

R言語事始め

プログラミング言語「R」とは



[\[Home\]](#)

Download

[CRAN](#)

R Project

[About R](#)

[Logo](#)

[Contributors](#)

[What's New?](#)

[Reporting Bugs](#)

[Conferences](#)

[Search](#)

[Get Involved: Mailing Lists](#)

[Get Involved: Contributing](#)

[Developer Pages](#)

[R Blog](#)

R Foundation

[Foundation](#)

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News

- [R version 4.5.0 \(How About a Twenty-Six\) prerelease versions](#) will appear starting Tuesday 2025-03-11. Final release is scheduled for Friday 2025-04-11.
- [R version 4.4.3 \(Trophy Case\)](#) has been released on 2025-02-28.
- The [useR! 2025](#) conference will take place at Duke University, in Durham, NC, USA, August 8-10.
- We are deeply sorry to announce that our friend and colleague Friedrich (Fritz) Leisch has died. [Read our tribute to Fritz here](#).
- [R version 4.3.3 \(Angel Food Cake\)](#) (wrap-up of 4.3.x) was released on 2024-02-29.
- You can support the R Foundation with a renewable subscription as a [supporting member](#).

R言語の開発を行っている [The R Project for Statistical Computing](#) のトップページ。Rに関する基本的な情報のほか、最新のリリース情報やマニュアルの閲覧ができる。

- 統計解析やデータ分析に特化したオープンソースのプログラミング言語
- 強力な可視化ライブラリ([ggplot2](#) など)や統計モデリング機能が充実しており、研究・データサイエンスの分野で広く利用されている

R言語の起源と歴史



R言語のロゴ



Ross Ihaka



Robert Clifford Gentleman

(写真はwikipediaより引用)

- Ross IhakaとRobert Clifford Gentleman(オークランド大学)によって、S言語(AT&Tベル研究所が開発)という統計解析言語を参考に1990年代前半に開発
- 1995年にGNU GPLライセンスのもとで公開され、オープンソースプロジェクトとして成長
- 1997年には統計学者のコミュニティ [R Core Team](#) (現在の公式サイト)が結成され、公式な開発がすすめられた。

R言語の起源と歴史



R言語のロゴ



Ross Ihaka



Robert Clifford Gentleman

(写真はwikipediaより引用)

- 2000年代以降, 統計分析やデータサイエンス分野で広く採用されるようになり, 多数のパッケージが開発される
- 特に, [tidyverse](#) (Hadley Wickhamらによる) などのエコシステムが拡充され, データ解析・可視化がより強力になった
- 現在も進化を続け, 機械学習やベイズ統計などの分野でも活用されている

R言語の基本的な特徴

- **統計解析に特化**：Rは統計解析・データ分析向けに設計されており、多様な統計手法を標準でサポート
- **オープンソース**：無料で利用可能. ソースコードが公開されているため誰でもカスタマイズ可能
- **インタプリタ言語**：**コードを(逐次解釈しながら)対話的に実行できる**ため、データ分析の試行錯誤がしやすい

対話的な実行の例

```
# 1行ずつ実行可能  
a <- 5  
b <- 2  
c <- a * b  
print(c) # 10
```

- 上から1行ずつ実行可能
- 実行後すぐに `c` の値を確認できる

R言語の基本的な特徴

- **統計解析に特化**：Rは統計解析・データ分析向けに設計されており、多様な統計手法を標準でサポート
- **オープンソース**：無料で利用可能。ソースコードが公開されているため誰でもカスタマイズ可能
- **インタプリタ言語**：コードを(逐次解釈しながら)対話的に実行できるため、データ分析の試行錯誤がしやすい

逐次解釈の例

```
x <- 10      # ここまでは問題なく実行される
y <- x + z    # z が未定義なのでここでエラー発生
print("Hello, R!") # ここには到達しない
```

実行結果(エラー発生)

```
Error in x + z : object 'z' not found
```

- `x <- 10` は正常に実行される
- `y <- x + z` で `z` が未定義のため エラーが発生
- エラー発生後以降のコードは実行されない

(補足) インタープリタ言語であることのデメリット

- 実行速度が遅い
 - インタープリタ言語はコードを逐次解釈しながら実行するため、コンパイル済みの言語(C, C++, Java など)に比べて処理速度が遅くなりがち
 - ループ処理 (for, while) や大規模データ処理ではパフォーマンスが低下することがある

