

# An Introduction to R

Reio Tanji

Osaka Univ., Graduate School of Econ.

April and May, 2022

## 関係資料

- このスライドを含めた資料は以下の URL からダウンロードできます
  - 最新版のスライド
  - スクリプトのサンプル
  - 名前を付けて保存→ファイルの種類を「すべてのファイル」に変更して、末尾が「.R」で終わる名前にする
  - 分析用のサンプルコードを追加しました。
- 適宜更新していく (予定) ので、たまにチェックしてみてください

## 自己紹介

- 丹治 伶峰 (たんじ れいお)
  - 大阪大学大学院 博士後期課程
  - 大阪大学経済学部卒業 (2018)
    - \* 学部時代は準硬式野球部に所属していました
  - 専門: 労働経済学・行動経済学
    - \* 野球のデータを使って色々してます
    - \* 野球以外もしてます
  - 趣味: 音楽聴く、Fantasy Baseball
- よろしくおねがいします

## R 言語を使う

- R 言語とは？
- R でできること
  - 研究・分析のフロー
- R 言語の基本構造

## R 言語とは？

- “*R is a free software environment for statistical computing and graphics*”(from The R Project for Statistical Computing)
  - 統計計算、グラフ作成を行うことができる無料のツール
  - データの収集、管理、分析から結果の出力・プレゼンテーションの作成までを一つの言語で行うことも出来る
  - この資料も R を使って作成 (R Markdown)
- 目標
  - 今回は、R の基本的な操作方法を学習した上で、論文執筆の上で最低限必要なアウトプットのための技術習得をめざす

## R でできること

- データの操作
  - データの取得：デフォルトのデータセット、csv ファイル等の読み込み
  - データの整理：必要な情報の抽出、データの変形、新たな変数の作成と保存
- データの要約・可視化
  - 基本統計量の作成
  - 変数間の関係を視覚的に描写：graphics, ggplot2
- データ分析
  - 計量経済学・統計学で用いられる様々な手法の実装
  - 結果の出力
- レポートの作成
  - 分析結果をスライドにして報告 RMarkdown
    - \* Word, Powerpoint 形式でファイルを出力
  - 出力の保存や体裁を整える手間が削減できる

## データの操作

- 利用するデータの読み込み

## データの例

企業	労働時間	月給	先輩
A	40	25	こわい
B	25	20	やさしい
C	60	50	いない

- データを加工する
  - 欲しい行だけ抜き出す、欲しい列だけ抜き出す
  - 元データの情報を使って、分析のための新しい変数を作る
  - 例えば、人口 50 万人以上の都市に 1, それ以外に 0 を入れる「大都市ダミー」を作成する

## データの要約・可視化

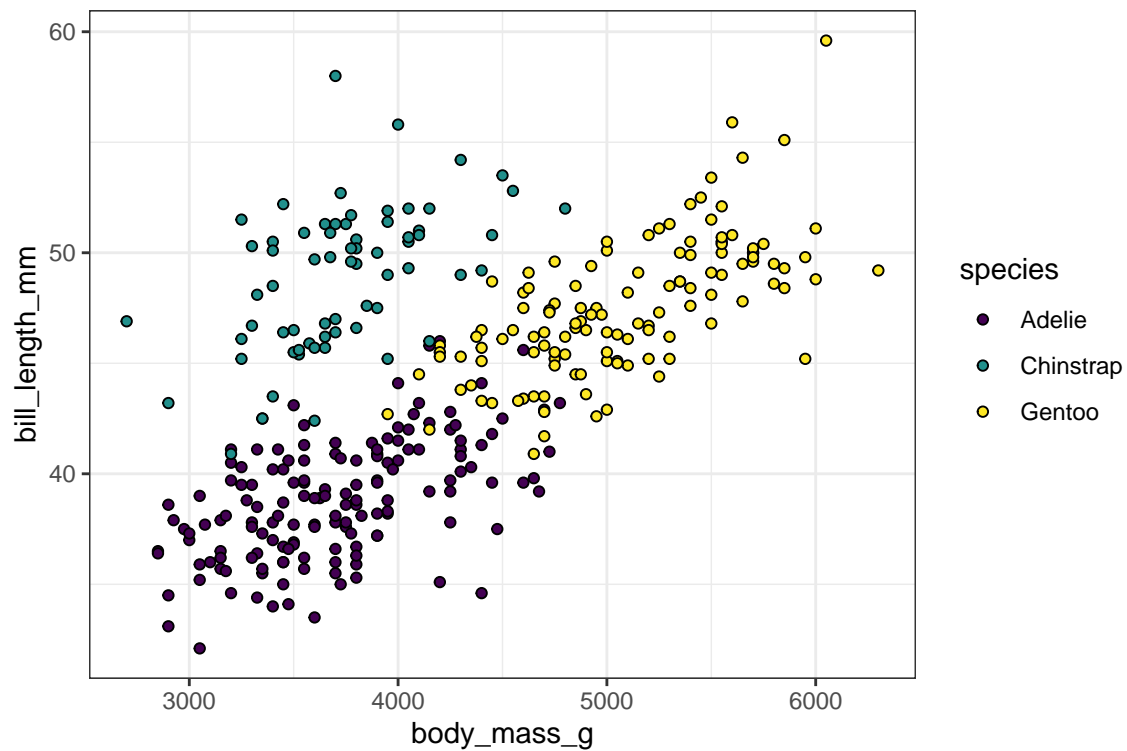
- 要約統計量の作成

変数名	平均	標準偏差	最小値	最大値
身長	173.5	6.2	163	192
体重	64.3	5.6	55	95
50m走	7.2	1.75	5.9	8.2
長座体前屈	57.4	20.1	-2	79
上体起こし	25.2	10.2	4	51

- データの概要を示す：各変数の平均・標準偏差など
- 性別ごと、年代ごとなど、カテゴリで分けて表示する場合も
- 可視化できるデータは可視化すると分かりやすい

## 可視化されたデータの例

```
df <- palmerpenguins::penguins
ggplot2::ggplot(df) +
  ggplot2::aes(x = body_mass_g, y = bill_length_mm, fill = species) +
  ggplot2::geom_point(colour = "black", shape = "circle filled") +
  ggplot2::scale_fill_viridis_d() +
  ggplot2::theme_bw()
```



## データ分析

- 回帰分析などの統計手法による分析を行い、結果をまとめる

$$\text{weight}_i = \alpha + \beta_1 \text{flipperSize}_i + \beta_2 \text{Species}_i + u_i$$

```
##
## Call:
## lm(formula = body_mass_g ~ flipper_length_mm + species, data = .)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -927.70 -254.82  -23.92   241.16 1191.68
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -4031.477    584.151  -6.901 2.55e-11 ***
## flipper_length_mm    40.705      3.071   13.255 < 2e-16 ***
## speciesChinstrap  -206.510     57.731  -3.577 0.000398 ***
## speciesGentoo     266.810     95.264   2.801 0.005392 **
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 375.5 on 338 degrees of freedom
##   ( 2 個の観測値が欠損のため削除されました )
## Multiple R-squared:  0.7826, Adjusted R-squared:  0.7807
## F-statistic: 405.7 on 3 and 338 DF,  p-value: < 2.2e-16
```

## レポートの作成

- Rで行った分析の結果を、Word やパワーポイントにまとめて出力、保存できる
- 習熟度によってはそのまま論文を書くことも可能、そこまでいかずとも色々と手間が省けて便利

## おまけ：データの取得

- ウェブサイトから情報を収集して分析を行いたい場合がある
- R のコードからウェブサイトを開き、中の要素を分析に使えるデータセットとして出力することができる

## 基本構造

- 基本的には、というか全ての命令は
  - R に命令を投げる→命令に従って計算 (描画・読み込みなど) を行う
  - 必要であればアウトプットを返す
- の繰り返し
- エレベーターの 3 階ボタンを押す→3 階に向かう、ドアを開ける
  - ボタンを押すエネルギーでエレベータが動いているわけではない
  - ワイヤーをどれだけ巻き取ればどれだけ上昇・下降するのか、ドアを開けるためにどの部分にどれだけ力を加えればいいのか、が 3 回ボタンを押したときに発せられる命令として書かれている
- この命令を一つ一つ書く作業を行う
- Excel や Word などのソフトウェアでは、こうした命令をコードを書く代わりにボタン操作などでまとめて行ってもらっている

## 参考

- 立命館大：森先生のサイトが大変勉強になります
  - 卒業論文のための R 入門

- その他、分からないことは google する力を付けましょう
  - “r (関数名)” とかで大体載ってます
  - GitHub など自身で作成したライブラリや関数の使い方などを解説しているものも多数存在
- R Tips
- RjpWiki

## その他

- ショートカット：マウスを極力使わない→作業効率の改善
- 共通の操作
  - Ctrl + X, C, V：順に切り取り・コピー・貼り付け
  - Ctrl + A：全範囲を選択
  - Ctrl + Z, Y：操作を戻す・進める
  - Ctrl + F：ウィンドウ内検索
  - Ctrl + S：(上書き) 保存
- R Studio 内の操作
  - Ctrl + Shift + N：新しいスクリプトを開く
  - Ctrl + O：保存したファイルを開く (Studio 以外でもだいたい使える)
  - Ctrl + W：タブを閉じる (Chrome とかブラウザも同じ)
  - Ctrl + Q：R Studio の終了
- Word, PowerPoint, Excel を扱うときもできるだけマウスを使わない
  - 慣れると作業効率がだいぶ変わる

## Setup

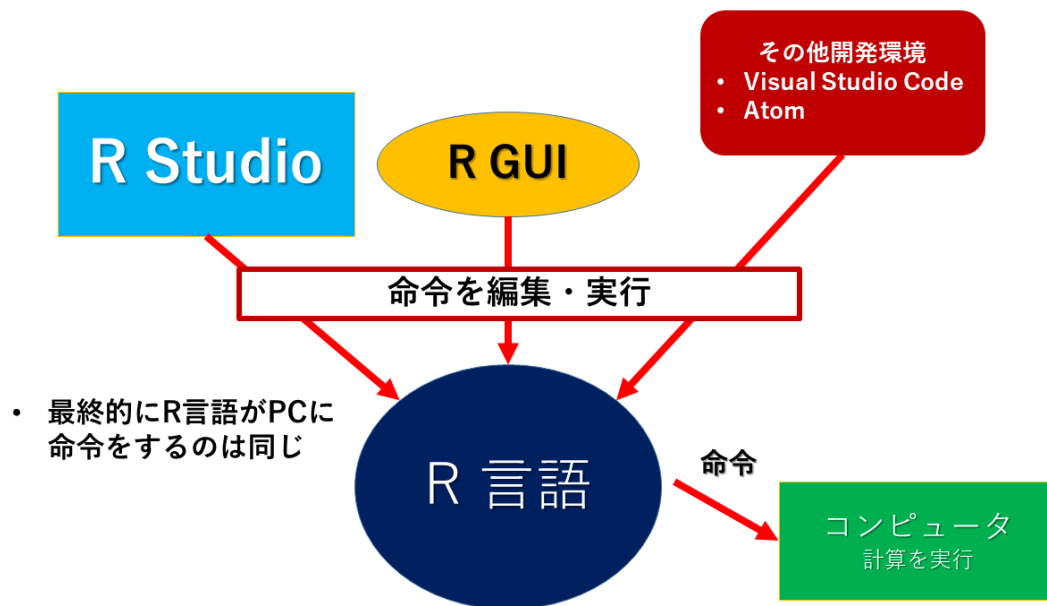
- R での作業に使うもの
  - 開発環境とは、R Studio のすすめ
- インストール
- 基本操作画面

## R での作業に必要なもの

- R 言語と R Studio の 2 つをインストール
  - R 言語をスムーズに利用するためのツール：開発環境が RStudio
  - デフォルトで R GUI と呼ばれる開発環境がインストールされるが、色々な理由から使いづらい
    - \* 罫線の引いてあるノートに書くか、自由帳に書くかみたいな違い
    - \* 最終的に R 言語で命令を書くのは同じ
  - その他にも Jupyter Notebook, VS Code, Atom など様々な開発環境があるので、好きなものを使えばよい

- R Studio Cloud: クラウド上で R Studio の機能を利用できるサービス
  - 無料の場合は月あたりの利用時間が制限されるなど問題はあるが、特に自前 PC のない人は検討してみてもよいかも

