究実践場面では，母平均やその差については知り得ないのだから，このような判断ミスが生じていたかどうかは分かり得ないことに留意しよう。

なお，ここではわかりやすく 2 群であることを示すために `X1`,`X2` と 2 つのオブジェクトを用意したが，実践的にはデータフレームの中で群わけを示す変数があり，`formula` の形で次のように書くことが多いだろう。

```
dataSet <- data.frame(group = c(rep(1, n1), rep(2, n2)), value = c(X1, X2)) %>%
  mutate(group = as.factor(group))
t.test(value ~ group, data = dataSet, var.equal = TRUE)
```

```
    Two Sample t-test

data:  value by group
t = -0.49924, df = 18, p-value = 0.6237
alternative hypothesis: true difference in means between group 1 and group 2 is not equal to 0
95 percent confidence interval:
 -1.506473  0.927980
sample estimates:
mean in group 1 mean in group 2
       4.547297        4.836543
```

8.3　二標本検定 (ウェルチの補正)

先ほどの t.test 関数には，var.equal = TRUE というオプションが追加されていた。これは 2 群の分散が等しいと仮定した場合の検定になる。t 検定は歴史的にこちらが先に登場しているが，2 群の分散が等しいかどうかはいきなり前提できるものでもない。等分散性の検定は，Levene 検定を行うのが一般的であり，R においては，car パッケージや lawstat パッケージが対応する関数を持っている。ここでは car パッケージの leveneTest 関数を用いる例を示す。

```
library(car)
leveneTest(value ~ group, data = dataSet, center = mean)
```

```
Levene's Test for Homogeneity of Variance (center = mean)
      Df F value Pr(>F)
group  1  2.9405 0.1035
      18
```

この結果を見ると，p 値から明らかなように，2 群の分散が等しいという帰無仮説が棄却**できなかった**ので，等しいと考えて t 検定に進むことができる。もしこれが棄却されてしまったら，2 群の分散が等しいという帰無仮説が成り立たないのだから，等分散性の仮定を外す必要がある。実行は簡単で，var.equal を FALSE にすれば良い。

```
result2 <- t.test(value ~ group, data = dataSet, var.equal = FALSE)
print(result2)
```

```
    Welch Two Sample t-test

data:  value by group
t = -0.49924, df = 13.421, p-value = 0.6257
alternative hypothesis: true difference in means between group 1 and group 2 is not equal to 0
95 percent confidence interval:
 -1.5369389  0.9584459
sample estimates:
mean in group 1 mean in group 2
       4.547297        4.836543
```

よく見ると，タイトルが Welch Two Sample t-test に変わっている。Welch の補正が入った t 検定という意味である。また自由度が実数 (13.421) になっているが，このように t 分布の自由度を調整することで等分散性の仮定から逸脱した場合の補正となる。もちろん報告する際は「$t($ 13.421 $)$ = -0.499, $p = 0.626$」のように書くことになるから，自由度が実数であれば補正済みであると考えられるだろう。

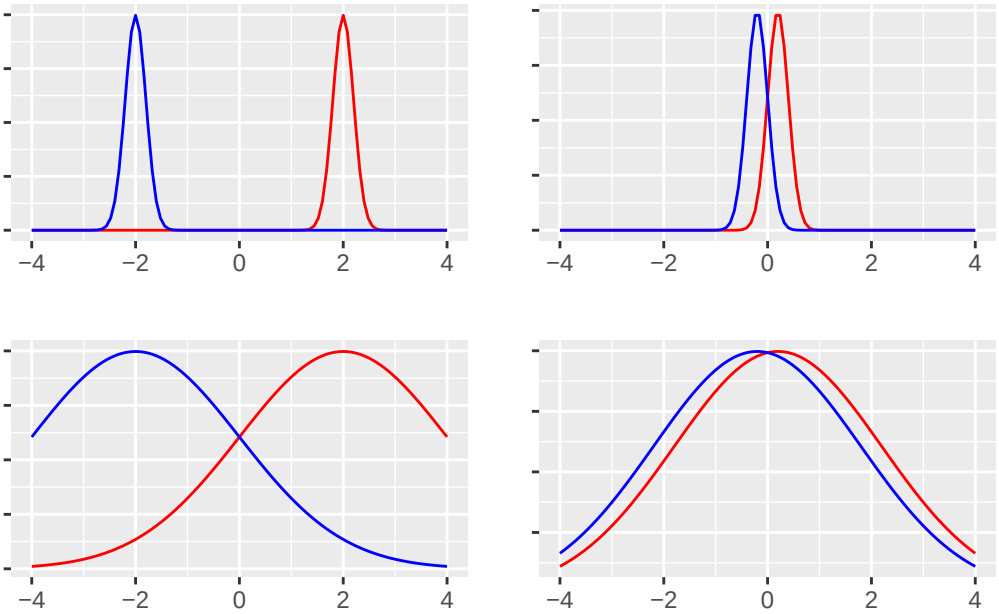
しかし，分散が等しいという仮定は，等しくない場合の特殊な場合であるから，最初から Welch の補正がはいった検定だけで十分である。このような考え方から，R における t.test 関数のデフォルトでは var.equal = FALSE となっており，特段の指定をしなければ等分散性の仮定をしない。こちらの方が検定を重ねることがないので，よ

り望ましい。

8.3.1　効果量の算出

今回の例は，仮想データとして $\mu_1 = 4, \mu_2 = \mu_1 + \sigma d$ であり，明らかに $\mu_1 \neq \mu_2$ なのだが，有意差を検出するには至らなかった。統計的な有意差はあくまでも「統計的な」観点からのものであり，我々が現実に検証したいのは本当に差があるかどうか，いわば「実質的な差」があるかどうかであるのだから，統計的な有意差を得ることを目的にするのははっきりと不適切な目標設定であると言えるだろう[*1]。

ところで，統計的に差があるとはっきり言えるのはどのような時だろうか。これは次の 4 つのデータの分布を見てもらうとわかりやすい。



左列は平均差が大きいデータ，右列は小さいデータである。上段は分散が小さいデータ，下段は大きいデータである。この 4 つそれぞれのシーンにおいて，「差がある」と判断しやすいのはどれかを考えてみるとよい。当然，左上のシーンが最も明確に差があると言えるであろう。なぜなら，両群が明確に分かれており，群間の重複がないからである。左下は同じ平均値差であっても，群内の広がりが大きいから群間の重複がみられるため，「差がある」という判断を受けても各群の中には該当しないケースがちらほらみられることだろう。右上パネルのようなケースでは，重複は少ないが差が小さいため，「差がある」と判断できるかどうかが微妙である。右下に至っては，差も小さく分布の重複も大きいから，「差がある」と判断しても該当しないケースが多くなる。たとえば「男性は女性よりも力が強い (体力・筋力に差がある)」というデータがあったとしても，「女性より非力な男性」もかなり多く存在するだろう。そういう反例が多くみられるような場合，統計的に差があるという結果が示されたとしても，受け入れられないのではないだろうか。

[*1] たとえば物理学などのシーンでは，測定の精度が高く，単一の物理世界を対象にした検証を行うのだから，予測が真であるか偽であるかを確率的に考えるような必要はない。そのような世界における検証 – あえて理論的な正しさが明確な世界，と表現するが – であれば，統計的な差があるかどうかの情報はあくまでも理論を支持するおまけ情報にすぎない。いわば統計的検定の結果を報告するのは，論文を書くためのレトリックである。翻って，人間を対象にした小サンプルの科学である心理学は，統計的な判断に頼らざるを得ないという側面はあるだろう。しかしだからと言って，実質的な差が本質的であることを忘れてしまっては本末転倒である。

ここから明らかなように，差の判断には平均値差だけでなく分散も関わってくる。そこで平均値差を標準偏差で割った，**標準化された差**が重要になってくるのであり，これが**効果量**と呼ばれるものである[*2]。

今回 2 群の差のデータを作る時に，σd としたが，平均値差の効果量 es は，

$$es = \frac{\mu_1 - \mu_2}{\sigma}$$

で表現されるから，d が効果量を表していたのである。もちろん我々は母平均，母 SD などを知り得ないのでこれもデータから推定する他ない。幸い R には `effsize` パッケージなど，効果量を算出するものが用意されている。

```
library(effsize)
cohen.d(value ~ group, data = dataSet)
```

```
Cohen's d

d estimate: -0.2232655 (small)
95 percent confidence interval:
    lower     upper
-1.165749  0.719218
```

```
cohen.d(value ~ group, data = dataSet, hedges.correction = TRUE)
```

```
Hedges's g

g estimate: -0.2138318 (small)
95 percent confidence interval:
     lower      upper
-1.1162608  0.6885973
```

平均値差の検定の後は，ここに示した Cohen の d や Hedges の g といった効果量を添えて報告することが一般的である。

8.4　対応のある二標本検定

実験群と統制群のように異なる 2 群ではなく，プレポスト実験のように対応がある 2 群の場合は，t 検定の定式化が異なる。対応がない t 検定の場合は，群平均の差 $\mu_1 - \mu_2$ の分布を考えたが，対応がある場合は個々の測定の差，つまり $X_{i1} - X_{i2} = D_i$ を考える。この一つの標本統計量を検定するのだから，一標本検定の一種であるとも言える。またこの D の分布の標準誤差は，標本標準誤差 U_D を使った $U_D/\sqrt{n}$ を使って推定する[*3]。検定統計量 T は，次式で算出される。

[*2] 統計的な有意差よりも効果量，効果量よりも実質的な差のほうが意味のある差であることを忘れてはならない。詳しくは豊田 (2009) を参照。

[*3] 対応があるケースを考えているので，当然 n は前後の群で同数である。

$$T = \frac{\bar{D}}{U_D/\sqrt{n}} = \frac{\sum D_i/n}{\sqrt{\frac{\frac{1}{n-1}\sum(D_i-\bar{D})^2}{n}}}$$

検定にあたっては，t.test 関数の引数 paired を TRUE にするだけで良い。

8.4.1 仮想データの組成

仮想データを作って演習してみよう。データの組成については，2 種類のアプローチで説明が可能である。ひとつは次のシミュレーションで表されるような形である。

```
n <- 10
mu1 <- 4
sigma <- 1
d <- 1
X1 <- rnorm(n, mu1, sigma)
X2 <- X1 + sigma * d + rnorm(n, 0, sigma)
t.test(X1, X2, paired = TRUE)
```

```
    Paired t-test

data:  X1 and X2
t = -1.8036, df = 9, p-value = 0.1048
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -1.4339112  0.1617193
sample estimates:
mean difference
     -0.6360959
```

すなわち，第一の群が μ_1 を平均にばらついた実現値として得られ，第二の群はその実現値に一定の効果 $\sigma * d$ が加わり，その測定にさらに誤差がつく形である。この方法は具体的なデータ生成プロセスをそのまま模したような形でデータを作っているが，測定誤差を二重に計上している点が気になるかもしれない。

もう一つの考え方は，プレポスト型のデータに限らず，何らかの形で「対応がある」ことも表現できるものである。対応があるということは，2 つのデータがそれぞれ独立した一変数正規分布から得られているのではなく，二変数正規分布から得られると考えるのである。二変数正規分布は，それぞれの変数は正規分布しているが，両者の間に相関があると考えるものである。変数が一つだけの正規分布は

$$X \sim N(\mu, \sigma)$$

で表現されているのに対し，複数の変数を同時に生成する多変数 (多次元) 正規分布 Multivariate Normal Distribution は，以下のように表現される。

$$\mathbf{X} \sim MVN(,)$$

ここで $\mathbf{X}$ や は n 次元ベクトルであり， は分散共分散行列を表している。二変数の場合は以下のように書くことができる。

$$= \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 \\ \rho_{21}\sigma_2\sigma_1 & \sigma_2^2 \end{pmatrix}$$

共分散 σ_{ij} は相関係数 ρ_{ij} を用いて書けることからわかるように，変数間に相関があることを想定してデータを生成するのである。この組成に従った仮想データの作成は以下のとおりである。

```
library(MASS) # 多次正規乱数を生成するのに必要
n <- 10
mu1 <- 4
sigma <- 1
d <- 1
mu <- c(mu1, mu1 + sigma * d)
rho <- 0.4
SIG <- matrix(c(sigma^2, rho * sigma * sigma, rho * sigma * sigma, sigma^2), ncol = 2, nrow = 2)
X <- mvrnorm(n, mu, SIG)
t.test(X[, 1], X[, 2], paired = TRUE)
```

```
    Paired t-test

data:  X[, 1] and X[, 2]
t = -2.4313, df = 9, p-value = 0.0379
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -1.96934592 -0.07095313
sample estimates:
mean difference
       -1.02015
```

効果量については，対応のない t 検定の場合と同じで良い。

```
cohen.d(X[, 1], X[, 2])
```

```
Cohen's d

d estimate: -1.04088 (large)
95 percent confidence interval:
      lower       upper
-2.04204357 -0.03971697
```

```
cohen.d(X[, 1], X[, 2], hedges.correction = TRUE)
```

```
Hedges's g

g estimate: -0.9968994 (large)
95 percent confidence interval:
     lower      upper
-1.9510179 -0.0427809
```

8.4.2 検定の方向性

ここまでの検定では，主に「差があるかどうか」といった仮説に対応するものを扱ってきた。差があるかどうか，というのはその差がプラスの方向にでているのか，マイナスの方向に出ているのかといったことを問題にしていない。そこで検定統計量の分布についても，分布の両裾を考えて有意水準を設定していた。

しかしプレポスト実験などでは，効果が「上がった」のか「下がった」のか，ということが大きな関心時でもあることが多いだろう。効果がある，ただし逆効果である，というのでは意味がないからである。このように方向性をもった仮説を検証する場合は，検定統計量の分布も一方向だけ考えればよく，`t.test` 関数には `alternative` オプションをつかって表現する。

`t.test(x,y,alternatives = "less")` とすると $x < y$ の帰無仮説を検証することになるし，`alternatives = "greater"`とすると $x > y$ の帰無仮説を検証することになる。デフォルトでは `alternatives = "two.sided"` であり，両側検定が選ばれている。

ただし，両裾から片裾にかわるということは，検定統計量が超えるかどうかの判断をする**臨界値**が小さくなることでもある。必然的に，片裾 (片側検定) のほうが緩やかな基準で検定をしていることにもなる。デフォルトで普段から厳しく検定しているから大丈夫だろう，というのも一つの考え方だが，やはり本来の研究仮説に適した帰無仮説の設定をするべきだろう。

8.5 課題

1. 平均が 50、標準偏差が 10 の正規分布からランダムに選んだ 30 個のサンプルを用意し，このサンプルの平均が母集団の平均と異なるかどうかを検定してください。検定結果を，心理学のフォーマット (心理学会編「論文執筆投稿の手引き」) に準拠した書き方で，結果を記述してください。

2. 以下のデータセットを使用して，2 つの独立した群の平均に差があるかどうかを t 検定してください。検定結果を，心理学のフォーマット (心理学会編「論文執筆投稿の手引き」) に準拠した書き方で，結果を記述してください。

$$group1 = \{45, 50, 55, 60, 65\}$$

$$group2 = \{57, 60, 62, 77, 75\}$$

3. 多次元正規分布を用いた仮想データ生成方で，対応のある t 検定の練習をしましょう。サンプルサイズを $n = 20$ とし, 平均ベクトル $\mu = (12, 15)$, 分散共分散行列 $\Sigma = \begin{pmatrix} 4 & 2.8 \\ 2.8 & 4 \end{pmatrix}$ の多次元正規分布から作られた乱数を使って，対応のある t 検定をしてください。検定結果を，心理学のフォーマット (心理学会編「論文執筆投稿の手引き」) に準拠した書き方で，結果を記述してください。

4. 自由度が 10, 20, 30 の t 分布のグラフを，標準正規分布のグラフとともに描画してください。自由度が増えると t 分布がどのように変化するでしょうか。

5. 自由度が 15 の t 分布において、有意水準 5% の片側検定と両側検定の臨界値 (検定の判断基準となる理論値) を求めてください。

第 9 章

多群の平均値差の検定

心理学実験においては，古典的に分散分析モデルが多用されてきた。平均値差を見ることで実験の効果，因果関係を明らかにできるように，巧妙に実験デザインが組み立てられる。その精緻さは理論的一貫性という意味である種の美しさを持ち，多くの研究者が魅了されてきた。

分散分析にのせることを目的にした実験計画であり，実験デザインの不自由さ (とにかく分散分析をしなければならない!) が批判的に論じられることもあるが，心理学が測定している対象が平均値差以上の精度で議論できる性質でないという反論もあるだろう。

今や分散分析を超えたより高度な統計モデルがあり，現在の研究においては分散分析はもはや過去のものにすぎないかもしれないが，以後のモデルも分散分析を基本としたその発展系であるので，改めて基本を押さえておくことも重要である。

9.1 分散分析の基礎

分散分析は「分散」の分析であるかのような名称であるが，平均値差を検定するためのものである。なぜ「分散」を冠するかといえば，効果量のところで見たように，平均値差の判断には群内分散の情報が必要だからである。