対策 C++と連携 ([Rcpp](#)): 計算量の多い部分を C++で記述し高速化

- メモリ効率が悪い
 - Rはメモリ内でデータを処理するため、大規模データの処理時にメモリ消費が大きくなる
 - 例えば数百万行のデータを扱う場合、RAMの消費が激しくなり処理が遅延・クラッシュする可能性がある

対策 データベース連携 ([DBI](#), [data.table](#)): メモリ負荷を減らし大規模データ処理を効率化

- 並列・分散処理が標準では弱い
 - Rはシングルスレッドでの実行が基本であり、並列処理のサポートは限定的

対策 [parallel](#), [future](#) などの並列計算用パッケージを利用して処理速度を向上

(補足) インタープリタ言語であることのデメリット

- エラーハンドリングが弱い
 - Rは実行時にエラーが発生すると即座に停止することが多く、例外処理 (tryCatch) を適切に記述しないとエラー耐性が低い
 - コンパイル型言語ではコンパイル時にエラーが検出されるが、Rでは実行時にエラーが出るためバグが見つかりにくい
- コードの最適化が難しい
 - インタープリタ言語ではコンパイラによる最適化が行われないため、コードの書き方によってはパフォーマンスが大幅に低下 することがある
 - ベクトル化 (apply, lapply, map など) を意識しないとループが遅くなりがち

対策 ベクトル化 ([apply](#), [dplyr](#)): ループを避け、最適化された関数を利用

インタープリタ言語であるRのメリット・デメリットを正しく認識認識して、快適なRコーディングライフを送りましょう

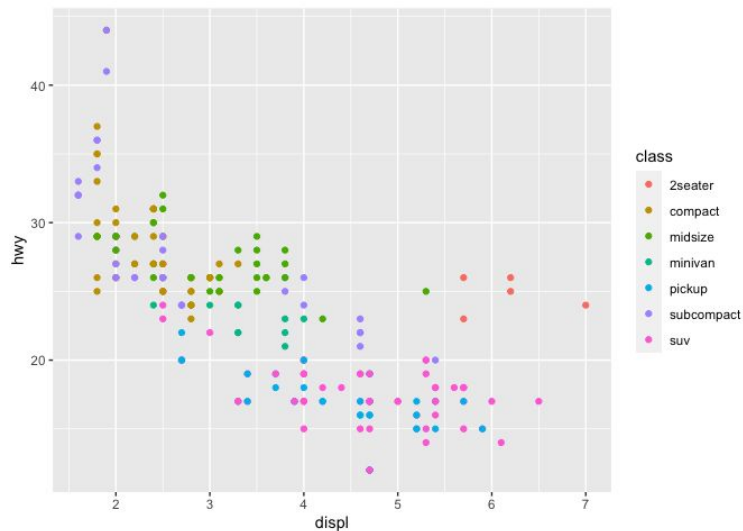
R言語の特徴: 豊富なパッケージとエコシステム

- [CRAN \(Comprehensive R Archive Network\)](#) による数千種類のパッケージ
- [tidyverse](#): [ggplot2](#), [dplyr](#) などデータ操作・可視化を効率化するパッケージ群
- 機械学習ライブラリ: randomForest, xgboost など多様なML手法をサポート

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](#)

A3	Accurate, Adaptable, and Accessible Error Metrics for Predictive Models
AalenJohansen	Conditional Aalen-Johansen Estimation
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
abclass	Angle-Based Large-Margin Classifiers
ABCOptim	Implementation of Artificial Bee Colony (ABC) Optimization
abcrf	Approximate Bayesian Computation via Random Forests
abcrlda	Asymptotically Bias-Corrected Regularized Linear Discriminant Analysis
abctools	Tools for ABC Analyses
abd	The Analysis of Biological Data
abdiv	Alpha and Beta Diversity Measures
abe	Augmented Backward Elimination
aberrance	Detect Aberrant Behavior in Test Data
abess	Fast Best Subset Selection



CRANに登録されたパッケージ

ggplot2による可視化の例

2. R言語超入門

RStudioを使ってみる

The screenshot displays the RStudio environment with the following components:

- Editor (Left):** Contains R code for installing ggplot2, creating a data frame 'df' with 10 observations, and plotting it as a scatter plot. A red label "エディタ" (Editor) is overlaid on this section.
- Environment (Top Right):** Lists objects in the global environment, including 'df' (10 obs. of 3 variables), 'hess', 'sample', 'solu', 'X', 'xsol', 'xsol1', 'xsol2', and 'z'. A red label "オブジェクト内容" (Object Content) is overlaid on this section.
- Console (Bottom Left):** Shows the execution of the R code from the editor. A red label "コンソール" (Console) is overlaid on this section.
- Plots (Bottom Right):** Displays a scatter plot titled "scatter plot" with 'X-axis' and 'Y-axis' labels. A red label "プロット表示" (Plot Display) is overlaid on this section.

