

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

2021-04-29

Contents

はじめに	9
想定読者	10
1 Introduction	11
1.1 How to analyze	12
Visualize/Transform	12
Model	13
Communicate	13
1.1.1 Import/Tidy	14
1.1.2 Program	14
1.2 Data Science Workflow	14
1.3 Tidyverse Eco System	15
I Program	17
2 Environments	19
3 R Basics	21
3.1 基本的な演算 (Basic Operations)	22
3.1.1 算術演算の基本	22

3.1.2	代入演算の基本	22
3.1.3	変数演算の基本	23
3.2	予約語 (Reserved word)	23
3.3	変数 (Variables)	24
3.3.1	データ型	24
3.3.2	変数型	25
3.3.3	ベクトル型	26
3.3.4	因子型	27
3.3.5	マトリクス型	28
3.3.6	アレイ型	28
3.3.7	データフレーム型	29
3.3.8	リスト型	30
3.4	定数 (Constant)	31
3.4.1	NA の型	32
3.5	参照・アクセス (Access Operators)	32
3.5.1	[演算子	32
3.5.2	\$ 演算子	36
3.6	検査・変換	37
3.7	演算子 (Operators)	39
3.7.1	算術演算子	40
3.7.2	比較演算子 Logical Operations	41
3.7.3	論理演算子 Boolean Operations	43
3.7.4	特殊演算子	43
3.7.5	演算子の優先順位	44
3.8	制御文	45
3.8.1	条件分岐	45
3.8.2	if, else	45

CONTENTS	5
3.8.3 switch	46
3.8.4 ifelse	48
3.8.5 繰り返し	48
3.8.6 for	50
3.8.7 while, repeat	51
 II Wrangle	 53
 4 Import	 55
4.1 readr	55
4.2 readxl	56
4.3 pdftools	56
 5 Tidy	 57
5.1 Tidy Data	57
5.2 longer	57
5.3 wider	57
 6 Transform	 59
6.1 filter	59
6.2 rename	59
6.3 select	59
6.3.1 select helpers	59
6.4 mutate	59
6.5 summarize	59

III Visualize	61
7 Base R	63
7.1 plot	63
7.2 boxplot	63
7.3 hist	63
8 ggplot2	65
IV Model/Infer	67
9 Test	69
10 Linear model	71
11 Machine Learning	73
V Communicate	75
12 R Markdown	77
VI Automate	79
13 shiny	81
VII Program	83
14 Environments 2	85
14.1 Google Colaboratory	85
14.1.1 Login Google	86

CONTENTS	7
14.1.2 Open Google Colab	86
14.1.3 Upload Template	86
14.1.4 Run R code	89
14.1.5 Save File	91
14.2 RStudio Cloud	91
14.2.1 Create Project	92
14.2.2 Install Packages	92
Appendix	94
A Install R/RStudio	95
A.1 Install R	96
A.1.1 Windows	96
A.1.2 macOS (OS X)	97
A.1.3 Linux	97
A.2 Install RStudio Desktop	97
A.2.1 動作確認	97
A.3 Install R packages	98
A.3.1 Linux 環境の場合	99
A.4 Install Git	99
A.4.1 Git	100
A.4.2 Git Client	100
B RStudio Server	101
B.1 Setup RStudio Sever with Docker	102
B.1.1 Enable Hyper-V (Windows Only)	102
B.1.2 Download and Install Docker Desktop	103
B.1.3 Download Docker Image	103
B.1.4 Run Container	104

C RStudio IDE	105
C.1 Overview	106
C.2 Keyboard Shortcuts	108
C.3 Writing R code	109
C.4 Global Options	112
C.4.1 General	112
C.4.2 Code	112
C.4.3 Appearance	113
C.4.4 Pane Layout	114
C.4.5 Packages	114
C.4.6 R Markdown	115
C.4.7 Sweave	116
C.4.8 Spelling	116
C.4.9 Git/SVN	116
C.4.10 Publishing	116
C.4.11 Terminal	117
C.5 Project Options	117
D Cloud IDE	119
D.1 RStudio Cloud GA	119
D.2 Exploratory	120
D.3 binder	120

はじめに

本書は クリエイティブ・コモンズ 表示 - 非営利 - 継承 4.0 国際 ライセンスの下に提供されています。

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。
- 継承 (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。



ソフトウェア開発において「データに基づく品質管理」が必要と言われるようになってから久しくなりますが、様々な理由でデータに基づく管理を実践している組織はまだ少数派ではないでしょうか？しかし、世の中の流れは「データドリブン」というキーワードに代表されるようにデータを使いこなせる組織が優位に立てる時代、数学が利益を生み出す数理資本主義の時代とされています。

『データ指向のソフトウェア品質マネジメント』は、日本のソフトウェア品質管理におけるデータ管理の必要性和データ分析に必要な知識を解説している数少ない書籍です。著者の一人である小池氏が主催しているデータ分析勉強会では、メトリクス分析に興味をもつ有志がこれらの知識を元に統計的コンピューティングによる品質管理を実践するための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うソフトウェア品質技術者をはじめとしRを用いてデータ分析を行いたい方々を想定読者としてRの基本的な知識を紹介しています。データ分析勉強会を通じて学んだ分析手法を実務で実践したい方の一助になれば幸いです。

想定読者

本書は基本的なコンピュータの知識と基本的な統計の知識を有している読者におすすめします。本書はRの実行環境構築に関する詳細な解説を行いませんので、実行環境の構築に不安がある方は次章で紹介している無料のクラウドサービスをご利用ください。

¥

Chapter 1

Introduction

データ分析を行うためには適切な分析ツールが必要不可欠です。R は統計計算に特化しているオープンソースの言語でデータ分析に最適なツールのひとつです。R がデータ分析に向いている理由をまとめているのが “Six Reasons To Learn R For Business”, R Blogger 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