多群の平均値の差，その散らばりを群間分散といい，群に含まれるデータの散らばりを群内分散とよぶ。分散分析は群内分散に対する群間分散の比が十分に大きいと考えられる場合，群間に統計的な有意差があると判断する。分散の比を表す確率分布は F 分布と呼ばれる。F 分布は群間・群内それぞれの自由度を母数にもつ。

また，実験計画は Between デザインと Within デザインに区分される。t 検定でみたような，対応のない独立した群を対象にしたデザインが Between，群間に相関が想定される対応のある群を対象にしたデザインが Within である。Within デザインは同じ個体から複数回の反応を得る (ex. period 1-2-3…) ため，反復測定デザイン Repeated measured design ともよばれることがある。この場合，群内分散から個人内の分散すなわち個人差を取り出すことができるため，これが分離できない Between デザインよりも基本的に Within デザインのほうが目的となる変動を捉えやすい。ただし，反復測定による個体への負担を考えると，毎回 Within デザインでいいというわけにもいかないところが難点である。

$$BetweenDesign：\text{全変動} = \text{群間変動} + \text{群内変動 (誤差)}$$

$$WithinDesign : 全変動 = 群間変動 + 個人差変動 + 誤差$$

分散分析は要因が複数ある場合も考えられるから，要因 A が Between，要因 B が Within といった場合は混合計画と呼ばれることがある。慣例的に，要因 Factor とその要因に含まれる水準 Level を同時に表現し，間2 × 間3 の分散分析 (二要因の分散分析で，いずれも Between デザインであり，水準数がそれぞれ 2 と 3)，といった言い方をすることがある。

9.2　分散分析のステップ

t 検定において等分散性の仮定が成立するかどうかが事前に問題になったように，Between デザインにおいても分散の等質性は仮定されており，Levene の検定などで事前に検証しておくべきである。また Within デザインにおいては，データの組成に関わる分散共分散行列の非対角要素が全て等しいことが望ましいが，実践的にはそこまでの仮定が成立しているとは考えにくい。ただし分散分析としては，等分散性の仮定よりも，より緩やかな球面性の仮定が成立していればよいとされており，これを事前に検定することが一般的である。Welch の補正のように，球面性の仮定が成立していない場合は，自由度を補正することで検定の精度が維持される。

分散分析は多要因・多水準の平均値差の検定である。各水準ごとに t 検定を繰り返せば良いのではないか，というアイデアは誰しも思いつくことであろうが，この方法は検定の目的である α 水準の制御ができなくなるという問題を含む。そこで多水準の場合は分散分析を行うことで，すべての要因・水準の母平均が同じであるという帰無仮説を検定し，効果の有無をまず明確にする。この帰無仮説が棄却されたらどこかに差があるわけだから，以後は慎重に α 水準を制御しつつ事後的な検定にすすむ。

水準間の差をみるための事後的な検定は，下位検定とも呼ばれる。その方法は多岐に渡り，ゴールドスタンダードは存在せず，往々にして分析者が利用しているソフトウェアが対応する手法が選択される。要因・水準が多くなると検証すべき組み合わせも多くなり，下位検定の手続きも非常に煩雑になる。統計ソフトウェアはそれこそ機械的に，幾重にも細かく分散分析表を分解して下位検定をつづけていってくれるが，いくら制御されているとはいえ検定を繰り返していることに変わりはないし，各下位検定の結果を一貫した総合的解釈をするのは困難である。実験計画はシンプルであるほうが望ましいし，複雑なモデルになるようであれば分散分析を超えた，階層線形モデルやベイジアンアプローチなどを取る方が良いだろう。

9.3　ANOVA 君を使う

分散分析を R で実行するには，基本関数である `aov` や car パッケージなどを用いることができる。もっとも，その出力は必ずしも親切ではないし，下位検定や効果量の算出などは別のパッケージ，別の関数を用いる必要がある。

筆者がお勧めするのは，大正大学の井関龍太が開発した anovakun である。パッケージ化されていないので，リンク先からソースコードを読み込んで anovakun 関数を実行する必要があるが，さまざまな実験デザインに対応し，また下位検定や効果量，球面性の補正などおよそ分散分析で必要な手法は網羅されている。以下ではこれを用いた実践を行う。

anovakun の読み込みは，ソースコードをプロジェクトフォルダにダウンロードして `source` 関数で読み込むか，

インターネットに繋がっている状態でリンク先から直接ソースファイル (anovakun_489.txt)[*1]を source 関数で読み込むといいだろう。

```
source("https://riseki.cloudfree.jp/?plugin=attach&refer=ANOVA%E5%90%9B&openfile=anovakun_489.txt")
```

読み込みが終わると Environ タブに anovakun 関数が含まれていることを確認しよう。

9.3.1 ANOVA 君の入力とデータ

ANOVA 君は伝統的にワイド型データから読み込むようになっている。すなわち，一行に 1 オブザベーション入っている形式である。Between 計画の場合は，データの前に水準数を表すインデックスと最終的な従属変数の形に整形したデータが必要である。Within 計画の場合は 1 行に 1Obs. なのだから，反復した水準の数だけ右にデータを入れていく形に整形する。

しかし Chapter 3.7 で述べたように，昨今は計算機にとって優しい型，ロング型での入力もおおく，ANOVA 君も version 4.4.0 からロング型での入力も許すようになった。その場合はオプション long=TRUE とロング型であることを明記する必要がある。

ANOVA 君を使う時は，関数 anovakun に，データ，要因計画の型，各要因の水準の順で入力する。ここで要因計画の型とは，文字列で Between/Within の違いを明示することになる。被験者のラベルを表す小文字の s を挟んで，左側に間 (Between) 要因，右側に内 (Within) 要因を入れる。例えば一要因 Between 計画の場合は"As"，二要因 Within 計画の場合は"sAB"，間 1 内 2 の混合計画であれば"AsBC"のようにする。

続いて入力する水準数は，要因の数だけ必要である。ただし，ロング型で入力した場合は自動的に水準数が計算されるので入力の必要がない。

このテキストでは，データの持ち替えについてすでに触れているので，色々扱いやすいロング型に整形して利用していくものとする。

9.4 Between デザイン

9.4.1 1way-ANOVA

もっとも単純な一要因 3 水準，Between 計画の例から始めよう。仮想データの生成を行うことで，分散分析のメカニズムと共に見ていくことにする。

```
set.seed(123)
# 各群のサンプルサイズ
n1 <- 5
n2 <- 4
n3 <- 6
# 母平均，効果量，母 SD
```

[*1] 2024.06.12 時点での最新バージョンが 4.8.9 である。リンク先 URL は，公式サイトからソースファイルのリンクをコピーして貼り付けると良い。

```
mu <- 10
delta <- 1
sigma <- 3
# 群平均
mu1 <- mu - (delta * sigma)
mu2 <- mu
mu3 <- mu + (delta * sigma)
# データセット
X1 <- rnorm(n1, mu1, sigma)
X2 <- rnorm(n2, mu2, sigma)
X3 <- rnorm(n3, mu3, sigma)
## 組み上げる
dat <- data.frame(
  ID = 1:(n1 + n2 + n3),
  group = as.factor(rep(LETTERS[1:3], c(n1, n2, n3))),
  value = c(X1, X2, X3)
)
## データの確認
dat
```

```
   ID group     value
1   1     A  5.318573
2   2     A  6.309468
3   3     A 11.676125
4   4     A  7.211525
5   5     A  7.387863
6   6     B 15.145195
7   7     B 11.382749
8   8     B  6.204816
9   9     B  7.939441
10 10     C 11.663014
11 11     C 16.672245
12 12     C 14.079441
13 13     C 14.202314
14 14     C 13.332048
15 15     C 11.332477
```

```
### 実行
anovakun(dat, "As", long = TRUE, peta = TRUE)
```

```
[ As-Type Design ]

This output was generated by anovakun 4.8.9 under R version 4.4.1.
It was executed on Thu Jul  4 08:21:48 2024.


<< DESCRIPTIVE STATISTICS >>

------------------------------
 group   n     Mean    S.D.
------------------------------
     A   5   7.5807  2.4331
     B   4  10.1681  3.9548
     C   6  13.5469  1.9483
------------------------------


<< ANOVA TABLE >>

== This data is UNBALANCED!! ==
== Type III SS is applied. ==

-------------------------------------------------------------
 Source       SS  df      MS  F-ratio  p-value      p.eta^2
-------------------------------------------------------------
  group  98.3840   2 49.1920   6.5897   0.0117 *     0.5234
  Error  89.5804  12  7.4650
-------------------------------------------------------------
  Total 187.9644  14 13.4260
                 +p < .10, *p < .05, **p < .01, ***p < .001


<< POST ANALYSES >>

< MULTIPLE COMPARISON for "group" >

== Shaffer's Modified Sequentially Rejective Bonferroni Procedure ==
== The factor < group > is analysed as independent means. ==
== Alpha level is 0.05. ==

------------------------------
```

```
 group   n     Mean    S.D.
------------------------------
     A   5   7.5807  2.4331
     B   4  10.1681  3.9548
     C   6  13.5469  1.9483
------------------------------


-----------------------------------------------------------
 Pair     Diff  t-value  df       p   adj.p
-----------------------------------------------------------
  A-C  -5.9662   3.6062  12  0.0036  0.0108  A < C *
  B-C  -3.3789   1.9159  12  0.0795  0.0795  B = C
  A-B  -2.5873   1.4117  12  0.1834  0.1834  A = B
-----------------------------------------------------------


output is over ---------------------///
```

出力結果は大きく分けて記述統計<< DESCRIPTIVE STATISTICS >>と，分散分析表<< ANOVA TABLE >>，下位検定<< POST ANALYSES >>に分けられる。記述統計はデータが正しく読み込めているかどうかのチェックに使おう。

一番のメインは分散分析表であり，平方和 sum of squares を自由度 df で割った，1 自由度あたりのデータの散らばりを，群間と群内 (誤差) との比で検証しているのが見て取れる。群間平方和が 98.38，群内平方和が 89.58 であり，それぞれ自由度 2(3 水準 -1) と 12($\sum_{j=1}^{3} n_j - 1$) から生じているので，平均平方 Mean Squares がそれぞれ 49.19 と 7.47 である。この比が 6.5897 で，自由度 $F(2, 12)$ の F 分布においてこの値以上の極端な数字が出る確率が 5% を下回っている (実に $p = 0.0117$ である) ため，統計的に有意であると判断できる。分散分析表の Total のところで，全体の SS が群間 SS+ 群内 SS に一致していること，自由度も全体 df=群間 df+ 群内 df になっていることを確認しておこう。

また，anovakun 関数の引数として peta = TRUE を指定したが，これは偏 η^2(partial eta) と呼ばれる効果量を出力するためのオプションである。

今回は分散分析の時点で統計的な有意差が認められたため ($F(2, 12) = 6.59, p < 0.05, \eta^2 = 0.52$)，続いて下位検定が表示されている。ANOVA 君は下位検定についても複数のオプションを持っているが，デフォルトでは Shaffer の修正 Bonferroni 検定が行われる。詳しくは専門書 (永田・吉田, 1997) を参照してほしいが，概略を説明すると，検証すべき仮説の数で有意水準を分割するという Bonferroni の方法を，競合する仮説の数も考慮して分母を調整するというものである。

この計算の結果，A 群と C 群の間にのみ統計的な有意差が確認された ($t(12) = 3.61, p < 0.05$) と言える。

9.4.2 2way-ANOVA

二要因の場合も見ておこう。ANOVA 君の表記方法は要因計画の型が変わるだけで大きな変更はないが，交互作用 interaction を考える必要があるところがポイントである。これも仮想データの組成を見ることでその意義がわかりやすくなるだろう。間 2× 間 2 の実験デザインを例に，まずは各水準の理論的平均値がどのようにつくられるかをみておこう。

```
set.seed(123)
# 各群のサンプルサイズ
n <- 10
# 全体平均，効果量，母 SD
mu <- 10
delta1 <- 1
delta2 <- 0 # ここではあえて要因 B の効果を 0 にしている
delta3 <- 2
sigma <- 3
# 効果の計算
effectA <- delta1 * sigma # Factor A
effectB <- delta2 * sigma # Factor B
effectAB <- delta3 * sigma # interaction
# 各群の平均
mu11 <- mu + effectA + effectB + effectAB
mu12 <- mu + effectA - effectB - effectAB
mu21 <- mu - effectA + effectB - effectAB
mu22 <- mu - effectA - effectB + effectAB
```

効果の現れ方は相対的だから，要因 A が第一水準に `+effectA` の形で現れたら，第二水準には`-effectA` の形で現れる。要因 B についても同様である。交互作用については組み合わせにおいて生じるから，要因 A の第一水準と要因 B の第一水準の組み合わせのところに `+effectAB` を充てる。ここでも効果は相対的に現れるという条件を守るために，要因 A の第一水準の中で `+effectAB` の効果を相殺するために，要因 A の第一水準と要因 B の第二水準の組み合わせの符号が反転する。同様に，要因 B の第一水準の中で相殺するために要因 A の第二水準と要因 B の第一水準には`-effectAB` が加わる。

このようにして考えられる理論的平均値に対して，外乱要因である誤差が生じて実現値が得られる。組み上げて得られたデータを確認しておこう。

```
X11 <- rnorm(n, mean = mu11, sd = sigma)
X12 <- rnorm(n, mean = mu12, sd = sigma)
X21 <- rnorm(n, mean = mu21, sd = sigma)
X22 <- rnorm(n, mean = mu22, sd = sigma)
dat <- data.frame(
  ID = 1:(n * 4),
```

```
  FactorA = rep(1:2, each = n * 2),
  FactorB = rep(rep(1:2, each = n), 2),
  value = c(X11, X12, X21, X22)
)
dat
```

```
   ID FactorA FactorB      value
1   1       1       1 17.3185731
2   2       1       1 18.3094675
3   3       1       1 23.6761249
4   4       1       1 19.2115252
5   5       1       1 19.3878632
6   6       1       1 24.1451950
7   7       1       1 20.3827486
8   8       1       1 15.2048163
9   9       1       1 16.9394414
10 10       1       1 17.6630141
11 11       1       2 10.6722454
12 12       1       2  8.0794415
13 13       1       2  8.2023144
14 14       1       2  7.3320481
15 15       1       2  5.3324766
16 16       1       2 12.3607394
17 17       1       2  8.4935514
18 18       1       2  1.1001485
19 19       1       2  9.1040677
20 20       1       2  5.5816258
21 21       2       1 -2.2034711
22 22       2       1  0.3460753
23 23       2       1 -2.0780133
24 24       2       1 -1.1866737
25 25       2       1 -0.8751178
26 26       2       1 -4.0600799
27 27       2       1  3.5133611
28 28       2       1  1.4601194
29 29       2       1 -2.4144108
30 30       2       1  4.7614448
31 31       2       2 14.2793927
32 32       2       2 12.1147856
33 33       2       2 15.6853770
34 34       2       2 15.6344005
```

```
35 35       2       2 15.4647432
36 36       2       2 15.0659208
37 37       2       2 14.6617530
38 38       2       2 12.8142649
39 39       2       2 12.0821120
40 40       2       2 11.8585870
```

もちろん実際には，計画に応じたデータセットが得られているはずであり，各群のサンプルサイズが異なるなどの事情もあるだろう。しかしこうして，理論的にデータの組成を見ておくことで，サンプルサイズを変えたり効果量を変えたりしながら，どのように結果が変わってくるかを確認しながら進めることができる[*2]。

それではこの仮想データを分析してみよう。

```
anovakun(dat, "ABs", long = TRUE, peta = TRUE)
```

```
[ ABs-Type Design ]

This output was generated by anovakun 4.8.9 under R version 4.4.1.
It was executed on Thu Jul  4 08:21:48 2024.


<< DESCRIPTIVE STATISTICS >>

----------------------------------------
 FactorA  FactorB   n     Mean    S.D.
----------------------------------------
       1        1  10  19.2239  2.8614
       1        2  10   7.6259  3.1142
       2        1  10  -0.2737  2.7924
       2        2  10  13.9661  1.5819
----------------------------------------


<< ANOVA TABLE >>

------------------------------------------------------------------------------
           Source        SS  df        MS  F-ratio  p-value      p.eta^2
```