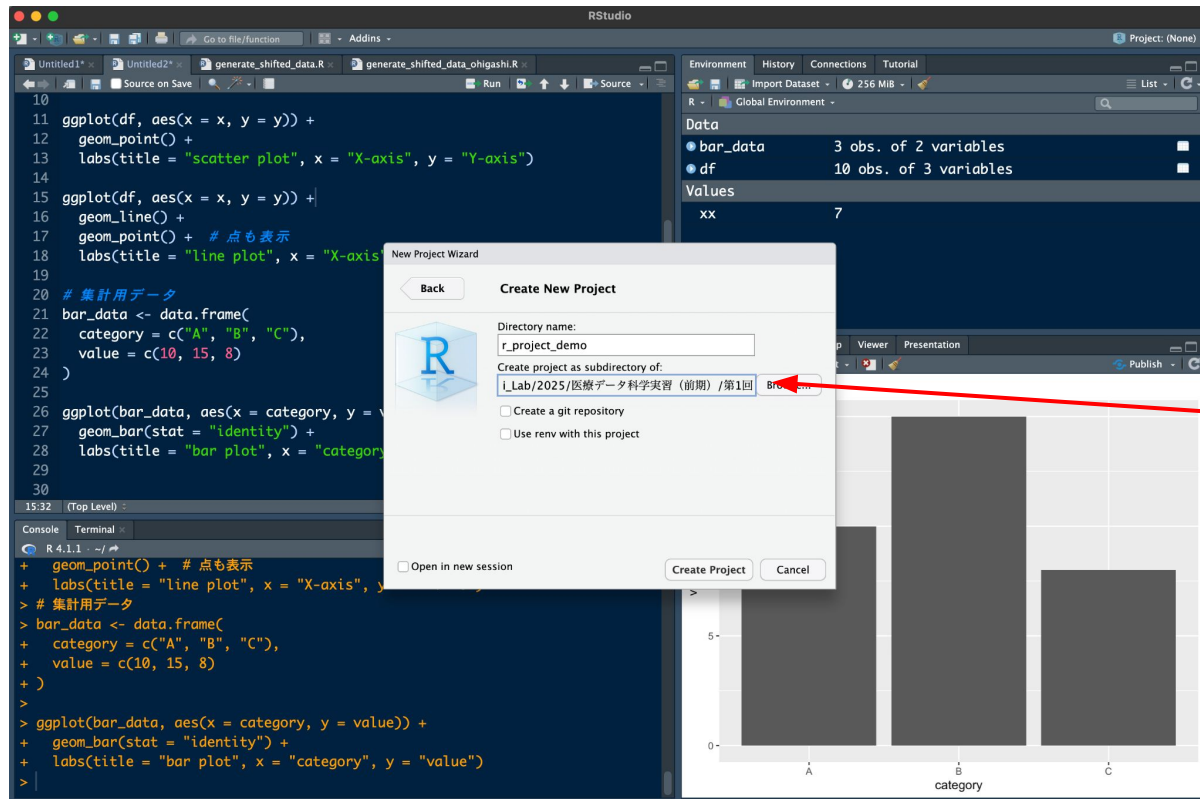
```
1 install.packages("ggplot2")
2 library(ggplot2)
3
4 # サンプルデータの作成
5 df <- data.frame(
6   x = 1:10,
7   y = c(3, 5, 4, 6, 8, 7, 9, 10, 9, 12),
8   category = rep(c("A", "B"), each = 5)
9 )
10
11 ggplot(df, aes(x = x, y = y)) +
12   geom_point() +
13   labs(title = "scatter plot", x = "X-axis", y = "Y-axis")
14 |
```

