



R に入門される方向けのセッション

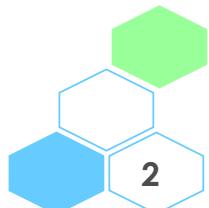
【事前のご準備・自習内容】

舟尾 暢男（武田薬品工業）



本日のメニュー

- Windows 版 R / RStudio のセットアップ方法 ← 当日までに自習
- R の基礎知識 ← 当日までに自習
- パッケージと作業ディレクトリ ← 当日までに自習
- 変数とベクトル
- ベクトルの型
- 関数とプログラミング
- データ加工とパイプ演算子
- グラフ作成(ggplot2)
- その他





The Comprehensive R Archive Network (CRAN) にアクセス

- CRAN
<https://cran.r-project.org/bin/windows/base/>
- 「Download R-X.X.X for Windows」をクリックし最新版の R (例えば、R-4.5.1-win.exe) をダウンロード

R-4.5.1 for Windows

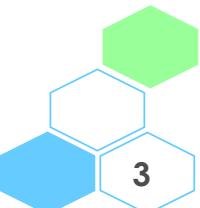
[Download R-4.5.1 for Windows](#) (86 megabytes, 64 bit)

[README on the Windows binary distribution](#)

[New features in this version](#)

This build requires UCRT, which is part of Windows since Windows 10 and Windows Server 2016. On older systems, UCRT has to be installed manually from [here](#).

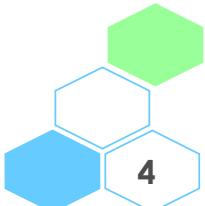
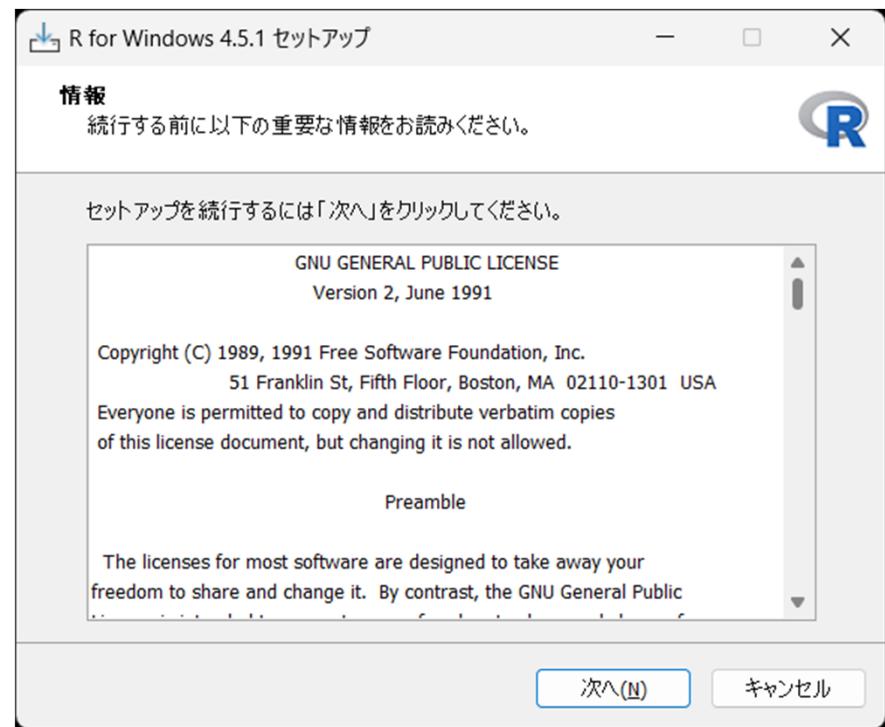
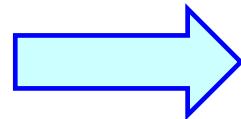
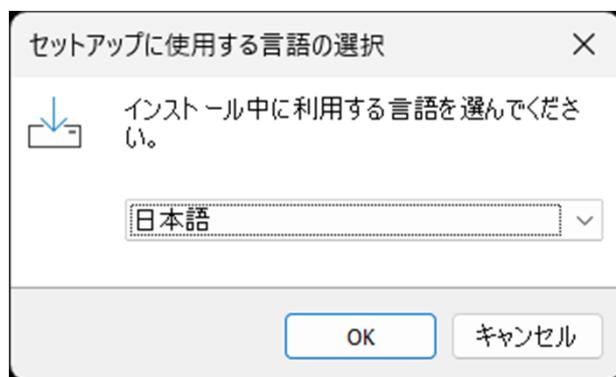
If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the [md5sum](#) of the .exe to the [fingerprint](#) on the master server.





R のインストール

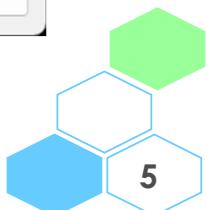
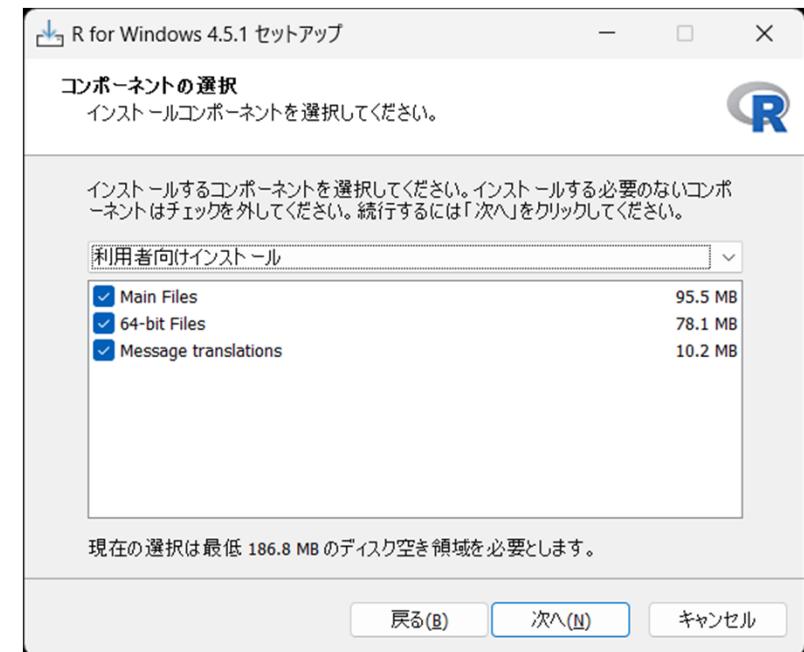
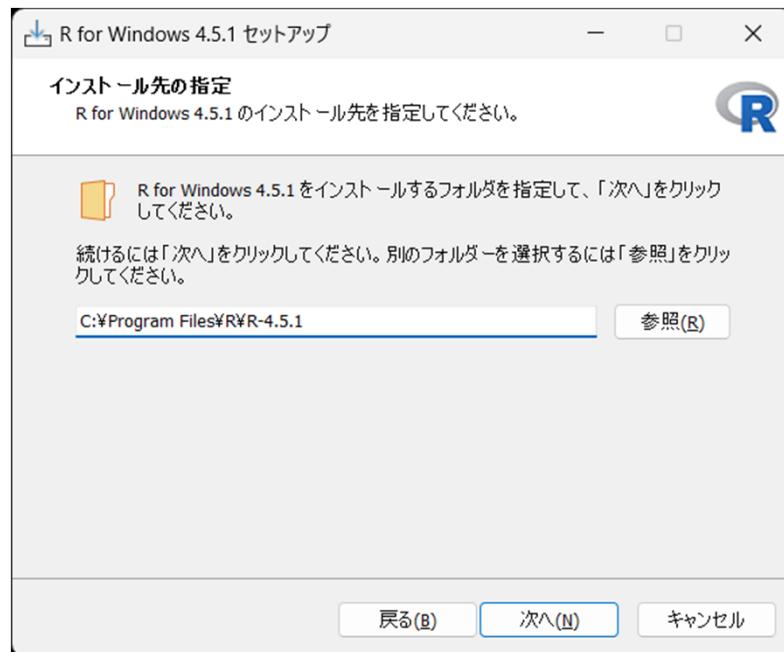
- ダウンロードが完了したら「R-4.5.1-win.exe」をダブルクリック
- セキュリティ関係の警告が出る場合があるが「実行」や「はい」をクリックすると言語選択の画面が表示されるので「日本語」を選択して [OK] をクリック
- その後、「次へ」をクリック



R のインストール



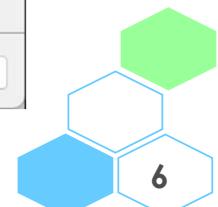
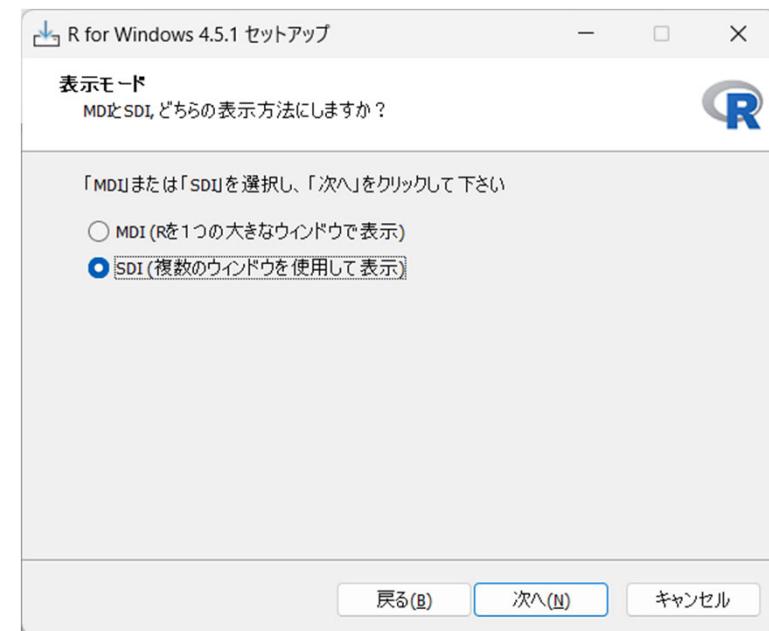
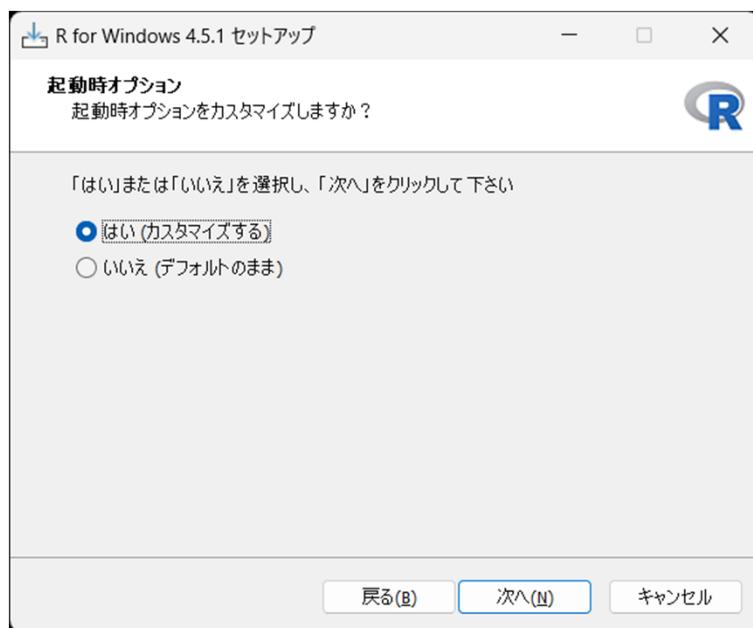
- 左下の画面でインストール先を指定することが出来るが、通常はそのまま「次へ」
- 右下の画面になるので、全てチェックを入れた上で「次へ」をクリック





R のインストール

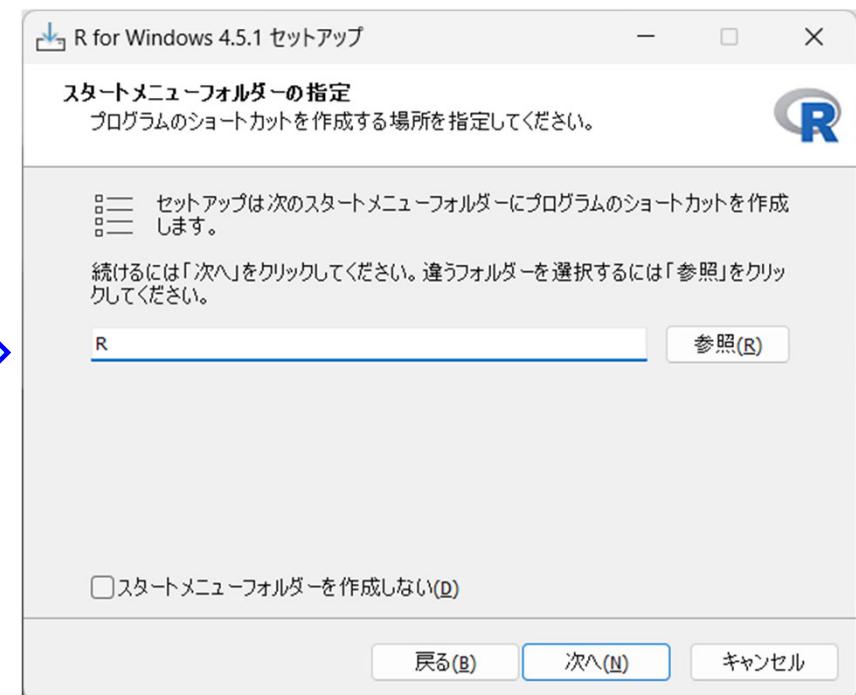
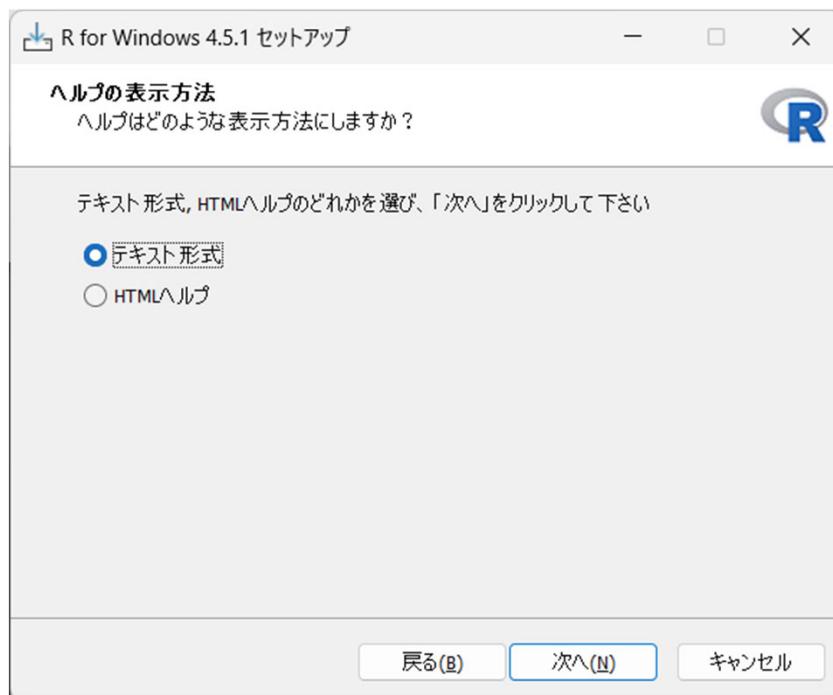
- 起動時オプションを設定するかどうかを選択する画面が表示される
- 設定しない場合は「いいえ(デフォルトのまま)」を選択するが、ここでは「はい(カスタマイズする)」を選択して「次へ」をクリック
- 「表示オプション」の選択画面で、「SDI(複数のウインドウ….)」を選択して「次へ」





R のインストール

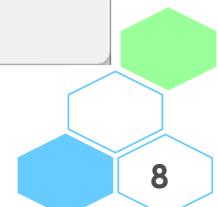
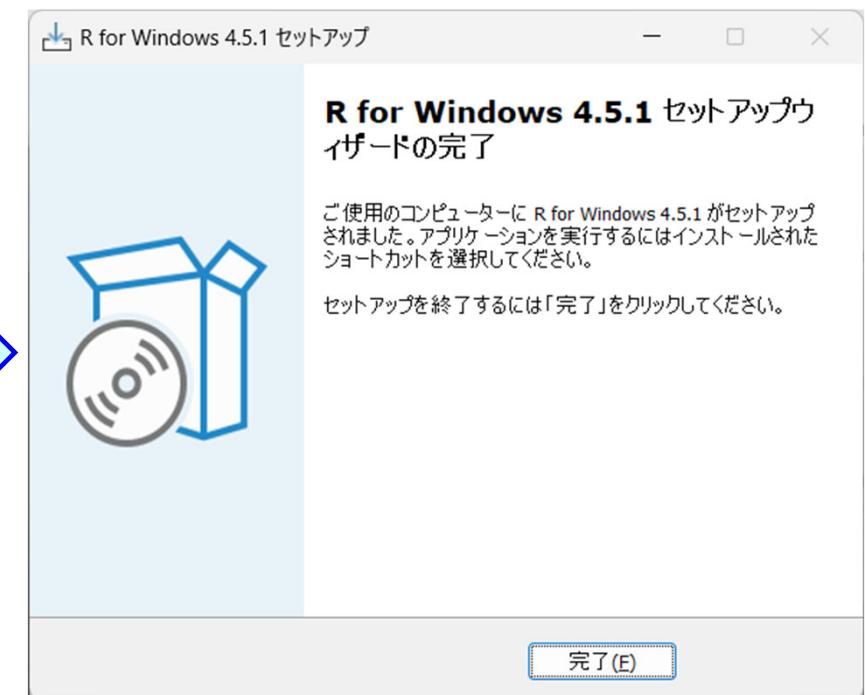
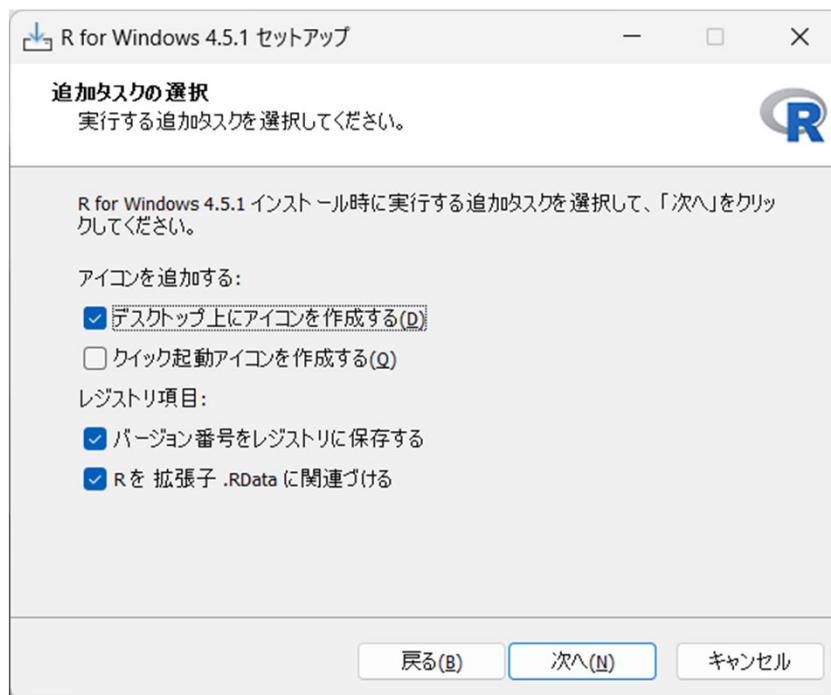
- 「ヘルプの表示方法」の選択画面で、いずれかを選択して「次へ」をクリック
- 「スタートメニューフォルダー」の指定画面では、通常そのまま「次へ」をクリック





R のインストール

- 追加タスクをおこなう画面が表示されるが、通常はそのまま「次へ」をクリック
- インストールが開始され、しばらくするとインストールが終了するので「完了」をクリック





Posit にアクセス

- Posit
<https://posit.co/download/rstudio-desktop/>
- サイトの上段にある「2: Install RStudio」で「Download RStudio Desktop for windows」となっていれば、そのアイコンをクリック
- または、サイトの中段に各 OS の RStudio が一覧となっているので、インストールファイル(例えば、RStudio-2025.09.0-387.exe)をダウンロード

Smarter science, better tools: Explore new product capabilities from Posit in our Fall 2025 Product Announcement [READ MORE](#) [X](#)

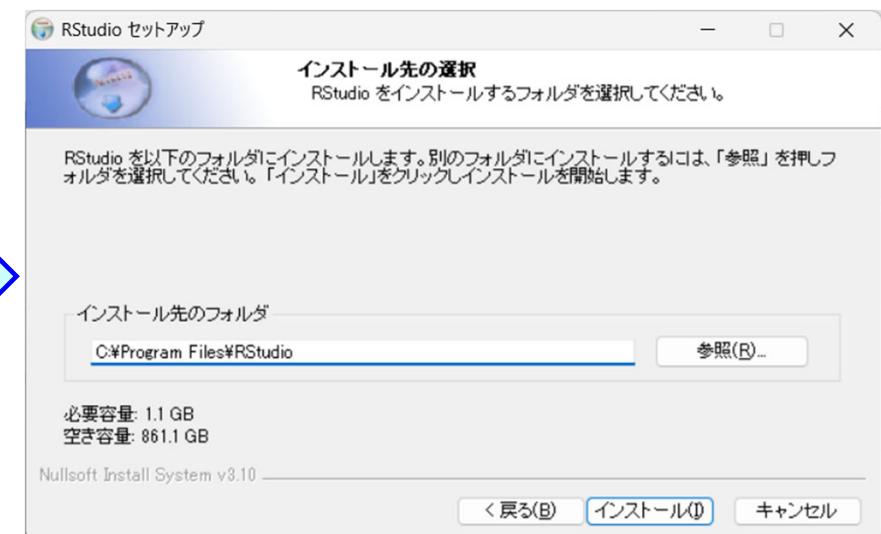
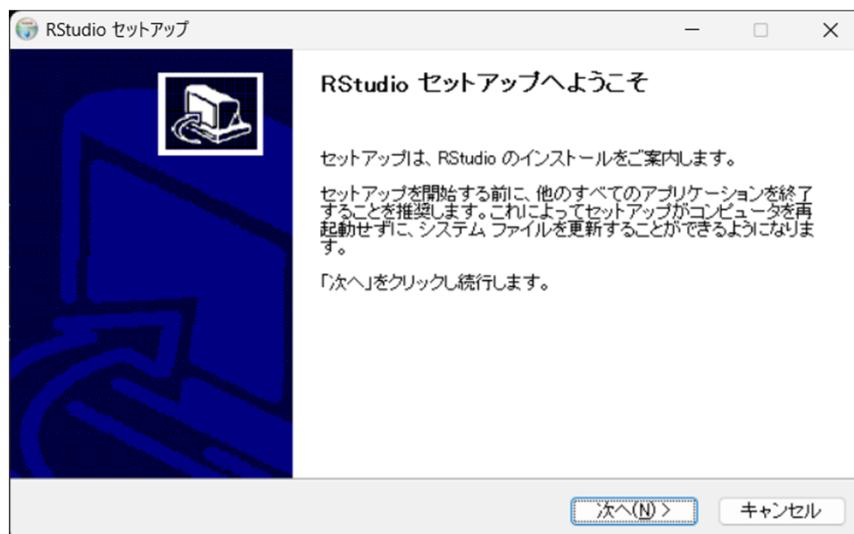
posit [PRODUCTS](#) [FREE & OPEN SOURCE](#) [USE CASES](#) [PARTNERS](#) [LEARN & SUPPORT](#) [ABOUT](#) [Q](#)

OS	File	Size	SHA256
Windows 10/11	RSTUDIO-2025.09.0-387.EXE	287.97 MB	8CE88C63
macOS 13+	RSTUDIO-2025.09.0-387.DMG	634.56 MB	8568D611
Ubuntu 22/Debian 12	RSTUDIO-2025.09.0-387-AMD64.DEB	216.76 MB	851FB642
Ubuntu 24	RSTUDIO-2025.09.0-387-AMD64.DEB	216.76 MB	851FB642



RStudio のインストール

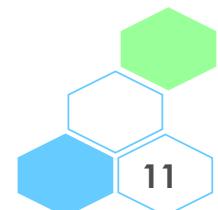
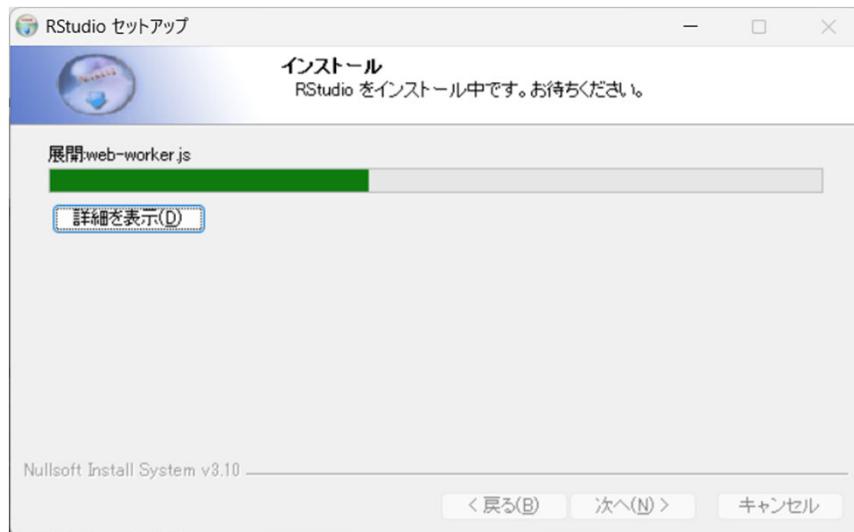
- ダウンロードが完了したら「RStudio-2025.09.0-387.exe」をダブルクリック
- セキュリティ関係の警告が出る場合があるが「実行」や「はい」をクリックするとセットアップ画面が表示されるので「次へ」をクリック
- 右下の画面でインストール先を指定することができるが、通常はそのまま「次へ」





RStudio のインストール

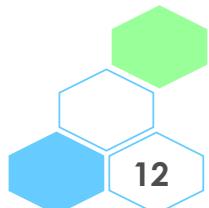
- インストールが開始され、しばらくするとインストールが終了するので「完了」をクリック ⇒ 以上で R / RStudio のセットアップが完了





本日のメニュー

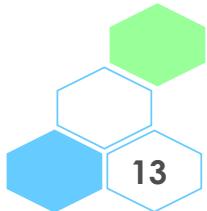
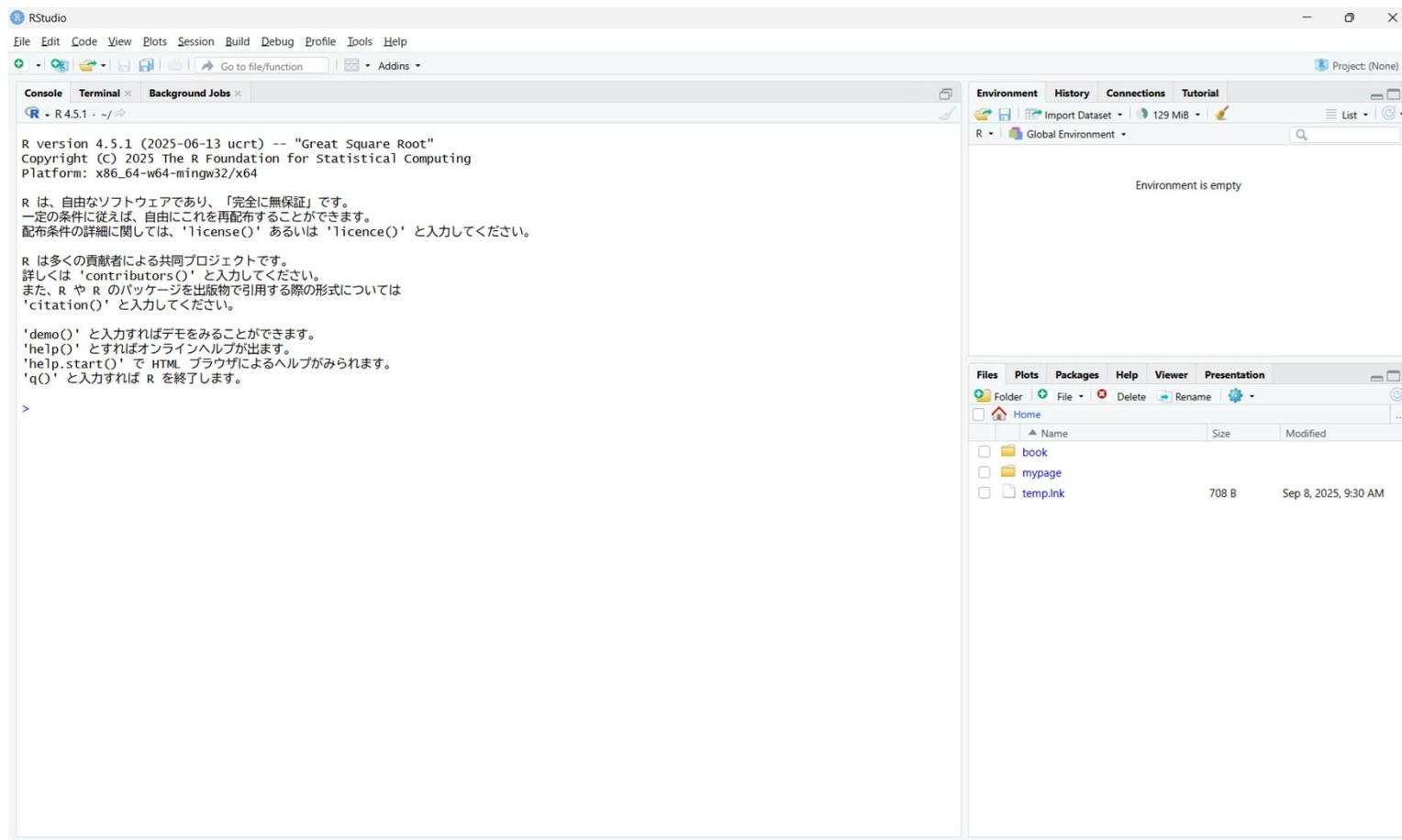
- Windows 版 R / RStudio のセットアップ方法 ← 当日までに自習
- R の基礎知識 ← 当日までに自習
- パッケージと作業ディレクトリ ← 当日までに自習
- 変数とベクトル
- ベクトルの型
- 関数とプログラミング
- データ加工とパイプ演算子
- グラフ作成(ggplot2)
- その他





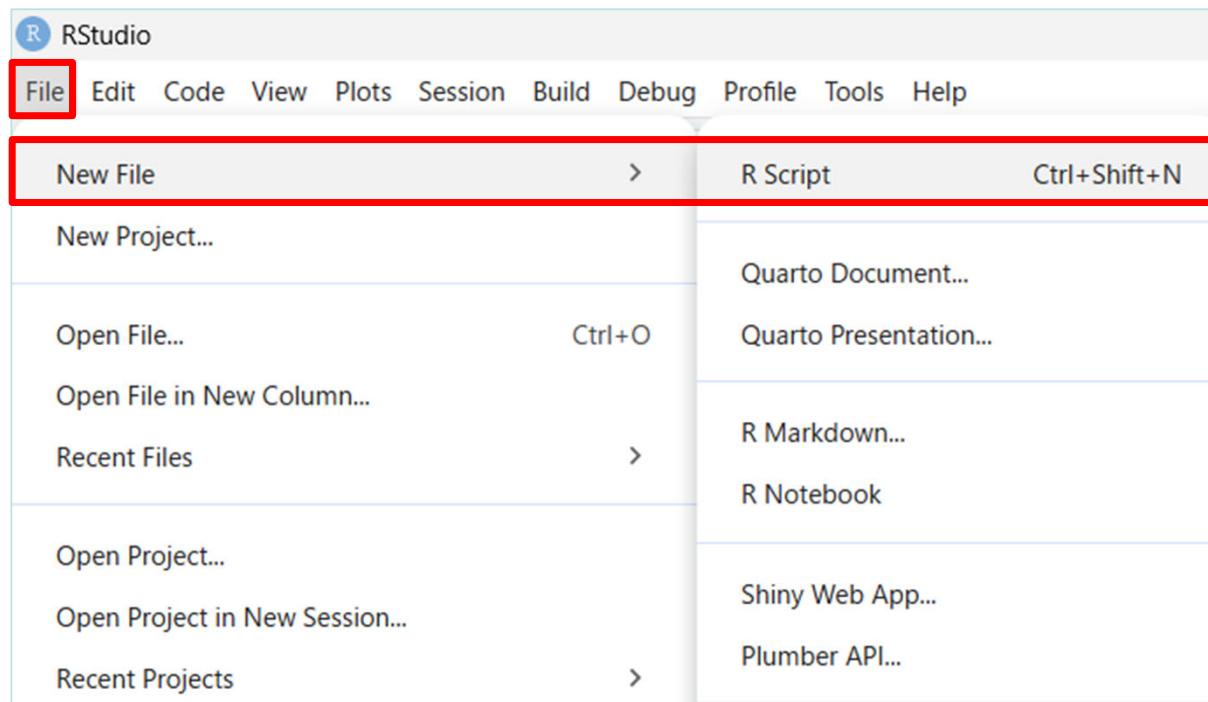
RStudio の起動

- デスクトップに作成されたショートカット、またはスタートアップにあるショートカットをクリックして RStudio を起動



RStudio の起動

- メニューバーの [File] → [New File] → [R Script] を選択、これで R でプログラミングを行う環境が整う



RStudio の起動



プログラムを書く場所
(ソース画面)

実行結果が表示される場所
(コンソール画面)

作成された変数リスト
実行したプログラムの履歴一覧

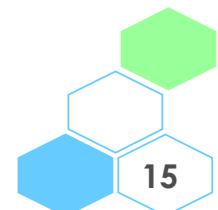
グラフの表示場所
ファイルやパッケージに
関する情報の表示場所

ウインドウ間のしきりを
マウスでドラッグすると
各ウインドウの大きさを
変更することが出来る

The screenshot shows the RStudio interface with several panes:

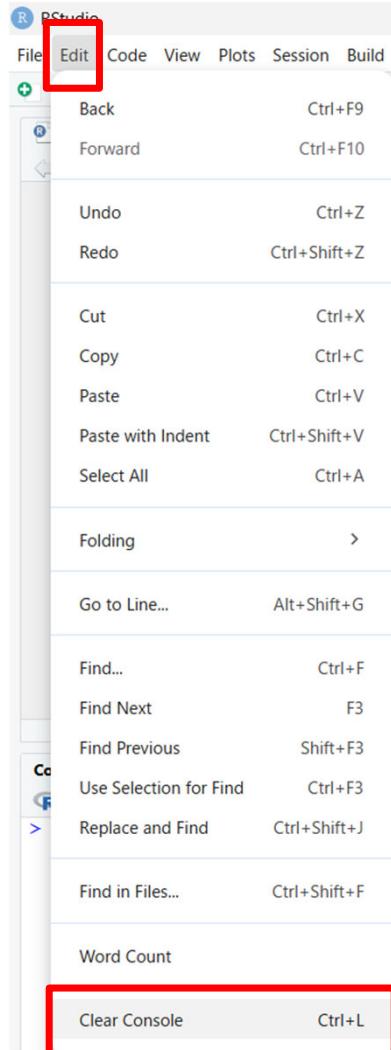
- Source pane:** Untitled1.R, showing a single line of code "1".
- Console pane:** Displays R version 4.5.1 startup message and help information.
- Environment pane:** Shows "Environment is empty".
- Files pane:** Shows a file tree with .Rhistory, book, mypage, and temp.lnk files.
- Plots pane:** Currently empty.

A red double-headed arrow is positioned between the Environment and Files panes, indicating they can be resized.