## R と R Studio



## インストール

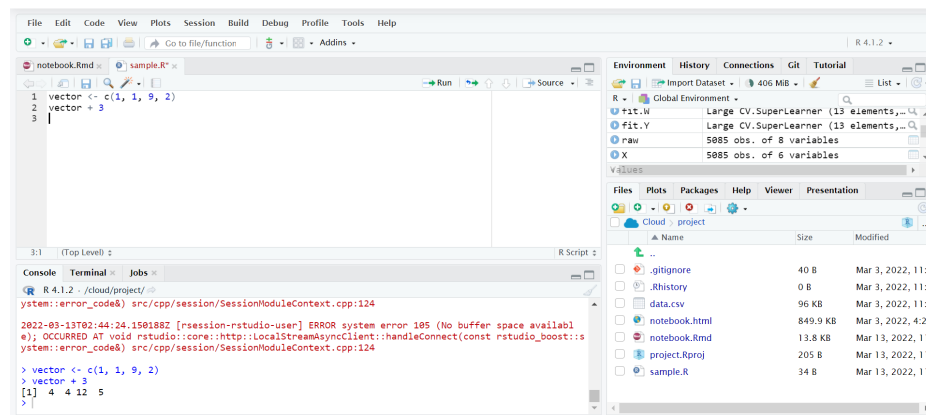
R のダウンロードはここから

RStudio Desktop

- **【定期】** 詰まったら 4 回生に訊いて下さい
- それぞれ必ず最新バージョンをダウンロードすること
  - 定期的にアップデートしておくのがおすすめ、たまに関数の仕様とかが大幅に変わる
  - たぶんいないと思いますが、Ver.4.0.0 以前のものを使用している 4 回生はインストールし直して下さい
- C ドライブに日本語フォントが含まれている人
  - C:/Users/.../...のところ
  - ファイルの保存などする際に非常に面倒なことになります
  - 特に動作で問題が出なければいいですが、コードの実行中に止まるなどあれば相談して下さい
  - 今から買う人は絶対に日本語フォントを入れないようにしましょう、一生

## R Studio の基本画面

- 4つの画面 (Pane) が表示される: コードを入力するのは Source, Console の2つ
  - Source: R スクリプトなどを編集・保存
  - Console: 実行するコードを直接入力・実行できる。
  - 右側の Pane では読み込んだデータフレームやオブジェクトの定義確認、図表出力のチェックなどが行える



## 触ってみる

- コンソールに適当なコードを入力してみよう

```
1 + 3 + 5
```

```
## [1] 9
```

- コンソールでは Enter, ソース (スクリプト) では Ctrl + Enter でコードを実行する
- 出力は>に続いて出る (文字色が変わるので分かりやすいはず)
- このスライド上では、コード、出力を四角囲みで、うち出力を##に続く形で表現する

## プロジェクトの作成、バージョン管理

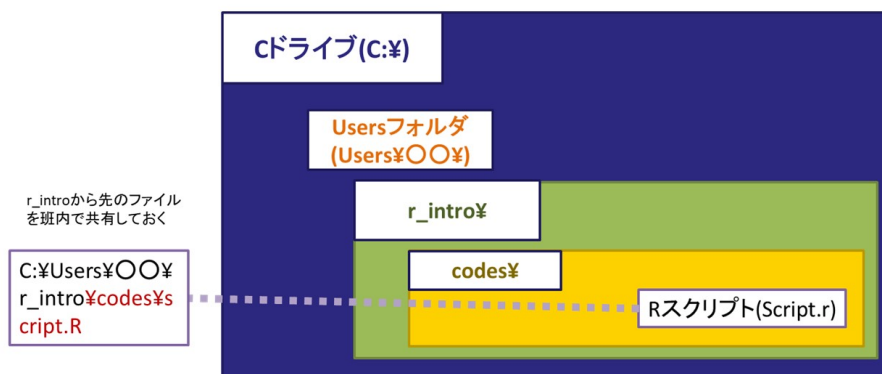
- ディレクトリとは
- ディレクトリの構造
- フォルダの共有方法
- R Studio でプロジェクトを作成する方法



## ディレクトリ

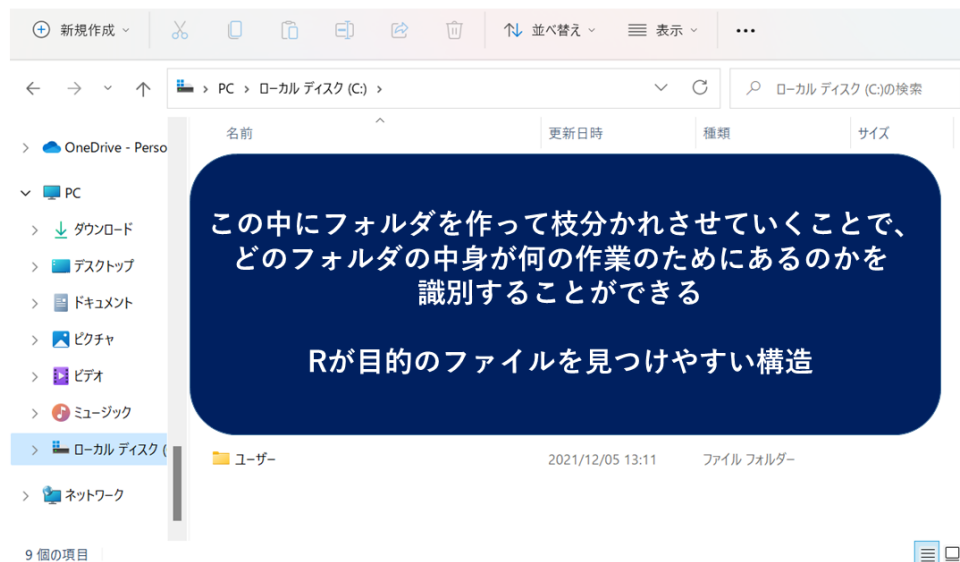
- ファイルやフォルダを参照する際に示す PC 内の住所
  - C:/...がそれ
  - 全てが同じ階層に入っているのは作業こそ楽だが、自分で参照するときに探すのが面倒
    - \* ゼミ論文に使う data.csv というファイルを保存→他の講義で同じ名前のファイルを受け取る  
→後から見たらどれがどれだか分からなくなる
- 整理の方法: 作業やファイルの種類ごとにフォルダを作り、異なる系統のファイルを識別
  - 何をやる作業のフォルダなのか?
  - データの種類: 整理する前の生データなのか、そのまま分析に使えるデータなのか?
  - アウトプット: 分析結果、図表
- R Studio では、特定のフォルダを「プロジェクト」として扱い、「誰の PC でも動くコード」を作ることができる

## ディレクトリの構造

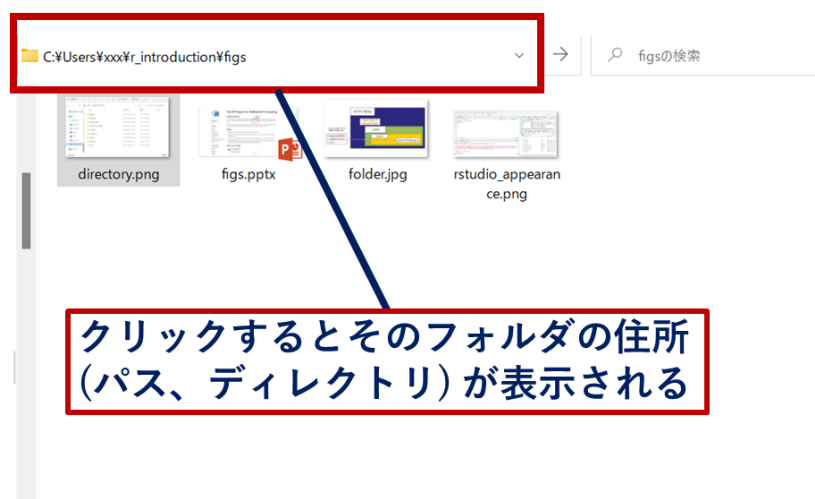


- 使いやすいフォルダの作成
  - 各自の PC で”r\_intro” フォルダを作り、必要なファイルをしまっておけば、誰が書いたコードでも個々人が PC で再現できる環境に
  - フォルダ名は\ (バックスラッシュ, ¥) を / (スラッシュ) に変えて記述

## フォルダ



## フォルダ (cont'd)



## ワーキングディレクトリの確認と変更

- ワーキングディレクトリ：R が見ているフォルダ
  - CSV などのファイルを読み込む際は、このフォルダから指定された名前のファイルを探す
- R Studio のプロジェクトを利用している時は、そのフォルダがある場所がワーキングディレクトリ
- `getwd` 関数で今のワーキングディレクトリを確認できる
  - `setwd` で変更できるが、基本的には相対パスで指定した方が再現性が高いのでおすすめ
- ファイル名：基本全て” “で囲む、ワーキングディレクトリの中にあるフォルダを開きたい場合は / で区切る
  - `"C:/folder1/folder2/data.csv"`

## 拡張子

- PC 上で扱うファイルには、それがどのアプリケーション (ソフト) で利用するものなのかが PC 側に伝わる印が付いていることが多い：拡張子
  - 例えば.csv は Microsoft Excel で開くと決めている (既定のアプリ)
  - .R は R スクリプトを表すことが分かるので R Studio や R GUI、.html はブラウザで開く
  - R Studio、Excel のそれぞれで csv ファイルを開いてみよう

## フォルダの共有方法

- フォルダの共有：データや書いたスクリプトを共有する
  - 何度も修正を加えたり、作った図表を共有するのはめんどい
  - 結果、分析をした人がスライドも全部作る...になりがち
- フォルダをメンバー間で同期する (共有する) ツールをマスターすると、グループでの作業が楽になる
  - Dropbox
  - GitHub
- 特に GitHub は RStudio と直接連携して簡単にクラウド共有・共有したデータのダウンロードも可能なので非常に便利
  - データをオープンで保有することになるので、扱うデータの種類によっては注意が必要

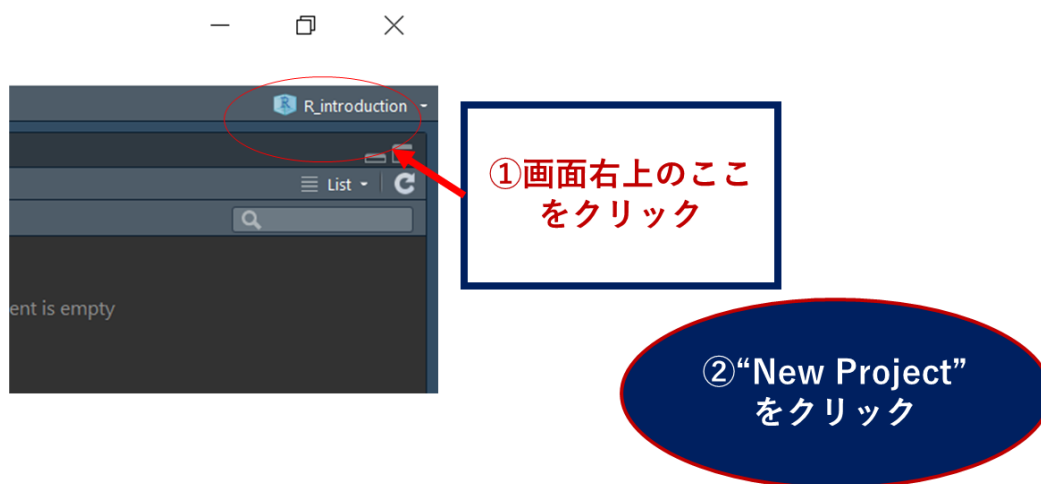
## フォルダの共有方法 (cont'd)

- Google Drive を使う方法もあり
  - Google Document, Spreadsheet, Slides を使った論文共同執筆はべんり
  - Office の Word, Excel, Powerpoint に比べると機能がかなり制限される弱点
  - 特にスライド作成に関しては不便なポイントが多いかも
- 文面は Google Drive で共有しながら作成しつつ、体裁を適宜 Word にダウンロードするなどして修正

する必要がある

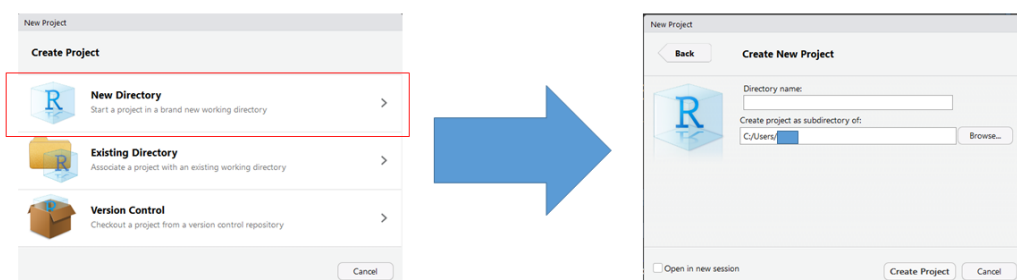
## プロジェクトの作り方

### R Studioのプロジェクトファイルを作成



## プロジェクトの作り方 (cont'd)

### R Studioのプロジェクトファイルを作成













“New Directory” →  
“New Project”