Console output:

```
R 4.1.1 ~ /
> # サンプルデータの作成
> df <- data.frame(
+   x = 1:10,
+   y = c(3, 5, 4, 6, 8, 7, 9, 10, 9, 12),
+   category = rep(c("A", "B"), each = 5)
+ )
> ggplot(df, aes(x = x,
+   geom_point() +
+   labs(title = "散佈図の例", x = "X軸", y = "Y軸")
> ggplot(df, aes(x = x, y = y)) +
+   geom_point() +
+   labs(title = "scatter plot", x = "X-axis", y = "Y-axis")
>
```

x	y
1	3
2	5
3	4
4	6
5	8
6	7
7	9
8	10
9	9
10	12

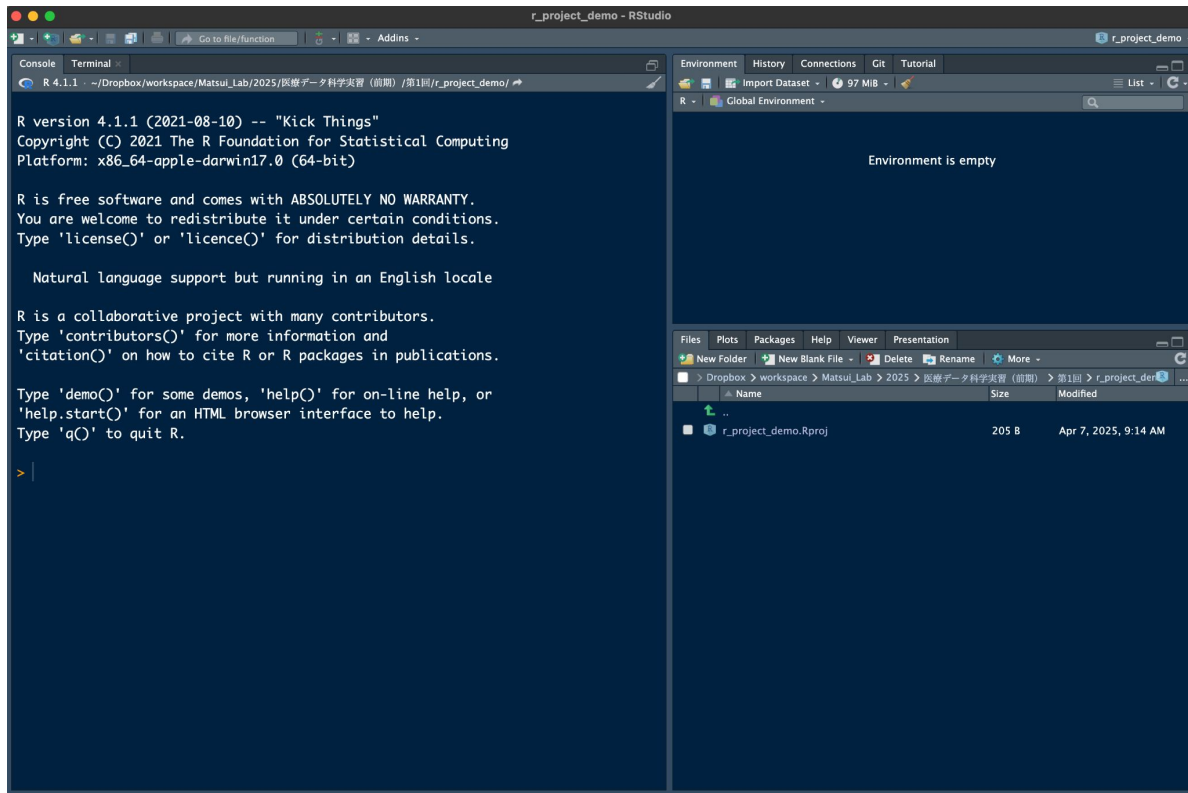
RStudioを使ってみる



新しいプロジェクトの始め方:

1. File -> New Project
2. New Directory
3. 新しいプロジェクトを置く親ディレクトリ(下)と作業ディレクトリ(上)を指定

RStudioを使ってみる



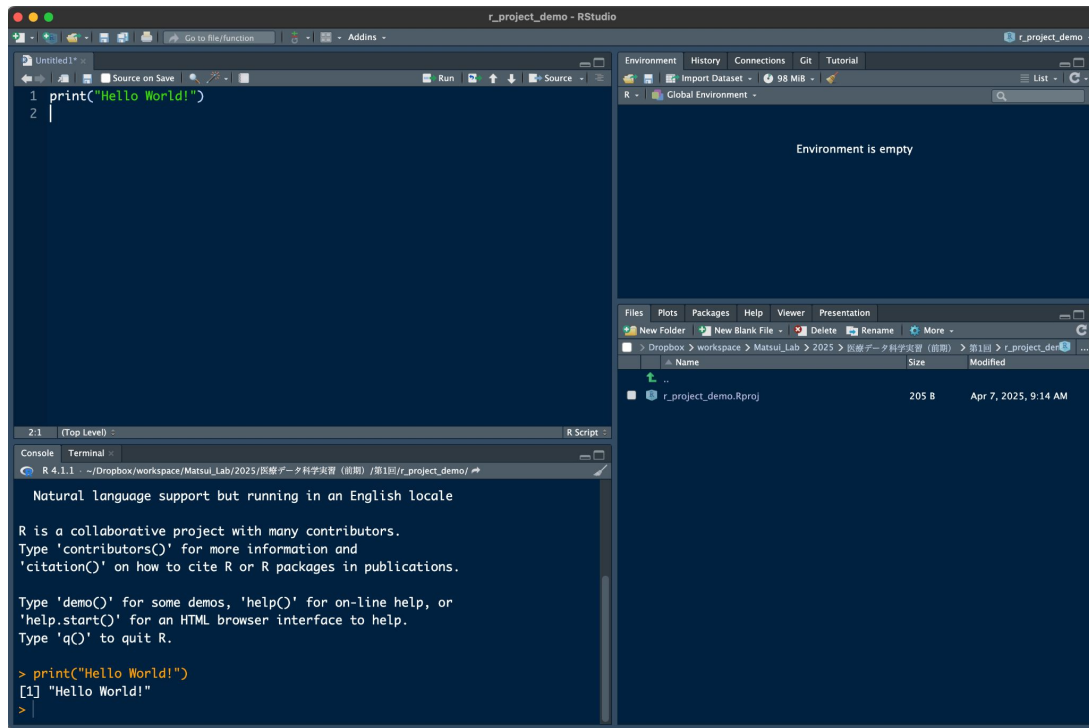
新しいRスクリプトの作り方:

File -> New File

-> R Script

新しいプロジェクトの初期画面

RStudioを使ってみる



演習 : 実行方法1-3全てでHello Worldを表示せよ

Rの実行手順の例:
エディタ or コンソールに
`print("Hello World!")`
と記述

[実行方法 1]

コンソール上でEnter

[実行方法 2]

以下のコマンドで実行

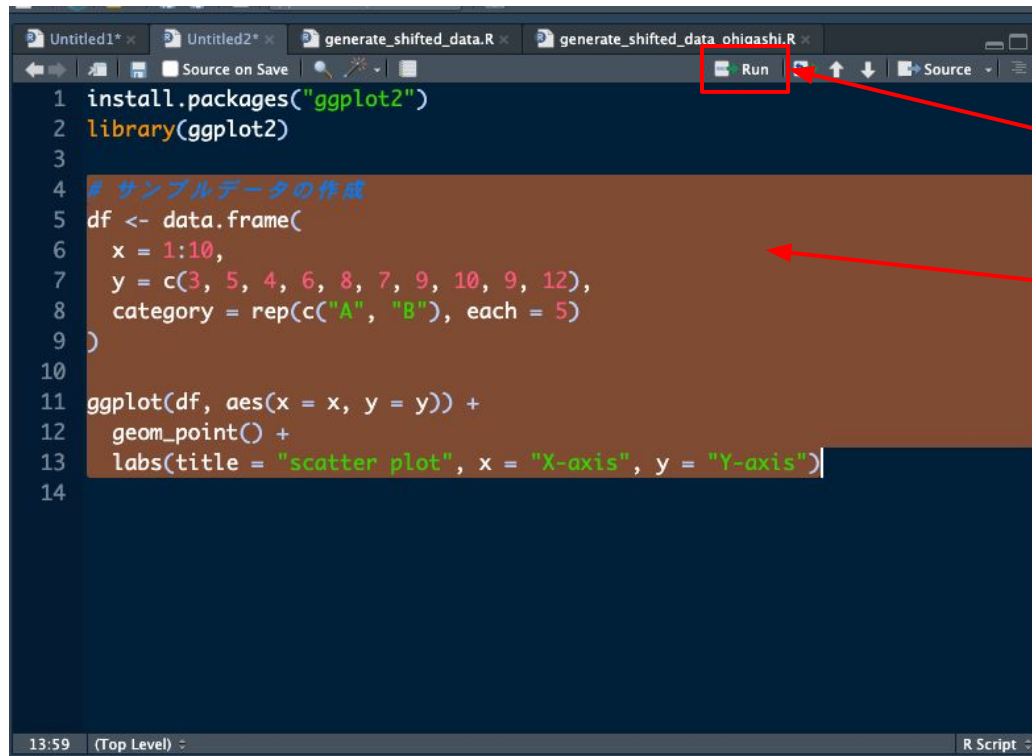
- Ctrl+Enter (Windows)
- command+return (Mac)

[実行方法 3]