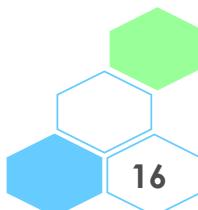




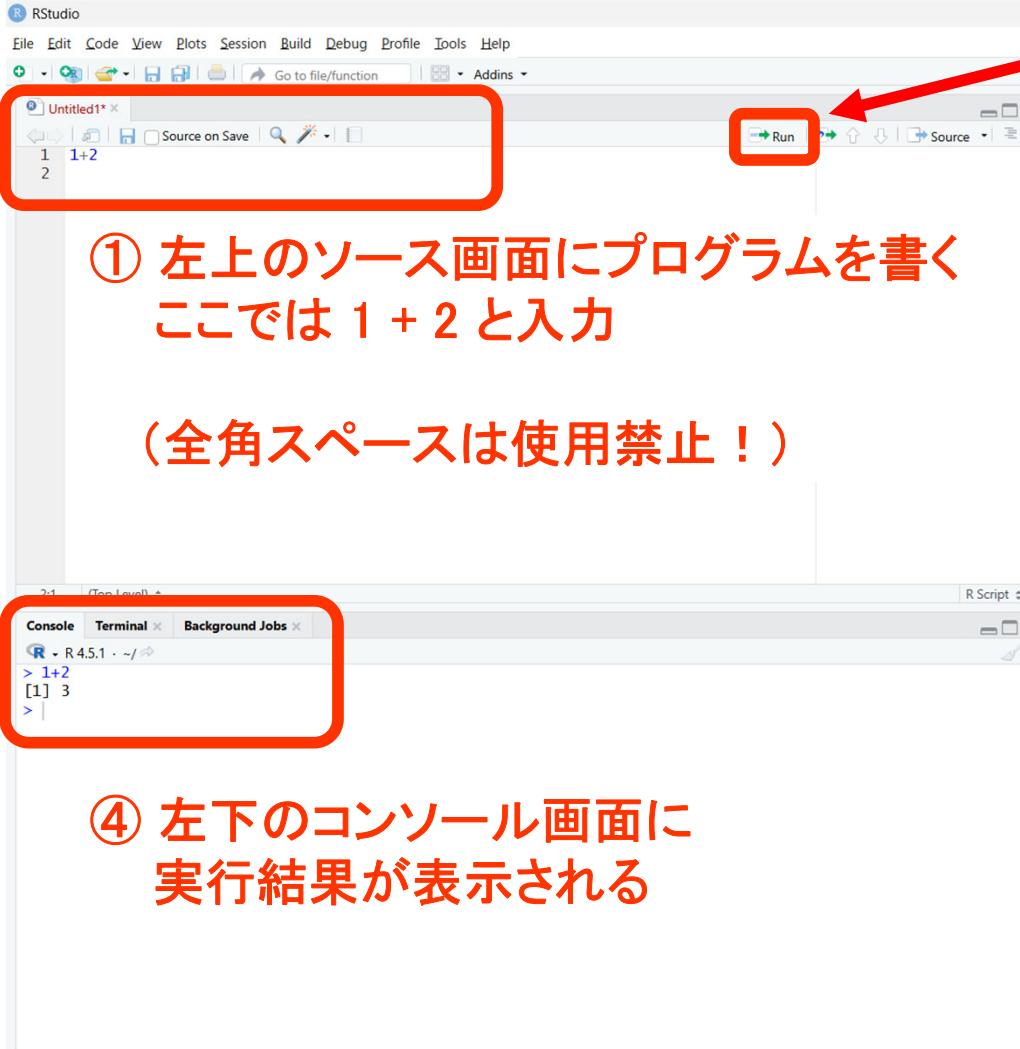
RStudio の起動



- 左上の「プログラムを書く場所(ソース画面)」にプログラムを書いて実行、コンソールに表示された結果を確認する流れ
 - 左下のコンソール画面にプログラムを直接入力することも出来る
- 長い時間プログラミングを行うと、コンソール画面が文字でいっぱいになる ⇒ コンソール画面をクリックした後にメニューバーの [Edit] → [Clear Console] を選択するか、コンソール画面をクリックした後に [Ctrl] + [L] キーでコンソール画面を奇麗にすることが出来る



基本的な計算



The screenshot shows the RStudio interface. The top panel displays the code editor with the following content:

```
Untitled1* 
1 1+2
2
```

The bottom panel shows the console output:

```
2:1 [Top Level] > R Script
Console Terminal Background Jobs
R 4.5.1 . ~/ 
> 1+2
[1] 3
>
```

Red boxes highlight the code editor and the console area. A red arrow points from the code editor to the 'Run' button in the toolbar.

- ① 左上のソース画面にプログラムを書く
ここでは $1 + 2$ と入力
(全角スペースは使用禁止！)

- ② [Run] をクリック、又は [Ctrl] + [Enter] キーでも実行出来る

- ③ 複数行のプログラムを一度に実行する場合は、
プログラムの該当部分をマウスなどで選択した後
[Run] をクリックするか [Ctrl] + [Enter] キーを
押してプログラムを実行する

- ④ 左下のコンソール画面に
実行結果が表示される

基本的な計算

```
> 1 + 2  
[1] 3  
>
```

- 「 $1 + 2$ 」の前に `>` というものが前についているが、これは実行したプログラム(の 1 行目)であることを表す
- 計算結果の「 3 」の前に `[1]` というものが前についているが、これは「結果の数字がひとつである」ことを表す ⇒ 「ベクトル」の説明の際に追加で解説予定
- `>` の記号が再び現れたが、これは「 $1 + 2$ の計算が終了したので、次の計算式を入力して」と R が要求していることを意味する ⇒ この後、新たに計算式を入力してプログラムを実行すれば R は再び計算処理をしてくれる
- 本資料では、例えば以下の様にプログラムと実行結果を一度に示すことがあるが、`>` や実行結果の `[1] 3`、# のコメント は入力する必要はない(「 $1 + 2$ 」だけを入力して実行)

```
> (1 + 2 - 3^4) * 5 / 6 # これはコメント  
[1] -65
```

演算子と数学関数

- 計算を行うための演算子として以下が用意されている

演算子	+	-	*	/	^
意味	加算	減算	乗算	除算	累乗

- $\sqrt{2}$ を計算する場合は関数 `sqrt()` を使用する

```
> sqrt(2)
```

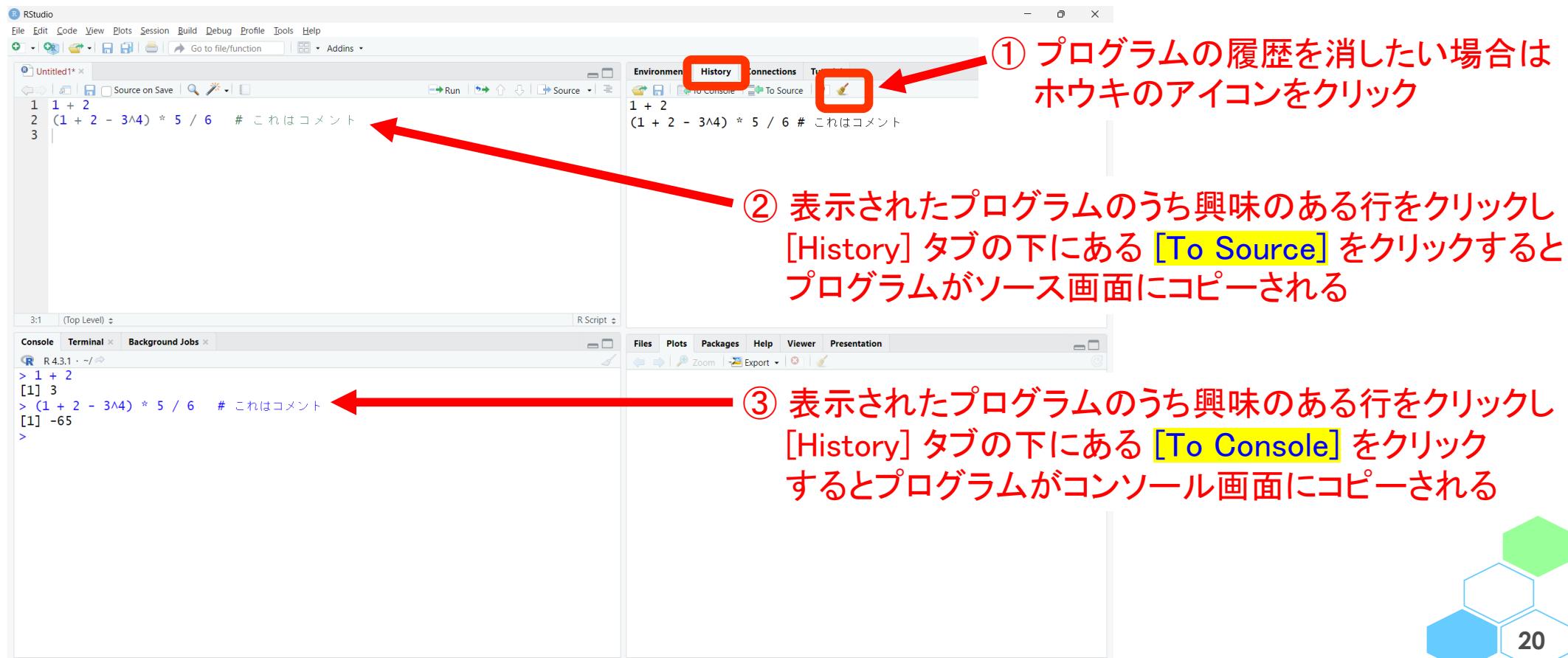
```
[1] 1.414214
```

- Rには以下のような数学関数が用意されている

関 数	<code>sin(x)</code>	<code>cos(x)</code>	<code>tan(x)</code>	<code>log(x)</code>	<code>log10(x)</code>
意味	<code>sinx</code>	<code>cosx</code>	<code>tanx</code>	<code>log_ex</code>	<code>log₁₀x</code>
関 数	<code>sinh(x)</code>	<code>cosh(x)</code>	<code>tanh(x)</code>	<code>exp(x)</code>	<code>sqrt(x)</code>
意味	<code>sinhx</code>	<code>coshx</code>	<code>tanhx</code>	e^x	ルート x
関 数	<code>abs(x)</code>	<code>trunc(x)</code>	<code>round(x)</code>	<code>floor(x)</code>	<code>ceiling(x)</code>
意味	絶対値	整数部分	丸め	切り下げ	切り上げ

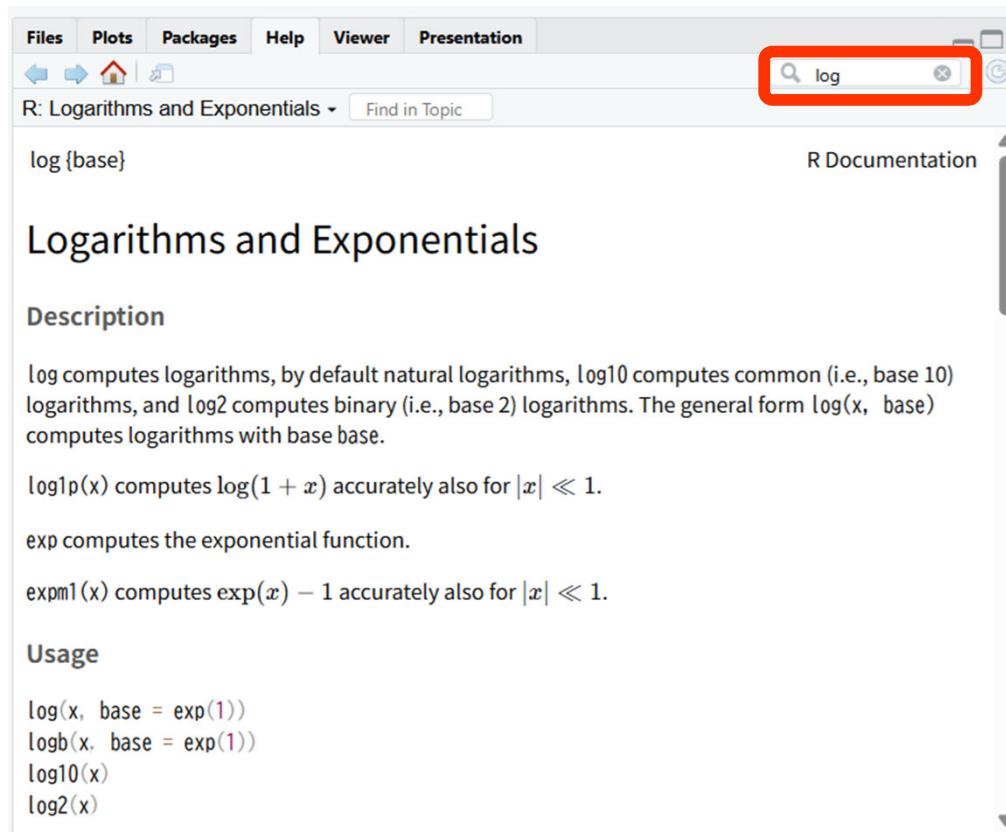
履歴

- 過去にどんなプログラムを実行したか確認したくなる場合がある
- 右上の画面の [History] タブをクリックすると、プログラムの履歴が表示される

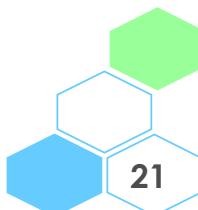


ヘルプを見る

- はじめのうちは関数の使い方がよく分からない場合がある ⇒ 例えば関数 `log()`
- 右下の画面の [Help] タブをクリックし、その下の虫めがねのアイコンがあるテキスト入力画面に関数名を入力した後 [Enter] キーを押すことでヘルプが表示される



The screenshot shows the R Help documentation interface. The search bar at the top right contains the text "log". Below the search bar, the title "log {base}" is displayed. The main content area is titled "Logarithms and Exponentials". Under "Description", it says: "log computes logarithms, by default natural logarithms, log10 computes common (i.e., base 10) logarithms, and log2 computes binary (i.e., base 2) logarithms. The general form log(x, base) computes logarithms with base base." It also mentions "log1p(x)" and "exp". Under "Usage", it lists the functions: "log(x, base = exp(1))", "logb(x, base = exp(1))", "log10(x)", and "log2(x)".



例: 関数 `log()` のヘルプ

log {base}

R Documentation

① Description

`log` computes logarithms, by default natural logarithms, `log10` computes common (i.e., base 10) logarithms, and `log2` computes binary (i.e., base 2) logarithms. The general form `log(x, base)` computes logarithms with base `base`.

`log1p(x)` computes $\log(1+x)$ accurately also for $|x| \ll 1$.

`exp` computes the exponential function.

`expm1(x)` computes $\exp(x) - 1$ accurately also for $|x| \ll 1$.

② Usage

```
log(x, base = exp(1))
logb(x, base = exp(1))
log10(x)
log2(x)

log1p(x)

exp(x)
expm1(x)
```

③ Arguments

`x` a numeric or complex vector.

`base` a positive or complex number: the base with respect to which logarithms are computed. Defaults to `e=exp(1)`.

④ Details

All except `logb` are generic functions: methods can be defined for them individually or via the [Math](#) group generic.

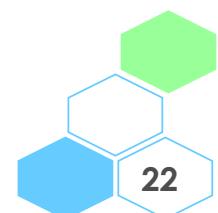
`log10` and `log2` are only convenience wrappers, but logs to bases 10 and 2 (whether computed via `log` or the wrappers) will be computed

① 説明

② 関数の雛形

③ 引数の説明

④ 補足説明



例：関数 `log()` のヘルプ

⑤

Value

A vector of the same length as `x` containing the transformed values. `log(0)` gives `-Inf`, and `log(x)` for negative values of `x` is `NaN`. `exp(-Inf)` is 0.

For complex inputs to the log functions, the value is a complex number with imaginary part in the range $[-\pi, \pi]$: which end of the range is used might be platform-specific.

⑥

S4 methods

`exp`, `expm1`, `log`, `log10`, `log2` and `log1p` are S4 generic and are members of the [Math](#) group generic.

Note that this means that the S4 generic for `log` has a signature with only one argument, `x`, but that `base` can be passed to methods (but will not be used for method selection). On the other hand, if you only set a method for the `Math` group generic then `base` argument of `log` will be ignored for your class.

⑦

Source

`log1p` and `expm1` may be taken from the operating system, but if not available there then they are based on the Fortran subroutine `dlnrel` by W. Fullerton of Los Alamos Scientific Laboratory (see <http://www.netlib.org/slatec/fnlib/dlnrel.f> and (for small `x`) a single Newton step for the solution of `log1p(y) = x` respectively).

⑧

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole. (for `log`, `log10` and `exp`.)

Chambers, J. M. (1998) *Programming with Data. A Guide to the S Language*. Springer. (for `logb`.)

⑨

See Also

[Trig](#), [sqrt](#), [Arithmetic](#).

⑩

Examples

```
log(exp(3))
log10(1e7) # = 7

x <- 10^{-(1+2*1:9)}
cbind(x, log(1+x), log1p(x), exp(x)-1, expm1(x))
```

⑤ 引数・返り値の補足

⑥ 無視

⑦ 注意書き

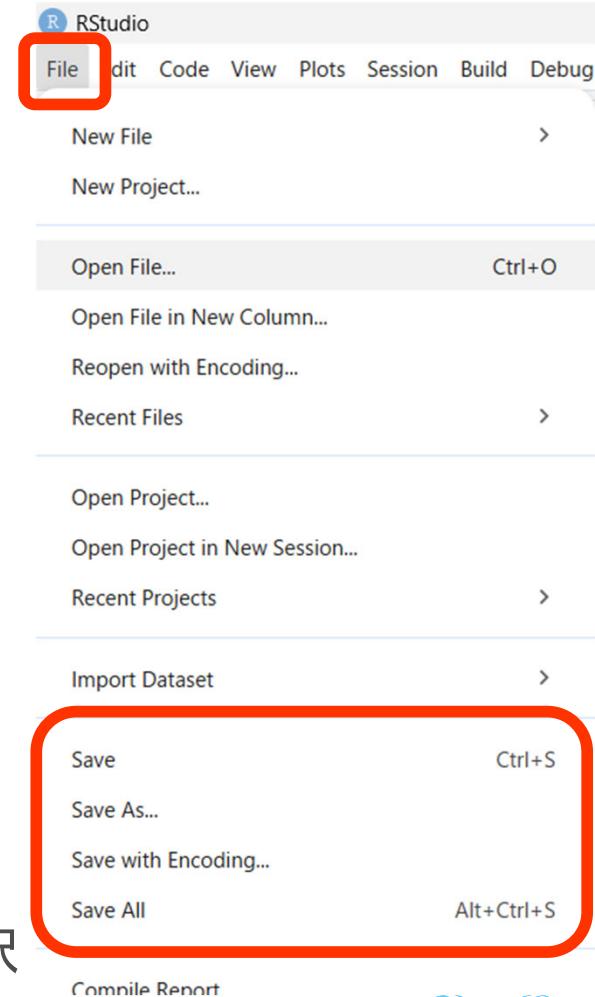
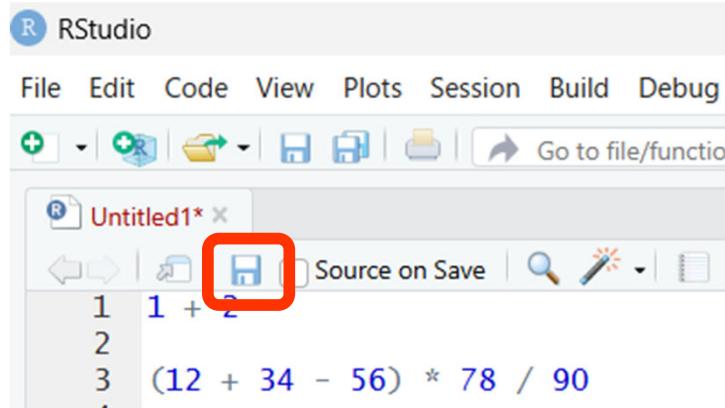
⑧ 参考文献

⑨ 関連の関数

⑩ 例

* とりあえず ①と② を見た後に ⑩ を実行すると良い？

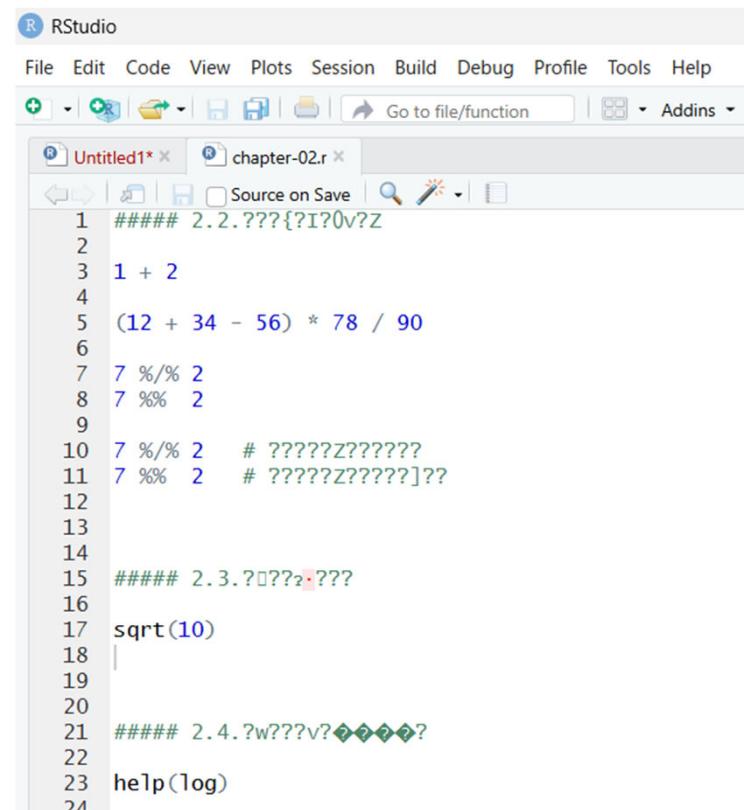
RStudio の終了



- ウィンドウの右上の [×] をクリック、又はメニューの [File] → [Quit RStudio...] を選択
- 終了する前に作成したプログラムを保存しておく場合は、
 - ソース画面をクリックし、フロッピーディスクのアイコンをクリック
 - メニューの [File] から [Save(上書き保存)]、[Save As...(ファイル名を変更して保存)]、[Save(上書き保存)]、[Save with Encoding...(文字コードを選択した上で保存)] 等を選択
⇒ 保存する際のファイルの拡張子は .r とする

R プログラムを開く

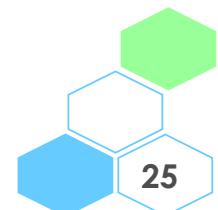
- 自分で保存したプログラム等、既存の R プログラムを開く場合は、メニューバーの [File] → [Open File] を選択し、目的の R プログラムを開く
- 環境によっては以下のように日本語等が文字化けする場合がある

```

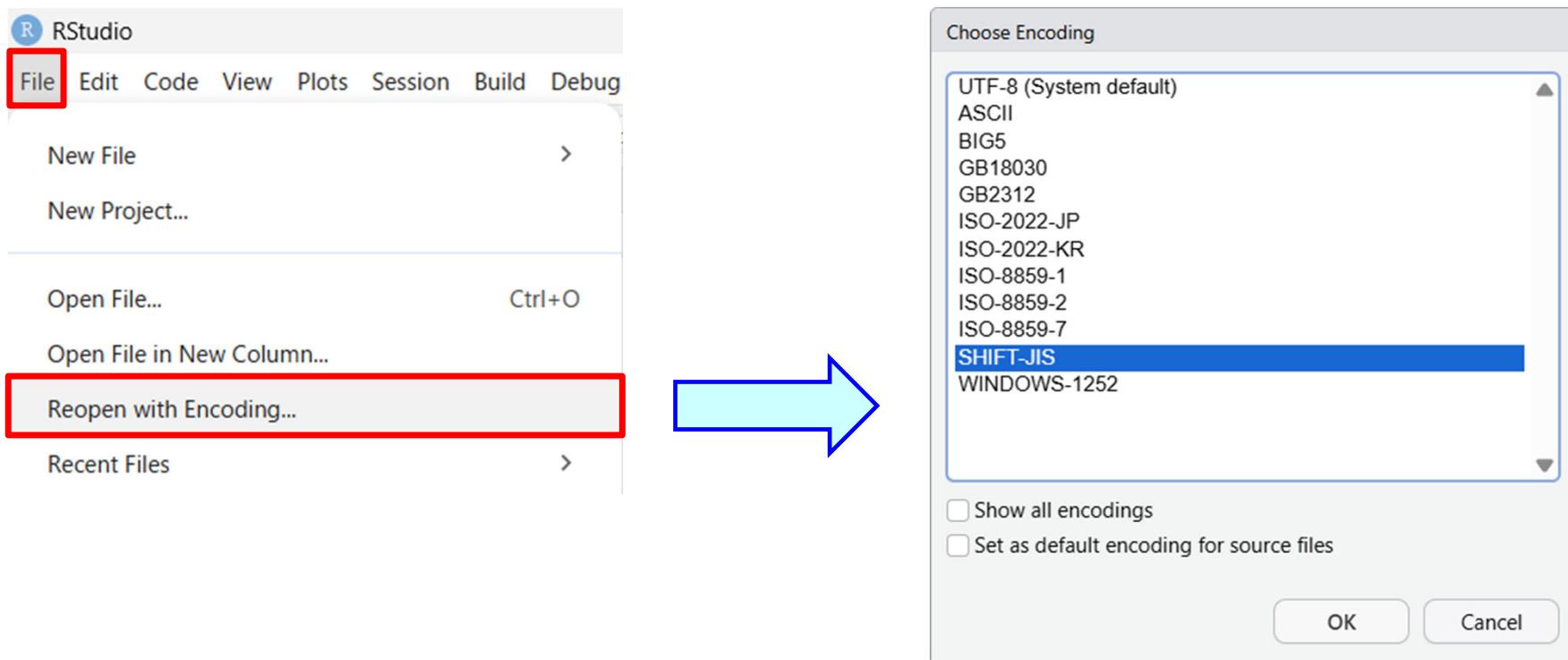
## 2.2.???{?I?0v?Z
1 + 2
(12 + 34 - 56) * 78 / 90
7 %% 2
7 %% 2
7 %% 2    # ?????Z??????
7 %% 2    # ?????Z?????]?
sqrt(10)
|
## 2.3.?□??z·??
help(log)

```



R プログラムを開く

- メニューバーの [File] → [Reopen with Encoding] を選択
- 適切な文字コード(例えば [SHIFT-JIS])を選択 ⇒ 適切な文字コードである場合は日本語等の文字化けが解消される
 - コンソール画面に文字化けしたプログラムが残っている場合は 16 頁の方法でお掃除



おまけ

- [Tab] キーを押すことで補完機能が働く

```
> sq Tab
```



途中まで入力して [Tab] キーを押すと残りのスペルが補完される

```
> sqrt
```

- コンソール画面で **↑** や **↓** で今までに実行したコマンドの履歴が辿れる

```
> ↑ 「1つ前に実行したコマンド」が表示される
```

- コンソール画面で、コマンド入力の途中で **↓** を押してしまっても…気にせずに続きを入力する ⇒ コマンド入力を中断する場合は **Esc** を押す

```
> log(2 ↓
```

```
+  
+
```

おまけ

- 記号 # を使ってコメントを付けることが出来る

```
> log(2)    # この文章は無視されます ↴
```

- 記号 ; で区切ることで複数の式を同時に実行出来る

```
> log(2); log(3); log(4) ↴  
[1] 0.6931472  
[1] 1.098612  
[1] 1.386294
```

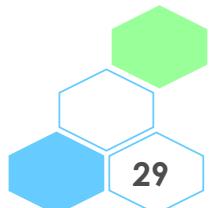
- コマンドによりヘルプ機能を利用する事も出来る

```
> help(log)          # 関数 log() のヘルプ  
 > help.search("log") # log の機能が含まれる関数  
 > apropos("log")    # "log"という文字列が含まれる関数
```



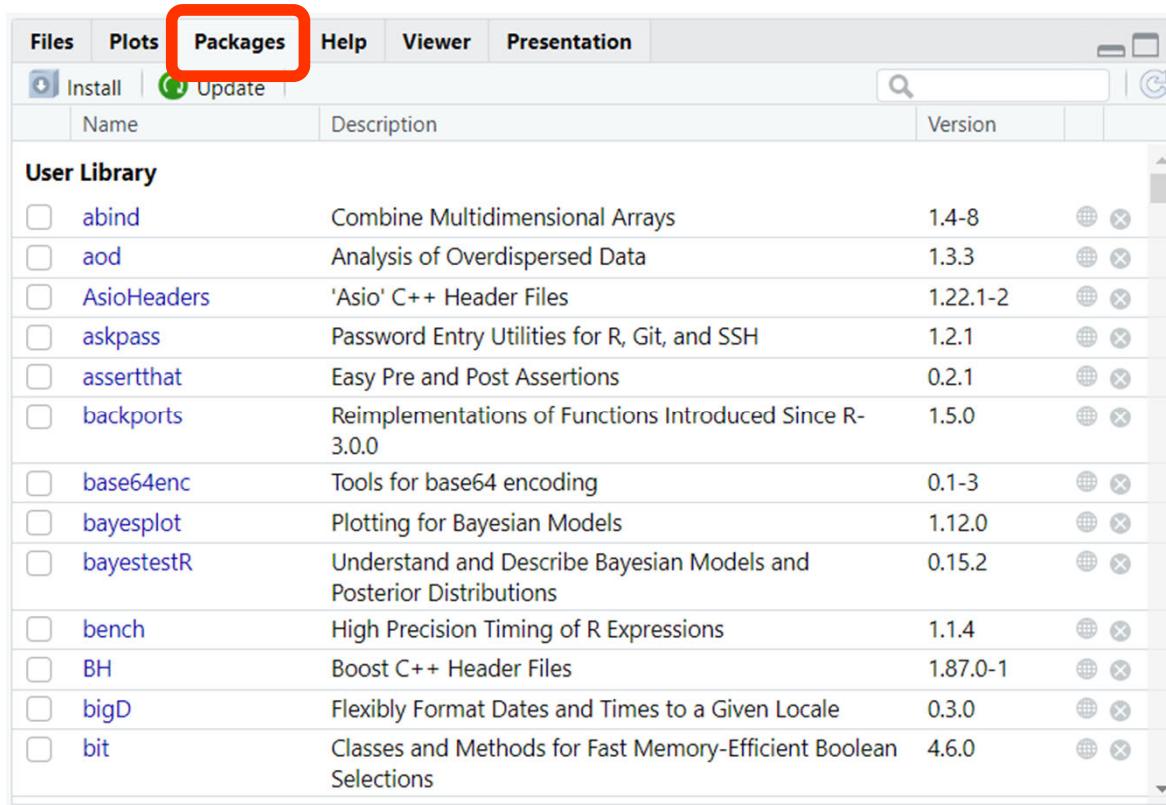
本日のメニュー

- Windows 版 R / RStudio のセットアップ方法 ← 当日までに自習
- R の基礎知識 ← 当日までに自習
- パッケージと作業ディレクトリ ← 当日までに自習
- 変数とベクトル
- ベクトルの型
- 関数とプログラミング
- データ加工とパイプ演算子
- グラフ作成(ggplot2)
- その他



パッケージとは

- R は関数とデータを機能別に分類して「パッケージ」という形にまとめている
- どのようなパッケージがあるかは関数 `library()` を実行するか、右下の画面の [Package] タブをクリックすることで確認出来る



The screenshot shows the RStudio interface with the 'Packages' tab selected. The window title is 'User Library'. The table lists various R packages with their names, descriptions, and versions. Each row includes checkboxes for selecting multiple packages.

Name	Description	Version	Action
User Library			
<input type="checkbox"/> abind	Combine Multidimensional Arrays	1.4-8	 
<input type="checkbox"/> aod	Analysis of Overdispersed Data	1.3.3	 
<input type="checkbox"/> AsioHeaders	'Asio' C++ Header Files	1.22.1-2	 
<input type="checkbox"/> askpass	Password Entry Utilities for R, Git, and SSH	1.2.1	 
<input type="checkbox"/> assertthat	Easy Pre and Post Assertions	0.2.1	 
<input type="checkbox"/> backports	Reimplementations of Functions Introduced Since R-3.0.0	1.5.0	 
<input type="checkbox"/> base64enc	Tools for base64 encoding	0.1-3	 
<input type="checkbox"/> bayesplot	Plotting for Bayesian Models	1.12.0	 
<input type="checkbox"/> bayestestR	Understand and Describe Bayesian Models and Posterior Distributions	0.15.2	 
<input type="checkbox"/> bench	High Precision Timing of R Expressions	1.1.4	 
<input type="checkbox"/> BH	Boost C++ Header Files	1.87.0-1	 
<input type="checkbox"/> bigD	Flexibly Format Dates and Times to a Given Locale	0.3.0	 
<input type="checkbox"/> bit	Classes and Methods for Fast Memory-Efficient Boolean Selections	4.6.0	 

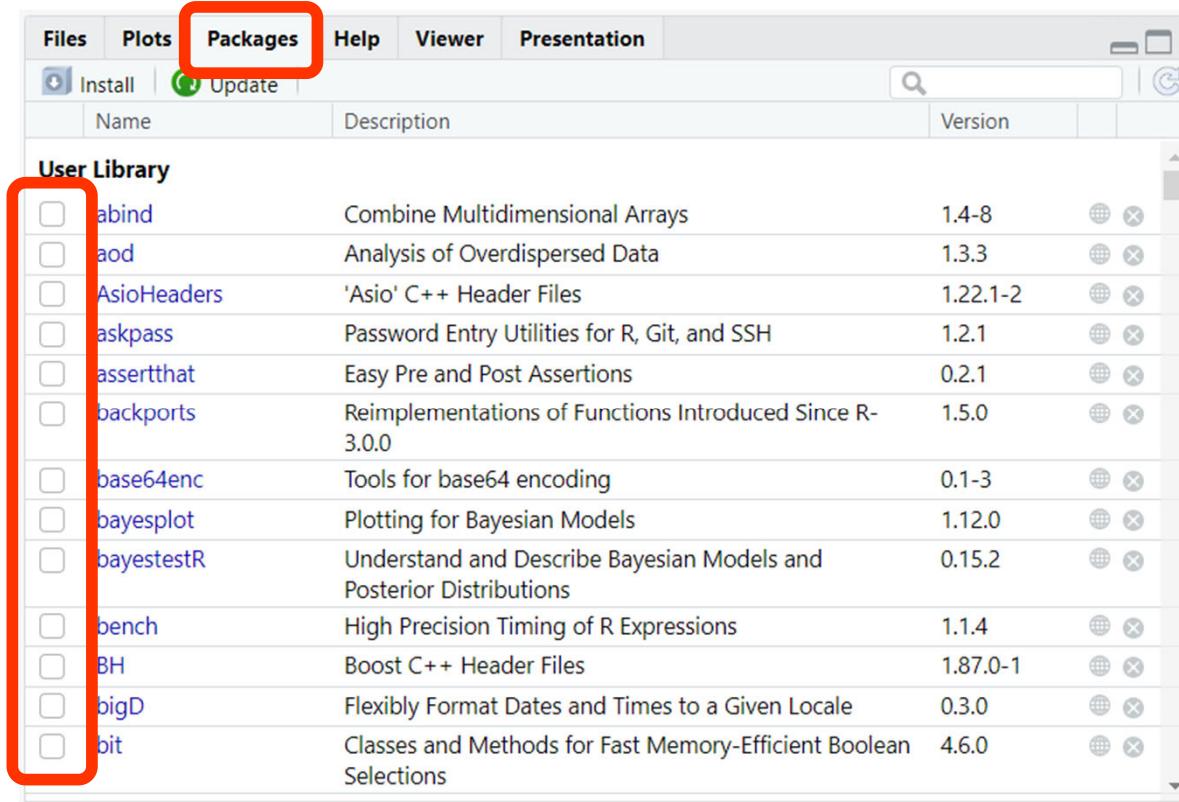
パッケージとは

- Rは関数とデータを機能別に分類して「パッケージ」という形にまとめている
(以下はほんの一例)

パッケージ名	解説
dplyr	データ加工に関するパッケージ
forcats	因子型データ(=カテゴリデータ)操作に関するパッケージ
ggplot2	グラフ作成用のパッケージ
readr	各種データファイルを読み込むためのパッケージ
stringr	文字列操作に関するパッケージ
survival	生存時間解析用のパッケージ
tibble	データフレーム(=データセット)の拡張版を提供するパッケージ
tidyverse	tidy data(我々が臨床試験で扱う様なデータ形式)に関するパッケージ群

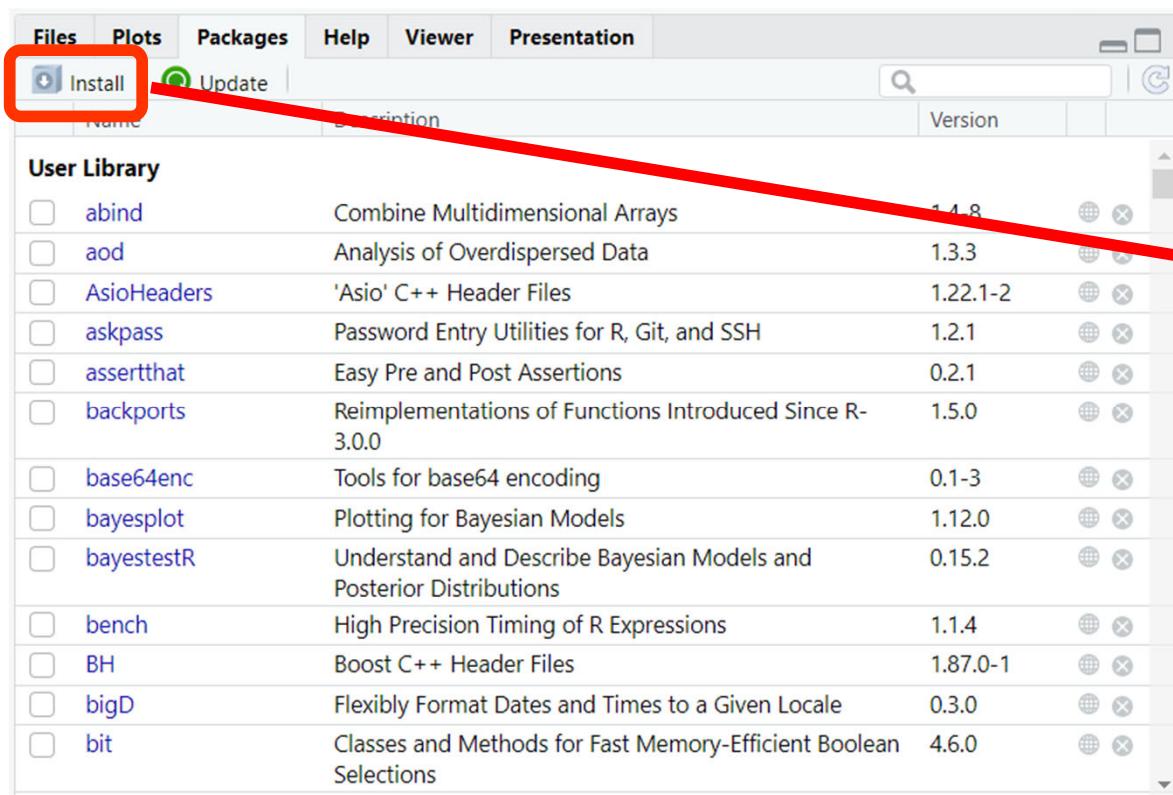
パッケージの呼び出し

- 「パッケージ」を呼び出す場合は、右下の画面の [Package] タブをクリックし、呼び出したいパッケージのチェックボックスをクリックする



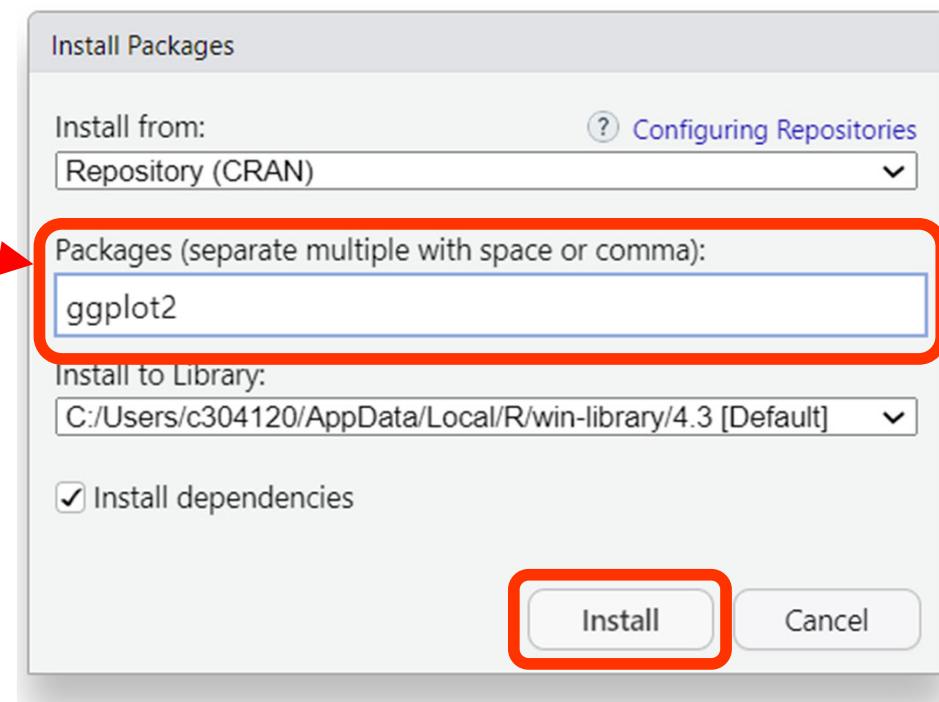
追加パッケージのインストール

- 新たに「パッケージ」をインストールする場合は、右下の画面の [Package] → [Install] タブをクリックし、インストールするパッケージ名を入力した後、[Install] ボタンをクリックする



The screenshot shows the RStudio interface with the 'Packages' tab selected. The 'User Library' section displays a list of installed packages. The 'Install' button in the top menu bar is highlighted with a red box.