Directory name: フォルダの名前  
⇒今回は “r\_intro” とでもしておいて下さい

Create project as subdirectory of:  
フォルダを作成する場所  
⇒まあどこでもいいですが、本体に作成するようにして下さい  
“Browse” からフォルダを参照することも可

## ディレクトリの管理

 data	2022/04/14 21:35	ファイル フォルダー	
 documents	2022/03/12 19:32	ファイル フォルダー	
 figs	2022/04/17 19:47	ファイル フォルダー	
 introduction1_files	2022/04/17 22:19	ファイル フォルダー	
 script	2022/04/14 21:35	ファイル フォルダー	
 slides	2022/03/12 19:31	ファイル フォルダー	
 .gitignore	2022/03/10 11:09	txtfile	1 KB
 .RData	2022/04/15 22:50	R Workspace	1 KB
 .Rhistory	2022/04/17 11:53	RHISTORY ファイル	1 KB
 introduction.css	2022/04/17 11:07	カスケード スタイル シート...	1 KB

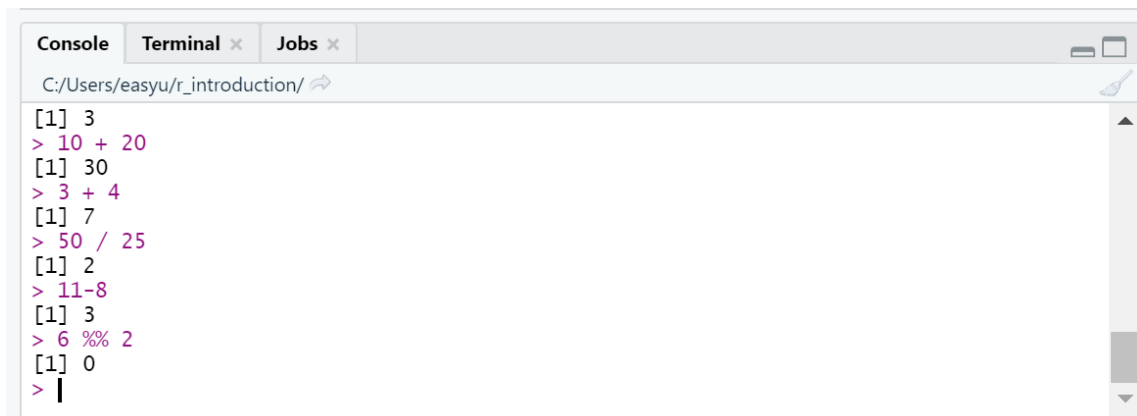
- 元データ、R スクリプト、作成した画像や表...など、作ったものをフォルダで整理すると便利
- とりあえず、作ったプロジェクトフォルダに「data」と「script」のフォルダを作成してみよう

## 基本操作

- コンソールに数値を打ち込む、計算する
- スクリプトの作成
- 変数の型
- オブジェクトに数値などを割り当てる
- 値を束にして扱う

## 入力画面

- コンソール (通常左下) の”>“が入力待ち状態を表す
- 命令を書いて Enter キーを押すと実行できる



```
Console Terminal x Jobs x
C:/Users/easyu/r_introduction/

[1] 3
> 10 + 20
[1] 30
> 3 + 4
[1] 7
> 50 / 25
[1] 2
> 11-8
[1] 3
> 6 %% 2
[1] 0
> |
```

## スクリプトの作成、演算子

- “r\_introduction” というプロジェクトを作成
- R Studio を開く
  - R Console にコードを直接入力して Enter、もしくは R スクリプトにコードを記述して Ctrl + Enter で実行
  - R スクリプトは Ctrl+Shift+N で作成可能
  - Ctrl + S で保存、上書き保存

```
1 + 2 * (3 / 4) # 掛け算は*で
```

```
## [1] 2.5
```

- その他、基本的な演算記号は R-Tipsなどを参照

Source: スクリプトやマークダウンファイルの編集



```
17 # 基本操作-----
18
19 1 + 2 * (3 / 4) # 掛け算は*で
20 13 %% 5 # 剰余
21
22 # 変数型-----
23 class(3.14) # class()はその変数の型を返す関数
24 class('1.90') # 文字列 character と判定される
25
26 "1" + "2" # エラーが出る
27
28 pi
29 value <- 8 # valueという文字列に8を代入
30 value + 10 # 今valueの値は8なので、8 + 10を計算した結果を返してくれる
31
32
```

- Ctrl + Enter で選択箇所のコードを実行
- 長いコードを途中で改行できることも利点
- Ctrl + S で適宜上書き保存できる

## 変数の型

- 数値だけでなく、文字も扱うことができる
  - 何を使っても良いわけではない：Excel も一緒
    - \* 電話番号を入力したのに頭の 0 が消える→ Excel が電話番号を数値として認識してしまっているから
    - \* Excel の場合、数値の形式を標準 (Excel に自己判断させる) から文字列に変更することで対処
  - 文字列を” “で囲って表記することで、それが文字列であることを R に伝えることができる
  - class 関数 (関数については後述) を使うと、その値の型が分かる

```
class(3.14) # class() はその変数の型を返す関数
```

```
## [1] "numeric"
```

```
class('1.90') # 文字列 character と判定される
```

```
## [1] "character"
```

## 変数の型 (cont'd)

- numeric, integer は足し引きできるが、character はできない

– integer は整数

```
"1" + "2" # エラーが出る
```

```
class(11L) # 整数の後ろに "L" を付けると整数として認識される
```

```
## [1] "integer"
```

## 変数の型 (cont'd)

- factor
  - 順序が付いた文字列
  - factor() で定義
  - アルファベット順や数字順以外の方法で並べたい場合に使う
  - まあ使うときにやればいいとおもいます

```
factor(c("January", "February", "March", "April")) # アルファベット順に並んでしまう
```

```
## [1] January February March April
```

```
## Levels: April February January March
```

```
factor(c("January", "February", "March", "April"),  
       levels = c("January", "February", "March", "April"))
```

```
## [1] January February March April
```

```
## Levels: January February March April
```

## オブジェクト

- では “で囲わずに入力した文字列はどう認識されるのか？”
  1. 特定の値と結びついている
    - pi: 円周率
  2. 自信が定義した任意の値が格納されている
    - オブジェクト: 数値やベクトル、データフレームやリストを格納、R Studio では右上の Environment タブに定義が表示される
    - 回帰分析などの計算結果を格納することも可能
    - <- を使って適当な値を定義する
    - wani <- 3 : wani というオブジェクトに値 3 を格納
- R Studio では Environment に定義したオブジェクトの中身が表示される



## オブジェクトの定義

```
pi # デフォルトで円周率が格納されている
```

```
## [1] 3.141593
```

```
value <- 8 # value という文字列に 8 を代入
```

```
value + 10 # 今 value の値は 8 なので、8 + 10 を計算した結果を返してくれる
```

```
## [1] 18
```

## 値を束にして扱う

- ぶっちゃけ、 $5 \times 5$  ぐらいの計算なら電卓でやればよい
- R、というか PC で計算ができる強みは、複数の計算や結果の保存を同時に行うことができる点
  - 人間の頭は複数の計算を同時に扱えない
  - 計量分析を行う上で不可欠な行列計算などと相性が良い
- R には複数の数値を扱うための束を作る記法が存在する
  - ベクトル：複数の値を順番付きで格納
  - 行列、データフレーム：行方向と列方向に展開、ベクトルを束にしたものともいえる
  - リスト：行列やデータフレームを束にして扱うことができる

## ベクトル

- ベクトル：複数の要素を含む列
  - `c(a, b, c, ...)` で定義される
  - 文字列など、他の数値型を利用しても OK

```
vec <- c(1192, 2960)
```

```
vec * 2
```

```
## [1] 2384 5920
```

- 関数 (後述) を用いて規則性のあるベクトルを簡単に定義することもできる
  - `seq(a, b, c)`:  $a$  から  $b$  まで、公差  $c$  の等差数列
    - \* 公差が 1 の場合は、`a:b` でも代替可能
  - `rep(a, b)`:  $a$  を  $b$  回繰り返す数列

```
seq(1, 10, 2)
```

```
## [1] 1 3 5 7 9
```

## データフレーム

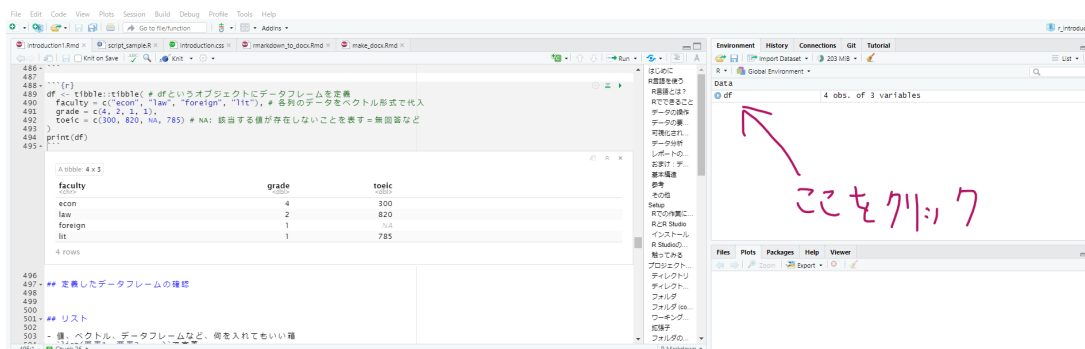
- 各行に観測単位 (個人、グループ、都道府県など)、各列に特定の情報を含んだデータ形式
  - 実際にデータ分析を行う際は、csv ファイルなどをこの形式で読み込むことで R で扱えるようにする
- 100 人の性別、学年、学部が分かるデータフレーム：100 行 × 3 列のデータフレームになる
- データフレームは `data.frame` 関数、もしくは `tibble` パッケージの `tibble` 関数で定義する

## データフレーム (cont'd)

```
df <- data.frame( # df というオブジェクトにデータフレームを定義
  faculty = c("econ", "law", "foreign", "lit"), # 各列のデータをベクトル形式で代入
  grade = c(4, 2, 1, 1),
  toeic = c(300, 820, NA, 785) # NA: 該当する値が存在しないことを表す=無回答など
)
print(df)
```

```
##   faculty grade toeic
## 1   econ     4   300
## 2   law      2   820
## 3 foreign    1    NA
## 4   lit      1   785
```

## 定義したデータフレームの確認



## リスト

- 値、ベクトル、データフレームなど、何を入れてもいい箱
- `list(要素 1, 要素 2, ...)` で定義
- 複数のデータフレームに対して同じ操作をしたい場合などに便利

```
listA <- list(data.frame(a = 1:5, b = 11:15, c = 100:104), df)
print(listA)
```

```
## [[1]]
##   a  b   c
## 1 1 11 100
## 2 2 12 101
## 3 3 13 102
## 4 4 14 103
## 5 5 15 104
##
## [[2]]
##   faculty grade toeic
## 1     econ     4   300
## 2      law     2   820
## 3 foreign     1    NA
## 4      lit     1   785
```

## リスト (cont'd)

```
print(listA[[1]]) # リストの中の一部の要素のみ利用する場合は、[[ ]] で指定する
```

```
##   a  b   c
## 1 1 11 100
## 2 2 12 101
## 3 3 13 102
## 4 4 14 103
## 5 5 15 104
```

- あんまり便利さが伝わらなさそうなので分析パートで後述

## 関数

- R で行う典型的な操作・計算を行う命令

- 操作を実行する対象となる値や、実行にあたって選択可能な様々なオプションを (関数名)(引数 1 = ., 引数 2 = ., ...) の形で記述

```
log(x = 100, base = 10) # 100 の対数、底 10 で計算
```

```
## [1] 2
```

```
log(x = 100, base = 5) # 底を 5 に変更
```

```
## [1] 2.861353
```