File->Save asでセーブした
後にSourceボタン

RStudioを使ってみる

エディタ



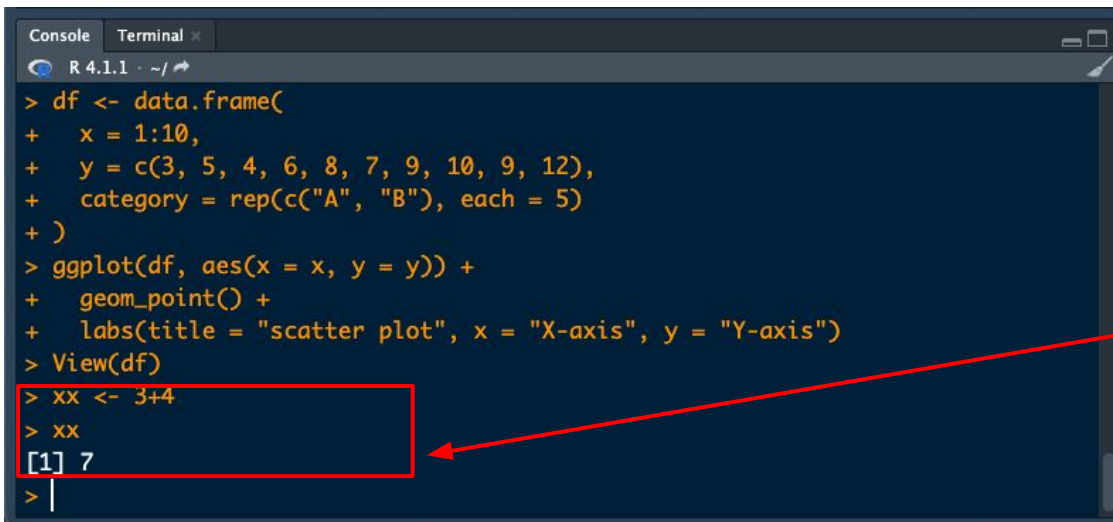
```
1 install.packages("ggplot2")
2 library(ggplot2)
3
4 # サンプルデータの作成
5 df <- data.frame(
6   x = 1:10,
7   y = c(3, 5, 4, 6, 8, 7, 9, 10, 9, 12),
8   category = rep(c("A", "B"), each = 5)
9 )
10
11 ggplot(df, aes(x = x, y = y)) +
12   geom_point() +
13   labs(title = "scatter plot", x = "X-axis", y = "Y-axis")
14
```

- プログラムコードを記述するテキストエディタ部分
- 実行したい行にカーソルを合わせてRunボタンで実行
- 選択行 or ドラッグした領域の実行は
 - Ctrl+Enter (Windows)
 - command+return (Mac)

演習：四則演算をエディタで記述して実行せよ

RStudioを使ってみる

コンソール



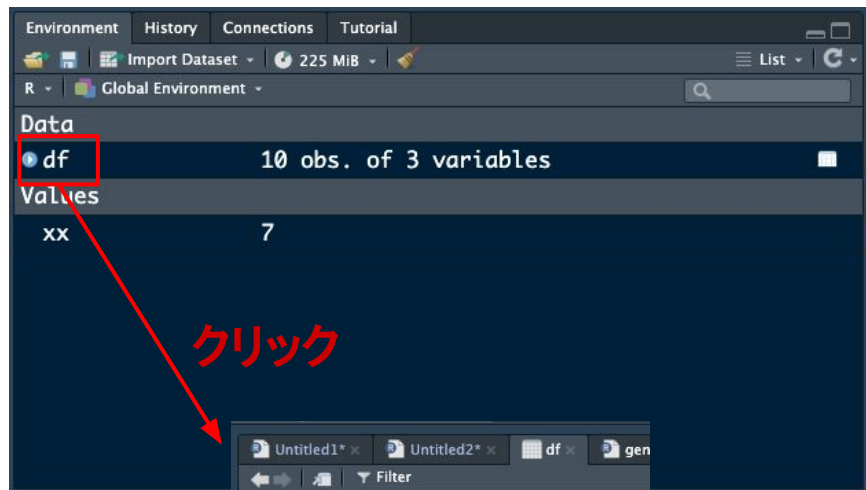
```
Console Terminal x
R 4.1.1 ~ /
> df <- data.frame(
+   x = 1:10,
+   y = c(3, 5, 4, 6, 8, 7, 9, 10, 9, 12),
+   category = rep(c("A", "B"), each = 5)
+ )
> ggplot(df, aes(x = x, y = y)) +
+   geom_point() +
+   labs(title = "scatter plot", x = "X-axis", y = "Y-axis")
> View(df)
> xx <- 3+4
> xx
[1] 7
> |
```