Name	Description	Version
abind	Combine Multidimensional Arrays	1.4-8
aod	Analysis of Overdispersed Data	1.3.3
AsioHeaders	'Asio' C++ Header Files	1.22.1-2
askpass	Password Entry Utilities for R, Git, and SSH	1.2.1
assertthat	Easy Pre and Post Assertions	0.2.1
backports	Reimplementations of Functions Introduced Since R-3.0.0	1.5.0
base64enc	Tools for base64 encoding	0.1-3
bayesplot	Plotting for Bayesian Models	1.12.0
bayestestR	Understand and Describe Bayesian Models and Posterior Distributions	0.15.2
bench	High Precision Timing of R Expressions	1.1.4
BH	Boost C++ Header Files	1.87.0-1
bigD	Flexibly Format Dates and Times to a Given Locale	0.3.0
bit	Classes and Methods for Fast Memory-Efficient Boolean Selections	4.6.0



パッケージの呼び出しとインストール

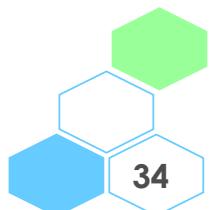
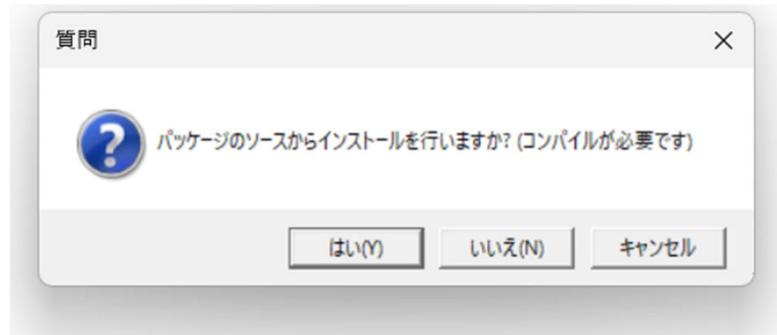
- コマンドでパッケージ「ggplot2」をインストールする場合

```
> options(repos="https://cloud.r-project.org")
> install.packages("ggplot2", dep=T)
```

- コマンドでパッケージ「ggplot2」を呼び出す場合

```
> library(ggplot2) #パッケージ ggplot2 を呼び出す
```

- インストール時に、それとなく以下のダイアログが出ている場合があるので注意
→「いいえ」を選択



パッケージ中の関数について

- パッケージ名にはリンクが張ってあり、クリックすると、そのパッケージに入っている関数の一覧が閲覧できる

Name	Description	Version	Actions
User Library			
abind	Combine Multidimensional Arrays	1.4-8	 
aod	Analysis of Overdispersed Data	1.3.3	 
AsioHeaders	'Asio' C++ Header Files	1.22.1-2	 
askpass	Password Entry Utilities for R, Git, and SSH	1.2.1	 
assertthat	Easy Pre and Post Assertions	0.2.1	 
backports	Reimplementations of Functions Introduced Since R-3.0.0	1.5.0	 
base64enc	Tools for base64 encoding	0.1-3	 
bayesplot	Plotting for Bayesian Models	1.12.0	 
bayestestR	Understand and Describe Bayesian Models and Posterior Distributions	0.15.2	 
bench	High Precision Timing of R Expressions	1.1.4	 
BH	Boost C++ Header Files	1.87.0-1	 
bigD	Flexibly Format Dates and Times to a Given Locale	0.3.0	 
bit	Classes and Methods for Fast Memory-Efficient Boolean Selections	4.6.0	 

パッケージに関する資料

- Google で「 CRAN パッケージ名 」で検索
- CRAN にアクセス → PDF 形式の資料

https://cran.r-project.org/web/packages/available_packages_by_name.html

Available CRAN Packages By Name

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)

[AalenJohansen](#)

Conditional Aalen-Johansen Estimation

[aamatch](#)

Artless Automatic Multivariate Matching for Observational Studies

[AATtools](#)

Reliability and Scoring Routines for the Approach-Avoidance Task

[ABACUS](#)

Apps Based Activities for Communicating and Understanding Statistics

[abasequence](#)

Coding 'ABA' Patterns for Sequence Data

[abbreviate](#)

Readable String Abbreviation

[abc](#)

Tools for Approximate Bayesian Computation (ABC)

[abc.data](#)

Data Only: Tools for Approximate Bayesian Computation (ABC)

[ABC.RAP](#)

Array Based CpG Region Analysis Pipeline

[ABCAnalysis](#)

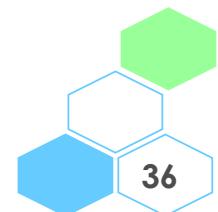
Computed ABC Analysis

[ABCDscores](#)

Summary Scores of the Adolescent Brain Cognitive Development (ABCD) Study

[abcclass](#)

Angle Based Classification



作業ディレクトリの設定と確認

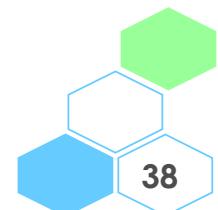
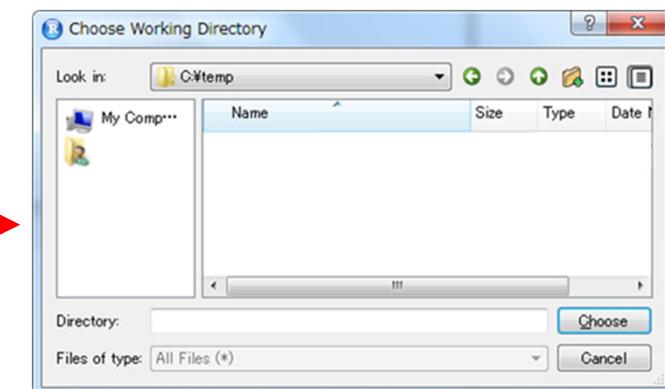
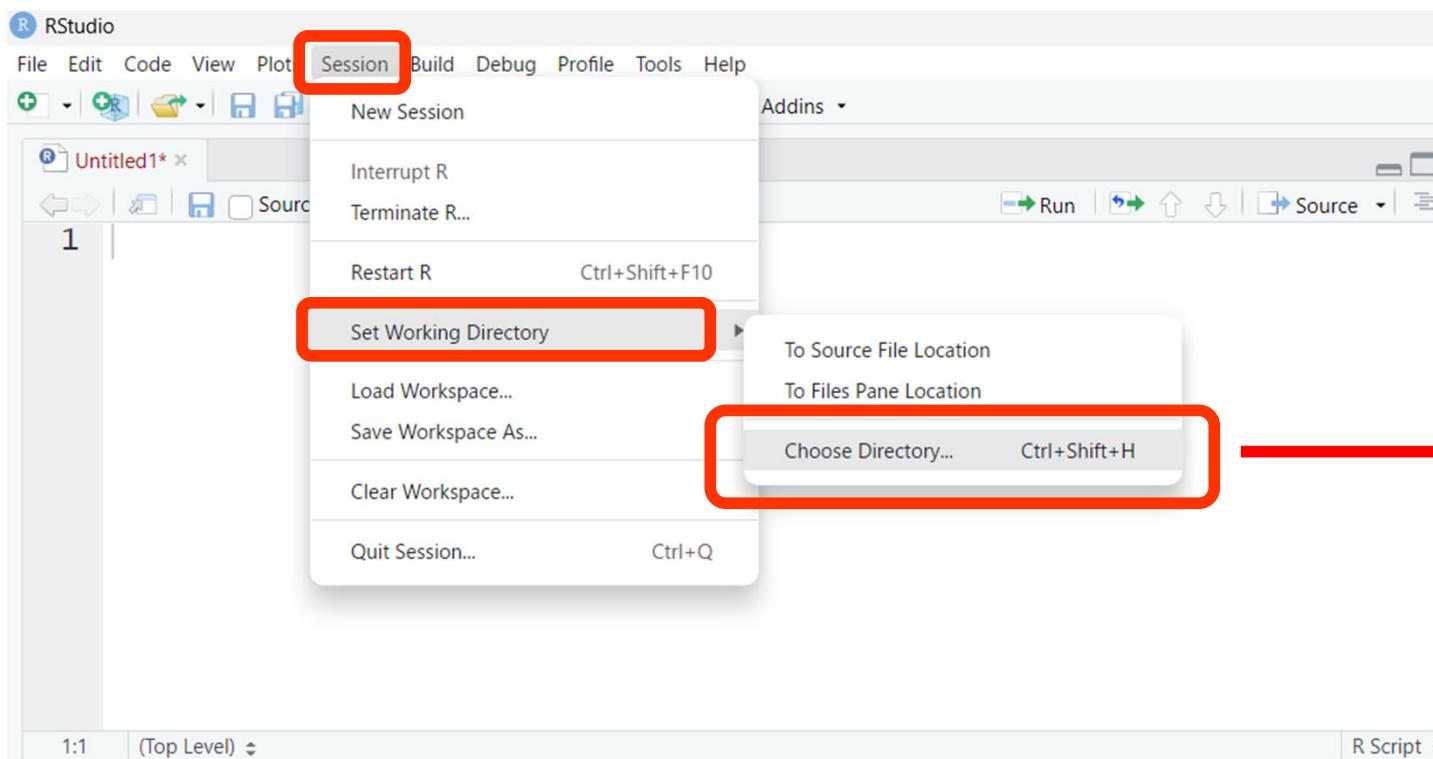
- R は、ファイルの読み込み/書き出し先として、まず作業ディレクトリを参照するので、頻繁に使うフォルダを作業ディレクトリとして設定しておくと、後々楽になる
- コマンドにて作業ディレクトリの設定や確認を行う場合は以下のようにする

```
> setwd("c:/temp")      # 作業ディレクトリを指定  
> getwd()                # 現在の作業ディレクトリを表示  
[1] "c:/temp"
```

注: R や多くのプログラム言語では ¥ を「¥¥」として表記する必要があるので、例えば上記で「setwd("c:¥temp")」とするとエラーとなる（正しくは「setwd("c.¥¥temp")」）
⇒ 面倒なので、Windows 版 R では ¥ の代わりに / を使う方が便利

作業ディレクトリの設定と確認

- メニュー [Session] → [Set Working Directory] → [Choose Directory...] から
フォルダを選択



演習 1

1. RStudio を起動して下さい
2. プログラムを書く画面を表示して下さい
3. $\sqrt{2}$ を計算して下さい
4. 「`1 +`」という命令を実行し、どうなるかを確認して下さい
その後、左下のコンソール画面に「`2`」と入力した後、[Enter] キーを押してください
5. 対数計算をするため `log(2)` を実行して下さい
6. 5. で `log(2)` を実行しましたが、底が 10 なのか `e` ののかよく分かりません。
関数 `log()` に関するヘルプを見て下さい
7. 3. を計算する命令を「履歴機能」を用いて実施して下さい

演習 1

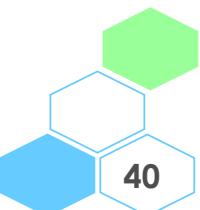
8. RStudio で以下を実行して、パッケージをインストールしてください

```
> install.packages(c("tidyverse", "readxl", "haven"), dep=T)
```

9. C ドライブの直下に「 temp 」という名前のフォルダを作成した後、作業ディレクトリを「 C:\temp 」に変更してください
(他の場所・フォルダ名が良ければ、そちらでも結構です)

10. 1. ~ 9. まで全て終了したら、RStudio を終了して下さい

★ 事前のご準備・自習内容は以上です。ありがとうございました！





R に入門される方向けのセッション

舟尾 暢男（武田薬品工業）



発表者の略歴

経歴

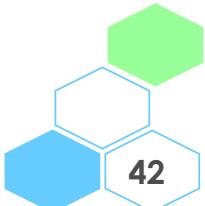
1998年：大阪教育大学教養学科数理科学専攻中退
2002年：大阪大学基礎工学部情報科学科数理科学コース中退
2004年：大阪大学大学院基礎工学研究科システム人間系数理科学分野修了
現在：製薬会社勤務(勤続 22 年)

仕事

臨床試験・市販後調査における統計解析業務、医薬品の承認申請業務、SAS や R 関連の外部活動もボチボチ

著書

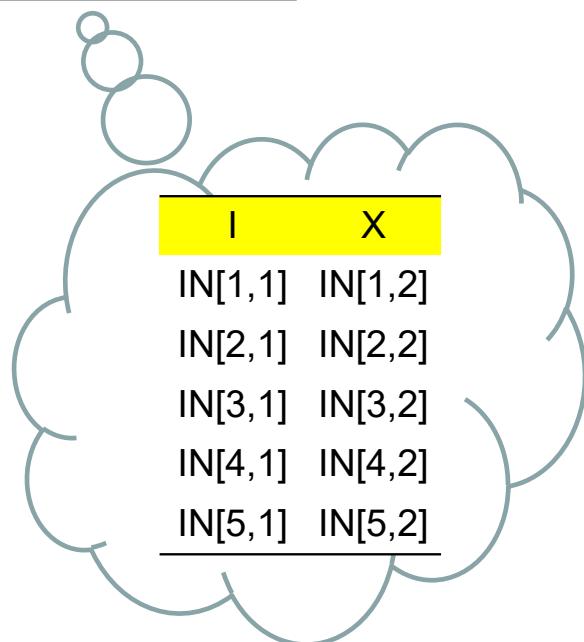
The R Tips 第 3 版(オーム社)
データ解析環境 R (工学社、共著)
R Commander ハンドブック(オーム社)
R で学ぶデータマイニング 1 データ解析編(オーム社、共著)
R で学ぶデータマイニング 2 シミュレーション編(オーム社、共著)
R 流！ イメージで理解する統計処理入門(カットシステム)
S-PLUS による混合効果モデル解析(丸善出版、翻訳作業のお手伝い)
統計解析ソフト SAS(工学社、共著)
R で学ぶプログラミングの基礎の基礎(カットシステム)
SAS Studio によるやさしい統計データ分析(オーム社、共著)
R によるデータ分析のレシピ(オーム社)



イントロ: SAS の特徴

IN	I	X
1	3	← 1 行目
2	6	← 2 行目
3	9	← 3 行目
4	12	← 4 行目
5	15	← 5 行目

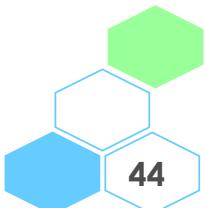
- 1 つのデータセットに対する処理を行うのが基本
- 変数名の大文字と小文字を区別しない
- フォーマットという概念がある
- 「変数 X について ■ の処理を行う」と命令すれば、全ての行に対して処理が行われる
 - 「変数 X の ● 番目の行」に対して処理を行うのが通常の言語
 - 特定の行に対して処理を行う場合は少々面倒
- 関数 (fcmp プロシージャ) やクラスの定義ではなくマクロ定義が主流





イントロ: 「SAS があるのに R を使う」理由

- 行列計算・数値計算を伴う解析（SAS/IML よりも R の方が融通が利く）
- グラフ作成
- シミュレーション（SAS でプロシージャを伴うと計算速度が急に遅くなる）
- SAS で出来ないが R で出来る解析(特に最近提案された統計手法)
- 会社・業界で R を使って解析している(例えば非線形回帰系)
- 同僚・上司・Global がよく R を使う





イントロ: 「SAS があるから R を使わない」理由

- データハンドリング
 - SAS で出来る解析(proc means、freq、glm、mixed、mcmc、…)
 - 自社の SAS マクロを使えば出来る解析
 - レポート出力(PDF、EXCEL、RTF への出力)
 - 正式な業務だから(例: CDISC・電子データ提出を予定した臨床試験における解析)

今後ここが変わる可能性

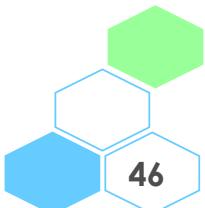


イントロ: 製薬業界で R を使う場面が増えています

- オープンソースの統計解析用フリーソフトウェア(90 年代後半に誕生)
- Windows、MacOS、Linux 系 OS に対応
- 全世界の開発者によって作成されている「パッケージ」で機能追加が可能
- 製薬業界では長らく、生物統計家による臨床試験データ分析のごく一部で R を活用する程度で留まっていたが、R Consortium R Submission Working Group (RCRSWG)は、承認申請のためのプログラム言語として R を採用、「 R にて申請パッケージを作成し、FDA 審査官に適切に転送できる」ことをテスト、FDA は解析結果の再現・フィードバックを実施する
- いくつかの製薬会社さんでも積極的に R を承認申請業務に積極活用、CRO さんでも既に R の業務を請け負っており、R パッケージを積極的に開発しているところも
- **本日は R with Pharma Lab の第 1 回ということで、「本日以降の皆様の発表演題を聴講・理解するために必要となる R の基礎 ⇒ 必要だけど面倒なので説明したくない事項を解説」「他の皆様の発表ネタ(データ加工、文字列処理、日時データ、帳票作成、shiny、電子データ関連、統計関連、楽しいグラフ、等)を食いつぶさない」ことを目的として説明させていただきます**

【参考】

- **R consortium**: 全世界の R ソフトウェアユーザや開発者のコミュニティをサポートするために組織されたグループ
- **RCRSWG**: **R consortium** に所属している製薬業界メンバーで構成、承認申請時に **R based** で作成された承認申請時の電子データを規制当局へ提出することを目標に活動している





本セッションの趣旨と進め方

- 当方の娘(高校生)の担任の先生

「数学とかの問題集を解くときは、1回目はすぐに解答を読む、何だったら問題を読んだ後いきなり解答を見ても良い、2回目に考えて(思い出して)解きなさい。解く問題が大量にある場合は、問題をいくつかのグループに分けて各グループで【1回目 → 2回目 →】のサイクルを回しなさい。」

- 本セッションの趣旨と進め方

- 各トピックについて、ざっと説明した後に演習があります
- 演習時間は短め、すぐに解説があります(事前にご提供した R プログラムをあらかじめ開いておいてください)
- 本セッションだけでは R の理解は出来ません、本セッション後に皆様にて復習いただくことで、はじめて R の理解が深まります

復習

1. RStudio を起動して下さい
2. プログラムを書く画面を表示して下さい
3. $\sqrt{2}$ を計算して下さい
4. 「`1 +`」という命令を実行し、どうなるかを確認して下さい
その後、左下のコンソール画面に「`2`」と入力した後、[Enter] キーを押してください
5. 対数計算をするため `log(2)` を実行して下さい
6. 5. で `log(2)` を実行しましたが、底が 10 なのか `e` なのかよく分かりません。
関数 `log()` に関するヘルプを見て下さい
7. 3. を計算する命令を「履歴機能」を用いて実施して下さい
8. C ドライブの直下に「`temp`」という名前のフォルダを作成した後、作業ディレクトリを
「`C:\temp`」に変更してください(他の場所が良ければ、そちらでも結構です)



本日のメニュー

- Windows 版 R / RStudio のセットアップ方法 ← 当日までに自習
- R の基礎知識 ← 当日までに自習
- パッケージと作業ディレクトリ ← 当日までに自習
- 変数とベクトル：25 分
- ベクトルの型：20 分、休憩 5 分
- 関数とプログラミング：関数 15 分、if 文 15 分、for 文 15 分
- データ加工とパイプ演算子：データ加工 25 分、休憩 5 分、パイプ演算子 10 分
- グラフ作成(ggplot2)：グラフ作成 25 分、カスタマイズは自習の予定
- その他：関数定義の見方は自習、shiny の紹介と「まとめ」で 5 分

変数と代入

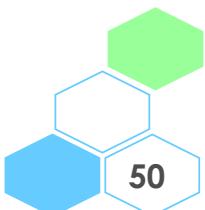
- 変数とは、値や計算結果を保存する箱のこと
- R では「変数名 \leftarrow 計算式」で計算結果を保存出来る
これを「変数」への「代入」という(\leftarrow は代入記号※)

```
> x <- 1 + 2    # x という変数に 1 + 2 の結果を代入
```

- 保存した値は変数名を入力することで再表示される

```
> x  
[1] 3
```

※ 「 $x = 1+2$ 」と、 $=$ 記号でも代入することも出来る(多くの場合、不具合は起きない)



変数と代入

- 変数を使ってさらに計算を行うことも出来る

```
> x^2  
[1] 9
```

- 変数を使った計算結果を、さらに変数に代入することも可

```
> y <- x^2      # y という変数に x^2 = 9を代入
```

- 丸括弧を使うと、値の代入と表示が同時に実行できる

```
> ( y <- x^2 )  
[1] 9
```

変数の命名規則

- 大文字と小文字を区別する → x と X は別物
- 変数名にはローマ字や数字を使うことができるが、変数名の先頭を数字にすることはできない → 例えば「1A」という変数名はエラーとなる
- 以下に挙げる名前は(R の処理系によって)先に予約されている予約語なので、変数名として使用できない

**break else FALSE for function if in Inf NA
NaN next NULL repeat TRUE while**

変数を使った計算例

```
> x <- 1      # x に 1 を代入する  
> y <- 2      # y に 2 を代入する  
> x + y      # x の中身と y の中身の和  
[1] 3  
  
> x <- 4      # 変数に値を代入し直すと古い値は破棄され  
               # 新しい値で上書きされる  
[1] 4
```

ベクトル

- ある 5 人の体重について、体重の合計値を求める

```
> x1 <- (55 + 60 + 65 + 70 + 75)  
> x1  
[1] 325
```

- 同じ 5 人の体重について平均値を求める

```
> x2 <- (55 + 60 + 65 + 70 + 75) / 5  
> x2  
[1] 65
```

- 同じ 5 人の体重について、今度は分散を求め…
(同じデータなのだから、手間を省く方法は無いの？？)

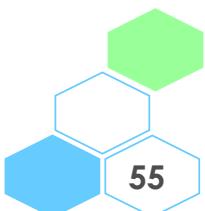
ベクトル

- 1つの変数に1つの値を代入することが出来たように、
1つの変数に複数の値を代入することも出来る

```
> x <- c(55, 60, 65, 70, 75)
```

- 複数の値が代入された変数を「ベクトル」とよぶ
※ 1つの値しか代入されていない変数も「ベクトル」

```
> x  
[1] 55 60 65 70 75
```



ベクトル

- R には「ベクトル」用の関数が多数用意されている

```
> sum(x)  
[1] 325
```

- ベクトルに入っているデータ数(要素数)を計算する場合は関数 `length()` を使用する

```
> length(x)  
[1] 5
```

ベクトル

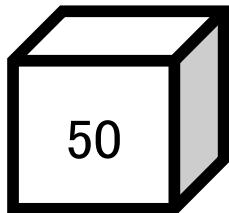
R で用意されているベクトル用関数(の一部)

関 数	<code>cor(x)</code>	<code>max(x)</code>	<code>mean(x)</code>	<code>median(x)</code>	<code>min(x)</code>
意 味	相関係数	最大値	平均値	中央値	最小値
関 数	<code>prod(x)</code>	<code>summary(x)</code>	<code>sd(x)</code>	<code>sum(x)</code>	<code>var(x)</code>
意 味	総積	要約統計量	標準偏差	総和	不偏分散

変数とベクトルの違い

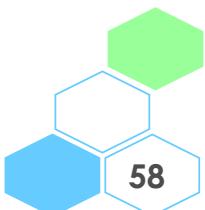
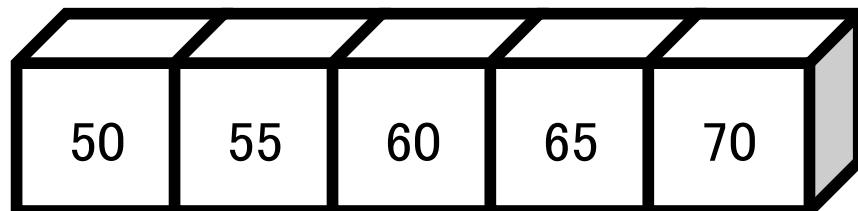
- 変数は箱が 1 個

変数 a:



- ベクトルは箱が 1 個以上 → 「ベクトルの箱が 1 つバージョン」が「変数」

ベクトル x:



ベクトル操作

- ベクトルの中の数値を取り出す場合は[]を使用する

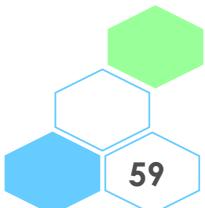
```
> x[2]          # 変数 x の 2 番目の値を取り出す  
[1] 60
```

- ベクトルとベクトルを結合する場合は以下の様にする

```
> c(x, 80)      # c(ベクトル, ベクトル)  
[1] 55 60 65 70 75 80
```

- ベクトルの中の数値を書き換える場合は以下の様にする

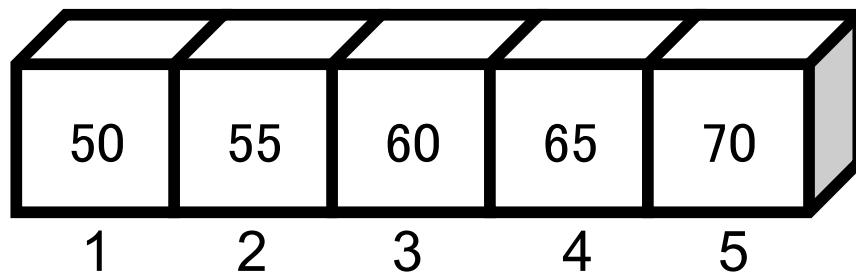
```
> x[2] <- 99    # 変数 x の 2 番目の値を 99 に  
> x  
[1] 55 99 65 70 75
```



変数とベクトルの違い

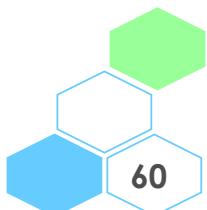
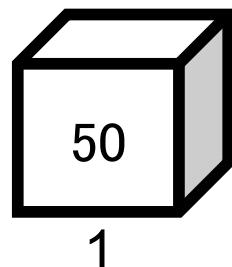
- ベクトル x について、 $x[1]$ の値は 50

ベクトル x :



- 変数 a について、 $a[1]$ の値は 50
→「変数」は「ベクトルの箱が 1 つしかないバージョン」

変数 a :



ベクトル操作

規則性のあるベクトルを生成する関数

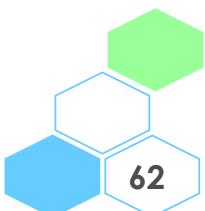
関数・コマンド	機能
<code>1:5</code>	<code>1, 2, 3, 4, 5</code> を生成する
<code>5:1</code>	<code>5, 4, 3, 2, 1</code> を生成する
<code>seq(1, 5, by=2)</code>	1 から 5 まで 2 ずつ増加する等差数列を生成する
<code>rep(1, 3)</code>	1 を 3 個繰り返した数列を生成する
<code>rep(c(1, 2, 3), 3)</code>	(1, 2, 3) を 3 個繰り返した数列を生成する

演習 2

1. 変数 x に 2 を代入した後、 $3 + x^2$ を計算してください
2. 次の文を実行すると、変数 a と変数 b には何が入っているでしょうか

```
a <- 1.5 -> b
```

3. 2. で生成した変数 a について、変数 a の整数部分を求める `trunc(a)` と
変数 a の切り下げを行う `floor(a)` を実行して下さい
→ `trunc(a)` と `floor(a)` の結果の違いは何でしょうか



演習 2

4. 以下のデータを作成し、変数 x に代入して下さい

9 8 7 6 5 4 3 2 1

5. 以下のデータを作成し、変数 y に代入して下さい

2 4 6 8

6. 変数 x の 4 番目の値を取り出して下さい

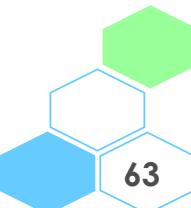
7. 変数 x の 2、4、6、8 番目の値を取り出し、変数 z へ代入して下さい

8. 変数 x と変数 z の平均値はどちらが大きいでしょう

9. 変数 x と変数 z の標準偏差はどちらが大きいでしょう

10. 以下を実行した後、変数 s より中央値を取り出して下さい

```
s <- summary(x)
```



おまけ①: 要素のラベル・*names* 属性

- ベクトル(やリスト)には *names* 属性と呼ばれる情報が付与出来、このラベルを使って要素を取り出すことができる

```
> x <- 1:5
> names(x) # 通常のベクトルには names 属性は付いていない
NULL
> names(x) <- c("a", "b", "c", "d", "e")
> x
a b c d e
1 2 3 4 5
> x["a"] # ラベルを用いた要素の参照
a
1
```

おまけ②: 関数 round()

- R の丸め関数 round() は IEEE 規約に基づいたもので、四捨五入とは微妙に異なる(五入ばかりでなく五捨もあり得る)
 - 一番近い丸め結果候補が 1 つだけの場合、その数に丸める
 - 一番近い丸め結果候補が 2 つある場合、末尾が偶数のものに丸める
 - 丸め処理は 1 段階で行わなければならない

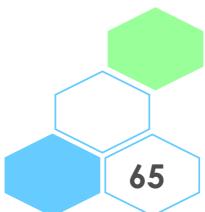
```
> round(122.5); round(122.51); round(123.5); round(123.461)
[1] 122    # round(122.5)   → 2. を適用し五捨 !
[1] 123    # round(122.51)  → 1. を適用し五入
[1] 124    # round(123.5)   → 2. を適用し五入
[1] 123    # round(123.461) → 1. と 3. を適用し四捨
```

- ちなみに SAS では…

```
data X ;
X=round(122.5) ;
run ;
```

→

VIEWTABLE: Work.X	
	X
1	123



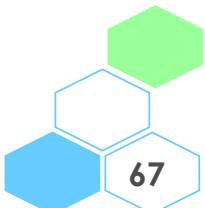
おまけ②: 関数 *myround()*

- R で四捨五入を行う関数 *myround()* を定義する

```
> myround <- function(x, n=0) {  
+   floor( round(abs(x)*10^(n)+0.5, 10) ) *sign(x)/10^(n)  
+ }  
  
> myround(c(122.5, 122.51, 123.5, 123.461))  
[1] 123 123 124 123  
  
> myround(c(0.145, -0.145), 2)  
[1] 0.15 -0.15
```

本日のメニュー

- Windows 版 R / RStudio のセットアップ方法 ← 当日までに自習
- R の基礎知識 ← 当日までに自習
- パッケージと作業ディレクトリ ← 当日までに自習
- 変数とベクトル
- ベクトルの型
- 関数とプログラミング
- データ加工とパイプ演算子
- グラフ作成(ggplot2)
- その他



ベクトルの型

- データの種類としては主に以下のようなものがある
 - 実数値: 1.5 や 3
 - 複素数: $1+2i$ (i は虚数)
 - 文字列: "ABC" ⇒ 文字列処理は他の方の発表演題にて(詳述しない)
 - 因子: "M" を「男性」、"F" を「女性」として扱う ⇒ "M" や "F" はカテゴリー
 - 論理値: TRUE(真)と FALSE(偽)
 - 日付値: "2013/10/15" ⇒ 日付値の処理は他の方の発表演題にて(ここでは割愛)
- 特殊な値についても、ついで紹介
 - NA : 「Not Available(欠測)」の略、「欠測」というデータがある
 - NULL: 「空っぽ」「何もない」という意味 ⇒ 0 や NA と違い、NULL は「0 や 欠測ですらない」
 - NaN : 「Not a Number」の略で、非数(0 / 0 の計算結果等)
 - Inf : 無限大(∞)

実数値ベクトル

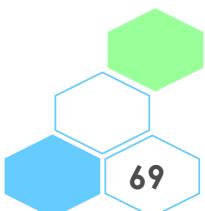
- 略して「数値ベクトル」とも呼ばれる
- 今まで扱っていたものは全て実数型ベクトル

```
> x <- c(1, 2, 3)
```

- R では(良くも悪くも)実数と整数をあまり区別せずに処理することが多いが、データを明示的に整数として扱いたい場合、関数 `as.integer()` で整数型ベクトルに変換することも出来る

```
> y <- as.integer(x)
```

- 整数型については後で個人的見解を述べる



文字型ベクトルと因子ベクトル

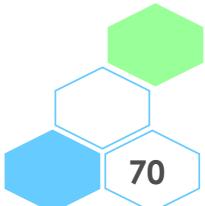
- 文字列を要素とするベクトルを文字型ベクトルと呼ぶ
- 文字列は"(ダブルクオーテーション)又は'(シングルクオーテーション)で囲む必要がある

```
> x <- c("L", "M", "H", "L")
```

- 関数 factor() を用いることで、カテゴリを要素としたベクトル(因子ベクトル)を作成することが出来、関数 levels(y) でカテゴリ化されているかを確認することが出来る

```
> x <- c("L", "M", "H", "L")
> ( y <- factor(x, levels=c("L", "M", "H")) ) 
[1] L M H L
Levels: L M H

> library(forcats)
> y <- fct_relevel(y, "M", "L")      # カテゴリの順番を入れ替え
> y
[1] L M H L
Levels: M L H
```