[*2] かつては分散分析は手計算でできる分析モデルであり，得られたデータを平方和に分解していくプロセスをたどりながら分散分析のメカニズムが体得されるという教育が多く見られた。ただしその方法は計算に時間がかかること，ミスが混在しやすいことに加え，手元のデータが唯一無二のものであるという印象を強くすることが懸念される。推測統計学においては，手元のデータはあくまでも実現値に過ぎないと考えるのであり，乱数を生成して幾つでも自在に作り出せる経験を得た方が教育効果として良いのではないか，と筆者は考えている。

```
------------------------------------------------------------------------
          FactorA  432.7854   1  432.7854  61.4190   0.0000 ***   0.6305
          FactorB   17.4478   1   17.4478   2.4761   0.1243 ns    0.0644
FactorA x FactorB 1668.9825   1 1668.9825 236.8545   0.0000 ***   0.8681
            Error  253.6721  36    7.0464
------------------------------------------------------------------------
            Total 2372.8878  39   60.8433
                           +p < .10, *p < .05, **p < .01, ***p < .001


<< POST ANALYSES >>

< SIMPLE EFFECTS for "FactorA x FactorB" INTERACTION >

------------------------------------------------------------------------
      Source        SS  df        MS  F-ratio  p-value      p.eta^2
------------------------------------------------------------------------
FactorA at 1 1900.7730   1 1900.7730 269.7492   0.0000 ***   0.8823
FactorA at 2  200.9950   1  200.9950  28.5243   0.0000 ***   0.4421
FactorB at 1  672.5693   1  672.5693  95.4480   0.0000 ***   0.7261
FactorB at 2 1013.8610   1 1013.8610 143.8826   0.0000 ***   0.7999
       Error  253.6721  36    7.0464
------------------------------------------------------------------------
                         +p < .10, *p < .05, **p < .01, ***p < .001

output is over --------------------///
```

基本的な結果の見方については，一要因のときと同じである。今回は要因 A と交互作用の効果を作り，正しく検出されている。下位検定については，要因 A が 2 水準であったためこちらの主効果の検証は必要なく (記述統計を見て群平均比較をすればよい)，交互作用についての単純効果の検証が行われている。

9.5　Within デザイン

Within デザインは対応のある t 検定の時と同じように，多次元正規分布からの生成として考えよう。すなわち各個体から得られるデータが相関しているという仮定をおくのである。以下のサンプルコードを読んで，データ生成過程を確認しよう。なお共分散は $\rho_{xy} = \frac{s_{xy}}{s_x s_y}$ より $s_{xy} = \rho_{xy} s_x s_y$ として整形している。

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
```

```
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
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
```

```
library(MASS)
```

```
次のパッケージを付け加えます: 'MASS'
```

```
以下のオブジェクトは 'package:dplyr' からマスクされています:
```

```
    select
```

```
set.seed(42)
# 各群のサンプルサイズ
n <- 10
# 全体平均，効果量，母 SD
mu <- 10
delta <- 1
s1 <- s2 <- s3 <- 1
rho12 <- 0.1
rho13 <- 0.3
rho23 <- 0.8
mus <- c(mu, mu + s1 * delta, mu - s1 * delta)
# 共分散行列の生成
Sigma <- matrix(NA, ncol = 3, nrow = 3)
Sigma[1, 1] <- s1^2
Sigma[2, 2] <- s2^2
Sigma[3, 3] <- s3^2
Sigma[1, 2] <- Sigma[2, 1] <- rho12 * s1 * s2
Sigma[1, 3] <- Sigma[3, 1] <- rho13 * s1 * s3
Sigma[2, 3] <- Sigma[3, 2] <- rho23 * s2 * s3
# データの生成
X <- mvrnorm(n, mus, Sigma) %>% as.data.frame()
# データの確認
X
```

```
          V1        V2       V3
```

```
1  10.579459  9.246155 7.676967
2  12.171646 10.248713 9.873523
3   8.578778 11.085783 8.825757
4   9.572344 10.864083 8.058858
5   9.850954 11.212887 8.059800
6  10.587381 10.763451 9.175102
7   9.087450  9.661022 7.665158
8   7.506689 11.471591 9.846169
9   6.978510 10.122028 7.153938
10 11.171906 10.438344 9.137808
```

```
# Long 型に整形
X <- X %>%
  rowid_to_column("ID") %>%
  pivot_longer(-ID) %>%
  print()
```

```
# A tibble: 30 x 3
      ID name  value
   <int> <chr> <dbl>
 1     1 V1    10.6
 2     1 V2     9.25
 3     1 V3     7.68
 4     2 V1    12.2
 5     2 V2    10.2
 6     2 V3     9.87
 7     3 V1     8.58
 8     3 V2    11.1
 9     3 V3     8.83
10     4 V1     9.57
# i 20 more rows
```

```
# 分析の実行
anovakun(X, "sA", long = TRUE, peta = TRUE, GG = TRUE)
```

```
[ sA-Type Design ]

This output was generated by anovakun 4.8.9 under R version 4.4.1.
It was executed on Thu Jul  4 08:21:49 2024.
```

```
<< DESCRIPTIVE STATISTICS >>

-----------------------------
 name   n     Mean    S.D.
-----------------------------
   V1  10   9.6085  1.6207
   V2  10  10.5114  0.7057
   V3  10   8.5473  0.9557
-----------------------------


<< SPHERICITY INDICES >>

== Mendoza's Multisample Sphericity Test and Epsilons ==

-------------------------------------------------------------------------
 Effect  Lambda  approx.Chi  df      p         LB     GG     HF     CM
-------------------------------------------------------------------------
   name  0.0213      6.8465   2 0.0326 *   0.5000 0.6349 0.6920 0.6005
-------------------------------------------------------------------------
                             LB = lower.bound, GG = Greenhouse-Geisser
                            HF = Huynh-Feldt-Lecoutre, CM = Chi-Muller


<< ANOVA TABLE >>

-------------------------------------------------------------
  Source      SS  df     MS  F-ratio  p-value      p.eta^2
-------------------------------------------------------------
       s 15.7129   9 1.7459
-------------------------------------------------------------
    name 19.3302   2 9.6651   8.4330   0.0026 **    0.4837
s x name 20.6299  18 1.1461
-------------------------------------------------------------
   Total 55.6729  29 1.9198
                +p < .10, *p < .05, **p < .01, ***p < .001


<< POST ANALYSES >>

< MULTIPLE COMPARISON for "name" >
```

```
== Shaffer's Modified Sequentially Rejective Bonferroni Procedure ==
== The factor < name > is analysed as dependent means. ==
== Alpha level is 0.05. ==

-----------------------------
 name   n     Mean    S.D.
-----------------------------
   V1  10   9.6085  1.6207
   V2  10  10.5114  0.7057
   V3  10   8.5473  0.9557
-----------------------------

------------------------------------------------------------
  Pair     Diff  t-value  df       p   adj.p
------------------------------------------------------------
 V2-V3   1.9641   7.2559   9  0.0000  0.0001  V2 > V3 *
 V1-V3   1.0612   2.1630   9  0.0588  0.0588  V1 = V3
 V1-V2  -0.9029   1.4770   9  0.1738  0.1738  V1 = V2
------------------------------------------------------------


output is over --------------------///
```

上記コードについていくつか解説をしておこう。今回，各群の分散は同じにしつつ，変数間相関に大きな違いを持たせた。あえて球面性の仮定が成立しないような例を得たかったからで，出力の`<< SPHERICITY INDICES >>`をみると統計量 λ のあとの p 値が 5% を下回っており，「球面性が成立している」という帰無仮説が棄却されていることがわかる。この時いくつかの補正法があるが，今回は Greenhouse-Geisser の補正を当てることにしている。それが anovakun 関数のなかの `GG=TRUE` の箇所である。

これを踏まえて分散分析表が示されている。ここでも全体平方和 `SS`，自由度 `df` が各行の要素の和になっていることが確認できるが，その因子名のところに `s` が含まれていることが確認できる。これが個体ごとの変動を表しており，誤差から個人差を取り除いて効果の検証ができていることがわかる。

分散分析は加法的，線形的な分解であるからわかりやすく，要因が複雑に組み合わさることがあっても基本的に今回のパーツを組み上げることで理解できる。言い換えると，データが先にある実践的な場合には平方和をひとつひとつ丁寧に紐解いていくことで理解できる。実に anovakun の前進である anova4 では 4 要因，anovakun では 26 要因までのデザインを分析することが可能である。もっとも 4 要因計画にもなると 3 次の交互作用まで考えられ，主効果と合わせてこれらの交互作用効果を解釈するのは困難である。anoakun は 2 次以上の交互作用が見られた場合，自動的に下位検定を行ってくれないので，要因の水準ごとにデータを分割して，分散分析表を解体しつつ分析する必要がある。[*3]

[*3] anovakun の補助関数 anovatan を用いることで，注目したい要因ごとにデータを分割してくれる。詳しくは公式サイトのマニュアル

しかしもちろん，これには検定の多重性の問題が関わってくるから，あまり推奨される手法ではない。ごく少ない要因で，主効果の有無を検証することを主眼においた丁寧な実験デザインを組み立てることを試みるべきである。

またここでは，仮想データを作ることで，得られたデータの背後にある生成メカニズムに注目した。「与えられたデータを分解する」のが分散分析であるのに対し，リバースエンジニアリングからアプローチしたのである。こうすることで，分散分析の見えない仮定に注意が向くことを期待している。簡便のために，いくつかのパラメータを均質化するなどしたが，実践的には群ごとのサンプルサイズが異なることも少なくないだろうし，群間の分散や共分散が均質であることを前提とするのは難しいだろう。これを考慮した細かい作り込みも，リバースエンジニアリングによって生成メカニズムがわかっている場合には応用が可能である。さらに，どこの水準間にどのような効果があると仮定されるか，といった精緻な仮説があるのなら，そこだけをターゲットにした分析を行うことも可能である。

分散分析は，あくまでも大雑把な全体的傾向を見るためのものであることに留意しよう。心理学のデータがより精緻な仮定に耐えうる精度を持つものになれば，分散分析は過去の遺物となるかもしれない。

9.6 課題

1. 以下のデータセットは一要因 4 水準 Between 計画で得られたものです。分散分析を行って，要因の効果があるかどうか，水準間に差があるとすればどこに見られるかを報告してください。なおこのデータセットはこちら ex_anova1.csv からダウンロード可能です。

	ID	group	value
1	1	A	14.37
2	2	A	15.11
3	3	A	16.11
4	4	A	11.17
5	5	A	14.51
6	6	A	7.85
7	7	A	10.65
8	8	B	16.45
9	9	B	11.76
10	10	B	19.11
11	11	B	19.62
12	12	C	2.92
13	13	C	6.27
14	14	C	1.82
15	15	C	-0.10
16	16	C	5.30
17	17	C	1.57
18	18	D	8.33

を参照してほしい。

```
19 19     D  2.71
20 20     D  5.97
21 21     D  4.97
22 22     D  1.65
23 23     D  8.73
24 24     D  5.93
25 25     D  4.27
```

2. 以下のデータセットは一要因 4 水準 Within 計画で得られたものです。分散分析を行って，要因の効果があるかどうか，水準間に差があるとすればどこに見られるかを報告してください。なおこのデータセットはこちら ex_anova2.csv からダウンロード可能です。

```
      V1    V2    V3    V4
1  11.66 13.33  9.89 -0.99
2  11.14 14.21 15.34  2.60
3  10.90 13.31 14.25 -0.03
4   8.72 11.57 14.24  0.72
5  11.03 12.85 12.77  1.35
6   9.80 16.71 12.86  1.92
7   9.40 14.31  9.16 -0.52
8  12.33 11.49 14.94  1.85
9  10.59 11.38 10.15 -2.58
10  9.85 15.34 13.08  2.29
11 10.55 11.91 11.59 -0.93
12  7.20  8.67 11.19 -1.68
```

3. 問 (3)× 問 (3) の分散分析モデルの仮想データセットを作りましょう。そのデータに分散分析を適用し，仮定した要因の効果がみられるか (あるいは効果がないと仮定した場合に正しく検出されないか) を確認しましょう。
4. 【発展課題】二要因混合計画分散分析 (間 × 内) の仮想データセットを作りましょう。そのデータに分散分析を適用し，仮定した要因の効果がみられるか (あるいは効果がないと仮定した場合に正しく検出されないか) を確認しましょう。

第 10 章

疑わしき研究実践とサンプルサイズ設計

ここまでシミュレーションを通じて仮想データを生成し，帰無仮説検定のステップをリバースエンジニアリングしながら検定を「モデル」の観点から確認してきた。

シミュレーションは仮想世界を作ることであり，いかようにもデータを作ることができるのだから，たとえば実践的にタブーとされていることを仮想的に検証してみることができる。このアプローチで，QRPs が具体的にどのように問題になるのかを体験してみよう。

10.1 疑わしき研究実践 Questionable Research Practicies

10.1.1 検定の繰り返し

帰無仮説検定は確率を伴った判断なので,「差がないのにあると判断してしまった (タイプ 1 エラー)」とか,「差があるのに検出できなかった (タイプ 2 エラー)」といった問題が生じうる。すでに述べたように，タイプ 2 エラーの方は本質的に知り得ないので (差がどの程度あるか，事前にわかっていることがない)，せめてタイプ 1 エラーは制御することを目指すことになる。

こうした検定は合理的に行われるべきもので，なんとか有意に「したい」といった研究者のお気持ちとは独立しているはずである。しかし (もしかすると) 意図せぬところで，この制御に失敗してしまっている可能性がある。

ひとつは検定の繰り返しに関する問題である。たとえば分散分析において,「主効果が出てから下位検定で各ペアの検証をするんだから，最初から各ペアの t 検定を繰り返せばいいじゃないか」と考える人がいるかもしれない。これで本当に問題ないのか，シミュレーションで確認してみよう。

以下のコードは，有意差のないデータセットを作り，1. 分散分析を行なって有意になるかどうか，2. 各ペアについて繰り返し t 検定を行い，どこかに有意差が検出されるかどうか，を比較している。分散分析は ANOVA 君ではなく，R 固有の `aov` 関数を用いた[*1]。また,「どこかに有意差が検出される」を `if` 文を使って書いているところを，注意深く確認しておいてほしい。

[*1] ANOVA 君は結果をコンソールに直接出力し，戻り値を持たない。ここでは結果の p 値が必要だったので，このようにした。

```r
library(tidyverse)
library(broom) # 分析結果を tidy に整形するパッケージ。ない場合は install しておこう
```

```r
alpha <- 0.05 # 有意水準を 0.05 に設定
n1 <- n2 <- n3 <- 10 # 各グループのサンプルサイズを 10 に設定
mu <- 10 # 平均値を 10 に設定
sigma <- 2 # 標準偏差を 2 に設定

mu1 <- mu2 <- mu3 <- mu # 各グループの平均値を同じに設定

set.seed(12345) # 乱数のシードを設定して再現性を確保
iter <- 1000 # シミュレーションの繰り返し回数を 1000 に設定

anova.detect <- rep(NA, iter) # ANOVA 検出結果の保存用ベクトルを初期化
ttest.detect <- rep(NA, iter) # t 検定検出結果の保存用ベクトルを初期化

for (i in 1:iter) { # 1000 回のシミュレーションを繰り返すループ
  X1 <- rnorm(n1, mu, sigma) # グループ 1 のデータを生成
  X2 <- rnorm(n2, mu, sigma) # グループ 2 のデータを生成
  X3 <- rnorm(n3, mu, sigma) # グループ 3 のデータを生成

  dat <- data.frame( # データフレームを作成
    group = c(rep(1, n1), rep(2, n2), rep(3, n3)), # グループ番号を追加
    value = c(X1, X2, X3) # データを追加
  )
  result.anova <- aov(value ~ group, data = dat) %>% tidy() # ANOVA を実行し結果を整形
  anova.detect[i] <- ifelse(result.anova$p.value[1] < alpha, 1, 0) # 有意差があるかを判定して保存

  # t 検定を繰り返す
  ttest12 <- t.test(X1, X2)$p.value # グループ 1 と 2 の t 検定
  ttest13 <- t.test(X1, X3)$p.value # グループ 1 と 3 の t 検定
  ttest23 <- t.test(X2, X3)$p.value # グループ 2 と 3 の t 検定

  ttest.detect[i] <- ifelse(ttest12 < alpha | ttest13 < alpha | ttest23 < alpha, 1, 0) # いずれかの t 検
}

ttest.detect %>% mean() # t 検定で有意差が検出された割合を計算
```

```
[1] 0.109
```

```
anova.detect %>% mean() # ANOVA で有意差が検出された割合を計算
```

```
[1] 0.04
```

結果を見ると，t 検定で有意差が検出された確率が 0.109 であり，設定した α 水準を大きく上回っていることがわかる。有意でないところに有意差を見出しているのだから，これはタイプ 1 エラーのインフレである。分散分析で検出された結果は 0.04 であり，正しく α 水準がコントロールできている。

検定を繰り返すことの問題は，確率的判断にある。5% の水準でタイプ 1 エラーが起こるということは，95% の確率で正しく判断できるということだが，2 回検定を繰り返すとその精度は $(1-0,05)^2 = 0.9025$ であり，3 回検定を繰り返すと $(1-0.05)^3 = 0.857375$ と，どんどん小さくなっていってしまう。検定はタイプ 1 エラーのハンドリングが目的であったことを忘れてはならない。

10.1.2 ボンフェローニの方法

一つの論文のなかに複数の研究 (Study1, Study2,...) があり，それぞれで検定による確率的判断を行っているとしよう。それぞれ別のデータセットに対する検定であっても，一つの露文の中で確率的判断が繰り返されていることに違いはない。このような場合は，どのようにして有意水準をコントロールすれば良いのだろうか。

最も単純明快な方法のひとつは，分散分析の下位検定でもみられた Bonferroni の補正である。すなわち，検定の回数で有意水準を割ることで，検定を厳しくするのである。5% 水準の検定を 5 回繰り返すのなら，$0.05/5 = 0.01$ とすることで全体的なタイプ 1 エラー率を抑制するのである。これが正しく機能するかどうか，シミュレーションで確認してみよう。

反復してデータを生成することになるので，仮想データ生成関数を別途事前に準備しておこう。