## 関数 (cont'd)

- 各関数において使用される引数の名前は決まっており、必要なオプションに対して一つひとつ情報を指定する
  - 引数には数値や文字列を取ったり、ベクトルやデータフレームを取る時もある
- 引数を指定しなければエラーが出るものと、指定しない場合のオプションを自動で選んでくれるものが存在
- パッケージ名:: 関数名 () でその関数がどのパッケージに属しているかを明示することもできる
- 実行結果のエラー
  - エラー：引数指定の不備などで計算が実行できなかった場合
  - 警告 (warning)：引数を自動補完した、計算結果に不備があるなどしたが、とりあえず結果は出た

## 関数とベクトル

- 関数の引数にベクトルやデータフレームを使用するのももちろん可能
- mean 関数、sd 関数は、それぞれ値の平均、標準偏差を計算する関数

```
values <- c(1:5, rep(3, 10), 4:10) #1, 2, ..., 5, 3, 3, ...
mean(values) # ベクトルの平均を計算
```

```
## [1] 4.272727
```

```
sd(values)
```

```
## [1] 2.333643
```

## パッケージの利用

- パッケージ：便利な関数をまとめて利用可能にする関数のセット
- 世界中のプログラマーが様々な分野における分析やデータ収集に関するパッケージを開発・公開している
  - これらを全て無料でダウンロードし、利用できるのが R の強み

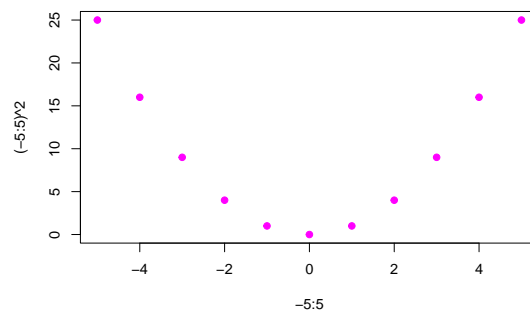
- `install.packages` 関数を用いて CRAN(公式のインストールサイト) からインストール
- その他、Github にパッケージを公開している人もいますので、その場合は別の関数を利用
- インストールしたパッケージは、毎回 R の起動時に `library` 関数、もしくは `require` 関数を用いて有効化する

```
install.packages("tidyverse") # tidyverse パッケージをインストール
library(tidyverse) # tidyverse パッケージを有効化：起動したときに毎回実行する
```

## 例：図形の描画

- $y = x^2$  のグラフの描画、定義域は-5 から 5
- `pch` はプロットの形を指定、`col` はプロットの色

```
graphics::plot(x = -5:5, y = (-5:5)^2, pch = 19, col = "magenta")
```



## データ操作

- tidyverse パッケージの活用
- R 内のデータセットを利用する
- データの中身を確認する
- csv ファイルの読み込み、保存
  - csv ファイルを保存する
  - PC に保存したデータを読み込む
  - ネットで公開されている csv ファイルを読み込む
  - その他のデータ形式
- データの詳細を調べる
  - サイトの参照
  - 要約統計量の作成：データの概観を掴む
- データを操作する
  - 不要な行・列の削除

- サンプルの分割
- 新しい列の作成・条件分岐
- 文字列の処理

## tidyverse パッケージ

- 外部からのデータの読み込みや整理、可視化 (次章で説明) に必要な関数を一通り揃えた便利なパッケージ
- パッケージの中に様々な目的で作成された複数のパッケージを含んでおり、実際に使用する時は `library(tidyverse)` で全ての関数を利用できる
  - readr: データの読み込みや書き出しを担当するパッケージ
  - ggplot2: データの可視化: 散布図やグラフの作成
  - dplyr: データの整形を行うパッケージ
  - purrr: 繰り返し計算 (ループ) を行うための関数
- `install.package` を活用してインストール

```
install.packages("tidyverse")
library(tidyverse)
```

## パイプ演算子について

- パイプ演算子: `%>%` で記述
- 一つのオブジェクト (多くはデータフレーム) について、複数の関数を連続して使用したい場合に活用
- 関数 1 `%>%` 関数 2 `%>%` 関数 3... のように記述
  - 中身は関数 3(関数 2(関数 1)) と同じ

```
1:10 %>% # ベクトルを作る、これが直後の関数で操作される
mean() %>% # 平均を求める
print() # 計算結果 (求めた平均を表示する)
```

```
## [1] 5.5
```

- より直感的に分かりやすいコードが書ける
- 計算過程をいちいち他のオブジェクトに置かなくていい

## データの取得

- 分析方法が決まったらデータを取得
- ここでは R で簡単に利用できるサンプルデータを取得して分析・可視化を行う
- `palmerpenguins` パッケージをインストール、`penguins_raw` データを使ってみる

```
install.packages("palmerpenguins")
```

- パッケージを読み込むとデータセットが利用できるようになる

```
library(palmerpenguins)
palmerpenguins::penguins
```

- “::” の前にパッケージ名、後ろに関数名 (or データフレーム名) を置く記法ならどのパッケージを利用しているのか分かりやすい、この辺はお好みで

## データを確認

Show  entries

Search:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
1	Adelie	Torgersen	39.1	18.7	181	3750	male	2007
2	Adelie	Torgersen	39.5	17.4	186	3800	female	2007
3	Adelie	Torgersen	40.3	18	195	3250	female	2007

Showing 1 to 3 of 344 entries

Previous  2 3 4 5 ... 115 Next

- penguin データの他、経済学向けのデータセットも利用できる
- AER パッケージをダウンロードすると良い

## Penguins データの概要

- ペンギンをとっ捕まえて大きさや重さを計測したデータ
- 元論文：Kristen B. Gorman ,Tony D. Williams, and William R. Fraser (2014)
- 論文引用のフォーマット
  - Gorman KB, Williams TD, Fraser WR (2014) Ecological Sexual Dimorphism and Environmental Variability within a Community of Antarctic Penguins (Genus *Pygoscelis*). PLoS ONE 9(3): e90081. [https://doi.org/10.1371/journal.pone.0090081]
  - 著者名、公刊年度、タイトル、学術誌名、その他 (URL など) の順で記載するのが一般的
  - “Cite” みたいなボタンを押すと簡単にフォーマットをコピーできることが多いです
- ここではこのデータセットを例に、基本的なデータ操作を学習する
- palmerpenguins パッケージには、成型前のデータ (penguins\_raw) も入っている

## データの一部を確認する

- `head`(データフレーム, 行数) でデータの行頭を表示させることができる
- `tail` なら下から

```
head(penguins_raw, 5)
```

```
## # A tibble: 5 x 17
##   studyName `Sample Number` Species      Region Island Stage `Individual ID`
##   <chr>          <dbl> <chr>          <chr>  <chr>  <chr> <chr>
## 1 PAL0708          1 Adelie Penguin ~ Anvers Torge~ Adul~ N1A1
## 2 PAL0708          2 Adelie Penguin ~ Anvers Torge~ Adul~ N1A2
## 3 PAL0708          3 Adelie Penguin ~ Anvers Torge~ Adul~ N2A1
## 4 PAL0708          4 Adelie Penguin ~ Anvers Torge~ Adul~ N2A2
## 5 PAL0708          5 Adelie Penguin ~ Anvers Torge~ Adul~ N3A1
## # ... with 10 more variables: `Clutch Completion` <chr>, `Date Egg` <date>,
## #   `Culmen Length (mm)` <dbl>, `Culmen Depth (mm)` <dbl>,
## #   `Flipper Length (mm)` <dbl>, `Body Mass (g)` <dbl>, Sex <chr>,
## #   `Delta 15 N (o/oo)` <dbl>, `Delta 13 C (o/oo)` <dbl>, Comments <chr>
```

- R Studio 中では `View` 関数を使うと別ウィンドウで開くのでこれが見やすい

## 外部ファイルでのデータの扱い

- R で利用できるデータセットはチュートリアルに便利だが、実証分析にそのまま使うのは難しい
  - データの種類
    - \* 二次データ：官庁や大学、民間のリサーチ会社やデータサイトなど、自分以外が作成したデータを借りる (目的に合うデータがあればこちらがベター)
    - \* 一次データ：自作のアンケートや経済実験などを実施し、自らデータセットを作成する
  - いずれにせよ、外部から取得してきたデータを R に読み込んでもらう必要がある
  - csv (comma separated values) ファイルは、代表的なデータ保存方法：データがカンマで区切られている
- ただし、取得したデータにはノイズ (欠損値がある観測や、質問を誤解している回答など) が含まれており、分析を行うためにこれらの問題に対処する必要がある
  - 分析の種類によっては必要ない情報が含まれていることも
  - データを読み込むと同時に、整理したものを保存しておく機能も必要

## 取得したデータを csv ファイルとして保存

- `readr::write_excel_csv` 関数を活用



- 保存したいデータフレーム名と、保存先のファイル名を指定、実行すれば、ファイルを csv 形式で保存してくれる
- 保存したものを Excel で開くことも可能
- プロジェクトファイル内に”data” というフォルダを作り、その中にデータを保存してみよう

```
penguin_df <- palmerpenguins::penguins
readr::write_excel_csv(penguin_df, file = "data/penguins.csv")
```

## CSV ファイルからデータフレームを読み込み

- デフォルトで利用できる read.csv 関数を使っても良いが、readr パッケージの read\_csv 関数の方が速い
- さっき保存したファイルをオブジェクト df に代入しておく
  - Tab キーを押すとフォルダ内のキーワード検索も可能

```
df <- readr::read_csv("data/penguins.csv")
```

```
## Rows: 344 Columns: 8
```

```
## -- Column specification -----
```

```
## Delimiter: ","
```

```
## chr (3): species, island, sex
```

```
## dbl (5): bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g, year
```

```
##
```

```
## i Use `spec()` to retrieve the full column specification for this data.
```

```
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

- read\_csv 関数の場合、読み込んだデータの各列の型を報告してくれる：num, chr, int, ...

## ネットに公開されている URL から csv ファイルを読み込む

- csv ファイルがネット上に公開されていることもある：この場合も、read\_csv 関数で読み込み可能
  - URL の文字列を入力し、read\_csv で読み込む

```
url <- 'https://raw.githubusercontent.com/chadwickbureau/register/master/data/people.csv' # URL は一俣
df <- read_csv(url)
```

## その他のデータ形式

- csv 以外にも様々なデータ形式が存在、それぞれ対応した関数を使うと読み込み可能：基本 read\_○○ などの関数名が指定されていることが多い
  - txt 形式のファイルは csv, tsv などとしてそのまま読み込める場合もある

- xlsx ファイル
- 大容量のデータを保存したい & Excel などのソフトでデータを扱う予定がない場合は rds ファイルと呼ばれる R 向けのバイナリファイルとして保存すると、ハードの保存領域を節約できる：read\_rds, write\_rds
- その他、知らないデータ形式が来たらとりあえず R で開けるか確認してみる

```
penguin_df <- palmerpenguins::penguins
readr::write_rds(penguin_df, file = "data/penguins.rds")
```

## データの詳細を調べる

- データを取得したら (取得する前に)、そのデータがどのような情報を含んでいるのかを必ず確認する
  - 一次データを取得する場合：質問項目や実験のデザインを慎重に検討する
- ウェブで公開されている二次データには各列に格納されている情報 (documentation) や、アンケートの質問用紙が記載されている
- 例) 大阪大学くらしの好みと満足度についてのアンケート：URL
  - 調査票
- データの簡単な要約も記載されていることがある

## 要約統計量

- 整理が終了したら (あるいは、整理を行うために)、データに含まれる情報の概観を記述する
- データセットに含まれる各列の平均や標準偏差、レンジなどの統計量をまとめて記載する表

変数名	平均	標準偏差	最小値	最大値
身長	173.5	6.2	163	192
体重	64.3	5.6	55	95
50m走	7.2	1.75	5.9	8.2
長座体前屈	57.4	20.1	-2	79
上体起こし	25.2	10.2	4	51

- 何らかの介入の効果 (法改正や就学、ナッジによる行動変容など...) を検証したい場合は、介入群と統制群が似た集団であることを確認するため、両者を分けて要約統計量を作成する場合も
- 様々な関数を利用できるので、目的に応じて好きなものを使う

## ペンギンデータの要約統計量

- 最もシンプルなのは summary 関数
  - 各変数の主要な統計量を記載してくれる