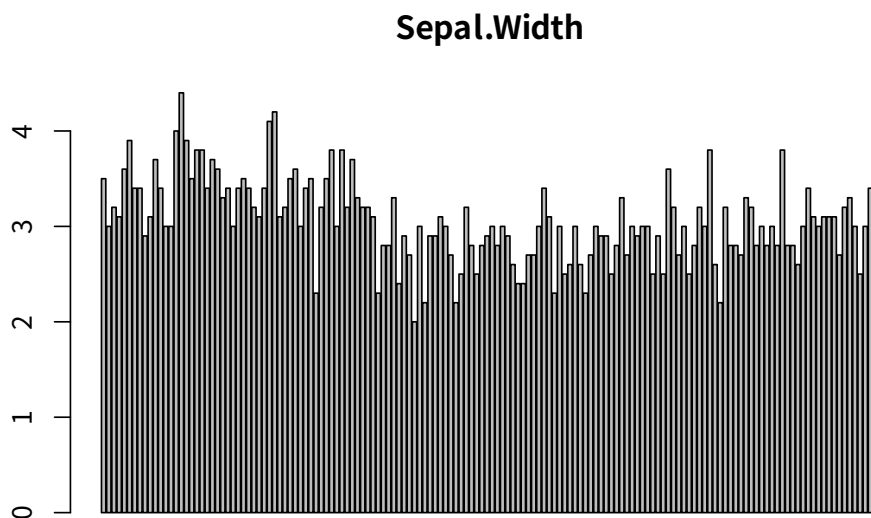
R はデータ分析に必要となるデータのハンドリングや可視化、モデリング、そして、レポーティングといった様々な機能を無料で利用することができます。また、R は逐次実行のインタプリタ型言語ですのでソフトウェアメトリクス分析のような探索的分析 (Exploratory data analysis) に適していると言えます。さらに、非常にフレンドリーかつ活発なコミュニティが日本でも形成されていますので、悩んだ時などに気軽に質問・相談ができるのも大きな強みです。

1.1 How to analyze

分析対象のデータがどのようなものかを知ることは、データ分析を行うにあたって重要なことです。どのような分布なのか、どのような値の範囲なのか、全てのデータが揃っているのかなど俯瞰して把握します。

Visualize/Transform

データを俯瞰するためには可視化（Visualize）が分かりやすい手法の一つです。例えば、計量値や計数値であれば棒グラフを用いて分布を確認したり、



ただ、棒グラフでは今ひとつ分布の状況が分かりませんので、度数を求めてヒストグラムを描いたり、五数要約を求めて箱ひげ図を描くことでよりデータの分布（状況）を把握することができるようになります。

```
with(iris, tapply(Sepal.Width, Species, fivenum))
```

```
## $setosa
## [1] 2.3 3.2 3.4 3.7 4.4
##
## $versicolor
## [1] 2.0 2.5 2.8 3.0 3.4
##
## $virginica
## [1] 2.2 2.8 3.0 3.2 3.8
```

```
with(iris, tapply(Sepal.Length, Species, fivenum))
```

```
## $setosa
## [1] 4.3 4.8 5.0 5.2 5.8
##
## $versicolor
## [1] 4.9 5.6 5.9 6.3 7.0
##
## $virginica
## [1] 4.9 6.2 6.5 6.9 7.9
```

```
with(iris, hist(Sepal.Width, plot = FALSE))$count
```

```
## [1] 4 7 13 23 36 24 18 10 9 3 2 1
```

つまり、データの把握（分布の把握）とは可視化（Visualize）と変換（Transform）の繰り返し作業と言えます。

Model

Communicate

1.1.1 Import/Tidy

1.1.2 Program

1.2 Data Science Workflow

データ分析の方法は様々ですが、そのプロセスは下図のように抽象化することができます。

この図は「Data Science Workflow」と呼ばれ、R コミュニティに多大な貢献をしている Hadley Wickham がその著書『R for Data Science』で提唱している概念図です。本書は、この Data Science Workflow に基づくページ構成になっており各プロセスのスコープ概略は下記の通りです。

Program

データ分析のすべてのプロセス (Tidy ~Communicate/Automate) で必要となるツールがプログラミングです。プログラミングを覚えることで効率的に分析処理を行えるようになります。

Import

分析対象となるデータを分析環境に取り込み分析をできるようにするのがインポートプロセスです。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インポートする必要があります。

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、一貫した形式 (Tidy data) に整理します。Tidy data はデータ分析において重要な概念です。

Transform

整理したデータ (Tidy data) がそのまま状態でデータ分析に使えることは稀です。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。

Tidy プロセスと合わせて **Wrangle** や **Data wrangling**、前処理と呼ばれることもあります。

Visualize

データを可視化することは様々な示唆を得ることと同義といえます。分析方針を考えるためにもデータがどういう傾向をもっているのかを把握するためのプロセスともいえます。

Model

可視化で得られた情報を元に数式可（モデル化）するのプロセスです。モデルは様々な

Communicate

分析結果を他人に伝えるためのプロセスです。結果を他人に伝えるだけでは不十分で 再現可能性（Reproducible research）が伴っていることも求められます。 3つの再現可能性

1.3 Tidyverse Eco System

このような Data Science Workflow を R で実現するための手段が Hadley Wickham により開発された tidyverse パッケージ群による Tidyverse Eco System です。tidyverse

Part I

Program

Chapter 2

Environments

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、本書では環境構築の必要がない binder を利用します。RStudio 環境を構築できる方はそちらを利用しても構いません。

Chapter 3

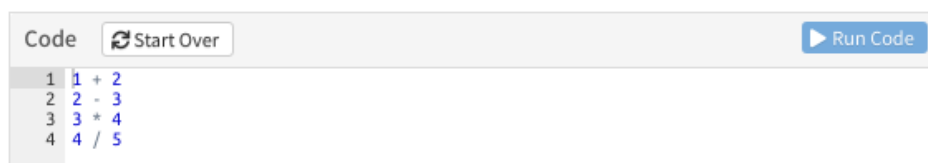
R Basics

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

と言われる R ですが、最たる特徴は統計処理に特化している点です。また、前出の Data Science Workflow 全体をカバーできるようになっており、データ構造の変換や特定の分析に対する数々のパッケージ、コマンド一つでグラフが描ける高度なグラフィクス、更には分析結果をレポートニングするための仕組みも用意されています。

データ分析をひとつの言語で一気通貫できるので覚えることは非常に多岐に渡りますが、初学者が R を学ぶ際には、まず、「ベクトル演算」を理解するのが一つのポイントだと考えます。

A screenshot of the Google Colab code editor interface. At the top, there is a 'Code' tab, a 'Start Over' button with a circular arrow icon, and a 'Run Code' button with a play icon. Below the toolbar, there is a code editor with four lines of R code:

```
1 1 + 2
2 2 - 3
3 3 * 4
4 4 / 5
```

Figure 3.1: Google Colab での実行例

なお、チュートリアルを起動後、何も操作をしないで放置しておくとも自動的にチュートリアルウィンドウがクローズします。クローズした場合は、再度、**Run Document** をクリックして再起動してください。

以降、見出しの括弧内の英語はチュートリアルの見出しと対応しています。

3.1 基本的な演算 (Basic Operations)

まずは最も基本となる演算を行ってみましょう。

3.1.1 算術演算の基本

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。

3.1.2 代入演算の基本

変数に演算結果を代入するには代入演算子 (`<-`) を用います。変数を使うための変数宣言は不要です。また、代入結果を確認 (表示) するためには変数名だけを記述して実行します。