```
# シミュレーション用の関数を定義
studyMake <- function(n, mu, sigma, delta) {
  X1 <- rnorm(n, mu, sigma) # グループ 1 のデータを生成
  X2 <- rnorm(n, mu + sigma * delta, sigma) # グループ 2 のデータを生成（平均値が異なる）
  dat <- data.frame( # データフレームを作成
    group = rep(1:2, each = n), # グループ番号を追加
    value = c(X1, X2) # データを追加
  )
  result <- t.test(X1, X2)$p.value # グループ間の t 検定を実行
  return(result) # p 値を返す
}
```

この関数は，引数としてサンプルサイズ `n`，平均値 `mu`，標準偏差 `sigma`，効果量 `delta` をとり，2 群の t 検定の結果である p 値を返す関数である。

```
# 使用例；t 検定の結果の p 値が戻ってくる
studyMake(n = 10, mu = 10, sigma = 1, delta = 0)
```

```
[1] 0.9444895
```

これ一回で 1 分析するので，これを複数回行って一つの研究とし，一つの論文のなかで num_studies 回の研究を行ったとしよう。今回は num_studies = 3 としている。R の replicate 関数で研究回数繰り返した p 値ベクトルを得て，どこかに差が検出されるかどうかをチェックする。「どこかに」を表現するために any 関数を使って判定する。判定する有意水準として，α と補正をかけた α_{adj} の 2 つを用意した。

```
set.seed(12345) # 乱数のシードを設定して再現性を確保
iter <- 1000 # シミュレーションの繰り返し回数を 1000 に設定
alpha <- 0.05 # 有意水準を 0.05 に設定
num_studies <- 3 # 研究の数を 3 に設定
alpha_adjust <- alpha / num_studies # 多重検定補正後の有意水準を計算

FLG.detect <- rep(NA, iter) # 検出結果を保存するベクトルを初期化
FLG.detect.adj <- rep(NA, iter) # 補正後の検出結果を保存するベクトルを初期化
for (i in 1:iter) { # 1000 回のシミュレーションを繰り返すループ
  p_values <- replicate(num_studies, studyMake(n = 10, mu = 10, sigma = 1, delta = 0)) # 各研究の p 値を
  FLG.detect[i] <- ifelse(any(p_values < alpha), 1, 0) # 補正前の有意差検出を判定して保存
  FLG.detect.adj[i] <- ifelse(any(p_values < alpha_adjust), 1, 0) # 補正後の有意差検出を判定して保存
}

FLG.detect %>% mean() # 補正前の有意差検出率を計算
```

```
[1] 0.145
```

```
FLG.detect.adj %>% mean() # 補正後の有意差検出率を計算
```

```
[1] 0.049
```

結果を見ると，α 水準のまま検定を行うと，論文全体でのタイプ 1 エラー率が 0.145 と 5% を上回っており，3 つの研究のどこかで間違った判断をしていることがわかる。補正すると 0.049 と正しく制御されている。

一連の研究をまとめた一つの論文に，複数の研究が含まれていることは少なくない。各検定結果をまとめて総合考察とすることも一般的である。総合考察は各分析結果から全体的な結論を導くのだが，その要素のどこかに間違いがあると，全体の論立てが崩れてしまうことにもなりかねない。いわば腐った支柱が紛れ込んでいる土台の上に家屋を建てるようなもので，研究の積み重ねを目的とする科学活動の一環である以上，正しく制御されていることは重要である。

10.1.3　N 増し問題

人間を対象にした研究を行って，データを一生懸命取る。その結果，効果があると見られた操作/介入から統計的な有意差が検出されなければ，「悔しい」という心情になることは理解できる。もう少し実験を工夫すれば，もう少しデータが違えばよかったのでは，と思うかもしれない。ではもう少し**頑張って**データを増やしてみればどうだろうか。

実はこの考え方は QRPs のひとつである。検定は真偽判定をする競技のようなものなので，ゲームの途中でプレイヤーの人数が変わるのはよろしくない。このことをシミュレーションで確認してみよう。

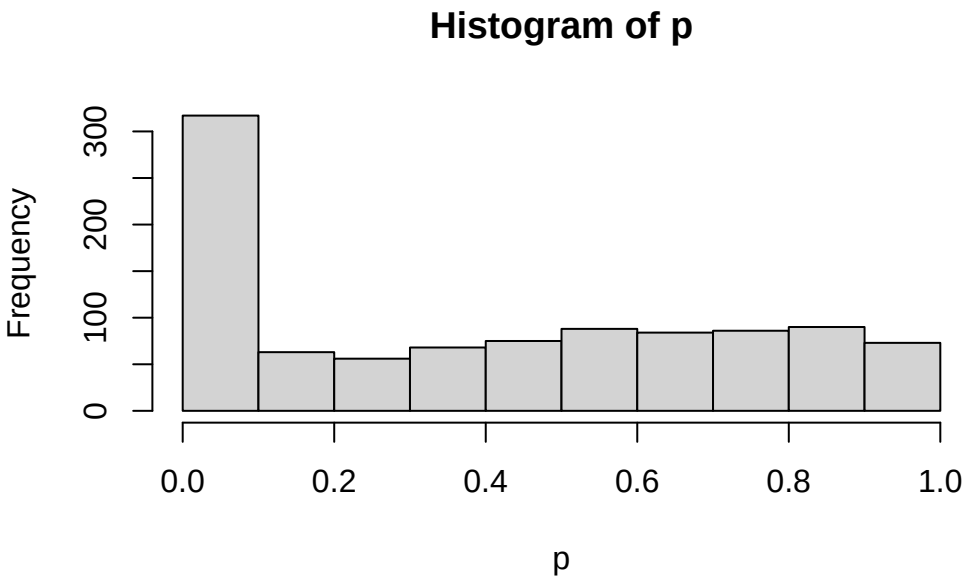
以下のコードは，t 検定のデータを最初 `n1=n2=10` で作成して行っている。タイプ 1 エラーの検証をするので，効果量は 0 である。ここで t 検定を行い，もしその p 値が α よりも大きかったら，つまり有意であると判断されなかったら，同じ方法でデータを 1 件追加する。そしてまた t 検定を行う。このサンプルの追加は，効果量 0 なので，偶然のお許しが出るまでいつまで経っても終わることがないため，上限を 100 にしてある。上限に達したら流石に諦めてもらうとして，さてそうした QRPs な努力の結果，α 水準はどれぐらいに保たれているだろうか。

```
iter <- 1000 # シミュレーションの繰り返し回数を 1000 に設定
alpha <- 0.05 # 有意水準を 0.05 に設定
p <- rep(0, iter) # p 値を保存するベクトルを初期化
add.vec <- rep(0, iter) # 増やした人数を保存するベクトルを初期化

set.seed(123) # 乱数のシードを設定して再現性を確保

n1 <- n2 <- 10 # 各グループのサンプルサイズを 10 に設定
mu <- 10 # 平均値を 10 に設定
sigma <- 2 # 標準偏差を 2 に設定
delta <- 0 # 平均の差を 0 に設定

## シミュレーション本体
for (i in 1:iter) { # 1000 回のシミュレーションを繰り返すループ
  # 最初のデータを生成
  Y1 <- rnorm(n1, mu, sigma) # グループ 1 のデータを生成
  Y2 <- rnorm(n2, mu + sigma * delta, sigma) # グループ 2 のデータを生成
  p[i] <- t.test(Y1, Y2)$p.value # t 検定を実行し p 値を保存
  # データを追加する
  count <- 0 # 追加したデータの数をカウント
  ## p 値が 5% を下回るか、データが 100 になるまでデータを増やし続ける
  while (p[i] >= alpha && count < 100) { # 条件を満たすまでループを繰り返す
    # 有意でなかった場合、変数ごとに 1 つずつデータを追加
    Y1_add <- rnorm(1, mu, sigma) # グループ 1 に新しいデータを 1 つ追加
    Y2_add <- rnorm(1, mu + sigma * delta, sigma) # グループ 2 に新しいデータを 1 つ追加
    Y1 <- c(Y1, Y1_add) # グループ 1 のデータを更新
    Y2 <- c(Y2, Y2_add) # グループ 2 のデータを更新
    p[i] <- t.test(Y1, Y2)$p.value # 新しいデータで t 検定を再度実行し p 値を更新
    count <- count + 1 # データを追加した回数をカウント
  }
  add.vec[i] <- count
}
```

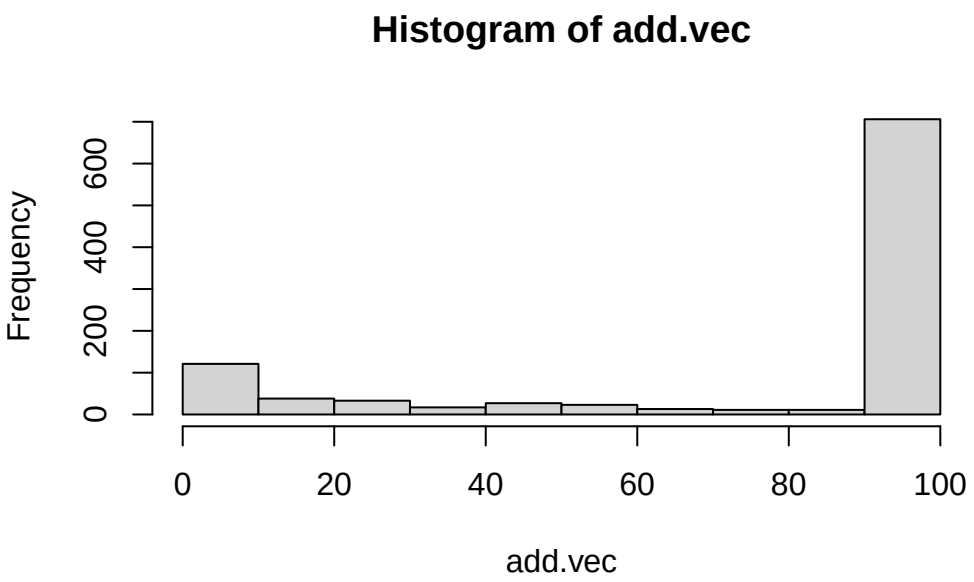
```
## 結果
ifelse(p < 0.05, 1, 0) |> mean() # p 値が 5% 未満の割合を計算
```

```
[1] 0.306
```

```
hist(p)
```




```
hist(add.vec)
```



結果をみると，0.306 とかなり逸脱して，誤った結論に辿り着いていることがわかる。努力の結果得られた有意差は，偶然の賜物でもあり，誤った研究実践による幻想にすぎない。加えたデータのヒストグラムからわかるように，悲しいかな，75% もの割合で上限 100 まで達してしまう。百害あって一利なしとはこのことである。

10.1.4 サンプルサイズを事前に決めないことの問題

サンプルサイズを事前に決めずに検定する，という状況を別の角度から見てみよう。Kruschke (2014 前田・小杉監訳 2017) は「コインフリップを 24 回して，うち 7 回表が出た」というシーンを例に挙げて説明している。7/24 は半分を下回っているから，やや裏が出やすいコインであるように思える。帰無仮説として，このコインは公平である (表と裏が出る確率が半々である)，というのを検証したいとする。

この 24 回中 7 回成功，という話の背後に「24 施行する」ということを決めていたかどうか (サンプルサイズを事前に決めていたか) ということを考えてみよう。

まずは正直に，最初から 24 回コインフリップすることを決めていたとする。コインフリップはベルヌーイ試行[*2]であり，それを繰り返すので二項分布に従うと考えられる。そこで，二項検定として次のように計算できるだろう。

```
N <- 24
# 7 回表が出る確率
pbinom(7, N, 0.5) * 2
```

```
[1] 0.06391466
```

二項分布の p 値を出すには pbinom を使いった。また帰無仮説として，このコインフリップは公平であると考えているのだから，$\theta = 0.5$ が帰無仮説の状態である。この $\theta = 0.5$ とした時に，$N = 24, k = 7$ という結果になる確率を計算し，かつ両側検定 (公平でない，が対立仮説なので裏が 7 回でもよい) であることを考えて確率を 2 倍した。p 値は 0.0639147 であるから，5% 水準では有意であると判定できない。これぐらいの確率はあるということだ。

しかしここで第二の状況を考えてみよう。24 回コインフリップすることを決めていたのではなくて，7 回成功するまでコインフリップを続けたところ，結果的に 24 回で終わったということだった，とするのである。このようなシーンの確率分布は負の二項分布と呼ばれ，pnbinom で次のように計算できる。

```
k <- 7
# 24 回以上必要な確率
pnbinom(k, 24, 0.5) * 2
```

```
[1] 0.003326893
```

この結果から，$\theta = 0.5$ の時に 7 回表がでるまでに 24 施行も必要とする確率は，0.0033269 だから，5% 水準で有意である。つまり，滅多にこんなことが起きないので，$\theta = 0.5$ という帰無仮説が疑わしいことになる。ここではシーンが異なると p 値が違っている，ということに注意してほしい。

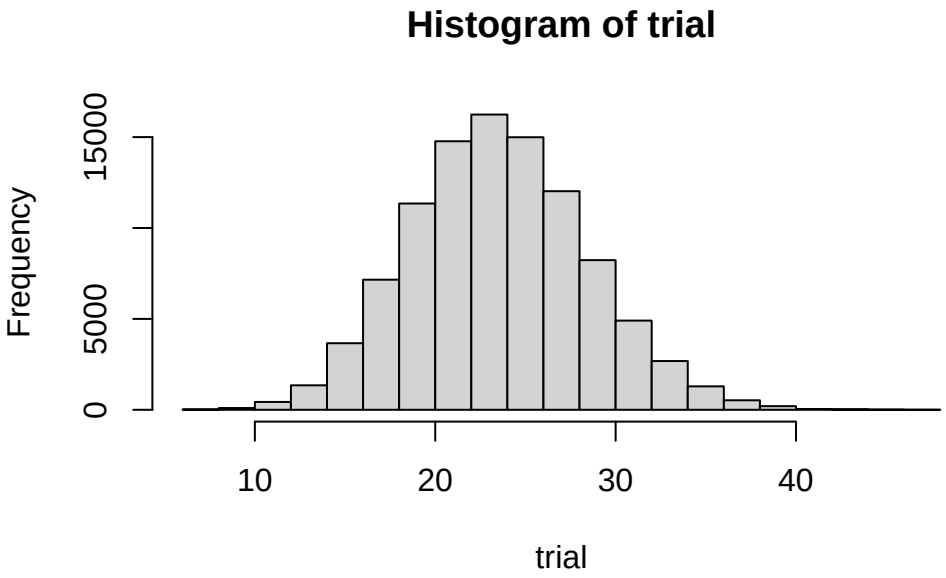
さらに第 3 のシーンを考えよう。これは「何回やるかは決めてないけど，まあ 5 分ぐらいかな」と試行にかける時間だけ決めていたという状況である。結果的に 24 回になったけど，もしかすると 23 回だったかもしれないし，25 回や 20 回，30 回だったかもしれない。これをシミュレーションするために，「24 がピークになるような頻度の

[*2] 表 (1) が出るか，裏 (0) が出るか，という 2 値の結果変数だけを持つ施行のことで，この確率変数がベルヌーイ分布にし違う。ベルヌーイ分布は，表が出る確率 θ をパラメータに持つ。$P(X = k) = \theta^k(1-\theta)^{1-k}$, ただし $k = \{0, 1\}$ という確率変数である。1/0 というのが生死，男女，成功失敗などさまざまなメタファに適用できるので応用範囲が広い。

分布」をポアソン分布を使って生成する [^10:3]。

[^10:3] ポアソン分布は正の整数を実現値に取る分布で，カウント変数の確率分布として用いられる。パラメータは λ だけであり，期待値と分散が λ に一致する，非常にシンプルな分布である。

```
set.seed(12345)
iter <- 100000 # 発生させる乱数の数
## 24 回がピークに来るトライアル回数
trial <- rpois(iter, 24)
hist(trial)
```



この各トライアルにおいて，二項分布で成功した回数を計算し，トライアル回数で割ることによって，表が出る確率のシミュレーションができる。その時の割合は，7/24 よりもレアな現象だろうか?

