R のすゝめ

- R によるデータ分析事始め -

Sampo Suzuki,

2021-05-08

Contents

License

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[1] 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。

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

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言われています。

『データ指向のソフトウェア品質マネジメント』[2] 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と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。この書籍の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[3] では、メトリクス分析に興味をもつ有志が統計分析を実践するために統計的コンピューティングを中心に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 R 言語（以降、**R** と記述）[4] 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を有しており **R** を用いて分析を行いたいと考えている方々を読者として想定しています。本書では **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**）[5] の利用をおすゝめします。

表記ルール

本書では以下の表記ルールを用いています。

[2] https://www.juse-p.co.jp/products/view/442
[3] https://sites.google.com/site/kantometrics/home
[4] https://www.r-project.org/
[5] https://colab.research.google.com/notebook#create=true&language=r

対象	表記方法	表記例
ハイパーリンク	脚注に URL を表記（PDF 形式の場合）	CRAN[1]
パス・ファイル名	モノフォント[a]	`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 () 付き表記	`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 [] 付き表記	[Ctrl]+[S]
数式	LaTeX 数式（math mode）	$y = ax^2 + b$
サービス名など	太字	**Google Colab**

[a] タイプライタフォントとも呼ばれる等幅フォントのこと

なぜ R を使うのか

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** は統計分析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[6] 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。

分析の手順

では、**R** を使った分析とはどのような手順になるかを簡単に見てみましょう。**R** は分析のための単なるツールですので、データ分析の常套手段してはツールに関係なく最初に分析対象となるデータがどのような分布なのか、どのような値の範囲にあるのか、全てのデータが揃っているかなどを俯瞰することからはじめます。例えば「フィッシャーの

[6] https://www.r-bloggers.com/six-reasons-to-learn-r-for-business/

あやめ（Fisher's or Anderson's iris)」を例にとってみましょう。「フィッシャーのあやめ」は **R** の例題でよく使われる下記のような 150 行のデータセットで R に標準で組み込まれています。

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
...	...	...	...	...	NA
148	6.5	3	5.2	2	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3	5.1	1.8	virginica

このデータセットの主な要約統計量は以下のようになり、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。

```
##   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
##
##
##
```

この要約の通りか箱ひげ図を描いて確かめてみますが、元のデータ形式では都合が悪いので以下のように変形しておきます。

Species	part	value
setosa	Sepal.Length	5.1
setosa	Sepal.Width	3.5
setosa	Petal.Length	1.4
setosa	Petal.Width	0.2
NA	NA	...
virginica	Sepal.Length	5.9
virginica	Sepal.Width	3
virginica	Petal.Length	5.1
virginica	Petal.Width	1.8

変形したデータを使って部位別の箱ひげ図を描くと確かに花弁（Petal）の方が萼片（Sepal）よりも小さい傾向にあることが分かります。

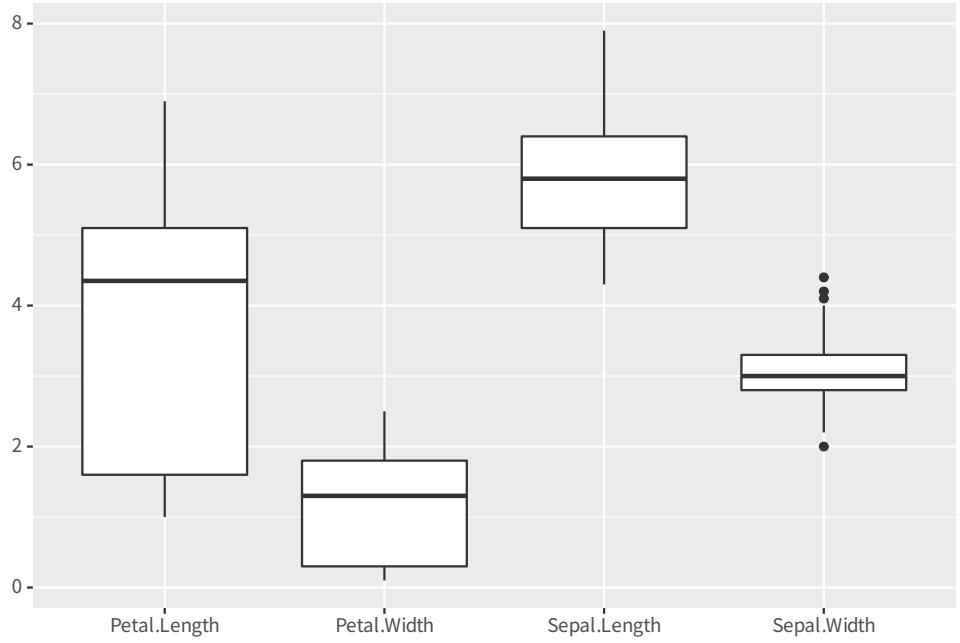
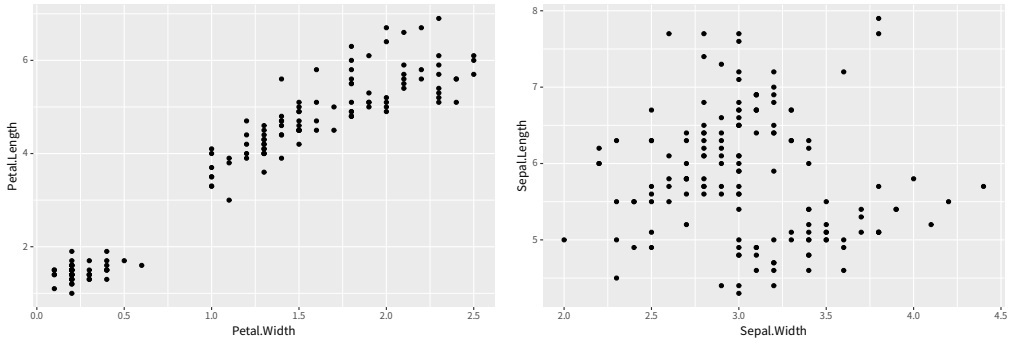