```
palmerpenguins::penguins %>%
  summary()
```

```
##      species      island bill_length_mm bill_depth_mm
## Adelie      :152  Biscoe      :168  Min.      :32.10  Min.      :13.10
## Chinstrap: 68  Dream      :124  1st Qu.:39.23  1st Qu.:15.60
## Gentoo      :124  Torgersen: 52  Median :44.45  Median :17.30
##                                     Mean      :43.92  Mean      :17.15
##                                     3rd Qu.:48.50  3rd Qu.:18.70
##                                     Max.      :59.60  Max.      :21.50
##                                     NA's      :2      NA's      :2
## flipper_length_mm body_mass_g      sex      year
## Min.      :172.0    Min.      :2700  female:165  Min.      :2007
## 1st Qu.:190.0    1st Qu.:3550  male   :168  1st Qu.:2007
## Median :197.0    Median :4050  NA's   : 11  Median :2008
## Mean      :200.9    Mean      :4202                      Mean      :2008
## 3rd Qu.:213.0    3rd Qu.:4750                      3rd Qu.:2009
## Max.      :231.0    Max.      :6300                      Max.      :2009
## NA's      :2      NA's      :2
```

## データを整理する

- 取得したデータはそのまま分析に使えるわけではない
  - 必要な情報が抜け落ちている場合や、アンケートの設問の誤解等で明らかにおかしな回答が存在することがある
  - 情報が全て入っているが、その分析に関しては不適当な観測が含まれる場合も
  - 分析の対象でない観測と一緒に含まれている場合 (例えば男性労働者の行動に注目したいときに、女性労働者を含めたまま分析を行うのは不適当)
  - 元データの情報を用いて、新たに変数を作りたい場合もある (例えば、5段階で回答された幸福度を1,2と3-5の2つに再分類した変数を作成する)
- これらの問題を解決し、分析に利用可能なデータセットを作成することを「データの整理」「データクリーニング」と呼ぶ
- tidyverse パッケージから、便利な関数がたくさん提供されている
  - filter 関数：行を絞る
  - select 関数：列を絞る
  - mutate 関数：新しい変数を作成する
  - split 関数：データフレームをリストに分割する

## 列名の付け直し

- 列名は原則アルファベットとアンダースコア ( \_ ) で付けておく方がエラーが起こりにくい
  - 日本語がギリセーフ、アンダースコア以外の記号や引用符は使わない
  - 数字は 2 文字目以降にしか使えない
- もし入っている場合は、`rename` 関数もしくは `set_names` 関数を用いて列名を変更した方が良い

```
palmerpenguins::penguins %>%
  dplyr::rename(weight = body_mass_g) %>% # body_mass_g を weight に
  head(2)

## # A tibble: 2 x 8
##   species island   bill_length_mm bill_depth_mm flipper_length_mm weight sex
##   <fct>   <fct>         <dbl>         <dbl>         <int>  <int> <fct>
## 1 Adelie  Torgersen         39.1           18.7           181   3750 male
## 2 Adelie  Torgersen         39.5           17.4           186   3800 female
## # ... with 1 more variable: year <int>
```

## 列名の付け直し (cont'd)

- `set_names` はそのデータセットの列名をまとめて変更する：列数と同じ長さの chr 型ベクトルを指定

```
newnames <- c("species", "island", "bill_lg", "bill_dep", "flipper", "weight", "sex", "yr")
palmerpenguins::penguins %>%
  purrr::set_names(newnames) %>%
  head(2) %>%
  print()
```

```
## # A tibble: 2 x 8
##   species island   bill_lg bill_dep flipper weight sex      yr
##   <fct>   <fct>         <dbl>   <dbl>   <int>  <int> <fct> <int>
## 1 Adelie  Torgersen         39.1     18.7     181   3750 male   2007
## 2 Adelie  Torgersen         39.5     17.4     186   3800 female 2007
```

## データを絞る

- 条件にあてはまる行を抜き出す：`filter` 関数を利用
- 条件式の書き方
  - 参照する列を記述し、「〇〇に一致する場合」「〇〇より大きい/小さい場合」などの条件を R のルールに従って記述する

- \* `A == B`: A 列の内容が B と完全に一致, B が文字列の場合はクォーテーションで囲う
- \* `A >= B`: B 以上の値, 逆なら `<=`
- \* `A %in% c(B, C, D)`: A 列の要素がベクトルの要素 B, C, D のいずれかに一致
- \* 否定は `!`: `A != B`, `!(A %in% c(a, b, c))`
- and 条件は `&`, or 条件は `|` で繋ぐ: 条件 1 | 条件 2 なら、1,2 いずれかの条件にあてはまるもの

```
df <- palmerpenguins::penguins %>%
  dplyr::filter(species == "Gentoo") # ジェンツーペンギンのみに絞る

print(df %>% head(2))
```

```
## # A tibble: 2 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g sex
##   <fct>   <fct>         <dbl>         <dbl>         <int>         <int> <fct>
## 1 Gentoo  Biscoe           46.1           13.2           211           4500 fema~
## 2 Gentoo  Biscoe           50            16.3           230           5700 male
## # ... with 1 more variable: year <int>
```

## 欠損値

- 欠損値 (分からない、無回答などの理由で値が入っていない) がある場合、そのセルを取り除いて分析を行うことがある
- R では、NA で表す: 型は勝手に判断してくれる場合と、してくれない場合とがある
  - `NA_real_`: num 型
  - `NA_character`: chr 型
- 「欠損である」という条件を表す関数: `is.na()`
  - 括弧内の値が欠損であるかどうかを返す: `filter` などの条件に指定

```
palmerpenguins::penguins %>% nrow() # 列数を返す
```

```
## [1] 344
```

```
palmerpenguins::penguins %>%
  filter(!is.na(body_mass_g)) %>% # 体重が欠損している個体を除外
  nrow() # 上より列数が減っている
```

```
## [1] 342
```

## 情報を絞る

- 全ての列を必要としない場合: `select` 関数を利用
- 残したい列を順番に列挙するだけで OK。列番号でもよい

- 落としたい列を-(マイナス) で指定してもよい

```
df <- palmerpenguins::penguins %>%
  dplyr::select(species, sex, 5, 6)

print(df %>% head(4))
```

```
## # A tibble: 4 x 4
##   species sex   flipper_length_mm body_mass_g
##   <fct>   <fct>             <int>         <int>
## 1 Adelie male             181           3750
## 2 Adelie female          186           3800
## 3 Adelie female          195           3250
## 4 Adelie <NA>              NA              NA
```

- 取り出した列をベクトルとして利用したい場合：pull 関数で抜き出し可能

## 新しい変数を作成する

- 今ある情報を元に、新しい列を作成することがある：mutate 関数を利用する
  - 列の変数型が数値の場合：四則演算の演算子をそのまま利用できる
  - 特定の条件にあてはまるか否かを判定して数値を代入したい場合：if\_else 関数、case\_when 関数を利用

```
df <- palmerpenguins::penguins %>%
  mutate(
    flipper_length_2 = flipper_length_mm^2, # 羽根の長さの 2 乗
    weight_size = if_else(condition = body_mass_g >= 4050, true = "L", false = "F"),
    flipper_size_3 = case_when(
      body_mass_g <= 3550 ~ "S",
      body_mass_g >= 3550 & body_mass_g <= 4750 ~ "M",
      TRUE ~ "L" # TRUE はそれ以外
    )
  )
```

## 作成した変数を確認

```
## # A tibble: 344 x 6
##   species island   body_mass_g flipper_length_2 weight_size flipper_size_3
##   <fct>   <fct>             <int>         <dbl> <chr>         <chr>
## 1 Adelie Torgersen           3750           32761 F          M
```

```
## 2 Adelie Torgersen      3800      34596 F      M
## 3 Adelie Torgersen      3250      38025 F      S
## 4 Adelie Torgersen       NA      NA <NA>      L
## 5 Adelie Torgersen      3450      37249 F      S
## 6 Adelie Torgersen      3650      36100 F      M
## 7 Adelie Torgersen      3625      32761 F      M
## 8 Adelie Torgersen      4675      38025 L      M
## 9 Adelie Torgersen      3475      37249 F      S
## 10 Adelie Torgersen     4250      36100 L      M
## # ... with 334 more rows
```

## サンプルを分割する

- サンプルを特定の変数の値ごとに分割したい時がある
  - 例えば、オスのサンプルとメスのサンプルを分割する
  - `filter` 関数を使ってもいいが、例えば雄雌それぞれのサンプルに同じ操作を適用したい場合などは、コードが冗長になる場合がある
  - `split` 関数と `pull` 関数を組み合わせて使うと、データフレームをリスト形式で分割できる

```
df_split <- palmerpenguins::penguins %>%
  select(species, island, sex) %>% # 表示の都合上列数を限定
  split(pull(., sex)) # 分割の基準にしたい変数を入力
print(df_split)
```

```
## $female
## # A tibble: 165 x 3
##   species island    sex
##   <fct>    <fct>    <fct>
## 1 Adelie Torgersen female
## 2 Adelie Torgersen female
## 3 Adelie Torgersen female
## 4 Adelie Torgersen female
## 5 Adelie Torgersen female
## 6 Adelie Torgersen female
## 7 Adelie Torgersen female
## 8 Adelie Torgersen female
## 9 Adelie Biscoe    female
## 10 Adelie Biscoe    female
## # ... with 155 more rows
##
```

```
## $male
## # A tibble: 168 x 3
##   species island sex
##   <fct>   <fct>   <fct>
## 1 Adelie  Torgersen male
## 2 Adelie  Torgersen male
## 3 Adelie  Torgersen male
## 4 Adelie  Torgersen male
## 5 Adelie  Torgersen male
## 6 Adelie  Torgersen male
## 7 Adelie  Torgersen male
## 8 Adelie  Biscoe   male
## 9 Adelie  Biscoe   male
## 10 Adelie Biscoe   male
## # ... with 158 more rows
```

## その他データ操作関連の関数

- stringr, stringi パッケージは文字列の処理に便利
  - 先頭から何文字、指定した条件に合致した文字を抜き出しなど、character 型の文字列を操作するのに非常に便利

## もっと便利な要約統計量

### dplyr パッケージの利用

- dplyr: tidyverse パッケージに含まれるデータ操作系の関数の一つ
- group\_by 関数は、以降の操作を指定した変数ごとに行うことを宣言する関数
  - 例えば、group\_by(species) とすると、以降の操作はペンギンの種類ごとに行われる
- summarise 関数は、データフレームの指定された列を任意の関数で集計するための関数

```
palmerpenguins::penguins %>%
  summarise(
    mean_bill_length_mm = mean(bill_length_mm, na.rm = T), # くちばしの長さの平均を取る、欠損値は除外
    mean_flipper_length_mm = mean(flipper_length_mm, na.rm = T),
    mean_body_mass_g = mean(body_mass_g, na.rm = T),
  )

## # A tibble: 1 x 3
##   mean_bill_length_mm mean_flipper_length_mm mean_body_mass_g
##               <dbl>               <dbl>               <dbl>
```



```
## 1          43.9          201.          4202.
```

- 任意の統計量、変数を自由に選択肢して表を作ることができる

## group\_by を組み合わせる

```
palmerpenguins::penguins %>%
  group_by(species) %>% # 種類ごとに
  summarise(
    N = n(), # サンプルサイズ
    mean_bill_length_mm = mean(bill_length_mm, na.rm = T), # くちばしの長さの平均を取る、欠損値は除外
    mean_flipper_length_mm = mean(flipper_length_mm, na.rm = T),
    mean_body_mass_g = mean(body_mass_g, na.rm = T),
  )
```

```
## # A tibble: 3 x 5
##   species      N mean_bill_length_mm mean_flipper_length_mm mean_body_mass_g
##   <fct>      <int>          <dbl>          <dbl>          <dbl>
## 1 Adelie    152            38.8            190.            3701.
## 2 Chinstrap  68            48.8            196.            3733.
## 3 Gentoo   124            47.5            217.            5076.
```

- ペンギンの種類間での平均の比較が可能に

## group\_by を組み合わせる

- 複数変数を指定しても OK

```
palmerpenguins::penguins %>%
  group_by(sex, species) %>% # 種類と性別ごとに
  summarise(
    N = n(),
    mean_bill_length_mm = mean(bill_length_mm, na.rm = T), # くちばしの長さの平均を取る、欠損値は除外
    mean_flipper_length_mm = mean(flipper_length_mm, na.rm = T),
    mean_body_mass_g = mean(body_mass_g, na.rm = T),
  )
```

```
## `summarise()` has grouped output by 'sex'. You can override using the `.groups` argument.
```

```
## # A tibble: 8 x 6
## # Groups:   sex [3]
##   sex      species      N mean_bill_length_mm mean_flipper_leng~ mean_body_mass_g
```