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

3.1.3 変数演算の基本

定数だけでなく変数同士、変数と定数の演算も可能です。ここでは変数同士および変数と定数の四則演算を行ってみます。

さらに変数同士の演習結果を変数に代入してみます。代入前後で変数の値を確認して変数がどのように変化するかのかを合わせて確認します。

```
W
```

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

```
X
```

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

```
W <- W + X
```

```
W
```

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

```
X
```

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

```
##### Variables #####
```

このように R での変数の扱いは非常にシンプルですが、簡単に変数を上書きできてしまう点に留意してください。

3.2 予約語 (Reserved word)

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
```

3.3 変数 (Variables)

R での変数の扱い方について触れておきます。まず、変数名の命名規則ですがスタイルガイド に準拠して「英小文字、数字、アンダースコア」のみを使うことを推奨します。また、変数名は分かりやすい名称にするように心掛けてください。例えば

```
number_int <- 10L
number_int
```

```
## [1] 10
```

なお、数字から始まる変数名は R の仕様により使用することができません。

```
1num <- 100
2_string <- "foo"
```

```
## Error: <text>:1:2: 想定外のシンボルです
## 1: 1num
##      ^
```

また、予約語以外でも変数型や関数に使われているような名前を変数名にすることはおススメできません。

3.3.1 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（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ものがあります。

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	

型	クラス	タイプ	モード	格納モード	備考
複素数型	<code>complex</code>	<code>complex</code>	<code>complex</code>	<code>complex</code>	
論理型	<code>logical</code>	<code>logical</code>	<code>logical</code>	<code>logical</code>	Boolean 型
文字型	<code>character</code>	<code>character</code>	<code>character</code>	<code>character</code>	
日付型	<code>Date</code>	<code>double</code>	<code>numeric</code>	<code>double</code>	Date 型 (POSIX 型もあり)

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考えてください。書籍などでよく出てくる `str` (structure) 関数が返す型は上表におけるクラスです。

3.3.2 変数型

前述のようなデータ型を持った値を入れる箱が変数です。変数には華表のような様々な形があります。中でも表形式のデータを扱えるデータフレーム型は R ならではの変数型といえます。また、R には他の言語ではよく見かけるスカラー型（一種類のデータ型の値を一つだけ格納できる変数型）はありません基本となる変数型はベクトル型になります。

変数型にもクラスがありますが、ベクトル型変数のクラスはデータ型のクラスと同じです。

変数の型	クラス	説明
ベクトル型	データ型と同じ	一種類のデータ型の値を任意の個数だけ格納できる変数型
因子型	<code>factor, ordered</code>	水準インデックスを持ったベクトル型変数
マトリクス型 (行列型)	<code>matrix</code>	二次元型の変数型 (ベクトル型とは異なる型)
アレイ型 (配列型)	<code>array</code>	多次元型の変数 (ベクトル型とは異なる型)
データフレーム型	<code>data.frame</code>	データ型の異なる複数の等長ベクトル型を格納できる変数型
リスト型	<code>list</code>	データ型の異なる複数のベクトル型変数を格納できる変数型

3.3.3 ベクトル型

ベクトル型は前述のように最も基本となる変数型です。ベクトル型変数を作成するには `c` 関数を使います。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` 関数を省略できます。

```
x <- 2L                                # 一つだけ代入（格納）する場合 `c` は省略可能です
str(x)                                # 値が一つだけの場合は長さ表示が省略されます
```

```
## int 2
```

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

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

文字（型）を代入する場合はクォート（ダブルまたはシングル）で囲みます。

```
x <- c("1", "2", "3")                # 文字として数字を代入してみます
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？

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

```
## 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クラス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変換する処理です。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。

便利ではありますが意図しない結果を招く可能性がありますので、このような変換があることは覚えておいてください。

3.3.4 因子型

因子型は名義尺度や順序尺度の変数を扱うのに便利な変数型です。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) という属性がついているベクトル型です。

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。

```
x <- factor(c("A", "B", "AB", "O", "A", "A", "A", "B"))
str(x)
```

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

```
levels(x)
```

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

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

```
x <- ordered(c("A", "B", "AB", "O", "A", "A", "A", "B"))
str(x)
```

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

```
levels(x)
```

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

3.3.5 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。

展開方向を変えるには `byrow` オプションを指定します。

数値だけでなく文字列も扱えます。

3.3.6 アレイ型

アレイ型は多次元配列を扱うのに便利な変数型です。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, 列, 次元）にしたがって展開します。

展開は列方向のみで `matrix` 型にある `byrow` オプションはありません。また、第一引数のデータ数が足りない場合は先頭からリサイクルして展開します。

```
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
```

3.3.7 データフレーム型

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。列ごとに異なるデータ型の変数を格納することが可能ですので、データベースのテーブルのようなものです。ただし、全ての列において行数が同一でなければなりません。データフレーム型を作成するには `data.frame` 関数を用います。

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

```
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
```

データフレーム型は列名を指定することが可能です。また、文字型のデータはデフォルトで因子型変数として扱われます。

3.3.8 リスト型

リスト型はデータフレーム型とは異なり不等長のベクトル型変数を格納できる変数型です。ベクトル型の他にもマトリクス型やデータフレーム型、リスト型地震を格納できる非常に柔軟な構造であるため様々な関数あが返り値と使っています。

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

```
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
```

3.4 定数 (Constant)

変数はその名の通り値を変更できますが、R には特別な意味を持った値を保持するための定数があります。なお、定数にもクラスがあります。

定数	クラス
TRUE	logical
FALSE	logical
NULL	NULL
NA	logical
NaN	numeric
Inf	numeric

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。以下のような変換が行われますので注意してください。

```
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
```

3.4.1 NA の型

欠損値を示す NA には明示的にデータ型を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に NA を指定する必要がありますので覚えておいてください。

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

3.5 参照・アクセス (Access Operators)

変数の中の値を参照する方法は変数型により多少異なりますが、基本的には参照演算子またはアクセス演算子と呼ばれる [や \$ を用います。

3.5.1 [演算子

[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。

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

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



```
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。

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

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

```
x[1, ]
```

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

```
x[, 1]
```

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

```
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。

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

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

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

第 1 次元の 1 行目を参照します。

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

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

第 1 次元の 1 列目を参照します。

```
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。

```
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。

```
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。

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

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

\$ 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。

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

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

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ているblood, dataが\$演算子で参照可能です。

