

R のすゝめ

～R によるデータ分析事始め～

Sampo Suzuki, CC 4.0 BY-NC-SA

2022-03-07

Contents

List of Figures	11
List of Tables	13
License	15
I Introduction	17
はじめに	19
想定読者	19
表記ルール	20
なぜ R か?	20
R でなにができるのか?	21
Data Science Workflow	26
Program	27
Import	27
Tidy	27
Transform	27
Visualize	28
Model	28
Communicate	28
Tidyverse Ecosystem	29
II Program	33
1 分析環境	35
1.1 主な分析環境	35
1.1.1 R Commander	35

1.1.2	Google Colab	36
1.1.3	RStudio	37
1.1.4	RStudio Cloud	38
1.1.5	Programing Editor	39
2	R の基本	41
2.1	尺度とデータ型	41
2.1.1	尺度水準	41
	名義尺度を扱う	42
	順序尺度を扱う	43
	間隔尺度を扱う	44
	比例尺度を扱う	44
2.1.2	まとめ	45
2.2	要約統計量	46
2.2.1	平均	46
	算術平均	46
	トリム平均	47
	幾何平均	47
	閑話	49
2.2.2	分散・標準偏差	50
2.2.3	モーメント	52
	歪度	53
	尖度	55
2.2.4	中央値	56
2.2.5	範囲	57
2.2.6	四分位数・四分位範囲	59
2.2.7	最頻値	60
2.2.8	まとめ	62
2.2.9	演習	63
	演習 1	63
	演習 2	63
	演習 3	63
	コラム	64
2.3	推定	65
2.3.1	母集団と標本	65

2.3.2 点推定	65
2.3.3 区間推定	65
2.3.4 まとめ	66
2.3.5 演習	66
演習 4	66
演習 5	66
演習 6	66
2.4 検定	66
2.4.1 F 検定	67
2.4.2 二項検定	67
2.4.3 まとめ	67
2.4.4 演習	67
演習 7	67
演習 8	67
演習 9	67
2.5 分散分析	67
2.5.1 まとめ	67
2.5.2 演習	67
演習 10	67
演習 11	67
演習 12	67
2.6 回帰分析	67
2.6.1 まとめ	67
2.6.2 演習	67
演習 13	67
演習 14	67
演習 15	67
III Wrangle	69
3 Import	71
3.1 readr	71
3.2 readxl	71
3.3 pdftools	71

4 Tidy	73
4.1 Tidy Data	73
4.2 longer	73
4.3 wider	73
5 Transform	75
5.1 filter	75
5.2 rename	75
5.3 select	75
5.3.1 select helpers	75
5.4 mutate	75
5.5 summarize	75
IV Visualize	77
6 Base R	79
6.1 plot	79
6.2 boxplot	79
6.3 hist	79
7 ggplot2	81
V Model/Infer	83
8 Test	85
9 Linear model	87
10 Machine Learning	89
VI Communicate	91
11 R Markdown	93
VII Automate	95
12 shiny	97

VIII APPENDIX	99
Appendix	101
A References	101
B R Basics	103
B.1 基本的な演算	104
B.1.1 算術演算	104
B.1.2 代入	105
B.1.3 代入結果の確認	105
B.1.4 合算	106
B.1.5 平均	107
B.1.6 変数の上書き	107
B.2 変数	108
B.2.1 予約語	108
B.2.2 データ型	108
B.2.3 变数型	109
B.2.3.1 ベクトル型	109
B.2.3.2 因子型	110
B.2.3.3 マトリクス型	111
B.2.3.4 アレイ型	112
B.2.3.5 データフレーム型	112
B.2.3.6 リスト型	114
B.3 定数	115
B.3.1 NA の型	115
B.4 検査・変換	116
B.5 演算子	117
B.5.1 参照演算子	117
B.5.1.1 [演算子	117
B.5.1.2 \$ 演算子	120
B.5.2 単項演算子	122
B.5.3 二項演算子	122
B.5.3.1 算術演算子	123
B.5.3.2 比較演算子	124
B.5.3.3 論理演算子	126

B.5.4 特殊演算子	126
B.5.5 優先順位	126
B.6 制御文	127
B.6.1 条件分岐	127
B.6.1.1 if, else	127
B.6.1.2 switch	128
B.6.1.3 ifelse	130
B.6.2 繰り返し	130
B.6.2.1 for	131
B.6.2.2 while, repeat	131
C 演習解答例	133
演習 1	133
解答例	133
演習 2	134
解答例	134
補足	135
演習 3	136
解答例	136
補足	137
演習 4	138
解答例	138
演習 5	138
解答例	138
演習 6	138
解答例	138
演習 7	138
解答例	138
演習 8	138
解答例	138
演習 9	138
解答例	138
演習 10	138
解答例	138
演習 11	138

解答例	138
演習 12	138
解答例	138
演習 13	138
解答例	138
D Environments 2	139
D.1 Google Colaboratory	139
D.1.1 Login Google	140
D.1.2 Open Google Colab	140
D.1.3 Upload Template	141
D.1.4 Run R code	142
D.1.5 Save File	143
D.2 RStudio Cloud	143
D.2.1 Create Project	144
D.2.2 Install Packages	145
E Install R/RStudio	147
E.1 Install R	148
E.1.1 Windows	148
E.1.1.1 Rtools	148
E.1.2 macOS (OS X)	148
E.1.3 Linux	149
E.2 Install RStudio Desktop	149
E.2.1 動作確認	149
E.3 Install R packages	149
E.3.1 Linux 環境の場合	150
E.4 Install Git	151
E.4.1 Git	151
E.4.2 Git Client	151
F RStudio Server	153
F.1 Setup RStudio Sever with Docker	154
F.1.1 Enable Hyper-V (Windows Only)	154
F.1.2 Download and Install Docker Desktop	154
F.1.3 Download Docker Image	154
F.1.4 Run Container	155

G RStudio IDE	157
G.1 Overview	158
G.2 Keyboard Shortcuts	160
G.3 Writing R code	160
G.4 Global Options	163
G.4.1 General	163
G.4.2 Code	164
G.4.3 Appearance	164
G.4.4 Pane Layout	165
G.4.5 Packages	165
G.4.6 R Markdown	166
G.4.7 Sweave	166
G.4.8 Spelling	166
G.4.9 Git/SVN	167
G.4.10 Publishing	167
G.4.11 Terminal	167
G.5 Project Options	167
H Cloud IDE	169
H.1 RStudio Cloud GA	169
H.2 Exploratory	170
H.3 binder	171
Bibliography	173

List of Figures

1	CC 4.0 BY-NC-SA	15
2	花弁と萼片の分布	22
3	幅と長さの関係	23
4	幅と長さの関係	24
5	基本となる分析プロセス	25
6	分析プロセス	25
7	Data Science Workflow	26
8	Data Science Workflow, CC BY-NC-ND 3.0 US, Hadley Wickham	26
9	Tidyverse Ecosystem	29
1.1	Rcmdr on Ubuntu(Linux)	36
1.2	Google Colab	36
1.3	RStudio Desktop on Windows	37
1.4	RStudio Server, Docker Container	38
1.5	RStudio Cloud	39
2.1	正規分布に近い分布	53
2.2	右に歪んだ分布（左に偏った分布）	54
2.3	左に歪んだ分布（右に偏った分布）	54
2.4	正規分布に近い分布	55
2.5	中心部が高く幅の狭い尖った分布	55
2.6	中心部が低く横に広がった分布	56
D.1	Google Colab, Theme: dark	140
D.2	Upload notebook file	141
D.3	Upload from GitHub	142
D.4	Warning dialog	143
D.5	RStudio Cloud, beta	144
D.6	Initial View	145

D.7	Packages Manager	145
D.8	Install Dialog	146
G.1	RStudio Desktop, Windows	158
G.2	RStudio Pane Layout, Windows	159
G.3	R Notebook file	161
G.4	R Notebook saved file	161
G.5	R Notebook insert chunk	161
G.6	R Notebook first code	162
G.7	R Notebook run code	162
G.8	R Notebook option	163
H.1	RStudio Cloud, beta	169
H.2	Exploratory Public	170

List of Tables

1	表記ルール	20
2	iris dataset	21
1.1	主な想定利用者と分析環境	35
2.1	尺度水準	42
2.2	尺度水準と使うデータ型の例	45
2.3	要約統計量のまとめ	62
2.4	統計量の表記ルール	64

License

本書はクリエイティブ・コモンズ表示 - 非営利 - 繙承 4.0 国際ライセンス¹の下に提供されています。あなたは以下の条件に従う限り自由に共有・翻案することができます。

あなたの従うべき条件は以下の通りです。

- 表示 (BY)
 - あなたは適切なクレジットを表示し、ライセンスへのリンクを提供し、変更があったらその旨を示さなければなりません。これらは合理的であればどのような方法で行っても構いませんが、許諾者があなたやあなたの利用行為を支持していると示唆するような方法は除きます。
- 非営利 (NC)
 - あなたは営利目的でこの資料を利用してはなりません。
- 繙承 (SA)
 - もしあなたがこの資料をリミックスしたり、改変したり、加工した場合には、あなたはあなたの貢献部分を元の作品と同じライセンスの下に頒布しなければなりません。



Figure 1: CC 4.0 BY-NC-SA

ただし、本書にて引用している文書、図、ロゴなどの権利は原著作者が保有しています。

¹<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.ja>

Part I

Introduction

はじめに

最近はデータを使いこなせる者、つまり、数学²を使いこなしたものがより優位に立ち利益を手にする数理資本主義³の時代と言われています。ソフトウェア開発においてもリリース前品質を担保するための〈データに基づく〉統計的な品質管理が有用であると言われていますが、この統計的品質管理を実践するためには統計分析と分析ツール（統計的コンピューティング）に関する知識が必要になります。

『データ指向のソフトウェア品質マネジメント』[野中誠 et al., 2012] は、ソフトウェア開発における統計的品質管理の必要性と品質データの分析に必要となる統計的コンピューティングに関する基本的な知識を解説している数少ない書籍です。この書籍の著者の一人である小池氏が主催し筆者も参加している データ分析勉強会⁴では、統計的コンピューティングに興味をもつ有志が様々な知識や手法を学んでいます。

本書は実務でメトリクス分析を行いたいソフトウェア品質技術者をはじめとした統計的コンピューティングに興味を持っている方々に統計に特化した R 言語（以降、R と記述）⁵の基本的な知識を紹介します。

想定読者

本書はデータ分析勉強会を通じて学んだ知識を実務に適用したい方や統計的コンピューティングに興味があり基本的なコンピュータの知識と基礎的な統計の知識⁶を有している方を想定しています。R を実行するための環境構築に関する詳細な解説は行いませんので、インストール手順などは市販の書籍やインターネットの情報を参考にしてください。なお、環境構築に不安があるけれどもとりあえず R を使ってみたいという方は Google Colaboratory（以降、Google Colab）⁷の利用をおすゝめします。

²純粋数学、応用数学、統計学、確率論、などの広範囲な概念としての数学を意味します

³https://www.meti.go.jp/shingikai/economy/risukei_jinzai/20190326_report.html

⁴<https://sites.google.com/view/kanto-metrics>

⁵<https://www.r-project.org/>

⁶統計検定 4 級程度を想定しています

⁷<https://colab.research.google.com/notebook#create=true&language=r>

表記ルール

本書では以下の表記ルールを用いています。

Table 1: 表記ルール

対象	表記方法	表記例
ハイパーリンク	脚注に URL 表記 ⁸	CRAN ^{No.}
脚注	ハイパーリンクと同じ	キーワード ^{No.}
パス・ファイル名	モノフォント ⁹	sample/sample.Rmd
パッケージ名	太字のモノフォント	tidyverse
変数・オブジェクト名	モノフォント	Sepela.Width
関数名	モノフォントで () 付き ¹⁰	print()
コード	モノフォント（プロンプトなし）	library(tidyverse)
コードの実行結果	モノフォント（プロンプトあり）	## output...
キーボードのキー	モノフォントで [] 付き	[Ctrl]+[S]
数式	LATEX 数式（math mode）	$y = ax^2 + b$
参考文献・資料 ¹¹	[] または () 付き	[Wickham, 2021]

なぜ R か？

統計的コンピューティングには適切なツールが必要不可欠です。R は統計分析に特化しているオープンソースのプログラミング言語です。R がデータ分析に向いている理由を簡潔にまとめているのが『Six Reasons To Learn R For Business』, R Blogger¹²です。

1. R Has The Best *Overall Qualities*
2. R Is Data Science *For Non-Computer Scientists*
3. Learning R Is *Easy With The Tidyverse*
4. R Has *Brains, Muscle, And Heart*
5. R Is Built *For Business*
6. R *Community Support*

R はデータ分析に必要となるデータのハンドリングや可視化、モデリング、そして、レポートといった様々な機能をほとんど無料で利用することができ、CRAN(The Comprehensive R Archive Network)¹³と呼ばれる R のリポジトリには 15,000 を超える

⁸PDF 形式のみで^{No.} の部分は章ごとの通し番号になっています

⁹タイプライタフォントとも呼ばれる等幅フォントです

¹⁰数式内に関数がある場合は数式と同じ表記になります

¹¹カッコ内は参考文献一覧への文書内ハイパーリンクです

¹²<https://www.r-bloggers.com/six-reasons-to-learn-r-for-business/>

¹³<https://cran.r-project.org/>

パッケージ（ライブラリ）が登録されています。それらのパッケージが網羅する分野は CRAN Task Views¹⁴で整理されています。対象分野は古典的な統計や金融統計から最新の機械学習・ベイズ統計など 40 を超えています。その中でも特筆すべき分野は Reproducible Research¹⁵と呼ばれる再現可能性の分野です。

再現可能性とは聞き慣れない言葉だと思いますが、ここでは「ある分析結果を再分析した際に同じ結果が得られること」を意味しています。元々は Research とあるように科学的研究の分野で使われている言葉です。他のプログラミング言語で、Reproducible Research をサポートしているという話は聞いたことがありませんので、このあたりは R の大きな特徴といえます。

また、R は逐次実行するインタプリタ型の言語ですのでソフトウェアメトリクス分析のような探索的分析（Exploratory data analysis）に適しています。加えて非常にフレンドリーかつ活発なコミュニティーが日本でも形成されていますので、悩んだ時などに気軽に質問・相談ができるのも大きな強みです。

ちなみに本書は R の bookdown¹⁶パッケージという文書作成をサポートしてくれるパッケージを用いています。このようなパッケージが用意されているということは R が単なる分析だけでなく研究やビジネスでのレポート作成にも向いていることが分かると思います。

R でなにができるのか？

実際にどのようなことができるかを簡単な分析を通して見てみましょう。分析の対象となるデータはフィッシャーのあやめ（Fisher's or Anderson's iris）¹⁷と呼ばれる下表のようなデータで、R にサンプルデータとして組み込まれています。

Table 2: iris dataset

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
...	NA

¹⁴<https://cran.r-project.org/web/views/>

¹⁵<https://cran.r-project.org/web/views/ReproducibleResearch.html>

¹⁶<https://bookdown.org/>

¹⁷https://en.wikipedia.org/wiki/Iris_flower_data_set

まずは、データがどのような特徴を持っているデータなのかを要約統計量（記述統計量）を求めて確認します。

```
##   Sepal.Length   Sepal.Width    Petal.Length   Petal.Width
##   Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##   1st Qu.:5.100  1st Qu.:2.800  1st Qu.:1.600  1st Qu.:0.300
##   Median  :5.800  Median  :3.000  Median  :4.350  Median  :1.300
##   Mean    :5.843  Mean    :3.057  Mean    :3.758  Mean    :1.199
##   3rd Qu.:6.400  3rd Qu.:3.300  3rd Qu.:5.100  3rd Qu.:1.800
##   Max.    :7.900  Max.    :4.400  Max.    :6.900  Max.    :2.500
##
##          Species
##          setosa   :50
##          versicolor:50
##          virginica:50
##
##
```

萼片 (Sepal) の長さと幅、花弁 (Petal) の長さと幅は、量的変数でその最小値 (Min.)、第1四分位値 (1st Qu.)、中央値 (Median)、平均値 (Mean)、第3四分位値 (3rd Qu.)、最大値 (Max.) を比較すると萼片 (Sepal) より花弁 (Petal) の方が小さい傾向にあると推測できます。品種 (Species) は質的変数 (カテゴリ変数) であり、三品種それぞれが同じ数のデータを持っていることが分かります。

量的変数の傾向の違いを箱ひげ図を用いて可視化してみます。

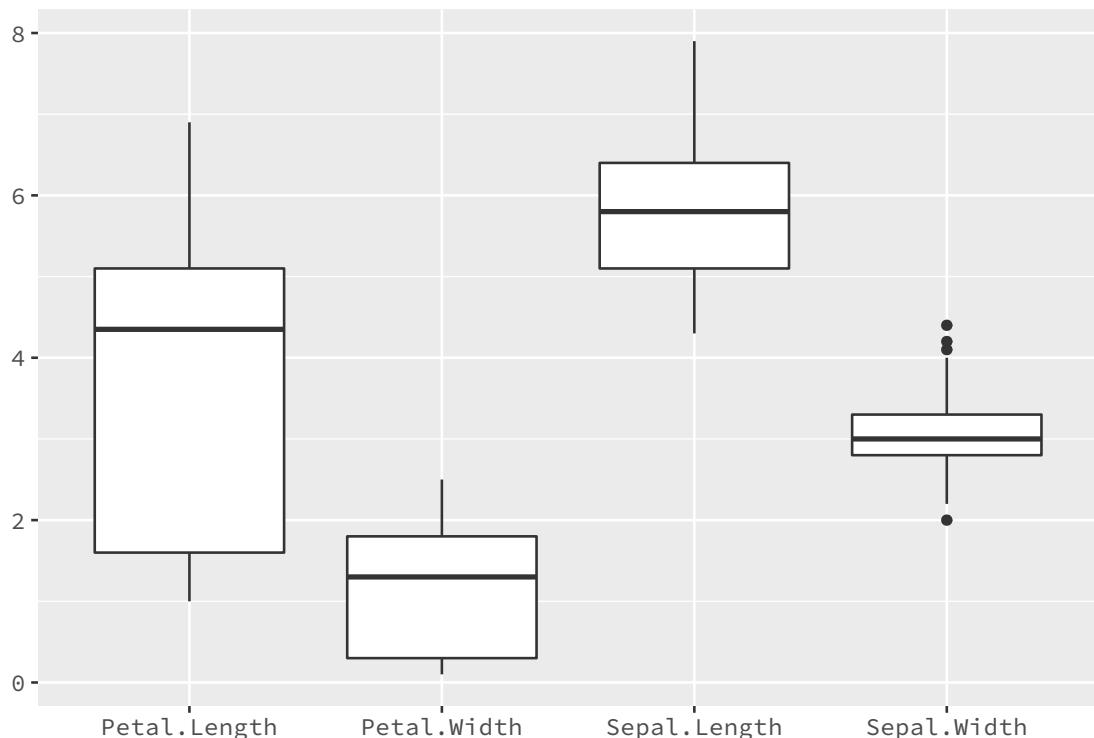


Figure 2: 花弁と萼片の分布

数値で比較したように萼片 (Sepal) より花弁 (Petal) の方が小さいことがひと目で分かります。次に花弁 (Petal) と萼片 (Sepal) の幅と長さの関係を散布図で確認してみましょう。

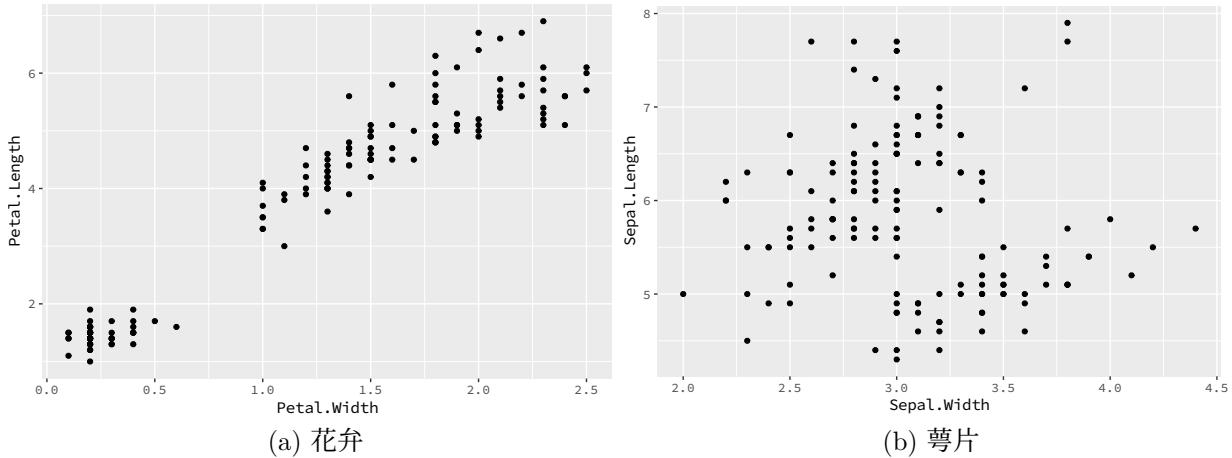


Figure 3: 幅と長さの関係

花弁 (Petal) の幅と長さに関係があり、萼片 (Sepal) の方には関係がないように見えますので、個々の幅と長さの相関を確認します。

```
## 
## Pearson's product-moment correlation
##
## data: Petal.Width and Petal.Length
## t = 43.387, df = 148, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.9490525 0.9729853
## sample estimates:
##          cor
## 0.9628654

##
## Pearson's product-moment correlation
##
## data: Sepal.Width and Sepal.Length
## t = -1.4403, df = 148, p-value = 0.1519
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
## -0.27269325 0.04351158
## sample estimates:
##          cor
## -0.1175698
```

花弁 (Petal) の方には強い相関（相関係数 = 0.9628654）があり、萼片 (Sepal) の方には相関があるとは言えないことが分かります。

花弁 (Petal) の幅と長さの関係を回帰モデル（単回帰式）として求めると下記のようになります。

```
## 
## Call:
## lm(formula = Petal.Length ~ Petal.Width)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -1.33542 -0.30347 -0.02955  0.25776  1.39453 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 1.08356   0.07297  14.85   <2e-16 ***  
## Petal.Width  2.22994   0.05140  43.39   <2e-16 ***  
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 0.4782 on 148 degrees of freedom
## Multiple R-squared:  0.9271, Adjusted R-squared:  0.9266 
## F-statistic: 1882 on 1 and 148 DF,  p-value: < 2.2e-16
```

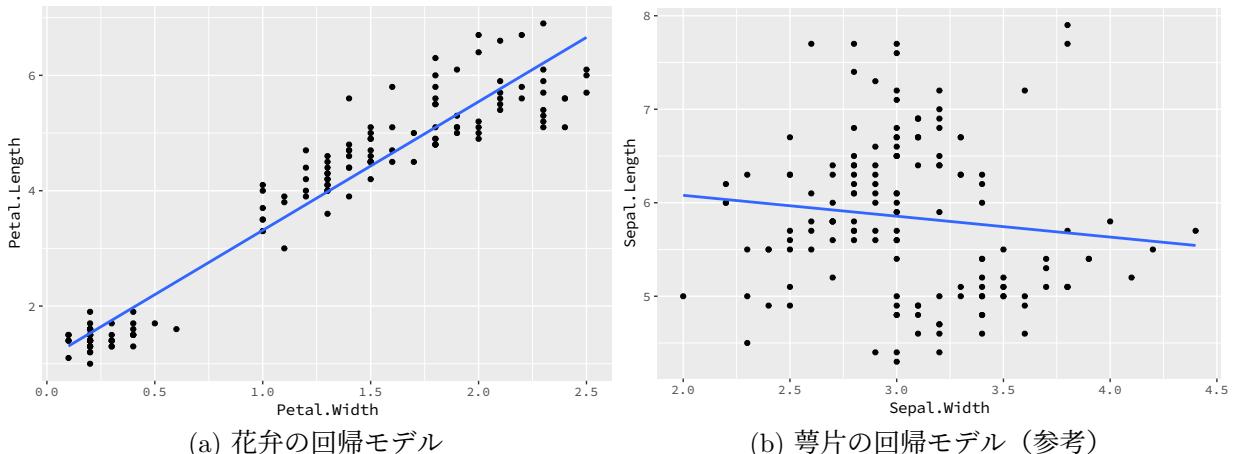


Figure 4: 幅と長さの関係

このように **R** を用いることで比較的簡単にデータを可視化したり統計量を求めることが出来るのが分かったのではないでしょか？なお、ここで利用した出力（計算結果）の読み方は別途解説しますので、こういうことができるという点だけは憶えておいてください。

前ページまでの例のような〈探索的〉データ分析では、対象のデータの確認や変形 (Transform)、可視化 (Visualize)、モデルの作成 (Model or Modeling) を繰り返すことで最適な（予測）モデルを探ります。この流れが分析プロセスの基本で下図のように表すことができます。