論理型ベクトル

- TRUE(真)と FALSE(偽)を要素とするベクトルを論理値型ベクトル(または「論理型ベクトル」と呼ぶ)
- TRUE(真)や FALSE(偽)はそれぞれ T と F と略記することも出来る
- 値同士の大小比較を行った結果は論理型ベクトルとなる

```
> 1 < 2      # 1 は 2 よりも小さい?  
[1] TRUE  
  
> 3 < 2      # 3 は 2 よりも小さい?  
[1] FALSE
```

ベクトル型のチェックと変換

- ベクトル型のチェックや「ある型から別の型へ変換」する際は以下の関数を使用する
- 型が予想できない場合は、関数 `str()` や関数 `typeof()` を用いるのが便利

型	型のチェックを行う関数	型の変換を行う関数
実数値	<code>is.numeric()</code>	<code>as.numeric()</code>
整数値	<code>is.integer()</code>	<code>as.integer()</code>
複素数	<code>is.complex()</code>	<code>as.complex()</code>
文字列	<code>is.character()</code>	<code>as.character()</code>
因子	<code>is.factor()</code>	<code>as.factor()</code>
論理値	<code>is.logical()</code>	<code>as.logical()</code>
日付値	—	<code>as.Date()</code>

ベクトル型のチェックと変換

```
> x <- "123"          # x は文字型  
  
> str(x)              # x が何なのか調べる便利な関数  
  
chr "123"  
  
> typeof(x)           # x の型を調べる便利な関数  
  
[1] "character"  
  
> x + 1                # 文字と数値は足し算できない  
  
Error in x + 1 : non-numeric argument to binary operator  
  
> y <- as.numeric(x)  # x を数値型に変換  
  
> y + 1                # 今度はうまく行く  
  
[1] 124  
  
> c(1L, 3L, 5L)        # おまけ：明示的に整数型とする方法  
  
[1] 1 3 5
```

値を整数として扱いたい場合

- 関数 `as.integer()` で整数型ベクトルに変換することも出来るが、例えば 1.5 を整数型に変換してしまうと値が 1 に丸められてしまう
- 個人的には実数値型ベクトルは実数値型のまま値をもっておき「値が整数かどうか」をチェックする場合は以下の関数を定義して使う

```
is.wholenumber <- function(x) { abs(x-round(x)) < (tol=.Machine$double.eps^0.5) }
```

- 関数 `is.wholenumber()` の使用例を挙げる → 結果は論理値で返される

```
> is.wholenumber(1)
[1] TRUE
> is.wholenumber(1.5)
[1] FALSE
```

文字列操作

- 数値以外のデータ処理の例として、文字列に関する処理の方法をいくつか紹介
 - より有用なパッケージ「stringr」: <https://rstudio.github.io/cheatsheets/html/strings.html>

関数	意味
chartr(文字ベクトル1, 文字ベクトル2, 文字ベクトル3)	文字ベクトル 3 について、文字ベクトル 1 を文字ベクトル 2 に置換
grep("文字列", 文字ベクトル)	文字ベクトルの各要素について、“文字列”が含まれているかどうかをチェック(該当する要素の番号、該当なしの場合は 0)
nchar(文字ベクトル)	文字ベクトルの文字数を数える
paste(文字ベクトル1, 文字ベクトル2, sep="区切り文字")	文字ベクトル 1 と文字ベクトル 2 を結合する (区切り文字は引数 sep で指定する)
substring(文字ベクトル, a, b)	文字ベクトルについて a 文字目から b 文字目までを抽出
toupper(文字ベクトル)	文字ベクトルを大文字に変換
tolower(文字ベクトル)	文字ベクトルを小文字に変換
unlist(strsplit(文字ベクトル, ","))	文字ベクトルをカンマごとに区切った後、ベクトルに変換

文字列操作

```
> ( x <- "You are fool." )
[1] "You are fool."
> chartr("f", "c", x)          # x の中の f を c に置換
[1] "You are cool."
> nchar("12組456789番")       # 半角文字も全角文字も 1 文字としてカウント
[1] 10

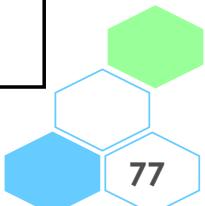
> paste("ABC", "DEF")         # デフォルトでは文字列間にスペースが入る
[1] "ABC DEF"
> paste("ABC", "DEF", sep=",") # 区切り文字は引数 sep で指定
[1] "ABC, DEF"
> paste("ABC", "DEF", sep="")   # 区切り文字を入れない場合
[1] "ABCDEF"
> substring("ABCDEF", 2, 5)    # "ABCDEF" の 2 ~ 5 文字目を抽出
[1] "BCDE"
```

演習 3

1. 変数 x に 1 を代入して下さい
2. 適当な関数を用いて、1. で作成した変数 x が整数かどうかをチェックして下さい
3. 変数 x を整数型に変換し、変数 y に代入した後、適当な関数を用いて、変数 y が整数かどうかをチェックして下さい
4. 関数 `is.wholenumber()` を使って、1. で作成した変数 x が整数かどうかをチェックして下さい
5. 文字列 “1,2,3,4,5” を変数 x に代入して下さい
6. 変数 x の文字数を調べて下さい
7. 変数 x について、5 文字目を抽出して下さい。
8. 以下の命令を実行した後、変数 y の型を調べて下さい

```
y <- unlist(strsplit(x, ","))
```

9. 変数 y を数値ベクトルに変換し、変数 z に代入して下さい



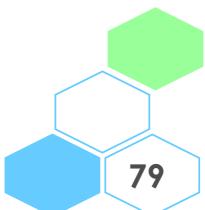
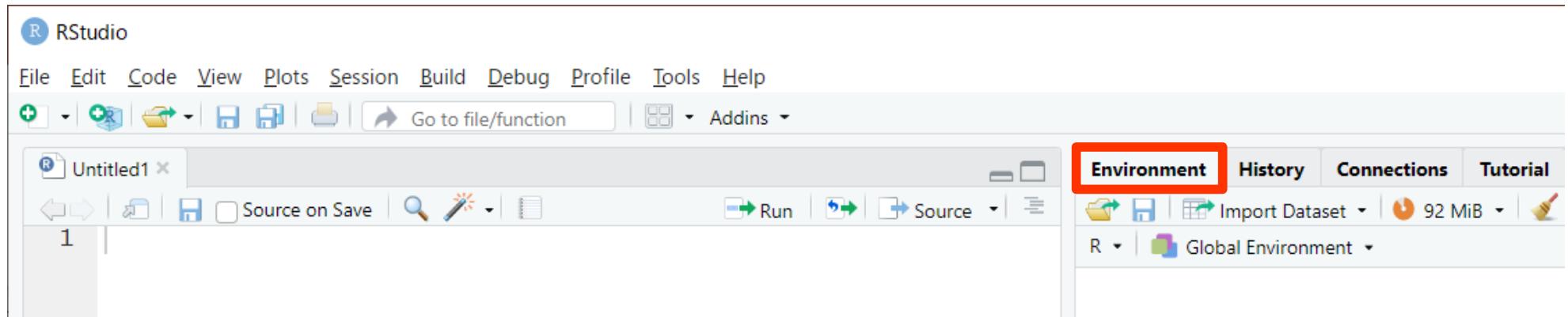
おまけ: 組み込み定数

- Rには5つの定数(定数ベクトル)が用意されている

```
> LETTERS      # アルファベットの大文字
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T"
[21] "U" "V" "W" "X" "Y" "Z"
> letters      # アルファベットの小文字
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
[21] "u" "v" "w" "x" "y" "z"
> month.abb   # 月の名前(略称)
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
> month.name   # 月の名前
[1] "January"    "February"    "March"        "April"        "May"          "June"
[7] "July"        "August"       "September"    "October"     "November"    "December"
> pi            # 円周率
[1] 3.141593
```

おまけ

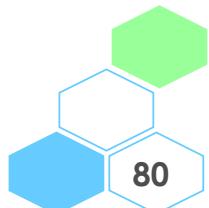
- 様々な変数や関数を作成したが、作成した変数や関数は右上の [Environment] のタブをクリックすることで一覧として見ることが出来る





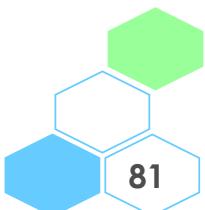
本日のメニュー

- Windows 版 R / RStudio のセットアップ方法 ← 当日までに自習
- R の基礎知識 ← 当日までに自習
- パッケージと作業ディレクトリ ← 当日までに自習
- 変数とベクトル
- ベクトルの型
- 関数とプログラミング
- データ加工とパイプ演算子
- グラフ作成(ggplot2)
- その他



プログラミングについて

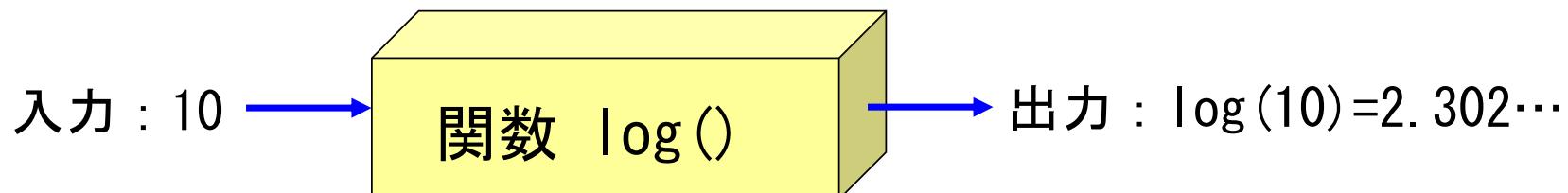
- プログラミングとは
 - 人間がコンピュータに命令をすること
 - R の場合「ユーザーが R のコマンドをひとつひとつ記述する」作業のこと
⇒ 大抵は関数を定義する
- R におけるプログラミングのための道具は…
 - 変数、ベクトル、**関数定義**
 - 条件分岐(if)
 - くり返し(for)



関数について

- R では「特別な機能を果たす命令」を「関数」という形で呼び出すことが出来る
- 関数 $\log(x)$ は「 x の値の対数を計算する関数」だが、関数に x の値を指定する場合には「引数」という形で関数に与える

```
> log(10)  
[1] 2.302585
```



関数について

- 引数に式や関数を指定すれば、それを評価した値が引数として使われる
- 引数が 2 個以上ある場合はコンマ , で区切って並べればよい

```
> log(10, base=10) # 底を指定する引数 base に 10 を指定  
[1] 1
```

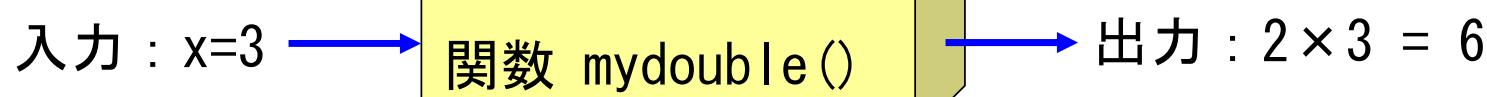


関数を定義する手順

1. 関数名を決める
2. 入力する変数の個数と種類を指定する
3. 計算式を 1 行ずつ記述し、最後に関数 `return()` で結果を出力する

【例】入力した値を 2 倍したものが output される関数 `mydouble(x)`

```
> mydouble <- function(x) {          #
+   y <- 2*x                         #
+   return(y)                          # 関数定義部分
+ }
> mydouble(3)                        # 関数を実行する
[1] 6
```



関数を定義する手順

1. 関数名を決める
2. 入力する変数の個数と種類を指定する
3. 計算式を 1 行ずつ記述し、最後に関数 `return()` で結果を出力する

【例】引数が無い関数 `myone()`

```
> myone <- function() {  
+   x <- 1  
+   return(x)  
+ }  
> myone()  
[1] 1
```

関数を定義する手順

1. 関数名を決める
2. 入力する変数の個数と種類を指定する
3. 計算式を 1 行ずつ記述し、最後に関数 `return()` で結果を出力する

【例】入力した 2 つの数値の積が出力される関数 `myprod(x, y)`

```
> myprod <- function(x, y) {  
+   return(x*y)  
+ }  
> myprod(2, 5)  
[1] 10
```

関数を定義する手順

1. 関数名を決める
2. 入力する変数の個数と種類を指定する
3. 計算式を 1 行ずつ記述し、最後に関数 `return()` で結果を出力する

【例】返り値が 2 つある関数 `myrev(x, y)` → **返したい結果をベクトルにすればよい**

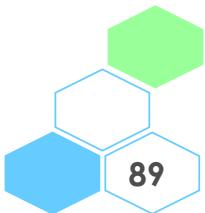
```
> myrev <- function(a, b) {  
+   x <- a  
+   y <- b  
+   z <- c(y, x)  
+   return(z)  
+ }  
> myrev(1, 2)  
[1] 2 1
```

演習 4

1. 引数が 1 つある関数を定義する練習として、入力した数値を 2 乗したものを結果として返す関数 myfunc1() を定義して下さい
2. 引数が 2 つある関数を定義する練習として、「引数 1」を「引数 2」乗したもの(例えば 43)を結果として返す関数 myfunc2() を定義して下さい
3. 返り値が 2 つある関数を定義する練習として、引数 1 を引数 2 で割り算したときの商と余りを結果として返す関数 myfunc3() を定義して下さい

プログラミングについて

- プログラミングとは
 - 人間がコンピュータに命令をすること
 - R の場合「ユーザーが R のコマンドをひとつひとつ記述する」作業のこと
⇒ 大抵は関数を定義する
- R におけるプログラミングのための道具は…
 - 変数、ベクトル、関数定義
 - 条件分岐 (if)
 - くり返し (for)





比較演算子と論理演算子

- 値同士の大小比較を行った結果は TRUE(真)または FALSE(偽)を要素とする論理型ベクトルとなる

```
> 1 < 2      # 1 は 2 よりも小さい?  
[1] TRUE  
> 3 < 2      # 3 は 2 よりも小さい?  
[1] FALSE
```

- 大小比較を行うための「比較演算子」を紹介する

記号

`==`

`!=`

`>=`

`>`

`<=`

`<`

意味

等しい

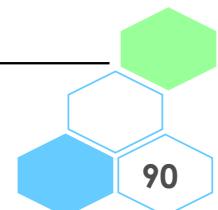
\neq

\geq

$>$

\leq

$<$



比較演算子と論理演算子

- 複数の比較結果を同時に考慮するための「論理演算子」を紹介する

記号	!	&&	
意味	否定	かつ	または

- 「比較演算子」と「論理演算子」の使用例

```
> x <- 2
> x >= 3      # 変数 x は 3 以上?
[1] FALSE
> x == 2       # 変数 x は 2 と等しい?
[1] TRUE
> 1 < x && x < 3 # 変数 x は 1 より大きく 3 より小さい?
[1] TRUE
```

おまけ: 1つの値の比較 vs. ベクトル同士の比較

- 1つの値同士の比較を行う場合、「かつ」と「又は」は「`&&`」と「`||`」を用いる
- ベクトル同士の比較を行う場合、「かつ」と「又は」は「`&`」と「`|`」を用いる
⇒ よって、データフレームにおける比較は「`&`」と「`|`」を用いる
- 以下に例を示す

```
> (1 > 2) || (3 < 4)      # 1つの値同士の比較
[1] TRUE
> c(1, 1) > c(2, 0)      # ベクトル同士の比較
[1] FALSE TRUE
> (c(1,1) > c(2,2)) || (c(3,3) < c(3,4)) # 間違い
Error in ... : 'length = 2' in coercion to 'logical(1)'
> (c(1,1) > c(2,2)) | (c(3,3) < c(3,4)) # 正しい !
[1] FALSE TRUE
> (c(1,1) > c(2,2)) & (c(3,3) < c(3,4)) # 正しい !
[1] FALSE FALSE
```

条件分岐[if]

- ある「条件」に合致する場合は処理を実行 ⇒ if を用いる

```
if (条件) { 条件に合致する場合に実行する処理 }
```

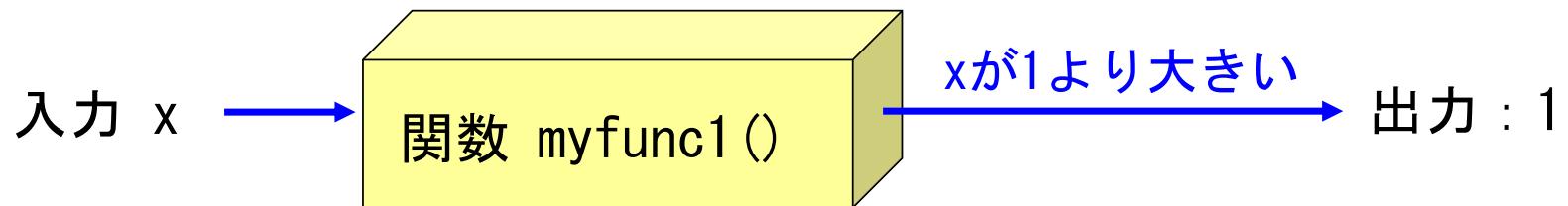
- 「条件」は「比較演算子」を使って判定する

```
> x <- -2  
> if (x < 0) x <- -x # x が 0 未満の場合は正に  
> if (x < 0) {           # {} でくくることも可  
+   x <- -x  
+
```

[if]の使用例

- 引数 x が 1 より大きい場合は 1 を出力する

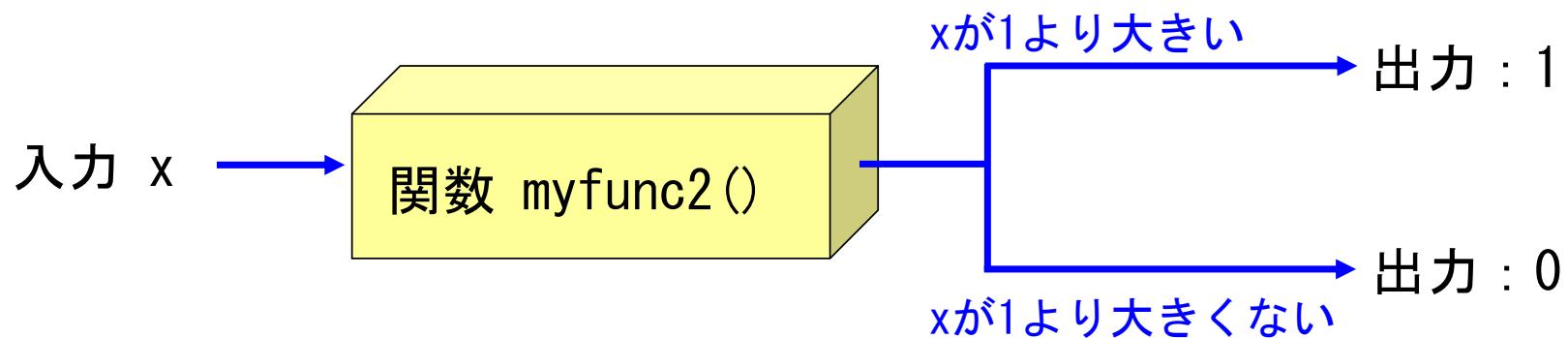
```
> myfunc1 <- function(x) {  
+   if (x > 1) return(1)  # 引数 x が 1 よりも大きい場合は 1  
+ }  
> myfunc1(2)  
[1] 1  
> myfunc1(0)          # 何も起こらない
```



[if + else] の使用例

- 引数 x が 1 より大きい場合は 1 を出力し、
引数 x が 1 より大きくない場合は 0 を出力する

```
> myfunc2 <- function(x) {  
+   if (x > 1) return(1)    # 引数 x が 1 よりも大きい場合は 1  
+   else                 return(0)    # そうでない場合は 0  
+ }  
> myfunc2(0)  
[1] 0
```



[if + else if + else] の使用例

- 引数 x が 1 より大きい場合は 1 を出力し、
引数 x が 0 ~ 1 の場合は 0 を出力し、
引数 x が 0 より小さい場合は -1 を出力する

```
> myfunc3 <- function(x) {  
+   if      (x >  1) return( 1)    # 引数 x が 1 よりも大きい  
+   else if (x >= 0) return( 0)    # 引数 x が 0 ~ 1  
+   else           return(-1)    # 上 2 つのどれでもない  
+ }  
> myfunc3(2)  
[1] 1  
> myfunc3(0)  
[1] 0  
> myfunc3(-1)  
[1] -1
```

演習 5

1. 変数 x に 2 を、変数 y に 3 を代入して下さい。
2. 以下の条件式を作成し、TRUE(真)か FALSE(偽)かを判定して下さい
 - x は 1 よりも小さい？
 - y は 3 未満？
 - x は y 以下？
3. 以下の条件式を作成し、TRUE(真)か FALSE(偽)かを判定して下さい
 - 「 x は 1 よりも小さい？」かつ「 x は y 以下？」
 - 「 x は 1 よりも小さい？」または「 x は y 以下？」
 - 「 x は y 以下？」でない
4. 引数 x が 1 と等しい場合は 1 を出力し、引数 x が 1 と等しくない場合は何もしない
関数 `myone()` を定義して下さい
5. 引数 x が 1 と等しい場合は 1 を出力し、引数 x が 1 と等しくない場合は 0 を出力
する関数 `myindex()` を定義して下さい
6. 引数 x と引数 y の比較を行い、 x と y の値が等しい場合は x の値を返し、
等しくない場合は x と y の値の大きい方を返す関数 `mymax()` を定義して下さい

おまけ①: 関数 *print()* と関数 *cat()*

- 関数の最後は必ず関数 `return()` で何らかの値を返していたが、必ずしも関数 `return()` を実行する必要はない
→ 例えば、関数 `print()` と関数 `cat()` で結果を返しても良い
⇒ 関数 `cat()` で “`\n`”(改行文字)を出力することが出来る

```
> myfunc4 <- function(x) {  
+   y <- x  
+   print(y)  # 変数 y の値を表示する  
+   cat(y)  
+   cat("\n") # cat() では明示的に改行文字が必要  
+   cat("入力された値は ")  
+   cat(y)  
+   cat(" です. \n")  
+ }  
> myfunc4(777)  
[1] 777  
777  
入力された値は 777 です
```

おまけ②: 複合文 { }

- 次のプログラムの 2 ~ 3 行目について、if と else を括弧なしで複数行に分けるとエラーとなる
- また、7 ~ 8 行目について } と else の間で改行するとエラーとなる
⇒ if (条件式){ 式 } の時点で「一連の式が終わった」と解釈されるので、その後に else が現れると「if がないのに else がきた」となる

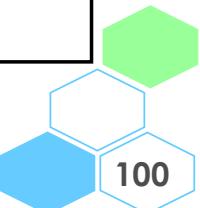
```
> a <- -777
> if (a < 0) b <- -a
> else      b <- a      # この後エラーが出る
Error: unexpected 'else' in "else"

> if (a < 0) {
+   b <- -a
+
> else b <- -a          # この後エラーが出る
Error: unexpected 'else' in "else"
```

おまけ②: 複合文 { }

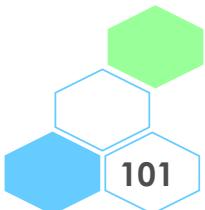
- よって、「一連の式はここからここまで」ということを R に教えるためには、中括弧 {} で囲んで複合式とするか、if 文の後～改行するまでに else 文を書く必要がある
⇒ 関数定義は {} でくくられるので、関数作成時は気にする必要はない

```
> a <- -777
> {
+   if (a < 0) b <- -a
+   else       b <- a    # エラーは出ない
+   if (a < 0) {
+     b <- -a
+   }
+   else b <- a          # エラーは出ない
+ }
> b
[1] 777
```



プログラミングについて

- プログラミングとは
 - 人間がコンピュータに命令をすること
 - R の場合「ユーザーが R のコマンドをひとつひとつ記述する」作業のこと
⇒ 大抵は関数を定義する
- R におけるプログラミングのための道具は…
 - 変数、ベクトル、関数定義
 - 条件分岐(if)
 - くり返し(for)



くり返し[for]

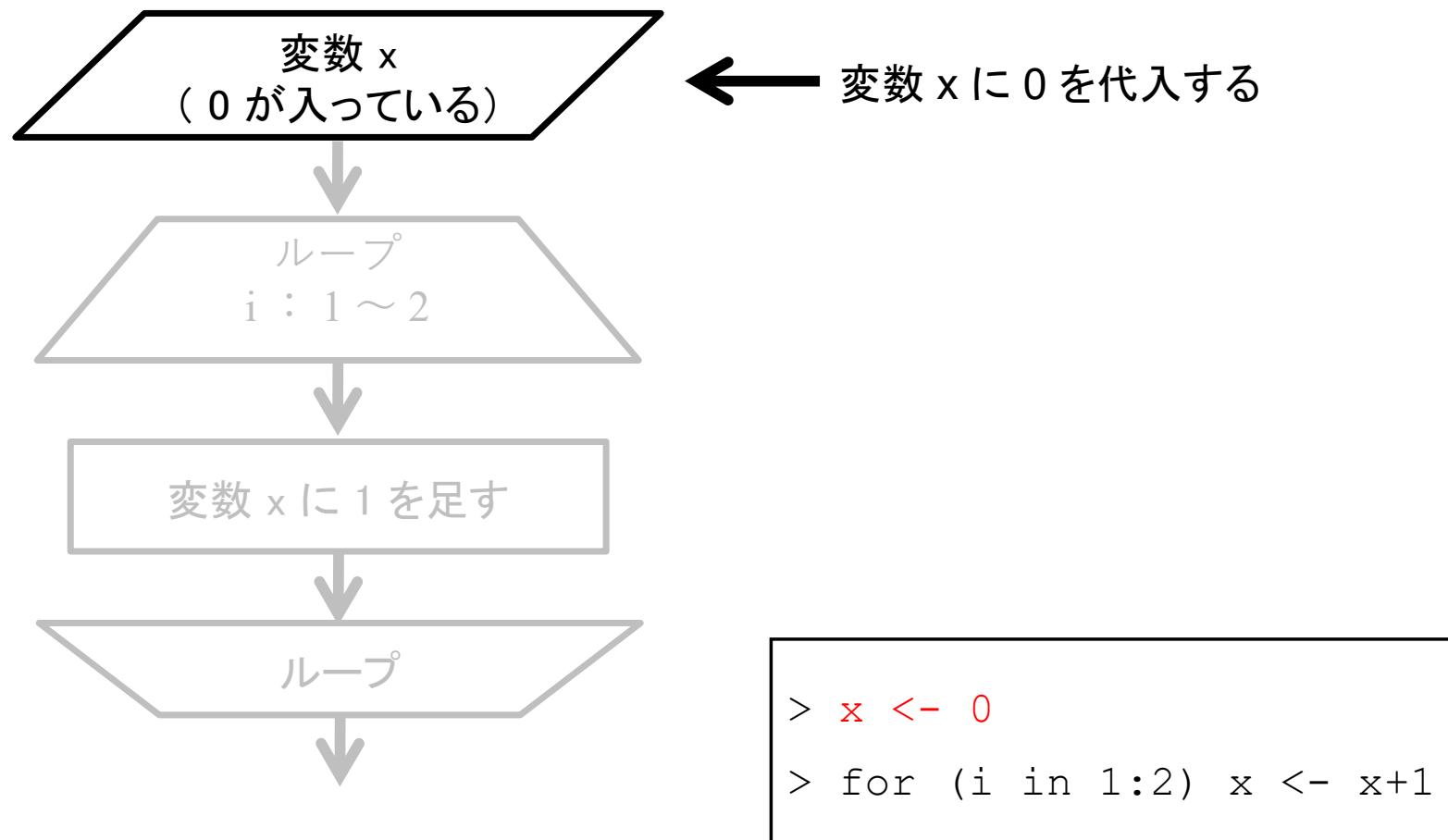
- ある「処理」をくり返し実行する ⇒ for を用いる

```
for (i in 1:くり返し数) { くり返し実行する処理 }
```

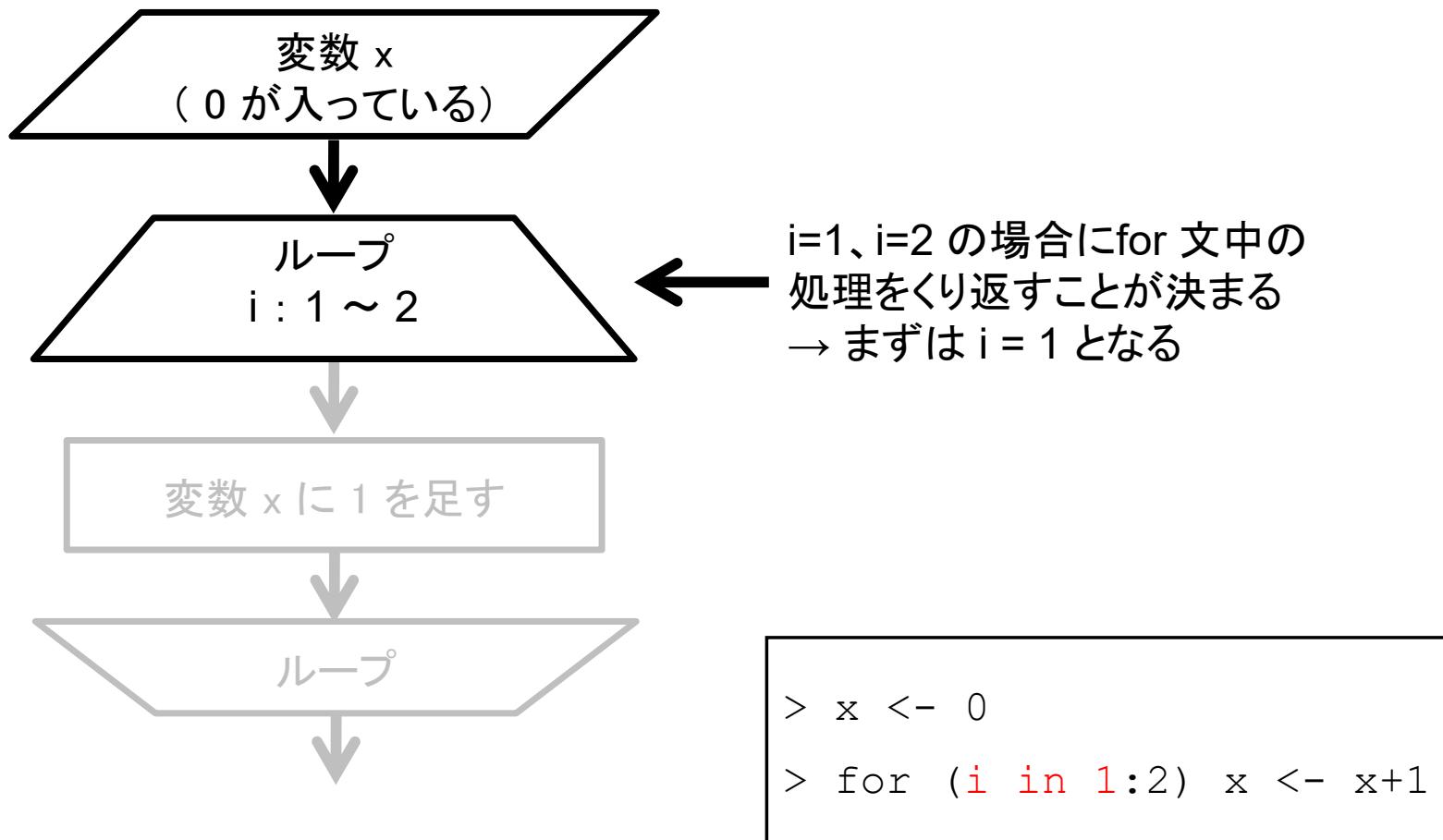
- 「処理」として「変数 x に 1 を足す」を 2 回くり返す例を挙げる

```
> x <- 0                      # x に 0 を代入  
> for (i in 1:2) x <- x+1    # 「x に 1 を足す」を  
                                # 2 回くり返す  
> x                            # x の中身を確認  
[1] 2
```

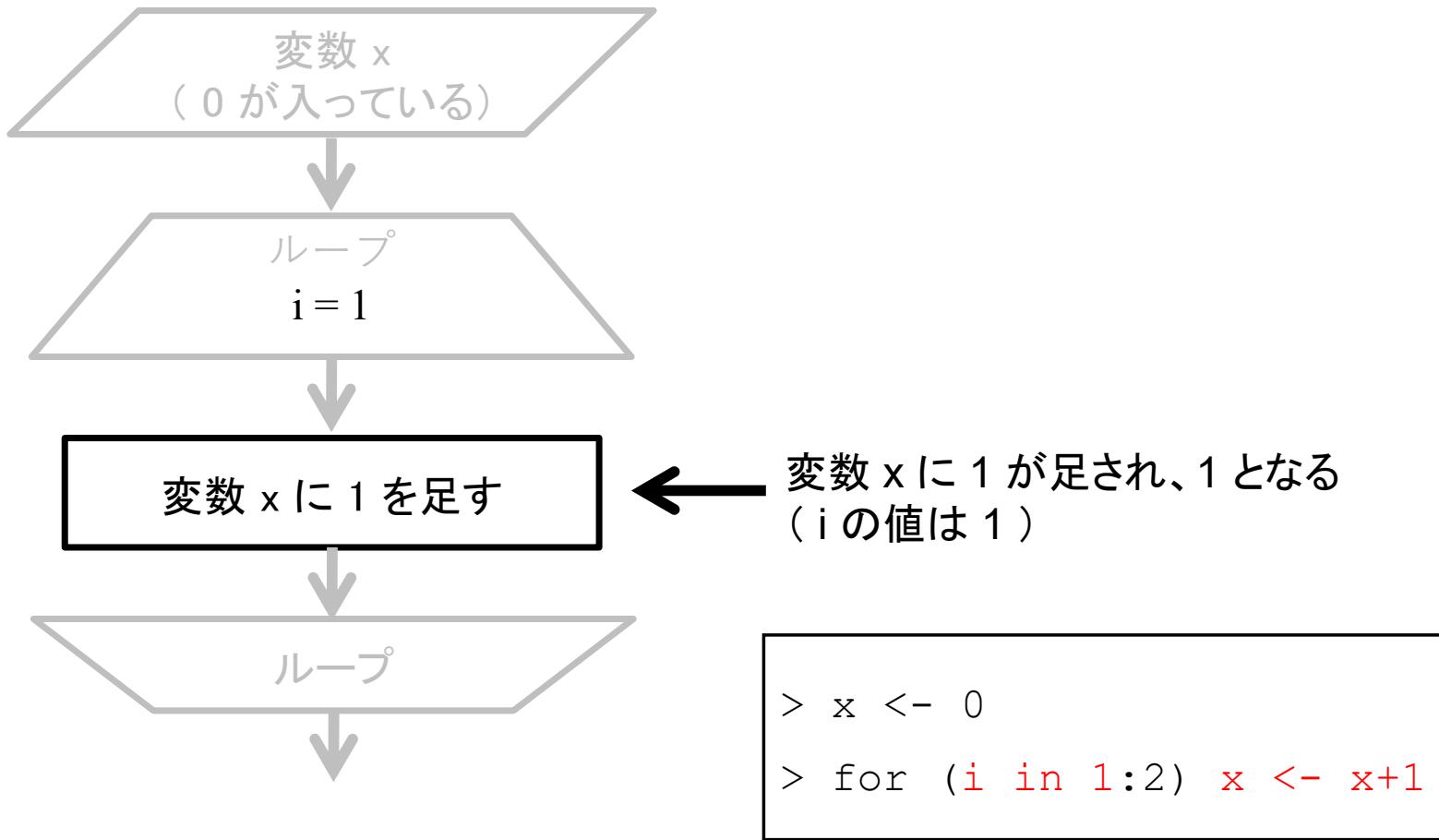
くり返し [for]