```
result <- rep(NA, iter)
for (i in 1:iter) {
  result[i] <- rbinom(1, trial[i], 0.5) / trial[i]
}
## 7/24 よりも小さい確率で起こった?
length(result[result < (7 / 24)]) / iter
```

```
[1] 0.02262
```

これを見ると，両側検定にしても 0.04524 なので，ギリギリ有意になるかどうか，というところだろうか。

さて判断にこまった。「24 回やる」と決めていたのであれば $\theta = 0.5$ は棄却されないし，「7 回成功するまで」と決めていたのであれば $\theta = 0.5$ は棄却される。「5 分間」と決めていても棄却されるが，そもそもこうした実験者の意図によって判断が揺らいで良いものだろうか?というのが Kruschke (2014 前田・小杉監訳 2017) の指摘する疑問点である。

問題は，「24 回中 7 回成功」という事実に，二項分布，負の二項分布，あるいは組み合わさった分布のような，確

率分布の情報が含まれていないことにある。この確率分布はデータが既知で母数が未知だから尤度関数であり，データ生成メカニズムであるとも言えるだろう。想定するメカニズムが明示されない検定は，ともすれば事後的に「実は負の二項分布を想定していたんですよ，へへ」ということも可能になってしまう。こうした点からも，研究者の自由度をなるべく少なくする研究計画の**事前登録**制度が必要であることがわかる。

10.2　サンプルサイズ設計

どのようにデータを取り，どのように分析・検定し，どのような基準で判断するかを事前に決めることに加え，事前にサンプルサイズを見積もっておく必要があるだろう。サンプルはとにかく多ければ多いほど良いか，というとそうではなく，過剰にサンプルを集めることは研究コストの増大であり，回答者の負担増でしかない。またサンプルサイズが大きくなると有意差を検出しやすくなるが，必要なのは有意差ではなく実質的に効果を見積もることであり，有意差が見つかれば良いというものではないことに注意が必要である。もちろん上で見てきたように，有意差が検出できるかどうかを指標にしてサンプルサイズを変動させてしまうのは，明らかに誤った研究実践である。

事前にサンプルサイズを決定するのに必要なのは，これまでのリバースエンジニアリングの演習例からもわかるように，効果量の見積もりである。[*3] これをどのように定めるかについては，先行研究を考えるとか，研究領域で「これぐらい差がないと意味がないよね」とコンセンサスが取れている程度で決めることになる。[*4]

10.2.1　対応のない t 検定

サンプルサイズの設計には，これまで使ってきた検定統計量に**非心度** non-centrality parameter というパラメータを加えて考える必要がある。

具体的に，対応のない t 検定を例にサンプルサイズ設計の方法を見てみよう。t 検定は言葉の通り，t 分布を用いて帰無仮説の元での検定統計量の実現値が問題になるのであった。帰無仮説は $\mu_0 = \mu_1 - \mu_2 = 0$ であり，検定統計量は次式で表されるのであった。

$$T = \frac{d - \mu_0}{\sqrt{U_p^2 / \frac{n_1 n_2}{n_1 + n_2}}}$$

この分子において，$d - \mu_0 = (\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)$ の第二項を,$\mu_1 - \mu_2 = 0$ と仮定するから，0 が中心の標準化された t 分布が用いられたのである。帰無仮説はこのように理論的に特定できる比較点をもとに置かれているのであって，帰無仮説下ではない現実の世界では，検定統計量は母平均の差 $\mu_1 - \mu_2$ に応じた分布から生じている。このように中心がずれている t 分布のことを**非心分布**といい，ズレの程度が非心度パラメータである。t 検定における非心度 λ は，以下の式で表される。

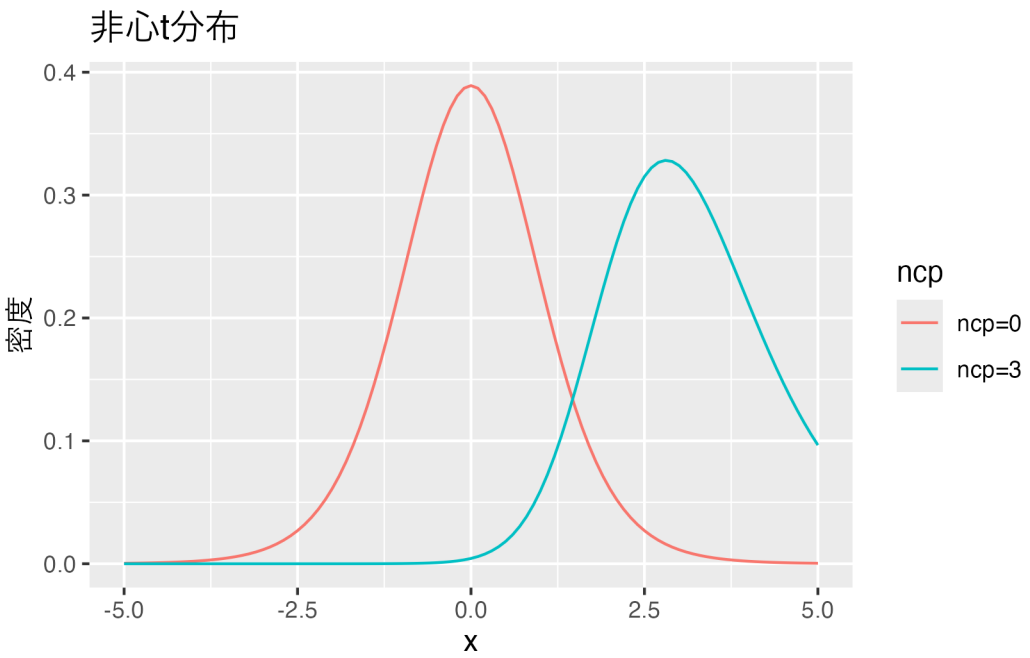
$$\lambda = \frac{(\mu_1 - \mu 2) - \mu_0}{\sigma\sqrt{n}}$$

[*3] もちろん基準となる有意水準 α，検出力 $1 - \beta$ も定める必要があるが，慣例的に $\alpha = 0.05$ であり，$1 - \beta = 0.8$ ぐらいが必要とされている。

[*4] この「最低限検出したい効果」のことを Smallest Effect Size of Interest, SESOI と呼ぶ。小杉他 (2023) も参照。

この非心度の分だけ，非心 t 分布は t 分布からズレていることになる。R では dt 関数に ncp パラメータがあり，デフォルトでは ncp=0 になっていた。これを変えて描画してみよう。

```
# データの準備
df <- 10 # 自由度を指定
# ggplot でプロット
ggplot(data.frame(x = c(-5, 5)), aes(x = x)) +
  stat_function(fun = dt, args = list(df = df, ncp = 0), aes(color = "ncp=0")) +
  stat_function(fun = dt, args = list(df = df, ncp = 3), aes(color = "ncp=3")) +
  labs(
    title = "非心 t 分布",
    x = "x",
    y = "密度",
    color = "ncp"
  )
```



ncp=0 の時は，中心が 0 にある帰無仮説の世界であり，これを使ってタイプ 1 エラー，つまり α が算出されたのであった。ncp を効果量で表現すれば，母平均の差がゼロでない時の分布が描けるのだから，タイプ 2 エラー，つまり β が計算できる。

自由度 df = 10，非心度 ncp = 3 の例で考えてみよう。タイプ 1 エラーになるのは，自由度 10 の t 分布で上 2.5% の臨界値以上の実現値が得られた時である。

```
qt(0.975, df = 10, ncp = 0)
```

```
[1] 2.228139
```

このとき，実際は ncp = 3 ほどずれていたのだから，タイプ 2 エラーが生じる確率は次のとおりである。

```
qt(0.975, df = 10, ncp = 0) %>% pt(df = 10, ncp = 3)
```

```
[1] 0.2285998
```

当然，ncp = 0 から離れるほどタイプ 2 エラーは生じにくくなる。非心度は母効果量 $\delta = \frac{(\mu_1 - \mu_2) - \mu_0}{\delta}$ を使って，$\lambda = \delta\sqrt{n}$ で表すことができる。

これを使って，t 検定のサンプルサイズを設計してみよう。話を簡単にするために，サンプルサイズは 2 群で等しいものとする。

検定統計量の式を思い出して，$\sqrt{n}$ にあたるところは 2 群のサンプルサイズから計算される，プールされた標本サイズから得られることに注意しよう[*5]。

```
alpha <- 0.05
beta <- 0.2
delta <- 0.5

for (n in 10:1000) {
  df <- n + n - 2
  lambda <- delta * (sqrt((n * n) / (n + n)))
  cv <- qt(p = 1 - alpha / 2, df = df) # Type1error の臨界値
  er <- pt(cv, df = df, ncp = lambda) # Type2error の確率
  if (er <= beta) {
    break
  }
}
print(n)
```

```
[1] 64
```

ここでは，サンプルサイズを 10 から徐々に増やしていき，1000 までの間で目標とする β まで抑えられたところで，カウントしていく for ループを break で脱出する，というかたちで組んでいる。結果的に，各群 64 名，合計 128 名のサンプルがあれば，目標が達成できることがわかる。サンプルサイズが 2 群で異なる場合など，詳細は @kosugi2023 に詳しい。

10.2.2　シミュレーションによるサンプルサイズ設計

非心 F 分布を使えば分散分析でもサンプルサイズができるし，そのほかの検定についても同様に非心分布を活用すると良い。しかし，非心分布の理解や非心度の計算など，ケースバイケースで学ぶべきことは多い。

そこで，電子計算機の演算力をたのみに，データ生成のシミュレーションを通じて設計していくことを考えてみ

[*5] t 統計量の実現値の式にある分母，$\sqrt{U_p^2/\frac{n_1 n_2}{n_1+n_2}}$ に見られる，プールされた不偏分散を割るための標本サイズであり，2 群の母分散が等しいと仮定して計算するなら，$\sigma^2(\frac{1}{n_1} + \frac{1}{n_2}) = \sigma^2(\frac{n_1+n_2}{n_1 n_2}) = \sigma^2/\frac{n_1 n_2}{n_1+n_2}$ から得られる。

よう。サンプルサイズや効果量を定めれば，仮想データを作ることができるし，それ対して検定をかけることもできる。仮想データの生成と検定を反復し，タイプ 2 エラーがどの程度生じるかを相対度数で近似することもできるだろう。であれば，その近似をサンプルサイズを徐々に変えることで繰り返してサンプルサイズを定めることもできる。

以下は，母相関が $\rho = 0.5$ とした時に，検出力が 80% になるために必要なサンプルサイズを求めるシミュレーションコードである。

```
library(MASS)
set.seed(12345)
alpha <- 0.05
beta <- 0.2
rho <- 0.5
sd <- 1
Sigma <- matrix(NA, ncol = 2, nrow = 2)
Sigma[1, 1] <- Sigma[2, 2] <- sd^2
Sigma[1, 2] <- Sigma[2, 1] <- sd * sd * rho

iter <- 1000

for (n in seq(from = 10, to = 1000, by = 1)) {
  FLG <- rep(0, iter)
  for (i in 1:iter) {
    X <- mvrnorm(n, c(0, 0), Sigma)
    cor_test <- cor.test(X[, 1], X[, 2])
    FLG[i] <- ifelse(cor_test$p.value > alpha, 1, 0)
  }
  t2error <- mean(FLG)
  print(paste("n=", n, "のとき，beta は", t2error, "です。"))
  if (t2error <= beta) {
    break
  }
}
```

```
[1] "n= 10 のとき，beta は 0.681 です。"
[1] "n= 11 のとき，beta は 0.639 です。"
[1] "n= 12 のとき，beta は 0.612 です。"
[1] "n= 13 のとき，beta は 0.566 です。"
[1] "n= 14 のとき，beta は 0.563 です。"
[1] "n= 15 のとき，beta は 0.471 です。"
[1] "n= 16 のとき，beta は 0.462 です。"
[1] "n= 17 のとき，beta は 0.419 です。"
```

```
[1] "n= 18 のとき，beta は 0.402 です。"
[1] "n= 19 のとき，beta は 0.385 です。"
[1] "n= 20 のとき，beta は 0.353 です。"
[1] "n= 21 のとき，beta は 0.344 です。"
[1] "n= 22 のとき，beta は 0.312 です。"
[1] "n= 23 のとき，beta は 0.285 です。"
[1] "n= 24 のとき，beta は 0.256 です。"
[1] "n= 25 のとき，beta は 0.265 です。"
[1] "n= 26 のとき，beta は 0.21 です。"
[1] "n= 27 のとき，beta は 0.227 です。"
[1] "n= 28 のとき，beta は 0.176 です。"
```

```
print(n)
```

```
[1] 28
```

ここではシミュレーション回数 1000，上限 1000，刻み幅を 1 にしているが，状況に応じて変更すると良い。

10.3 課題

1. 一要因 3 水準の Between デザインの分散分析において、1. 分散分析で有意差が見られる場合と、2. 任意の 2 水準の組み合わせのどこかで有意差が見られる場合を考えたとき、タイプ 2 エラーはどのように異なるかをシミュレーションで確かめてみましょう。設定として、`n1=n2=n3=10`、標準偏差も各群等しく `sigma = 1` とし、効果量 `delta = 2` でモデル化してみましょう。
2. N 増し問題は相関係数の検定の時も生じるでしょうか。母相関が $\rho = 0.3$ のとき、サンプルサイズを 10 から始めて、有意になるまでデータを追加する仮想研究を 1000 回行ってみましょう。データ追加の上限は 100、有意水準は $\alpha = 0.05$ として、最終的に有意になる割合を計算してみましょう。
3. $\alpha = 0.05, \beta = 0.2$ とし、効果量 $\delta = 1$ とした時の対応のない t 検定のサンプルサイズ設計をしたいです。1. 非心 t 分布を使った解析的な方法と、2. シミュレーションによる近似的な方法の両方で、同等の結果が出ることを確認しましょう。

第 11 章

重回帰分析の基礎

11.1　回帰分析の基礎

ここでは回帰分析を扱う。説明変数 x と被説明変数 y の関数関係 $y = f(x)$ に，次の一次式を当てはめるのが単回帰分析である。

$$y_i = \beta_0 + \beta_1 x_i + e_i = \hat{y}_i + e_i$$

一次式を $\hat{y}$ とまとめたものを予測値といい，予測値と実測値 y の差分 e_i を残差 residuals という。

空間上の一次直線の切片，傾きを求めるというのが基本的な問題であり，二点であれば一意に定めることができるが，データ分析の場面では 3 点以上の多くのデータセットの中に直線を当てはめることになるので，なんらかの外的な基準が必要になる。この時，「残差の分散が最も小さくなるように」と考えるのが**最小二乗法**の考え方であり，「残差が正規分布に従っていると考え，その尤度が最も大きくなるように」と考えるのが**最尤法**の考え方である。前者は記述統計的な，後者は確率モデルとしての感が過多になっていることに注意してほしい。また確率モデルの推定方法としては，事前分布を用いた**ベイズ推定**が用いられることもある。

最小二乗法による推定値は，次の式で表される。証明は他書 (小杉, 2018; 西内, 2017) に譲るが，ロジックとして残差の二乗和 $\sum e_i^2 = \sum (y_i - (\beta_0 + \beta_1 x_i))^2$ を最小にすることを考え，この式を展開するか偏微分を用いて極小値を求めることで算出できるとだけ伝えておこう。いずれにせよ，平均値 $\bar{x}, \bar{y}$ や分散・共分散 s_x, s_y, r_{xy} など標本統計量から推定できるのはありがたいことである。

$$\beta_0 = \bar{y} - \beta_1 \bar{x}, \quad \beta_1 = r_{xy} \frac{s_y}{s_x}$$

また，ここでは x, y ともに連続変数を想定しているが，説明変数 x が二値，あるいはカテゴリカルなものであれば y の平均値を通る直線を探すことになる。直線の傾きが 0 であれば「平均値が同じ」という線形モデルであり，これは平均値差の検定における帰無仮説と同等である。このように，t 検定や ANOVA は回帰分析の特殊ケースとも考えられ，まとめて**一般線形モデル**と呼ばれる。一般線形モデルは，被説明変数が連続的で，線形モデルによる平均値に正規分布に従う残差が加わったものとして考えるという意味で統一的に表現される。

ANOVA の場合は，二つ以上の要因による効果を考えることもあった。交互作用項を考慮しなければ，2 要因のモデルは次のように表現することができる。

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + e_i$$

このように説明変数が複数ある回帰分析を特に重回帰分析 Multiple Regression Analysis と呼ぶ。一次式なので，ある変数に限れば線形性が担保されているから，これも線形モデルの仲間である。重回帰分析を用いる場合は，説明変数同士を比較して「どちらの説明変数の方が影響力が大きいか」ということが論じられることが多いが，係数は当然 x_n, y の単位に依存するため，素点の回帰係数は使い勝手が悪い。そこですべての変数を標準化した標準化係数が用いられることが多い。

11.2　回帰分析の特徴

以下，具体的なデータを用いて回帰分析の特徴を見てみよう

11.2.1　パラメータリカバリ

回帰分析のモデル式にそってデータを生成し，分析によってパラメータリカバリを行ってみよう。

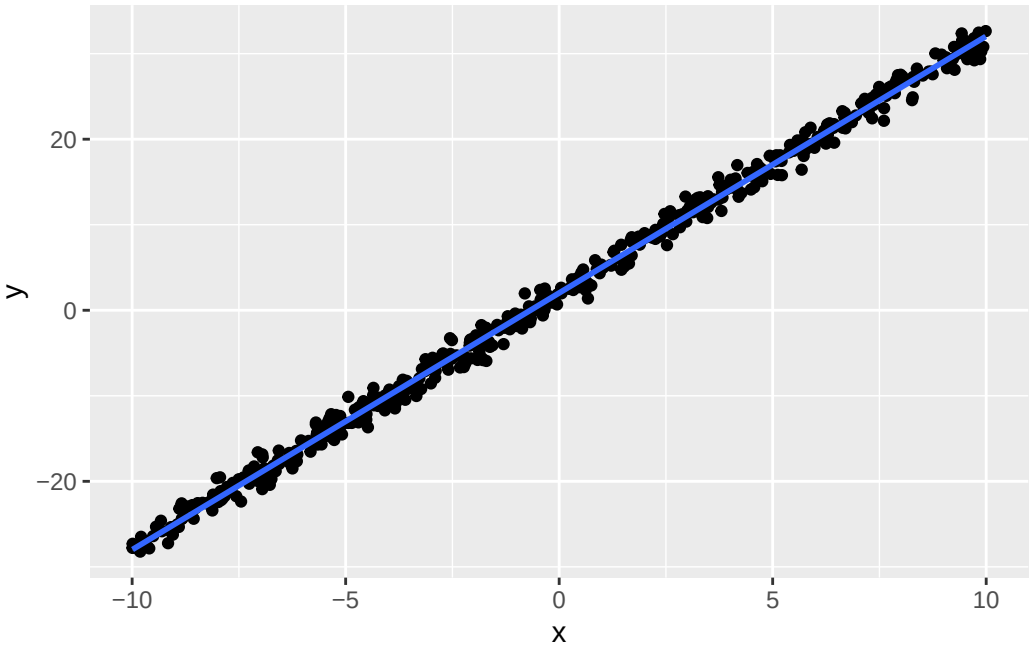
説明変数については制約がないので一様乱数から生成し，平均 0，標準偏差 σ の誤差とともに被説明変数を作り，