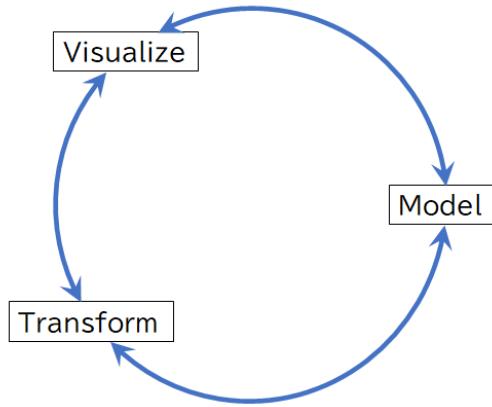


Figure 5: 基本となる分析プロセス

実際には対象となるデータを読み込み (Import)、処理がしやすいように整形 (Tidy) してから可視化などを行います。最後に分析結果を報告 (Communicate) しますので、分析プロセスの全体像下図のようになります。

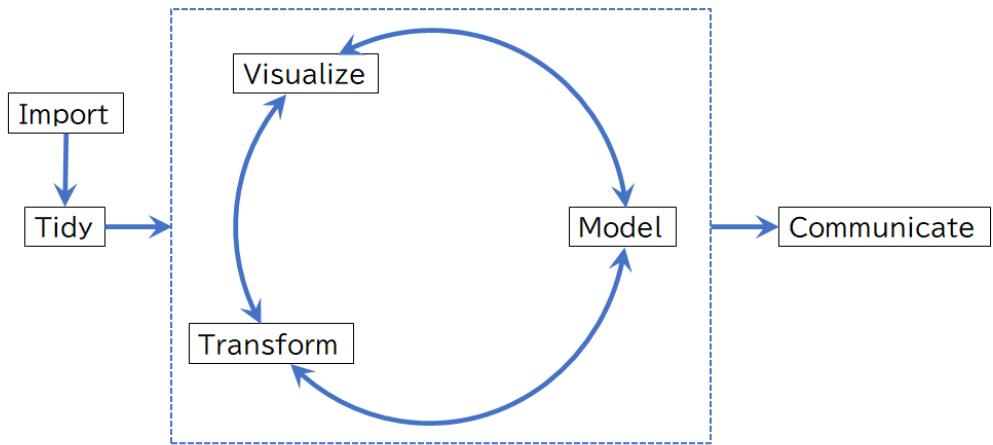


Figure 6: 分析プロセス

Data Science Workflow

分析プロセスを円滑に回すためにはプログラム（統計的プログラミング）などの補助的な仕組みが必要です。全ページの分析プロセスにプログラム（統計的プログラミング）を加えたものが「Data Science Workflow」と呼ばれるフロー図です。

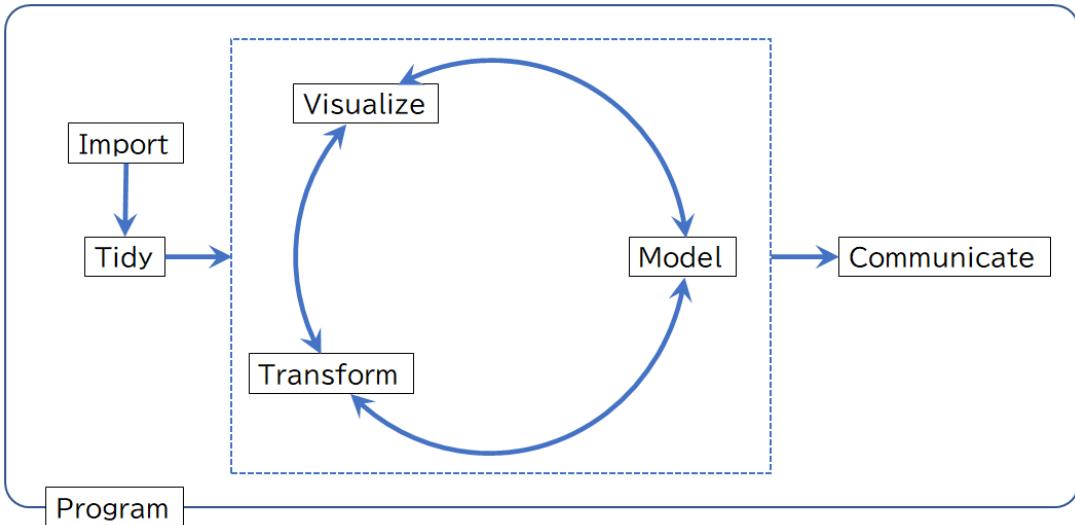


Figure 7: Data Science Workflow

「Data Science Workflow」自体は R コミュニティに多大な貢献をしている Hadley Wickham¹⁸が著書『R for Data Science』¹⁹において提唱している概念です。

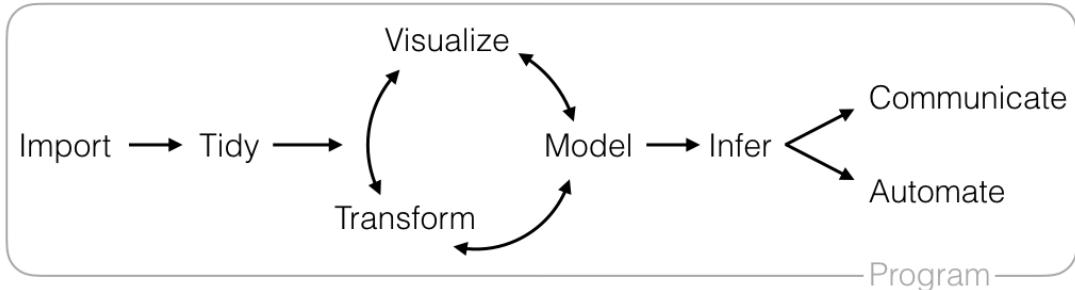


Figure 8: Data Science Workflow, CC BY-NC-ND 3.0 US, Hadley Wickham

Hadley の図には Infer と Automate というプロセスが入っていますが、本書では Infer は Model に、Automate は Communicate に含まれるものとして整理しています。

¹⁸<http://hadley.nz/>

¹⁹<https://r4ds.had.co.nz/>

Program

データ分析のすべてのプロセス（Import～Communicate）で必要となるツール（分析環境）が **Program** です。R では様々なパッケージを使うことで外部アプリケーションとの連動を図り、Rを中心用いることで全てのプロセスを完結させることができます。Ecosystem（後述）と呼べる体系が出来ています。

Import

分析対象となるデータを分析環境に取り込むのが Import プロセスです。データは様々な形式（文字コード、ファイル形式など）で保存されていますので、それらに見合った方法で取り込む必要があります。

Tidy

インポートしたデータは必ずしもデータ分析に適した形式になっているとは限りませんので、扱いやすいような形式（Tidy Data）に整形します。Tidy Data²⁰[Wickham, 2014]はデータ分析において非常に有用で重要な概念で、以下の条件を満たしたデータを意味します。

- 個々の変数が一つの列をなす（Each variable forms a column.）
- 個々の観測が一つの行をなす（Each observation forms a row.）
- 個々の値が一つのセルをなす（Every cell is a single value.）
- 個々の観測の構成単位の累計が一つの表をなす（Each type of observational unit forms a table.）

端的に表現すればデータの「構造と意味が合致する」²¹と言えます。日本語では整然データと呼ばれ、対義語は雑然データ（Messy Data）です。

Transform

整然データ（Tidy Data）に変換したとしても、データをそのまま状態で分析に使えることは稀です。実際のデータにはデータの一部が欠損していたり、分析には必要なデータが含まれていたりしますので、不要なデータを削除したり（クレンジング）、必要なデータだけに絞り込んだり、新しい変数を計算したりする必要があります。これらの変換を事なうのが Transform プロセスです。

Tidy プロセスと合わせて Wrangle や Data Wrangling、前処理などと呼ばれることもあります。本書では Import、Tidy、Transform をあわせて Wrangle と称しています。

²⁰<https://www.jstatsoft.org/article/view/v059i10>

²¹https://ja.wikipedia.org/wiki/Tidy_data

Visualize

文字通りデータの可視化を行うのが Visualize プロセスです。R には伝統的な `plot()` 関数系を用いた可視化に加えて、`ggplot2` パッケージを用いた統一された文法による可視化があります。

Model

データを数式を用いてモデルにするのが Model プロセスです。本書では Model プロセスと Infer プロセスを合わせて Model プロセスと称しています。

Communicate

モデルが作成したら最後は分析結果を伝え（報告し）なければなりません。この報告のプロセスが Communicate で、プロセスにおいて重要な点が再現可能性（Reproducible research）です。再現可能性の重要性については「統計解析の再現可能性を高めるために」²²をお読みください。

²²http://www.igaku-shoin.co.jp/paperDetail.do?id=PA03357_03

Tidyverse Ecosystem

Data Science Workflow を R で実現するための手段が Hadley Wickham が中心となって開発している **tidyverse** パッケージ群による「Tidyverse Ecosystem」です。

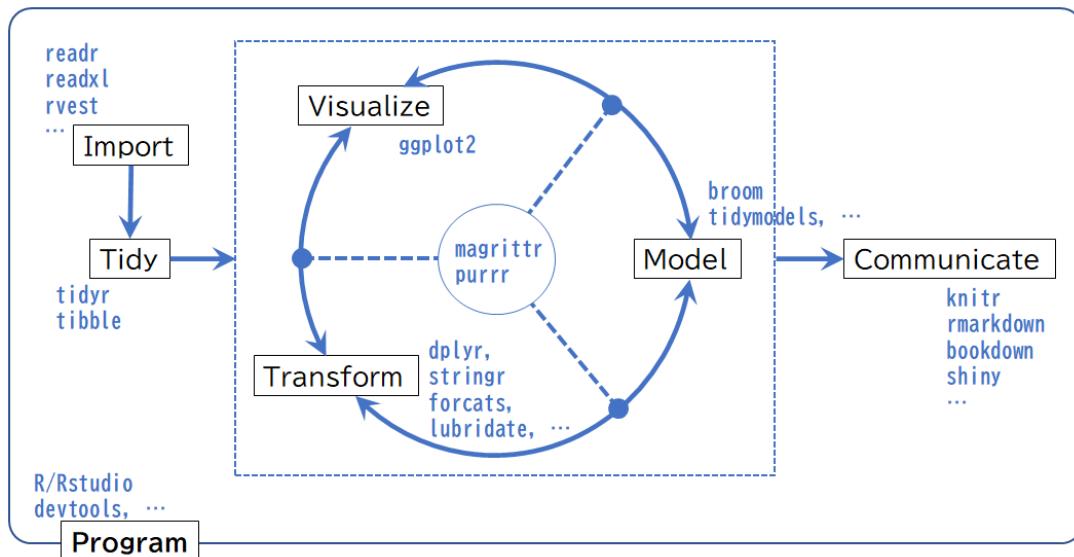


Figure 9: Tidyverse Ecosystem



Ecosystem とは、本来、生態系を意味しますが、ITにおいてはお互いに連携することで利益や効果などを得られるシステムやプロセスなどを生態系になぞらえて Ecosystem と呼ぶことがあります。

ここでは、主要なパッケージをいくつか紹介するに留めておきます。詳細は各分析プロセスの章、または、『RStartHere』 [RStudio, PBC, 2021] をご覧ください。

- Import

- **readr** [Wickham et al., 2022b]
 - * 様々なテキスト形式テーブルを読み込むためのパッケージ
- **readxl** [Wickham and Bryan, 2019]
 - * Microsoft Excel のファイルを読み込むためのパッケージ
- **rvest** [Wickham, 2021b]
 - * Web スクレイピングのためのパッケージ
- **googlesheets4** [Bryan, 2021]
 - * Google API v4 経由でスプレッドシートを読み込むためのパッケージ
- **DBI** [R Special Interest Group on Databases (R-SIG-DB) et al., 2021]
 - * SQL 系の各種データベースと接続するためのパッケージ

- Tidy
 - **tidyverse** [Wickham and Girlich, 2022]
 - * Tidy Data の作成を強力にサポートしてくれるパッケージ
 - **tibble** [Müller and Wickham, 2021]
 - * より厳密に Tidy Data を扱うためのパッケージ
 - **zoo** [Zeileis et al., 2021]
 - * 時系列 (TS) データを効率よく扱うためのパッケージ
- Visualize
 - **ggplot2** [Wickham et al., 2021]
 - * 統一された文法で描画が行えるパッケージ
 - **htmlwidgets** [Vaidyanathan et al., 2021]
 - * JavaScript のウィジェットを利用できるパッケージ
 - **patchwork** [Pedersen, 2020]
 - * **ggplot2** オブジェクトをひとつにまとめてレイアウトするためのパッケージ
- Transform
 - **dplyr** [Wickham et al., 2022a]
 - * Tidy Data を様々な方法で操作するためのパッケージ
 - **stringr** [Wickham, 2019]
 - * 文字列処理をするためのパッケージ
 - **forcats** [Wickham, 2021a]
 - * 因子型を操作するためのパッケージ
 - **lubridate** [Spinu et al., 2021]
 - * 日付データを簡単に変換するためのパッケージ
- Model
 - **broom** [Robinson et al., 2022]
 - * 各種モデリング結果を整然データにするためのパッケージ
 - **tidymodels** [Kuhn and Wickham, 2020, 2021]
 - * 機械学習を中心としたモデリングフレームワーク

- Communicate

- **knitr** [Xie, 2014, 2021b]
 - * R のコードと実行結果を PDF や HTML などに埋め込むためのパッケージ
- **rmarkdown** [Allaire et al., 2022]
 - * マークダウン書式を用いてレポートを作成するためのパッケージ
- **bookdown** [Xie, 2016, 2021a]
 - * 書籍や長いドキュメンを作成するためのパッケージ
- **shiny** [Chang et al., 2021]
 - * インタラクティブな Web アプリケーションを作成するためのパッケージ

プロセスのハブ・スポークを担うパッケージも Ecosystem の一部です。

- Othres

- **magrittr** [Bache and Wickham, 2022]
 - * パイプ演算子 (%>%) によるスムースな処理を実現するパッケージ
- **purrr** [Henry and Wickham, 2020]
 - * 関数型プログラミングによる反復処理を提供するパッケージ

次パートでは、R を使うための基礎知識を紹介します。

Part II

Program

Chapter 1

分析環境

R に限らずプログラミング（言語）を学ぶ際にはプログラミングできる環境を用意しておくべきですが、初学者にとって最も厄介なものが環境構築作業でもあります。そこで、初学者の方には環境構築の必要がない後述の Google Colab の利用をおすめします。RStudio の環境構築に関しては書籍や資料が多数ありますので、R に慣れてきたらチャレンジしてみるも良いと思います。

1.1 主な分析環境

R を利用した分析環境には以下のようなものがあります。

Table 1.1: 主な想定利用者と分析環境

想定利用者	環境	コード記述	再現可能性	備考
初学者	R Commander	不要	低	本書ではスコープ外
初学者	Exploratory	不要 ¹	高	同上
初学・中級者	Google Colab	要	高	
中上級者	RStudio Desktop	要	高	
中上級者	RStudio Cloud	要	高	
中上級者	RStudio Server	要	高	Docker での利用が前提
開発者	R + VS Code/Emacs	要	高	本書ではスコープ外

1.1.1 R Commander

R Commander（以降、Rcmdr）²は、本書ではスコープ外ですが、SQIP 研究会のソフトウェアメトリクス演習コース（以降、メトリクス演習コース）³で利用されるツールですので紹介のみしておきます。

¹必要に応じてコードを記述することも可能です

²<https://socialsciences.mcmaster.ca/jfox/Misc/Rcmdr/>

³<https://www.juse.or.jp/sqip/workshop/outline/index.html>

Rcmdr は R のパッケージとして提供されており、起動すると GUI ベースの対話型分析環境になります。分析にあたって R のコードを記述する必要がありませんので、プログラミングの経験のない方でも手軽に利用することができます。ただし、実行できる機能（関数）が限定されている点と分析対象のデータの扱いが特有な点には注意してください。

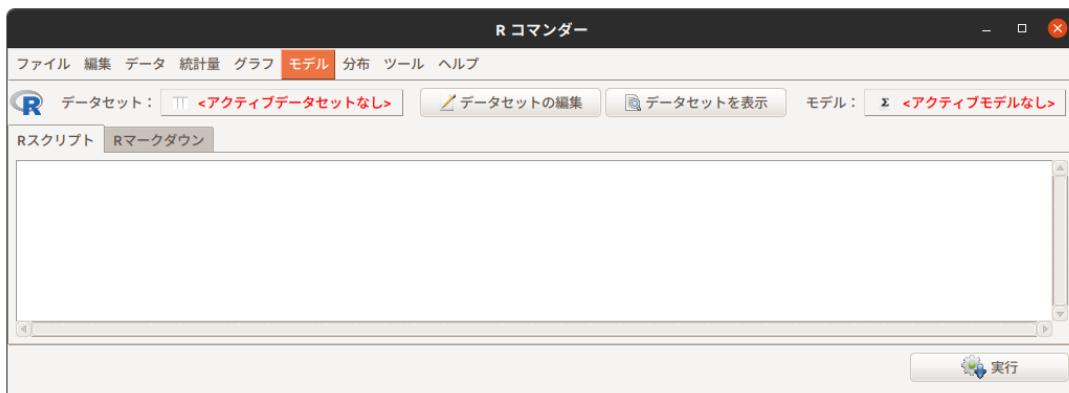


Figure 1.1: Rcmdr on Ubuntu(Linux)

1.1.2 Google Colab

Google Colab は Google アカウントを持っていれば誰でも利用可能な Python 向けの Jupyter Notebook サービスです。Jupyter Notebook は R をエンジンとして利用⁴することができますので、環境を構築することなく使い始めたい方に向いています。プログラミング経験のない初学者の演習環境としては、おすすめの環境です。

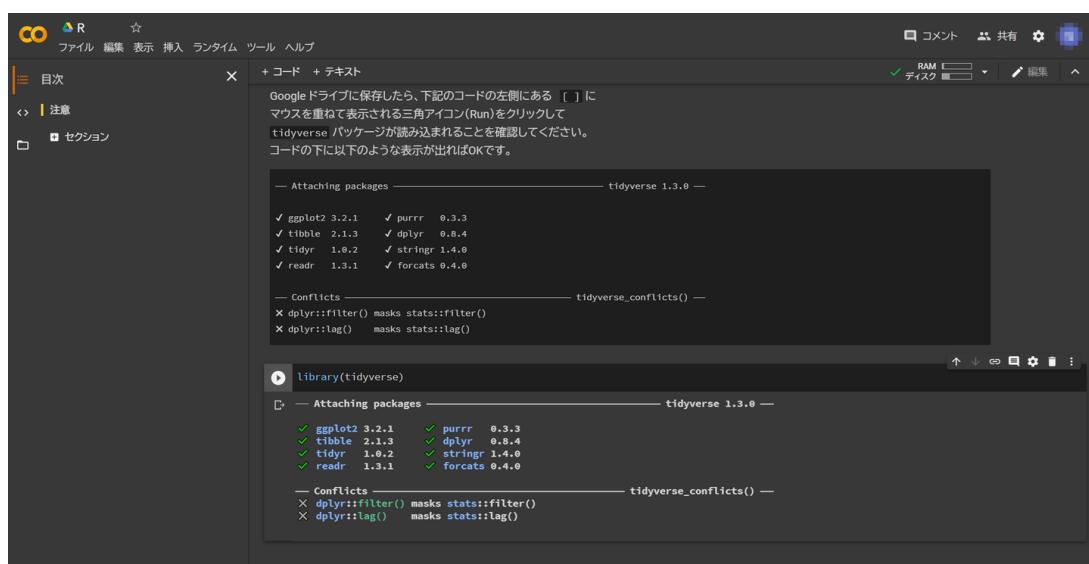


Figure 1.2: Google Colab

⁴<https://colab.research.google.com/notebook#create=true&language=r>

1.1.3 RStudio

再現可能性を確保した探索的データ分析を行うのに最も適しているのが **RStudio**⁵です。無償で使えるオープンソース版には、PC 上のアプリケーションとして動作する **Desktop** と Web サーバとして動作する **Server** の二種類があります。RStudio は R のデファクトスタンダード的な統合開発環境 (IDE) であり、Tidyverse Ecosystem を活かせる分析環境（開発環境）です。その特徴としては、

- R のコードを記述するのに適したエディタを備えている
- R のパッケージをインストール・管理するためのパッケージマネージャを備えている
- R Markdown⁶や Pandoc⁷との連携による再現可能性を確保するための仕組みを備えている
- 外部リソースからのデータを取り込む仕組み（RStudio Connect）を備えている
- 複数の分析をプロジェクト単位で管理する仕組みを備えている
- Git⁸などの外部プログラムと連携したソースの版管理の仕組みを備えている

などが挙げられます。

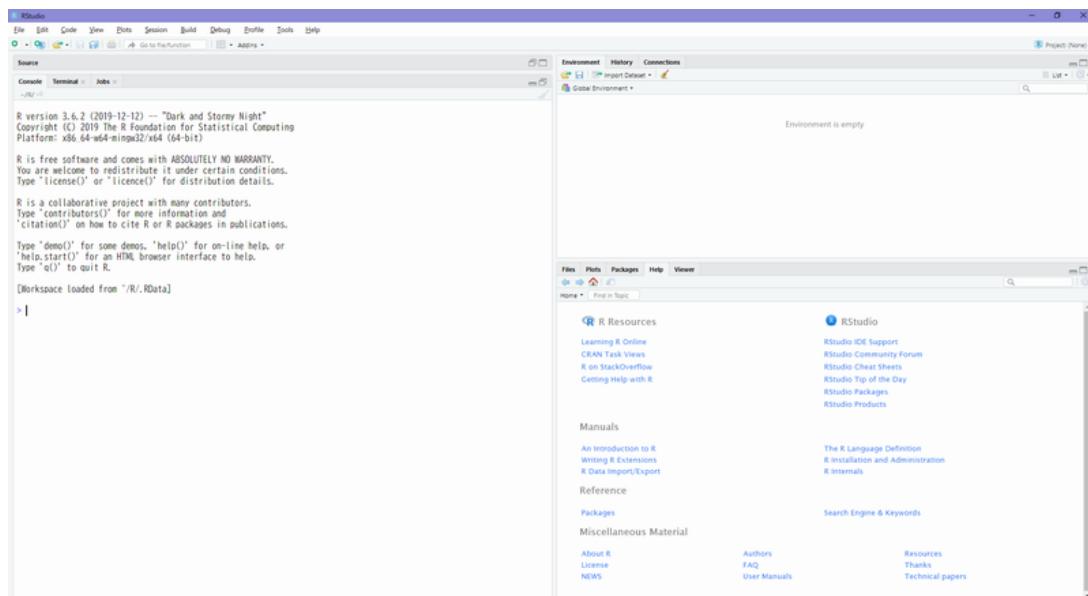


Figure 1.3: RStudio Desktop on Windows

RStudio Server は RStudio をブラウザから使う Linux 上で動作するサーバーアプリケーションです。Docker コンテナとして動作させることが可能ですので、個々の PC での分析環境を固定したい場合などには非常に便利です。

⁵<https://www.rstudio.com/products/rstudio/>

⁶<https://rmarkdown.rstudio.com/>

⁷<https://pandoc.org/>

⁸<https://git-scm.com/>

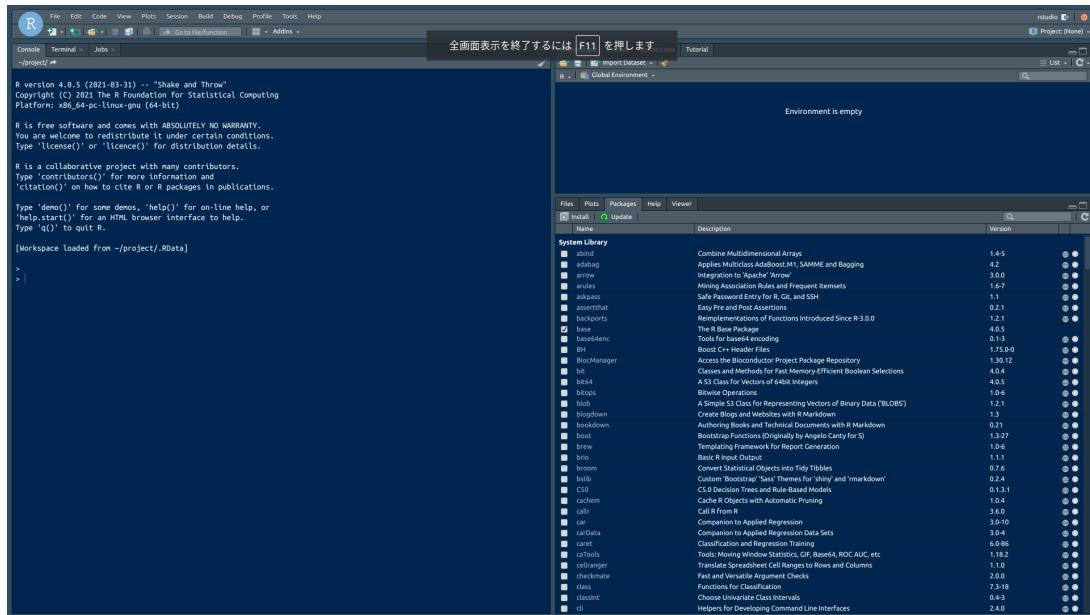


Figure 1.4: RStudio Server, Docker Container



Docker Desktop の商用利用は有料です。営利組織内で利用する場合は利用条件^aを確認の上、利用規約に抵触しなようにご注意ください。

^a<https://www.docker.com/products/docker-desktop>

1.1.4 RStudio Cloud

RStudio Cloud⁹は、その名の通りクラウド版の RStudio です。商用版の RStudio Server Pro をベースしていますので、任意のバージョンの R に切り替えて使ったり RStudio Package Manager とも連携していますのでパッケージ管理が楽になっています。また、英語版しかありませんがチュートリアル機能が充実しているのが特徴です。