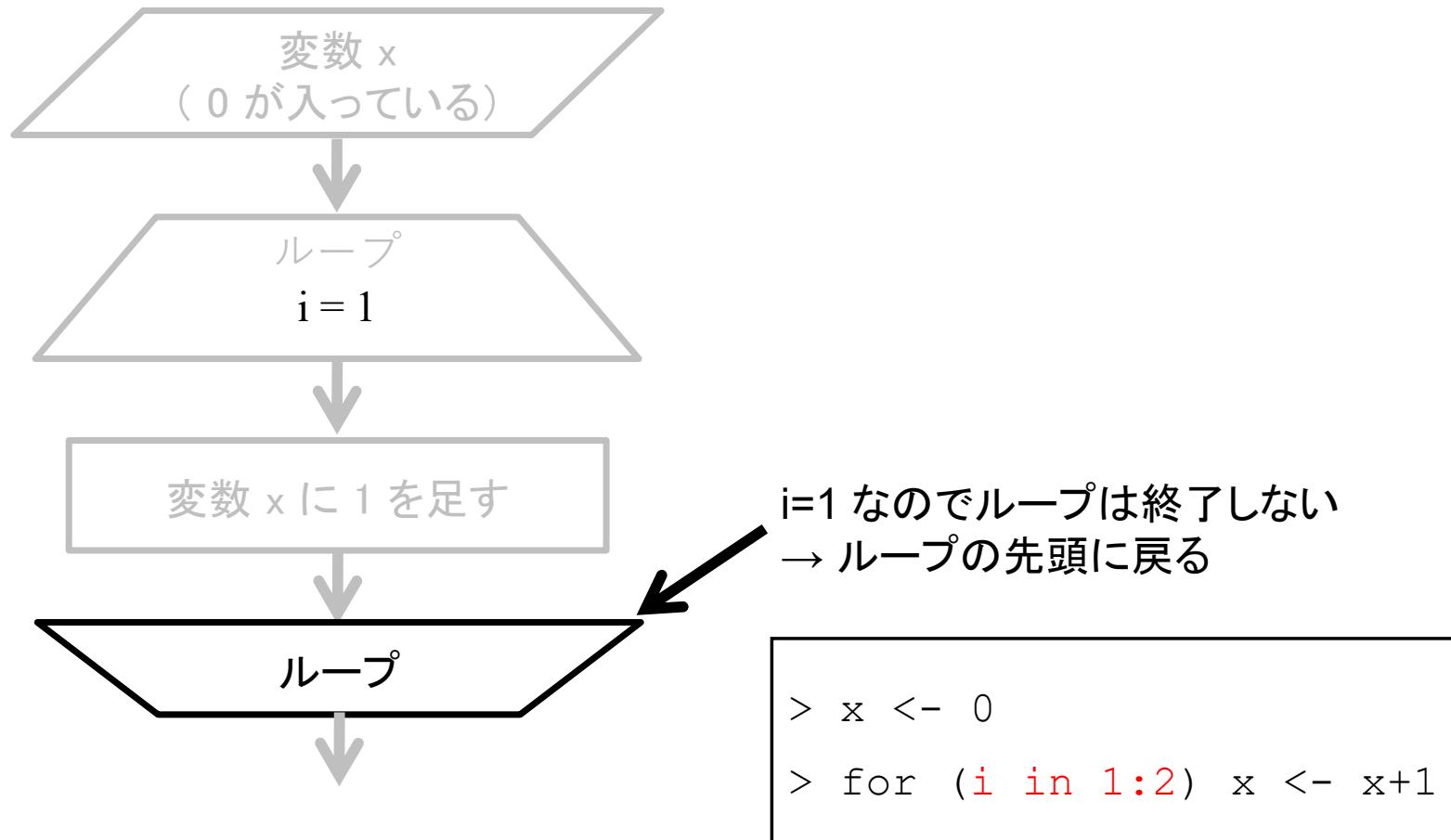
くり返し [for]



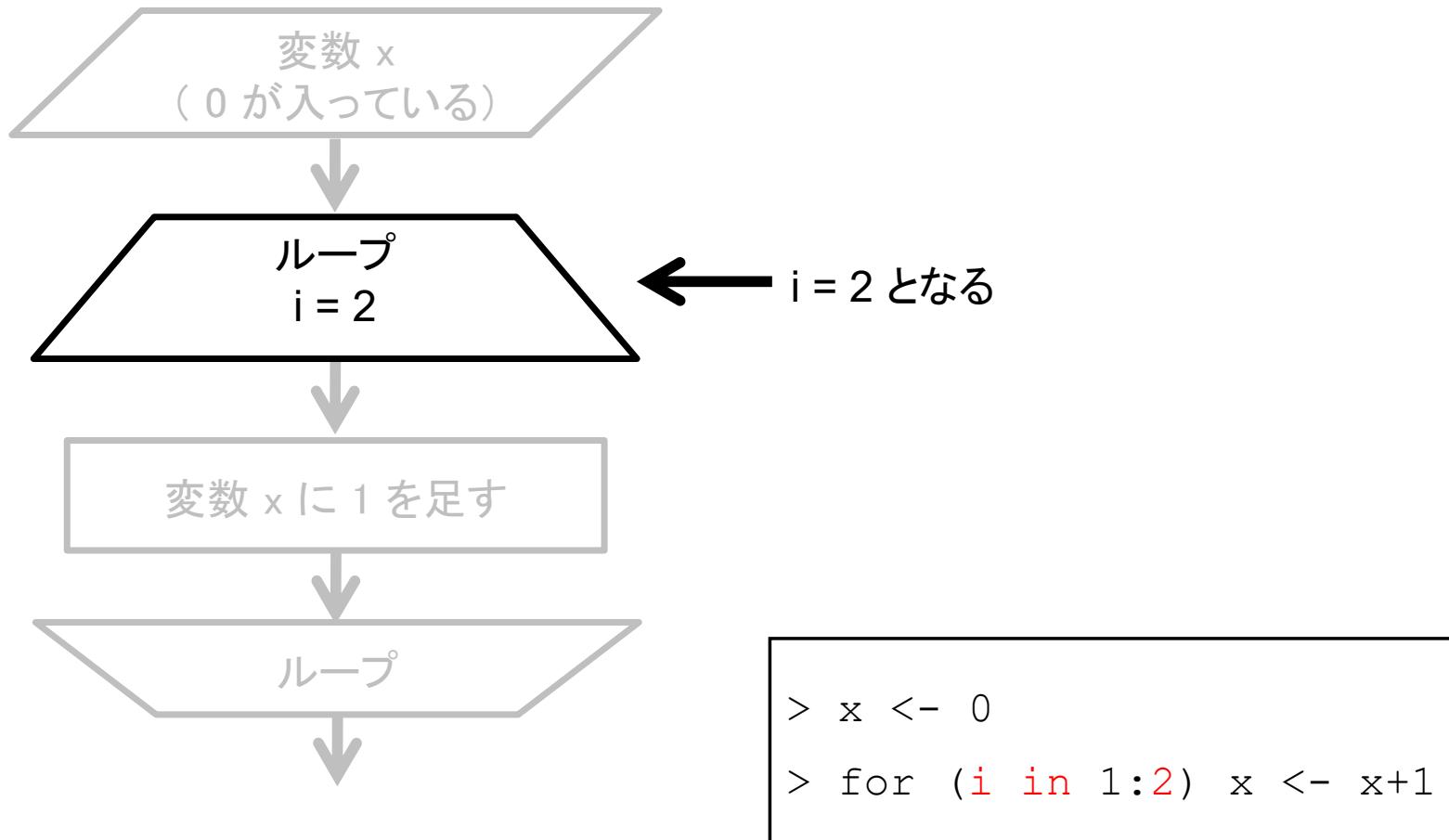
くり返し [for]



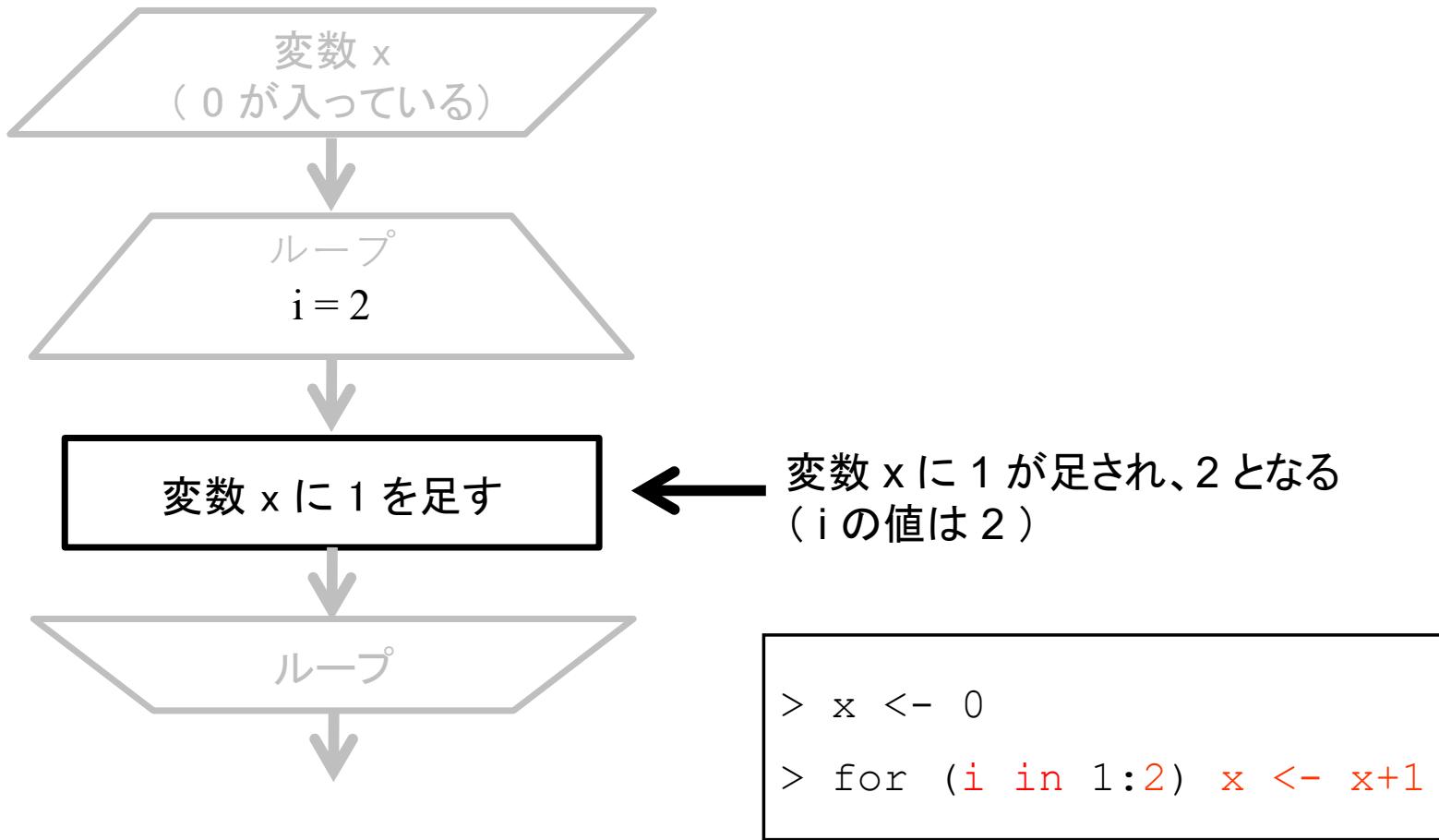
くり返し [for]



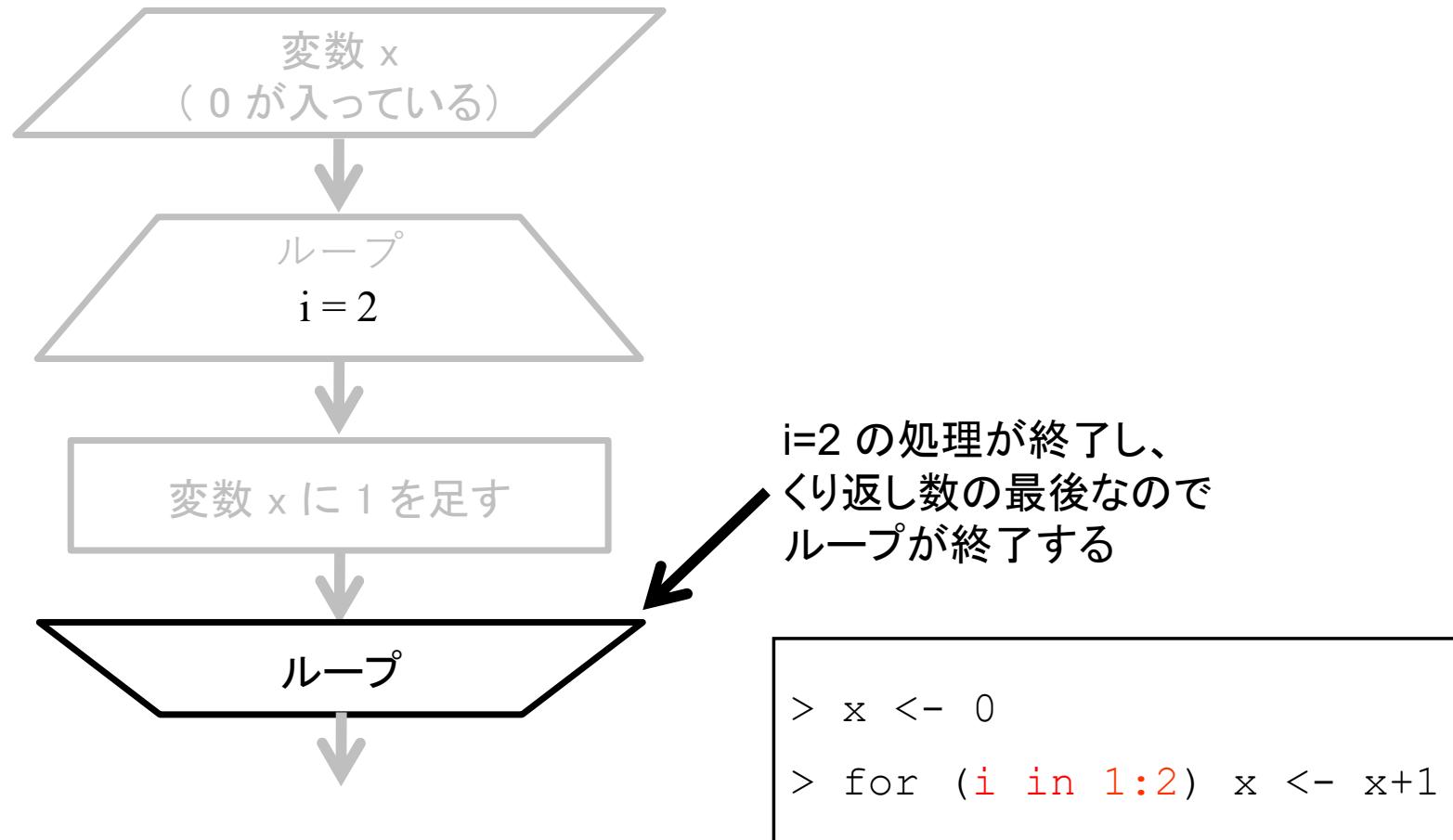
くり返し [for]



くり返し [for]



くり返し [for]



くり返し [for]

- 「変数 x に k を足す」を 5 回くり返す

```
> x <- 0                      # x に 0 を代入  
> for (k in 1:5) x <- x+k    # x に k を足す  
> x                            # x を表示  
[1] 15
```

- 「ベクトル x に i をくっつける」を 5 回くり返す

```
> x <- c()                    # 空のベクトルを用意  
> for (i in 1:5) x <- c(x, i) # x に i をくっつける  
> x                            # x を表示  
[1] 1 2 3 4 5
```

[for + if] の使用例

- 「1からxまでの間の偶数を全て足し合わせる」という関数 myeven() を定義する

```
> myeven <- function(x) {  
+   a <- 0                                # a に 0 を代入  
+   for (i in 1:x) {  
+     if (i%%2 == 0) a <- a+i    # i を 2 で割った余り  
+   }                                         # が 0 なら i を足す  
+   return(a)  
+ }  
> myeven(10)                               # 1～10 の偶数の和  
[1] 30
```

演習 6

1. 正の整数値 n を入れると 1 から n までの積($n!$)を返す関数 `myprod()` を定義して下さい
2. 余力がある方は、関数 `return()` で結果を返す前に、関数 `print()` や関数 `cat()` と、適当な関数を組み合わせることにより、以下の様な出力となるよう関数 `myprod()` を改変して下さい

```
> myprod(5)
[1] "1から5までの積:"
[1] 120
```

おまけ: くり返し[*while*]

- *while* 文は、ある条件が成り立っている間(TRUE の間)はずっと処理をくり返す
- 「ループ変数のリストの個数だけ処理を行うとくり返しが終了する」*for* 文と異なり、「条件式が TRUE しかとり得ない」場合はプログラムが永遠に実行され続けるので注意
- もし暴走した場合は、実行結果が返ってこず、Windows の砂時計のアイコンがクルクル回ります
→ その時は [Esc] キーを押すことで処理を止めることができます

```
while ( 条件式 ) { くり返し実行する処理 }
```

- 「変数 *x* に 1 を足す」を 2 回くり返す例を挙げる

```
> x <- 0
> i <- 1          # くり返しを制御するための変数
> while (i <= 2) {
+   x <- x + 1
+   i <- i + 1    # くり返し回数をカウント
+ }
> x
[1] 2
```

おまけ: while 文の学習 ≈ Google Blockly Games で遊ぶ

<https://blockly.games/maze?lang=ja>

Blockly Games : 迷路 6 / 10

日本語 GO キャラクター選択

まっすぐ進む

左を向くひ▼

右を向くひ▼

「まで繰り返す」実行

もし左に進めるならひ▼ 実行

「まで繰り返す」実行

まっすぐ進む

もし左に進めるならひ▼ 実行

▶ プログラムを実行

本日のメニュー

- Windows 版 R / RStudio のセットアップ方法 ← 当日までに自習
- R の基礎知識 ← 当日までに自習
- パッケージと作業ディレクトリ ← 当日までに自習
- 変数とベクトル
- ベクトルの型
- 関数とプログラミング
- データ加工とパイプ演算子
- グラフ作成(ggplot2)
- その他

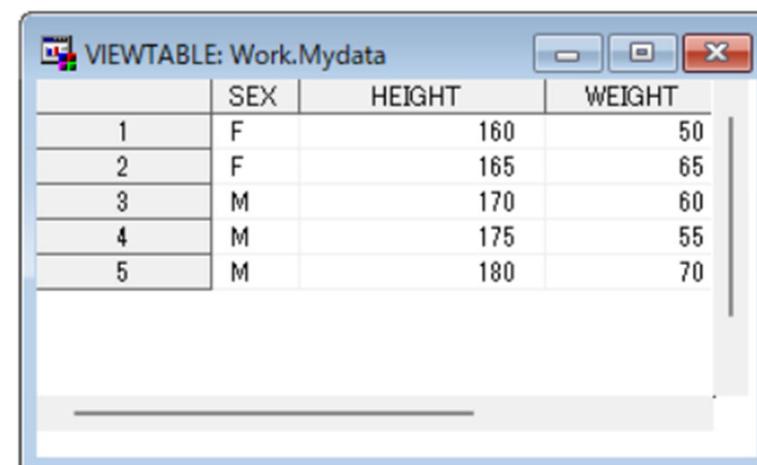
データフレームとは

- データ解析を行うデータの形式は様々
 - (R上で)データを手で入力して…
 - テキストファイル、EXCEL、SAS (.sas7bdat / .xpt)などの形式
- Rでデータ解析を行う際はデータフレームという形式にデータを変換する



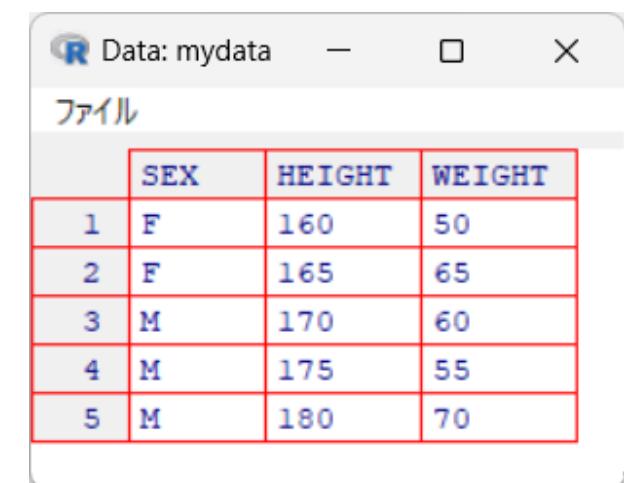
	A	B	C
1	SEX	HEIGHT	WEIGHT
2	F	160	50
3	F	165	65
4	M	170	60
5	M	175	55
6	M	180	70

EXCEL:シート



	SEX	HEIGHT	WEIGHT
1	F	160	50
2	F	165	65
3	M	170	60
4	M	175	55
5	M	180	70

SAS:データセット



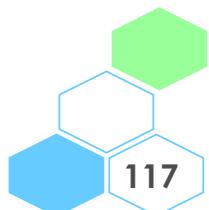
	SEX	HEIGHT	WEIGHT
1	F	160	50
2	F	165	65
3	M	170	60
4	M	175	55
5	M	180	70

R/python:データフレーム

データフレームとは

- 数値ベクトルや文字ベクトル、因子ベクトルなどの異なる型のベクトルをまとめて一つ変数
- 外見は EXCEL シートの様な表形式
- 各列の要素の型はバラバラでも可
- データフレームの各行・各列はラベルを持ち、ラベルによる操作が可能
- SAS とは異なりフォーマットを付与できない
- データ加工・データハンドリングの導入のみ紹介(パイプ演算子の紹介のため)
⇒ 詳細は他の方の発表演題にて

SEX	HEIGHT	WEIGHT
F	160	50
F	165	65
M	170	60
M	175	55
M	180	70



データフレームの作成

- R でベクトルデータを作成した後、データフレームを作成(いわゆる手入力)
 - 「性別」「身長」「体重」データをベクトルで用意した後、関数 [data.frame\(\)](#) や [tibble\(\)](#) で 1 つのデータフレームに変換する
 - 関数 [read.table\(\)](#) でデータをベタ打ちする
- ファイルからデータを読み込んで、データフレームを作成
 - パッケージ `readr` の関数 [read_csv\(\)](#) で CSV ファイルを読み込む
 - パッケージ `readxl` の関数 [read_excel\(\)](#) で EXCEL ファイルを読み込む
 - パッケージ `haven` の関数 [read_xpt\(\)](#) や [read_sas\(\)](#) でSAS ファイルを読み込む

データフレームの作成(手入力)

SEX	HEIGHT	WEIGHT
F	160	50
F	165	65
M	170	60
M	175	55
M	180	70

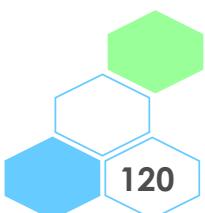
data.frame()

SEX	HEIGHT	WEIGHT
F	160	50
F	165	65
M	170	60
M	175	55
M	180	70

```
> sex      <- c("F", "F", "M", "M", "M")
> height   <- c(160, 165, 170, 175, 180)
> weight   <- c( 50,  65,  60,  55,  70)
> mydata <- data.frame(SEX=sex, HEIGHT=height, WEIGHT=weight)
```

データフレームの作成(手入力)

```
> mydata <- read.table(text='  
+ SEX HEIGHT WEIGHT  
+ F 160 50  
+ F 165 65  
+ M 170 60  
+ M 175 55  
+ M 180 70 header=TRUE)
```





外部データの読み込み

```
library(readr); library(readxl); library(haven)
```

- tidyverse の以下のパッケージにより各種データファイルからの読み込みが可能
 - <https://rstudio.github.io/cheatsheets/html/data-import.html>
 - https://haven.tidyverse.org/reference/read_sas.html

データ形式	関数	使用例
CSV	<code>read.csv()</code> <code>read_csv()</code>	<code>read.csv("c:/temp/mydata.csv", fileEncoding="shift_jis"※ or "utf8")</code> <code>read_csv("c:/temp/mydata.csv", locale=locale(encoding="shift-jis" or "utf8"),</code> <code>col_types=cols(変数1=col_double(), 変数2=col_character(), ...)</code> <code>⇒ 出力する場合は write_csv(mydata, "c:/temp/mydata.csv")</code>
タブ区切り	<code>read.delim()</code> <code>read_delim()</code>	<code>read.delim("c:/temp/mydata.txt")</code> <code>read_delim("c:/temp/mydata.txt", delim="¥t")</code>
EXCEL	<code>read_excel()</code>	<code>read_excel("c:/temp/mydata.xlsx", sheet="tg")</code>
SAS	<code>read_sas()</code> <code>read_xpt()</code>	<code>read_sas("c:/temp/mydata.sas7bdat")</code> <code>read_xpt("c:/temp/mydata.xpt") ⇒ 出力: write_xpt(mydata, "xx.xpt", version=5 or 8)</code> <code>※ CP932 とも言う、エンコーディングは guess_encoding("ファイル名") で確認可</code>

EXCEL のセルをコピーして作成

- EXCEL のセルをコピーして、そのまま R に貼り付けることも出来る

	A	B	C
1	SEX	HEIGHT	WEIGHT
2	F	160	50
3	F	165	65
4	M	170	60
5	M	175	55
6	M	180	70

コピー

列名をコピーした場合

```
x <- read.delim(pipe("pbpaste"), header=T)
```

列名をコピーしなかった場合

```
x <- read.delim(pipe("pbpaste"), header=F)
```

MacOS の場合

列名をコピーした場合

```
x <- read.delim("clipboard", header=T)
```

列名をコピーしなかった場合

```
x <- read.delim("clipboard", header=F)
```

Windowsの場合

データフレームの閲覧

- データフレームの中身を確認したいときは、データフレーム名を入力するか、
- 右上の画面の [Environment] → データフレーム名をクリック → ソートや条件抽出も出来る

データフレームの最大表示列数を確認

```
> rstudioapi::readRStudioPreference("data_viewer_max_columns", TRUE)  
[1] 50
```

データフレームの最大表示列数を300列に変更

```
> rstudioapi::writeRStudioPreference("data_viewer_max_columns", 300L)
```

The screenshot shows the RStudio interface. On the left, there is a data viewer window displaying a table with columns SEX, HEIGHT, and WEIGHT, containing 5 rows of data. A yellow arrow points from this viewer towards the top right. On the right side, the Environment tab is selected in the top navigation bar. Below it, the Global Environment pane shows a data object named 'x' with the description '5 obs. of 3 variables'. Underneath, the 'Values' section lists the three variables: height, sex, and weight, each with its corresponding type and first few values.

	SEX	HEIGHT	WEIGHT
1	F	160	50
2	F	165	65
3	M	170	60
4	M	175	55
5	M	180	70

Environment History Connections Tutorial

x 5 obs. of 3 variables

height num [1:5] 160 165 170 175 180
sex chr [1:5] "F" "F" "M" "M" "M"
weight num [1:5] 50 65 60 55 70

おまけ: 欠測・欠損の扱い

- 手入力でデータフレームを作成する場合で欠測が含まれているデータを読み込む場合は、ベクトル中の欠測部分を NA としておけば、該当部分に欠測値(NA)が入る
- SAS データでは、欠測文字としてピリオドが使われるため、関数によっては欠測文字を指定(例えば引数 na.strings=". " を指定)する必要がある

```
> sex      <- c("F", NA, "M"); height <- c(158, 162, NA)
> weight <- c(51, 55, 72)
> ( x <- data.frame(SEX=sex, HEIGHT=height, WEIGHT=weight) )

  SEX HEIGHT WEIGHT
1   F     158      51
2 <NA>    162      55
3   M      NA      72

> is.na(x$HEIGHT)    # 身長 (HEIGHT) が欠測かどうか
[1] FALSE FALSE  TRUE
```

パッケージ *tidyverse* (実質は *dplyr* / *tidyr*) のデータ加工用関数

関数	機能
<code>filter(x, SEX=="F" & HEIGHT>160)</code>	「SEX=="F" & HEIGHT>160」を満たすレコードを抽出
<code>arrange(x, SEX, HEIGHT)</code>	変数 SEX、HEIGHT の小さい順に並べ替え
<code>arrange(x, SEX, desc(HEIGHT))</code>	変数 SEX の小さい順&変数 HEIGHT の大きい順に並べ替え
<code>distinct(x, SEX, HEIGHT, .keep_all=TRUE)</code>	変数 SEX、HEIGHT をキーとして、重複行を削除
<code>select(x, ID, SEX)</code>	変数 ID、SEX のみ抽出 → -変数名 で変数を削除
<code>select(x, where(is.character))</code>	文字列変数のみ抽出、where(関数)で関数の返り値が TRUE の列を抽出
<code>relocate(x, SEX), relocate(y, HEIGHT, .before=SEX)</code>	変数 SEX を 1 列目に移動、変数 HEIGHT を変数 SEX の前に移動
<code>mutate(x, M=HEIGHT/100), mutate(x, M=9, .after=SEX)</code>	新しい変数 M の追加、新しい変数 M を変数 SEX の後に追加
<code>transmute(x, M=HEIGHT/100)</code>	新しい変数の追加(新しい変数のみ返す)
<code>rename(x, H=HEIGHT)</code>	変数 HEIGHT の名前を H に変更
<code>summarise(x, mean(HEIGHT)), count(x, SEX)</code>	変数 HEIGHT の平均値を算出、変数 SEX の頻度を集計
<code>summarise(group_by(x, SEX), N=n(), M=mean(HEIGHT))</code>	変数 SEX のカテゴリごとに、レコード数と変数 HEIGHT の平均値を算出
<code>inner_join(), left_join(), right_join(), full_join()</code>	データフレームをある変数をキーとして結合
<code>bind_rows(x, y)</code>	データフレームを縦結合
<code>bind_cols(x[1:4,], y)</code>	データフレームを横結合
<code>tidyr::pivot_wider() / tidyr::pivot_longer()</code>	SAS での転置に相当

※ x はデータフレーム、「かつ」と「または」は、それぞれ **&** と **|** である点に注意

データの整列(ソート)

```
> library(tidyverse)

> arrange(mydata, HEIGHT)          # HEIGHTの小さい順に整列
  SEX HEIGHT WEIGHT
1   F     160     50
2   F     165     65
3   M     170     60
4   M     175     55
5   M     180     70

> arrange(mydata, desc(SEX), HEIGHT) # SEXの大きい順, HEIGHTの小さい順に整列
  SEX HEIGHT WEIGHT
1   M     170     60
2   M     175     55
3   M     180     70
4   F     160     50
5   F     165     65
```

データの条件抽出、重複行の削除

```
> filter(mydata, SEX=="F")          # SEX=="F" を満たすレコード  
SEX HEIGHT WEIGHT  
1   F     160      50  
2   F     165      65  
  
> filter(mydata, SEX=="F" & HEIGHT>160) # SEX=="F" & HEIGHT>160 を満たすレコード  
SEX HEIGHT WEIGHT  
1   F     165      65  
  
> distinct(mydata, SEX, .keep_all=TRUE) # 変数SEXをキーとして重複行を削除  
SEX HEIGHT WEIGHT  
1   F     160      50  
2   M     170      60
```

データの条件抽出 ⇒ 内部ではベクトル(同士)の比較

⇒ 92 頁をご覧下さい



```
> filter(mydata, SEX=="F") # SEX=="F" を満たすレコード
```

SEX	HEIGHT	WEIGHT	
F	160	50	TRUE
F	165	65	TRUE
M	170	60	FALSE
M	175	55	FALSE
M	180	70	FALSE



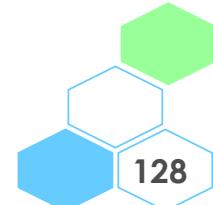
SEX	HEIGHT	WEIGHT
F	160	50
F	165	65

```
> filter(mydata, SEX=="F" & HEIGHT>160) # SEX=="F" & HEIGHT>160 を満たすレコード
```

SEX	HEIGHT	WEIGHT	
F	160	50	FALSE
F	165	65	TRUE
M	170	60	FALSE
M	175	55	FALSE
M	180	70	FALSE



SEX	HEIGHT	WEIGHT
F	165	65



データの変数選択

```
> select(mydata, SEX, HEIGHT) # 変数SEXとHEIGHTを抽出
```

	SEX	HEIGHT
1	F	160
2	F	165
3	M	170
4	M	175
5	M	180

```
> select(mydata, -WEIGHT) # 変数WEIGHTを削除
```

	SEX	HEIGHT
1	F	160
2	F	165
3	M	170
4	M	175
5	M	180

データの要約統計量

```
> tmp <- filter(mydata, SEX=="F")          # SEX=="F" を満たすレコードについて
> summary(tmp)                           # 要約統計量
   SEX                      HEIGHT          WEIGHT
Length:2           Min.    :160.0      Min.    :50.00
Class :character  1st Qu.:161.2      1st Qu.:53.75
Mode  :character  Median  :162.5      Median  :57.50
                  Mean    :162.5      Mean    :57.50
                  3rd Qu.:163.8      3rd Qu.:61.25
                  Max.    :165.0      Max.    :65.00

# SEXのカテゴリごとに要約統計量
> summarise( group_by(mydata, SEX), N=n(), M=mean(HEIGHT), SD=sd(HEIGHT) )
# A tibble: 2 × 4
  SEX     N     M     SD
  <chr> <int> <dbl> <dbl>
1 F       2    162.  3.54
2 M       3    175   5
```

演習 7

1. 作業ディレクトリを「C:¥temp」に変更し、パッケージ tidyverse を呼び出してください
2. 以下によりデータフレーム mydata を作成してください

```
> sex      <- c("F", "F", "M", "M", "M")
> height   <- c(160, 165, 170, 175, 180)
> weight   <- c( 50,  65,  60,  55,  70)
> ( mydata <- data.frame(SEX=sex, HEIGHT=height, WEIGHT=weight) )
```

3. 以下によりデータフレーム mydata を CSV に出力 ⇒ 再び読み込んでください

```
> write_csv(mydata, "c:/temp/mydata.csv")
> mydata <- read_csv("c:/temp/mydata.csv")
```

4. データフレーム mydata について以下の操作を行って下さい

- ① 関数 filter() を用いて、SEX="M" を満たすレコードに絞り、データフレーム tmp に代入
- ② 関数 select() を用いて、変数 HEIGHT, WEIGHT に絞り、データフレーム tmp に代入
- ③ データフレーム tmp について、関数 summary() を用いて要約統計量を算出

おまけ: データを R のオブジェクトのまま保存

- CSV で出力すると、いろいろ属性が消えてしまうので、データを R のオブジェクト (.rds) のまま保存することが出来る

```
> write_rds(mydata, "c:/temp/mydata.rds")
> ( mydata <- read_rds("c:/temp/mydata.rds") )
   sex height weight
1   F     160     50
2   F     165     65
3   M     170     60
4   M     175     55
5   M     180     70
```

パイプ演算子: |> の左にあるデータを、右の関数の第 1 引数に渡す

- パイプ演算子 |> を用いることで入れ子等を防ぐことが出来る
 - 左 |> 右 → 左が右の第 1 引数になる

```
> tmp <- filter(mydata, SEX=="F")          # SEX=="F" を満たすレコードについて
> summary(tmp)                            # 要約統計量
```



```
> mydata |>
+   filter(SEX=="F") |>
+   summary()
    SEX           HEIGHT        WEIGHT
  Length:2       Min.   :160.0   Min.   :50.00
  Class :character 1st Qu.:161.2   1st Qu.:53.75
  Mode  :character   Median :162.5   Median :57.50
                           Mean   :162.5   Mean   :57.50
                           3rd Qu.:163.8   3rd Qu.:61.25
                           Max.   :165.0   Max.   :65.00
```

パイプ演算子: |> の左にあるデータを、右の関数の第 1 引数に渡す

- 関数の第 1 引数がデータフレームを指定するようになつてないと使えない
- パッケージ tidyverse(dplyr)のデータフレームに関する関数は、ほぼ全て「第 1 引数にデータフレームを指定する」ようになつている

```
> summarize( group_by(mydata, SEX), N=n(), M=mean(HEIGHT), SD=sd(HEIGHT))
```



```
> mydata |>
+   group_by(SEX) |>
+     summarize(N =n(),
+                M =mean(HEIGHT),
+                SD=sd(HEIGHT))
# A tibble: 2 × 4
  SEX      N      M      SD
  <chr> <int> <dbl> <dbl>
1 F        2    162.   3.54
2 M        3    175    5
```

演習 8

1. データフレームにmydataについて以下の操作を**|> を使って**行って下さい
(演習7のプログラムをパイプ演算子で書き換える演習です)

- ① パッケージ tidyverse を呼び出し
- ② 関数 filter() を用いて、SEX=="M"を満たすレコードに絞り、データフレーム tmp に代入
- ③ 関数 select() を用いて、変数 HEIGHT, WEIGHT に絞り、データフレーム tmp に代入
- ④ データフレーム tmp について、関数 summary() を用いて要約統計量を算出

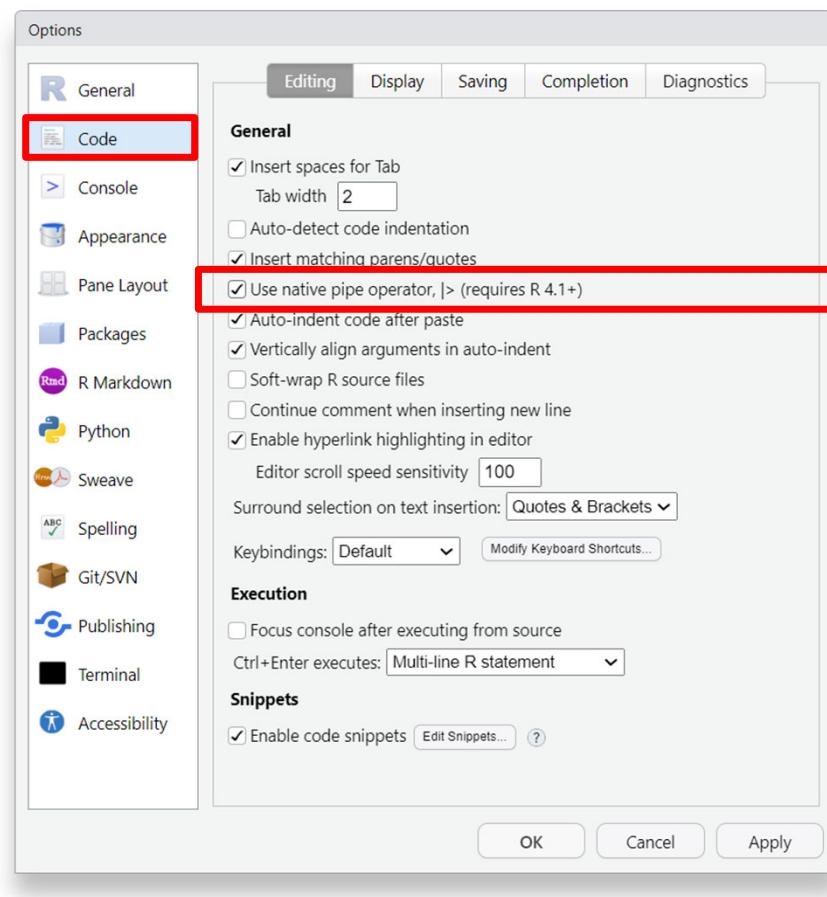
```
> library(tidyverse)

> sex      <- c("F", "F", "M", "M", "M")
> height   <- c(160, 165, 170, 175, 180)
> weight   <- c( 50,  65,  60,  55,  70)
> ( mydata <- data.frame(SEX=sex, HEIGHT=height, WEIGHT=weight) )

> tmp <- filter(mydata, SEX=="M")      # SEX=="M" を満たすレコード
> tmp <- select(tmp, HEIGHT, WEIGHT)  # 変数HEIGHT, WEIGHTに絞る
> summary(tmp)                      # 要約統計量
```

おまけ①: パイプ演算子のショートカット

- [Ctrl] + [Shift] + [m] でパイプ演算子が入力される
- 事前に [Tools] → [Global Options] → [Code] の下記チェックボックスを ON



おまけ②: パッケージ「*magrittr*」のパイプ演算子 `%>%`

- パイプ演算子は、いくつかのパッケージで定義されてきたが、少し前まではパッケージ「*magrittr*」のパイプ演算子 `%>%` が主流だった
 - 他にも、*pipeR* パッケージの `%>>%` 等、チラホラ発明されていた
 - その後、base R 4.1.0 以降で `|>` が登場、今後は `|>` が主流に？

```
> library(magrittr)

> mydata %>%
+   filter(SEX=="F") %>%
+   summary()

> mydata %>%
+   group_by(SEX) %>%
+   summarize(N =n(),
+             M =mean(HEIGHT),
+             SD=sd(HEIGHT))
```

おまけ③: 関数 *group_by()* の注意点

- 複数の変数で関数 *group_by()* を適用(グループ化)した場合、結果(データフレーム)に中途半端にグループ化情報が残ってしまう
 - ここからさらにソート等で処理した場合、グループごとに処理を行おうとする為、エラーの原因となる

```
> ToothGrowth |>
+   group_by(supp, factor(dose)) |>
+   summarize(N=n(), M=mean(len))
`summarise()` has grouped output by 'supp'. You can override using the `groups` argument.
# A tibble: 6 × 4
# Groups:   supp [2]
  supp `factor(dose)`     N     M
  <fct> <fct>       <int> <dbl>
1 OJ    0.5            10  13.2
2 OJ    0.5            10  13.2
3 OJ    0.5            10  13.2
4 OJ    0.5            10  13.2
5 OJ    0.5            10  13.2
6 OJ    0.5            10  13.2
```

- 関数 *ungroup()* でグループ化情報をクリアすることが出来る

```
> ToothGrowth |>
+   group_by(supp, factor(dose)) |>
+   summarize(N=n(), M=mean(len)) |>
+   ungroup()    # 全てのグループ情報を削除 → ungroup(変数名) で特定の変数の情報のみ削除
```

おまけ④: 条件分岐

- if 文の関数版として、関数 ifelse(条件式, TRUEの時の値, FALSEの時の値) があるが、内部では少々乱暴なことをしており、例えば因子型や日付型のデータについて処理を行うと型落ちする場合もある
- パッケージ dplyr の関数 if_else() の方が良く、パッケージ dplyr の関数 case_when() であれば 3 つ以上の条件にも対応できる(ちなみに、`x %in% c("a", "c")` で x の要素に "a" 又は "c" があるかを判定)

```

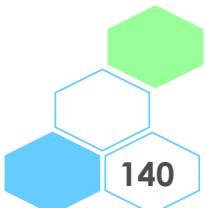
> x <- 0
> ifelse(x > 0, "Posi.", "Zero or Nega.")
[1] "Zero or Nega."
> x <- factor(c("a", "b", "c"))
> ifelse(x %in% c("a", "c"), x, NA) # 因子型→数値型に...
[1] 1 NA 3
> x <- 0
> if_else(x > 0, "Posi.", "Zero or Nega.")
[1] "Zero or Nega."
> x <- NA
> if_else(x > 0, "Posi.", "Zero or Nega.", missing="NA") # NULL以外の欠測の場合も考慮可
[1] "NA"
> case_when(x > 0 ~ "Posi.",
+             x < 0 ~ "Nega.",
+             is.na(x) ~ "N.A.", # 欠測の場合
+             .default = "Zero") # その他
[1] "N.A."