Figure 2: 花弁と萼片の分布

次に花弁（Petal）と萼片（Sepal）の幅と長さの関係を見ると花弁（Petal）の幅と長さに相関関係があるように見えます。



(a) 花弁　　(b) 萼片

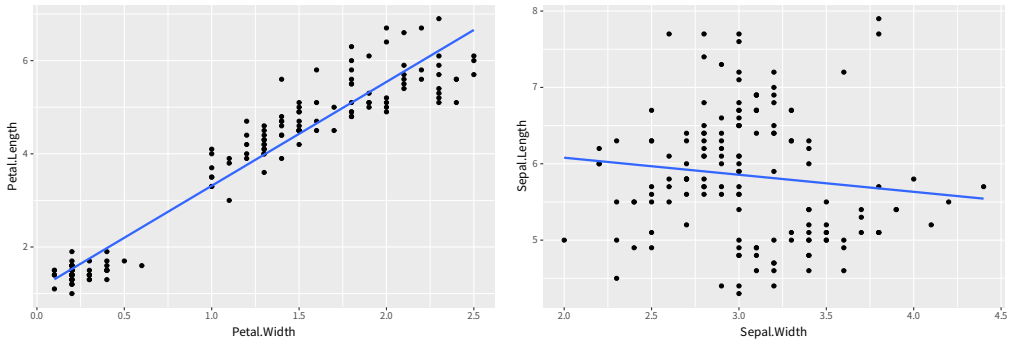
Figure 3: 幅と長さの関係

そこで、花弁（Petal）の幅（`Petal.Width`）と長さ（`Petal.Length`）の回帰関係を求めてみます。

```
##
## 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
```

モデルの当てはまり具合が良さそうなので、回帰関係を可視化してみます。ついでに萼片（Sepal）の方も可視化してみます。



(a) 花弁の回帰関係　　(b) 萼片の回帰関係

Figure 4: 幅と長さの関係

実際にはこれだけではデータ分析はできませんので、分析対象を

これがデータサイエンスワークフロー（Data Science Workflow）と呼ばれる分析フローです。

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。

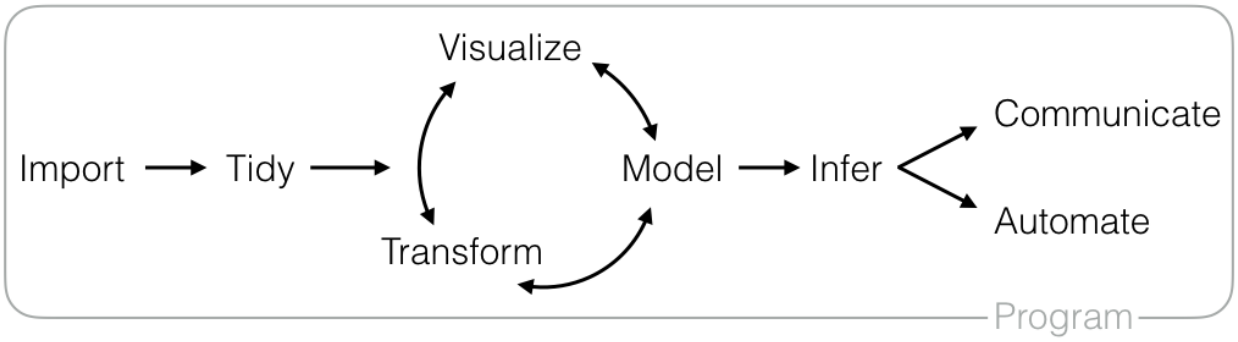


Figure 5: Data Science Workflow, CC BY-NC-ND 3.0 US, Hadley Wickham

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[7] が、その著書『R for Data Science』[8] 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 [9] 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。

[7] http://hadley.nz/
[8] https://r4ds.had.co.nz/
[9]

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）[10] が伴っていることも求められます。3 つの再現可能性[11]

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

[10]