無料プランが用意されていますが利用時間が 25 時間/月¹⁰に限定されていますので、繋げっぱなしでの長時間利用や機械学習などには不向きです。

⁹<https://rstudio.cloud/>

¹⁰厳密には「25 project hours」なので、CPU と RAM の割当を増やすと短くなります

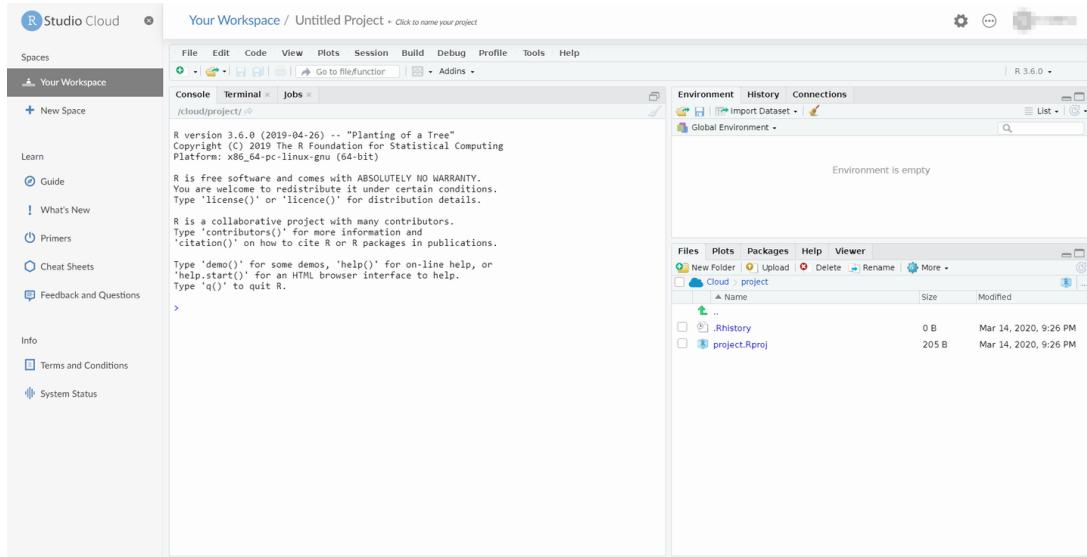


Figure 1.5: RStudio Cloud

1.1.5 Programming Editor

R の本体ははインタプリタ（対話的に逐次実行する処理系）として提供されていますので、R 単体で動作させることができます。区別するために単体の R を R Console と呼ぶことがあります。

一部のプログラミングエディタでは機能拡張などをを利用して直接 R Console と連携して IDE のように R を利用することが可能です。古くは GNU Emacs 用の ESS や最近人気のある Microsoft Visual Studio Code 用の機能拡張を使えば、Visual Studio Code から直接 R を使うことができます。

Chapter 2

R の基本

本書では少し実用的な面から R のプログラミングに必要な知識を解説します。なお、基本文法については Appendix R Basics をご覧ください。

2.1 尺度とデータ型

分析対象となるデータは大きく質的データと量的データに分類することができます。質的データは性別や品名、会社名など主に文字で表すことができる非数値データです。量的データは、温度や身長・体重、売上金額といった数値で表すことができるデータです。

2.1.1 尺度水準

質的データや量的データはデータ自体がもつ意味により下記のような尺度と呼ばれる水準で分類することができます。

- 名義尺度
 - 区別のためのラベルとしての意味を持つ
 - 同値か否かの比較しかできない
- 順序尺度
 - 区別のためのラベルとしての意味に加えて順番や大小関係の意味を持つ
 - 順番や大小関係の間隔には意味がない（加減算はできない）
 - 同値または大小比較の比較演算が可能
- 間隔尺度
 - 大小関係に加えて数値の差に意味をもつ
 - 数値の間隔には意味がありゼロは相対的な意味しかもたない
 - 比較演算に加えて加減算が可能

- 比例尺度

- 大小関係、数値の差に加えて数値の比にも意味をもつ
- 数値の間隔に意味がありゼロは原点としての（絶対的な）意味をもつ
- 比較演算に加えて加減乗除が可能
- 比尺度とも呼ばれる

Table 2.1: 尺度水準

尺度水準	可能な演算	データの例
名義尺度	同値比較	名前、性別、背番号、血液型など
順序尺度	比較演算	着順、ランキング、段階評価など
間隔尺度	比較演算、加減算	温度、時刻、日付など
比例尺度	比較演算、加減乗除算	長さ、重さ、金額など



以降に出てくる灰色に網掛けされた部分は R のコード、コードの下の ‘##’ で始まる部分は実行結果（出力）です。コードによっては実行結果が出力されない場合もあります。

名義尺度を扱う

名義尺度を扱うには文字型 (character)、もしくは、順序なし因子型 (factor) というデータ型を用います。因子型は名義尺度をあつかうのには最も適したデータ型ですが、文字型と異なり、操作が少々厄介なのが難点です。

例えば、血液型を文字型として扱う場合、R では以下のようにしてデータを作成します。

```

1 blood <- c("A", "B", "A", "O", "AB", "A", "A", "O")
2 blood

```

```
## [1] "A"  "B"  "A"  "O"  "AB" "A"  "A"  "O"
```

中身を確認すると文字が並んでいるだけなのがわかります。

```
1 str(blood)
```

```
## chr [1:8] "A" "B" "A" "O" "AB" "A" "A" "O"
```

血液型を因子型として扱う場合は、上記で作成した文字型データを利用して以下のようになります。

```
1 fblood <- as.factor(blood)
2 fblood

## [1] A B A O AB A A O
## Levels: A AB B O
```

データの中身を確認すると文字型とは異なり `Levels` という水準が割当られていることが分かります。

```
1 str(fblood)
```

```
## Factor w/ 4 levels "A","AB","B","O": 1 3 1 4 2 1 1 4
```

因子型は水準がインデックスとして機能しますので、インデックスを利用した演算やグルーピング処理が簡単にできます。

順序尺度を扱う

順序尺度は名義尺度のデータに順番や大小関係という属性を付与したデータですので、順序付き因子型（`ordered`）というデータ型を用いるのが便利です。

例えば、成績のような段階評価は以下のようにしてデータを作成します。

```
1 rank <- c("A", "B", "A", "C", "D", "A", "A", "C")
2 orank <- as.ordered(rank)
3 orank
```

```
## [1] A B A C D A A C
## Levels: A < B < C < D
```

順序なし因子型（`factor`）とは異なり `Levels` に < 記号がついています。この記号は A, B, C, D の順を示しています。データの中身を確認しても順序が設定されていることが分かります。

```
1 str(orank)
```

```
## Ord.factor w/ 4 levels "A"<"B"<"C"<"D": 1 2 1 3 4 1 1 3
```

間隔尺度を扱う

間隔尺度は数値のデータになりますのでデータが取る値に合わせて、整数型 (`integer`)、実数型 (`numeric`)、日付型 (`Date`) のデータ型を用います。

整数型 (`integer`) を明示的に指定する場合は数字に `L` をつけます。

```
1 inum <- c(1L, 2L, 1L, 3L, 1L, 4L, 12L)
2 str(inum)
```

```
## int [1:7] 1 2 1 3 1 4 12
```

実数型 (`numeric`) は単に数字を並べるだけです。

```
1 rnum <- c(1, 2, 1, 3.2, 1, 4, 1.2)
2 str(rnum)
```

```
## num [1:7] 1 2 1 3.2 1 4 1.2
```

比例尺度を扱う

比例尺度は間隔尺度と同様に整数型 (`integer`) や実数型 (`numeric`) のデータ型を用います。データの作成方法は間隔尺度と同じですので省略します。

2.1.2 まとめ

データ型は扱うデータの尺度水準に応じたものを使うようにすると便利です。Rを使いこなしていくうちに自然と理解できると思いますが、因子型変数は文字型変数にはない利点があります。

Table 2.2: 尺度水準と使うデータ型の例

尺度水準	データの例	データ型の例
名義尺度	名前、性別、背番号、血液型など	character or factor
順序尺度	着順、ランキング、段階評価など	orderd
間隔尺度	温度、時刻、日付など	integer, numeric or Date
比例尺度	長さ、重さ、金額など	integer or numeric

2.2 要約統計量

データを分析する前には分析対象となるデータがどのような特徴を持っているか確認しておくことが重要です。この特徴を見るために要約統計量を使います。要約統計量はデータの分布の特徴を表すもので、記述統計量や基本統計量と呼ばれることもあります。

2.2.1 平均

算術平均（相加平均）は要約統計量の中でよく使われ、下式で求められます。

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{1}{n} \sum_{i=1}^n x_i \quad (2.1)$$

算術平均以外にも幾何平均（相乗平均）や

$$\sqrt[n]{x_1 \times x_2 \times \cdots \times x_n} = \sqrt[n]{\prod_{i=1}^n x_i} \quad (2.2)$$

値の下位・上位を任意の割合で除いてから平均（算術平均）を求めるトリム平均（刈込み平均・調整平均）という平均もあります。トリム平均は異常値や外れ値の影響を排除して平均を求めたい場合に利用されます。例えば体操競技では極端な点数を出す審査員の影響を排除するために使われています。

平均といつてもこのように様々な求め方がありますので、実際に R を使ってこれらの平均を求めてみます。

算術平均

式(2.1)の算術平均は `mean()` 関数で求めることができます。最初に平均の計算対象となるデータ（変数）を `c()` 関数で作成します。`c()` 関数はベクトル型の変数を作成する関数です。

```
1 x <- c(1, 3, 3, 5, 7)
```

`x` は値を代入（格納）する変数、`<-` は代入演算子¹です。代入結果を確認するには代入した変数名（`x`）を実行します。

¹R では代入に `=` を使わないようにしてください

```
1 x
```

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

算出平均は `mean()` 関数に計算対象となる `x` を指定して実行します。

```
1 mean(x)
```

```
## [1] 3.8
```

トリム平均

トリム平均は算術平均のコードに引数 `trim` を指定するだけです。引数 `trim` は計算対象外にする割合を指定するオプションです。今回は五つのデータから最小値・最大値の二つを除いてトリム平均を求めますので $\frac{2}{5} = 0.2$ を指定します。

```
1 mean(x, trim = 0.2)
```

```
## [1] 3.666667
```

関数に指定できるオプションの引数を確認した場合 `?mean` に続けて関数名を () 付きで打ち込んで実行してください。ヘルプが表示されます。

```
1 ?mean()
```

幾何平均

式(2.2)の幾何平均（相乗平均）を求めるにはパッケージを追加インストールする必要があります。幾何平均を求める関数を定義しているパッケージは複数ありますが、ここでは `psych` パッケージをインストールして使います。なお、インストールに際してはインターネット接続が必要です。パッケージを使えるようにするための手順は下記の通りです。

1. パッケージをインストールする
2. パッケージを読み込む

パッケージのインストールは `install.packages()` 関数を用います。`install.packages()` 関数を実行すると環境によっては CRAN ミラーサイトの選択を促される場合があります。その場合は、最も近い地域のミラーサイトを選択してください。なお、パッケージのインストールは一度行えば次に使う場合のインストールは不要です。

```
1 install.packages("psych")
```

インストールが完了しましたら `library()` 関数を使ってパッケージを読み込みます。

```
1 library(psych)
```

使用している環境によっては他のパッケージの関数をマスクしているなどのメッセージが出力されますが、今は気にする必要ありません。

これで `psych` パッケージを使う準備が整いましたので、`geometric.mean()` 関数で幾何平均を求めます。

```
1 geometric.mean(x)
```

```
## [1] 3.159818
```

インストールしたパッケージの関数を使う場合、どのパッケージの関数を使っているかを明示的に示すために下記のように`::` 演算子（名前空間へのアクセス演算子）でパッケージ名と関数名をつなげて記述することもできます。

```
1 psych::geometric.mean(x)
```

```
## [1] 3.159818
```

この表記方法については賛否ありますが、どのパッケージの関数を使っているかが分かりやすいので本書ではこの記述方法を用います。

閑話

なぜパッケージを使うのか？

幾何平均を求めるために psych パッケージを利用しましたが、R には総積 ($x_1 \times x_2 \times \dots \times x_n$) を求める prod() 関数と累乗根 ($x^{\frac{1}{n}}$) を求めるべき乗演算子 (^) がありますので、これを組み合わせれば幾何平均を求めることができます。

```
1 prod(x) ^ (1 / length(x))
```

```
## [1] 3.159818
```

length() 関数は変数 x の長さ（変数 x の中にある値の個数）を求める関数です。

幾何平均は全ての値を prod() 関数で乗算しますが、データの数が多い場合やデータの値が大きい（または、小さい）場合には prod() 関数がオーバーフロー（または、アンダーフロー）を起こしてしまうことがあります。例えば、1 から 200 までを乗算するとオーバーフローを起こします。

```
1 prod(c(1:200))
```

```
## [1] Inf
```

一方、geometric.mean() 関数はフロー対策が施されていますのでオーバーフローを起こすことなく幾何平均を求めることができます。

```
1 psych::geometric.mean(c(1:200))
```

```
## [1] 74.90045
```

このように既にパッケージで提供されている関数があれば、自分でコードを組むよりはパッケージを導入した方が早くて確実です（ここでの本来の目的は幾何平均を求ることでなくデータの特徴を掴むことです）。特に CRAN に登録されているパッケージは審査が行われていますので、使わないよりは使った方が確実に目的を達成しやすくなります。R には先人の知恵が結集していますので、車輪の再発明に挑戦するより先人の知恵を活用する方をおすゝめします。

2.2.2 分散・標準偏差

データの散らばり具合を見るには分散・標準偏差がよく使われます。例えば、以下の x と y が取る値の範囲は同一ですが、同じ散らばり具合と言えるでしょうか？

```
1 x <- c(1, 5, 5, 5, 9)
2 range(x)
```

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

```
1 y <- c(1, 1, 5, 9, 9)
2 range(y)
```

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

散らばり具合を見るためにデータの個々の値と平均値との差の二乗の算術平均を分散（標本分散 s^2 ）、

$$s^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \quad (2.3)$$

単位を戻すために分散の平方根をとったものを標準偏差（標本標準偏差 s ）と呼びます。

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2} \quad (2.4)$$

しかし、R にはこの標本分散 (s^2) を求める関数が用意されていません。代わりに不偏分散 ($\hat{\sigma}^2$) を求める `var()` 関数が用意されています。

$$\begin{aligned} \hat{\sigma}^2 &= \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \\ &= var(x) \end{aligned} \quad (2.5)$$

式(2.3)(2.5)から標本分散 (s^2) は `var()` 関数を用いると

$$\begin{aligned}
 s^2 &= \frac{n-1}{n} \hat{\sigma}^2 \\
 &= \frac{\text{length}(x) - 1}{\text{length}(x)} \times \text{var}(x)
 \end{aligned} \tag{2.6}$$

のように求めることができます。標本標準偏差 (s) は式(2.6)の平方根を取ればいいこともわかります。

同じ値の範囲を撮っている x と y の標準偏差を求めると x の方が散らばり具合が狭いことが分かります。

```
1 sqrt(((length(x) - 1) / length(x)) * var(x))
```

```
## [1] 2.529822
```

```
1 sqrt(((length(y) - 1) / length(y)) * var(y))
```

```
## [1] 3.577709
```

!

不偏分散は母分散 (σ^2) の不偏推定量 ($\hat{\sigma}^2$) で、不偏標本分散と呼ばれることもあります。母分散は未知の値ですので、標本からの不偏推定量で代用しています。

2.2.3 モーメント

本節の内容はデータ分析ではありませんので読み飛ばして頂いて構いません。

平均（分布の重心）と分散（分布の広がり）は、歪度（分布の左右非対称度合い）、尖度（分布の峰の尖り度合い）を含めて「モーメントから求められる要約統計量」と呼ぶことがあります。

n 個のデータの平均値を

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (2.7)$$

とした場合、平均値まわりの m 次の中央モーメント \bar{x}_m を

$$\bar{x}_m = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^m \quad (m = 2, 3, 4) \quad (2.8)$$

と定義します。 $m = 2$ の場合、

$$\bar{x}_2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 = s^2 \quad (2.9)$$

となりますので、 \bar{x}_2 は分散 s^2 であることがわかります。

$m = 3$ の場合、

$$\bar{x}_3 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^3 \quad (2.10)$$

となり、これを標準偏差の三乗 (s^3) で割ったもの

$$\gamma_1 = \frac{\bar{x}_3}{s^3} \quad (2.11)$$

を歪度 (γ_1) と呼びます。

$m = 4$ の場合、

$$\bar{x}_4 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^4 \quad (2.12)$$

となり、これを標準偏差の四乗、または、分散の二乗 (s^4) で割ったもの

$$\gamma_2 = \frac{\bar{x}_4}{s^4} \quad (2.13)$$

を尖度 (γ_2) と呼びます。

歪度や尖度を求める関数は標準では用意されていませんので `e1071` パッケージを用います。

```
1 install.packages("e1071")
2 library(e1071)
```

歪度

歪度はデータ分布の左右対象度合いです。左右対象となるデータ分布であれば 0、右に歪んでいれば正の値を、左に歪んでいれば負の値を取ります。例えば下図のような分布では

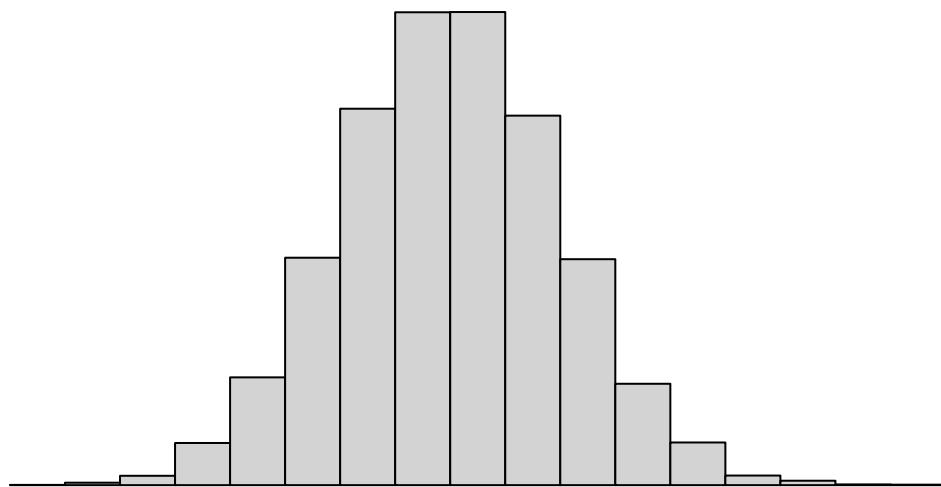


Figure 2.1: 正規分布に近い分布

```
1 e1071::skewness(x)
```

```
## [1] 0.04548819
```

とゼロに近い値となります。次に下図のような分布では

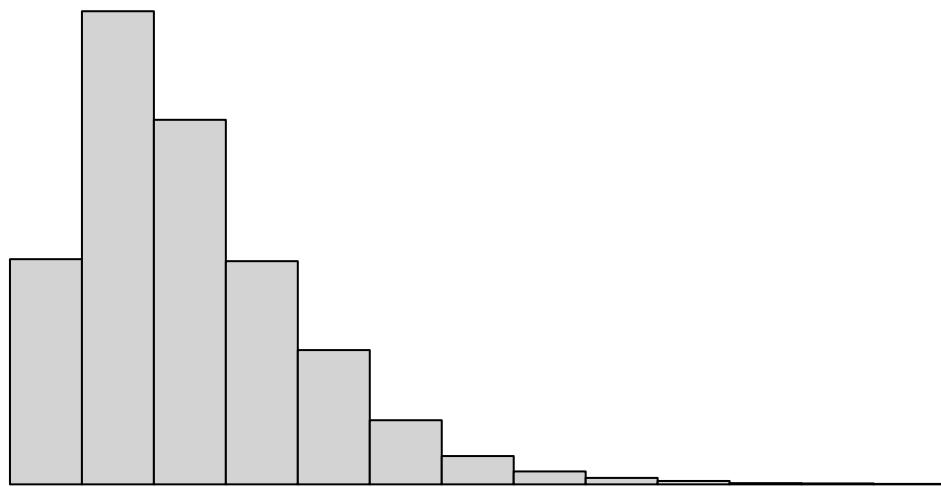


Figure 2.2: 右に歪んだ分布（左に偏った分布）

```
1 e1071::skewness(x)
```

```
## [1] 1.318739
```

正の値をとります。上図とは逆の分布では

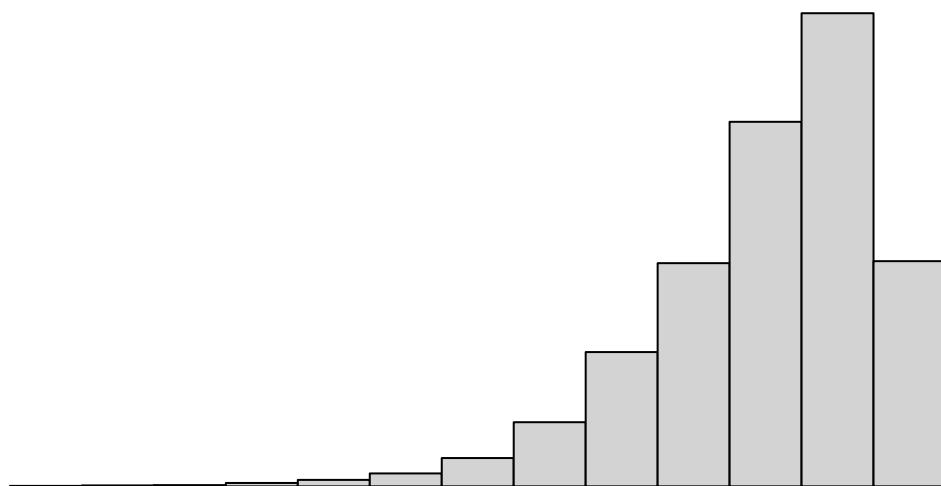


Figure 2.3: 左に歪んだ分布（右に偏った分布）

```
1 e1071::skewness(x)
```

```
## [1] -1.318739
```