```



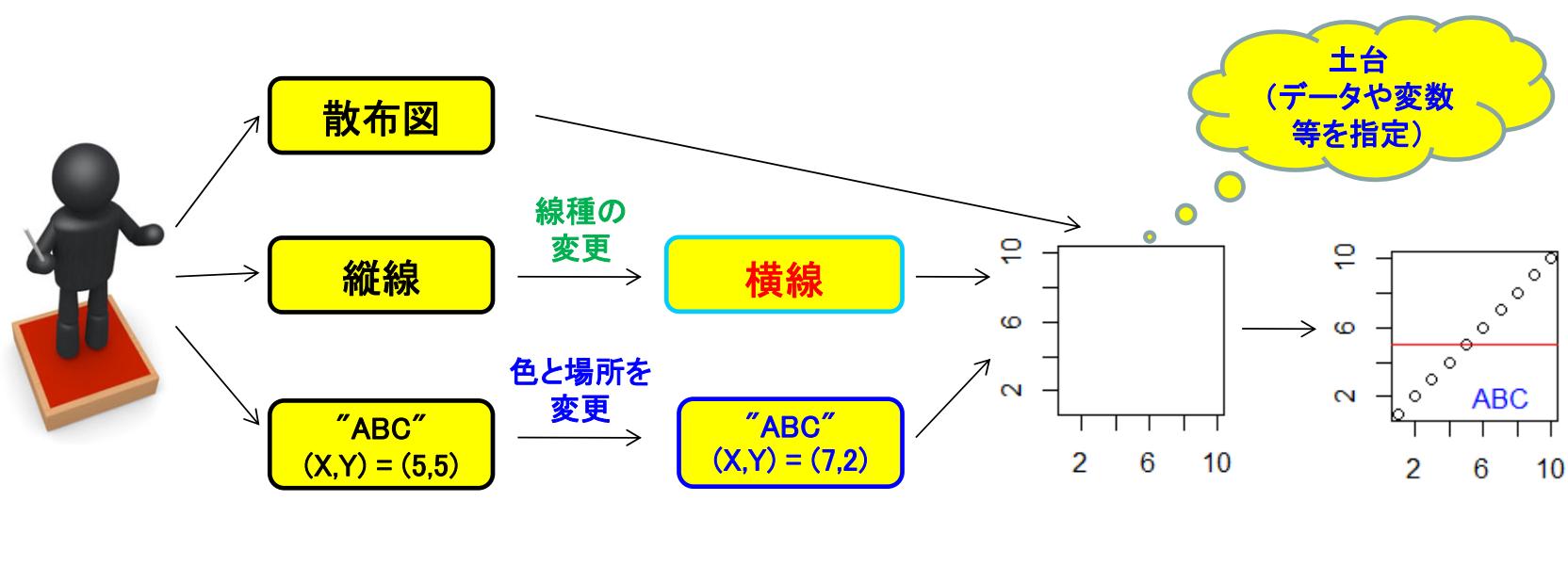
本日のメニュー

- Windows 版 R / RStudio のセットアップ方法 ← 当日までに自習
- R の基礎知識 ← 当日までに自習
- パッケージと作業ディレクトリ ← 当日までに自習
- 変数とベクトル
- ベクトルの型
- 関数とプログラミング
- データ加工とパイプ演算子
- グラフ作成(ggplot2)
- その他



パッケージ *ggplot2* にてグラフ作成

- 「グラフに関するオブジェクト(≒変数)」を使って描くスタイル
- ggplot() で土台となるグラフを作った後、点や線や文字に関する変数を geom_XXX() で作成し、必要に応じてカスタマイズした後、土台に貼り付けるスタイル
⇒ 作成した変数は再利用が可能



使用するデータ: *ToothGrowth* ⇒ データフレーム *tg*

- モルモットにビタミン C 又はオレンジジュースを与えた時の歯の長さを調べる
 - `len`: 長さ(mm)
 - `supp`: サプリの種類(VC(ビタミンC) 又は OJ(オレンジジュース))
 - `dose`: 用量(0.5mg, 1.0mg, 2.0mg) ⇒ 関数 `factor()` で因子ベクトルに変換したものを変数 `dose_c` に

```
> tg      <- ToothGrowth
> tg$dose_c <- factor(tg$dose)
> head(tg, n=3)  # 先頭 3 行
  len supp dose dose_c
1 4.2  VC  0.5    0.5
2 11.5 VC  0.5    0.5
3  7.3  VC  0.5    0.5
> tail(tg, n=3)  # 末尾 3 行
  len supp dose dose_c
58 27.3  OJ    2      2
59 29.4  OJ    2      2
60 23.0  OJ    2      2
```

ggplot2 事始

```
> library(ggplot2)
> base <- ggplot(tg, aes(x=dose_c, y=len, color=dose_c))
```

- 関数 `ggplot()`: 土台を作成する
 - `ggplot(データフレーム名, aes(x 座標の変数, y 座標の変数, その他のエステ属性※))`
- 関数 `aes()`: x 座標や y 座標の変数等のエステ属性を指定する(全て指定する必要は無い)
- エステ属性: x 座標や y 座標の変数、色、大きさ、線の種類、プロット点の形等
 - ここでは x 座標や y 座標の変数の他に、用量(`dose_c`)と色(`color`)を紐付けしており、用量ごとに色を変えたり、用量の情報を凡例に盛り込む際の手掛かりとなる
- 土台(変数 `base`)を作成しただけなので、これだけではグラフを作成したことにならない

※ aesthetic attribute: 審美的属性、気持ち悪い言葉ですが随所で出てきますので慣れましょう

ggplot2 事始

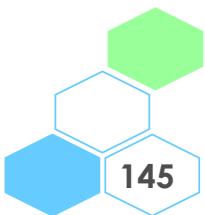
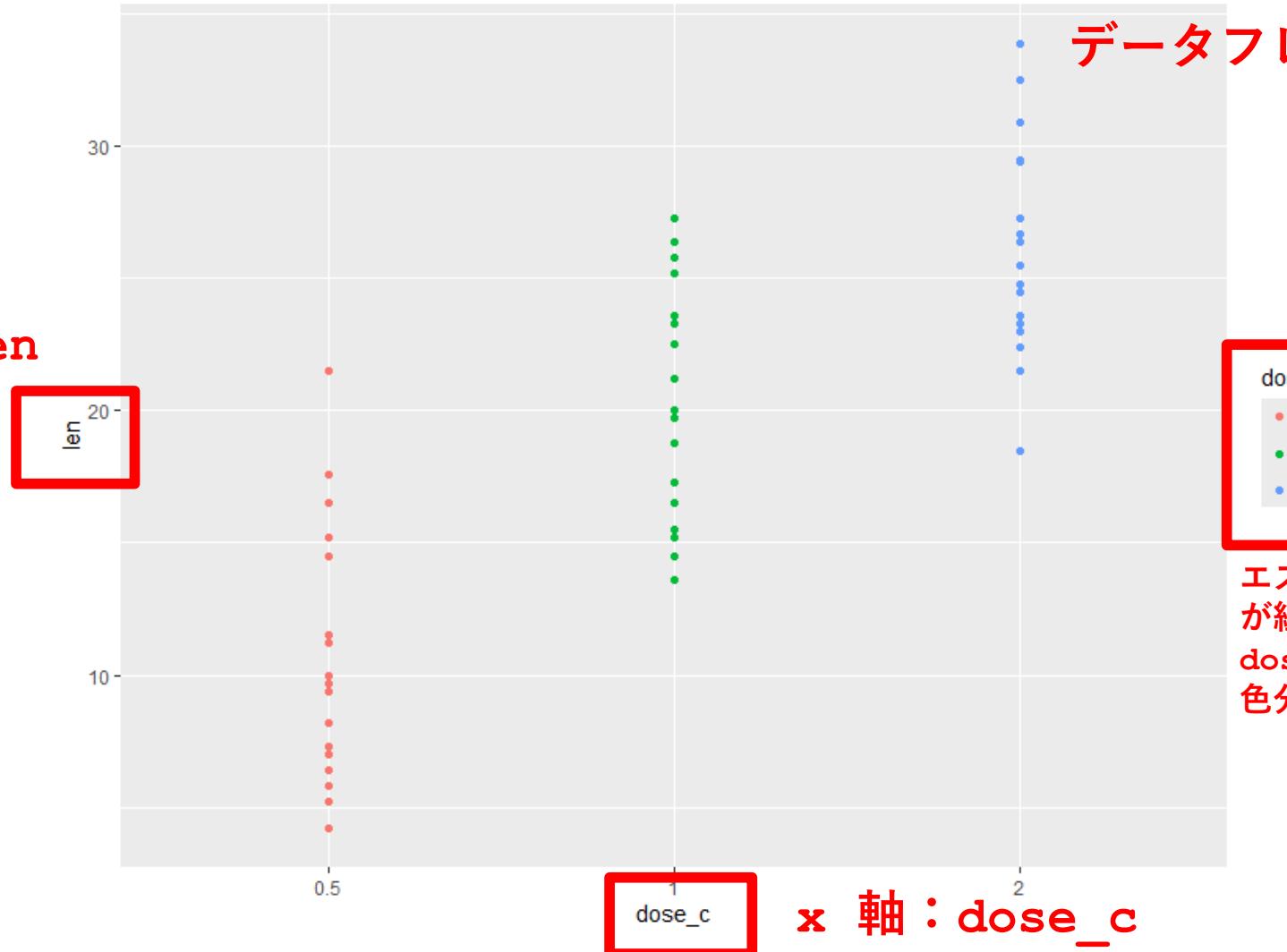
```
> points <- base + geom_point()  
> plot(points)  
  
> base + geom_point()      # plot(points) と同じ働き
```

- 先程作成した土台(変数 base)にレイヤー※を追加した変数を作成する
 - レイヤーとは「データに関連する要素」のこと、例えば上記の関数 geom_point() では「点レイヤー」を追加、すなわち「グラフの種類は散布図ですよ」という属性を変数 base に与えていることになる
 - レイヤーを追加する場合は + を用いる ⇒ パイプ演算子 |> と少し似ている…
- 最後に関数 plot() に変数 points を指定することでグラフが表示される
 - 巷では「base + geom_point()」でグラフを作成することが多いので本資料でもこちらの方針で

※ あまり聞き慣れない言葉かもしれませんが慣れましょう(レイヤーの種類は後述)

ggplot2 事始

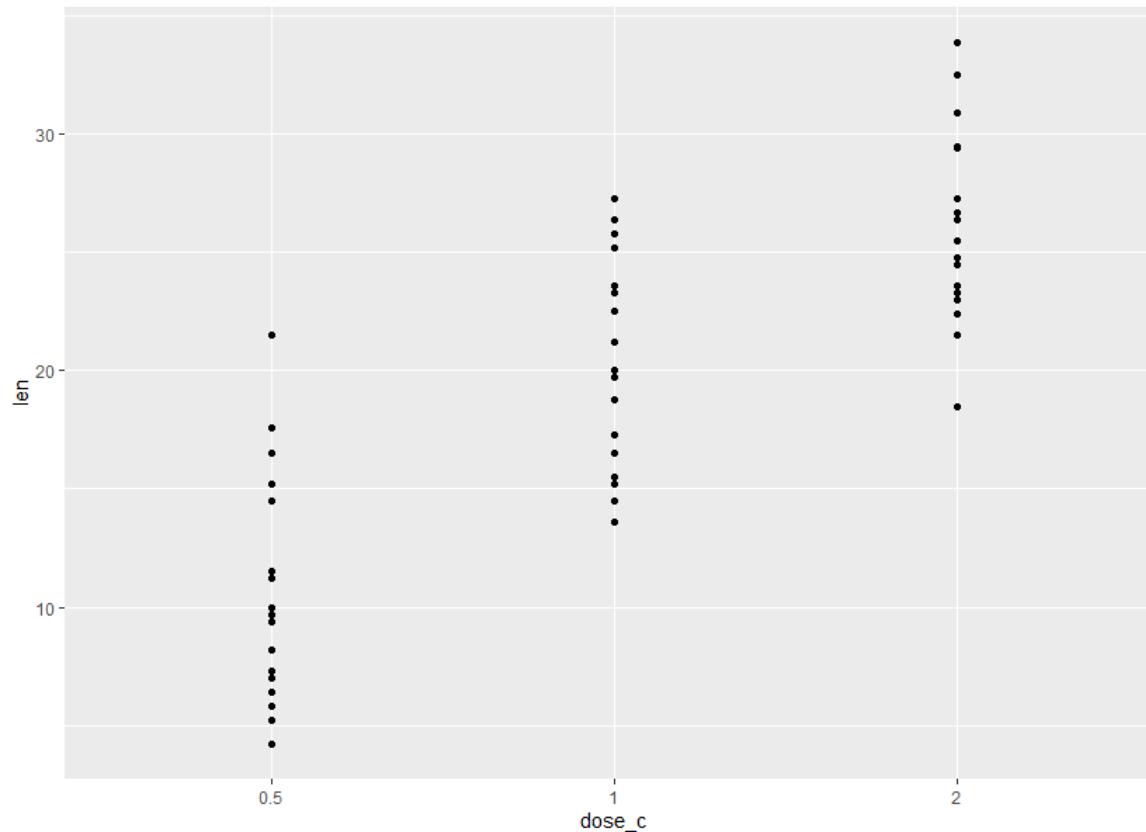
y 軸 : len



ggplot2 事始

- x 座標や y 座標の変数以外のエステ属性を指定しなければ、以下の様なグラフとなる

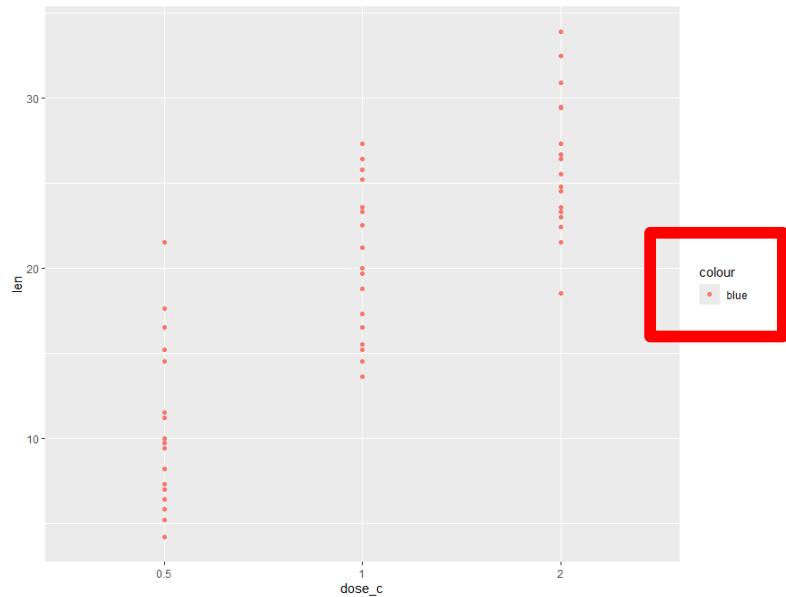
```
> base <- ggplot(tg, aes(x=dose_c, y=len))  
> base + geom_point()
```



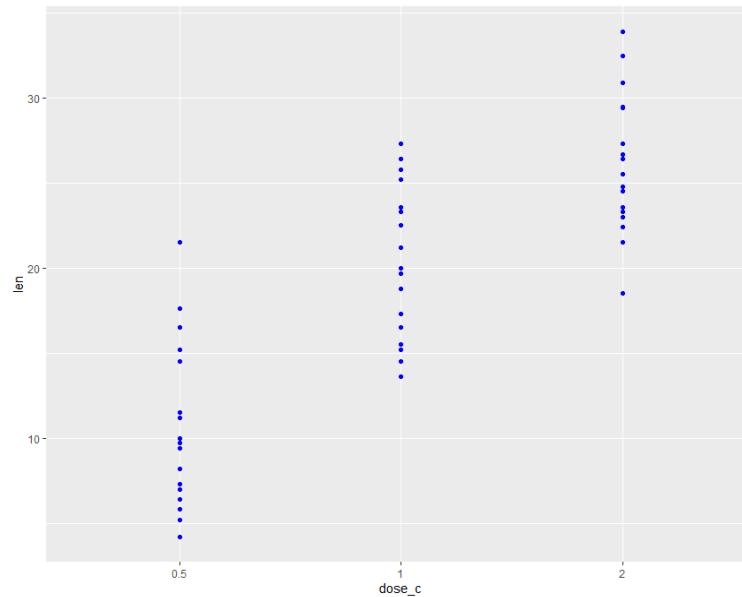
ggplot2 事始

- 固定値(例えば color="blue")を指定したい場合に、エステ属性に直接指定すると変になる
 - 土台(変数 base)ではなく、個々のレイヤー関数に固定値(例えば color="blue")を指定すること

```
> base <- ggplot(tg, aes(x=dose_c, y=len, color="blue"))
> base + geom_point()
> base <- ggplot(tg, aes(x=dose_c, y=len))
> base + geom_point(color="blue")
```



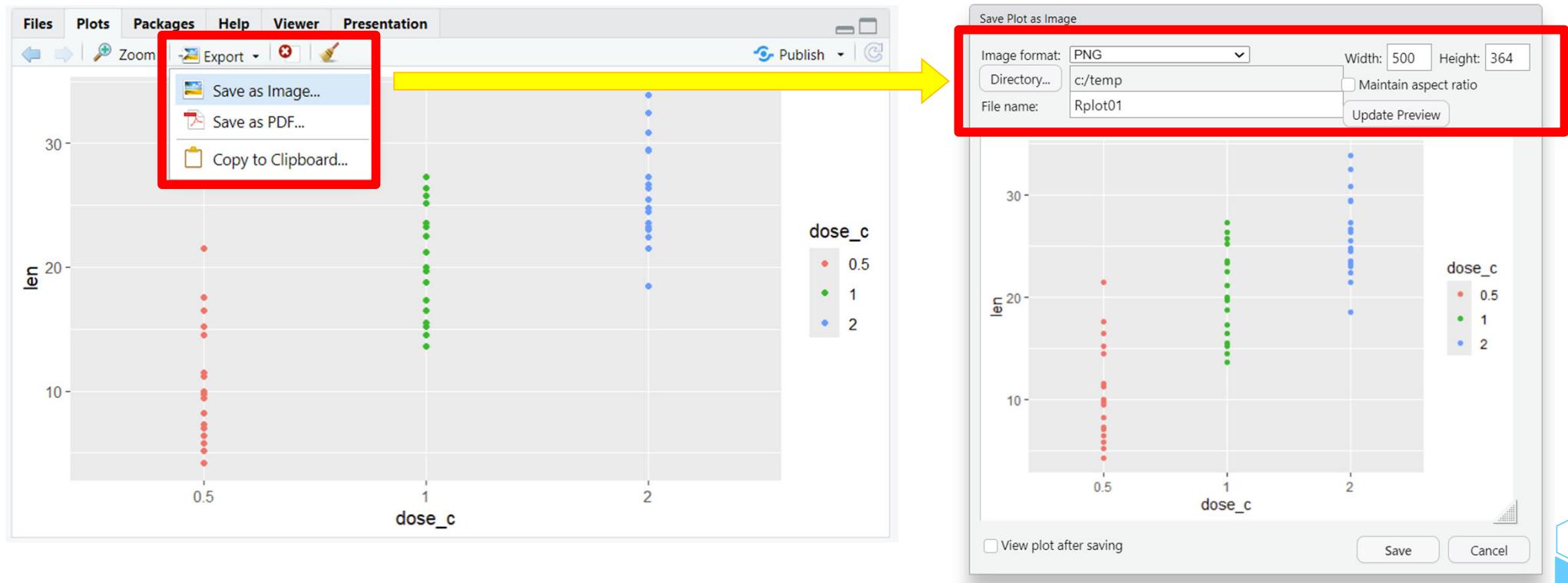
【失敗例】



【成功例】

ggplot2 事始

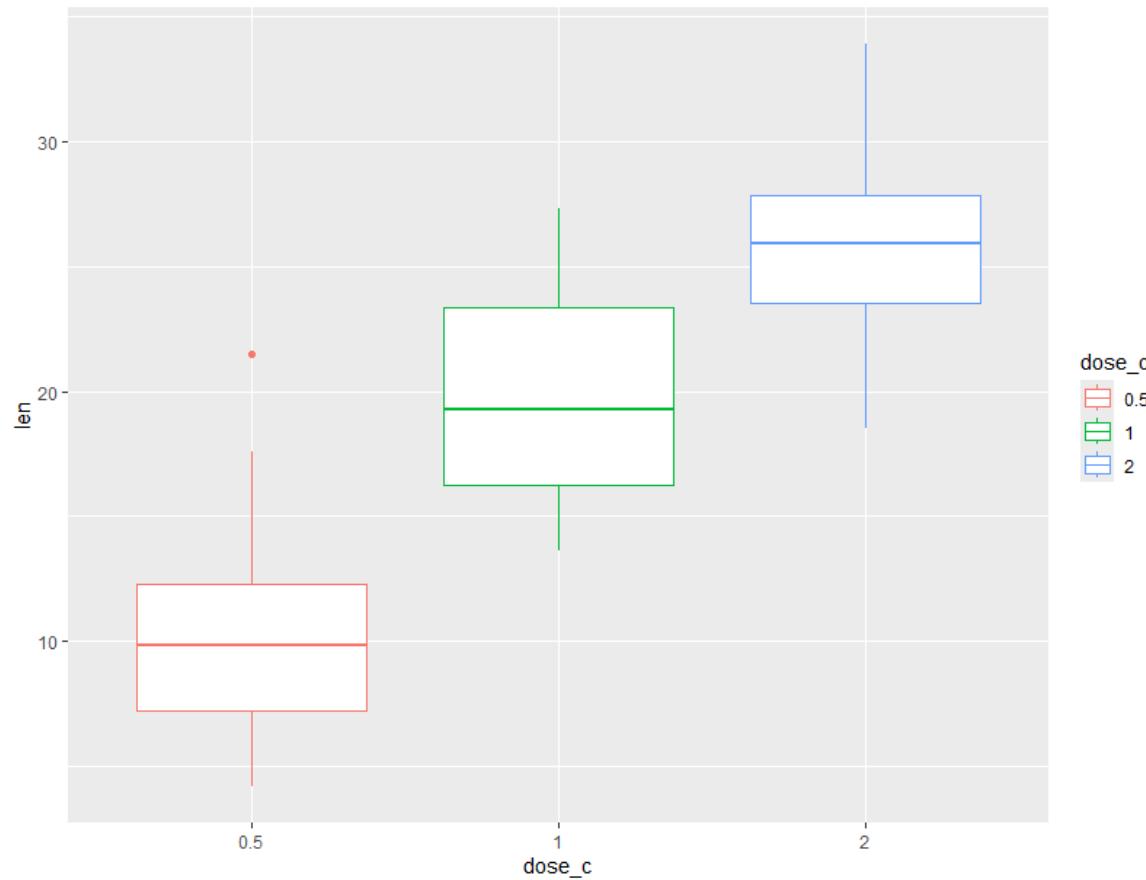
- グラフを保存する場合はメニュー [Export] から、又は関数 ggsave() を使用する
 - PNG 形式: `ggsave("myplot.png")`
 - PDF形式: `ggsave("myplot.pdf", width=20, height=20, units="cm")`



ggplot2 事始

- 変数 base を再利用して箱ひげ図を描く

```
> base + geom_boxplot()
```



主な関数 *geom_XXX()*

関数	種類	エステ属性
geom_abline	直線(切片と傾きを指定)	intercept, slope
geom_area	曲線下の塗りつぶし	x or y , ymin or xmin , ymax or xmax , alpha, color, fill, group, linetype, linewidth
geom_bar, geom_col	棒グラフ	x , y , alpha, color, fill, group, linetype, linewidth, width
geom_boxplot	箱ひげ図	x or y , lower or xlower , upper or xupper , middle or xmiddle , ymin or xmin , ymax or xmax , alpha, colour, fill, group, linetype, linewidth, shape, size, weight, width
geom_density	密度曲線	x , y , alpha, color, fill, group, linetype, linewidth, weight
geom_dotplot	ドットプロット	x or y , alpha, color, fill, group, linetype, stroke, weight, width
geom_errorbar	誤差・エラーバー(縦)	x or y , ymin or xmin , ymax or xmax , alpha, color, fill, group, linetype, linewidth
geom_errorbarh	誤差・エラーバー(横)	x or y , ymin or xmin , ymax or xmax , alpha, color, fill, group, linetype, linewidth
geom_function	数学関数を描く	x , y , alpha, color, group, linetype, linewidth
geom_histogram	ヒストグラム	x or y , alpha, color, fill, group, linetype, linewidth, width
geom_hline	水平線を描く	yintercept
geom_line	線を描く	x , y , alpha, color, group, linetype, linewidth
geom_point	散布図	x , y , alpha, color, fill, group, shape, size, stroke
geom_smooth	平滑線	x , y , alpha, color, fill, group, linetype, linewidth, weight, ymax, ymin
geom_step	階段関数	x , y , alpha, color, group, linetype, linewidth
geom_violin	バイオリン・プロット	x , y , alpha, color, fill, group, linetype, linewidth, weight, width
geom_vline	縦線を描く	xintercept

参考: <https://ggplot2.tidyverse.org/reference/index.html> ※ 青字: 必ず指定しなければいけない引数(or はどちらか一方)

おまけ①: 要約統計量を算出してからグラフ作成

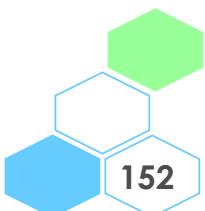
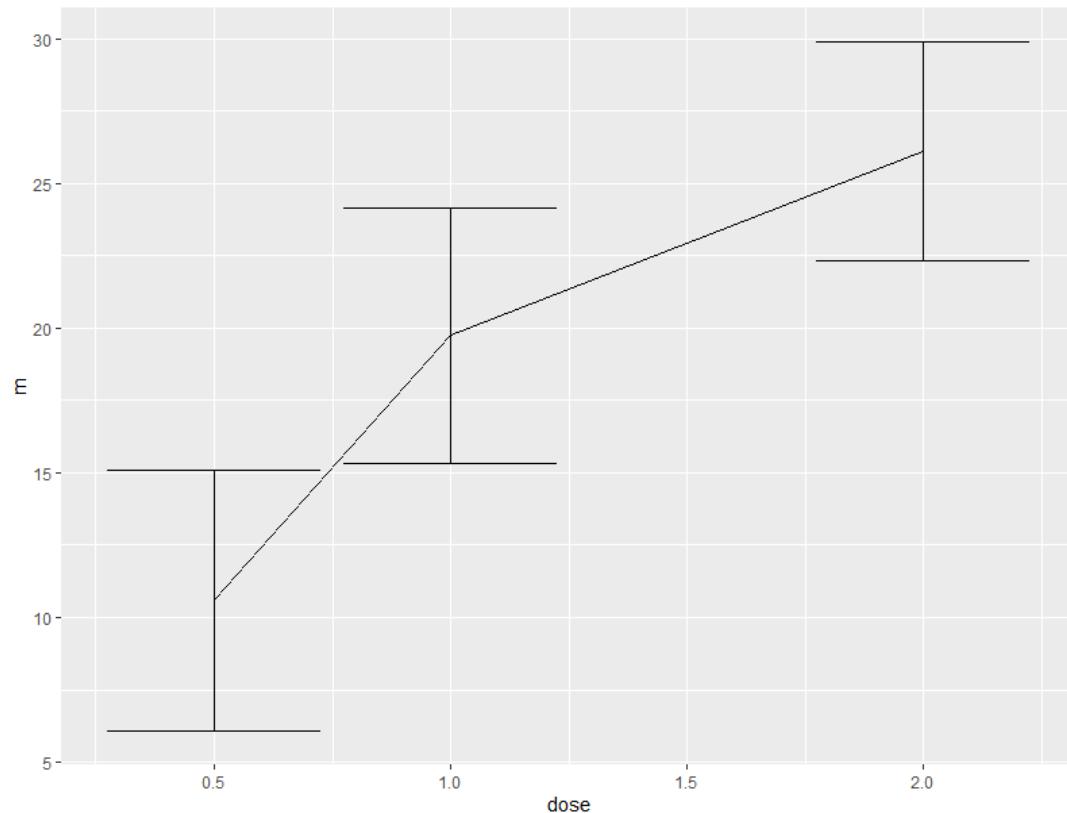
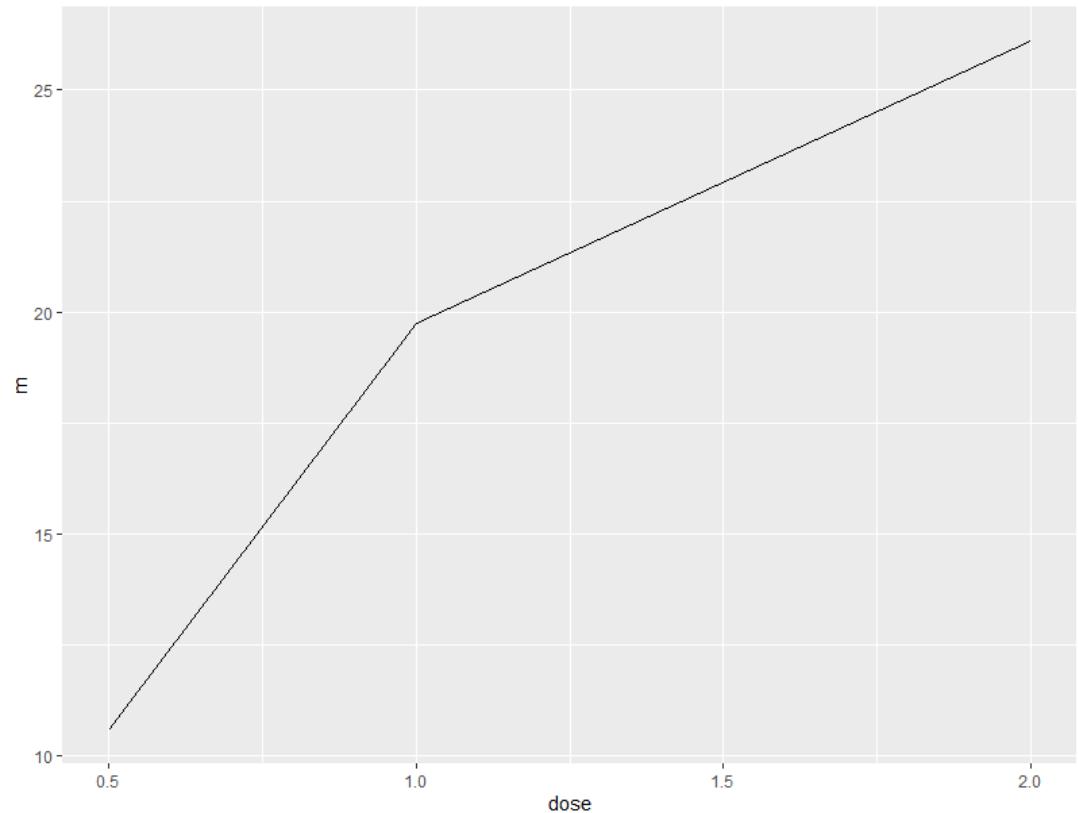
- プログラムが複数行にわたる場合、演算子 **+** を行の最後に持ってくること

```
> ( MS <- ToothGrowth |>
+     group_by(dose) |>
+     summarise(m=mean(len), s=sd(len)) )
# A tibble: 3 × 3
  dose      m      s
  <dbl> <dbl> <dbl>
1   0.5    10.6   4.50
2     1    19.7   4.42
3     2    26.1   3.77

> base <- ggplot(MS, aes(x=dose, y=m))
> base + geom_line() # 折れ線グラフ

> ggplot(MS, aes(x=dose, y=m)) +
+   geom_line() +
+   geom_errorbar(aes(ymin=m-s, ymax=m+s)) # 土台となる変数を作成しない
```

おまけ①：要約統計量を算出してからグラフ作成



演習 9

- 以下を実行して変数 base (土台)を作成してください

```
> base <- ggplot(tg, aes(x=len, color=supp))
```