```
library(tidyverse)
set.seed(123)
n <- 500
beta0 <- 2
beta1 <- 3
sigma <- 1
# データの生成
x <- runif(n, -10, 10)
e <- rnorm(n, 0, sigma)
y <- beta0 + beta1 * x + e

dat <- data.frame(x, y)
# データの確認
head(dat)
```

```
          x          y
1 -4.248450 -11.120952
2  5.766103  18.736432
3 -1.820462  -3.805302
4  7.660348  25.071541
5  8.809346  30.026546
```

```
6 -9.088870 -25.355175
```

```
dat %>% ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = "y~x") # 線形モデルの描画
```



このデータに基づいて回帰分析を実行した結果が以下のとおりである。

```
result.lm <- lm(y ~ x, data = dat)
summary(result.lm)
```

```
Call:
lm(formula = y ~ x, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-2.82796 -0.61831  0.03553  0.69367  2.68062

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.021928   0.045010   44.92   <2e-16 ***
x           3.002194   0.007919  379.09   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.006 on 498 degrees of freedom
```

```
Multiple R-squared:  0.9965,    Adjusted R-squared:  0.9965
F-statistic: 1.437e+05 on 1 and 498 DF,  p-value: < 2.2e-16
```

ここでは $\beta_0 = 2, \beta_1 = 3, \sigma = 1$ と設定しており，ほぼ理論通りの係数がリカバリーできていることを出力から確認しておこう。もちろんリカバリの精度は，データの線形性の強さに依存するから，残差の分散が大きかったりサンプルサイズが小さくなると，必ずしもうまくリカバリできないことがあることは想像に難くないだろう。

11.2.2　残差の正規性と相関関係

lm 関数が返した結果オブジェクトには，表示されていない多くの情報が含まれている。例えば予測値や残差も含まれているので，これを使って回帰分析の特徴を見てみよう。

```
dat <- bind_cols(dat, yhat = result.lm$fitted.values, residuals = result.lm$residuals)
summary(dat)
```

```
       x                   y                yhat             residuals       
 Min.   :-9.99069   Min.   :-28.216   Min.   :-27.9721   Min.   :-2.82796  
 1st Qu.:-5.08007   1st Qu.:-13.074   1st Qu.:-13.2294   1st Qu.:-0.61831  
 Median :-0.46887   Median :  0.301   Median :  0.6143   Median : 0.03553  
 Mean   :-0.09433   Mean   :  1.739   Mean   :  1.7387   Mean   : 0.00000  
 3rd Qu.: 4.65795   3rd Qu.: 15.963   3rd Qu.: 16.0060   3rd Qu.: 0.69367  
 Max.   : 9.98809   Max.   : 32.638   Max.   : 32.0081   Max.   : 2.68062  
```

予測値 $\hat{y}$ は fitted.values として保存されている。これの平均値が被説明変数 y の平均値に一致していることが確認できる。回帰分析は説明変数 x を伸ばしたり (β_1 倍する) ズラしたり (β_0 を加える) しながら，被説明変数 y に当てはめるのであり，位置合わせがなされた予測値の中心が被説明変数の中心と一致することは理解しやすいだろう[*1]。

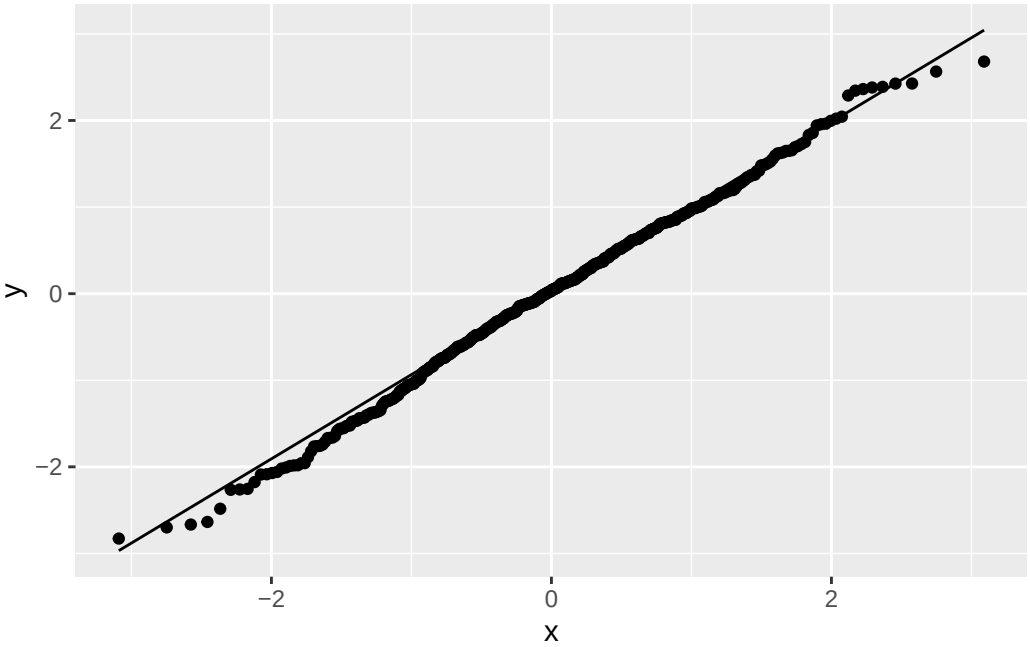
次に，残差の平均が 0 になっていることも確認しておこう。これが 0 でない c であれば，回帰係数が常に c だけズレていることになるので，そのような系統的ズレは最適な線形の当てはめにおいて除外されているべきだからである[*2]。

また，回帰分析において残差は正規分布に従うことが仮定されていた。これを検証するには Q-Q プロットを見ると良い。

```
dat %>%
  ggplot(aes(sample = residuals)) +
  stat_qq() +
  stat_qq_line()
```

[*1] もちろん証明できる。$\beta_0 = \bar{y} - \beta_1 \bar{x}, \beta_1 = r_{xy}\frac{s_y}{s_x}$ より，$\bar{\hat{y}} = \frac{1}{n}\sum(\bar{y} - \beta_1\bar{x} + \beta_1 x_i) = \bar{y} - \beta_1\bar{x} + \beta_1\frac{1}{n}\sum x_i = \bar{y}$ である。

[*2] もちろん証明できる。$\bar{e} = \frac{1}{n}\sum e_i = \frac{1}{n}\sum(y_i - \hat{y}_i) = \bar{y} = \bar{\hat{y}} = 0$ である。



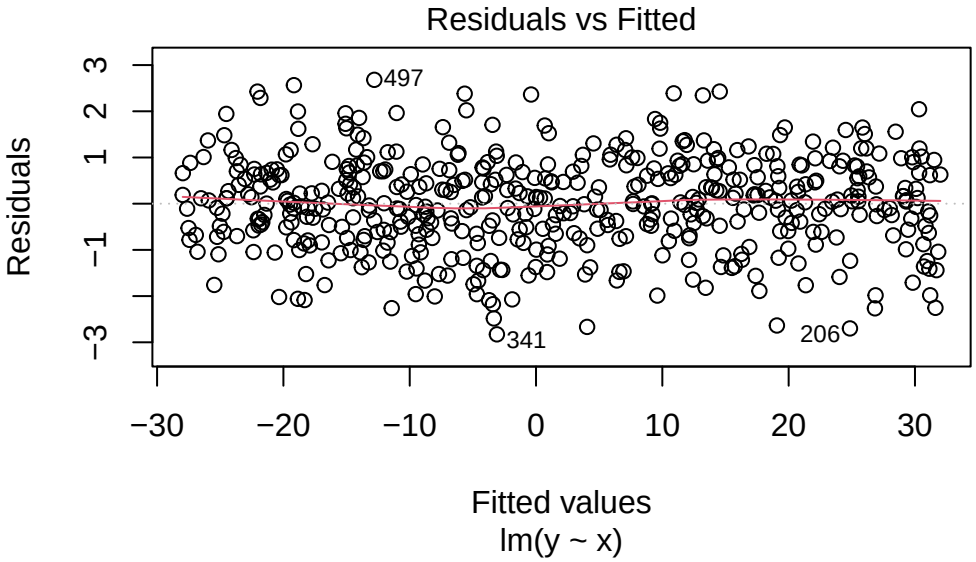
Q-Q プロットとは 2 つの確率分布を比較するためのグラフであり，横軸には理論的分布の分位点が，縦軸に実データが並ぶもので，右上がりの直線上にデータが載っていれば分布に従っている，と判断するものである。直線から逸脱している点は理論的分布からの逸脱と考えられる。今回の結果はほとんどが正規分布の直線上にあることから，大きな逸脱がないことが認められる。

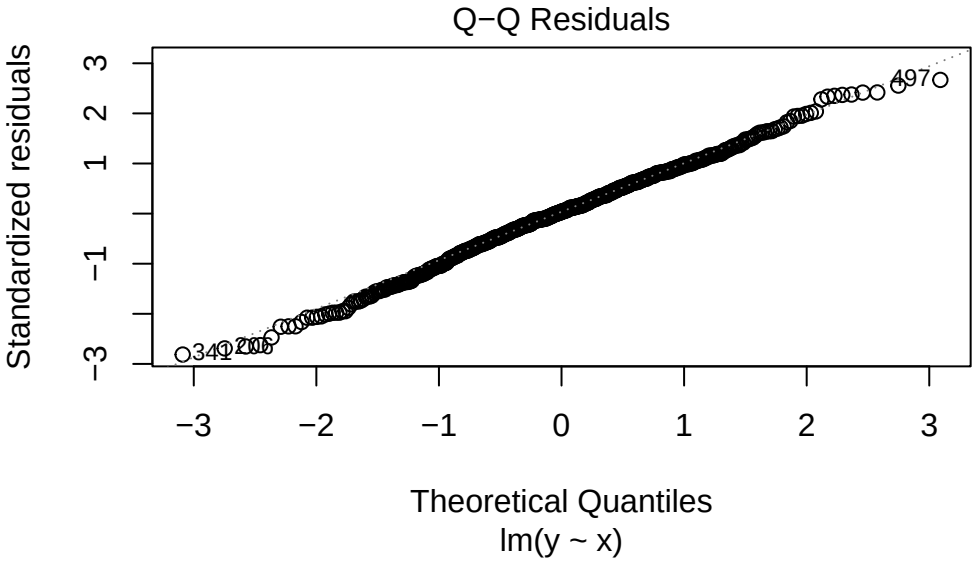
データ生成メカニズムによっては，被説明変数が二値的であったり，順序的であったり，カウント変数であったり，と正規分布がそぐわないものもあるだろう。そのようなデータに無理やり回帰分析を当てはめることは適切ではない。いかなる時もデータは可視化して，モデルを当てはめることの適切さをチェックすることを忘れたはならない。

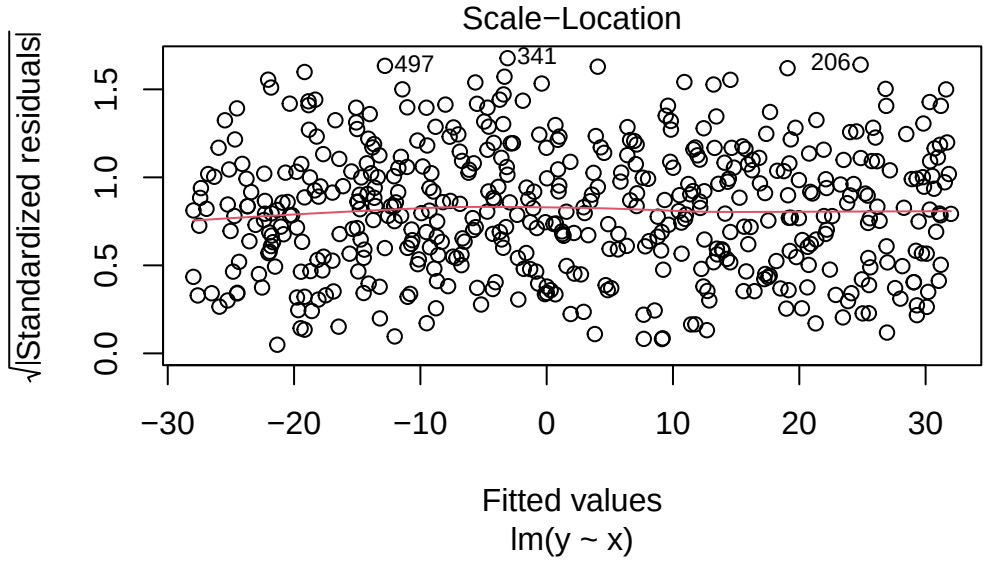
ちなみに出力結果を直接 `plot` 関数に入れてもよい。ここから残差と予測値の相関関係や，Q-Q プロット，標準化残差のスケールロケーションプロット，レバレッジと標準化残差[*3]などがプロットされる。

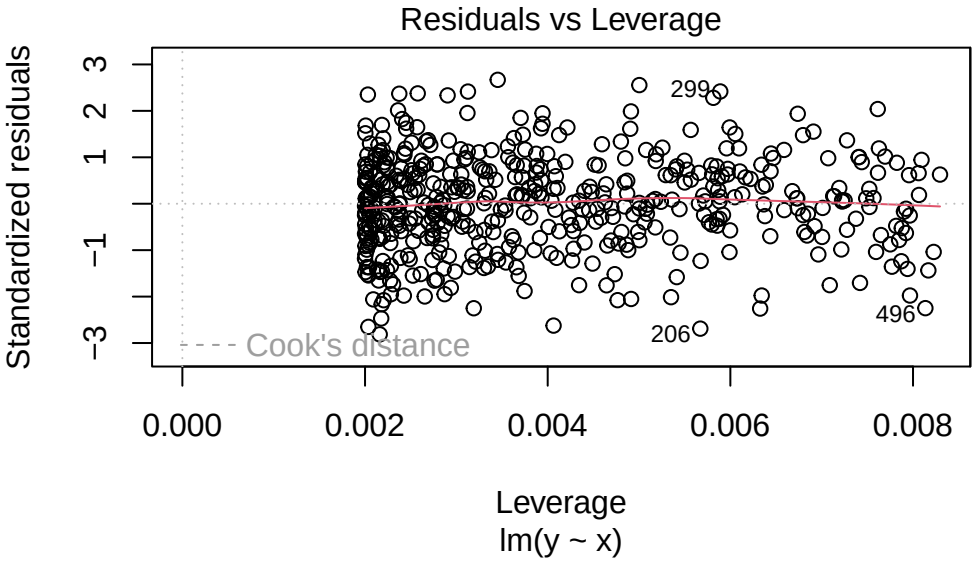
```
plot(result.lm)
```

[*3] 縦軸の標準化された残差の大きな値は解釈に注意が必要な外れ値である可能性が高い。レバレッジも同様に回帰係数に大きな影響を与える値の指標であり，この図の端に位置する変数は注意が必要，と考える。


Residuals vs Fitted
Residuals
497
341
206
−30
−20
−10
0
10
20
30
−3
−1
1
2
3
Fitted values
lm(y ~ x)

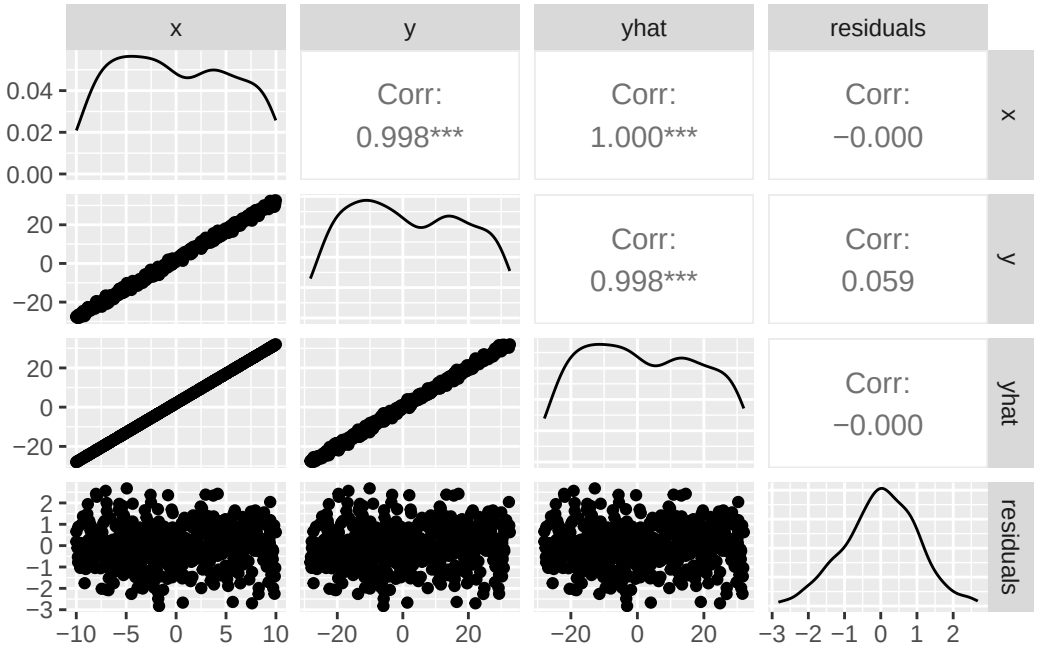

Q−Q Residuals
Standardized residuals
497
341
206
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(y ~ x)


Scale−Location
√|Standardized residuals|
497
341
206
0.0
0.5
1.0
1.5
−30
−20
−10
0
10
20
30
Fitted values
lm(y ~ x)



残差と予測値のプロットからも想像できるが，両者の相関はゼロである。図で確認しておこう。