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

```
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。

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

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

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

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

3.6 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に強制型変換が行われますので、タイポ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
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</code> or <code>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>	

定数用の `is.` 関数群もあります。

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（有限値か否かを確認する <code>is.finite</code> 関数もあり）

3.7 演算子 (Operators)

R はベクトル演算が可能ですので演算子は基本的にベクトル演算に対応しています。

演算子は単項演算子と二項演算子に大別できます。二項演算子には算術演算子、比較演算子、論理演算子、特殊演算子があります。等長のベクトル型変数どうし、もしくは、ベクトル型変数とスカラ型変数の間の演算が可能です。不等長のベクトル変数に対して二項演算子を利用した場合、足りない値は先頭からリサイクルされて演算されます。条件によってはワーニングも表示されませんので不等長のベクトル演算は注意してください。

```
c(1:10) * c(1:2)
```

```
## [1] 1 4 3 8 5 12 7 16 9 20
```

```
c(1:10) * c(1:3)
```

```
## Warning in c(1:10) * c(1:3): 長いオブジェクトの長さが短いオブジェクトの長さ
## 数になっていません
```

```
## [1] 1 4 9 4 10 18 7 16 27 10
```

、### 単項演算子 単項演算子は文字通り単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。

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

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

```
-x
```

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

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

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

```
!x
```

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

3.7.1 算術演算子

算術演算子は基本中の基本とも言える四則演算子である加算、減算、乗算、除算、ならびに、べき算（べき乗算）、整数除算（商、剰余）の六つの演算子があります。

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

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

```
b
```

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

```
a + b      # 加算
```

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



```
a - b          # 減算
```

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

```
a * b          # 乗算
```

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

```
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
```

```
a ^ b          # べき乗算
```

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

```
a %/% b        # 整数除算 (商)
```

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

```
a %% b         # 整数除算 (剰余)
```

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

3.7.2 比較演算子 Logical Operations

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。

```
a
```

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

```
b
```

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

```
a < b
```

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

```
a > b
```

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

```
a <= b
```

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

```
a >= b
```

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

```
a == b
```

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

```
a != b
```

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

3.7.3 論理演算子 Boolean Operations

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

3.7.4 特殊演算子

特殊演算子は% 文字と% 文字で挟まれた演算子です。算術演算子の整数除算 (商、剰余) 厳密に言えば特殊演算子になりますが本資料では算術演算子として記載しています。

R では特殊演算子を用いて任意の演算を定義することができます。パッケージによっては特殊演算子を用意している場合もあります。

特殊演算子	演算内容
%*%	内積 (スカラー積)
%in%	マッチング

特殊演算子	演算内容
%O%	外積（ベクトル積）
%X%	クロネッカー積

3.7.5 演算子の優先順位

演算子の優先順位は下表の通りとなります。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。下記以外はヘルプで **Syntax** を検索すると確認できます。

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

3.8 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。

3.8.1 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）

3.8.2 `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。

```
x <- FALSE

if (x != TRUE) print("TRUE") else print("FALSE")

## [1] "TRUE"
```

```
if (x == TRUE) {  
  print("TRUE")  
} else {  
  print("FALSE")  
}
```

```
## [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。

```
x <- 10L  
y <- 5  
  
if ((x > 5) && (y < x)) {  
  print("match, (x > 5) && (y < x)")  
} else if ((x > 5) && (y >= x)) {  
  print("match, (x > 5) && (y >= x)")  
} else {  
  print("else")  
}
```

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

3.8.3 switch

分岐する条件の数が多い場合は if 文でなく switch* 文を利用するのが便利です。if 文と同じで評価式は TRUE か FALSE が単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。

3.8.3.1 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。

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

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

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

3.8.3.2 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。

```
x <- "2"
switch(x,
  "1" = "x is 1",      # 引数が"1"と一致する場合 (`if (x == "1")` に等価)
  "x is others",       # 一致するものがない場合 (`else` に等価)
  "2" = "x is 2")      # 引数が"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。

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

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

3.8.4 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。