- プログラムコードの実行画面
- エディタ上で実行したプログラムコードと実行結果が表示される
- コンソール上で直接プログラムの記述と実行も可能
(生Rのコンソールと同じ)

演習 : 四則演算をコンソールで記述して実行せよ

RStudioを使ってみる

オブジェクト内容



- プログラムで生成されたオブジェクトの一覧
 - Data: データ構造型オブジェクト
 - データフレーム (data.frame)
 - マトリクス (matrix)
 - リスト (list)
 - 配列 (array)
 - Values: スカラ・ベクトル型オブジェクト
 - 数値 (numeric)
 - 文字列 (character)
 - 論理値 (logical)
 - ベクトル (atomic vector)
- オブジェクト名をクリックするとエディタに中身が表示される

補足: DataとValues

演習: Rの変数とデータ型 

種類	表示カテゴリ	内容例	表示方法
numeric	Values	<code>x <- 10</code>	<code>print(x)</code>
character	Values	<code>name <- "R"</code>	<code>print(name)</code>
vector	Values	<code>v <- c(1,2,3)</code>	<code>print(v)</code>
data.frame	Data	<code>df <- data.frame(...)</code>	<code>View(df)</code>
list	Data	<code>l <- list(a=1, b=2)</code>	<code>str(l)</code>

”

Dataに分類されるオブジェクトの例:

データフレーム

```
df <- data.frame(name = c("A", "B"), score = c(90, 85))
```

Valuesに分類されるオブジェクトの例:

数値やベクトル

```
x <- 5  
y <- c(1, 2, 3)
```

補足: Rは変数の型宣言や配列宣言は不要

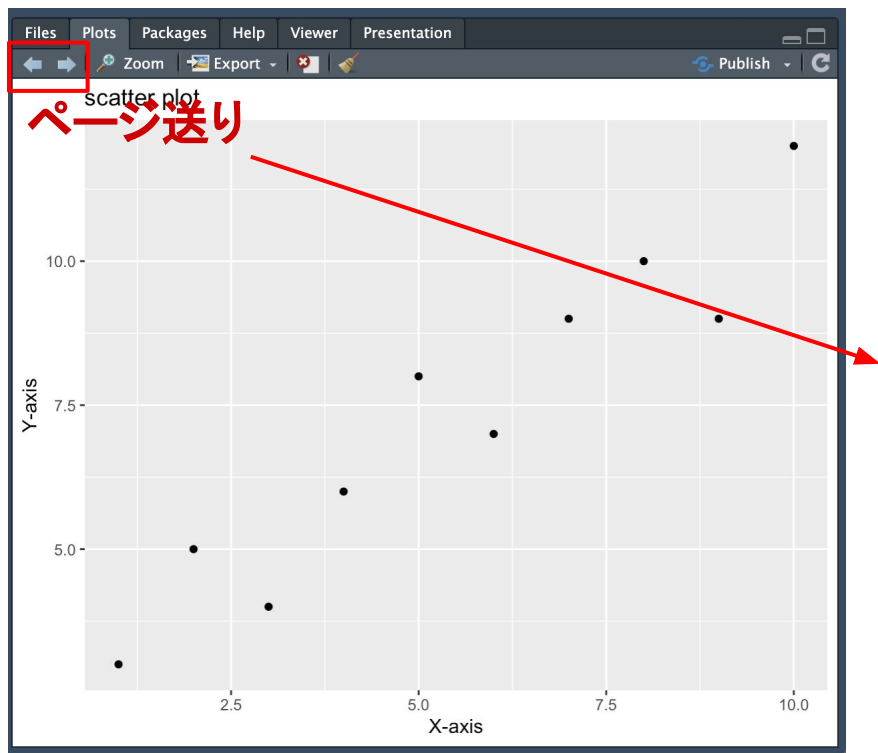
整数 → x <- 5
実数 → y <- 4.3
 → z <- x + y
 z

- このように書くと, 自動的に x, y, z は numeric 型になる (C言語など他のプログラミング言語のように変数の型を明示的に考えなくても良い)
- (厳密には) numeric 型の変数には integer 型 (整数型) と double 型 (倍精度浮動小数点型, 簡単にいえば実数) がある
 - 自分で作成する numeric 型のベクトルはすべて double 型になる
 - 整数型のベクトルを作成したいときは, 数値の後ろに L を付けることでそれを整数として扱うことを宣言する: x <- 5L