1. の命令でエステ属性は何でしょうか
- 変数 base と関数 geom_histogram() を用いてヒストグラムを作成してください
- 変数 base と関数 geom_density() を用いて密度曲線を作成してください
- 以下の情報が入った変数 base (土台)を作成してください
 - データフレーム: tg、x 軸の変数: dose、y 軸の変数: len、エステ属性 color: supp
5. の変数 base と関数 geom_point() と関数 geom_smooth(method=lm, se=F) を用いて回帰直線付きの散布図を描いて下さい
5. ~ 6. の作業を、変数 base を作成せずに行って下さい

補足：演習 9 の問題 3 ~ 4

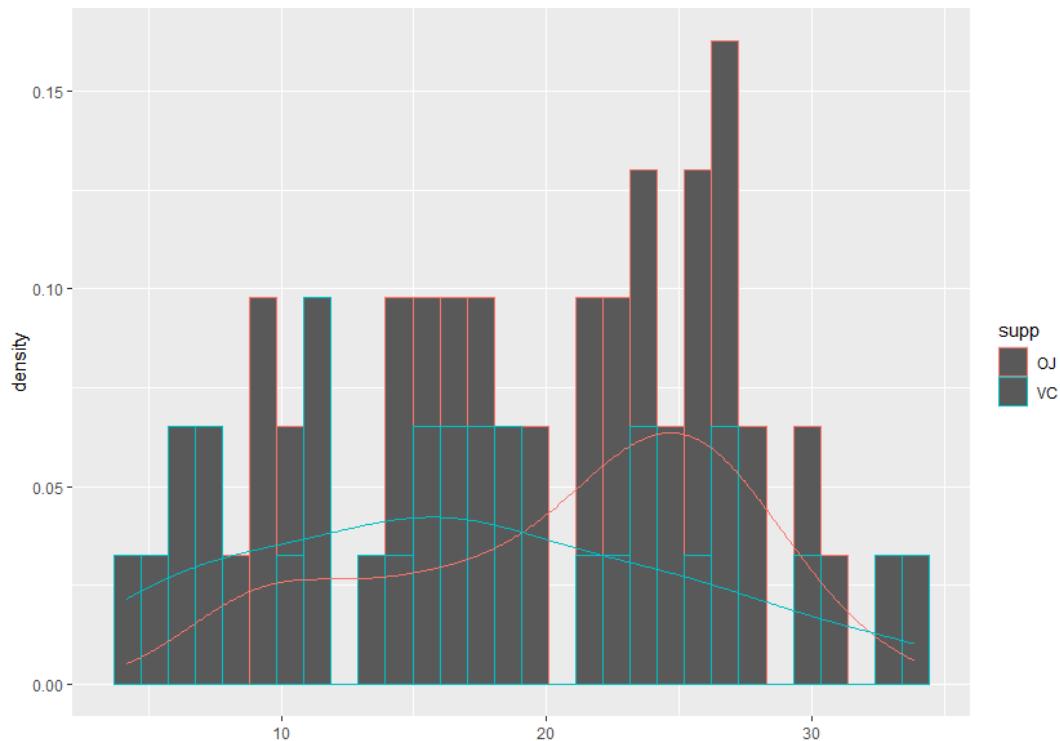
- ヒストグラムと密度曲線を重ね書きすることが出来る

```
> base <- ggplot(tg, aes(x=len, y=after_stat(density), color=supp))  
> base + geom_histogram() + geom_density()
```

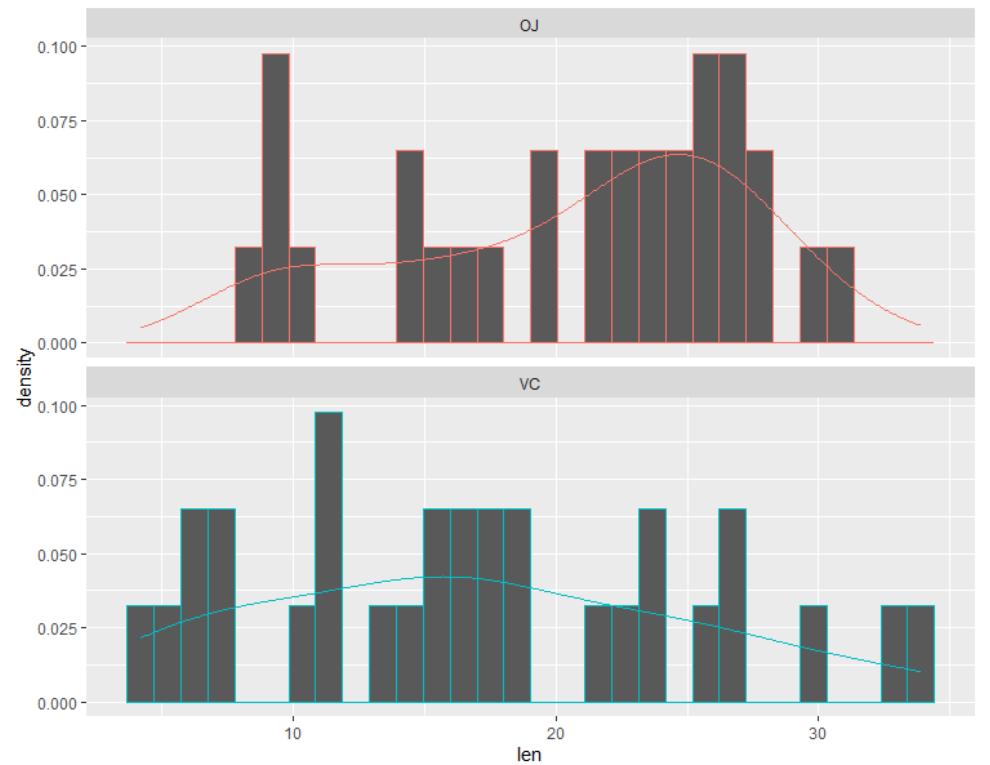
- ただ、ヒストグラムを重ね書きすると少し見づらい
⇒ 関数 `facet_wrap(~ グループ変数)` で、層別グラフを描くことが出来る

```
> base <- ggplot(tg, aes(x=len, y=after_stat(density), color=supp))  
> base + geom_histogram() + geom_density() + facet_wrap(~supp, ncol=1)
```

補足：演習 9 の問題 3 ~ 4



【1つ目】



【2つ目】

グラフのカスタマイズ前

- 平均値の推移図を描く

```
> ( MS <- ToothGrowth |>
+     group_by(supp, dose) |>
+     summarise(m=mean(len), s=sd(len)) )
`summarise()` has grouped output by 'supp'. You can override using the `groups` argument.
# A tibble: 6 × 4
# Groups:   supp [2]
  supp   dose     m     s
  <fct> <dbl> <dbl> <dbl>
1 OJ      0.5 13.2  4.46
2 OJ      1    22.7  3.91
3 OJ      2    26.1  2.66
4 VC      0.5  7.98  2.75
5 VC      1    16.8  2.52
6 VC      2    26.1  4.80

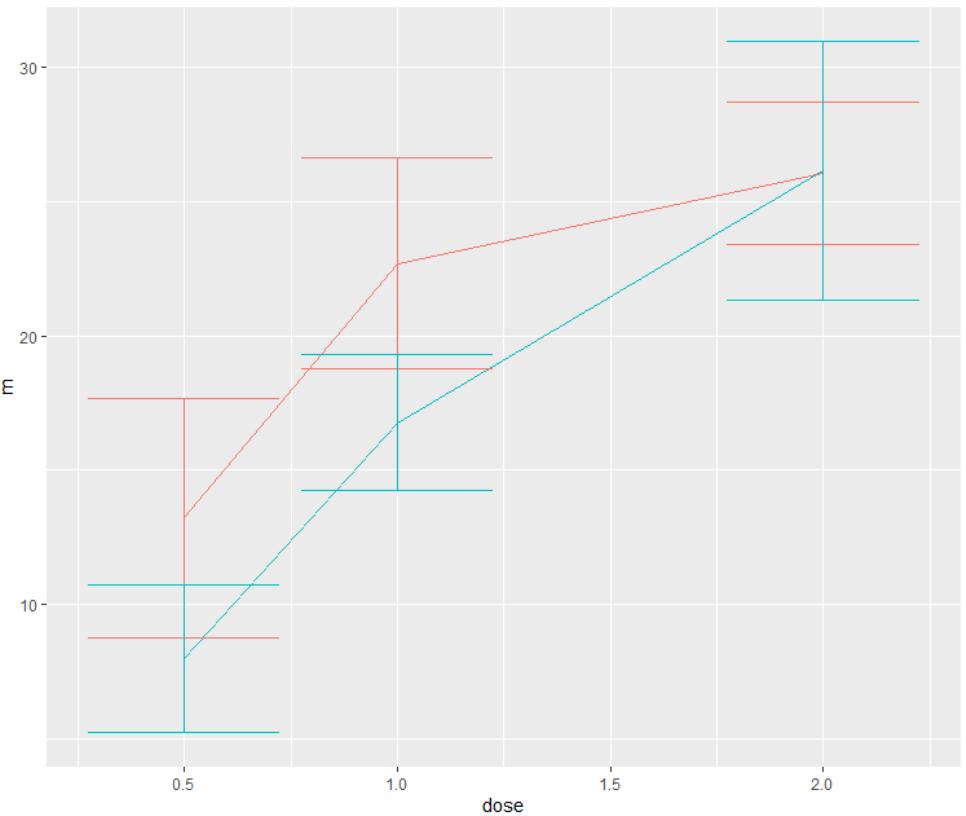
> ggplot(MS, aes(x=dose, y=m, color=supp)) +
+   geom_line() +
+   geom_errorbar(aes(ymin=m-s, ymax=m+s))
```

グラフのカスタマイズ

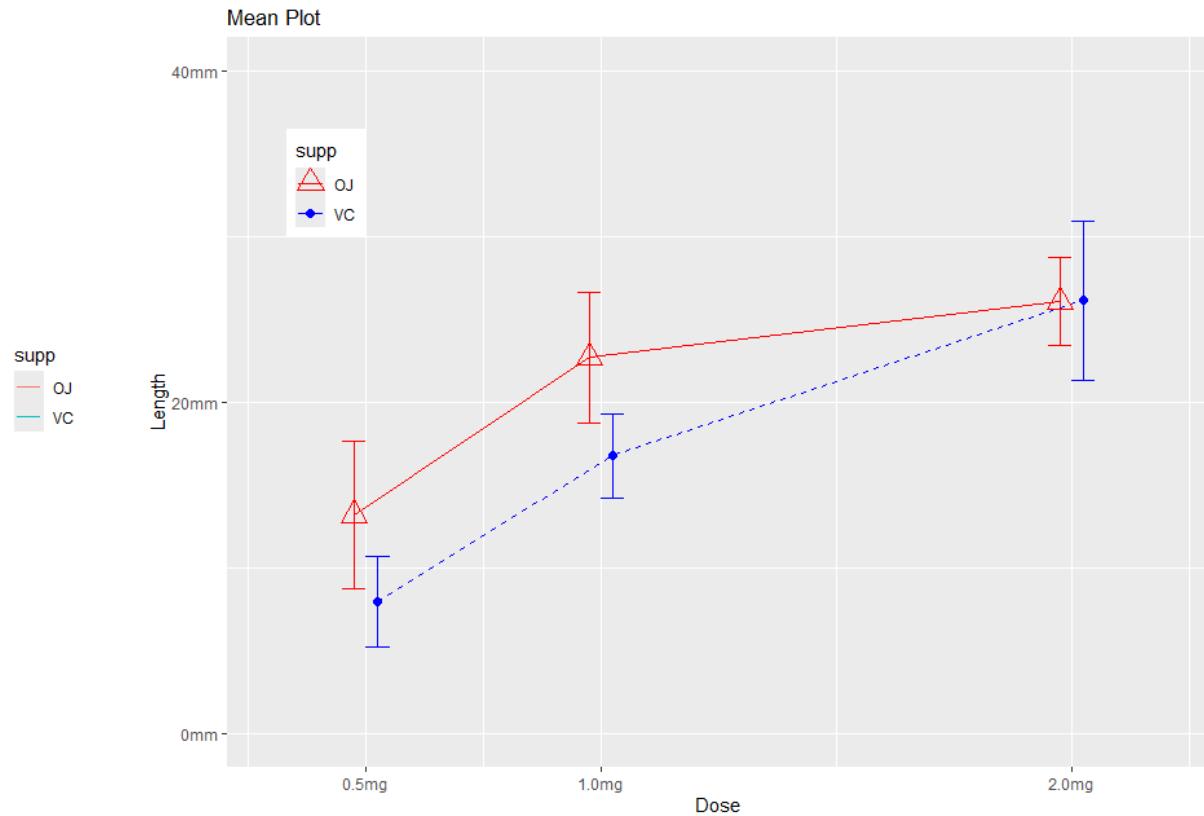
- 平均値の推移図を描く

```
> pd <- position_dodge(.1) # プロットをズラす幅  
  
> ggplot(MS, aes(x=dose, y=m, color=supp)) +  
+   geom_errorbar(aes(ymin=m-s, ymax=m+s), width=.1, position=pd) +  
+   geom_line(aes(linetype=supp), position=pd) +  
+   geom_point(aes(shape=supp, size=supp), position=pd) +  
+   scale_color_manual(values=c("red","blue")) +  
+   scale_linetype_manual(values=c("solid", "dashed")) +  
+   scale_shape_manual(values=c(2,19)) +  
+   scale_size_manual(values=c(5,2)) +  
+   theme(legend.position="inside", legend.position.inside=c(0.1,0.8)) +  
+   xlab("Dose") + ylab("Length") + ggtitle("Mean Plot") +  
+   scale_x_continuous(limits=c(0.3,2.2), breaks=c(0.5,1,2),  
+                      labels=c("0.5mg","1.0mg","2.0mg")) +  
+   scale_y_continuous(limits=c(0,40), breaks=seq(0,40,20),  
+                      labels=c("0mm","20mm","40mm"))
```

グラフのカスタマイズ



【カスタマイズ前】



【カスタマイズ後】

グラフのカスタマイズ

```
> ggplot(MS, aes(x=dose, y=m, color=supp)) +
+   geom_errorbar(aes(ymin=m-s, ymax=m+s), width=.1, position=pd) +
+   geom_line(aes(linetype=supp), position=pd) +
+   geom_point(aes(shape=supp, size=supp), position=pd) + ...
```

- 関数 `geom_errorbar()` でエラーバーのプロット
 - 引数 `ymin` と `ymax` でエラーバーの長さを指定、x 軸と y 軸の変数は既に紐づけ済
 - 引数 `width` でエラーバーの線の幅を指定
 - 引数 `position` で「プロットをズラす幅」を指定
- 関数 `geom_line()` で線のプロット
 - 引数 `position` で「プロットをズラす幅」を指定、`linetype=supp` をエステ属性に紐付け
 - x 軸と y 軸の変数、色(`color=supp`)は既にエステ属性に紐付けられている
- 関数 `geom_point()` で点のプロット
 - 引数 `position` で「プロットをズラす幅」を指定、`shape=supp` をエステ属性に紐付け
 - x 軸と y 軸の変数、色(`color=supp`)は既にエステ属性に紐付けられている

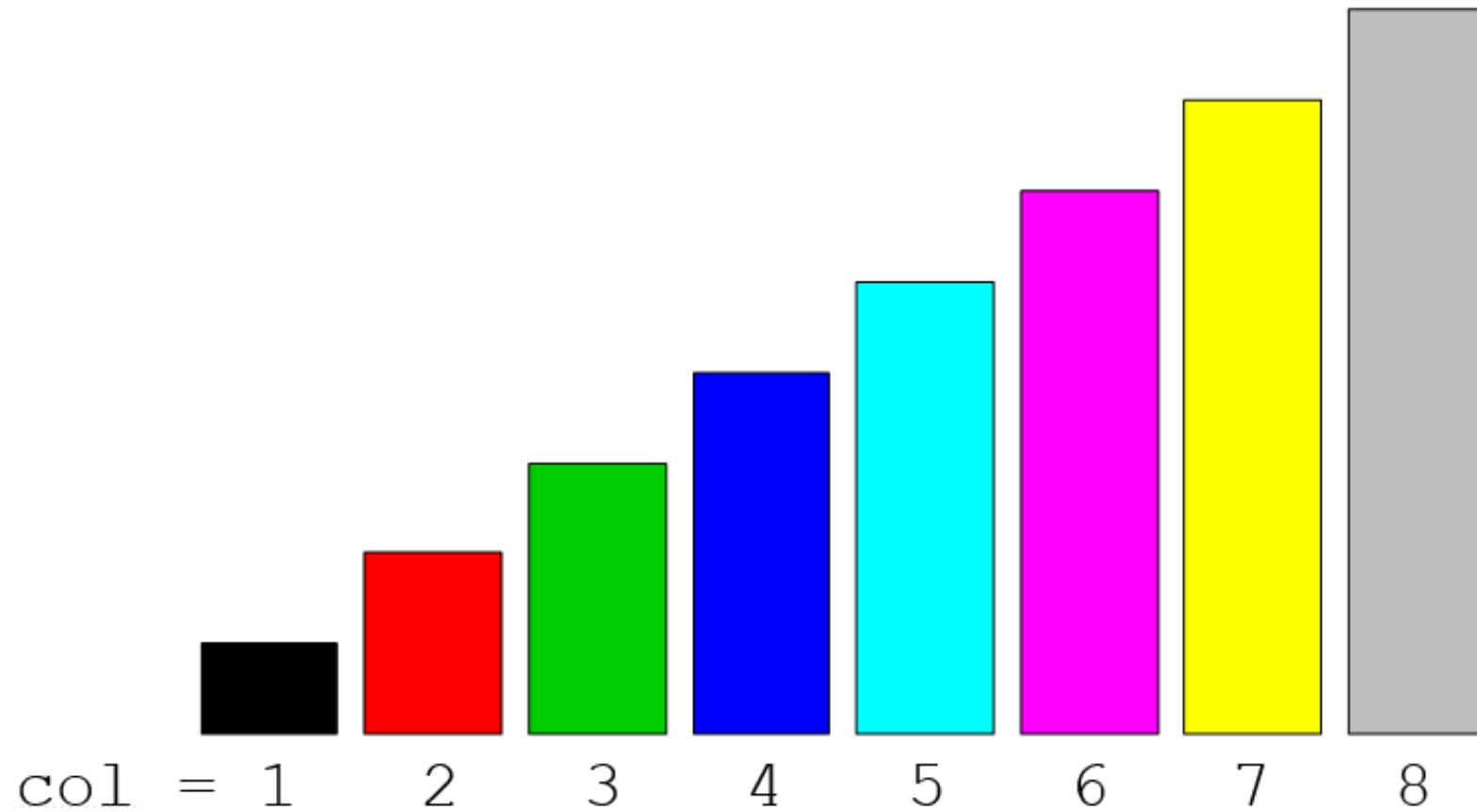
グラフのカスタマイズ

```
> ggplot(MS, aes(x=dose, y=m, color=supp)) +  
+   geom_errorbar(aes(ymin=m-s, ymax=m+s), width=.1, position=pd) +  
+   geom_line(aes(linetype=supp), position=pd) +  
+   geom_point(aes(shape=supp, size=supp), position=pd) +  
+   scale_color_manual(values=c("red","blue")) + ...
```

- データをエステ属性(**color**、**linetype**、**shape**….)に紐づけておくと、関数 `scale_XXX_manual()` で各群のプロットの見栄えが変更可能
 - まず、関数 `ggplot()` で `color` のエステ属性を指定しているので、グラフ全体として変数 `supp` のカテゴリごとに色分けがなされる
 - 次に、色自体の調整は、関数 `scale_color_manual()` にて行う
 - R で指定できる色、線、点の種類は次頁以降にて

色の種類

```
> barplot(1:8, col=1:8, axes=F)
```



色の種類

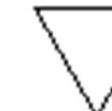
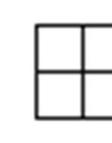
```
> colors()

[1] "white"           "aliceblue"        "antiquewhite"
[4] "antiquewhite1"   "antiquewhite2"   "antiquewhite3"
[7] "antiquewhite4"   "aquamarine"      "aquamarine1"
[10] "aquamarine2"    "aquamarine3"    "aquamarine4"
[13] "azure"           "azure1"          "azure2"
[16] "azure3"          "azure4"          "beige"
[19] "bisque"          "bisque1"         "bisque2"
[22] "bisque3"         "bisque4"         "black"
[25] "blanchedalmond"  "blue"            "blue1"
[28] "blue2"           "blue3"           "blue4"
[31] "blueviolet"       "brown"           "brown1"
[34] "brown2"          "brown3"          "brown4"
...
[655] "yellow3"         "yellow4"         "yellowgreen"
```

線の種類

引数 lty	種類
lty=0 , lty="blank"	(透明)
lty=1 , lty="solid"	——
lty=2 , lty="dashed"	- - -
lty=3 , lty="dotted"
lty=4 , lty="dotdash"	- . - .
lty=5 , lty="longdash"	-----
lty=6 , lty="twodash"	- - - -

点の種類

pch	0	1	2	3	4	5	6	7	8
									
pch	9	10	11	12	13	14	15	16	17
									
pch	18	19	20	21	22	23	24	25	
									

グラフのカスタマイズ

```
> ggplot(MS, aes(x=dose, y=m, color=supp)) +  
+   geom_errorbar(aes(ymin=m-s, ymax=m+s), width=.1, position=pd) +  
+   geom_line(aes(linetype=supp), position=pd) +  
+   geom_point(aes(shape=supp, size=supp), position=pd) +  
+   scale_color_manual(values=c("red","blue")) +  
+   scale_linetype_manual(values=c("solid", "dashed")) +  
+   scale_shape_manual(values=c(2,19)) +  
+   scale_size_manual(values=c(5,2)) + ...
```

- 特定のグラフや図形にのみエステ属性を紐づけることも出来る
 - 例えば、関数 `geom_line()` では `linetype` のエステ属性を指定しているので、線グラフのみ変数 `supp` のカテゴリごとに線の種類分けがなされる
 - 関数 `geom_point()` についても同様の仕組み
- 関数 `scale_XXX_manual()` で各群のプロットの見栄えが変更可能
 - 例えば、線種の調整は、関数 `scale_line_manual()` にて行う

関数 *scale_XXX_manual()*

```
> ggplot(MS, aes(x=dose, y=m, color=supp)) + ...
+   scale_color_manual(values=c("red","blue")) +
+   scale_linetype_manual(values=c("solid", "dashed")) +
+   scale_shape_manual(values=c(2,19)) +
+   scale_size_manual(values=c(5,2)) + ...
```

- エステ属性を変更するための関数は以下、引数 values に点・線・色などの種類を指定
 - `scale_color_manual(values, ...)` : 点や線の色
 - `scale_fill_manual(values, ...)` : 塗りつぶしの色
 - `scale_size_manual(values, ...)` : 大きさ
 - `scale_shape_manual(values, ...)` : 点の種類
 - `scale_linetype_manual(values, ...)` : 線の種類
 - `scale_linewidth_manual(values, ...)` : 線の太さ
 - `scale_alpha_manual(values, ...)` : 図形の透明度

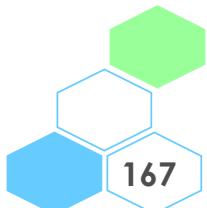
凡例やタイトルの調整

```
> ggplot(MS, aes(x=dose, y=m, color=supp)) + ...  
+   theme(legend.position="inside", legend.position.inside=c(0.1,0.8)) +  
+   xlab("Dose") + ylab("Length") + ggtitle("Mean Plot") + ...
```

- 凡例の位置
 - theme(legend.position="inside", legend.position.inside=c(0.1,0.8)): 特定の位置を指定(0 ~ 1)
 - theme(legend.position="top", "right", "bottom", "left", "inside", "none")なる指定が可
 - 関数 theme() はオプション多数 → 詳細は下記頁や R Graphics Cookbook (2nd Edition) を参照
<https://ggplot2.tidyverse.org/reference/theme.html>
- タイトル関係
 - xlab("Dose"): x 軸のタイトルを指定
 - ylab("Length"): y 軸のタイトルを指定
 - ggtitle("Mean Plot"): グラフのタイトルを指定

★ 上記 3 つの命令は labs(x="Dose", y="Length", title="Mean Plot") の 1 つで実現可

<https://ggplot2.tidyverse.org/reference/labs.html>



座標やスケールの調整

```
> ggplot(MS, aes(x=dose, y=m, color=supp)) + ...  
+   scale_x_continuous(limits=c(0.3,2.2), breaks=c(0.5,1,2),  
+                       labels=c("0.5mg","1.0mg","2.0mg")) +  
+   scale_y_continuous(limits=c(0,40), breaks=seq(0,40,20),  
+                       labels=c("0mm","20mm","40mm"))
```

- x 軸や y 軸の変数が連続変数である場合は `scale_x_continuous()` と `scale_y_continuous()` で x 軸や y 軸の範囲等を調整することが出来る
 - **単に x 軸や y 軸の範囲を調整する場合は `xlim(c(0.3, 2.2)) + ylim(c(0, 40))` が簡便**
 - 引数 `trans` にて "asn"(tanh-1(x))、"exp"(ex)、"identity"(無変換)、"log"(log(x))、"log10"(log10(x))、"log2"(log2(x))、"logit"(ロジット関数)、"pow10"(10x)、"probit"(プロビット関数)、"recip"(1/x)、"reverse"(-x)、"sqrt"(平方根)等の座標変換
- x 軸や y 軸の変数がカテゴリ変数である場合は `scale_x_discrete()` と `scale_y_discrete()` で x 軸や y 軸を調整等、変数の種類に応じた関数が用意されている

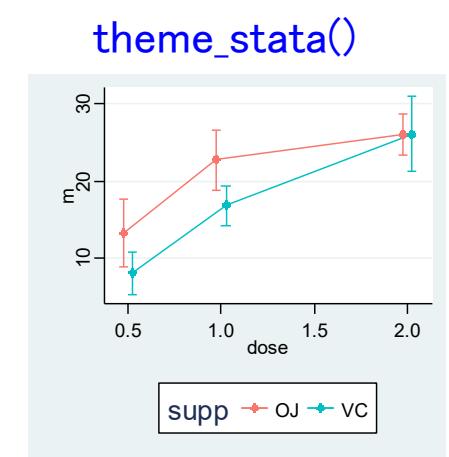
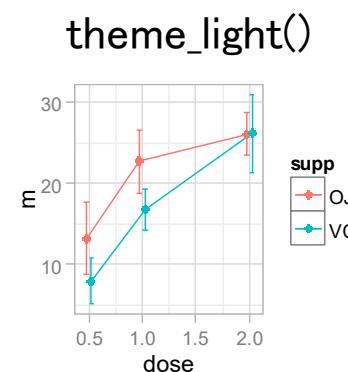
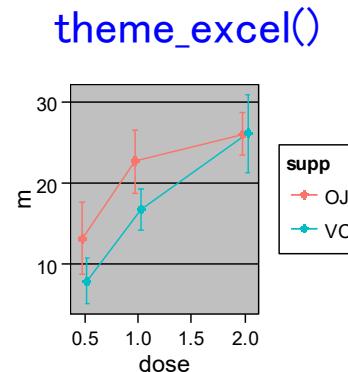
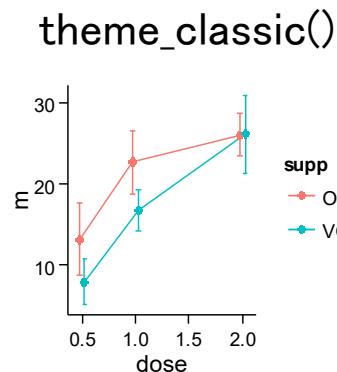
<https://ggplot2.tidyverse.org/reference/index.html#scales>

座標やスケールの調整

- 座標の範囲
 - `xlim(c(0,2.5))`、`ylim(c(0,40))`: x 軸、y 軸の範囲
 - `scale_x_continuous(breaks=NULL, expand=c(0, 0))`: x 軸の目盛を非表示、x軸のマージンを 0 に
 - `scale_y_continuous(limits=c(0,40), breaks=seq(0,40,10), labels=c("0mm","20mm","40mm"))`: y 軸の範囲 + 刻み幅に関する情報
※ 離散データの場合は `scale_x_discrete()`、`scale_y_discrete()` を使用
- スケールの指定
 - `scale_x_date()`, `scale_y_date()`: 日付型
 - `scale_x_datetime()`, `scale_y_datetime()`: 日時型
 - `scale_x_reverse()`, `scale_y_reverse()`: 逆順に表示
- 座標系の指定
 - `coord_fixed(ratio=1/2)`: 表示の際、y/x の比を 1/2 に
 - `coord_flip()`: x 軸と y 軸を逆に表示
 - `coord_polar()`: 極座標表示
 - `coord_trans(x="関数", y="関数")`: 座標変換

見た目の変更

theme_bw()	theme_foundation()	theme_minimal()
theme_calc()	theme_gdocs()	theme_pander()
theme_classic()	theme_gray() , theme_grey()	theme_solarized()
theme_economist()	theme_hc()	theme_solarized_2()
theme_economist_white()	theme_igray()	theme_solid()
theme_excel()	theme_light()	theme_stata()
theme_few()	theme_linedraw()	theme_tufte()
theme_fivethirtyeight()	theme_map()	theme_wsj()
theme_dark()	theme_test()	※ 青字:追加パッケージ「ggthemes」の中の関数



演習 10

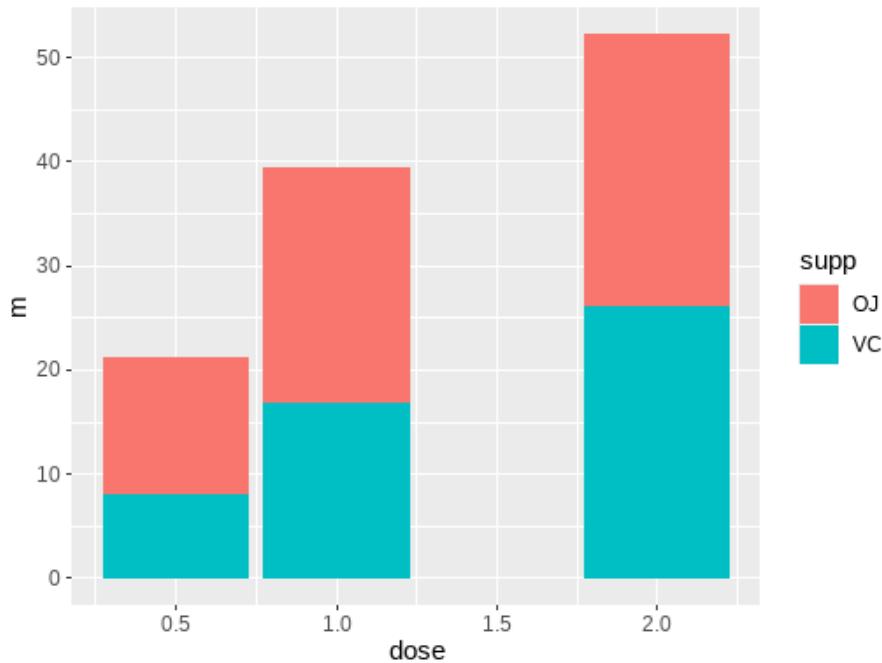
- 以下を実行して変数 dose & supp の各カテゴリの平均値に関する棒グラフを作成してください

```
> ( MS <- ToothGrowth |>
+     group_by(supp, dose) |>
+     summarise(m=mean(len)) )
> ggplot(MS, aes(x=dose, y=m, color=supp, fill=supp)) +
+   geom_col()
```

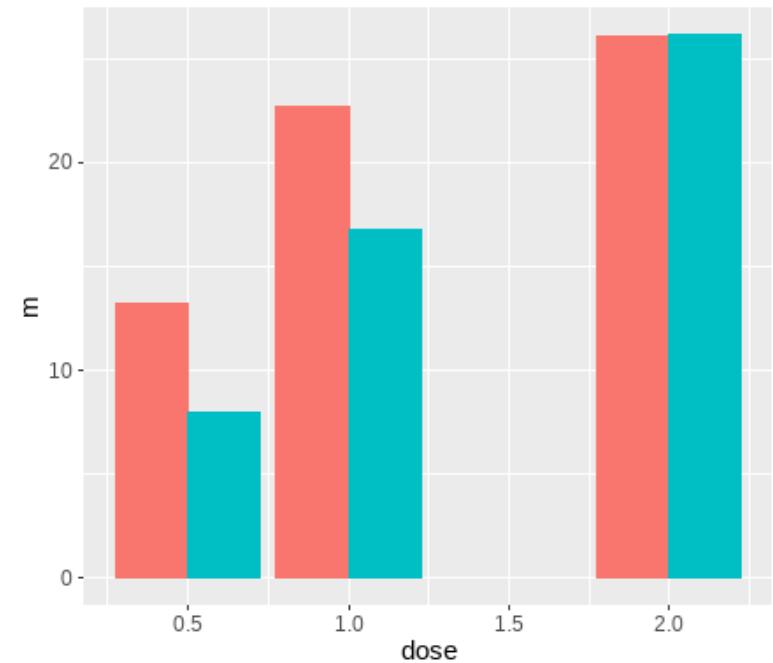
1. では「積み上げ棒グラフ」になるので、以下を実行して「横並び」の棒グラフを作成してください

```
> ggplot(MS, aes(x=dose, y=m, color=supp, fill=supp)) +
+   geom_col(position = "dodge")
```

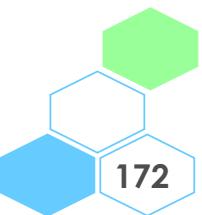
演習 10



【1つ目】



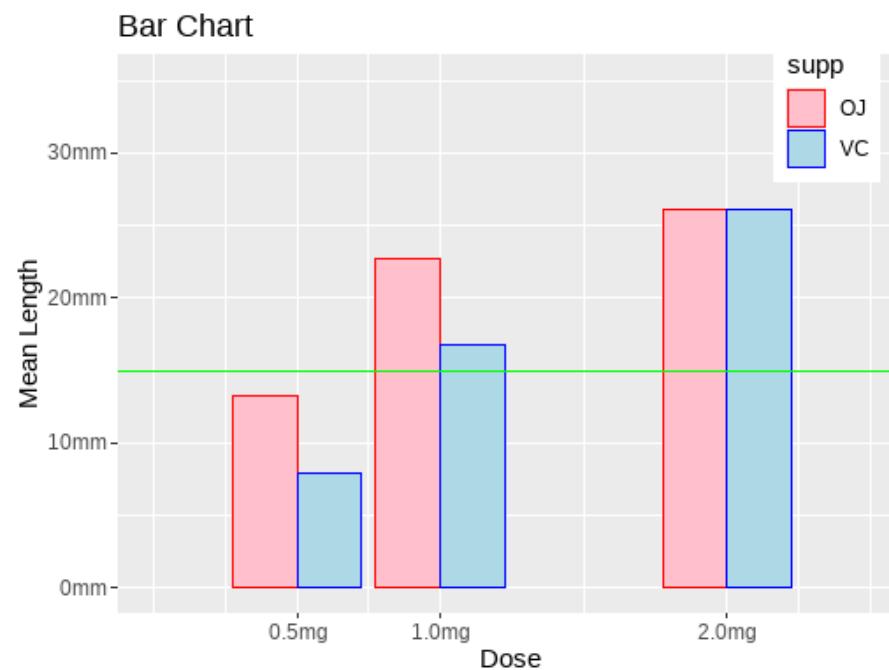
【2つ目】



演習 10

3. 2. のプログラムを使って下のグラフを作成してください(以下は使用する関数の候補です)

- 横線: geom_hline()
- 棒の色: scale_color_manual()、scale_fill_manual()
- 凡例: theme()
- ラベル: xlab()、ylab()、ggtitle()、labs()
- 軸の制御: scale_x_continuous()、scale_y_continuous()