負の値をとります。

尖度

尖度はデータ分布の尖り具合です。標準正規分布と等しければ 0、標準正規分布より尖った t 分布のような分布では正の値、広がった分布であれば負の値をとります。

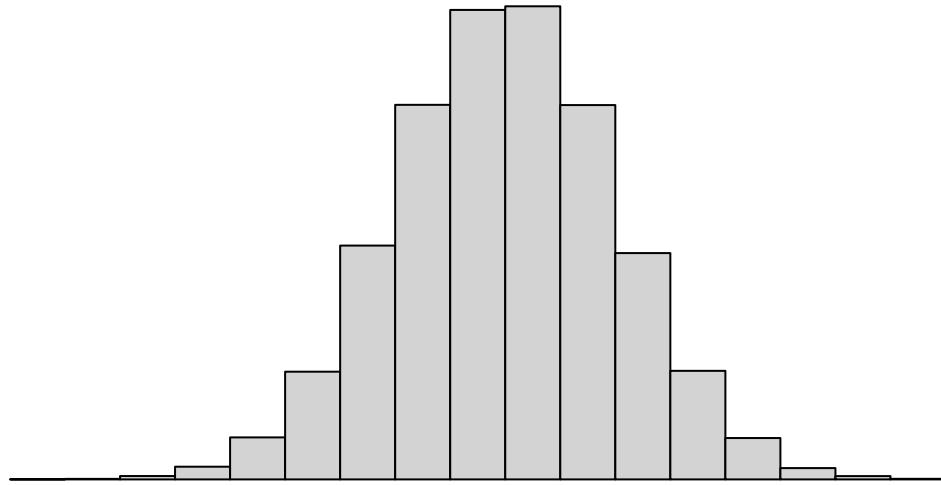


Figure 2.4: 正規分布に近い分布

```
1 e1071::kurtosis(x)
```

```
## [1] 0.01457407
```

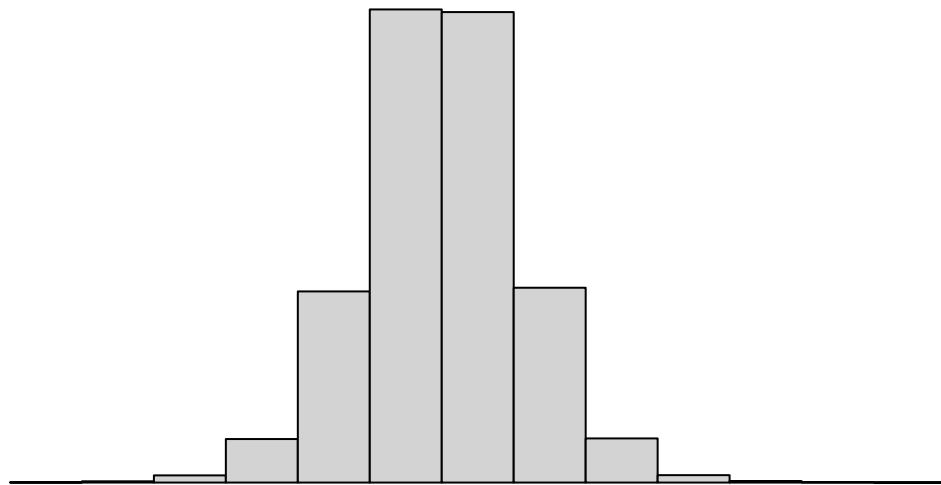


Figure 2.5: 中心部が高く幅の狭い尖った分布

```
1 e1071::kurtosis(x)
```

```
## [1] 0.7919764
```

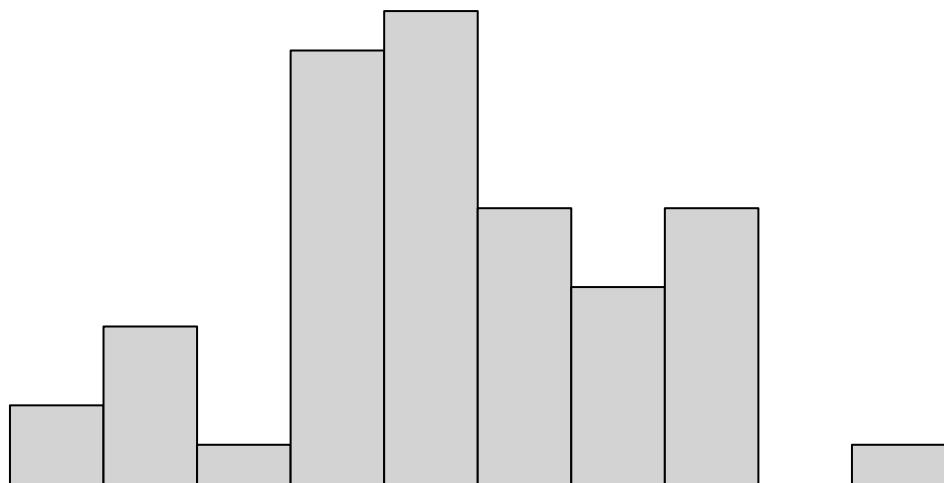


Figure 2.6: 中心部が低く横に広がった分布

```
1 e1071::kurtosis(x)
```

```
## [1] -0.3767132
```

2.2.4 中央値

平均値や分散・標準偏差はデータの分布が平均値を中心に左右対象になっているような場合は有用な統計量ですが、データの分布が大きく歪んでいる対象に対しては適した統計量とは言えません。このような場合、平均値の代わりに中央値（中位数）や最頻値、分散・標準偏差に代わり範囲・分位範囲（分位値）を用います。これらの統計量はデータの中の位置を用いますので、データの分布に影響されることが少ないロバストネス（頑健性の高い）な統計量です。

中央値は文字通り中央（真ん中）に位置する値です。データを小さい順に並べて真ん中に位置する値で、`median()` 関数を使って求めます。データが順番に並んでいなくても `median()` 関数内でソート処理を行います。

```
1 x <- c(1, 3, 5, 9, 7)
2 median(x)
```

```
## [1] 5
```

データの長さ（値の数）が奇数の場合は、単純に中央の値（上記の場合は 1, 3, 5, 7, 9 なので中央に位置する 5）を返しますが、データの長さ偶数の場合は中央に近い両側の値の平均値（相加平均）を返します。

```
1 x <- c(1, 3, 5, 9, 7, 6)
2 median(x)
```

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

例えば、上記のようにデータの長さが偶数（6）で 1, 3, 5, 6, 7, 9 というデータですの
で、中央に最も近い 5, 6 の平均値を中央値としています。

中央値と平均値を比較すると中央値がデータの分布に影響されにくい（ロバストネ
ス）なことが分かります。

```
1 x <- c(1, 3, 5, 9, 7)
2 y <- c(1, 3, 5, 9, 28)
```

```
1 median(x) # 中央値の比較
```

```
## [1] 5
```

```
1 median(y) # 中央値の比較（大きい値には引っ張られない）
```

```
## [1] 5
```

```
1 mean(x) # 平均値の比較
```

```
## [1] 5
```

```
1 mean(y) # 平均値の比較（大きい値に引っ張られている）
```

```
## [1] 9.2
```

2.2.5 範囲

前述の `x` と `y` は、中央値で見る限りと同じデータになってしまいます。この結論はい
ささか不合理ですので、データのばらつきを範囲（レンジ）で見てみます。範囲を求
めるには、その名の通りの `range()` を用います。

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

```
1 range(x)
```

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

```
1 y
```

```
## [1] 1 3 5 9 28
```

```
1 range(y)
```

```
## [1] 1 28
```

範囲でみると y の方が x よりばらついていることが分かります。

range() 関数は最小値と最大値を一度に取得しています。最小値は min() 関数、最大値は max() 関数で求められますので、range() 関数は以下のように処理をしていることが分かります。

```
1 c(min(x), max(x))
```

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

最小値から最大値の幅を求めたい場合には min() 関数と max() 関数を用いるよりは diff() 関数を用いた方が簡単です。

```
1 diff(range(x))
```

```
## [1] 8
```

```
1 diff(range(y))
```

```
## [1] 27
```

2.2.6 四分位数・四分位範囲

中央値は中位数と呼ばれるようにデータを二分の一にした場所にあるデータです。さらに細かくして全体を四分の一にした場所にあるデータを四分位数と呼びます。小さい方から見て最初の四分の一にある値を第一四分位数 (25% 点、 Q_1)、次の四分の一にある値を第二四分位数 (50% 点、 Q_2)、次の四分の一にある値を第三四分位数 (75% 点、 Q_3) といいます。第二四分位数 (50% 点、 Q_2) は中央値（中位数）といえます。四分位数にも範囲があり、四分位範囲 (IQR , interquartile range) と呼ばれ以下のように定義されます、

$$IQR = Q_3 - Q_1 \quad (2.14)$$

四分位数は `quantile()` 関数で求めることができます。オプションの引数を指定しなければ第一～第三四分位数に加えて最小値 (0%)・最大値 (100%) を加えた五つの値が返ってきます。

```
1 quantile(x)
```

```
##    0%   25%   50%   75% 100%
##     1     3     5     7     9
```

任意の四分位数を求めたい場合は引数 `prob` を指定します。`prob` は 0~1 の間で指定する点に中位してください。

```
1 quantile(x, prob = 0.25)
```

```
## 25%
## 3
```

複数の四分位数を同時に求めることも可能です。

```
1 quantile(x, prob = c(0.25, 0.75))
```

```
## 25% 75%
##   3    7
```

四分位範囲 (*IQR*) を求める場合は `IQR()` 関数が便利です。

```
1 IQR(x)
```

```
## [1] 4
```

2.2.7 最頻値

最頻値（モード）とは文字通り、最も頻繁に出てくる値のことです。例えば、以下の `x` と `y` は、平均値・中央値・範囲は全て同じ値になるデータです。

```
1 x <- c(1, 1, 3, 5, 9)
2 y <- c(1, 3, 3, 3, 9)
3 mean(x)
```

```
## [1] 3.8
```

```
1 mean(y)
```

```
## [1] 3.8
```

```
1 median(x)
```

```
## [1] 3
```

```
1 median(y)
```

```
## [1] 3
```

```
1 diff(range(x))
```

```
## [1] 8
```

```
1 diff(range(y))
```

```
## [1] 8
```

このようなデータに対しては最頻値を用いると平均値・中央値・範囲では見えなかった差異が見える場合があります。最頻値を求める関数は標準では用意されていませんので modeest パッケージを用います。

```
1 install.packages("modeest")
2 library(modeest)
```

```
1 modeest::mfv(x)
```

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

```
1 modeest::mfv(y)
```

```
## [1] 3
```

2.2.8 まとめ

本節に出てきた要約統計量と関数は下表の通りです。

Table 2.3: 要約統計量のまとめ

要約統計量	表記	R での求め方 ²
標本平均	\bar{x}	<code>mean(x)</code>
トリム平均		<code>mean(x, trim = t), t = 0 to 0.5</code>
幾何平均		<code>psych::geometric.mean(x)</code>
標本分散	s^2	<code>(length(x) - 1) / length(x) * var(x)</code>
不偏分散	$\hat{\sigma}^2$	<code>var(x)</code>
標本標準偏差	s	<code>sqrt((length(x) - 1) / length(x) * var(x))</code>
不偏標本偏差	$\hat{\sigma}$	<code>sd(x)</code>
歪度	γ_1	<code>e1071::skewness(x)</code>
尖度	γ_2	<code>e1071::kurtosis(x)</code>
中央値		<code>median(x)</code>
範囲	R^3	<code>range(x) or diff(range(x))</code>
四分位数	Q_n	<code>quantile(x)</code>
第 n 四分位数	Q_n	<code>quantile(x, probs = u), u = 0.25*n, n = 1, 2, 3^4</code>
四分位範囲	IQR	<code>IQR(x)</code>
最頻値		<code>modeest::mfv(x)</code>

²引数 x は計算対象となる数値データが格納されたベクトル型の変数

³重相関計数を意味する場合もあり

⁴引数 $probs$ はベクトル指定も可能 (例: $probs = c(0.25, 0.75)$)

2.2.9 演習

演習 1

標本分散 (s^2) と不偏分散 ($\hat{\sigma}^2$) の関係式(2.6)を用いて以下の二つのデータの標本分散 (s^2) を求めなさい。

```
1 x <- c(1, 5, 5, 5, 9)
2 y <- c(1, 1, 5, 9, 9)
```

演習 2

欠損値を含む以下のデータの平均値 (\bar{x})、不偏分散 ($\hat{\sigma}^2$) ならびに標本分散 (s^2) を求めなさい。

```
1 z <- c(1, 5, NA, 7, 9)
```

引数 `na.rm` を指定します

演習 3

`iris` データセットの各変量に対する四分位数を求めなさい。

コラム

英字？ギリシャ文字？

統計の書籍では様々な統計量を英字で表記する場合とギリシャ文字で表記する場合があり混乱しやすいのですが、英字（アルファベット）で表記されている場合は標本統計量、ギリシャ文字で表記されている場合は母集団の統計量もしくは母集団の推定統計量を意味しています。

推定統計量を表記する場合、推定であることを意味するハット ($\hat{\cdot}$) という記号をつけます。平均と分散・標準偏差を使って整理すると下表のようになります。

Table 2.4: 統計量の表記ルール

統計量	平均	分散	標準偏差	備考
母集団の統計量	μ	σ^2	σ	神のみぞ知る統計量
(不偏) 推定量	$\hat{\mu}$	$\hat{\sigma}^2$	$\hat{\sigma}$	標本から母集団を推定した統計量
標本統計量	\bar{x}	s^2	s	標本から求める統計量

ハット ($\hat{\cdot}$) を省略している書籍や資料も散見されますので混乱しないように注意してください。詳しくは『統計的方法のしくみ』[永田, 1996] の「3. 母数と統計量の区別」や『統計解析のはなし』[大平, 2006] の「2. 素人探偵物語」などを参照してください。

2.3 推定

前節で扱った要約統計量は与えられたデータに関する特徴を求めるもので、一般的には記述統計と呼ばれる手法に分類されます。一部、不偏推定量の記述がありますが、こちらは確率変数と確率分布の確率論をベースとした推測統計と呼ばれる手法に分類されます。推測統計は大きく統計的推定と統計的仮説検定に二分されます。本節では前者の統計的推定を扱います。

2.3.1 母集団と標本

母集団とは分析の結果をもって説明したいデータの全体集合です。母集団の全てのデータを調べることができれば前述の記述統計の手法を適用することができますが、大抵の場合、母集団はとてもなく大きな集団であったり、全数調査が困難⁵だったりします。そこで、標本と呼ばれる母集団から抽出した部分集合のデータが示す特徴を用いて母集団の特徴を推測することになります。しかしながら、標本のとり方よってはデータが偏ってしまい母集団の特徴からはずれてしまう可能性があります。そこで、確率論を用いて標本と母集団とのずれ（誤差の大きさ）を評価し信頼度つきで母集団の特徴を推測する推測統計の出番です。

2.3.2 点推定

点推定は標本から求められる一つの推定値をもって母集団の特徴となる統計量の推定値とする方法です。例えば、前節で出てきた不偏分散は、まさにこの点推定の結果です。不偏推定値と言われることもあります。

母平均、母分散

2.3.3 区間推定

一方、区間推定は文字通り幅を持たせて母集団の特徴となる統計量を推定する方法です。この幅を信頼区間（CI: Confidence Interval）と呼びます。

母平均、母比率、母分散

母平均の区間推定には大きく以下の二つの方法があります。

- 母分散が既知の場合
- 母分散が未知の場合

通常は

⁵ 例えばテレビの世帯視聴率のように全世帯のデータを集めるとコスト的に見合わないような調査

2.3.4 まとめ

2.3.5 演習

演習 4

演習 5

演習 6

2.4 検定

データを分析する前には分析対象となるデータがどのような特徴を持っているか確認しておくことが重要であるとよく言われます。この特徴を見るには要約統計量を使う場合が多いです。要約統計量はデータの分布の特徴を表すもので、記述統計量や基本統計量と呼ばれることもあります。

2.4.1 F 検定

2.4.2 二項検定

2.4.3 まとめ

2.4.4 演習

演習 7

演習 8

演習 9

2.5 分散分析

2.5.1 まとめ

2.5.2 演習

演習 10

演習 11

演習 12

2.6 回帰分析

2.6.1 まとめ

2.6.2 演習

演習 13

演習 14

演習 15

Part III

Wrangle

Chapter 3

Import

Import は

Google Colab を利用する場合は [+コード]（[Ctrl] + [M] + [Ctrl] + [B]）ボタンでコードを追加してコードブロックを挿入してからコードを記載します。コードブロックの移動や削除はブロック右側に表示されているサブメニューで行います。[+テキスト] ボタンでテキストブロックを挿入すればコメントなどを書き込むことができます。

RStudio を利用する場合はメニューから[File]-[New File]-[R Notebook]を実行して R Notebook を作成します。キーボードショートカット[Ctrl/Cmd] + [Alt/Option] + [I] でコードチャンクを挿入しチャンクにコードを記述します。チャング以外にコメントなどを書き込むことができます。

3.1 readr

3.2 readxl

3.3 pdftools

Chapter 4

Tidy

4.1 Tidy Data

4.2 longer

4.3 wider

Chapter 5

Transform

5.1 filter

5.2 rename

5.3 select

5.3.1 select helpers

5.4 mutate

5.5 summarize

Part IV

Visualize

Chapter 6

Base R

6.1 plot

6.2 boxplot

6.3 hist

Chapter 7

ggplot2

Part V

Model/Infer

Chapter 8

Test

Chapter 9

Linear model

Chapter 10

Machine Learning

Part VI

Communicate

Chapter 11

R Markdown

Part VII

Automate

Chapter 12

shiny

Part VIII

APPENDIX

Appendix A

References

文中で参照している文献・資料に関しては文献一覧のページを参照してください。なお、文献一覧は PDF 版では巻末に「Bibliography」として、HTML 版では参照ページの下部の「References」項にまとめてあります。

- R と RStudio のインストールと初期設定 (PDF), 矢内高知工科大学
 - Windows 編¹
 - macOS 編²
 - Linux(Ubuntu) 編³
- 総務省 ICT スキル総合習得プログラム⁴
 - 2017 年度総務省「総務省 ICT スキル総合習得プログラム」事業の成果資料
- r-wakalang⁵
 - 国内最大? の slack コミュニティ

¹https://yukiyanai.github.io/jp/resources/docs/install-R_ubuntu.pdf

²https://yukiyanai.github.io/jp/resources/docs/install-R_macOS.pdf

³https://yukiyanai.github.io/jp/resources/docs/install-R_ubuntu.pdf

⁴https://www.soumu.go.jp/ict_skill/

⁵<https://github.com/tokyor/r-wakalang>

Appendix B

R Basics

R の一番良いところは統計学者が作っているところだ。

R の一番悪いところは統計学者が作っているところだ。

出典¹

R は統計的コンピューティングに特化している言語ですが、その開発は上記にもあるように統計学者が中心となって行われてきました。このような背景があるため他のコンピュータ言語を知っている方が使うと奇妙に感じる言語仕様があるかも知れません。しかし、R の便利な点は統計的コンピューティングを行うに際して必須と言えるベクトル演算がデフォルトで使えることがあります。まずは、このベクトル演算に慣れることから始めます。

コードの表記は以下のように灰色で網掛けされた部分が実行するコード、## から始まる部分が実行結果となります。実行結果の表示がない場合は、次に実行するコードとあわせてコードが表示されます。なお、コード内の # で始まる部分はコメントですので実行されません。

```
1 x <- 1          # 変数への代入時は実行結果は表示されない  
2 x             # 表示させたい場合は、別途、変数のみで実行する
```

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

```
1 print(x)      # もしくは print() 関数を用いる
```

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

¹<https://www.slideshare.net/shuyo/r-4022379>

実行結果の `##` の後に表示されている `[]` 内の数値は実行結果の出力の数を示すためのインデックスです。実行環境の表示幅により表示される値が変わります。例えば 1 から 100 までの数字を表示した場合

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
## [19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
## [37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
## [55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
## [73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
## [91] 91 92 93 94 95 96 97 98 99 100
```

このように 1 行目は最初の値から表示されるので `[1]` に、2 行目は 1 行目で 18 個表示されており 19 個目からの表示となるため `[19]` になっていることが分かります。

なお、以降に出てくるコードは RStudio 上で動作するチュートリアルとしてパッケージ化してあります。興味のある方は パッケージ `kmetrics`² をインストールして使ってみてください。なお、パッケージ概要や動作環境はリンク先のページでご確認ください。

B.1 基本的な演算

最初に R の演算がどのような形式で行われるのかを紹介します。

B.1.1 算術演算

算術演算の基本である加減乗除算の四則演算は他のプログラミング言語や OS に付属の電卓アプリなどと同じです。

```
1 1 + 2      # 加算
```

```
## [1] 3
```

```
1 2 - 3      # 減算
```

```
## [1] -1
```

²<https://github.com/k-metrics/kmetrics>

```
1  3 * 4      # 乗算
```

```
## [1] 12
```

```
1  4 / 5      # 除算
```

```
## [1] 0.8
```

B.1.2 代入

上記の演算結果を変数に代入してみます。代入には代入演算子（`<-`）を用います。変数を使うための変数宣言は不要です。

```
1  w <- 1 + 2
2  x <- 2 * 3
3  y <- 3 - 4
4  z <- 4 / 5
```

B.1.3 代入結果の確認

代入結果を確認するには変数名だけで実行するか `print()` 関数を用います。

```
1  w
```

```
## [1] 3
```

```
1  x
```

```
## [1] 6
```

```
1  y
```

```
## [1] -1
```

```
1   z  
## [1] 0.8
```

```
1   print(w)
```

```
## [1] 3
```

```
1   print(x)
```

```
## [1] 6
```

```
1   print(y)
```

```
## [1] -1
```

```
1   print(z)
```

```
## [1] 0.8
```

B.1.4 合算

全ての変数を合算します。単純に加算演算子 (+) を用いても構いませんが、`sum()` という関数が用意されていますので、これを使います。

```
1   sum(w, x, y, z)      # 3 + 6 + -1 + 0.8
```

```
## [1] 8.8
```

B.1.5 平均

全ての変数の平均値を求めます。平均には

- 算術平均（相加平均）
- 幾何平均（相乗平均）
- トリム平均

がありますが、ここでは算術平均の値を求めます。算術平均の値を求めるには `mean()` という関数を用います。

```
1 mean(w, x, y, z) # (3 + 6 + -1 + 0.8) / 4
```

```
## [1] 3
```

`sum()` 関数と同じ指定をすると計算結果が期待と違います。これは `mean()` 関数がとる引数が `sum()` 関数と異なりひとつに限られるためです。そこで、ベクトル変数を作成する `c()` 関数を用いて以下のように記述します。

```
1 x <- c(w, x, y, z)
2 mean(x)
```

```
## [1] 2.2
```

これで期待通りの値を得ることができました。

B.1.6 変数の上書き

さて、最初に `x` に代入した値は 6 でしたが、この時点では以下のように複数の値が代入されています。

```
1 x
```

```
## [1] 3.0 6.0 -1.0 0.8
```

これは平均値を求める際に `x` という変数を再利用したためですが、このように R では警告なしに既存の変数を上書きすることができます。注意してください。

このように R での演算は、他の言語と同様に何らかの値を変数に入れたり、何らかの値を関数で処理したりします。

B.2 変数

変数の命名規則は tidyverse スタイルガイド（以降、スタイルガイド）³ と呼ばれる記述ルールに準拠することをおすゝめします。スタイルガイドでは変数名に使える文字を以下の組合せであることを推奨しています。

- 英数小文字 (A-Z, a-z)
- 数字 (0 1 2 3 4 5 6 7 8 9)
- アンダースコア (_)

ただし、数字から始まる変数名は R の仕様により使うことができません（エラーになります）。また、日本語の変数名を使うことは可能ですが、様々な環境を考慮すると変数名に日本語を使用することはおすゝめできません。本書はスタイルガイドに準拠した記述になっています。

B.2.1 予約語

プログラミング言語には予約語（Reserved Word）といわれるものがあり予約語は変数名として使えません。R では以下が予約語になっています。

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

また、予約語以外でも変数型や関数に使われている名前を変数名として使うことはおすゝめできません。

B.2.2 データ型

他の言語でも同じですが変数には値を入れることができます。データ型はどのような値のデータが入っているかを識別するためのものです。R の代表的なデータ型には以下のようないがあります。

型	クラス	タイプ	モード	格納モード	備考
実数型	numeric	double	numeric	double	倍精度浮動小数点
整数型	integer	integer	numeric	integer	
複素数型	complex	complex	complex	complex	
論理型	logical	logical	logical	logical	Boolean 型
文字型	character	character	character	character	
日付型	Date	double	numeric	double	Date 型 ^b

^b 日付型には POSIX 型もあります

³<https://style.tidyverse.org/syntax.html#object-names>

Rは開発の経緯から様々な型の見かたがありますが、基本的に同じようなものだとと考えてください。書籍などでよく出てくる`str()`関数が返す型は上表におけるクラスです。

B.2.3 変数型

変数型はどのような形でデータを格納できるかを識別するためのものです。

変数型	クラス	説明
ベクトル型	データ型に同じ	基本となる変数型
因子型	<code>factor</code> , <code>ordered</code>	インデックスを持つ変数型
マトリクス型（行列型）	<code>matrix</code>	二次元のベクトル型
アレイ型（配列型）	<code>array</code>	多次元のベクトル型
データフレーム型	<code>data.frame</code>	等長なベクトル型
リスト型	<code>list</code>	自由度が最も高い変数型

B.2.3.1 ベクトル型

ベクトル型は基本となる変数型です。ベクトル型には一種類のデータ型のデータ（値）しか格納することができません。格納できる個数は任意です。ベクトル型の変数を作成するには`c()`関数を用います。代入して`str()`関数でクラス、長さ（値の個数）と値を確認してみます。長さが1の場合は`c()`関数を省略することができます。

```
1 x <- 2L          # 一つだけ代入する場合 `c()` は省略可能
2 str(x)           # 値が一つだけの場合は長さ表示が省略
```

```
## int 2
```

```
1 x <- c(1, 2, 3)      # 実数の 1 から 3 の三つの値が代入
2 str(x)               # [] の部分が長さ表示
```

```
## num [1:3] 1 2 3
```

文字（型）を代入する場合はクオート（ダブルまたはシングル）で囲みます。

```

1 x <- c("1", "2", "3")    # 文字として数字を代入
2 str(x)

```

```
## chr [1:3] "1" "2" "3"
```

では文字（型）と数字（型）を混在させたらどうなるでしょう？

```

1 x <- c(1L, 2, "3")      # 整数、実数、文字としての数字を代入
2 str(x)                   # 強制型変換により最も柔軟度の高い文字型に

```