```
library(GGally) # 必要ならインストールしよう
ggpairs(dat)
```



この関係から明らかなように，残差は説明変数や予測値と相関しない[*4]。説明変数と残差に相関関係があるとすると，説明変数でまだ説明できていない分散が残っていることになるし，予測値と残差に相関がないことは予測値が高いか低いかにかかわらず，残差が一様に分布していることを意味する。このことを踏まえて，重回帰分析の特徴を理解していこう。

[*4] 論文が早期公開された後，心理学会が主催するオンラインシンポジウムでは著者とこの論文で取り上げられた論文の著者が登場して，議論が交わされた (日本心理学会 YouTube ライブ・話題の論文について著者と語るシリーズ,2021 年 7 月 2 日 20 時-21 時 40 分)。平日の夜という設定，早期公開版における議論であったにも関わらず，1700 名近い視聴者が参加した。

11.3　重回帰分析の特徴

重回帰分析においては，回帰係数は偏回帰係数 partial regression coefficients と呼ばれる。この「偏」の一文字が意味することを考えていこう。

11.3.1　回帰係数と偏回帰係数

単回帰分析の回帰係数は，説明変数 x が一単位上昇した時の被説明変数の変化量，と解釈すればよい。これに対して重回帰分析の偏回帰係数を，「説明変数 x_1 が一単位上昇した時の被説明変数の変化量」とすることはできない。というのも，説明変数が複数 $(x_2, x_3, ...)$ あり，他の説明変数の次元についての変化を考慮していない変化量になっているからである。

重回帰分析において，説明変数が完全に無相関で直交しているのであれば，x_1 の変化と x_2 の変化を独立して説明できるが，往々にしてそのようなことはない。偏回帰係数は当該変数以外の変動を**統制した**回帰係数である。

上で単回帰係数において，説明変数と残差が相関しないことを確認した。言い換えれば，説明変数で説明でき分散は全て説明し尽くされており，残差は説明変数で説明できない被説明変数の分散，つまり説明変数の影響を除外した被説明変数の分散と考えることができる (被説明変数の分散=説明変数が説明する分散 + 残差の分散)。

ここで第二の変数 x_2 があったとする。第一の変数 x_1 で y を説明した残差 e_y と，第一の変数で第二の変数を説明した残差 e_{x2} との相関を**偏相関 partial correlation** という。これは第一の変数 x_1 からの影響を両者から取り除いているので，x_1 で統制した相関係数ということができる。偏相関は単純な相関が「見せかけの関係」でないことを検証するための重要な指標である。

偏相関係数を計算してみよう。

```
library(MASS)
library(psych)
Sigma <- matrix(c(1, 0.3, 0.5, 0.3, 1, 0.8, 0.5, 0.8, 1), ncol = 3)
X <- mvrnorm(1000, c(0, 0, 0), Sigma, empirical = TRUE) %>% as.data.frame()
## 相関行列
cor(X)
```

```
    V1  V2  V3
V1 1.0 0.3 0.5
V2 0.3 1.0 0.8
V3 0.5 0.8 1.0
```

```
## 回帰分析をして残差を求める
result.lm1 <- lm(V2 ~ V1, data = X)
result.lm2 <- lm(V3 ~ V1, data = X)
cor(result.lm1$residuals, result.lm2$residuals)
```

```
[1] 0.7867958
```

```
## 偏相関を求めるR関数で確認
psych::partial.r(X)[2, 3]
```

```
[1] 0.7867958
```

最後は psych パッケージの偏相関行列を求める関数で検証した。確かに残差同士の相関係数が偏相関係数になっていることが確認できたと思う。

そして，ここでは残差同士の相関係数として算出しているが，残差をつかった回帰分析の係数が偏回帰係数になるのである。このデータセットの第一変数を従属変数にした重回帰分析の結果から，これを確認してみよう。

```
result.mra <- lm(V1 ~ V2 + V3, data = X)
# 回帰係数を取り出す
result.mra$coefficients
```

```
  (Intercept)            V2            V3
 1.171322e-17 -2.777778e-01  7.222222e-01
```

```
# 残差をつかって偏回帰係数を確認する
result.lm3 <- lm(V1 ~ V3, data = X)
result.lm4 <- lm(V2 ~ V3, data = X)
result.lm5 <- lm(result.lm3$residuals ~ result.lm4$residuals)
#
result.lm5$coefficients
```

```
         (Intercept) result.lm4$residuals
        5.381952e-18        -2.777778e-01
```

重回帰分析の結果 result.mra の V2 から V1 への回帰係数は-0.2778 である。また，V3 で V1,V2 を統制した残差同士をつかい，回帰係数を求めた結果は-0.2778 と，同じ値になっていることが確認できただろう。

V3 から V1 への偏回帰係数も同様で，V2 で両者を統制した残差同士による回帰係数になっている。このように，重回帰分析の回帰係数は，他の説明変数で統制した値になっており，日本語で説明するなら「他の変数の値が同じであると想定した条件つきの，当該変数の影響力」とでもいうべき値になっている。

なぜこのような持って回った説明をするかというと，つい「条件付きの」という話を忘れて報告，解釈してしまうことが多いからで，吉田・村井 (2021) の論文での指摘は議論を呼んだのは記憶に新しい[*5]。たとえば今回の例でも，回帰係数が-0.2778 であったのに対し，V1 と V2 の単相関が 0.3 であったことを思い出そう。符号が反転しているため，解釈は真逆になってしまう。実際の単相関は正の関係であるから，条件付きであることを忘れて「V2 は負の影響，V3 は正の影響」と表現してしまうと，ミスリーディングなことになるからである。

[*5] 論文が早期公開された後，心理学会が主催するオンラインシンポジウムでは著者とこの論文で取り上げられた論文の著者が登場して，議論が交わされた (日本心理学会 YouTube ライブ・話題の論文について著者と語るシリーズ,2021 年 7 月 2 日 20 時-21 時 40 分)。平日の夜という設定，早期公開版における議論であったにも関わらず，1700 名近い視聴者が参加した。

また，豊田 (2017) は重回帰分析のこうした誤用を避けるために，独立変数を事前に直交化したデザインで行うコンジョイント分析の積極的な利用を提案している。我々が重回帰分析をうまく使いこなせないのであれば，そうした手法も有用であるだろう。

11.3.2　多重共線性

偏回帰係数の解釈が難しい理由の一つは，説明変数同士に相関関係がみられることにある。特に，説明変数間の相関関係が高くなることは**多重共線性** Multicollinearity の問題という。この問題は，回帰係数の標準誤差がインフレを起こすことを指す。

例えば先ほどの例で，説明変数 V2 と V3 は相関係数 0.8 を持っていた。この時の回帰係数の標準誤差を確認しておこう。

```
summary(result.mra)
```

```
Call:
lm(formula = V1 ~ V2 + V3, data = X)

Residuals:
     Min       1Q   Median       3Q      Max
-2.59118 -0.54717 -0.03692  0.55044  2.90735

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.171e-17  2.690e-02   0.000        1
V2          -2.778e-01  4.486e-02  -6.192 8.65e-10 ***
V3           7.222e-01  4.486e-02  16.100  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8507 on 997 degrees of freedom
Multiple R-squared:  0.2778,    Adjusted R-squared:  0.2763
F-statistic: 191.7 on 2 and 997 DF,  p-value: < 2.2e-16
```

標準誤差は 0.0449 であり，それほど大きくない標準誤差で問題がないようである。しかし両者の相関係数がより高くなり，一方が他方に線形的に従属してしまうと係数の推定値が不安定になるため，注意が必要である。

このインフレを確認するための指標が Variance Infration Factor: VIF である。R では car パッケージにある vif 関数に重回帰モデルを入れることでこの指標が算出される。一般に VIF が 3，あるいは 10 を超えると多重共線性が生じており，解釈に注意が必要と言われている[*6]。

[*6] 2 変数重回帰分析モデルで，VIF が 3 であれば説明変数間の相関は $r = 0.81$ 程度である。VIF が 10 であれば $r = 0.97$ にもなる。詳しくは小杉他 (2023) を参照。

```
library(car) # なければ入れておこう
vif(result.mra)
```

```
      V2       V3
2.777778 2.777778
```

幸い，今回の値はこれらの基準を下回っていたので許容範囲内である。

11.3.3 変数の投入順序

重回帰分析の場合は複数の説明変数があるが，これを投入するときに全ての変数を同時に投入するか，順番をつけて投入するかといった手法の違いがある。前者を強制投入法と呼ぶこともある。順番をつけて投入する方法は，逐次投入と呼ばれる。この場合は，適合度指標などを参考に変数を追加あるいは削除して，適合度が統計的に有意に向上するかどうかを考えながら進めていく。

重回帰係数の予測値 $\hat{y}$ と，被説明変数 y の相関係数 $R_{y\hat{y}}$ は**重相関係数**と呼ばれ，予測がうまくいっているかどうかを表す適合度の一つである。相関係数なので -1 から $+1$ までの値を取りうるが，-1 は逆に完全に合致していることになるので，この相関係数の符号は大して情報を持たない。そこでこれを二乗した $R^2_{y\hat{y}}$ を考える。これは**決定係数**とも呼ばれ，説明変数の分散のうち予測値の分散が占める割合を表している。[*7]

説明変数の逐次投入は，説明変数を持たないヌルモデルから一つずつ追加していく Forward Selection，全ての変数を投入してから一つずつ減らしていく Backward Selection がある。Forward のほうは追加することによって R^2 が有意に増加するか，Backward のほうは削除しても有意に R^2 が減らないか，を確認しながら進めることになる。この方法は手元のデータに最も適した説明変数のペアを選出できる方法ではあるが，検定を繰り返していることの問題と，手元のデータ以外に一般化する時の根拠の乏しさから，用いられないこともある。

逐次投入法には別の観点からの手法もある。それが階層的回帰分析である。この手法は，重回帰分析における交互作用項の投入を検討する文脈で発展した。重回帰分析では，説明変数同士の相関がない，もしくは小さい方が望ましい。しかし，交互作用とは分散分析における組み合わせの効果を表すものであり，実験デザインによっては交互作用効果が重要な変動であることも少なくない。回帰分析と分散分析は，一般線形モデルという形で統一的に理解されるが，回帰分析でも連続的に変化する組み合わせの効果を考えることができる。交互作用があるということは説明変数間に相関があることを意味するため，回帰分析の大前提に抵触する可能性があり，その投入には慎重を期する必要がある。

こうした文脈から，まずは要因の効果を投入し，次に交互作用項を投入してモデル適合度の有意な改善がみられるかどうかを検証する手順が推奨されている。この逐次投入法を特に階層的回帰分析と呼ぶ。ここでの「階層」とは，手順が重要度順に進められていることを意味し，データの特徴に関するものではないことに注意が必要である[*8]。

[*7] もちろん証明できる。小杉 (2018) を参照。

[*8] これに対して，データの階層性 (ex. 学級 ⊂ 市区町村 ⊂ 都道府県) を考慮する線形モデルのことを，階層線形モデル Hierarichal Linear Model:HLM という。

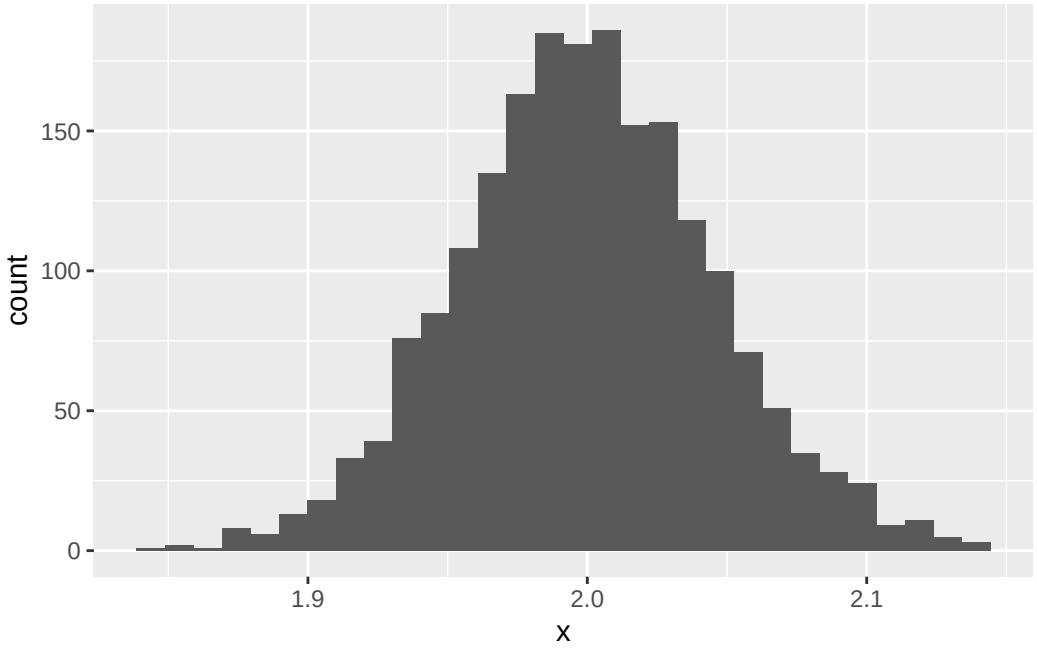
11.4　係数の標準誤差と検定

11.4.1　係数の検定

サンプルが母集団から得られた確率変数であるのだから，(偏) 回帰係数もまた確率変数である。すなわち，サンプルが変わるごとに変化し，その揺らぎがある確率分布に従うと考えられる。これを確認するためには，データ生成過程をモデリングし，反復することで近似させて理解するのがいいだろう。

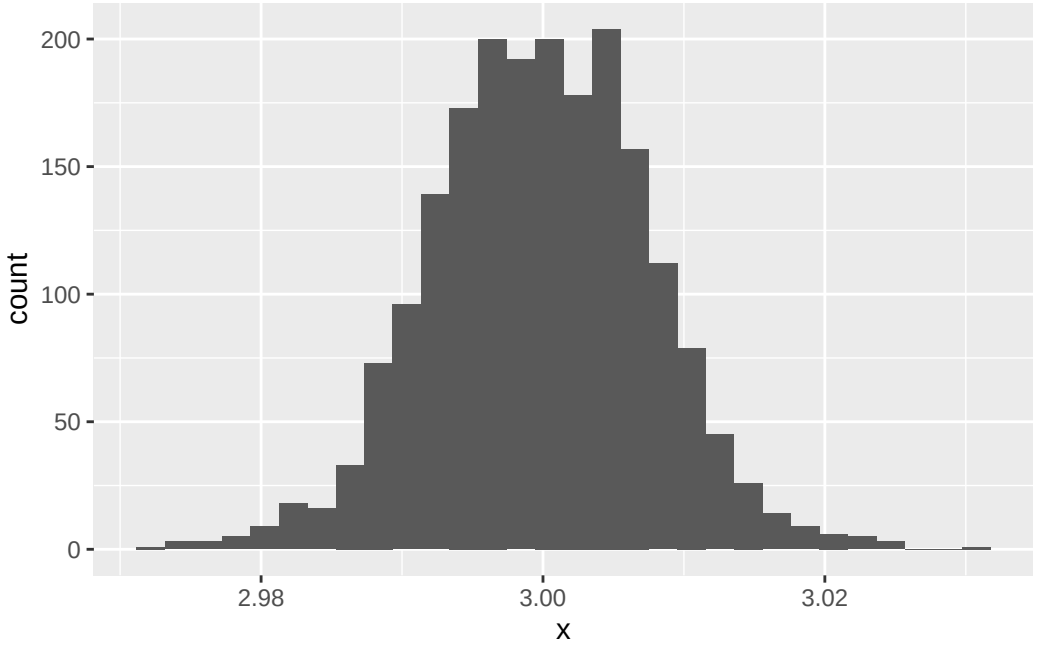
```
set.seed(123)
n <- 500
beta0 <- 2
beta1 <- 3
sigma <- 1
# データ生成関数
dataMake <- function(n, beta0, beta1, sigma) {
  x <- runif(n, -10, 10)
  e <- rnorm(n, 0, sigma)
  y <- beta0 + beta1 * x + e
  dat <- data.frame(x, y)
  return(dat)
}