```
##   <fct> <fct>      <int>          <dbl>          <dbl>          <dbl>
## 1 female Adelie      73            37.3            188.           3369.
## 2 female Chinstrap   34            46.6            192.           3527.
## 3 female Gentoo     58            45.6            213.           4680.
## 4 male   Adelie      73            40.4            192.           4043.
## 5 male   Chinstrap   34            51.1            200.           3939.
## 6 male   Gentoo     61            49.5            222.           5485.
## 7 <NA>   Adelie      6            37.8            186.           3540
## 8 <NA>   Gentoo     5            45.6            216.           4588.
```

- 同様に、標準偏差などを掲載すると良い

## skimr パッケージの利用

- skim 関数は各行にデータセットのカラム (列)、各列に統計量を記載した扱いやすい記述統計量を作成してくれる

```
library(skimr)
descriptive <- palmerpenguins::penguins %>%
  skim()
print(descriptive)

## -- Data Summary -----
##                               Values
## Name                         Piped data
## Number of rows                344
## Number of columns             8
## -----
## Column type frequency:
##   factor                      3
##   numeric                    5
## -----
## Group variables              None
##
## -- Variable type: factor -----
## # A tibble: 3 x 6
##   skim_variable n_missing complete_rate ordered n_unique
## * <chr>        <int>        <dbl> <lgl>      <int>
## 1 species      0          1 FALSE      3
## 2 island       0          1 FALSE      3
## 3 sex          11        0.968 FALSE      2
```

```
## top_counts
## * <chr>
## 1 Ade: 152, Gen: 124, Chi: 68
## 2 Bis: 168, Dre: 124, Tor: 52
## 3 mal: 168, fem: 165
##
## -- Variable type: numeric -----
## # A tibble: 5 x 11
##   skim_variable    n_missing complete_rate   mean     sd    p0    p25    p50
## * <chr>          <int>         <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 bill_length_mm      2         0.994  43.9  5.46  32.1  39.2  44.4
## 2 bill_depth_mm       2         0.994  17.2  1.97  13.1  15.6  17.3
## 3 flipper_length_mm   2         0.994  201.  14.1  172   190   197
## 4 body_mass_g         2         0.994 4202. 802.   2700  3550  4050
## 5 year                0          1    2008.  0.818 2007  2007  2008
##   p75    p100 hist
## * <dbl> <dbl> <chr>
## 1  48.5   59.6 <U+2583><U+2587><U+2587><U+2586><U+2581>
## 2  18.7   21.5 <U+2585><U+2585><U+2587><U+2587><U+2582>
## 3  213    231  <U+2582><U+2587><U+2583><U+2585><U+2582>
## 4 4750   6300  <U+2583><U+2587><U+2586><U+2583><U+2582>
## 5 2009   2009  <U+2587><U+2581><U+2587><U+2581><U+2587>
```

## 質的変数と量的変数

- yank 関数：skim で要約した列のうち、特定の型を持つ値の要約のみを抜き出して記載する関数

```
palmerpenguins::penguins %>%
  skim() %>%
  yank(., skim_type = "numeric") %>% # numeric 型の要約統計量を表示
  print()
```

```
##
## -- Variable type: numeric -----
## # A tibble: 5 x 11
##   skim_variable    n_missing complete_rate   mean     sd    p0    p25    p50
## * <chr>          <int>         <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 bill_length_mm      2         0.994  43.9  5.46  32.1  39.2  44.4
## 2 bill_depth_mm       2         0.994  17.2  1.97  13.1  15.6  17.3
## 3 flipper_length_mm   2         0.994  201.  14.1  172   190   197
```

```
## 4 body_mass_g          2          0.994 4202.  802.    2700   3550   4050
## 5 year              0          1      2008.    0.818 2007   2007   2008
##      p75    p100 hist
## *   <dbl>  <dbl> <chr>
## 1    48.5    59.6 <U+2583><U+2587><U+2587><U+2586><U+2581>
## 2    18.7    21.5 <U+2585><U+2585><U+2587><U+2587><U+2582>
## 3   213     231   <U+2582><U+2587><U+2583><U+2585><U+2582>
## 4  4750    6300   <U+2583><U+2587><U+2586><U+2583><U+2582>
## 5  2009    2009   <U+2587><U+2581><U+2587><U+2581><U+2587>
```

## グループごとの統計量

- skim にも group\_by 関数を適用可能

```
sum <- palmerpenguins::penguins %>%
  group_by(island) %>%
  skim() %>%
  yank(., skim_type = "numeric")

sum %>%
  filter(skim_variable == "bill_length_mm") %>%
  filter(island %in% c('Biscoe', 'Dream')) %>%
  select(skim_variable, island, complete_rate, mean, sd) # select で必要な統計量だけ出す
```

Variable type: numeric

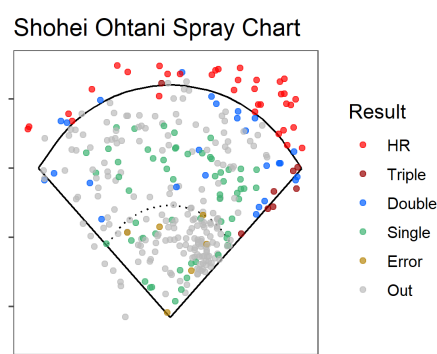
skim_variable	island	complete_rate	mean	sd
bill_length_mm	Biscoe	0.99	45.26	4.77
bill_length_mm	Dream	1.00	44.17	5.95

## データの概観：可視化

- データを可視化する：ggplot2 パッケージの利用
- ggplot の記法を覚える
  - ヒストグラム
  - 散布図
  - 棒グラフ

## データを可視化する

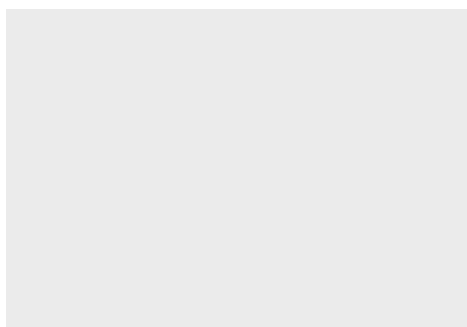
- 前章では要約統計量を作成することでデータの概観を把握する方法を学習
- しかし、変数の分布や二変数間の相関、時系列での数値の変化など、要約統計量だけでは捉えきれない性質も存在
- データをグラフの形で可視化することで、分かりやすい・伝わりやすい数値化が可能
- デフォルトで利用できる graphics パッケージは早くて便利だが、ggplot2 パッケージならより詳細&手軽な操作で美しいグラフを描画できる



## ggplot の記法

- ggplot2 パッケージの関数はやや特殊：たくさんの関数を「足し算」することでグラフに手を加えていく
- ggplot 関数で台紙を作る：どのデータフレームを使って可視化するのかもここで宣言

```
df <- palmerpenguins::penguins  
ggplot2::ggplot(data = df) # この時点では白紙
```



## ggplot の記法 (cont'd)

- ggplotによるグラフの作成
  - グラフを描くための関数や、  
図のオプションを設定するための関数が色々ある  
→それらを足し算の形でつなげて一つのグラフを作る



## 図形の描画：軸の設定

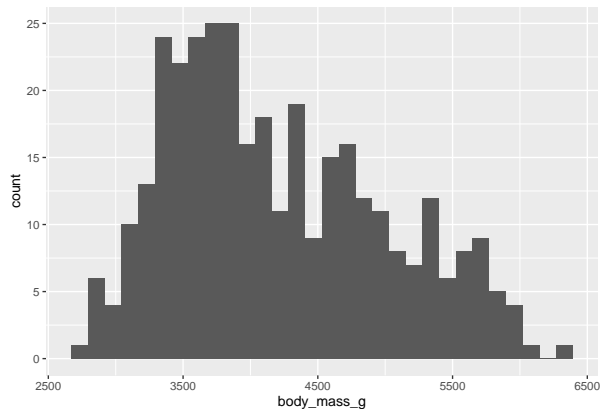
- 例として、ペンギンの体重のヒストグラムを作ろう：体重という 1 変数の分布を見る
- `aes` 関数を使い、グラフの横軸・縦軸や色分けの情報を加えていく
  - 横軸: `x`, 縦軸: `y`, グループ分け: `group`, 色分け: `colour`, 塗りつぶし: `shape`, 形: `shape` あたりを覚えておく
  - 今回はヒストグラムなので、「横軸が体重ですよ」という情報を渡してやればよい
- その上で、ヒストグラムを描画する関数 `geom_histogram` 関数を加える

## ヒストグラムの描画

```
ggplot2::ggplot(data = df) +  
  aes(x = body_mass_g) +  
  geom_histogram()
```

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

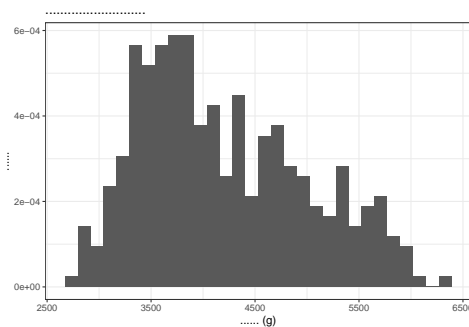
```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



## 細かな体裁の変更

- グラフの背景や軸ラベルを設定する
  - 背景: `theme_〇〇`関数で様々な書式を設定できるので、お好みで好きなものを利用。なるべくシンプルなものが多い
  - 軸ラベルなど: `labs` 関数で文字を指定、各オプションが対応する場所のテキストが変更される

```
ggplot2::ggplot(data = df) +
  aes(x = body_mass_g, y = ..density..) +
  geom_histogram() +
  theme_bw() + # 背景の設定
  labs(x = "体重 (g)", y = "密度", title = "ペンギんの体重分布")
```

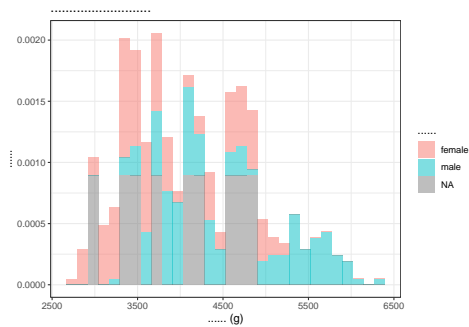


## グループを分けて描画する

- `aes` に `group` 引数を指定すると、それぞれの描画をグループごとにやってくれる
- 同様に `colour`・`fill` を指定すると、グループごとに色分けしてくれる
  - 通常 `colour` は縁取り、`fill` は塗りつぶし部分を担当

```
ggplot2::ggplot(data = df) +
  aes(x = body_mass_g, y = ..density..., group = sex, fill = sex) +
```

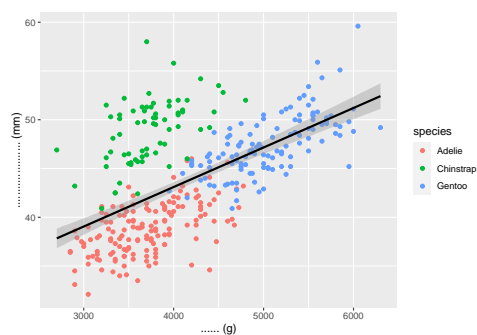
```
# 横軸に..density..を指定すると、計数の代わりに密度を出してくれる
geom_histogram(alpha = .5) + # alpha は透明にするためのオプション
theme_bw() + # 背景の設定
labs(x = "体重 (g)", y = "密度", title = "ペンギンの体重分布", fill = "性別")
```



## 散布図：二変数間の関係を可視化する

- 散布図は二変数間の関係を示すのに便利：geom\_point 関数を利用
- 横軸と縦軸両方に変数が必要
- 

```
ggplot2::ggplot(df) +
  ggplot2::aes(x = body_mass_g, y = bill_length_mm, colour = species) +
  ggplot2::geom_point() +
  ggplot2::geom_smooth(method = "lm", colour = "black") + # 最小二乗法で近似曲線を描画
  ggplot2::labs(x = "体重 (g)", y = "くちばしの長さ (mm)")
```

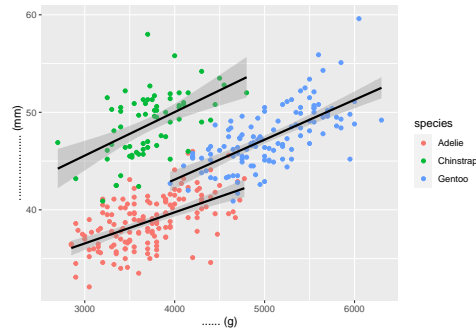


## 散布図 (cont'd)

```
ggplot2::ggplot(df) +
  ggplot2::aes(x = body_mass_g, y = bill_length_mm, colour = species, group = species) +
  ggplot2::geom_point() +
```



```
ggplot2::geom_smooth(method = "lm", colour = "black") + # 最小二乗法で近似曲線を描画
ggplot2::labs(x = "体重 (g)", y = "くちばしの長さ (mm)")
```