```
## chr [1:3] "1" "2" "3"
```

エラーにはならずベクトル型変数は文字型の変数として作成されます。これは強制型変換という処理が行われるためです。強制型変換は複数のデータ型が混在した場合により柔軟度の高いデータ型に自動的に変換する処理で、論理型、整数型、実数型、複素数型、文字型の順に変換されます。強制型変換は便利ですが意図しない変換結果を招く場合もありますので、このような変換が行われることは憶えてください。

B.2.3.2 因子型

因子型は名義尺度や順序尺度の変数を扱う際に便利な変数型です。前述のベクトル型変数に格納された値を識別するためのインデックスがついたデータベーステーブルのような仕組みを持っています。因子型を作成するには `factor()` 関数を使う順序なしの因子型と `ordered()` 関数を使う順序ありの因子型があります。どちらも `levels`（水準）という属性がつきます。この水準にインデックスとしての役割があります。後述のデータフレーム型の中で使うと「層別」という処理が楽になります。

```

1 x <- factor(c("A", "B", "AB", "0", "A", "A", "A", "B"))
2 str(x)

```

```
## Factor w/ 4 levels "A","AB","B","0": 1 3 2 4 1 1 1 3
```

```
1 levels(x)
```

```
## [1] "A"  "AB" "B"  "0"
```

`ordered` 型は `levels` に順序がついている点が `factor` 型と異なる点です。

```

1 x <- ordered(c("A", "B", "AB", "0", "A", "A", "A", "B"))
2 str(x)

```

```
## Ord.factor w/ 4 levels "A"<"AB"<"B"<"0": 1 3 2 4 1 1 1 3
```

```
1 levels(x)
```

```
## [1] "A"   "AB"  "B"   "0"
```

B.2.3.3 マトリクス型

マトリクス型は文字通り二次元配列を扱うための変数型です。作成するには `matrix()` 関数を利用します。引数のベクトル型を列方向から二次元に展開します。

```
1 matrix(c(10, 20, 30, 40, 50, 60), 2, 3)      # 2行3列のマトリクス型
```

```

##      [,1] [,2] [,3]
## [1,]    10    30    50
## [2,]    20    40    60

```

展開方向を変えるには引数自体を変える方法もありますが `byrow` オプションを利用する方がスマートです。

```
1 matrix(c(10, 20, 30, 40, 50, 60), 2, 3, byrow = TRUE)
```

```

##      [,1] [,2] [,3]
## [1,]    10    20    30
## [2,]    40    50    60

```

整数型や実数型の数値だけでなく文字型など他のデータ型も扱えます。

```
1 matrix(c("a", "b", "x", "y"), 2, 2)
```

```

##      [,1] [,2]
## [1,] "a"  "x"
## [2,] "b"  "y"

```

```
1 matrix(c("a", "b", "x", "y"), 2, 2, byrow = TRUE)
```

```
##      [,1] [,2]
## [1,] "a"  "b"
## [2,] "x"  "y"
```

マトリクス型を使う機会はあまり多くありませんが、関数の引数や返り値として使われることがありますので、どういう変数型なのかを憶えておいてください。

B.2.3.4 アレイ型

アレイ型はマトリクス型を複数ならべたような多次元配列を扱うための変数型です。作成するには `array()` 関数を利用します。第一引数で指定したベクトル型のデータを第二引数で指定した構造（行数、列数、次元数）にしたがって多次元配列に展開します。展開は列方向のみで `matrix()` 関数がもつ `byrow` オプションはありません。

```
1 array(c(1:12), c(2, 3, 2))
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]     1     3     5
## [2,]     2     4     6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]     7     9    11
## [2,]     8    10    12
```

なお、第一引数のデータ数が第二引数で指定した総数（行数 × 列数 × 次元数）よりも少ない場合は警告メッセージ（以降、ワーニング）を出力して第一引数のデータを先頭からリサイクルして埋めます。これは `matrix()` 関数でも同様です。アレイ型もマトリクス型同様に使うと機会はあまり多くありません。

B.2.3.5 データフレーム型

データフレーム型はデータベースのテーブルのような形式の変数型で最も使われる変数型と言えます。制約として全ての列の行数が等しい必要があります。すなわち、等

長のベクトル型変数を列方向にならべた変数型と言えます。例えば前出の `iris` データセットはデータフレーム型変数です。

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
...	NA

150 のデータを持つ実数型のベクトル型変数が四つ、150 のデータを持つ因子型のベクトル型変数が一つ、計五つのベクトル型変数から構成されています。加えて個々の行がひとつの計測結果になっています。様々な車の諸元をまとめた `mtcars` データセットの方がイメージを掴みやすいかも知れません。

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.62	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.88	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
...

このようにデータフレーム型は表形式のデータを扱うのに非常に便利な変数型です。データフレーム型を作成するには `data.frame()` 関数を用います。

```

1 x <- data.frame(col1 = c(1:5),
2                   col2 = c("A", "B", "C", "D", "E"),
3                   col3 = c(10, 11, 12, 13, 14),
4                   col4 = c(TRUE, TRUE, FALSE, TRUE, FALSE))
5 str(x)

```

```

## 'data.frame':   5 obs. of  4 variables:
## $ col1: int  1 2 3 4 5
## $ col2: chr  "A" "B" "C" "D" ...
## $ col3: num  10 11 12 13 14
## $ col4: logi  TRUE TRUE FALSE TRUE FALSE

```

```
1 x
```

```
##   col1 col2 col3  col4
```

```
## 1   1   A   10  TRUE
## 2   2   B   11  TRUE
## 3   3   C   12 FALSE
## 4   4   D   13  TRUE
## 5   5   E   14 FALSE
```

なお、`data.frame()` 関数では R のバージョンによっては文字型のデータを因子型として扱うことがあります。このよう場合には `stringsAsFactors = FALSE` オプションを指定すると強制的に文字型として扱えるようになります。逆に因子型として扱いたい場合には `stringsAsFactors = TRUE` を指定することで因子型になりますがこちらは非推奨なオプション指定です。

B.2.3.6 リスト型

リスト型はデータ格納の自由度が高いため関数の返り値として使われることが多い変数型です。データフレーム型との大きな違いは不等長のベクトル型を複数格納できる点にあります。さらにマトリクス型やデータフレーム型、リスト型など基本的に全ての変数型をリスト型内に格納可能です。

```
1 x <- list(c(1:10), c(0.5:5.5), seq(1, 4, 0.2), c("A", "B", "AB", "O"), x)
2 str(x)
```

```
## List of 5
## $ : int [1:10] 1 2 3 4 5 6 7 8 9 10
## $ : num [1:6] 0.5 1.5 2.5 3.5 4.5 5.5
## $ : num [1:16] 1 1.2 1.4 1.6 1.8 2 2.2 2.4 2.6 2.8 ...
## $ : chr [1:4] "A" "B" "AB" "O"
## $ : 'data.frame':    5 obs. of  4 variables:
##   ..$ col1: int [1:5] 1 2 3 4 5
##   ..$ col2: chr [1:5] "A" "B" "C" "D" ...
##   ..$ col3: num [1:5] 10 11 12 13 14
##   ..$ col4: logi [1:5] TRUE TRUE FALSE TRUE FALSE
```

```
1 x
```

```
## [[1]]
## [1] 1 2 3 4 5 6 7 8 9 10
##
## [[2]]
## [1] 0.5 1.5 2.5 3.5 4.5 5.5
```

```

## 
## [[3]]
## [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8 4.0
##
## [[4]]
## [1] "A"  "B"  "AB" "O"
##
## [[5]]
##   col1 col2 col3 col4
## 1     1     A    10 TRUE
## 2     2     B    11 TRUE
## 3     3     C    12 FALSE
## 4     4     D    13 TRUE
## 5     5     E    14 FALSE

```

B.3 定数

変数はその名の通り値を変更できますが、定数は特別な意味を持った値を保持するためもので予約語になっています。定数にもクラスがあり、下表のようになっています。

定数	クラス	意味・説明
TRUE	logical	Boolean の真を意味する (1 と等価)
FALSE	logical	Boolean の偽を意味する (0 と等価)
NULL	NULL	空（何も存在しない）を意味する (0 や NA とは異なる)
NA	logical	欠損値 (Not Available で、データの欠損を意味する)
NaN	numeric	非数 (Not a Number で非数 ^c を表す)
Inf	numeric	無限大 (0 除算時等は NaN ではなく Inf/-Inf)

^c $\frac{0}{0}$ のような数値では表現できないものを意味する

NULL を除く定数はデータ型のクラスですので強制型変換の対象となります。

```
1 c(10L, 2.5, 3 + 4i, TRUE, NaN, NA, Inf, -Inf)
```

```
## [1] 10.0+0i 2.5+0i 3.0+4i 1.0+0i NaN+0i NA Inf+0i -Inf+0i
```

B.3.1 NA の型

欠損値を示す NA にはデータ型を明示的に示すためのバリエーションがあります。関数によっては明示的な NA を指定する必要があります。

NA	データ型
NA_integer_	整数型
NA_real_	実数型
NA_complex_	複素数型
NA_character_	文字型

B.4 検査・変換

R では変数内に複数のデータ型が混在している場合は前述のような強制型変換が行われますので、タイプなどがあると知らぬ間に意図しないデータ型になっていることがあります。作成した変数のデータ型を検査するために `is.()` 関数群が用意されています。

データ型	関数	備考
論理型	<code>is.logical()</code>	
整数型	<code>is.integer()</code>	
実数型	<code>is.double()</code>	
数値型	<code>is.numeric()</code>	整数型または実数型の場合 TRUE が返る
複素数型	<code>is.complex()</code>	
文字型	<code>is.character()</code>	

同様に変数型を検査するための `is.()` 関数群も用意されています。

変数型	関数	備考
ベクトル型	<code>is.vector()</code>	
因子型	<code>is.factor(),is.ordered()</code>	
マトリクス型	<code>is.matrix()</code>	
アレイ型	<code>is.array()</code>	
データフレーム型	<code>is.data.frame()</code>	
リスト型	<code>is.list()</code>	

定数は比較演算子 (`==` や `!=`) では比較できませんので判別には `is.()` 関数群を使います。

定数	関数	備考
NULL	<code>is.null()</code>	NULL 値か否か
NA	<code>is.na()</code>	欠損値か否か
NaN	<code>is.nan()</code>	非数か否か
inf	<code>is.infinite()</code>	無限値か否か

定数	関数	備考
	is.finite()	有限値か否か

B.5 演算子

R の演算子には算術的・論理的な演算子だけでなく変数内の特定位置を参照のための演算子や任意の定義が可能な特殊演算子があります。

B.5.1 参照演算子

変数の中の値を参照する方法は変数型により異なりますが、基本的には参照演算子もしくはアクセス演算子と呼ばれる演算子を用います。

B.5.1.1 [演算子]

ベクトル型の特定位置の値を参照する演算子です。記述の際は [] と閉じる必要があります。

演算子はベクトル型系の要素を参照するための演算子です。例えば 5 番目の値を参照するには以下のようにします。

```
1 x <- c(1:10)
2 x
```

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

```
1 x[5]
```

```
## [1] 5
```

マトリクス型では、行・列・セルの三通りの参照が可能です。

```
1 x <- matrix(c(1:12), nrow = 3)
2 x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
1 x[1, ]
```

```
## [1] 1 4 7 10
```

```
1 x[, 1]
```

```
## [1] 1 2 3
```

```
1 x[2, 3]
```

```
## [1] 8
```

アレイ型でも同様の参照が可能です。ただし、マトリクス型とは異なり次元が絡んできますので、表示は少しややこしくなります。以下の 2×2 の 4 次元アレイで説明します。

```
1 x <- array(c(1:16), dim = c(2, 2, 4))
2 x
```

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

```
##  
## , , 3  
##  
## [,1] [,2]  
## [1,] 9 11  
## [2,] 10 12  
##  
## , , 4  
##  
## [,1] [,2]  
## [1,] 13 15  
## [2,] 14 16
```

第1次元を参照します。

```
1 x[, , 1]
```

```
## [,1] [,2]  
## [1,] 1 3  
## [2,] 2 4
```

第1次元の1行目を参照します。

```
1 x[1, , 1]
```

```
## [1] 1 3
```

第1次元の1列目を参照します。

```
1 x[,1 , 1]
```

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

全次元の1行目を参照します。参照結果は列が各次元になる点に注意してください。

```
1 x[1, , ]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]     1     5     9    13
## [2,]     3     7    11    15
```

全次元の1列目を参照します。行の参照と同様に参照結果は列が各次元になります。

```
1 x[, 1, ]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]     1     5     9    13
## [2,]     2     6    10    14
```

全次元の1行1列目を参照します。

```
1 x[1, 1, ]
```

```
## [1] 1 5 9 13
```

B.5.1.2 \$ 演算子

\$ 演算子はデータフレーム型やリスト型の要素を参照するための演算子です。

```
1 x <- data.frame(blood = c("A", "B", "A", "O", "A"), age = c(18, 25, 22, 35, 19))
2 x
```

```
##   blood age
## 1     A  18
## 2     B  25
## 3     A  22
## 4     O  35
## 5     A  19
```

```
1 x$blood
```

```
## [1] "A" "B" "A" "O" "A"
```

さらに要素内を参照するには前述の [演算子と組み合わせます。

```
1 x$blood[3]
```

```
## [1] "A"
```

```
1 ##### Operators #####
```

リスト型を \$ 演算子で参照する場合は要素が names 属性を持っていることが前提です。以下のリスト型変数では \$ 表示の後ろに要素名が表示されいている blood, data が \$ 演算子で参照可能です。

```
1 x <- list(blood = c("A", "B", "AB", "O"), c("M", "F"),
2           data = data.frame(blood = c("A", "B", "A", "O", "A"),
3                               age = c(18, 25, 22, 35, 19)))
4 str(x)
```

```
## List of 3
## $ blood: chr [1:4] "A" "B" "AB" "O"
## $      : chr [1:2] "M" "F"
## $ data :'data.frame': 5 obs. of 2 variables:
##   ..$ blood: chr [1:5] "A" "B" "A" "O" ...
##   ..$ age  : num [1:5] 18 25 22 35 19
```

要素名が表示されていない二番目の要素を参照するには [[演算子を用います。

```
1 x[[2]]
```

```
## [1] "M" "F"
```

さらに要素内の値を参照する場合は前述のデータフレーム型同様に [演算子を用います。

```
1 x$blood[2]
```

```
## [1] "B"
```

```
1 x[[2]][1]
```

```
## [1] "M"
```

B.5.2 単項演算子

単項演算子は文字通り一つの項に作用する演算子です。単項演算子には算術演算子の-（マイナス）と論理型演算子の!（否定, NOT）があります。

```
1 x <- c(1:5)
```

```
2 x
```

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

```
1 -x
```

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

```
1 x <- c(TRUE, TRUE, TRUE, FALSE, TRUE)
```

```
2 x
```

```
## [1] TRUE TRUE TRUE FALSE TRUE
```

```
1 !x
```

```
## [1] FALSE FALSE FALSE TRUE FALSE
```

単項演算子

B.5.3 二項演算子

二項演算子とは二つの項に作用する演算子です。

B.5.3.1 算術演算子

演算子は四則演算（加算、減算、乗算、除算）ならびに、べき算（べき乗算）、整数除算（商、剰余）の六つがあります。

```
1 a <- c(1:10)
2 b <- c(10:1)
3 a
```

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

```
1 b
```

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

```
1 a + b          # 加算
```

```
## [1] 11 11 11 11 11 11 11 11 11 11
```

```
1 a - b          # 減算
```

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

```
1 a * b          # 乗算
```

```
## [1] 10 18 24 28 30 30 28 24 18 10
```

```
1 a / b          # 除算
```

```
## [1] 0.1000000 0.2222222 0.3750000 0.5714286 0.8333333 1.2000000
## [7] 1.7500000 2.6666667 4.5000000 10.0000000
```

```
1 a ^ b          # べき乗算
```

```
## [1] 1 512 6561 16384 15625 7776 2401 512 81 10
```

```
1 a %/% b      # 整数除算（商）
```

```
## [1] 0 0 0 0 0 1 1 2 4 10
```

```
1 a %% b      # 整数除算（剰余）
```

```
## [1] 1 2 3 4 5 1 3 2 1 0
```

B.5.3.2 比較演算子

比較演算子は関係演算子とも呼ばれ、二変数の関係を調べる演算子です。同値関係を調べる等号記号や大小関係を調べる不等号などがこれにあたります。返り値は論理型となります。

小なり	大なり	小なりイコール	大なりイコール	イコール	ノットイコール
<	>	<=	>=	==	!=

例えば以下の二つのベクトル型変数に対する比較を行ってみます。

```
1 a
```

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

```
1 b
```

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

```
1 a < b
```

```
## [1] TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE
```

```
1 a > b
```

```
## [1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE
```

```
1 a <= b
```

```
## [1] TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE
```

```
1 a >= b
```

```
## [1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE
```

```
1 a == b
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
1 a != b
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

B.5.3.3 論理演算子

論理演算子はブール関数を評価するものです。論理積（AND）・論理和（OR）は演算対象により二種類の演算子があります。

演算	演算子	説明
論理積	&	AND (ベクトル演算用)
論理和		OR, (ベクトル演算用)
排他的論理和	xor	eXclusive OR (ベクトル演算用)
否定	!	NOT (単項演算子, ベクトル演算可)
論理積	&&	条件式における論理積
論理和		条件式における論理和

B.5.4 特殊演算子

特殊演算子は% 文字と% 文字で任意の文字を挟んだ演算子で二項演算子の一種です。算術演算子で出てきた整数除算（商、剰余）は厳密に言えば特殊演算子に分類されますが、本書では算術演算子として分類しています。また、任意の特殊演算子を定義することも可能で、パッケージによっては様々な特殊演算子を用意しているものもあります。

特殊演算子	演算内容
%*%	内積 (スカラー積)
%in%	マッチング
%o%	外積 (ベクトル積)
%x%	クロネッカー積

B.5.5 優先順位

演算子には下表のような優先順位があります。優先順位を変えたい場合は数学と同様に()を利用して明示的に優先順位を指定をしてください。下記以外はコンソールで? Syntax+[Enter]と打てばヘルプが表示され確認できます。

演算子	説明	順位
::	名前空間へのアクセス (パッケージ内への明示的アクセス)	高
\$	要素へのアクセス (データフレーム型、リスト型)	
[], [[]]	要素へのアクセス (ベクトル型、マトリクス型、アレイ型、リスト型)	

演算子	説明	順位
<code>^</code>	べき乗	
<code>-</code>	マイナス（単項演算子、 <code>+</code> も単項演算子として使用可）	
<code>:</code>	等差数列 (<code>c(1:10)</code> のような数列)	
<code>%nay%</code>	特殊演算子（二項演算子）	
<code>*, /</code>	乗算、除算（二項演算子）	
<code>+, -</code>	加算、減算（二項演算子）	
<code><, >, <=, >=</code>	比較演算子（大小関係）	
<code>==, !=</code>	比較演算子（同値関係、大小関係と優先順位は同列）	
<code>!</code>	否定（単項演算子）	
<code>&, &&, , </code>	論理積、論理和（論理演算子）	
<code>~</code>	フォーミュラ	
<code><-</code>	代入演算子	低

B.6 制御文

制御文はプログラムの流れをコントロールするためのもので大抵の言語で予約語になっています。制御文には条件分岐と繰り返し（ループ）の二種類があります。
あ

B.6.1 条件分岐

条件分岐には以下のようなものがあります。その他、パッケージなどで条件分岐のための関数が提供されています。

文・関数	説明
<code>if else</code>	基本的な条件分岐（予約語）
<code>switch</code>	条件が多数に分岐する場合に便利（予約語）
<code>ifelse</code>	Excel の IF 関数に似た条件分岐（関数）

B.6.1.1 if, else

`if` 文と `else` 文は最も基本的な条件分岐です。評価式には論理演算子または論理型変数を用います。コーディングスタイルとして以下のどちらも可能です。

```

1 x <- FALSE
2
3 if (x != TRUE) print("TRUE") else print("FALSE")

```

```
## [1] "TRUE"
```

```

1 if (x == TRUE) {
2   print("TRUE")
3 } else {
4   print("FALSE")
5 }

```

```
## [1] "FALSE"
```

`if` 文は入れ子にしたり `else if` 文として組み合わせて使うことも可能です。

```

1 x <- 10L
2 y <- 5
3
4 if ((x > 5) && (y < x)) {
5   print("match, (x > 5) && (y < x)")
6 } else if ((x > 5) && (y >= x)) {
7   print("match, (x > 5) && (y >= x)")
8 } else {
9   print("else")
10 }

```

```
## [1] "match, (x > 5) && (y < x)"
```

B.6.1.2 switch

分岐する条件の数が多い場合は `if` 文でなく `switch` 文を利用する方が便利です。`if` 文と同じで評価式は `TRUE` か `FALSE` が单一で返るようにしなければなりません。注意しなければならないのは、引数により構文が異なる点です。

B.6.1.2.1 引数が整数の場合 引数に整数 n を指定した場合、 n 番目の処理文の結果が返ります。

```

1 x <- 2
2 switch(x,
3     "x is 1",      # 分岐のための引数
4     "x is 2",      # 1番目の処理文
5     "Error")       # 2番目の処理文
6                         # 3番目の処理文

```

```
## [1] "x is 2"
```

注意しなければならないのは条件分岐数と一致しない場合は NULL が返される点です。

```

1 x <- 5L
2 is.null(switch(x,          # 分岐のための引数
3     "x is 1",    # 1番目の処理文
4     "x is 2",    # 2番目の処理文
5     "Error"))    # 3番目の処理文

```

```
## [1] TRUE
```

B.6.1.2.2 引数が文字の場合 一方、引数が文字の場合、if/else 文と同様の処理が行われます。if/else 文と異なるのは else 文に相当する分岐が途中になっていても正しく処理してくれる点です。

```

1 x <- "2"
2 switch(x,
3     "1" = "x is 1",    # 引数が"x is 1"と一致する場合 ('if (x == "1")' に等価)
4     "x is others",   # 一致するものがいない場合 ('else' に等価)
5     "2" = "x is 2")    # 引数が"x is 2"と一致する場合 ('if (x == "2")' に等価)

```

```
## [1] "x is 2"
```

引数に整数を指定しても動作しますが、引数が整数の場合と同様の動きをします。

```

1 x <- 3
2 switch(x,           # 分岐のための引数が整数になると
3     "1" = "x is 1",   # 1番目の処理文
4     "x is others",    # 2番目の処理文
5     "2" = "x is 2")   # 3番目の処理文

## [1] "x is 2"

```

B.6.1.3 ifelse

`base::ifelse()` は予約語でなく関数です。`if/else` 文と異なるのはベクトル型の評価が一度に行える点です。第一引数に `TRUE` か `FALSE` が返る評価式であればベクトル型でも構いません。

```

1 ifelse(TRUE, 1, 0)

```

B.6.2 繰り返し

繰り返しは文字通り処理を任意の回数繰り返す場合に用いるもので予約語になっています。繰り返し文の処理は時間がかかるため Rにおいては好ましくなく繰り返しは使わずベクトル演算で処理すべきと言われていますが、R-3.4.0 から JIT コンパイラと呼ばれる繰り返し処理の高速化がデフォルトで有効化されており今後は処理記述の流れが変わる可能性があります。処理の高速化についてはこちらの参考資料⁴で確認してください。なお、繰り返し処理で注意すべき点は繰り返し文中では明示的に出力を指定しないと出力がなされない点です。

文	説明
<code>for</code>	条件式に与えたベクトルやリストが空になるまで任意の回数繰り返す
<code>while</code>	条件式に与えた条件が成立している限り繰り返す
<code>repeat</code>	無限に繰り返すが繰り返し処理中の <code>break</code> 文で繰り返しを終了できる

また、繰り返しを条件式以外で変更する処理用の文として以下が用意されています。

⁴<http://masato-613.hatenablog.com/entry/2017/04/25/064632>

これらも予約語です。

文	説明
next	この文が実行された時点で強制的に次の繰り返し処理に入ります
break	この文が実行された時点で繰り返し処理を終了します

B.6.2.1 for

for 文は最も基本となる繰り返し処理で、条件式としてベクトルやリストを指定できる点が他の言語と異なる点です。

```
1 for (i in c(1:5, 7, 9:15)) {  
2   if (i == 4) {  
3     next  
4   } else if (i >= 10) {  
5     break  
6   } else {  
7     print(as.character(i))  
8   }  
9 }
```

```
## [1] "1"  
## [1] "2"  
## [1] "3"  
## [1] "5"  
## [1] "7"  
## [1] "9"
```