```
ifelse(TRUE, 1, 0)
```

3.8.5 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](#) 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
for	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
while	条件式に与えた条件が成立している限り繰り返す
repeat	無限に繰り返すが繰り返し処理中の break 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。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

3.8.6 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。

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

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

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

3.8.7 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。

Part II

Wrangle

Chapter 4

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。チャンク以外にコメントなどを書き込むことができます。

4.1 readr

4.2 readxl

4.3 pdftools

Chapter 5

Tidy

5.1 Tidy Data

5.2 longer

5.3 wider

Chapter 6

Transform

6.1 filter

6.2 rename

6.3 select

6.3.1 select helpers

6.4 mutate

6.5 summarize

Part III

Visualize

Chapter 7

Base R

7.1 plot

7.2 boxplot

7.3 hist

Chapter 8

ggplot2

Part IV

Model/Infer

Chapter 9

Test

Chapter 10

Linear model

Chapter 11

Machine Learning

Part V

Communicate

Chapter 12

R Markdown

Part VI

Automate

Chapter 13

shiny

Part VII

Program

Chapter 14

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	要 Google アカウント
応用学習フェーズ	RStudio Cloud	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。

14.1 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は 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
5. アップロードしたファイルを Google ドライブに保存する

14.1.1 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すめします。

まず、ブラウザで Google 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。

14.1.2 Open Google Colab

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

画面テーマは右上の歯車ボタンから変更できます。

14.1.3 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。

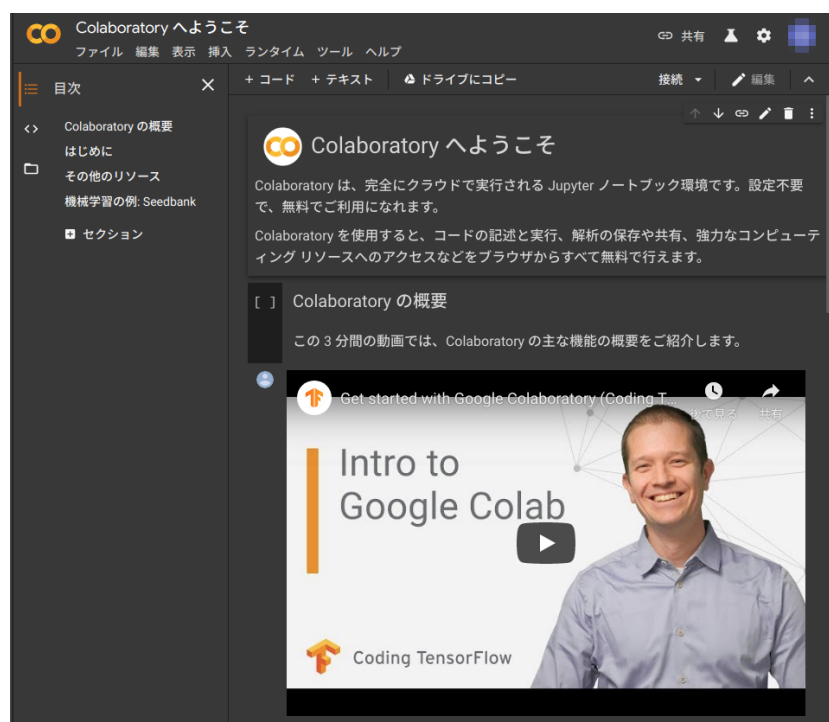


Figure 14.1: Google Colab, Theme: dark

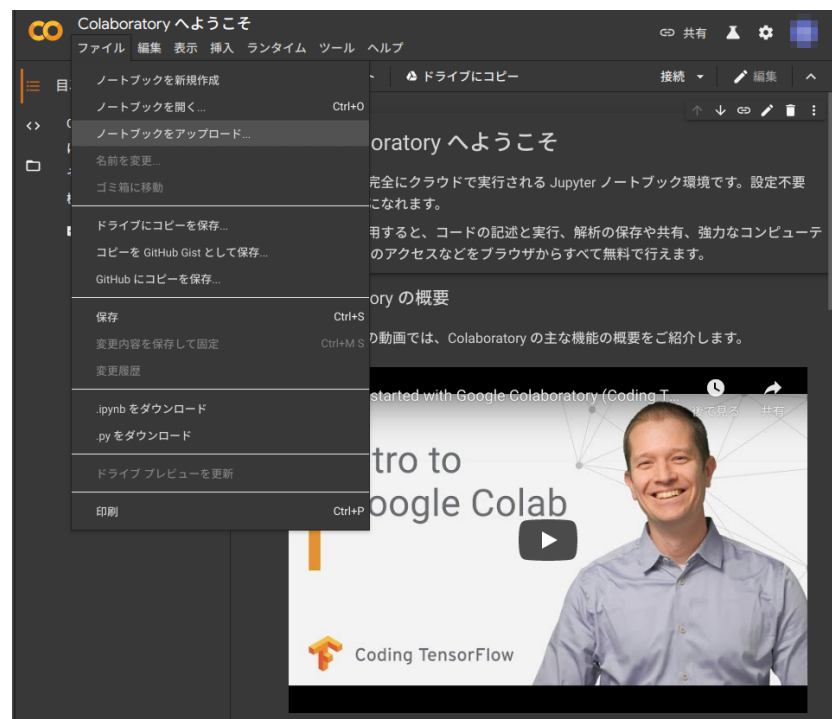


Figure 14.2: Upload notebook file

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/464ffb4d4d00e328560cd55966e7d4b8>

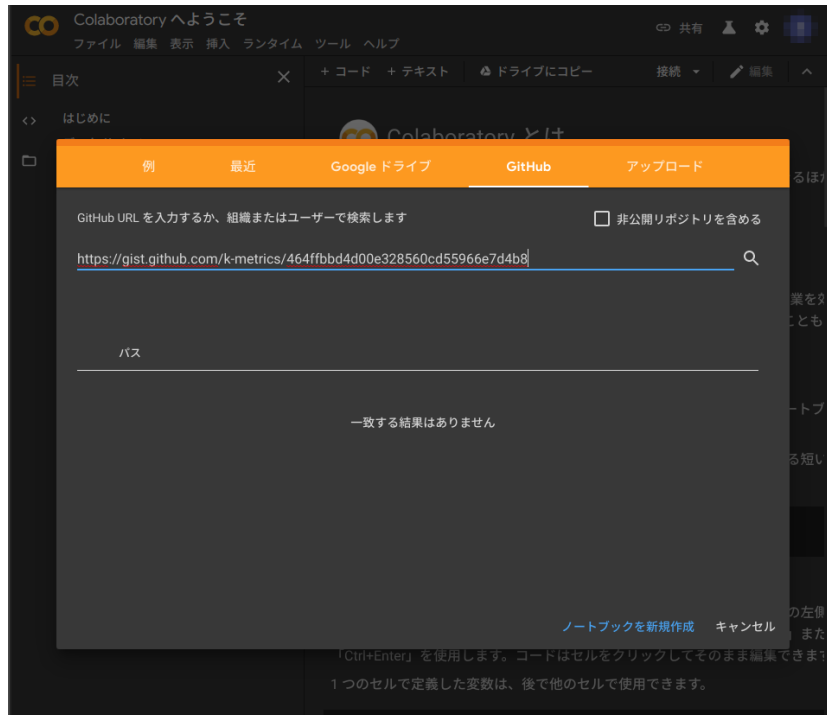


Figure 14.3: Upload from GitHub