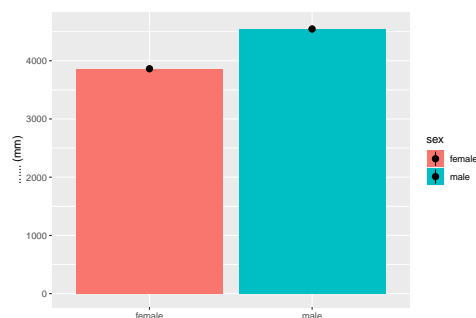


- group 化すると種類ごとに近似曲線を引いてくれる

## 棒グラフ：グループ間の変数の比較

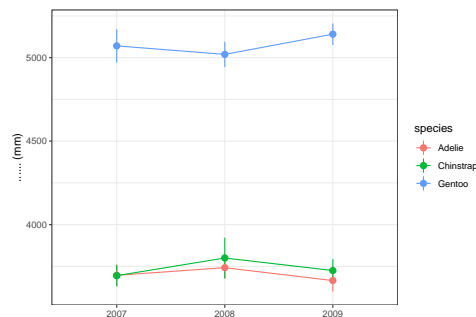
- 棒グラフを描画する関数は `geom_bar` 関数だが、データフレームから各値の平均を計算してグラフを作成するのはちょっと面倒くさい
- データの平均値を比較したい場合：`stat_summary` で、「平均値を」「棒グラフの形で」描画することをオーダーできる

```
df_clean <- df %>%
  filter(!is.na(sex)) # 性別不明を取り除く
ggplot2::ggplot(df_clean) +
  ggplot2::aes(x = sex, y = body_mass_g, fill = sex) +
  ggplot2::stat_summary(geom = "bar", fun = "mean") + # 棒グラフを、平均を示す形で
  ggplot2::stat_summary(geom = "pointrange", fun.data = "mean_se") +
  ggplot2::labs(x = "", y = "体重 (mm)")
```



## 折れ線グラフ：

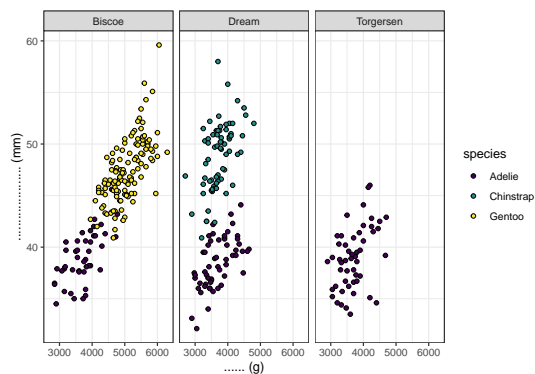
```
ggplot2::ggplot(df) +  
  ggplot2::aes(x = factor(year), y = body_mass_g, colour = species, group = species) +  
  ggplot2::stat_summary(geom = "point", fun = "mean") +  
  ggplot2::stat_summary(geom = "line", fun = "mean") +  
  ggplot2::stat_summary(geom = "pointrange", fun.data = "mean_se") +  
  ggplot2::theme_bw() +  
  ggplot2::labs(x = "", y = "体重 (mm)")
```



## グラフそのものを系統ごとに分ける

- 重ねると見づらいから複数に分けたい
- `facet_wrap` 関数を使うと、サンプルを分割して別々に表示してくれる
  - 「～変数名」で分割の基準となるサンプル、`nrow =` , `ncol =` で出力の並べ方を指定

```
ggplot2::ggplot(df) +  
  ggplot2::aes(x = body_mass_g, y = bill_length_mm, fill = species) +  
  ggplot2::geom_point(colour = "black", shape = "circle filled") +  
  ggplot2::scale_fill_viridis_d() + # 色分けに関する設定を行う関数：塗る色の指定  
  ggplot2::theme_bw() +  
  ggplot2::labs(x = "体重 (g)", y = "くちばしの長さ (mm)") +  
  ggplot2::facet_wrap(facets = ~ island, ncol = 3, nrow = 1) # 島ごとに分けてみる
```



## グラフを保存する

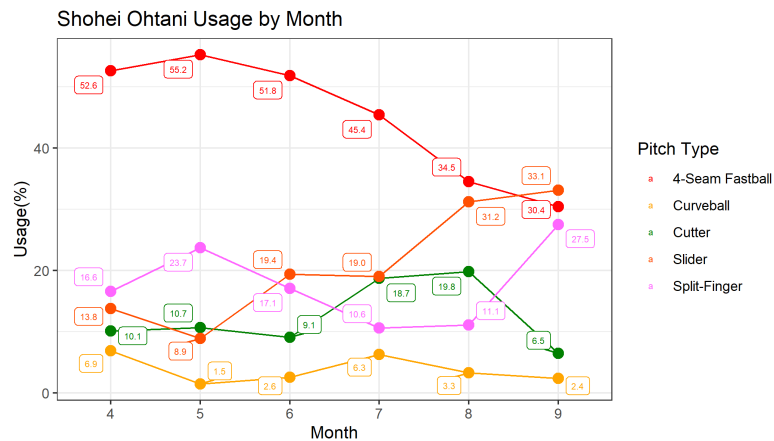
- 出来上がったグラフを保存する
- R Studio なら、右下の Plots に表示されたグラフをクリック操作で保存することもできる
- プロットを ggsave 関数に通して保存できる

```
pl <- ggplot2::ggplot(df) + # さっきのやつをオブジェクトに保存
  ggplot2::aes(x = body_mass_g, y = bill_length_mm, colour = species) +
  ggplot2::geom_point(colour = "black", shape = "circle filled") +
  ggplot2::scale_fill_viridis_d() +
  ggplot2::theme_bw() +
  ggplot2::labs(x = "体重 (g)", y = "くちばしの長さ (mm)") +
  ggplot2::facet_wrap(facets = ~ island, ncol = 3, nrow = 1)

ggsave(plot = pl, filename = "figs/plot_sample.png")
```

## その他の可視化

- 様々な可視化ができます
- ggplot something で作りたいものを調べれば出てくるので参考にして下さい
- 個人的に就職に一番役立ちそうなのはここだと思う
- 暇も潰せる



## 簡単な回帰分析

- OLS(最小二乗法) で回帰分析を行う
- 特に計量経済学において使用する回帰分析に必要なパッケージ：estimatr
  - lm 関数よりこちらを使用するようにする
  - 残差の不均一分散に対応した頑健な標準誤差を計算
- ここでは適切な分析手法やその内容について詳しく説明しないので、計量分析の教科書を1冊学習してみることをお勧めします
  - 高橋将宜, 『統計的因果推論の理論と実装』
  - Cunningham, ‘Causal Inference the Mixtape’
  - Huntington-Klein, ‘The Effect: An Introduction to Research Design and Causality’

## 回帰分析

- 観測されたデータ  $(y_i, X_i)$  for  $i = 1, \dots, N$  について、
 
$$y_i = \alpha + \beta X_i + u_i$$
- という線形関係の予測モデルを推定する
  - 独立変数  $X_i$  が変動した時に従属変数  $y_i$  がどれだけ変動するか： $y_i$  の実現値の分散を  $X_i$  のそれがどれだけ説明するかを推定
  - 関心があるのは独立変数の係数  $\beta$
- 例
  - 高等教育を無償化すると、将来の賃金が何 % 上昇するか？
  - 残業代アップと支払の厳格化は労働時間の削減に繋がるか？
- 多くの問題には「内生性」の問題が生じるため、シンプルな回帰分析では推定が難しいことも多い

## 因果推論

- 因果関係と相関関係の区別
  - 感染症の流行で太った人が多い：感染症は肥満を引き起こすウイルス？
  - 「人は話し方が 9 割」「人は聞き方が 9 割」：足したら 18 割だけど
  - 「大学で勉強したことは役に立たない」：って上司が言ってた
- 特に経済学や医療分野では「A が B の原因であること」が重要

## 回帰分析の例

- ペンギンのデータセットで分析
- ペンギンの嘴の長さを予測するモデルを作ろう
- 各個体  $i$  について

$$\text{billSize}_i = \alpha + \beta_1 \text{weight}_i + \beta_2 \text{Species}_i + u_i$$

- でかいペンギンはくちばしもでかいのでは、ペンギンの種類でも関係がありそう
- 説明変数に体重と種類の情報を入れる

## lm\_robust 関数を用いた推定と結果の確認

```
model <- df %>%
  estimatr::lm_robust(formula = bill_length_mm ~ body_mass_g + species, data = .)
summary(model)

##
## Call:
## estimatr::lm_robust(formula = bill_length_mm ~ body_mass_g +
##   species, data = .)
##
## Standard error type:  HC2
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|) CI Lower CI Upper DF
## (Intercept)  24.919471  1.0673395  23.347 1.790e-72 22.820006 27.018936 338
## body_mass_g    0.003748  0.0002887  12.983 1.502e-31  0.003181  0.004316 338
## speciesChinstrap 9.920884  0.3955951  25.078 3.850e-79  9.142746 10.699023 338
## speciesGentoo   3.557978  0.4560389   7.802 7.673e-14  2.660946  4.455009 338
##
## Multiple R-squared:  0.808 , Adjusted R-squared:  0.8063
```

## F-statistic: 476.3 on 3 and 338 DF, p-value: < 2.2e-16

## 係数の解釈

- ではくちばしの長さは体重によって決まるのか？：直感的に考えてそうではない
  - し、「体重がくちばしの長さを決める」ことが分かることはそこまで重要か？
- ペンギンの体格に環境変異が影響している可能性は？
  - 捕獲した島におけるエサの取りやすさやその他の環境要因が同じ種類のペンギンでも異なる淘汰を招いたのでは？
- くちばしの予測に島を表すダミー変数を加えてみよう

## 分析と結果の表示

- 最小二乗法と結果の表示
- 分析手法の設定
  - 固定効果モデル
  - 一般化線形モデル、最尤法
    - \* ロジスティック回帰
  - 操作変数法
  - DID (差の差分析)
  - RDD (回帰不連続デザイン)

## 計量分析を行う

- データの要約・可視化が終わったら、因果関係を識別するための分析に取り組む
- 最も一般的なのは回帰分析

$$y_i = \alpha + \beta X_i + u_i$$

- 説明変数の係数を推定し、説明変数の限界的な変化が被説明変数に対してどれだけ影響するかを推定したい
- ここでは、R で利用できるパッケージを使って分析の一例を示す

## AER パッケージの導入

- ペンギンのデータは簡潔で便利だが、経済学の分析に使うのはちょっと厳しそう
- 経済学向けのサンプルデータがあればいい
- あります：AER パッケージ
  - CRAN

- パッケージをライブラリで起動後、利用したいデータセットを `data()` 関数で起動、データフレームが自動で取り込まれる

```
install.packages("AER")
library(AER)
data("DataSetName") #好きなデータセットの名前、一覧は pdf を参照
```

## AER パッケージの利用

```
data("CollegeDistance")
print(head(CollegeDistance))
```

```
##   gender ethnicity score fcollege mcollege home urban unemp wage distance
## 1  male      other 39.15      yes      no  yes   yes   6.2 8.09      0.2
## 2 female      other 48.87      no      no  yes   yes   6.2 8.09      0.2
## 3  male      other 48.74      no      no  yes   yes   6.2 8.09      0.2
## 4  male      afam 40.40      no      no  yes   yes   6.2 8.09      0.2
## 5 female      other 40.48      no      no   no    yes   5.6 8.09      0.4
## 6  male      other 54.71      no      no  yes   yes   5.6 8.09      0.4
##   tuition education income region
## 1 0.88915         12   high  other
## 2 0.88915         12    low  other
## 3 0.88915         12    low  other
## 4 0.88915         12    low  other
## 5 0.88915         13    low  other
## 6 0.88915         12    low  other
```

## 分析に必要なパッケージ

- `estimatr` パッケージ：OLS、固定効果、操作変数法など、様々な計量経済分析を行うことができる
- `huxtable` パッケージ：結果の出力
  - `xlsx` や `pptx` 形式への出力も可能
- その他、結果を出力するためのパッケージが色々あるので好きなものを使う
  - `stargazer`
  - `modelsummary`：最近知ったので勉強中、皆さんも勉強して教えて下さい