パッケージ *ggplot2* について

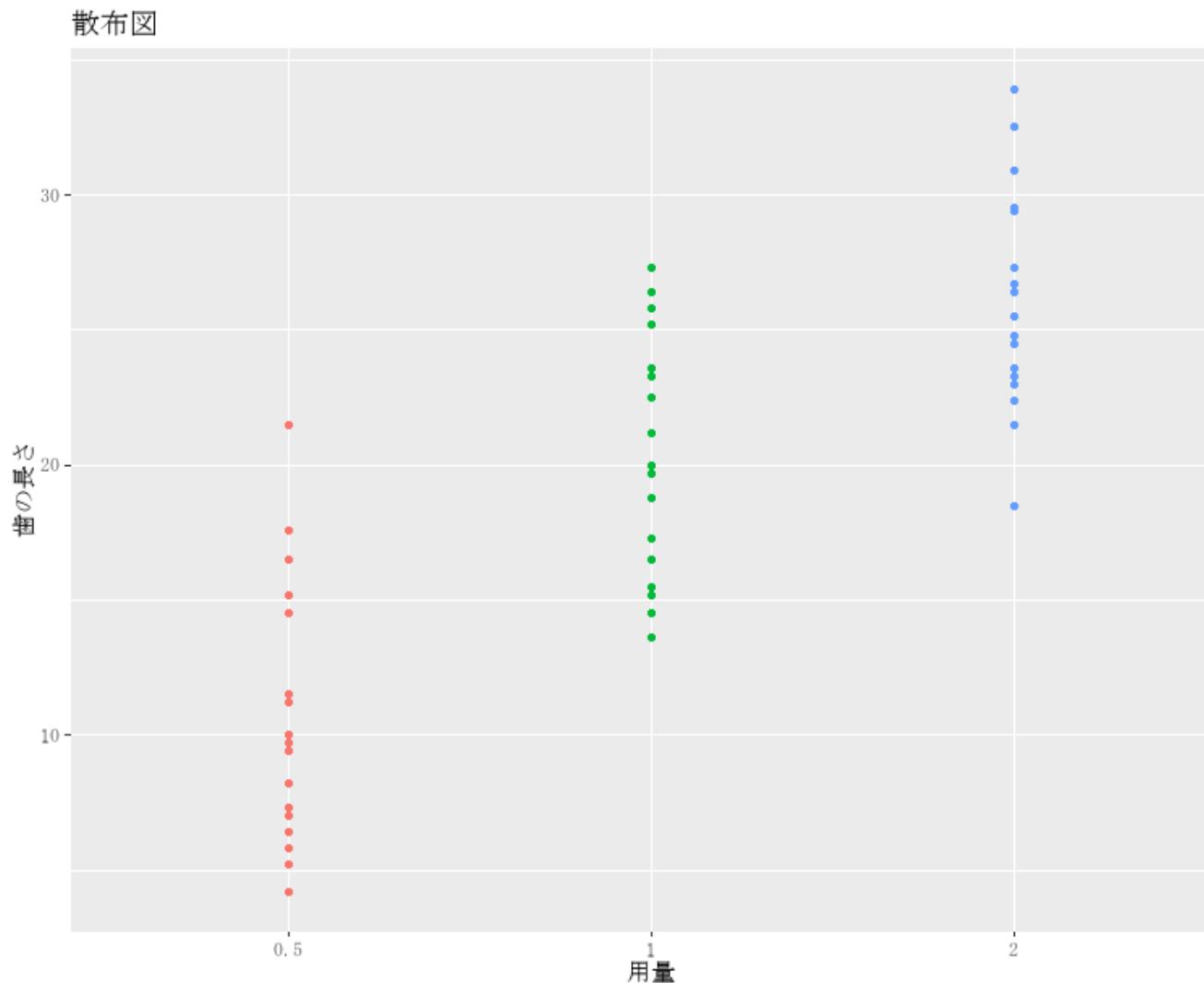
- *ggplot2* の仕組みは体系だっているように見えますが、実際にいくつかグラフを作成すると、結構困ることが出できます（理屈に合っていないと思われるものも…）
- 個人的には以下の流れで習得されるのが宜しいかと思います
 1. 本資料の基本的な事項を押さえる（基本的な事項を知らずに 2. 以降に進むと混乱します）
 2. 実際にグラフを描いてみる、事例集としては以下がお勧め
 - R Graphics Cookbook, 2nd edition: <https://r-graphics.org/>
 - Winston Chang 著、石井 弓美子 他翻訳、Rグラフィックスクックブック 第2版（オンライン）
 3. エラーが出た場合はヘルプや下記 reference を参照する
 - <https://ggplot2.tidyverse.org/reference/index.html>
 4. それでも分からぬときは Google や生成 AI に質問
⇒ 良く仕様が変更されるため（たまに破壊的仕様変更も）資料の作成時期に注意

おまけ②: 日本語フォントを使ったグラフ作成 + PDF 出力例

- 事前にパッケージ showtext、Cairo、sysfonts をインストールする

```
> # install.packages("showtext")
> # install.packages("Cairo")
> library(showtext)
> library(Cairo)
>
> font_add(family="MS Mincho", regular="msmincho.ttc")
> showtext_auto()
>
> point <- ggplot(tg, aes(x=dose_c, y=len, color=dose_c)) +
+   geom_point() +
+   labs(title="散布図", x="用量", y="歯の長さ", color="用量") +
+   theme_gray(base_family="MS Mincho")
> print(point)
>
> CairoPDF("C:/temp/sample.pdf", width=6, height=6)
> print(point)
> dev.off()
```

おまけ②: 日本語フォントを使ったグラフ作成 + PDF 出力例



おまけ②: 日本語フォントを使ったグラフ作成 + PDF 出力例

- Windows の場合で使用できる「主な」日本語フォントは以下:

```
> # install.packages("sysfonts")
> library(sysfonts)
> x <- subset(font_files(), grepl("Hiragino|Osaka|Yu|MS|Meiryo", family, ignore.case=TRUE))
> select( distinct(x, family, .keep_all=TRUE), file, family)

      file           family
1  arialuni.ttf     Arial Unicode MS
3   meiryo.ttc       Meiryo
4 msgothic.ttc      MS Gothic
5 msmincho.ttc      MS Mincho
13 YuGothB.ttc      Yu Gothic
14 YuGothL.ttc      Yu Gothic Light
15 YuGothM.ttc      Yu Gothic Medium
16   yumin.ttf       Yu Mincho
17 yumindb.ttf     Yu Mincho Demibold
18  yuminl.ttf      Yu Mincho Light
```

おまけ③: 数学関数のプロット

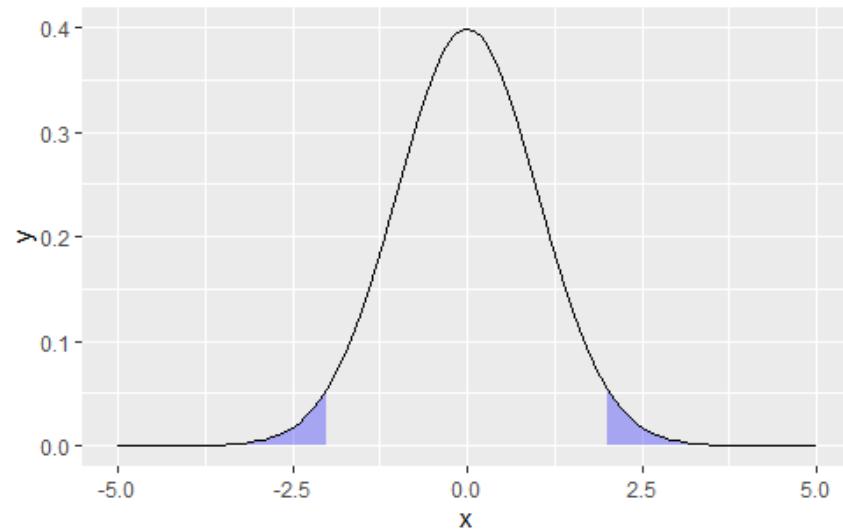
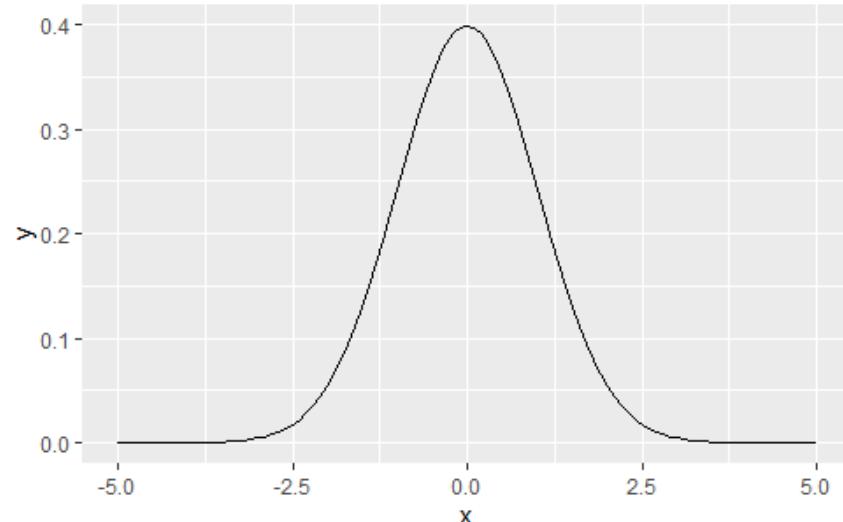
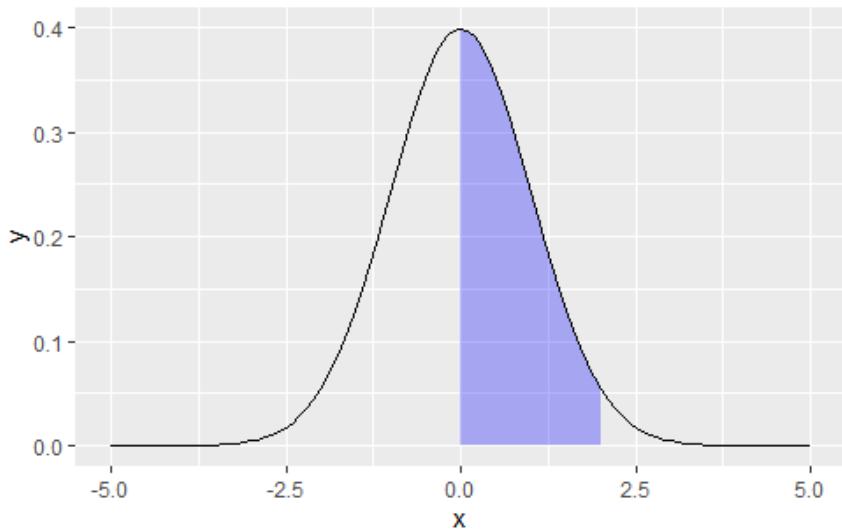
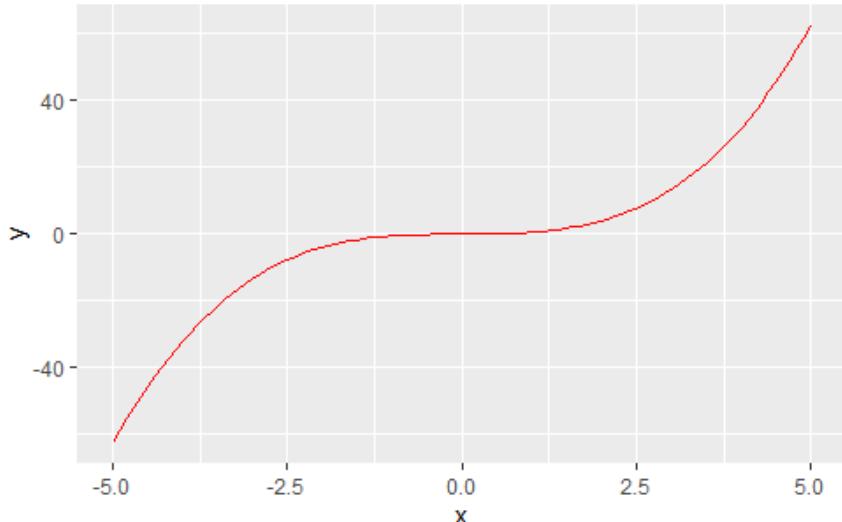
```
> f <- function(x) x^3/2
> ggplot(data.frame(x=c(-5, 5)), aes(x)) +
+   geom_function(fun=f, color="red")

> ggplot(data.frame(x=c(-5, 5)), aes(x)) +
+   geom_function(fun=dnorm, args=list(mean=0, sd=1))

> xlimit <- function(x) { y <- dnorm(x); y[x<0 | x>2] <- NA; y }
> ggplot(data.frame(x = c(-5, 5)), aes(x)) +
+   stat_function(fun=xlimit, geom="area", fill="blue", alpha=0.3) +
+   geom_function(fun=dnorm)

> q      <- qnorm(0.975)
> mydnorm <- function(x) ifelse(x < -q | x > q, dnorm(x), NA)
> ggplot(data.frame(x=c(-5, 5)), aes(x)) +
+   stat_function(fun=mydnorm, geom="area", fill="blue", alpha=0.3) +
+   geom_function(fun= dnorm, args=list(mean=0, sd=1))
```

おまけ③: 数学関数のプロット





本日のメニュー

- Windows 版 R / RStudio のセットアップ方法 ← 当日までに自習
- R の基礎知識 ← 当日までに自習
- パッケージと作業ディレクトリ ← 当日までに自習
- 変数とベクトル
- ベクトルの型
- 関数とプログラミング
- データ加工とパイプ演算子
- グラフ作成(ggplot2)
- その他

関数の定義を見る

- 関数がどのような式で定義されているかを確認するには、関数の名前のみ(括弧なし)を入力すればよい

```
> f <- function (x) {  
+   y <- 2*x  
+   return(y)  
+ }  
> f  
function (x) {  
  y <- 2*x  
  return(y)  
}
```

関数の定義を見る

- 元々 R に用意されている関数の定義を見る場合も、基本的には同じ

```
> sd
function (x, na.rm = FALSE)
sqrt(var(if (is.vector(x) || is.factor(x)) x else as.double(x),
na.rm = na.rm))
<bytecode: 0x0000000010068b20>
<environment: namespace:stats>
```

- ただ、多くの関数は、上記の方法だけでは定義を見ることが出来ない

```
> mean
function (x, ...)
UseMethod("mean")
<bytecode: 0x0000000006ef7738>
<environment: namespace:base>
```

関数の定義を見る

- 前頁の様な問題が起きた場合、関数定義を見るための作業・試行錯誤が必要になる
- 1. パッケージが GitHub にあれば、その「R」フォルダのプログラムを確認
例: dplyr であれば <https://github.com/tidyverse/dplyr/tree/main/R>
- 2. CRAN / PC にインストールされているパッケージのフォルダのソースプログラムを確認
(少々面倒?)
- 3. 1. ~ 2. 以外の方法としては以下(今回は ① と ② を紹介)
 - ① 通常の関数 (S3 クラスの関数) は、関数 `methods()` を使用することで表示される関数名を入力する
 - ② 特定のパッケージに含まれている関数は「...」を使って定義を見る
 - ③ S4 クラスの関数は、関数 `showMethods()` と `getMethod()` で定義を見る
(マニアックなのとS4クラスの関数を扱うことが少ないので、詳細は割愛)

① 関数 *methods()* を用いる方法

- 多くの場合、XXX.default という名前の関数に、目的の関数定義が入っていることが多い

```
> methods(mean)
[1] mean.Date      mean.default   mean.difftime mean.POSIXct
[5] mean.POSIXlt
see '?methods' for accessing help and source code
> mean.default    # mean.default の定義を見る
function (x, trim = 0, na.rm = FALSE, ...)
{
  if (!is.numeric(x) && !is.complex(x) && !is.logical(x)) {
    warning("argument is not numeric or logical: returning NA")
    return(NA_real_)
  }
  .....
}
<bytecode: 0x000000006efd020>
<environment: namespace:base>
```

① 関数 *methods()* を用いる方法

- 関数 *methods* の結果、* が付いている関数は前頁の方法が適用出来ない
→ 関数 *getAnywhere()* を用いる

```
> methods(as.matrix)
[1] as.matrix.data.frame as.matrix.default      as.matrix.dist*
[4] as.matrix.ftable*    as.matrix.noquote    as.matrix.POSIXlt
[7] as.matrix.raster*
see '?methods' for accessing help and source code

> as.matrix.dist
Error: object 'as.matrix.dist' not found

> getAnywhere(as.matrix.dist)
A single object matching 'as.matrix.dist' was found
.....
function (x, ...)
{
  size <- attr(x, "Size")
  df <- matrix(0, size, size)
  .....
```

* * が付いている関数はパッケージからエクスポートされておらず定義が確認できない、そこで 関数 *getAnywhere()* を用いることでこの様な関数の定義(やその他の情報)を探索する、関数 *getS3method("関数名", "パッケージ/クラス名")* でも可

② :: を用いる方法

- パッケージに含まれる関数は「パッケージ名::関数名」を使って定義を見ることが出来る(失敗することもある…)

```
> t.test          # 定義は見られない  
function (x, ...)  
UseMethod("t.test")  
<bytecode: 0x000000008514f48>  
<environment: namespace:stats>  
  
> stats:::t.test      # これで見られる関数もあるが今回はダメ  
function (x, ...)  
UseMethod("t.test")  
<bytecode: 0x000000008514f48>  
<environment: namespace:stats>
```

② :: を用いる方法

- 関数 methods() と「::」を組み合わせることで、ほとんどの関数は定義を見ることが出来る(それでも失敗することもある……)

```
> methods(t.test)          # methods() で検索 → t.test.default??
[1] t.test.default* t.test.formula*
see '?methods' for accessing help and source code

> t.test.default          # これでも見られない
Error: object 't.test.default' not found

> stats:::t.test.default # :::付きで閲覧成功
function (x, y=NULL, alternative=c("two.sided", "less", "greater"),
  mu=0, paired=FALSE, var.equal=FALSE, conf.level=0.95, ...)
{
  alternative <- match.arg(alternative)
  if (!missing(mu) && (length(mu) != 1 || is.na(mu)))
  ....
```

おまけ: 「::」演算子

- いろいろパッケージをインストールしていると、同名の関数が存在することがある（後に呼び出された関数の定義が優先）
- その場合、「パッケージ名::関数名」で、パッケージ名を明示して関数を呼び出すことが出来る

```
> library(dplyr)
以下のオブジェクトは 'package:base' からマスクされています:
  intersect, setdiff, setequal, union
> intersect
function (x, y, ...)
UseMethod("intersect")
<environment: namespace:dplyr>
> base::intersect
function (x, y)
{
  y <- as.vector(y)
  unique(y[match(as.vector(x), y, 0L)])
}
<bytecode: 0x00000000103d8510>
<environment: namespace:base>
```

おまけ: 「::」演算子

- ②の方法は、本来は以下の手順で関数定義を見るべき
 - 「パッケージ名::関数名」で、パッケージの関数定義を見る
 1. で失敗する場合として、パッケージからエクスポートされていない関数の定義を見ようとした場合が挙げられる。そのような場合は「パッケージ名:::関数名」で、パッケージの関数定義を見る
- ※ ただ、パッケージのエクスポートの有無に関わらず「パッケージ名:::関数名」で関数定義が確認可

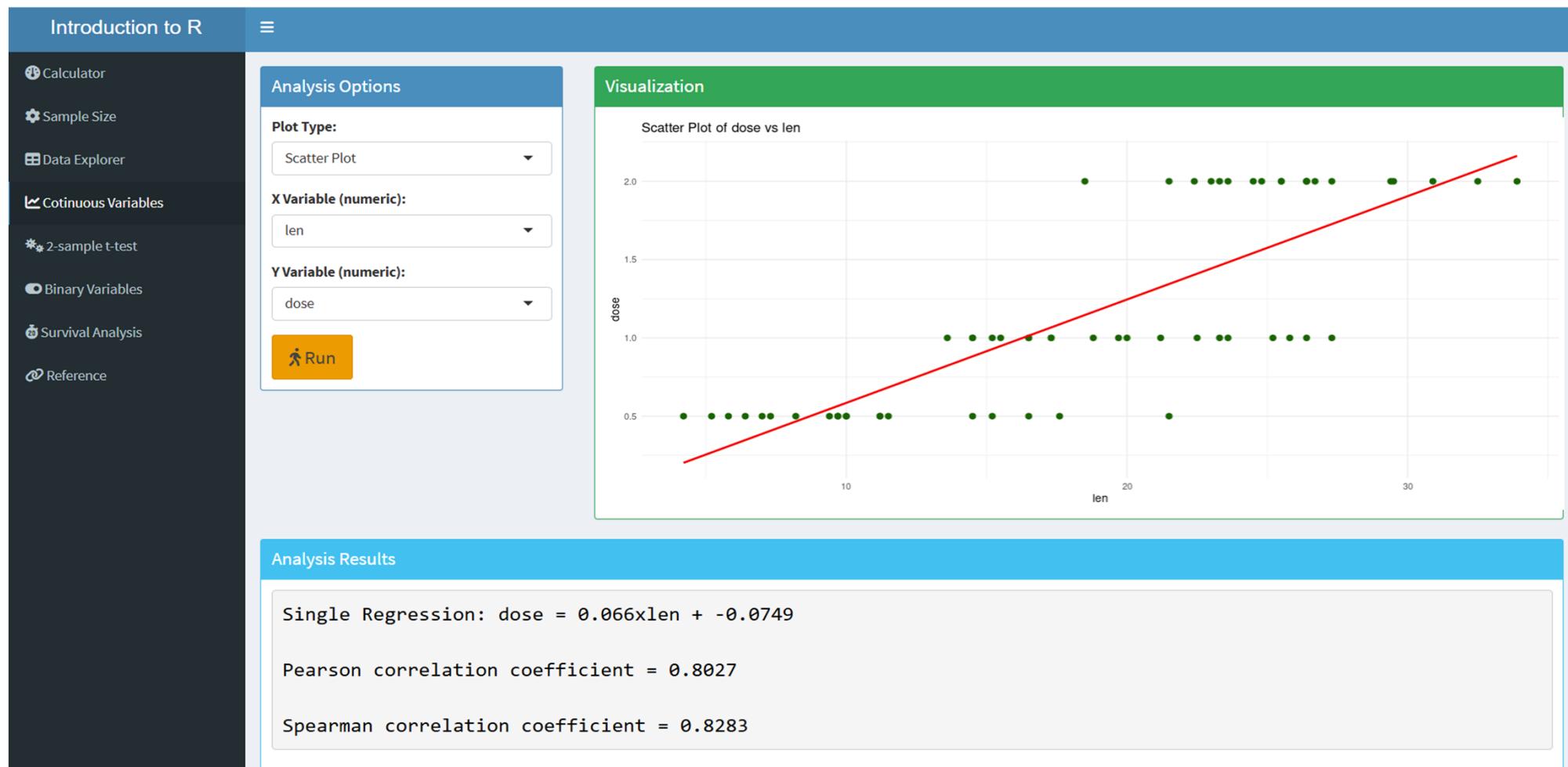
```
> base:::which
function (x, arr.ind = FALSE, useNames = TRUE) {
  wh <- .Internal(which(x))
  if (arr.ind && !is.null(d <- dim(x)))
    arrayInd(wh, d, dimnames(x), useNames = useNames)
  else wh
}
<bytecode: 0x0000000065249f8>
<environment: namespace:base>
> base::::which
function (x, arr.ind = FALSE, useNames = TRUE) {
  wh <- .Internal(which(x))
  if (arr.ind && !is.null(d <- dim(x)))
    arrayInd(wh, d, dimnames(x), useNames = useNames)
  else wh
}
<bytecode: 0x0000000065249f8>
<environment: namespace:base>
```

演習 11

1. 条件分岐用の関数 `ifelse()` の中身を確認してください
2. 分位数を求める関数 `quantile()` の中身を確認してください
3. 中央値を求める関数 `median()` の中身を確認してください
4. 時間がある方は、関数 `median()` のプログラムを「自分が理解出来るレベル」までプログラムを簡素化して下さい。

パッケージ「shiny」

- Rでshinyアプリを作成する例：<https://nobuo-funao.shinyapps.io/recipe/>





パッケージ「shiny」

- posit 社(旧 Rstudio 社)が開発した、R で Web アプリケーションを作るためのパッケージ
- shiny を使えば、アプリ開発の知識なしで簡単に Web アプリケーションが作成できる
 - R で簡単アプリ作成～shiny 超入門: <https://nfunao.web.fc2.com/files/shiny.pdf>
 - shiny 演習 (1): https://nfunao.web.fc2.com/files/shiny2_1.pdf
 - shiny 演習 (2): https://nfunao.web.fc2.com/files/shiny2_2.pdf
- 後ほど、西川 寛来さんよりパッケージ「teal」(shiny の進化版)のご説明があります

最後に

- 「メモ」「アンチヨコ」「自分用のコマンド集」を作つて下さい
- ヘルプの見方、分からぬことが出てきたときの検索方法を身につけて下さい
(ヘルプ、Google、生成 AI、R に関する HP や書籍等、作成時期に注意)
⇒ 分からぬときに自分で何とかする癖をつけて下さい
- 最初のうちはエラーが出ても気にしないで下さい
- 「人様が書いたプログラムを実行」⇒「プログラムの一部を修正して実行」という作業に慣れて下さい

参考文献

- **R の基礎 (tidyverse / pharmaverse は含まず)**
 - 簡潔: R で学ぶプログラミングの基礎の基礎 (カットシステム)
 - 辞書: The R Tips 第 3 版 (オーム社)
- **R の基礎 (tidyverse 前提)**
 - 松村 優哉 他著、改訂 2 版 R ユーザのための RStudio [実践] 入門 (技術評論社)
- **R / ggplot2 でグラフ作成**
 - ggplot2, 3rd edition : <https://ggplot2-book.org/>
 - R Graphics Cookbook, 2nd edition: <https://r-graphics.org/>
 - Winston Chang 著、石井 弓美子 他翻訳、R グラフィックス クックブック 第 2 版 (オライリージャパン)
- **R でデータ解析の基礎の基礎**
 - R によるデータ分析のレシピ (オーム社)



本日のメニュー

- Windows 版 R / RStudio のセットアップ方法
- R の基礎知識
- パッケージと作業ディレクトリ
- 変数とベクトル
- ベクトルの型
- 関数とプログラミング
- データ加工とパイプ演算子
- グラフ作成(ggplot2)
- その他
- 宿題！！！

宿題 1: 変数とベクトル

- 以下の 5 つのデータを変数 y に格納して下さい

1

2

3

4

5

- 変数 y の値と変数 Y (大文字)の値を表示して下さい
1. で作成した変数 y の平均と標準偏差を求めて下さい
- 変数 y の後ろに値 9 を結合し、結果を変数 z に格納して下さい
4. で作成した変数 z の 6 番目の値を表示して下さい
- 変数 z の 3 番目の値を 7 に変更して下さい
- 「101、102、…、200」と、101 から順に 1 ずつ増えるベクトル x を作成して下さい
- 変数 x の奇数番目の要素のみを取り出し、変数 $odds$ に代入して下さい
- 変数 $odds$ の長さ(要素の数)を求めてください

宿題 2: ベクトルの型

- 以下の命令を実行し、文字型ベクトル x と因子ベクトル y を作成してください

```
> x <- c("L", "M", "H", "L")
> ( y <- factor(x, levels=c("L", "M", "H")) )
[1] L M H L
Levels: L M H
```

- 関数 table(y) にて、因子ベクトル y の度数分布表を作成してください
- 関数 nlevels(y) にて、因子ベクトル y の水準数(カテゴリ数)を求めてください
- 以下の命令を実行し、因子ベクトル y の水準(カテゴリ)のラベルを変更してください

```
> levels(y)          # 現在の水準を確認
[1] "L" "M" "H"
> levels(y) <- 1:3 # 水準を変更
```

- 以下の命令を実行して下さい

```
> factor(c(1, 2, 3, 1)) == y
```

宿題 2: ベクトルの型

- 以下の命令を実行し、因子ベクトル z を作成してください

```
> ( z <- factor(x, levels=c("L", "M")) )  
[1] L      M      <NA>  L  
Levels: L M
```

- 関数 addNA(z) にて、因子ベクトル z に「 NA 」という水準を追加してください
- 因子ベクトル z の 2 番目の要素を取り出してください
- 因子ベクトル z の 1 番目と 4 番目の要素を取り出し、新たな変数 u に代入してください
- 以下の命令を実行して下さい

```
> droplevels(u)
```



宿題 3: 関数とプログラミング

1. ユーザーが正の整数を 1 つ指定すると、「3 の倍数」のときは文字「X」を、「5 の倍数」のときは文字「Y」を、「3 の倍数」かつ「5 の倍数」のときは文字「Z」を、上記以外の場合はそのままの値を返すような関数 nabe() を作って下さい。
2. 1 以上の奇数 n を入れると 1 から n のうち奇数だけを掛け算した二重階乗($n!!$)を返す関数 myprod2() を定義して下さい
→ 例えば、 $n=7$ のときは「 $1 \times 3 \times 5 \times 7 = 105$ 」となります。
3. 正の整数値 n(奇数かもしれませんし、偶数かもしれません)を入れると、n が奇数の場合は 1 から n のうち奇数だけを掛け算した値、n が偶数の場合は 1 から n のうち偶数だけを掛け算した値を返す関数myprod3() を定義して下さい。
4. 興味のある方は Google Blockly Games を解いてみて下さい
<https://blockly.games/maze?lang=ja>

宿題 4: データ加工とパイプ演算子

1. 以下のプログラムを実行して、データフレーム y を作成してください

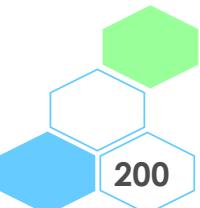
```
> id      <- c(1, 2, 3, 4, 5)
> sex     <- c("F", "F", "M", "M", "M")
> height   <- c(160, 165, 170, 175, 180)
> weight    <- c( 50,  65,  60,  55,  70)
> ( mydata2 <- data.frame(ID=id, SEX=sex, WEIGHT=weight) )
```

2. データフレーム mydata2 に対して以下の処理を行ってください

- 新規変数 HEIGHT として、変数 height の値を代入(変数名は HEIGHT)
- 「 sex = F 」の行のみ抽出、変数を ID と HEIGHT に絞る

3. データフレーム mydata2 に対して、パイプ演算子を用いて以下の処理を 1 タスクで行ってください

- 新規変数 HEIGHT として、変数 height の値を代入(変数名は HEIGHT)
- 「 SEX = F 」の行のみ抽出、変数を ID と HEIGHT に絞る
- 関数 summary() を適用し、要約統計量を算出

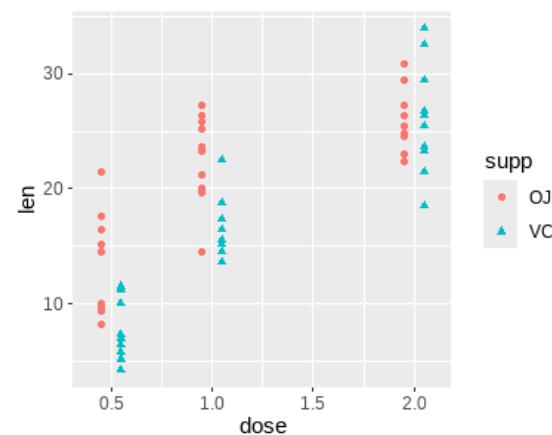
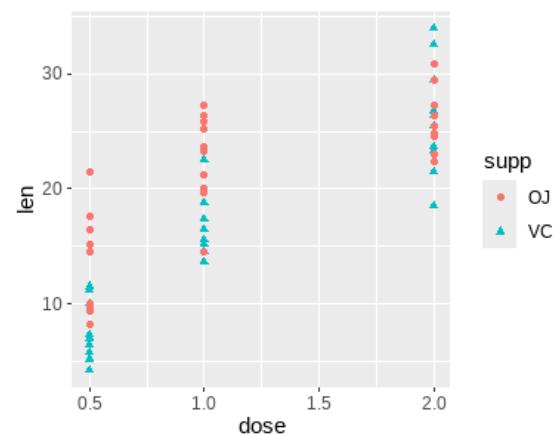


宿題 5: グラフ作成

1. ggplot2 のグラフの土台を下記条件にて作成し、変数 base に代入して下さい
 - データフレーム ToothGrowth に関するグラフ
 - 引数 x に変数 dose、引数 y に変数 len を指定
 - 変数 supp を引数 color と shape に紐付け
2. 以下の命令で、1. を土台とした散布図(左図)を作成して下さい

```
base + geom_point()
```

3. 右図のように、プロット点の重なりを解消して下さい



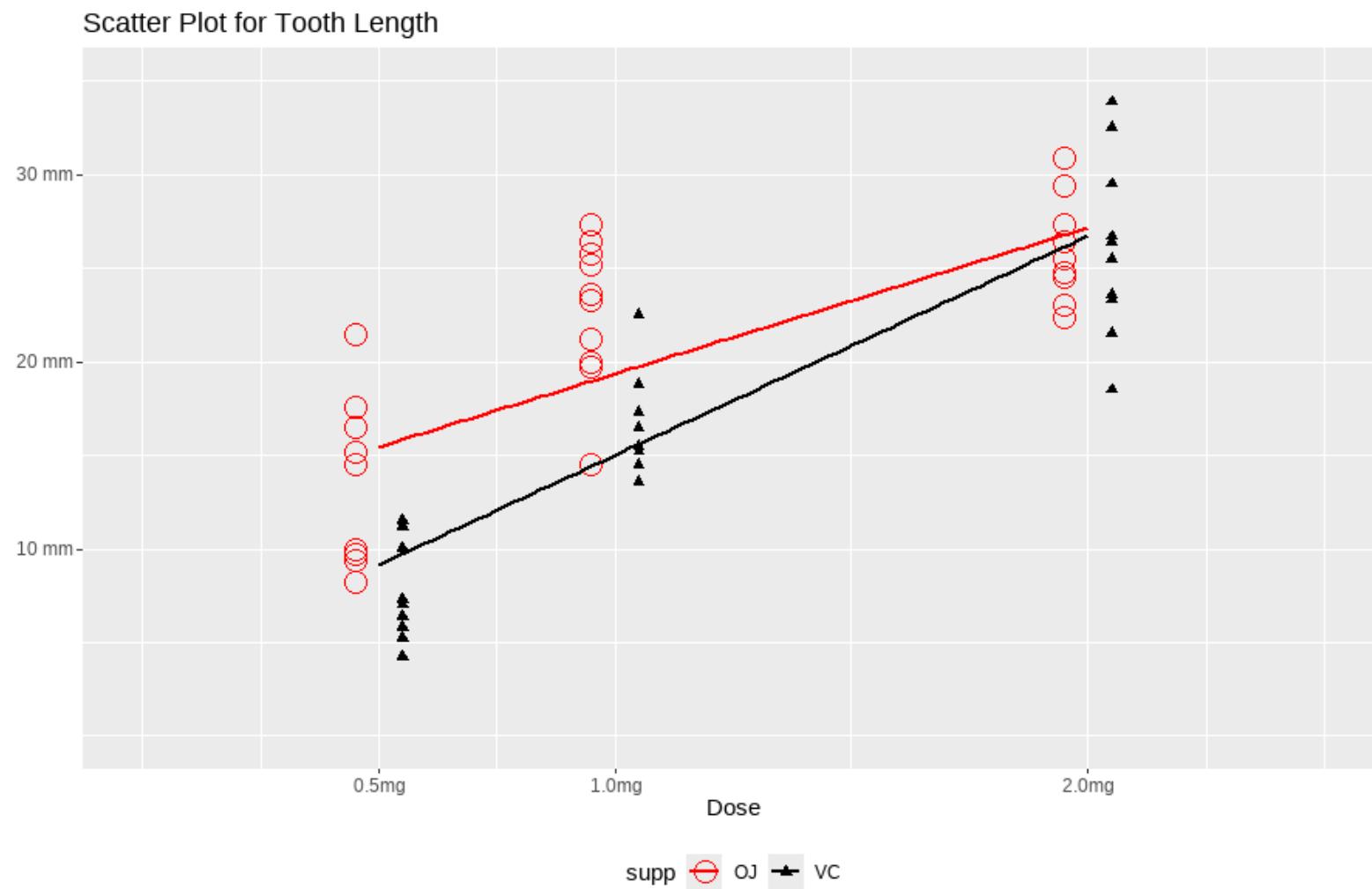
宿題 5: グラフ作成

4. 3. で実行したプログラムについて、関数 `geom_point()` に引数 `aes(size=supp)` を追加し、実行して下さい
5. 4. で実行したプログラムを修正することにより、グラフに下記の変更を加えて下さい

	SUPP="OJ"	SUPP="VC"
プロット点の色	赤色	黒色
プロット点の形	○	▲
プロット点の大きさ	少し大きめ	少し小さめ

6. 5. で実行したプログラムを修正することにより、グラフに下記の変更を加えて下さい
 - 回帰直線をレイヤー・関数 `geom_smooth(method=lm, se=F)` にて追記
 - 凡例: グラフの下、x 軸ラベル: "Dose (mg)"、y 軸ラベル: なし
 - タイトル: グラフの上真ん中に "Scatter Plot for Tooth Length"
 - 横軸の範囲: 0 ~ 2.5、縦軸の範囲: 0 ~ 35
 - 見本は次頁のグラフです

宿題 5: グラフ作成



宿題 6: 関数の定義を見る

1. 関数 `seq()` のヘルプを参照して下さい
2. 関数 `seq()` の関数定義を閲覧して下さい
3. 関数 `library()` 等を用いてパッケージ `dplyr` を呼び出して下さい
4. 関数 `help(union, package="base")` にて、
パッケージ `base` の関数 `union()` のヘルプを参照して下さい
5. パッケージ `dplyr` の関数 `union()` のヘルプを参照して下さい
6. パッケージ `base` の関数 `union()` の関数定義を閲覧して下さい
7. ベクトル `c(1,2,3)` とベクトル `c(2,3,4)` の和集合を求めて下さい
8. 関数 `wilcox.test()` の関数定義を閲覧して下さい



– End of File –