テンプレートがアップロードされ表示されます。

14.1.4 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ありませんので [このまま実行] をクリックしてください。

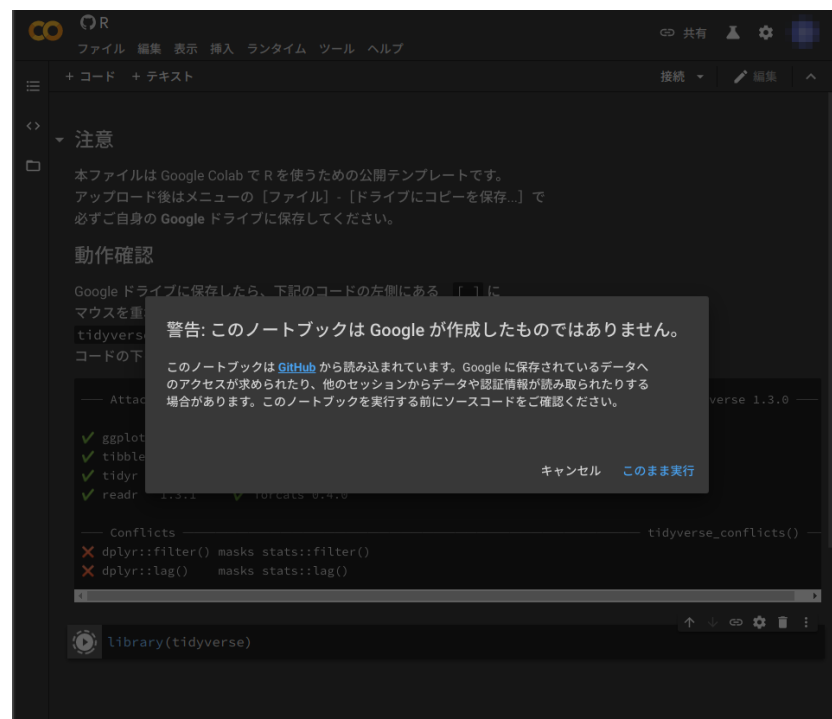


Figure 14.4: Warning dialog

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。

14.1.5 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。

14.2 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。

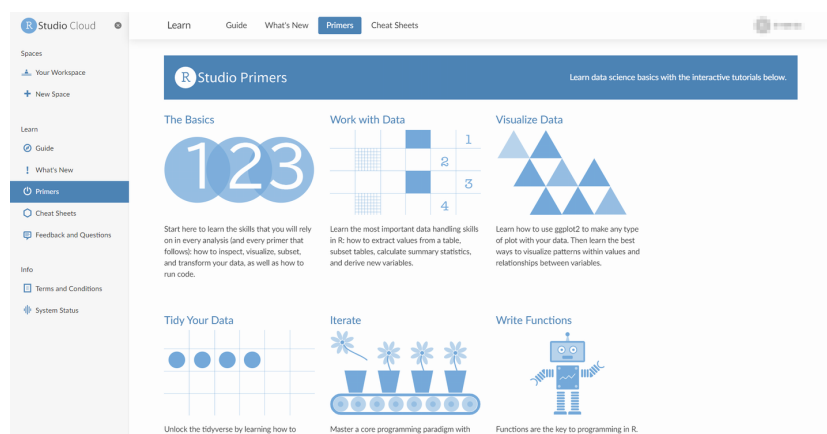


Figure 14.5: RStudio Cloud, beta

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 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

14.2.1 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。

14.2.2 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。

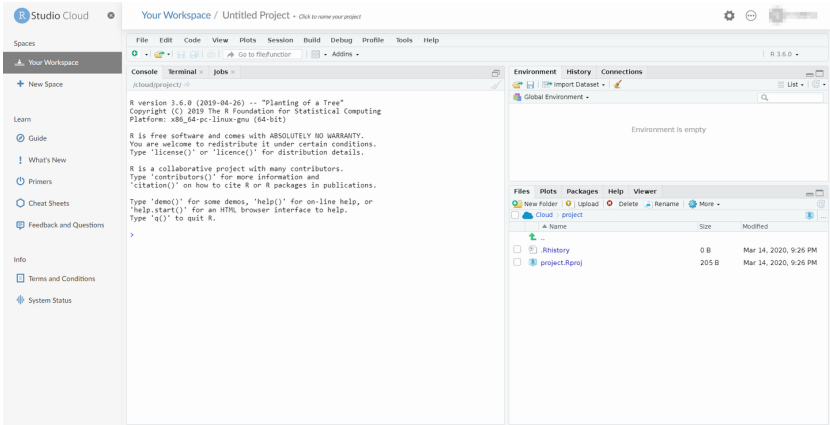


Figure 14.6: Initial View

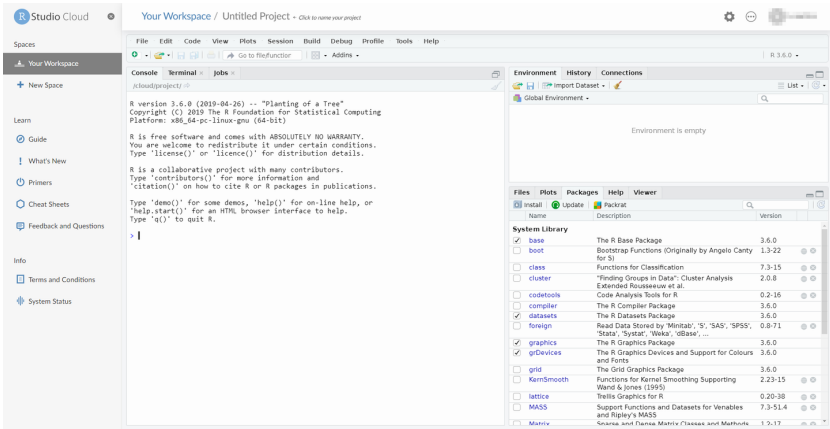


Figure 14.7: Packages Manager

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。

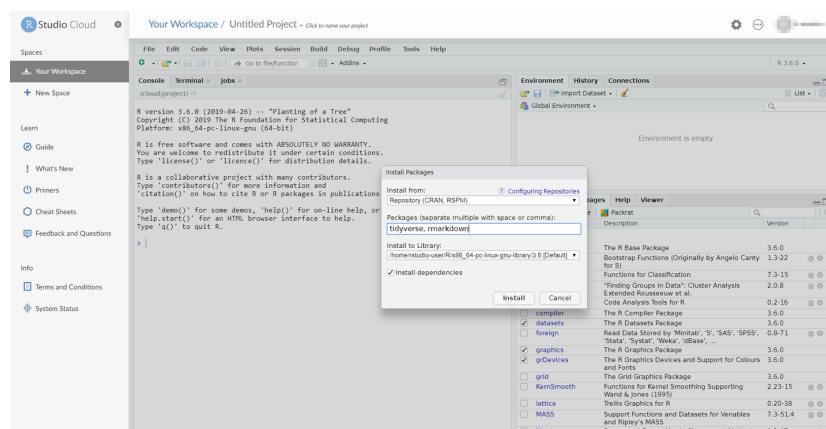


Figure 14.8: Install Dialog

以上で、RStudio Cloud の準備は完了です。

Appendix A

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/ 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 が利用してる文字コード（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	Windows 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

Git 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。

A.1 Install R

R は CRAN (The Comprehensive R Archive Network) 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 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) にまとめられています。

A.1.1 Windows

Windows では特段の理由がない限り CRAN 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 Windows 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) などのサイトを参考にしてください。

A.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 Windows 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。

A.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 CRAN で確認の上でインストールしてください。

A.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 README 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。

A.2 Install RStudio Desktop

R のインストールが完了しま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 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。

A.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。