B.6.2.2 while, repeat

while 文と repeat 文については、あまり使うこともないと思いますので省略します。

Appendix C

演習解答例

演習 1

標本分散 (s^2) と不偏分散 ($\hat{\sigma}^2$) の関係式(2.3)を用いて R で以下の二つのデータの標本分散を求めなさい。

```
1 x <- c(1, 5, 5, 5, 9)
2 y <- c(1, 1, 5, 9, 9)
```

解答例

```
1 n <- length(x)
2 (n - 1) / n * var(x)
```

```
## [1] 6.4
```

```
1 n <- length(y)
2 (n - 1) / n * var(y)
```

```
## [1] 12.8
```

参考までに不偏分散は以下の通りです。

```
1 var(x)
```

```
## [1] 8
```

```
1 var(y)
```

```
## [1] 16
```

演習 2

欠損値を含む以下のデータの平均値 (\bar{x})、不偏分散 ($\hat{\sigma}^2$) ならびに標本分散 (s^2) を求めなさい。

```
1 z <- c(1, 5, NA, 7, 9)
```

ヒント：`na.rm` オプションを使います

解答例

```
1 mean(z, na.rm = TRUE)
```

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

```
1 var(z, na.rm = TRUE)
```

```
## [1] 11.66667
```

```
1 variance <- function(x, na.rm = FALSE) {
2   n <- sum(!is.na(x))
3   return ((n - 1) / n * var(x, na.rm = na.rm))
4 }
5 variance(z, na.rm = TRUE)
```

```
## [1] 8.75
```

補足

`length()` 関数は指定したデータの長さ（値の個数）を求める関数です。欠損値（NA）も値としてカウントされます。`length()` 関数には `na.rm` オプションはありません。

```
1 z
## [1] 1 5 NA 7 9
```

```
1 length(z)
```

```
## [1] 5
```

そこで、値が NA か否かを調べる `is.na()` 関数を用います。

```
1 is.na(z)
## [1] FALSE FALSE TRUE FALSE FALSE
```

返り値は論理型（TRUE/FALSE）ですが、TRUE は 1、FALSE は 0 と等価であることを利用し、論理演算子の否定（!）と合計を求める `sum()` 関数を組み合わせると NA を除くデータの長さを求めることができます。

```
1 sum(!is.na(z))
## [1] 4
```

不偏分散を求める `var()` 関数は NA が含まれるデータを指定する場合は、NA を除外するか否かを指定する `na.rm` オプションを指定する必要があります。

```
1 var(z)
## [1] NA
```

`na.rm = TRUE` オプションを指定することで、NA を含むデータの不偏分散を求ることができます。

```
1 var(z, na.rm = TRUE)
```

```
## [1] 11.66667
```

演習 3

iris データセットの各変量に対する四分位数を求めなさい。

解答例

iris データセットは R に標準で組み込まれているサンプルデータです。iris と実行するだけでデータを表示することができますが、全てを表示すると長いのでここでは先頭の六行だけを head() 関数を用いて表示します。

```
1 head(iris)
```

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

一番上の行は変量名（変数名）で、一番右側の Species は文字変量ですので、これを除いた四変量（Sepal.Length, Sepal.Width, Petal.Length, Petal.Width）に対して四分位数を求める quantile() 関数を適用します。

```
1 quantile(iris$Sepal.Length, probs = c(0.25, 0.50, 0.75))
```

```
## 25% 50% 75%
## 5.1 5.8 6.4
```

```
1 quantile(iris$Sepal.Width, probs = c(0.25, 0.50, 0.75))
```

```
## 25% 50% 75%
## 2.8 3.0 3.3
```

```
1 quantile(iris$Petal.Length, probs = c(0.25, 0.50, 0.75))
```

```
## 25% 50% 75%
## 1.60 4.35 5.10
```

```
1 quantile(iris$Petal.Width, probs = c(0.25, 0.50, 0.75))
```

```
## 25% 50% 75%
## 0.3 1.3 1.8
```

\$ は参照演算子と呼ばれるデータセットの変量名を用いてデータを参照するための演算子です。

補足

解答例のように個々の変量毎に `quantile()` 関数を適用するより `apply()` 関数による繰り返し処理を用いた方が簡単です。`apply()` 関数は `iris` データセットのような形式のデータに対して行方向 (`MARGIN = 1`)、または、列方向 (`MARGIN = 2`) に `FUN` オプションで指定する関数を適用します。`FUN` オプションの後ろには `FUN` オプションで指定した関数（この場合は `quantile()` 関数）へ渡す引数を指定することができます。`iris` データセットは五列目の `Species` が文字変量ですので `iris[-5]` と指定することで取り除きます。

```
1 apply(iris[-5], MARGIN = 2, FUN = quantile, probs = c(0.25, 0.50, 0.75))
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
## 25%	5.1	2.8	1.60	0.3
## 50%	5.8	3.0	4.35	1.3
## 75%	6.4	3.3	5.10	1.8

このように `apply()` 関数は同じ処理を繰り返したい場合には非常に有用な関数ですので憶えておいて損はありません。

演習 4**解答例****演習 5****解答例****演習 6****解答例****演習 7****解答例****演習 8****解答例****演習 9****解答例****演習 10****解答例****演習 11****解答例****演習 12****解答例****演習 13****解答例**

Appendix D

Environments 2

Rについて学ぶ前にRが使えるように環境を構築する必要がありますが、環境構築は初学者にとって厄介な部分でもあります。そこで、本書では学習レベルに合わせて以下のように環境を使い分けることをおすすめします。

学習フェーズ	環境	備考
基礎学習フェーズ	Google Colaboratory ¹	要 Google アカウント
応用学習フェーズ	RStudio Cloud ²	beta edition

環境を構築するための基本的な知識がある方は最初から RStudio Desktop (以降、RStudio) を利用しても構いません。

D.1 Google Colaboratory

Rの言語仕様など基礎的な学習フェーズでは環境構築の手間がかからないクラウド型のGoogle Colaboratory (以降、Google Colab) の利用をおすすめします。Google ColabではJupyter Notebookというデータ分析用のツールが使えます。ただし、デフォルトの状態でRを使うのは少し不便なので、以下の手順でファイルを準備します。

1. ブラウザでGoogleアカウントにログインする
2. Google Colabを開く
3. R用のテンプレートファイルをアップロードする
4. Rのコードが実行できることを確認する

¹<https://colab.research.google.com/?hl=ja>

²<https://rstudio.cloud/>

5. アップロードしたファイルを Google ドライブに保存する

D.1.1 Login Google

Google Colab は名前通り Google が提供しているサービスですので Google のアカウントを持っていることが前提になります。また、Chrome 系（含む Chromium 系）のブラウザで利用することをおすゝめします。

まず、ブラウザで Google³ のページを開きます。ページの右上に [ログイン] と表示されている場合は [ログイン] をクリックしてログインしておきます。

D.1.2 Open Google Colab

Google で Google Colab を検索して Colaboratory - Google Colab⁴ のリンクをくと以下のような画面が表示されます。

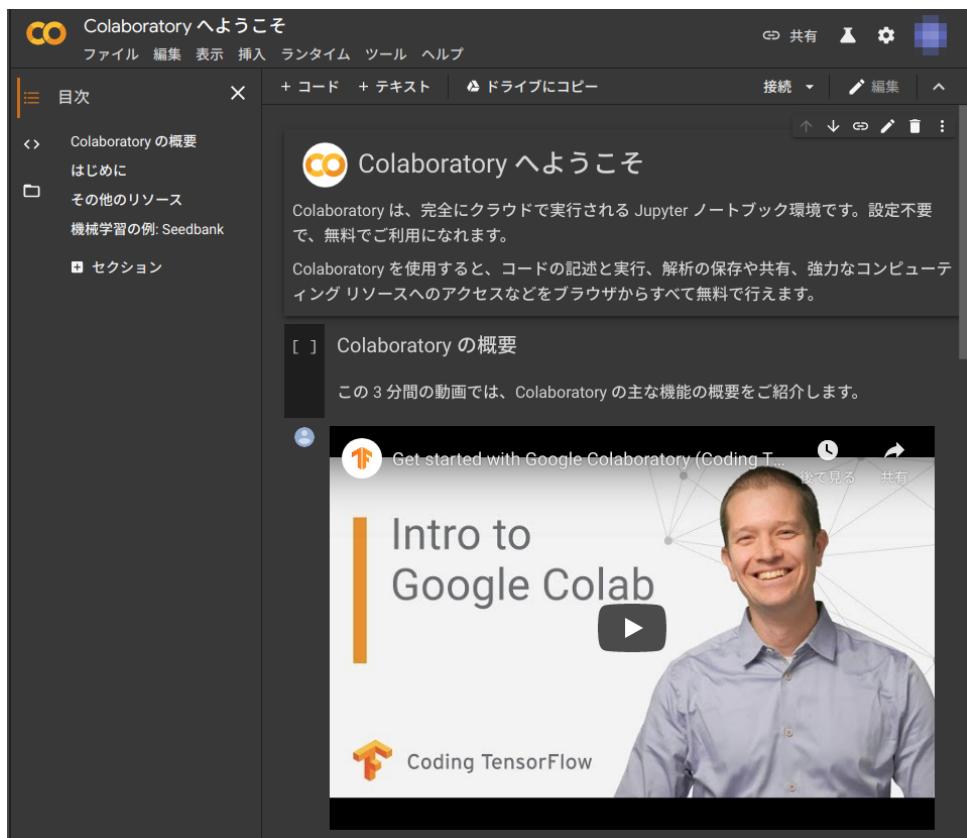


Figure D.1: Google Colab, Theme: dark

³<https://www.google.co.jp>

⁴<https://colab.research.google.com/notebooks/welcome.ipynb?hl=ja>

画面テーマは右上の歯車ボタンから変更できます。

D.1.3 Upload Template

Google Colab が立ち上がりましたら上部にあるメニュー [ファイル] - [ノートブックをアップロード...] を実行します。

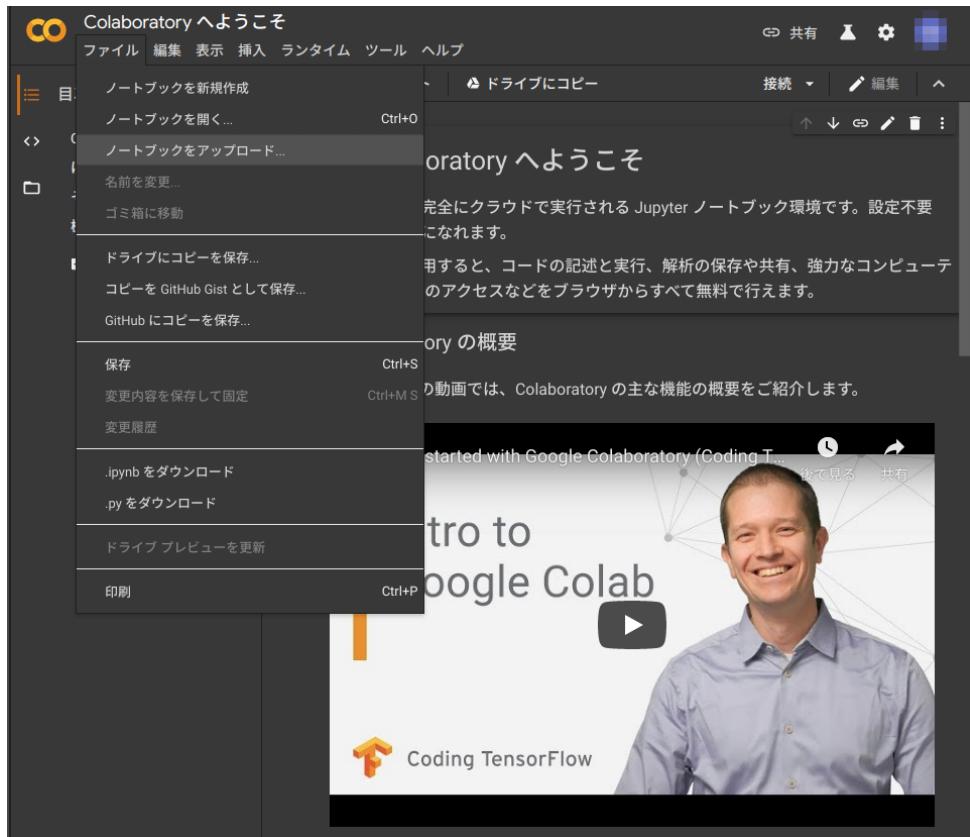


Figure D.2: Upload notebook file

アップロード用のダイアログが開きますので [GitHub] タブをクリックし、上段のライン（画像の青線部分）に下記の URL を入力します。入力後、右端にある虫眼鏡アイコンをクリックします。

<https://gist.github.com/k-metrics/464ffbbd4d00e328560cd55966e7d4b8>

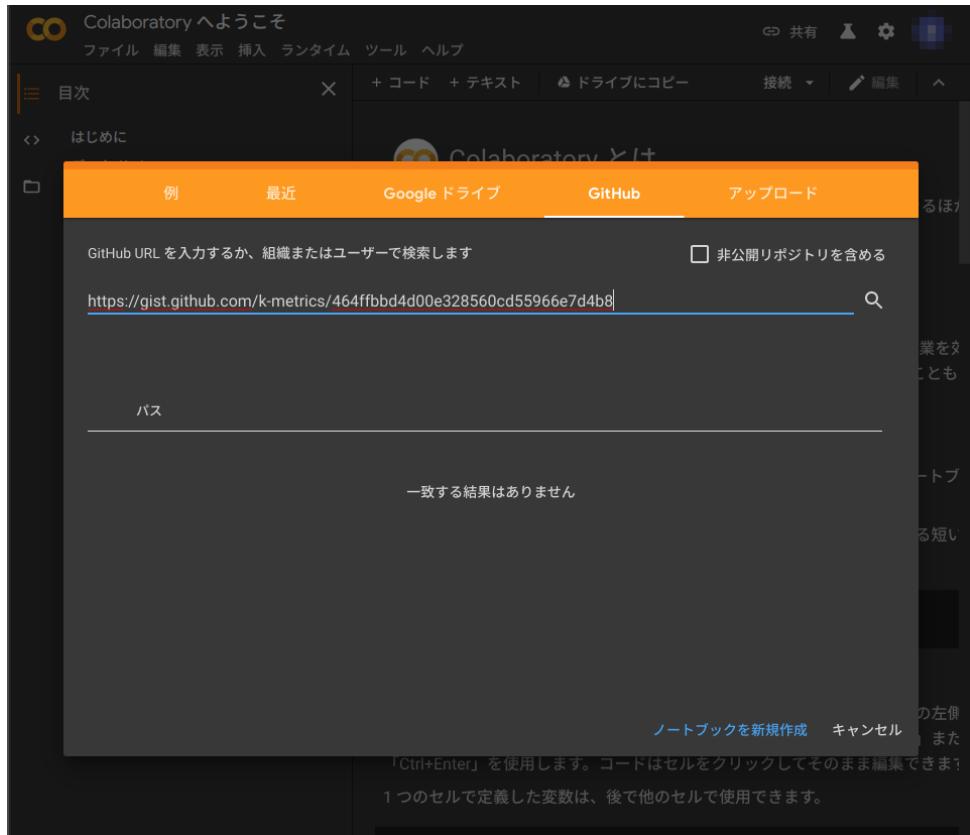


Figure D.3: Upload from GitHub

テンプレートがアップロードされ表示されます。

D.1.4 Run R code

テンプレートがアップロードできましたらテンプレートファイルの記述にしたがってコードを実行してみます。その際に下記のようなダイアログが表示されますが認証情報などを読み取ることはできませんので [このまま実行] をクリックしてください。

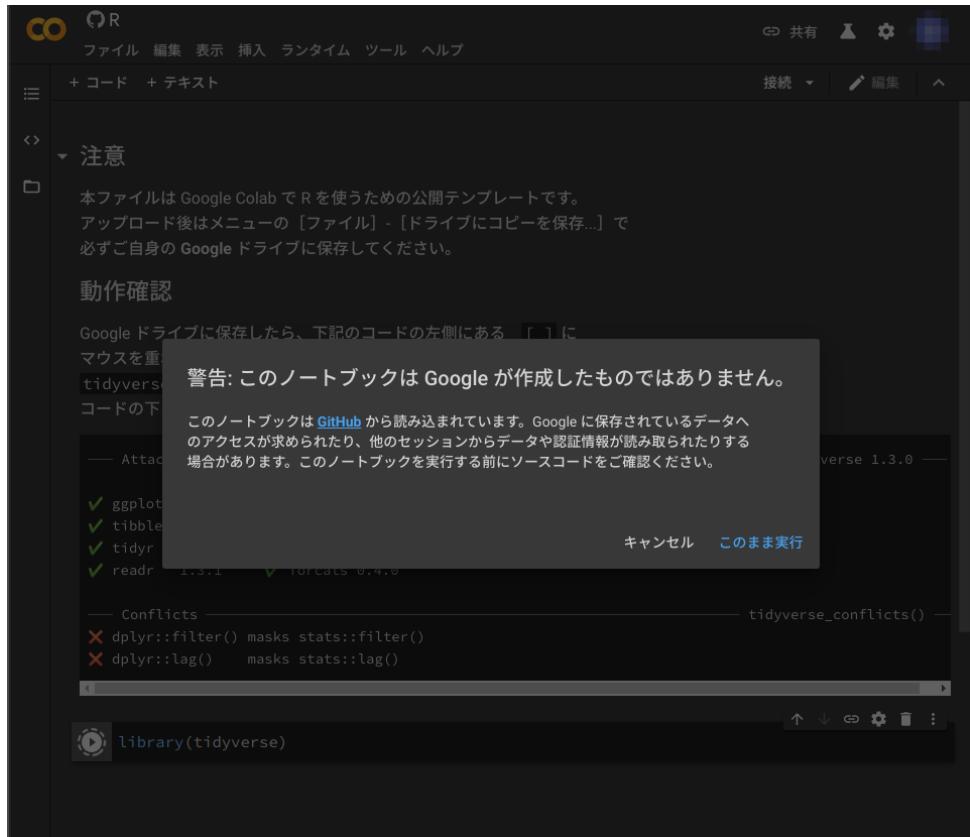


Figure D.4: Warning dialog

サーバ（ホスト型ランタイム）との接続するため実行までに多少時間がかかります。

D.1.5 Save File

コードの実行が確認できましたらメニューの [ファイル] - [ドライブにコピーを保存...] を実行してコピーを Google Drive に保存します。以降、この保存したファイルを利用してください。

D.2 RStudio Cloud

Google Colab では R Markdown などのレポーティング機能は使用できませんので、このような場合にはクラウド上で RStudio が利用できる RStudio Cloud が便利です。RStudio Cloud は統合開発環境の RStudio だけでなく種々のチュートリアルコンテンツを備えています。

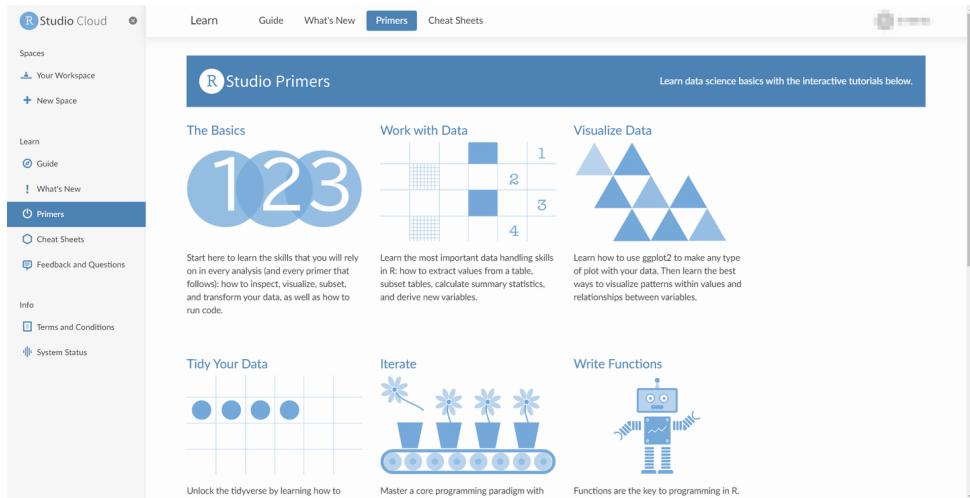


Figure D.5: RStudio Cloud, beta

執筆時点では無償で利用することができ、無制限のプロジェクトとプライベートプロジェクトの作成が可能です。RStudio Cloud を利用するにはアカウントを取得するだけです。

1. ブラウザで RStudio Cloud ⁵ を開く
2. 右上の [sign up] をクリックする
3. RStudio Cloud のアカウントを作成してサインアップするか、Google または GitHub のアカウントでログインする

D.2.1 Create Project

RStudio Cloud ではプロジェクトという単位で分析を管理しますので、最初にプロジェクトを作成します。作成手順については RStudio Cloud メニューにある [Guide] で確認してください。ガイドは全て英語ですが、Chrome 系のブラウザであれば「Google 翻訳」機能拡張を用いれば日本語に翻訳表示できます。

プロジェクトを作成すると下図のような統合開発環境の RStudio が表示されます。RStudio 自体の説明は Appendix を参照してください。

⁵<https://rstudio.cloud/>

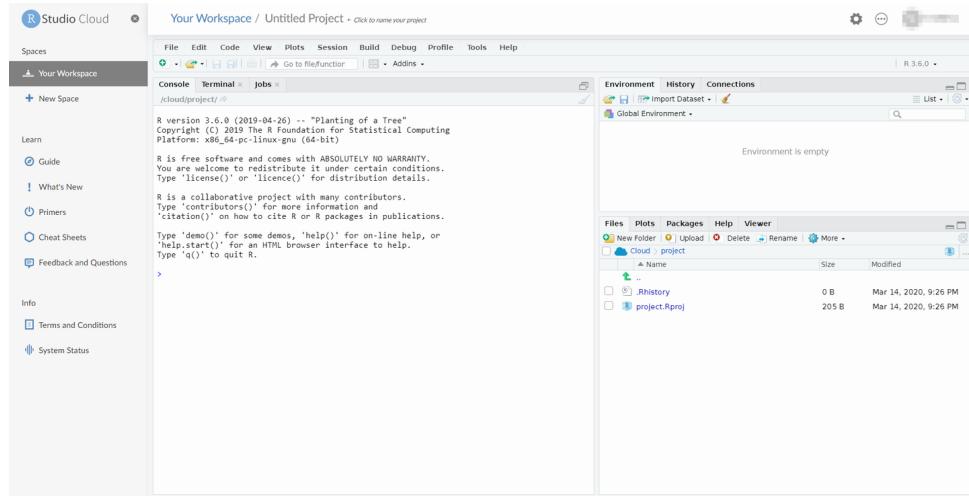


Figure D.6: Initial View

D.2.2 Install Packages

RStudio Cloud の初期状態では R のパッケージは Base R しかインストールされていません。最も利用する `tidyverse` パッケージと `rmarkdown` パッケージをインストールするために右下のエリアにある Packages タブをクリックしてパッケージマネージャを表示させます。

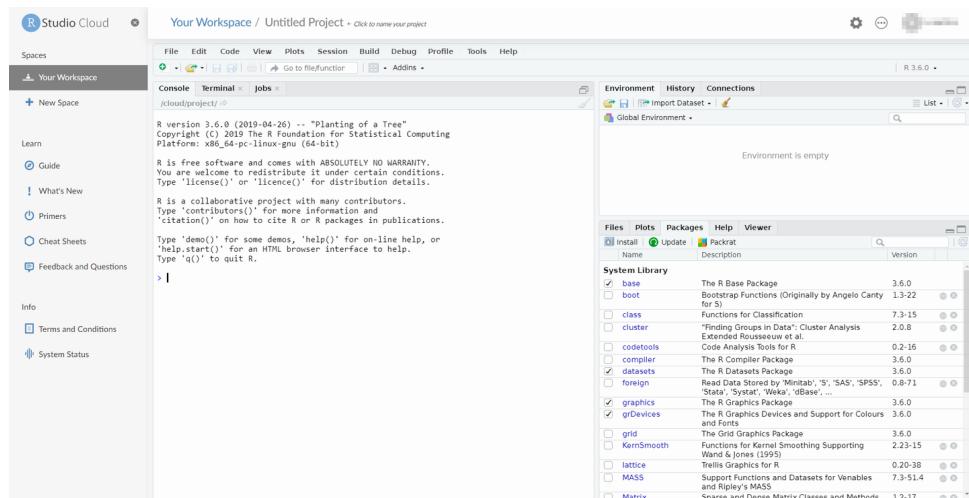


Figure D.7: Packages Manager

次にパッケージマネージャの上部に表示されている `install` ボタンをクリックし表示されたダイアログに `tidyverse`, `rmarkdown` と入力し `[install]` ボタンをクリックしてインストールします。

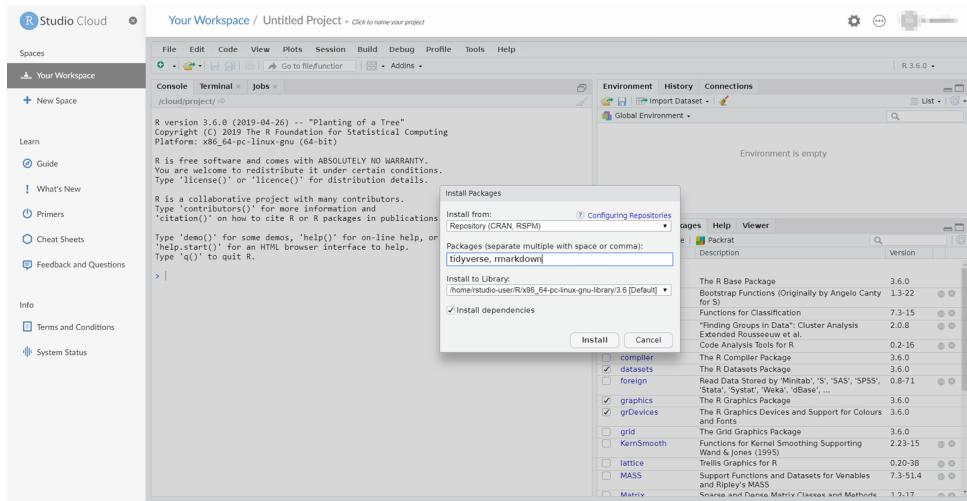


Figure D.8: Install Dialog

以上で、RStudio Cloud の準備は完了です。

Appendix E

Install R/RStudio

Rについて学ぶ前にRが使えるように環境を整えます。本書はR, RStudio, tidyverse/¹ パッケージならびにその他必要なパッケージの利用を前提としています。

RならびにRStudioはマルチプラットフォーム対応（マルチOS対応）ですのでWindows, macOS, Linuxのどのプラットフォームを選択しても構いません。ただし、64bitプラットフォームであることが条件です。なお、日本語版WindowsではWindowsが利用する文字コード（CP932, Shift JIS）に起因する不具合が散見されています。日本語版Windows環境を利用する場合はその点を認識の上で利用してください。

環境を整えるための手順は以下のようになります。

手順	実施内容	備考
1	Rのインストール	64bit プラットフォーム
2	Rtoolsのインストール	Winodwsのみ
3	RStudioのインストール	Desktop版
4	パッケージのインストール	tidyverse, rmarkdown
5	Gitのインストール	任意

Git²はVCS(Version Control System)と呼ばれるソースの版管理を行うシステムです。必要な場合のみインストールしてください。

¹<https://www.tidyverse.org/>

²<https://git-scm.com/>

E.1 Install R

R は CRAN (The Comprehensive R Archive Network)³ と呼ばれる公式リポジトリから入手してインストールします。CRAN には ミラーサイト⁴ も多数ありますので、利用しているインターネット環境に応じて近いサイトからダウンロードしてください。

よくある質問は FAQ(Frequently Asked Questions)⁵ にまとめられています。

E.1.1 Windows

Winodws では特段の理由がない限り CRAN⁶ から最新バージョンをインストールしてください。

旧バージョンをインストールしたい場合は Previous Releases of R for Windows⁷ から当該バージョンをダウンロードしインストールしてください。

日本語によるインストール方法が必要な場合は非公式ページですが R 初心者の館 (R と RStudio のインストール、初期設定、基本的な記法など)⁸ などのサイトを参考にしてください。

E.1.1.1 Rtools

Windows ではコンパイラなどの開発ツール類が標準装備されていませんので、R のパッケージをインストールする際に必要となる Rtools と呼ばれるツールキットをインストールしておきます。Building R for Windows⁹ のページからインストールした R のバージョン用の Rtools をダウンロードしてインストールしてください。なお、インストールの際はデフォルトオプションでインストールしてください。インストールディレクトリなどを変更すると正しく動かない場合があります。

E.1.2 macOS (OS X)

macOS ではインストールできるバージョンが限られていますので CRAN¹⁰ で確認の上でインストールしてください。

³<https://cran.r-project.org/>

⁴<https://cran.r-project.org/mirrors.html>

⁵<https://cran.r-project.org/doc/FAQ/R-FAQ.html>

⁶<https://cran.r-project.org/bin/windows/>

⁷<https://cran.r-project.org/bin/windows/base.old/>

⁸<https://das-kino.hatenablog.com/entry/2019/11/07/125044>

⁹<https://cran.r-project.org/bin/windows/Rtools/>

¹⁰<https://cran.r-project.org/bin/macosx/>

E.1.3 Linux

R がサポートしているディストリビューションは Debian, RedHat, Suse, Ubuntu のみです。Fedora を利用したい場合には README¹¹ を参照の上で RedHat Software のリポジトリからインストールしてください。

Linux の場合、ディストリビューションごと・バージョンごとにインストール方法が異なりますので各ディストリビューション用のディレクトリ内の README ファイルを参考にインストールしてください。

E.2 Install RStudio Desktop

R のインストールが完了したら統合開発環境（IDE）である RStudio Desktop をインストールします。Download ページ¹² から使用している環境（OS）用の RStudio をダウンロードしてインストールしてください。

E.2.1 動作確認

RStudio のインストールが完了したら RStudio を起動します。下図のようなウィンドウが立ち上がり左側の Console ペインに R のバージョンなどが表示されます。

Console ペインのプロンプト（> 表示）の部分に `2 * 3` と打ち込んで [Enter] キーを押し [1] 6 と表示されることを確認してください。

¹ `2 * 3`

[1] 6

E.3 Install R packages

次に必要となるいくつかのパッケージをインストールします。パッケージをインストールする場合はインターネットに接続されている必要があります。Console ペインのプロンプトに以下のコードを入力し [Enter] キーを押して実行します。

¹¹<https://cran.r-project.org/bin/linux/redhat/README>

¹²<https://rstudio.com/products/rstudio/download/#download>

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

インストールが終わったらパッケージが正しくインストールされていることを確認するために **Console** ペインに以下のコードを入力して実行します。