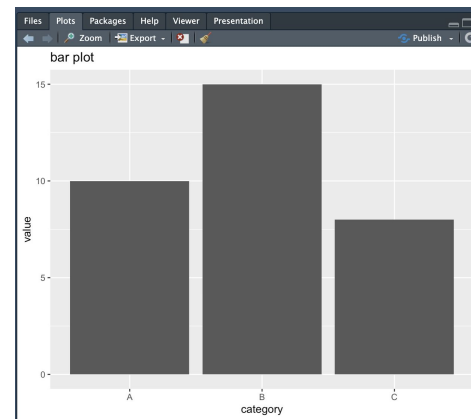
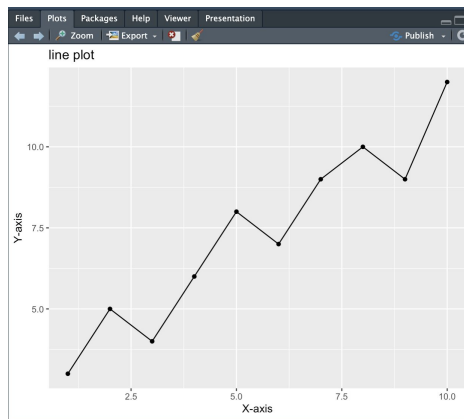
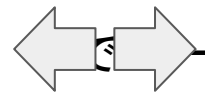
演習: double 型とinteger 型の変数を定義し型を確認せよ (typeof() を使う)

RStudioを使ってみる

プロット表示



- プログラムで生成されたプロットファイルを表示
- これまで作成したプロットはジ送り表示できる



RStudioを使ってみる

実行結果の外部ファイルへの出力と外部ファイルからの読み込み

- Rでは `write()` や `cat()`, `write.csv()` 関数などを使って文字列や変数, データフレームの内容を外部ファイルに出力できる.
- 外部ファイルは `readLines()` や `read.csv()` でR内に読み込むことができる

演習: 次のコードをRStudioで実行し, 結果を確認せよ

1. `message <- "これはRから出力されたメッセージです。" # 文字列をテキストファイルに書き込む`
`write(message, file = "output_message.txt")`
2. `cat("行1\n行2\n行3", file = "output_lines.txt") # 複数行を cat() で出力`
3. `readLines("output_message.txt") # テキストファイルを読み込む`

RStudioを使ってみる

実行結果の外部ファイルへの出力と外部ファイルからの読み込み

- Rでは `write()` や `cat()`, `write.csv()` 関数などを使って文字列や変数, データフレームの内容を外部ファイルに出力できる.
- 外部ファイルは `readLines()` や `read.csv()` でR内に読み込むことができる

演習: 次のコードをRStudioで実行し, 結果を確認せよ

1.

```
df <- data.frame(Name = c("Alice", "Bob"), Score = c(90, 85))  
write.csv(df, file = "scores.csv", row.names = FALSE)
```
2.

```
read.csv("scores.csv") # CSVファイルを読み込む
```

RStudioを使ってみる

実行結果の外部ファイルへの出力と外部ファイルからの読み込み

- Rでは `write()` や `cat()`, `write.csv()` 関数などを使って文字列や変数, データフレームの内容を外部ファイルに出力できる.
- 外部ファイルは `readLines()` や `read.csv()` でR内に読み込むことができる

演習: 次のコードをRStudioで実行し, 結果を確認せよ

1. 任意の文字列を変数に代入し, それを `cat()` を使って `my_message.txt` に保存せよ
2. 好きな名前と点数からなるデータフレームを作成し, それを `my_scores.csv` という名前で保存せよ
3. 作成したCSVファイルを読み込み, 内容を表示せよ

Rによる数値計算とベクトル演算

演習: [Rの変数とデータ型](#) 

- Rでは、**ベクトル演算**を活用することで繰り返し処理を大幅に効率化できる(むしろこれを使わないと非常に処理が遅い)

例: 1千万要素の和の計算

ベクトル演算を使用

```
# ベクトルを準備
x <- runif(1e7)
y <- runif(1e7)

# 2. ベクトル演算による加算
system.time({
  result_vec <- x + y
})
```

計算時間(colab): 0.164 (sec)

forループで計算

```
system.time({
  result_loop <- numeric(1e7) # 初期化
  for (i in 1:n) {
    result_loop[i] <- x[i] + y[i]
  }
})
```

計算時間(colab): 1.492 (sec)