```
2 * 3
```

```
[1] 6
```

A.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。

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

インストールが終わりまし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。

```
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。

```
Loading required package: tidyverse
- Attaching packages _____ tidyverse 1.3.0
☒ ggplot2 3.2.1      ☒ purrr 0.3.3
☒ tibble 2.1.3       ☒ dplyr 0.8.3
☒ tidyr 1.0.0        ☒ stringr 1.4.0
☒ readr 1.3.1        ☒ forcats 0.4.0
- Conflicts _____ tidyverse_conflicts()
☒ dplyr::filter() masks stats::filter()
☒ dplyr::lag()     masks stats::lag()
```

続いて `rmarkdown` 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。

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

A.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 site 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。

```
apt-get install -y libicu-dev
apt-get install -y make
apt-get install -y libcurl4-openssl-dev
apt-get install -y libssl-dev
apt-get install -y pandoc
apt-get install -y libxml2-dev
```

A.4 Install Git

RStudio にはソースコードの版管理を行うインタ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
- Subversion(SVN)

どちらを利用しても構いませんが GitHub 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。

A.4.1 Git

Windows および macOS は Git のダウンロードページ 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 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。

A.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。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	Yes	Yes	Yes	Free 版は機能制限あり
SmartGit	Yes	Yes	Yes	Free 版でも機能制限なし ¹
GitEye	Yes	Yes	Yes	
Sourcetree	No	Yes	Yes	日本語版あり
GitHub Desktop	No	Yes	Yes	

¹: 非商用利用の場合

Appendix B

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 Server 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、Docker 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可能です。

OS	Docker app	System Requirements
macOS	Docker Desktop for Mac	refer docker docs
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

B.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。

B.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 必要があります。

B.1.2 Download and Install Docker Desktop

利用している環境に応じた Docker Desktop 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 hub 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 docs を参照してください。

B.1.3 Download Docker Image

Docker Desktop をインストール・起動したら 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	Version-stable base R and RStudio, tidyverse, devtools
rocker/verse	Adds TeX and related packages to rocker/tidyverse
ykunisato/paper-r-jp	Dockerfile of writing paper by R Markdown
kmetrics/jverse	Japanized rocker/verse

rocker 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も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/jverse 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。

```
docker pull rocker/tidyverse
```

B.1.4 Run Container

Appendix C

RStudio IDE

データ分析勉強会では長らく R Commander（以降、Rcmdr）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 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て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 操作の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）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
- 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 記述が使える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る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。

C.1 Overview

RStudio を起動すると以下のような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

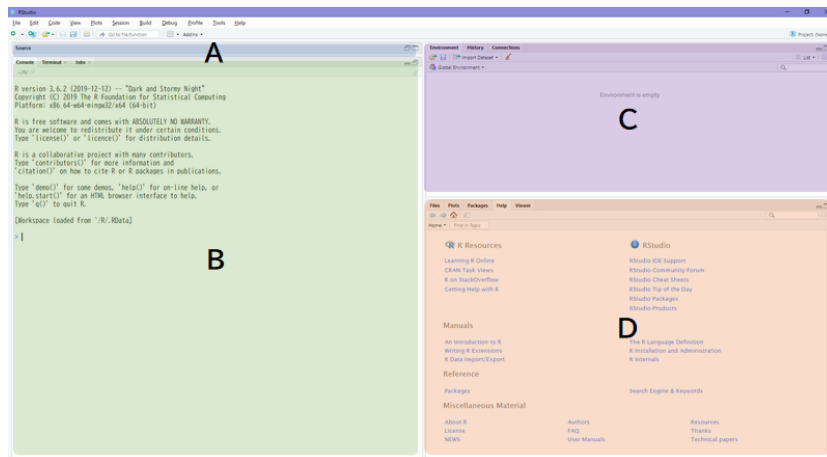


Figure C.1: RStudio Desktop, Windows

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。

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。

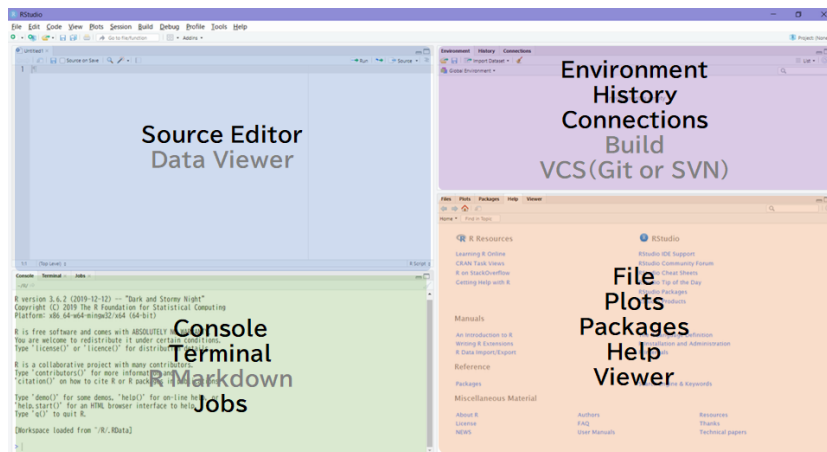


Figure C.2: RStudio Pane Layout, Windows

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可能）
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などを表示するビューア
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有）
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 を有効にしている場合のみ）
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）
16	D	Viewer	HTML 等の表示が可能なビューア

C.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

Keyboard Shortcuts	Description
[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。

C.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す。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。

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作成しておきます。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。

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。

すると上図のように三連のバッククォート（```）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。

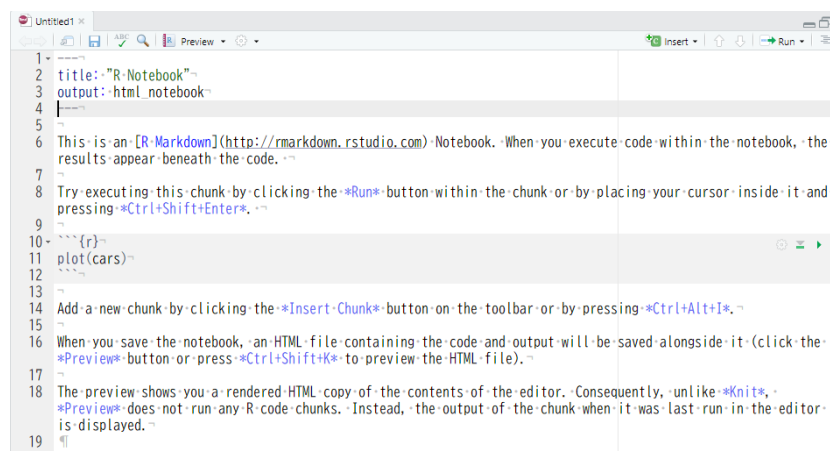


Figure C.3: R Notebook file



Figure C.4: R Notebook saved file



Figure C.5: R Notebook insert chunk

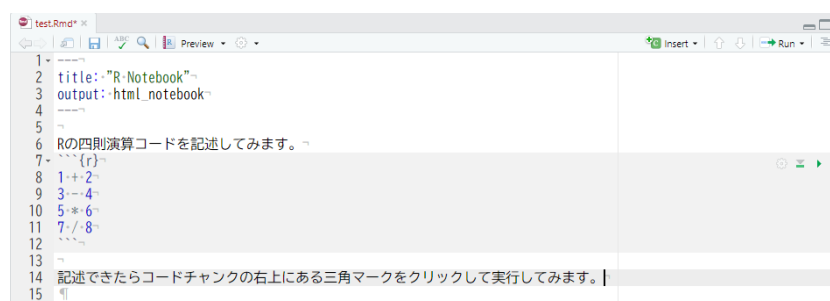


Figure C.6: R Notebook first code

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、コードチャンク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。