```
1 library(tidyverse)
```

以下のようなメッセージが表示されることを確認します。インストール時期によってはバージョン表記などが下記と異なる場合があります。なお、日本語版 Windows 環境では一部の文字が化けします。

```
1 Loading required package: tidyverse
2 ─ Attaching packages ─────────────────────────────────── tidyverse 1.3
3   ┌─ ggplot2 3.2.1 ┘ ┌─ purrr  0.3.3
4   ┌─ tibble  2.1.3 ┘ ┌─ dplyr   0.8.3
5   ┌─ tidyverse 1.0.0 ┘ ┌─ stringr 1.4.0
6   ┌─ readr   1.3.1 ┘ ┌─forcats 0.4.0
7 ─ Conflicts ─────────────────────────────────── tidyverse_conflicts
8   └─ dplyr::filter() masks stats::filter()
9   └─ dplyr::lag()   masks stats::lag()
```

続いて rmarkdown¹³ パッケージをインストールします。tidyverse パッケージのときと同様に以下のコードを **Console** ペインに入力して実行します。

```
1 install.packages("rmarkdown")
```

E.3.1 Linux 環境の場合

Linux 環境ではプラットフォーム側のライブラリなどが足りずにパッケージのインストールが完了できない場合があります。その場合は RStudio Package Manager, demo site¹⁴ にてインストールしたいパッケージが必要とするライブラリなどを確認、インストールしてから再度パッケージをインストールしてください。

例えば Ubuntu 18.04LTS で R に tidyverse パッケージをインストールする場合には以下のようなライブラリなどが OS 側にインストールされている必要があります。

¹³<https://rmarkdown.rstudio.com/>

¹⁴<https://demo.rstudiopm.com/client/#/>

```
1 apt-get install -y libicu-dev
2 apt-get install -y make
3 apt-get install -y libcurl4-openssl-dev
4 apt-get install -y libssl-dev
5 apt-get install -y pandoc
6 apt-get install -y libxml2-dev
```

E.4 Install Git

RStudio にはソースコードの版管理を行うインターフェースが標準で用意されていますが、版管理システム（以降、VCS）を別途インストールする必要があります。RStudio で利用できる VCS は以下の二つです。

- Git ¹⁵
- Subversion(SVN) ¹⁶

どちらを利用しても構いませんが GitHub ¹⁷ などのクラウドサービスが充実している Git の利用をおすすめします。

E.4.1 Git

Windows および macOS は Git の ダウンロードページ ¹⁸ から最新バージョンをダウンロードしてインストールします。Linux はリポジトリからインストールするか ダウンロードページ ¹⁹ から最新バージョンをダウンロードしてインストールしてください。

E.4.2 Git Client

RStudio には簡易的な Git のクライアント機能が標準で用意されていますが、きめ細かな操作を行いたい場合には Git の GUI クライアントをインストールしてください。

¹⁵<https://git-scm.com/>

¹⁶<https://subversion.apache.org/>

¹⁷<https://github.com/>

¹⁸<https://git-scm.com/downloads>

¹⁹<https://git-scm.com/downloads>

い。代表的な Git Client を以下に列挙しておきます。

Git GUI Client	Ubuntu	Mac	Windows	Memo
GitKraken ²⁰	Yes	Yes	Yes	Free 版は機能制限あり
SmartGit ²¹	Yes	Yes	Yes	Free 版でも機能制限なし ¹
GitEye ²²	Yes	Yes	Yes	
Sourcetree ²³	No	Yes	Yes	日本語版あり
GitHub Desktop ²⁴	No	Yes	Yes	

¹ : 非商用利用の場合

²⁰<https://www.gitkraken.com/>

²¹<https://www.syntevo.com/smartgit/>

²²<https://www.collab.net/downloads/giteye>

²³<https://www.sourcetreeapp.com/>

²⁴<https://desktop.github.com/>

Appendix F

RStudio Server

R/Rstudio Desktop は前述のようにマルチプラットフォーム対応ですがプラットフォームごとに以下のような制約があります。

- 日本語版 Windows 環境では文字コード (CP932, Shift JIS) が原因で日本語を正しく処理できない事例が散見される
- 18.04LTS より前の Ubuntu 環境では RStudio Desktop で日本語入力ができない
 - * 有志による日本語入力パッチ（非公式パッチ）はあり
- macOS 環境ではグラフの日本語が文字化けする * いわゆる豆腐文字問題

特に日本語版 Windows 環境での問題は Windows が利用している文字コード (CP932, Shift JIS) に起因しているため問題は根本的な解決を期待できません。詳細については伝説とも言われている「Why are you using SJIS?」というキーワードで検索してみてください。

日本語版 Windows 環境における文字コード問題を回避するためには、RStudio Server¹ を利用する方法が考えられます。 RStudio Server は Linux 環境で動作する Web サーバベースの IDE ですが、Docker² のコンテナ技術を利用することで Windows や macOS 環境で動作させることができます。

OS	Docker app	System Requirements
macOS	Docker Desktop for Mac	refer docker docs ³
Windows	Docker Desktop for Windows	Hyper-V(Windows10 64bit Pro or Higher) or WSL2 ¹

¹ WSL2 は Windows10 version 2004 から利用可能になる予定です

¹<https://rstudio.com/products/rstudio/download-server/>

²<https://www.docker.com/>

F.1 Setup RStudio Sever with Docker

Windows または macOS 環境で Docker を利用し RStudio Server を起動するためには以下の手順が必要です。

手順	実施内容	備考
1	Hyper-V の有効化	Windows のみ
2	Docker Desktop のインストール	
3	Docker Image のダウンロード	
4	Docker Container の起動	

なお、Linux 環境での手順は割愛します。

F.1.1 Enable Hyper-V (Windows Only)

Windows 環境ではインストールする前に Hyper-V を有効にする⁴ 必要があります。

F.1.2 Download and Install Docker Desktop

利用している環境に応じた Docker Desktop⁵ をダウンロードしてインストールします。なお、ダウンロードには docker hub⁶ でアカウント登録が必要です。

詳細な手順や設定方法は Docker docs⁷ を参照してください。

F.1.3 Download Docker Image

Docker Desktop をインストール・起動しましたら RStudio Server の Docker Image をダウンロードします。様々な方が RStudio Server の Docker Image を公開されていますが代表的な Docker Image には次のようなものがあります。

Image	Description
rocker/tidyverse ⁸	Version-stable base R and RStudio, tidyverse, devtools
rocker/verse ⁹	Adds TeX and related packages to rocker/tidyverse
ykunisato/paper-r-jp ¹⁰	Dockerfile of writing paper by R Markdown

⁴<https://docs.microsoft.com/ja-jp/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v>

⁵<https://www.docker.com/products/docker-desktop>

⁶<https://hub.docker.com/>

⁷<https://docs.docker.com/get-docker/>

Image	Description
kmetrics/jverse ¹¹	Japanized rocker/verse

rocker¹² は準公式とも言えるような R に関連する Docker Image を継続的に提供しているプロジェクトです。様々なイメージを提供していますが残念ながら日本語フォントの追加などの日本語対応がなされていません。グラフで日本語を利用しない限りは rocker のイメージを利用して何ら問題はありません（ソースなどの表示はブラウザに依存しているのでコードに日本語を記述することが可能です）。

グラフで日本語を利用したい場合は著者が rocker/verse に日本語フォントなどを追加して日本語対応させた kmetrics/jverse¹³ を利用するか rocker が公開している Dockerfile を改修して日本語対応させたイメージを利用してください。

利用する Docker Image を決めたらコンソール（コマンドプロンプト）で以下のコマンドを実行してイメージをダウンロードしてください。

```
1 docker pull rocker/tidyverse
```

F.1.4 Run Container

⁸<https://hub.docker.com/r/rocker/tidyverse>

⁹<https://hub.docker.com/r/rocker/verse>

¹⁰<https://hub.docker.com/r/ykunisato/paper-r-jp>

¹¹<https://hub.docker.com/r/kmetrics/jverse>

¹²<https://github.com/rocker-org/rocker>

¹³<https://hub.docker.com/r/kmetrics/jverse>

Appendix G

RStudio IDE

データ分析勉強会では長らく R Commander (以降、Rcmdr)¹ が利用されています。勉強会の母体となっている SQiP 研究会² のソフトウェアメトリクスに関する演習コースでも同様です。これはプログラミングに縁の薄いソフトウェア品質管理技術者が短期間で R を用いた分析を行えるようにとの配慮からです。実際、Rcmdr はコードを記述しなくともデータの可視化や分析ができますのでデータ分析の初学者にとっては R の恩恵を簡単に受けられる非常に便利な道具です。

しかし、Rcmdr は R のごく一部の関数を GUI で使えるようにしたラッパープログラムですので、できることが非常に限られています。加えて GUI 操作なため操作自体が記録に残りません。つまり、探索的にデータを分析を行ってもその手順分析者の記憶に依存してしまいますので分析再現性の観点から見ると好ましい分析環境とは言えません。

本格的な探索的データ分析を行うには、出来ることが限られる Rcmdr ではなく R のスクリプトを用いるべきです。しかし、R 本体 (R Console) は非常に機能が限られていますので、それだけで探索的データ分析を行うのは非常に困難です。そこで、初学者には様々な機能を予め備えている統合開発環境 (IDE - Integrated Development Environment) を利用をおすすめします。

R 用統合開発環境のデファクトスタンダードと言えるのが RStudio, PBC の RStudio IDE (以降、RStudio)³ です。無償版である Open Source Edition でも全ての基本的な機能を利用できます。

初学者にとって RStudio には以下のような便利な機能があります。

- 補完機能が強力
 - 関数名・変数名・パッケージ名などを補完してくれますので入力負荷が大幅に減ります

¹<https://www.rcommander.com/>

²<https://www.juse.or.jp/sqip/workshop/outline/index.html>

³<https://rstudio.com/products/rstudio/>

- エディタ機能が強力
 - キーひとつでヘルプの参照が可能ですので即座に疑問が解決できます
 - 部分的にコードを実行できますので手順を確認しながらコーディングできます
 - Markdown 記述が使えますので分析と報告書作成を同時に進められます
 - * コードの直下に実行結果を表示することができますのでコードと実行結果の関係性が一目でわかります
- パッケージ管理が分かりやすい
 - インストールされているパッケージが一目でわかります
 - パッケージの検索・読み込み・インストールが GUI 操作で簡単にできます
- その他の便利な機能
 - 作成した変数を一覧で確認できると共に値も確認できます
 - プロジェクト管理機能が使えますので分析ごとにファイルなどをセパレートできます
 - バージョンコントロールシステムを用いた履歴管理ができます
 - Python などの他言語もサポートしています

上記は機能のほんの一部を紹介したにすぎません。RStudio は R を利用した探索的データ分析を効率的かつ強力にしかも無償でサポートしてくれる道具です。

G.1 Overview

RStudio を起動すると以下のようないい画面が表示されます。画面は大きく以下の四つのエリアに分割されており、左上の A のエリアはソースエディタが表示されるエリアなので初めて起動した際には表示されません。

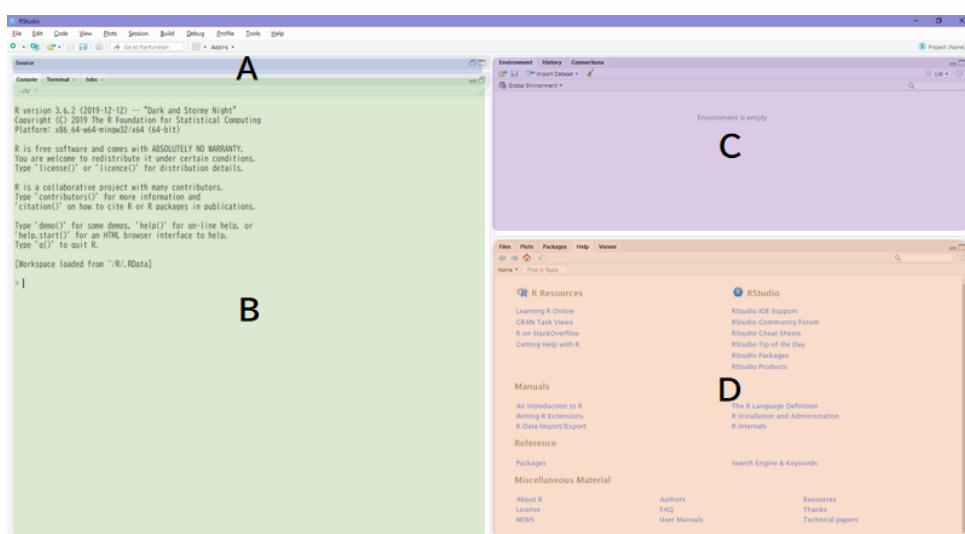


Figure G.1: RStudio Desktop, Windows

各エリアのサイズ（ウィンドウ内での比率）は任意に調整できますが、横幅に関しては A と B、C と D が常に同サイズとなります。各エリアにはペインと呼ばれるタブ切り替え型のサブエリアが表示されます。ペインは常時表示されるペイン（下図の黒文字）と機能が呼び出されたり利用を設定している場合にのみ表示されるペイン（下図の灰文字）があります。

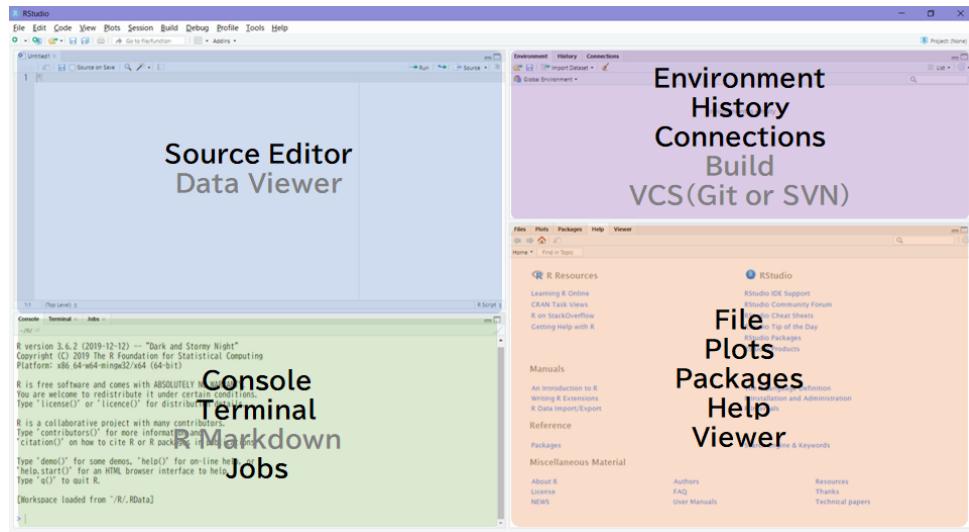


Figure G.2: RStudio Pane Layout, Windows

RStudio のバージョンにより多少ペイン構成が異なりますが以下のペインが用意されています。これらのペインはグローバルオプションで表示位置の変更や表示・非表示の切り替えができます。

No	Area	Pane name	Descriptions
1	A	(File name)	ソースエディタ（ファイルが開かれていない場合は未表示）
2	A	(Data name)	データフレーム型の変数などを表示するデータビューア
3	B	Console	文字通り R のコンソール（実行結果の表示だけでなくここから実行する機能）
4	B	Terminal	OS のターミナル（RStudio v1.1 から）
5	B	R Markdown	R Markdown ファイルをレンダリングした際にレンダリング情報を表示する
6	B	Jobs	ローカルジョブの実行マネージャ（RStudio v1.2 から）
7	C	Environment	オブジェクト（変数、関数）の表示と参照ができる環境マネージャ
8	C	History	実行履歴マネージャ（コンソールでの実行、ソースからの実行共に記録）
9	C	Connections	データソース接続マネージャ（RStudio v1.1 から）
10	C	Build	ビルドツール（プロジェクトオプションで有効にしている場合のみ）
11	C	Git or SVN	簡易 VCS クライアント（プロジェクトオプションで VCS を有効にして）
12	D	Files	簡易なファイルマネージャ
13	D	Plots	グラフィック専用プロットエリア（ヒストリ機能、出力機能付き）
14	D	Packages	パッケージ管理を行うためのパッケージマネージャ
15	D	Help	ヘルプビューア（ソースエディタやコンソールと連動したヘルプ表示が可能）
16	D	Viewer	HTML 等の表示が可能なビューア

G.2 Keyboard Shortcuts

キーボードショートカットは効率的なコーディングに役立ちますので、最低限、以下のショートカットを覚えましょう。

Keyboard Shortcuts	Description
[TAB]	入力中のコード（オブジェクト）を補完
[Alt/Option] + [-]	代入演算子 (<-) をカーソル位置に挿入する
[Ctrl/Cmd] + [Shift] + [M]	パイプ演算子 (%>%) をカーソル位置に挿入する
[Ctrl/Cmd] + [Shift] + [C]	選択行をコメント・アンコメントする（トグル動作）
[Ctrl/Cmd] + [Alt/Option] + [I]	カーソル位置にコードチャンクを挿入する（R Markdown のみ）
[Ctrl/Cmd] + [Enter]	選択したコードを実行する（行選択、部分選択どちらも可）
[Ctrl/Cmd] + [Shift] + [Enter]	コードチャンク内の全てのコードを実行する（R Markdown のみ）
[F1]	選択またはカーソル位置の関数のヘルプを呼び出す
[Ctrl/Cmd] + [F]	アクティブなペイン内の検索

上記以外のショートカットはメニュー [Tools] - [Keyboard Shortcuts Help] を選択すると表示できます。

G.3 Writing R code

では、実際に RStudio を利用して簡単なコードを書いてみましょう。初学者が学習のために R のコードを記述するには R Notebook 形式が便利です。R Notebook 形式はマークダウン言語とコードを混在できる R Markdown 形式を簡易にしたもので、コード以外に説明などを記述できるのでアウトプットしながらの学習が可能です。R Notebook 形式を使うにはメニューから [File] - [New File] - [R Notebook] を実行します。すると下図のようなソースエディタ（以降、エディタ）が開きます。

```
Untitled1 x
ABC Preview Insert Run
1 ----
2 title:"R-Notebook"
3 output:html_notebook
4 |----|
5 |
6 This is an [R·Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.
7 |
8 Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.
9 |
10 ``{r}
11 plot(cars)
12 ````|
13 |
14 Add a new chunk by clicking the *Insert·Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
15 |
16 When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).
17 |
18 The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.
19
```

Figure G.3: R Notebook file

この時点ではファイルとして保存されていませんので、メニューから [File] - [Save As...] を実行して適当な場所に適当な名前で保存しておきます。ここでは test という名前を入力して保存します。ファイルの拡張子が自動的に付与されますのでタブの表示は test.Rmd となります。