## 回帰分析の結果の表示

- 回帰分析の結果を表に出力する
  - 複数の計量モデルや説明変数・固定効果の変更が推定結果に及ぼす形を比較できるような出力が

必要

- summary 関数の出力結果をスクショ・コピーしてそのままスライドや論文に貼る→罰金
  - 見映えが悪いし、コピーするよりは適切なコードを覚える方が労力が少なくて済む
  - 結果を Excel やパワーポイントに直接出力する
- huxtable, ftExtra, kableExtra パッケージが表出力全般に長けているのでこれらを利用する

## stargazer パッケージ

- html・latex 形式でデータを出力
- OLS は lm 関数でないと出力できない (不均一分散に頑健な標準誤差が一発で表示できない)
- 色々方法はあるが、パッケージの更新が止まっていて新しい出力形式が使えないことや、そもそも出力が Word などでの報告と相性が良くないなどの理由から非推奨
  - html 出力してスクショなどすれば一応綺麗な形で出力はできる

## huxtable パッケージ

- html をはじめとした様々な表出力に対応
- データフレームも出力できるので、記述統計量の出力にも使える

```
CollegeDistance <- CollegeDistance %>%
  mutate(
    univ = if_else(education >= 16, "Yes", "No"),
    high_income = if_else(income == "high", 1, 0),
  )
model1 <- CollegeDistance %>%
  estimatr::lm_robust(high_income ~ education, data = .)
model2 <- CollegeDistance %>%
  estimatr::lm_robust(high_income ~ education + score, data = .)
model3 <- CollegeDistance %>%
  estimatr::lm_robust(high_income ~ education + score + ethnicity, data = .)

table <- huxtable::huxreg(model1, model2, model3)
```

## huxtable の表示

### 出力の保存

- huxtable, huxreg で作った表は、quick\_(ファイル形式) 関数で保存することができる
- ファイルを保存したら、



	(1)	(2)	(3)
(Intercept)	-0.478 *** (0.050)	-0.578 *** (0.052)	-0.422 *** (0.055)
education	0.055 *** (0.004)	0.044 *** (0.004)	0.047 *** (0.004)
score		0.005 *** (0.001)	0.002 * (0.001)
ethnicityafam			-0.118 *** (0.018)
ethnicityhispanic			-0.155 *** (0.016)
N	4739	4739	4739
R2	0.048	0.055	0.075

\*\*\* p < 0.001; \*\* p < 0.01; \* p < 0.05.

## modelsummary パッケージ (要出典)

- べんきょうちゅうです、適宜追加します
- このサイトにざっと使い方が書いてあります
- 同じく、回帰分析の結果を様々なデータ形式、あるいは他のパッケージで可視化に利用するデータ型に変換して出力できる
  - 新たな行の付け足しなども比較的容易にできる

## modelsummary

```
table_list <- list("(1)" = model1, "(2)" = model2, "(3)" = model3) # リスト形式で回帰の結果を保存

modelsummary::msummary(table_list) # msummary 関数で一撃
```

	(1)	(2)	(3)
(Intercept)	−0.478 (0.050)	−0.578 (0.052)	−0.422 (0.055)
education	0.055 (0.004)	0.044 (0.004)	0.047 (0.004)
score		0.005 (0.001)	0.002 (0.001)
ethnicityafam			−0.118 (0.018)
ethnicityhispanic			−0.155 (0.016)
Num.Obs.	4739	4739	4739
R2	0.048	0.055	0.075
R2 Adj.	0.048	0.055	0.074
Std.Errors	HC2	HC2	HC2

## モデル選択

- OLS (一般的な重回帰分析) で因果推論を行うには、試したい処置がランダムに割り振られている、という仮定が必要
  - 実験室実験：コントロール群と介入群をランダムに割り振ることが可能
  - フィールド実験：一部抜け漏れや逸脱行動 (処置を受ける群に割り当てられたが拒否することができするなど) がある
  - その他、システム上ランダムに割り振られる違いを利用して比較を行う場合もある：MLB の審判員は、特定のチームや選手に偏ることなく (あるいは無関係に) ほぼランダムな形で割り振られる
- そうではない場合、因果関係の存在を示すための適切な実証方法を利用する必要がある

## 適切なモデルの選択

- 固定効果モデル：パネルデータ分析 (同一人物の複数期間にわたるデータが取得されている場合)
- 一般化線形モデル、最尤法 (被説明変数が制限されている場合：制限従属変数など)
- DID (処置群と統制群の前後比較を行う)
- 操作変数法：説明変数に内生性がある場合
- 回帰不連続デザイン (RDD)：閾値の前後で不連続に処置が変わるようなシステムを利用する

## 固定効果モデル

- パネルデータ：個人  $i = 1, \dots, I$  について、複数期間  $t = 1, \dots, T$  にわたる観測データが得られる場合

$$y_{it} = X_{it}\beta + \delta_i + \gamma_t + u_{it}$$

- 個々人・時間の効果を取り除いた上で、関心のある変数  $X_{it}$  の変化に対する反応を見る
  - 本質的にはグループごとのダミー変数を入れるのと同じ
  - 個々の係数には注目しない場合が多いので、実際に表示する結果の表には「個人レベルの固定効果を取り除きました」などと載せておけば OK
- `lm_robust` 関数を利用して推定可能
  - `fixed_effects` オプションに固定効果として制御する

## 一般化線形モデル

- 観測されたデータに対して、最も当てはまりが良い (尤もらしい) パラメータを選択する
- OLS と同様の回帰分析においても利用可能であるが、ここでは被説明変数が 0,1 の二値変数である場合に利用する「ロジスティック回帰」を紹介
  - その他、被説明変数が幸福度など段階のある離散の変数として観測される場合に使う「順序ロジット・プロビットモデル」、一定の値を超えないと値そのものが観測されない「トービットモデル」など様々なモデルが利用できる
  - OLS よりも残差の分布などについてやや強い仮定を用いる

## ロジスティック回帰

- データから観察できない被説明変数  $y_i^*$  が存在しており、これが説明変数  $x_i$  を含むモデル  $y_i^* = x_i\beta + e_i$  で定まる
- $y_i^*$  が特定の値 (便宜上ゼロ) を超えた時に従属変数  $y_i = 1$ , そうでないときに  $y_i = 0$  が観測される
- $x_i$  が決まった時、 $y_i = 1$  が観測される条件付確率は、 $e_i$  の分布関数  $F(\cdot)$  によって決まる
  - この分布関数にロジスティック分布を仮定するのがロジスティック回帰、正規分布を仮定するのがプロビット回帰
  - ロジスティック分布の方が計算上扱いやすいのでこちらが使われてきたが、コンピュータの発達により、近年はどちらも計算速度が変わらないと言われる

$$F(-x_i\beta) = \frac{\exp(-x_i\beta)}{1 + \exp(-x_i\beta)}$$

- データと  $e_i$  に関する分布形の仮定を元に、実際の選択行動を最もよく説明するようなパラメータ  $\beta$  を探す：最尤法による推定
- `glm` 関数で推定可能

## 限界効果

- 先にある通り、ロジスティック回帰、プロビット回帰で推定された係数  $\beta$  は、観測される従属変数  $y_i$  に対する直接の効果ではない
  - $x_i$  の値が変動した時に、被説明変数がどれくらい変化するかを表示するのが一般的

## DID

- difference-in-difference (差の差) 分析
- 処置群が介入を受けた前後の統制群・処置群両方のデータを取得することで、両者に共通の時間トレンドを考慮した介入効果の推定を行う
  - 処置群は検証したい介入の影響に加えて、時間の変化や全国一律の制度変更など、様々な効果を受ける：こうした「それ以外の影響」について、統制群と処置群とが受けた影響が同一であるなら、両者の差分を出すことによって介入の純粋な効果を推定できるのでは、というのがアイデア
  - 一般的には3期間以上の長期間、3単位以上の個人などにわたる観測データを利用することが多い
  - パラレルトレンドの仮定 (経時的な影響は両群に共通) など、やや強い仮定が必要

$$y_{it} = \beta \text{Treated}_i \times \text{After}_t + \gamma X_i + \alpha_i + \delta_t + u_i$$

## 操作変数法

- 一般的な OLS が孕む最大の問題である内生性の問題に対処
- データからは観測できない何らかの要因が存在し、それが被説明変数に対して影響を及ぼしている場合
- 例：教育年数と賃金との関係

$$\log w_i = \alpha + \beta x_i + u_i$$

- 教育年数が長くなる：進学すれば賃金が上昇する？
  - データから観測できない変数：個人の能力  $A_i$
  - 教育を長く受けることが賃金を上げるのではなく、そもそも稼得能力の高い、才能のある個人が進学を選択しているだけ？
  - 例えば「高等教育を無償化してこれまで進学していなかった人に教育の機会を与える」という政策の是非を問う上で、こうした問題は無視できない

## 操作変数法の発想

- Angrist and Kruger の発想： $x_i$  には関係するが、 $w_i$  には直接関係しないような変数を探してきて、これをコントロールしよう

- このような変数  $z_i$  を操作変数 (instrumental variable) と呼ぶ
- 二段階最小二乗法による推定で、先の内生性の問題を解決できる
  1. 操作変数  $z_i$  で  $x_i$  を回帰し、予測値  $\hat{x}_i$  を得る
  2.  $\hat{x}_i$  と  $z_i$  で  $y_i$  を回帰する
- 直感的には、「 $x_i$  が大きくなりやすい人」を探してその影響をコントロールすること
  - 教育年数→賃金：最寄りの大学までの距離
  - ソーシャルキャピタル→幸福度：幼少期に近くに神社などがあったか否か
  - 価格→需要量：その財のコストにかかわる変化
- 実装
  - AER パッケージにある `ivreg` 関数は操作変数法による回帰を行う方法
  - estimatr パッケージの `iv_robust` 関数は `lm_robust` と同様に使える

## 回帰不連続デザイン

- 操作変数と同様、内生性のある問題を部分的に解決するアプローチの一つ
- ある処置を受けられるかどうかを決めるシステム (assignment rule) の不連続な切れ目に注目し、その切れ目の前後を比較することで処置効果を推定する
  - 大学教育の効果：入試の合格最低点周りの集団は、合格した人も不合格だった人もほぼ同様な能力を持った人と言える
  - 医療費の自己負担割合：特定の年齢に到達すると負担割合が不連続に下がる
  - プロ野球選手の身長
- 切れ目の「前後」とはどこからどこまでを指すのか？：対象とする観測の設定など、独特な推定・分析が必要になる
- rdrobust パッケージなど、推定を行うにあたって必要な分析を行うパッケージが公開されている

## テーマ探しと並行してぜひ取り組んでほしい書籍

- R の基本操作
  - 松村 et al., 『R ユーザのための RStudio [実践] 入門』
  - igjit, atusy, hanaori, 『R が生産性を高める』
- 因果推論
  - 浅野正彦, 矢内勇生, 『R による計量政治学』
  - 高橋将宜, 『統計的因果推論の理論と実装』
  - Cunningham, ‘Causal Inference the Mixtape’
  - Huntington-Klein, ‘The Effect: An Introduction to Research Design and Causality’

## RMarkdown を用いたレポートの作成

- Markdown 形式のドキュメント

- 数式フォントの利用
- コードブロックの作成
- html ドキュメントの作成
- Word ファイルへの変換
- Powerpoint ファイルへの変換
- R Markdown を併用して論文作成・スライド作成の手間を省く

## Markdown とは

- 主に html(ウェブサイトなどで利用される形式) を手軽に出力するために考案された言語
- R の結果出力などに特化した形式：R Markdown
  - ここでは R Markdown について扱う
  - html だけでなく、Word や PowerPoint など、使い慣れた形式にも変換可能
- 分析結果をいちいちスクリーンショットしたり、体裁を整えるために出力をやり直したりする必要がなくなる
  - 全てを R Markdown で完結させる必要はないので、例えば図や分析結果の出力をするための Word ファイルを作り、できたものをコピペするなどして使えば微調整も容易

## rmarkdown ファイルをいじってみる

- リンクからマークダウンファイルをダウンロード、開く
- knitr パッケージがあれば動かせるはずです
  - knitr: Ctrl + Shift + K

## おまけ：バージョン管理

- 論文執筆・輪読の資料報告は班単位で行うので、スライドや分析結果を複数人で作成・共有する必要がある
- Dropbox, Github, Google Drive などファイルごと共有しておく、スライドをくっつけたり各自が修正したものをすり合わせる作業が削減できる、たぶん
- 覚えておいて損はないのでまあ興味があれば