# 結果オブジェクトの準備
iter <- 2000
beta0.est <- rep(NA, iter)
beta1.est <- rep(NA, iter)
# simulation
for (i in 1:iter) {
  sample <- dataMake(n, beta0, beta1, sigma)
  result.lm <- lm(y ~ x, data = sample)
  beta0.est[i] <- result.lm$coefficients[1]
  beta1.est[i] <- result.lm$coefficients[2]
}

data.frame(x = beta0.est) %>% ggplot(aes(x = x)) +
  geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
data.frame(x = beta1.est) %>% ggplot(aes(x = x)) +
  geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



図から明らかなように，回帰係数も確率的に分布する。ただしその平均は理論値に近似している。

```
mean(beta0.est)
```

```
[1] 1.999257
```

```
mean(beta1.est)
```

```
[1] 2.999798
```

この分布の幅が回帰係数の標準誤差である。

```
sd(beta0.est)
```

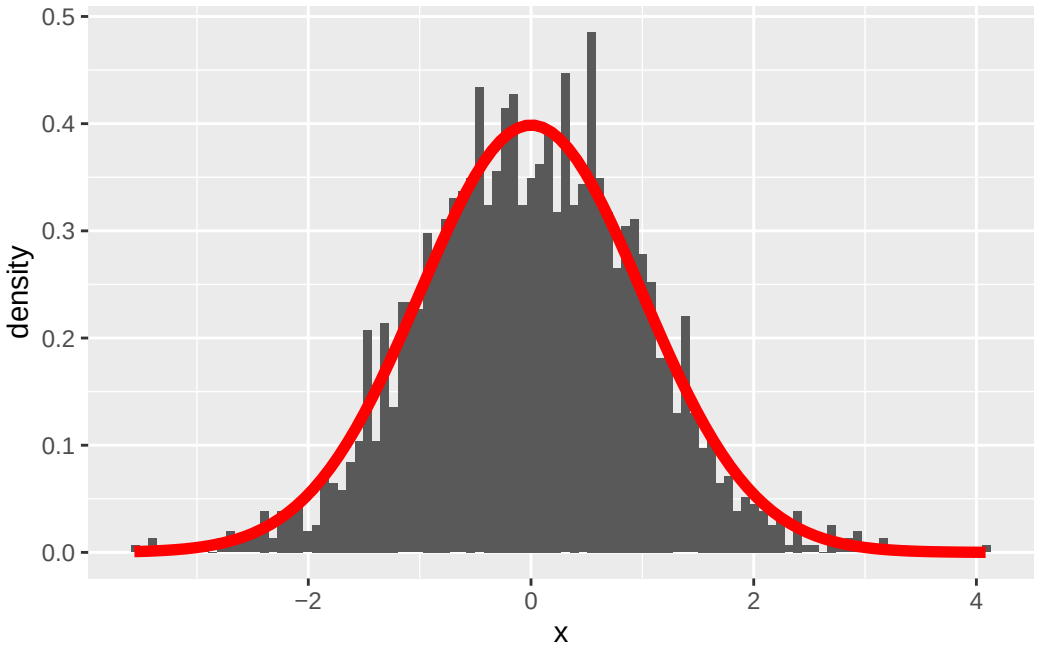
```
[1] 0.04580387
```

```
sd(beta1.est)
```

```
[1] 0.007659277
```

回帰係数は t 分布に従い，その自由度はサンプルサイズからモデルで用いる係数の数を引いたものになる。先ほどのヒストグラムを基準化し，理論分布を重ねて描画してみることで確認しておこう。

```
data.frame(x = beta1.est) %>%
  scale() %>%
  ggplot(aes(x = x)) +
  geom_histogram(aes(y = after_stat(density)), bins = 100) +
  stat_function(fun = function(x) dt(x, df = n - 2), color = "red", linewidth = 2)
```



この t 分布を用いて，係数が 0 の母集団から得られたサンプルなのかどうかの検定が行われる。

11.4.2　モデル適合度の検定

一方で，出力の最後には F 統計量による検定も行われていたことを確認しておこう。次に示すのは重回帰分析の例である。

```r
set.seed(123)
n <- 500
beta0 <- 2
beta1 <- 0
beta2 <- 0
sigma <- 1
x1 <- runif(n, -10, 10)
x2 <- runif(n, -10, 10)
e <- rnorm(n, 0, sigma)
y <- beta0 + beta1 * x1 + beta2 * x2 + e
sample <- data.frame(y,x1,x2)
result.lm <- lm(y ~ x1 + x2, data = sample)
summary(result.lm)
```

```
Call:
lm(formula = y ~ x1 + x2, data = sample)

Residuals:
     Min       1Q   Median       3Q      Max
-2.85235 -0.68275 -0.01436  0.67809  2.70488

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.996999   0.045263  44.120   <2e-16 ***
x1          -0.006453   0.007970  -0.810    0.418
x2          -0.003928   0.007795  -0.504    0.615
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.012 on 497 degrees of freedom
Multiple R-squared:  0.001893,  Adjusted R-squared:  -0.002124
F-statistic: 0.4713 on 2 and 497 DF,  p-value: 0.6245
```

上の例では，統計量 F が，自由度 $F(\ 2{,}497\ )$ のもとで，0.4713 であり，統計的に有意ではないと判断される (p=0.6245,n.s.)。

これは重相関係数に対する検定であり，母集団においてモデル全体としての説明力が 0 である，という帰無仮説を検証しているものである。この有意性検定には，説明変数の数 p，サンプルサイズ n，重相関係数 R^2 を用いて，以下の式で用いられる検定統計量 F を利用する (南風原, 2014)。

$$F = \frac{R^2}{1-R^2} \cdot \frac{n-p-1}{p}$$

ここで右辺第一項目は Cohen の効果量 ($f^2 = \frac{R^2}{1-R^2}$) といわれ，サンプルサイズ設計においてはこの指標が利用される。

11.5　サンプルサイズ設計

重回帰分析のサンプルサイズ設計は，変数の効果の大きさ (回帰係数) が事前にわかっているのであれば，n を徐々に増やしていくシミュレーションによって行える。しかしそのようなケースは稀であり，実際には R^2 の検定を用いて，ある効果量と検出力の下で，正しく検出できるサイズを算出することになる。

サンプルサイズの算出には，非心 F 分布を用いる。この時の非心度は，効果量 f^2 に n をかけたものになる。これを使ってサンプルサイズ設計をする例は以下のとおりである。

```
f2 <- 0.15      # 効果量
alpha <- 0.05   # タイプ 1 エラー率
beta <- 0.2     # タイプ 2 エラー率
p <- 5          # 説明変数の数

for(n in 10 : 500){
    lambda <- f2 * n
    df1 <- p
    df2 <- n - p - 1
    cv <- qf(p = 1- alpha, df1, df2)
    t2error <- pf(q = cv, df1, df2, ncp  = lambda)
    if(t2error < beta){
        break
    }
}

print(n)
```

```
[1] 92
```

この設定では，n = 92 以上であればモデルとして影響力がないとは言えない，ということがわかる。

11.6　まとめ

重回帰分析は，人文社会科学で多用される技術ではあるが，技術が先行して理解が伴わないまま利用されているケースも少なくない。繰り返しになる点もあるが，以下に注意点をまとめておく。

- **偏回帰係数の意味**；重回帰分析における偏回帰係数は，ほかの変数を統制した上での値であり，あたかも各係数が独立直交しているかのように解釈するのは適切ではない。
- **誤差の正規性**；誤差は正規分布に従っているという仮定があり，二値データや整数しかとらないカウントデータなどに盲目的にモデルを適用してはならない。誤差の正規性が満たされているかどうかは，分析後に Q-Q プロットを用いて確認する。
- **誤差の均質性**；誤差はモデル全体にわたって同じ正規分布に従っているというのもモデルの仮定である。すなわち，独立変数に応じて誤差分散が変わるといった均質でないデータの場合は，正しく推定されない。誤差の均質性については，分析後の Q-Q プロットを用いて確認する。
- **誤差間の独立性**；誤差はモデル全体にわたって同じ正規分布から独立に生成されている (i.i.d) というのがモデルの仮定である。時系列データのように，誤差間に対応 (自己回帰) がみられるデータの場合は回帰分析は適切な手法とならない。状態空間モデルなど，誤差間関係を適切にモデリングしたものを当てはめる必要がある。
- **モデルの適切な定式化**；モデルには被説明変数に影響を与えるすべての変数が正しく含まれている必要がある。例えば，影響を与えることがわかっている変数 X_o を意図的に除外して分析をしたとする。そのモデルに含まれる変数 X_a が被説明変数 y に影響を与えていたとしても，X_a と X_o に相関があれば，X_o の影響力が X_a を通じて y に伝播する k から，X_a の影響力が課題に評価されることになる。自らの仮説のために，意図的に変数を選択するのは QRPs に該当する。
- **説明変数間の相関関係**；説明変数のうちにあまりにも相関関係が高い変数ペアがあれば，多重共線性の疑いが生じる。多重共線性は推定値の不安定さとなって現れる。このような場合は，説明変数を主成分分析で合成変数にまとめるといった対応が考えられる。また，高い相関ではないが交互作用効果が見たいといった場合は，逐次投入など慎重に個々の影響を考えながら投入するようにする (階層的重回帰分析)。なお交互作用項は，各変数の平均からの偏差をかけ合わせたものにすることが一般的である。

11.7 課題

1. 以下のデータセットは被説明変数 y，説明変数 $x1, x2$ からなる重回帰分析のサンプルデータです。画面には一部しか表示しておらず，全体 ($n = 100$) はこちら ex_regression1.csv からダウンロード可能です。このデータセットを用いて重回帰分析を行い，結果を出力してください。

```
           y         x1          x2
1  1.8685595 -4.248450  1.9997792
2 -0.5728781  5.766103 -3.3435292
3  1.0321850 -1.820462 -0.2277393
4 10.0468488  7.660348  9.0894765
5 -1.1968078  8.809346 -0.3419521
6  9.6719213 -9.088870  7.8070044
```

2. 以下のデータセットは被説明変数 y，説明変数 $x1, x2$ からなる重回帰分析のサンプルデータです。画面には一部しか表示しておらず，全体 ($n = 300$) はこちら ex_regression2.csv からダウンロード可能です。このデータセットを用いて重回帰分析を行ってください。結果のプロットから，上に挙げた重回帰分析の仮定に反しているところを指摘してください。

```
          y         x1           x2
1  3.586304 -0.4248450  0.132767341
2  8.599252  0.5766103  0.922713561
3  2.397115 -0.1820462  0.053684622
4  3.505236  0.7660348  0.007801881
5  6.517720  0.8809346  0.633076091
6 -1.394231 -0.9088870 -0.895346802
```

3. $R^2 = 0.3$ を目標として，説明変数の数 $p = 10$ の重回帰分析を行う際に，必要なサンプルサイズはいくつになるか，計算してみましょう。ここで，$\alpha = 0.05, \beta = 0.2$ とします。

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

第 13 章

多変量解析の入り口

第 14 章

ベイジアン分析

第 15 章

ベイジアンモデリング

第 16 章

演習問題

References

Bernaards, Coen A. & Jennrich, Robert 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, Jonah, Češnovar, Rok, & Johnson, Andrew (2023). *cmdstanr: R Interface to 'CmdStan'*. https://mc-stan.org/cmdstanr/, https://discourse.mc-stan.org.

Hadley, Wickham (2014). Tidy Data. *Journal of Statistical Software, 59*, 1–23. https://doi.org/10.18637/jss.v059.i10

南風原 朝和 (2014). 心理統計学の基礎——続・統合的理解のために——　有斐閣

Healy, Kieran. (2018). *Data Visualization: A Practical Introduction*. Princeton Univ Pr.
(キーラン・ヒーリー　瓜生 真也・江口 哲史・三村 喬生 (訳) (2021). データ分析のためのデータ可視化入門　講談社)

平岡 和幸・堀 玄 (2009). プログラミングのための確率統計　オーム社

池田 功毅・平石 界 (2016). 心理学における再現可能性危機：問題の構造と解決策 心理学評論, *59* (1), 3–14. https://doi.org/10.24602/sjpr.59.1_3

河野 敬雄 (1999). 確率概論　京都大学学術出版会

小杉 考司 (2018). 言葉と数式で理解する多変量解析入門　北大路書房

Kruschke, John K. (2014). *Doing Bayesian Data Analysis*. Elsevier.
(クルシュケ, J.K.　前田 和寛・小杉 考司 (監訳) 前田 和寛・小杉 考司・井関 龍太・井上 和哉・鬼田 崇作・紀ノ定 保礼・国里 愛彦・坂本 次郎・杣取 恵太・高田 菜美・竹林 由武・徳岡 大・難波 修史・西田 若葉・平川 真・福屋 いずみ・武藤 杏里・山根 嵩史・横山 仁史　(訳) (2017). ベイズ統計モデリング: R, JAGS, Stan によるチュートリアル 原著第 2 版　共立出版)

Lander, J.P. (2017). *R for Everyone*. Addison-Wesley Professional.
(ランダー, J.P.　高柳 慎一・津田 真樹・牧山 幸史・松村 杏子・簑田 高志 (訳) (2018). みんなの R 第 2 版　マイナビ出版)

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

西内 啓 (2017). 統計学が最強の学問である [数学編]——データ分析と機械学習のための新しい教科書——　ダイヤモンド社

Ren, Kun. (2016). *Learning R Programming*. Packt Publishing.
(　株式会社ホクソエム (監訳) 湯谷 啓明・松村 杏子・市川 太祐・ホクソエム　(訳) (2017). R プログラミング本格入門: 達人データサイエンティストへの道　共立出版)

Revelle, William (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, Yves (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

佐藤 坦 (1994). はじめての確率論——測度から確率へ——　共立出版

Stevens, Stanley Smith (1946). On the theory of scales of measurement. *Science, 103* (2684), 677–680.

高橋 康介　石田 基広 (編) (2018). 再現可能性のすゝめ　共立出版

豊田 秀樹 (2009). 検定力分析入門——R で学ぶ最新データ解析ー——　東京図書

豊田 秀樹 (2017). もうひとつの重回帰分析　東京図書

Wickham, Hadley. (2015). *Advanced R*. Taylor & Francis Group.
(　石田 基広・市川 太祐・高柳 慎一・福島 真太朗 (訳) (2016). R 言語徹底解説　共立出版)

Wickham, Hadley, Averick, Mara, Bryan, Jennifer, Chang, Winston, McGowan, Lucy D'Agostino, François, Romain, Grolemund, Garrett, Hayes, Alex, Henry, Lionel, Hester, Jim, Kuhn, Max, Pedersen, Thomas Lin, Miller, Evan, Bache, Stephan Milton, Müller, Kirill, Ooms, Jeroen, Robinson, David, Seidel, Dana Paige, Spinu, Vitalie, … Yutani, Hiroaki (2019). Welcome to the tidyverse. *Journal of Open Source Software, 4* (43), 1686. https://doi.org/10.21105/joss.01686

永田 靖・吉田 道弘 (1997). 統計的多重比較法の基礎　サイエンティスト社

吉田 寿夫・村井 潤一郎 (2021). 心理学的研究における重回帰分析の適用に関わる諸問題 心理学研究, *92* (3), 178–187. https://doi.org/10.4992/jjpsy.92.19226

吉田 伸生 (2021). 確率の基礎から統計へ 新装版 日本評論社

Zeileis, Achim (2005). CRAN Task Views. *R News, 5* (1), 39–40.

シ　(2016). 計算機言語のまとめノート　暗黒通信団

小杉 考司・紀ノ定 保礼・清水 裕士 (2023). 数値シミュレーションで読み解く統計のしくみ〜R でためしてわかる心理統計　技術評論社

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.