```
test.Rmd x Insert ▾
```

Figure G.4: R Notebook saved file

ファイルを保存したら 6 行目の「This is an ...」から 18 行目の「in the editor is displayed.」までを削除し、カーソルの位置（6 行目）でキーボードショートカット [Ctrl/Cmd] + [Alt/Option] + [I] を押下してコードを記述するためのブロックを挿入します。

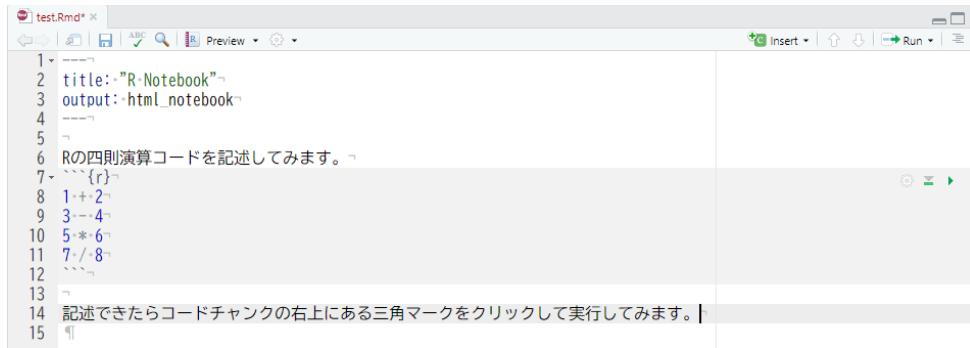


The screenshot shows the RStudio interface with an R Notebook file open. The title bar says "test.Rmd*". The left pane displays the Rmd code:

```
1 ----  
2 title:"R-Notebook"  
3 output:html_notebook  
4 ----  
5 ~  
6 ``{r}  
7 |  
8 ~  
9 ~  
10 ||
```

Figure G.5: R Notebook insert chunk

すると上図のように三連のバッククオート (``') で囲まれたブロックが挿入されます。このブロックはコードチャンクと呼ばれる R のコードを記述する部分です。コードチャンクの前後は自由な記述が出来ますので、以下のように入力してみてください。



A screenshot of the RStudio IDE showing an R Notebook file named 'test.Rmd'. The code editor pane contains the following R code:

```

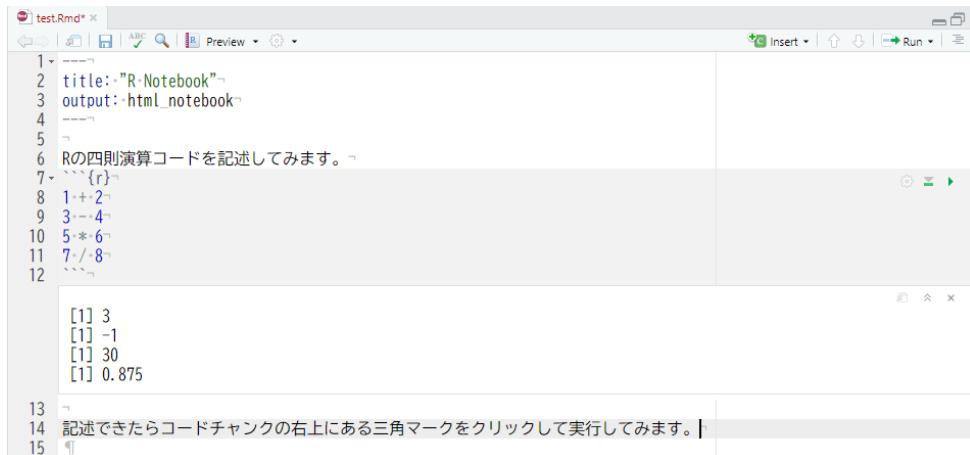
1 ----
2 title::"R-Notebook"
3 output::html_notebook
4 -----
5 
6 Rの四則演算コードを記述してみます。
7 ``{r}
8 1 + 2
9 3 - 4
10 5 * 6
11 7 / 8
12 ``-
13 
14 記述できたらコードチャunkの右上にある三角マークをクリックして実行してみます。
15

```

The interface includes standard RStudio menu bars (File, Edit, View, Insert, Run) and toolbars. A green triangle icon in the top right corner of the code chunk indicates it is ready to be run.

Figure G.6: R Notebook first code

上図のように R Notebook では説明とコードを混在することができます。では、コードチャunkの右上にある緑色の三角マークをクリックしてコードを実行してみましょう。



A screenshot of the RStudio IDE showing the same R Notebook file after running the code. The code editor pane now shows the following output:

```

1 ----
2 title::"R-Notebook"
3 output::html_notebook
4 -----
5 
6 Rの四則演算コードを記述してみます。
7 ``{r}
8 1 + 2
9 3 - 4
10 5 * 6
11 7 / 8
12 ``-
13 
14 [1] 3
15 [1] -1
16 [1] 30
17 [1] 0.875
18 
19 記述できたらコードチャunkの右上にある三角マークをクリックして実行してみます。
20

```

The output is displayed in the console pane below the code editor. The green triangle icon has turned grey, indicating the chunk has already been run.

Figure G.7: R Notebook run code

コードチャunkの下と Console ペインに実行結果が表示されます。コードチャunkの下に実行結果が表示されない場合は下図のように歯車アイコンをクリックし表示したメニューから Chunk Output Inline にチェックをつけ、再度、緑色の三角マークをクリックしてコードを実行してください。コードチャunkの下に実行結果が表示されます。

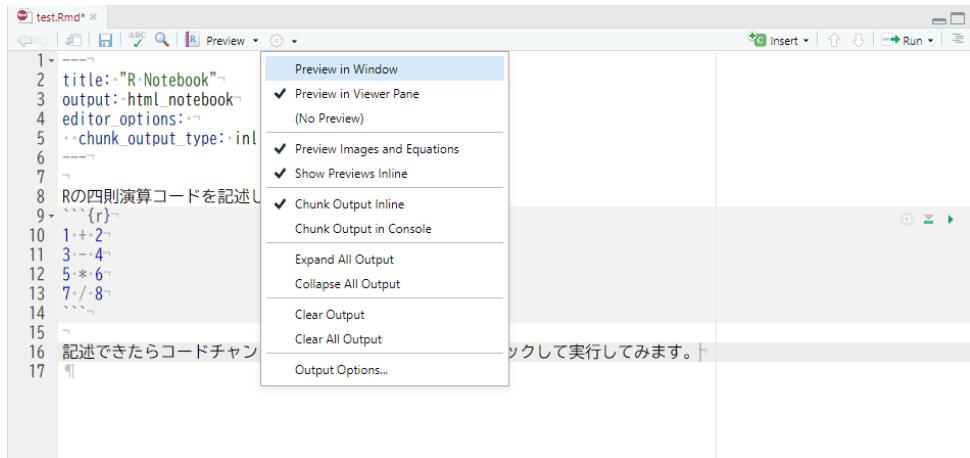


Figure G.8: R Notebook option

最後にフロッピーディスクアイコンをクリックするかキーボードショートカットの [Ctrl/Cmd] + [S] を押下してファイルを保存しておきます。

G.4 Global Options

メニュー [Tools] - [Global Options...] を選択すると表示できます。以降に推奨設定項目を記載しておきますので参考にしてください。記載されていないオプションは好みで設定してください。

G.4.1 General

General オプションは RStudio の全般的な動作に関する設定です。Basic と Advanced の二種類の設定がありますが、初学者の方は Basic のみ以下のように設定しておくと便利です。

大項目 (Tab)	中項目 (太文字)	設定項目	推奨設定
Basic	R Sessions	R version	Default (Windows)
Basic	R Sessions	Default working directory	任意のディレクトリ
Basic	R Sessions	Restore most recently opened project at startup	Unchecked
Basic	Workspace	Restore .RData into workspace at startup	Checked
Basic	Wrokspace	Save workspace to .RData on exit	“Ask” or “Always”
Basic	Ohter	Automatically notify me of updates to RStudio	Checked

特に “Default working directory” はプロジェクトを作成・管理するディレクトリに設定しておくと便利です。

G.4.2 Code

Code オプションはソースエディタの動作に関する設定です。ソースの記述はスタイルガイド (The tidyverse style guide)⁴ に準拠することをおすゝめしますので、設定例もスタイルガイドに沿ったものになっています。なお、Python などの他言語を併用する場合は適切な設定に変更してください。

大項目 (Tab)	中項目 (太文字)	設定項目	推奨設定
Editing	General	Insert spaces for tab	Checked
Editing	General	Tab width	2
Editing	General	Auto-detect code indentation	Checked
Editing	General	Insert matching parens/quotes	Checked
Editing	General	Auto-indent code after paste	Checked
Editing	General	Vertically align arguments in auto-indent	Checked
Editing	General	Surround selection on text insertion	“Quotes & Brackets”
Editing	Execution	Always save R scripts before sourcing	Checked
Editing	Execution	Ctrl+Enter executes	“Multi-line R statement”
Display	General	Highlight selected word	Checked
Display	General	Highlight selected line	Checked
Display	General	Show line numbers	Checked
Display	General	Show margin	Checked
Display	General	Margin column	80
Display	General	Show whitespace characters	Checked
Display	General	Show syntax highlighting in console input	Checked
Saving	General	Restore last cursor position when opening file	Checked
Saving	Serialization	Line ending conversion	“Posix (LF)”
Saving	Serialization	Default text encoding	“UTF-8”

G.4.3 Appearance

Appearance オプションは RStudio の見た目に関する設定です。フォント設定のみ日本語の固定ピッチフォントに変更し、その他はお好みでどうぞ。

大項目 (Tab)	中項目 (太文字)	設定項目	推奨設定
N/A	N/A	Editor font	任意の日本語等幅フォント

⁴<https://style.tidyverse.org/>

日本語等幅フォントは好みで構いませんが、無償ダウンロード可能な以下のフォントがおすすめです。

- BIZ UD ゴシック (macOS, Windows) - MORISAWA PASSPORT
- Source Han Code JP (Linux, macOS) - SIL Open Font License
- IPA ゴシック (Linux, macOS, Windows) - IPA フォントライセンス

なお、日本語版 Windows の RStudio では一部の日本語等幅フォントを正しく表示できないバグがあるようですので、フォントの選択には注意してください。

G.4.4 Pane Layout

Pane Layout オプションは前述のペインの表示場所や表示・非表示を変更するためのオプションですので、初学者はデフォルト設定のまま利用することをおすめします。

G.4.5 Packages

Packages オプションはパッケージマネジメントに関する設定です。Management と Development の二種類の設定がありますが、Development はパッケージ自体を開発するためのオプションですので Management のみ設定してください。

大項目

(Tab)

³ https://docs.deepl.com/doc/api-for-r/#/r/			推奨設定
Management	Package Management	Primary CRAN repository	任意の https サイト ¹
Management	Package Management	Enable packages pane	Checked
Management	Package Management	Use secure download method for HTTP	Checked
Management	Package Management	Use Internet Explorer library/proxy for HTTP	Checked ²

¹ ネットワーク的に最も速い（近い）サイトを選んでください ² プロキシサーバーを利用している場合に設定してください

G.4.6 R Markdown

R Markdown オプションは R Markdown に関する設定です。

大項目 (Tab)	中項目 (太文字)	設定項目	推奨設定
N/A	R Markdown	Show inline toolbar for R code chunk	Checked
N/A	R Markdown	Enable chunk background highlight	Checked
N/A	R Markdown	Show output preview in	“Viewer Pane”
N/A	R Markdown	Show output inline for all R Markdown documents	Checked
N/A	R Markdown	Show equation and image previews	“Inline” or “In a popup”
N/A	R Markdown	Evaluate chunks in directory	“Document”
N/A	R Notebooks	Execute setup chunk automatically in notebooks	Checked
N/A	R Notebooks	Hide console automatically when executing notebook chunks	Checked

G.4.7 Sweave

Sweave オプションは R + LaTeX によるドキュメント作成に関する設定です。Sweave を利用しない限り基本的に変更する必要はありませんが、R Markdown で PDF ファイルを作成する場合は PDF ビューアに関する設定のみお好みのビューアを設定してください。

大項目 (Tab)	中項目 (太文字)	設定項目	推奨設定
N/A	PDF Preview	Preview PDF after compile using	お好みのビューア

G.4.8 Spelling

Spelling オプションはスペルチェックのための設定です。UK または US の English を指定するのが無難です。

G.4.9 Git/SVN

Git/SVN オプションはバージョンコントロールシステム（VCS）に対する設定です。VCS を利用する場合のみ設定してください。

G.4.10 Publishing

Publishing オプションは RStudio, Inc. が提供しているサービスヘドキュメントを発行する場合に利用する設定ですので、当該のサービスを利用する場合のみ設定してください。

G.4.11 Terminal

Terminal オプションは OS のターミナルを RStudio の Terminal ペインから利用するための設定です。Terminal ペインを利用する場合のみ設定してください。

大項目 (Tab)	中項目 (太文字)	設定項目	推奨設定
N/A	Shell	New terminals open with	任意のシェル
N/A	Connection	Connect with WebSockets	Terminal が起動しない場合はチェック

G.5 Project Options

メニュー [Tools] - [Project Options...] を選択すると表示できます。Build Tools と Git/SVN を除いて基本的にグローバルオプションと同一の設定で構いません。

Build Tools オプションは R Markdown Website や Bookdown を利用する場合に以下のように設定するのをおすゝめします。

大項目	中項目 (太文字)	設定項目	推奨設定
Build Tools	N/A	Project build tools	“Website”
Build Tools	N/A	Preview book after building	Checked
Build Tools	N/A	Re-knit current preview when supporting files change	Checked

Git/SVN オプションは VCS を利用する場合に利用する VCS を選択してください。VCS がインストールされていない場合は有効にできません。

Appendix H

Cloud IDE

開発環境のクラウドサービス化も進んでいます。

H.1 RStudio Cloud GA

RStudio Colud¹ は RStudio, PBC が提供している RStudio Server によるクラウドサービスです。2020 年 2 月末時点では無料プランでも無制限のプロジェクトならびにプライベートなプロジェクトの作成が可能です。また、`learnr` パッケージを用いた初学者用のチュートリアルなど学習資料が多数用意されているのも特徴です。

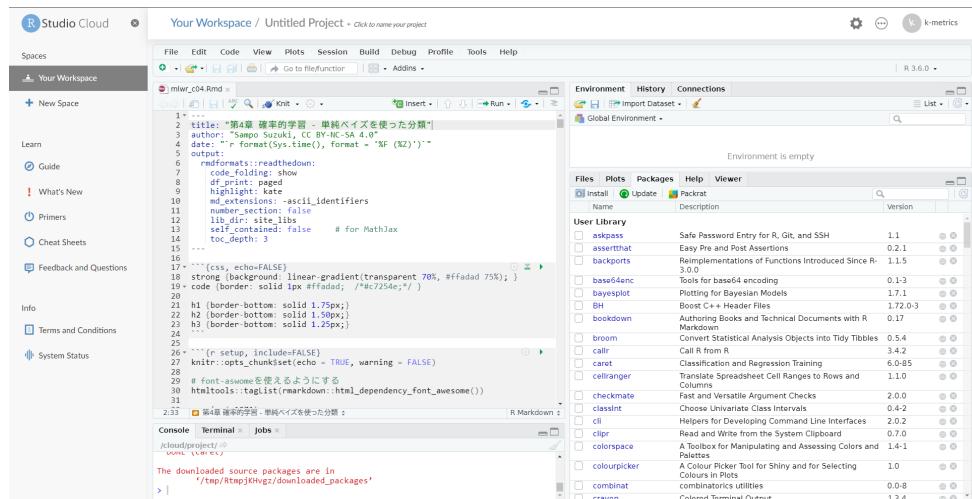


Figure H.1: RStudio Cloud, beta

ただし、無料プランで使えるリソースはメモリ 1GB・1CPU と限られていますので、ナイーブ・ベイズのようなメモリを必要とする機械学習プログラミングなどには

¹<https://rstudio.cloud/>

向いていません。なお、Google Colab のように 24 時間でインスタンスが消滅するというようなことは無いようです。

H.2 Exploratory

Exploratory² は BI (Business Intelligence) ツールのような操作で R を持った探索的データ分析 (EDA) が行える利用できる専用クライアントアプリケーションを用いるクラウドサービスです。無料で利用できますがオンライン限定・パブリックシェアオンリーとなりますので注意してください。

何ができるのか見てみる³ ページで多数の分析サンプルが公開されています。
また、使い方ガイド⁴ ページにも様々な説明資料が用意されています。

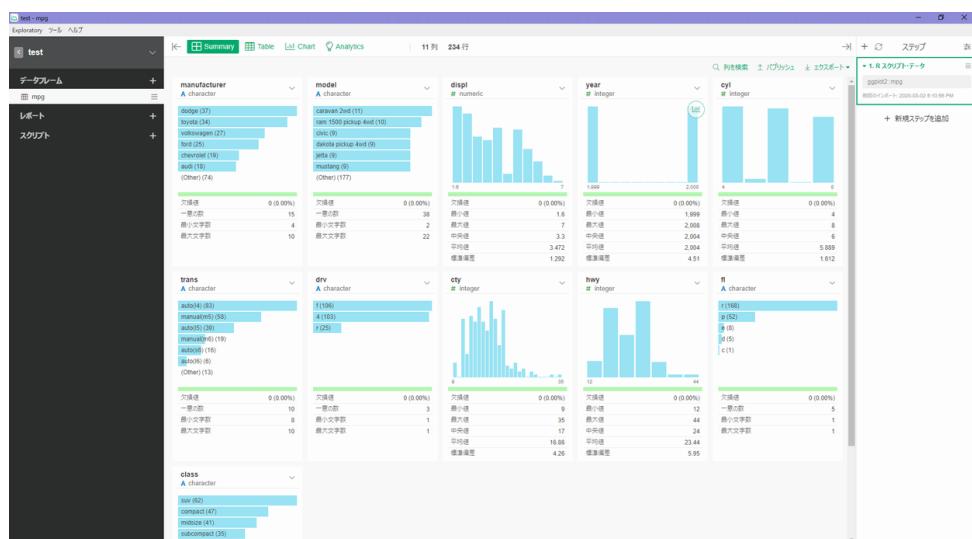


Figure H.2: Exploratory Public

価格⁵ ページからお好みのプランを選んでアカウントを取得します。クライアントアプリケーションは、mac または Windwos でしか動作しません。

²<https://exploratory.io/>

³<https://exploratory.io/insight?type=note&q=tag%3Avisualization%20tag%3Ahow-to%20tag%3A%22team%20exploratory%22&sort=top-viewed&language=ja>

⁴<https://exploratory.io/howto?language=ja>

⁵<https://exploratory.io/pricing>

H.3 binder

binder⁶ は実行環境の再現性を確保するためのクラウドサービスです。指定した Git のリポジトリから自動的に Jupyter Notebook のコード実行環境（Docker イメージ）を構築しクラウド上で実行することによりリポジトリにあるソースコードを動作させることができます。リポジトリに設定ファイルを置くことで RStudio Server や Shiny 環境を構築・実行することも可能です。

Google Colab や RStudio Cloud・Exploratoy などと異なりアカウントを取得する必要はありませんが、専用の環境を構築するわけではなく、あくまでも一時的な試用環境である点に注意してください。継続的に使える環境が必要な場合はローカルに環境を構築するか RStudio Cloud のようなクラウドサービスを利用してください。

⁶<https://mybinder.org/>

Bibliography

JJ Allaire, Yihui Xie, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, and Richard Iannone. *rmarkdown: Dynamic Documents for R*, 2022. URL <https://CRAN.R-project.org/package=rmarkdown>. R package version 2.12.

Stefan Milton Bache and Hadley Wickham. *magrittr: A Forward-Pipe Operator for R*, 2022. URL <https://CRAN.R-project.org/package=magrittr>. R package version 2.0.2.

Jennifer Bryan. *googlesheets4: Access Google Sheets using the Sheets API V4*, 2021. URL <https://CRAN.R-project.org/package=googlesheets4>. R package version 1.0.0.

Winston Chang, Joe Cheng, JJ Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, and Barbara Borges. *shiny: Web Application Framework for R*, 2021. URL <https://shiny.rstudio.com/>. R package version 1.7.1.

Lionel Henry and Hadley Wickham. *purrr: Functional Programming Tools*, 2020. URL <https://CRAN.R-project.org/package=purrr>. R package version 0.3.4.

Max Kuhn and Hadley Wickham. *Tidymodels: a collection of packages for modeling and machine learning using tidyverse principles.*, 2020. URL <https://www.tidymodels.org>.

Max Kuhn and Hadley Wickham. *tidymodels: Easily Install and Load the Tidymodels Packages*, 2021. URL <https://CRAN.R-project.org/package=tidymodels>. R package version 0.1.4.

Kirill Müller and Hadley Wickham. *tibble: Simple Data Frames*, 2021. URL <https://CRAN.R-project.org/package=tibble>. R package version 3.1.6.

Thomas Lin Pedersen. *patchwork: The Composer of Plots*, 2020. URL <https://CRAN.R-project.org/package=patchwork>. R package version 1.1.1.

R Special Interest Group on Databases (R-SIG-DB), Hadley Wickham, and Kirill Müller. *DBI: R Database Interface*, 2021. URL <https://CRAN.R-project.org/package=DBI>. R package version 1.1.2.

David Robinson, Alex Hayes, and Simon Couch. *broom: Convert Statistical Objects into Tidy Tibbles*, 2022. URL <https://CRAN.R-project.org/package=broom>. R package version 0.7.12.

RStudio, PBC. *RStartHere*, 2021. URL <https://github.com/rstudio/RStartHere>. A guide to some of the most useful R Packages that we know about.

Vitalie Spinu, Garrett Grolemund, and Hadley Wickham. *lubridate: Make Dealing with Dates a Little Easier*, 2021. URL <https://CRAN.R-project.org/package=lubridate>. R package version 1.8.0.

Ramnath Vaidyanathan, Yihui Xie, JJ Allaire, Joe Cheng, Carson Sievert, and Kenton Russell. *htmlwidgets: HTML Widgets for R*, 2021. URL <https://github.com/ramnathv/htmlwidgets>. R package version 1.5.4.

Hadley Wickham. Tidy data. *Journal of Statistical Software*, 59, 2014. URL <https://www.jstatsoft.org/article/view/v059i10>.

Hadley Wickham. *stringr: Simple, Consistent Wrappers for Common String Operations*, 2019. URL <https://CRAN.R-project.org/package=stringr>. R package version 1.4.0.

Hadley Wickham. *forcats: Tools for Working with Categorical Variables (Factors)*, 2021a. URL <https://CRAN.R-project.org/package=forcats>. R package version 0.5.1.

Hadley Wickham. *rvest: Easily Harvest (Scrape) Web Pages*, 2021b. URL <https://CRAN.R-project.org/package=rvest>. R package version 1.0.2.

Hadley Wickham and Jennifer Bryan. *readxl: Read Excel Files*, 2019. URL <https://CRAN.R-project.org/package=readxl>. R package version 1.3.1.

Hadley Wickham and Maximilian Girlich. *tidyverse: Tidy Messy Data*, 2022. URL <https://CRAN.R-project.org/package=tidyr>. R package version 1.2.0.

Hadley Wickham, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2021. URL <https://CRAN.R-project.org/package=ggplot2>. R package version 3.3.5.

Hadley Wickham, Romain François, Lionel Henry, and Kirill Müller. *dplyr: A Grammar of Data Manipulation*, 2022a. URL <https://CRAN.R-project.org/package=dplyr>. R package version 1.0.8.

Hadley Wickham, Jim Hester, and Jennifer Bryan. *readr: Read Rectangular Text Data*, 2022b. URL <https://CRAN.R-project.org/package=readr>. R package version 2.1.2.

Yihui Xie. knitr: A comprehensive tool for reproducible research in R. In Victoria Stodden, Friedrich Leisch, and Roger D. Peng, editors, *Implementing Reproducible Computational Research*. Chapman and Hall/CRC, 2014. URL <http://www.crcpress.com/product/isbn/9781466561595>. ISBN 978-1466561595.

Yihui Xie. *bookdown: Authoring Books and Technical Documents with R Markdown*. Chapman and Hall/CRC, Boca Raton, Florida, 2016. URL <https://bookdown.org/yihui/bookdown>. ISBN 978-1138700109.

Yihui Xie. *bookdown: Authoring Books and Technical Documents with R Markdown*, 2021a. URL <https://CRAN.R-project.org/package=bookdown>. R package version 0.24.

Yihui Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2021b. URL <https://yihui.org/knitr/>. R package version 1.37.

Achim Zeileis, Gabor Grothendieck, and Jeffrey A. Ryan. *zoo: S3 Infrastructure for Regular and Irregular Time Series (Z's Ordered Observations)*, 2021. URL <https://zoo.R-Forge.R-project.org/>. R package version 1.8-9.

平大平. 『統計解析のはなし』. 日科技連出版, 改訂版 edition, 2006. URL <https://www.juse-p.co.jp/products/view/196>. ISBN 978-4-8171-8028-5.

靖永田. 『統計的方法のしくみ』. 日科技連出版, first edition, 1996. URL <https://www.juse-p.co.jp/products/view/27>. ISBN 978-4-8171-0294-2.

野中誠, 小池利和, and 小室睦. 『データ指向のソフトウェア品質マネジメント』. 日科技連出版, 初版 edition, 2012. URL <https://www.juse-p.co.jp/products/view/442>. ISBN 978-4-8171-9447-3.