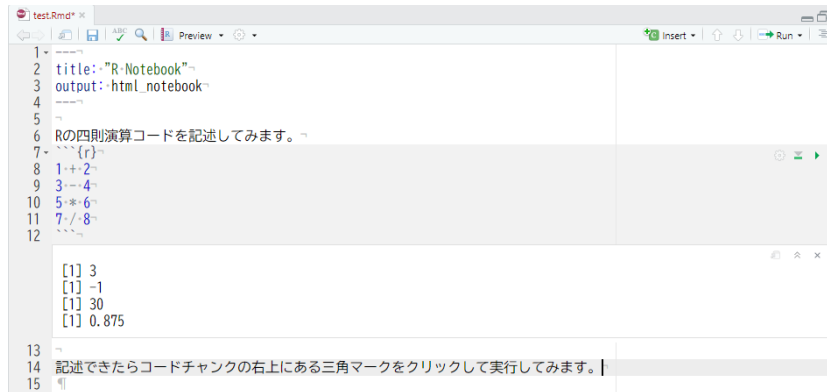


Figure C.7: R Notebook run code

コードチャンクの下と **Console** ペインに実行結果が表示されます。コードチャンク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。コードチャンクの下に実行結果が表示されます。

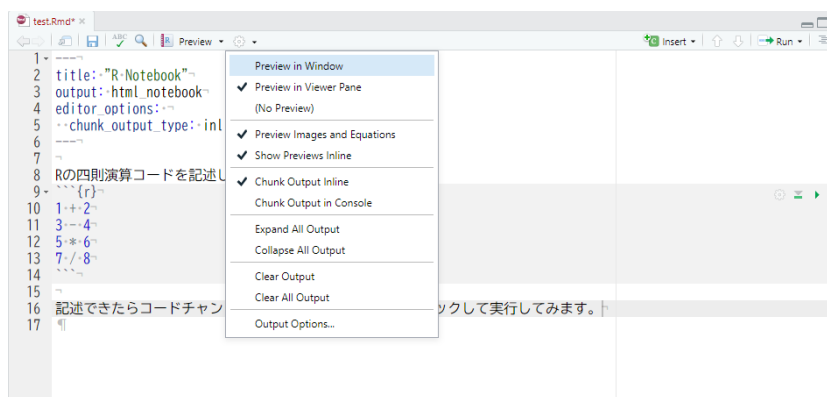


Figure C.8: R Notebook option

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。

C.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お好みで設定してください。

C.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
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	Workspace	Save workspace to .RData on exit	“Ask” or “No”
Basic	Other	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くと便利です。

C.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) に準拠することをおすすめしますので、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 atuo-indent	Checked
Editing	General	Surround selection on text insertion	“Quotes & Bracket
Editing	Execution	Always save R scripts before sourcing	Checked
Editing	Execution	Ctrl+Enter executes	“Multi-line R state
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 coloumn	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”

C.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

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

のフォントがおススメです。

- 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。

C.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ススメします。

C.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)	中項目 (太文字)	設定項目	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

大項目 (Tab)	中項目 (太文 字)	設定項目	推奨設定
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

C.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

C.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

C.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。

C.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。

C.4.10 Publishing

Publishing オプションは RStudio, Inc. が提供しているサービスへ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。

C.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を

C.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すめします。

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 D

Cloud IDE

開発環境のクラウドサービス化も進んでいます。

D.1 RStudio Cloud GA

RStudio Cloud 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。

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。

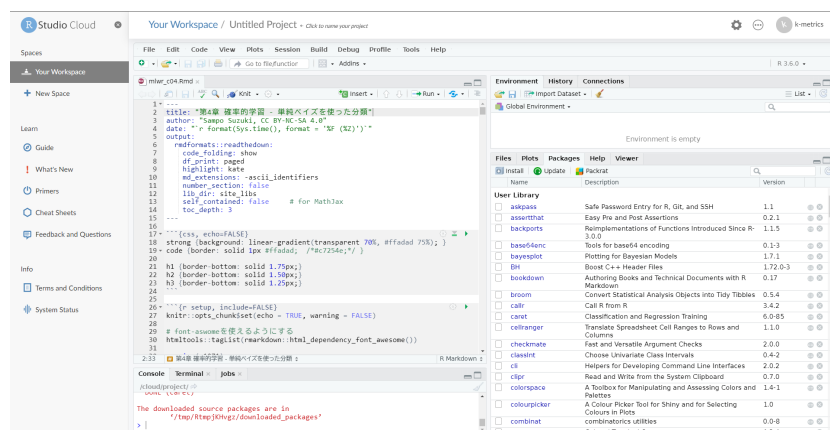


Figure D.1: RStudio Cloud, beta

D.2 Exploratory

Exploratory は BI (Business Intelligence) 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 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 ページにも様々な説明資料が用意されています。

価格 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 まはた Windwos でしか動作しません。

D.3 binder

binder 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

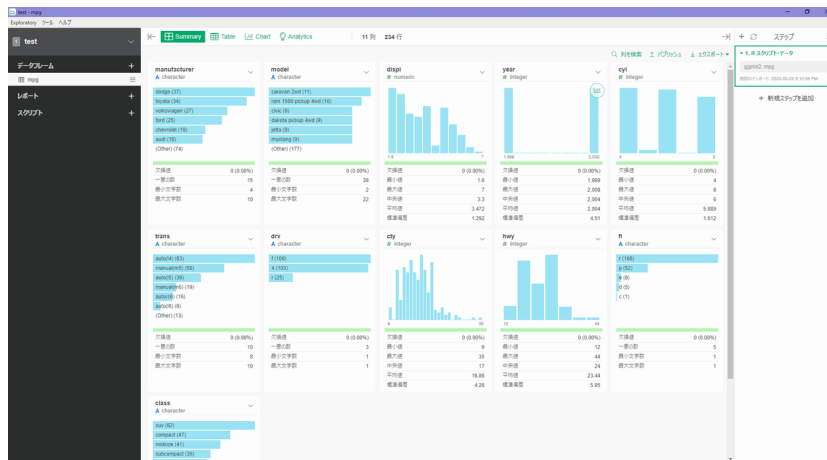


Figure D.2: Exploratory Public

(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・Exploratory 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。