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

Part II

Program

Chapter 1

分析環境

R について学ぶ際には R が使えるような環境を構築しておくべきですが、環境構築は初学者にとって最も厄介な作業です。そこで、本書では環境構築の必要がない **Google Colab** の利用をおすゝめします。**RStudio** 環境を構築できる方は **RStudi** でも構いません。

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。

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	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	
開発者	R + VS Code/Emacs	要	高	

1.1.1 R Commander

R Commander（以降、**Rcmdr**）[1] は、本書ではスコープ外ですが、SQiP 研究会の演習コースソフトウエアメトリクス（以降、メトリクス演習コース）[2] におけるデフォルトツールですので簡単に紹介しておきます。**Rcmdr** は **R** のパッケージとして提供されている GUI ベースの対話型分析環境です。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に特有の考え方を用いてる点には注意が必要です。

[1] https://socialsciences.mcmaster.ca/jfox/Misc/Rcmdr/
[2] https://www.juse.or.jp/sqip/workshop/outline/index.html

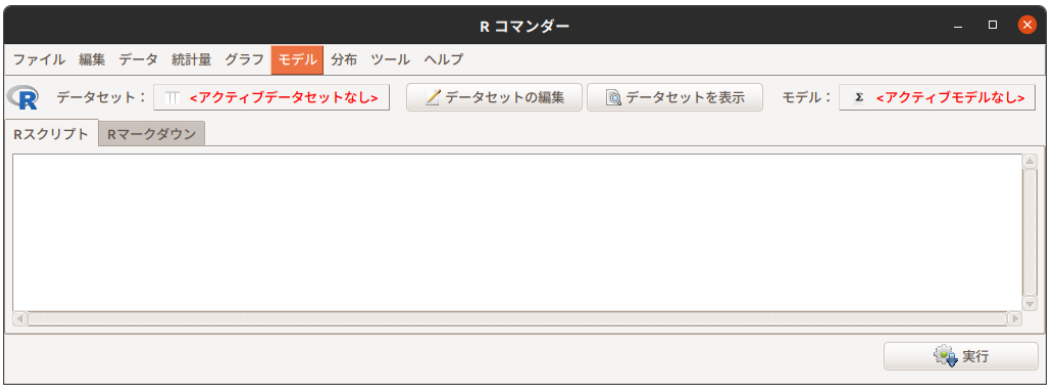


Figure 1.1: Rcmdr, Ubuntu

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** 向けの環境である **Jupyter Notebook** サービスです。**Jupyter Notebook** は **R** をエンジンとして利用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ることができますので、初学者の演習環境としておすゝめです。

Figure 1.2: Google Colab

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** 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 Eco System の中核とも言えます。その特徴として

- **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 Markdown**[3]や **Pandoc**[4] 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
- Git[5] 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

などがあります。

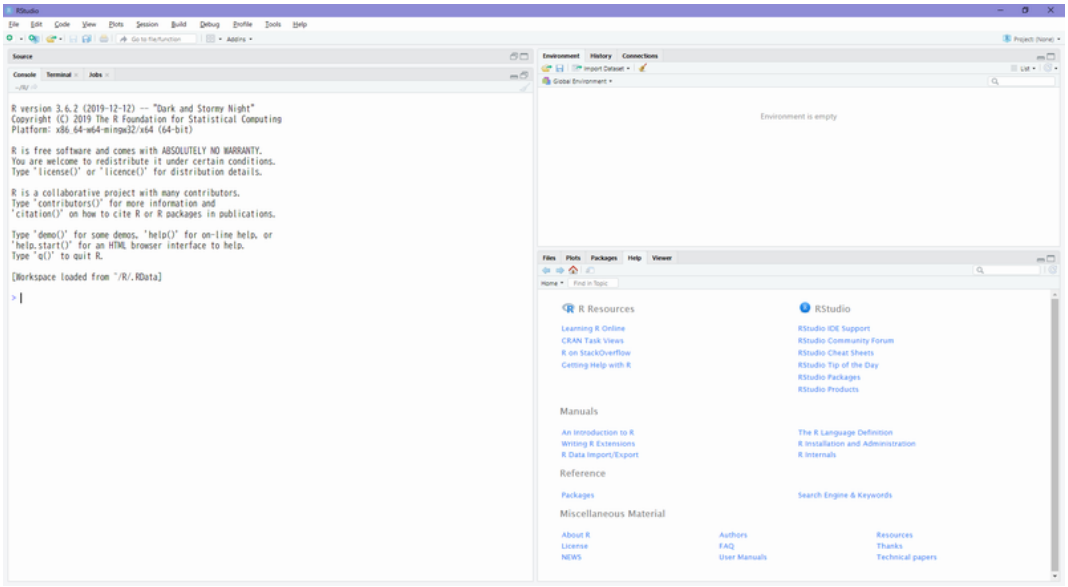


Figure 1.3: RStudio Desktop, Windows

RStudio Server は **RStudio** をブラウザ経由で使う Linux 上で動作するサーバ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も可能ですので、個々の PC での分析環境を固定さたい場合には非常に便利です。

[3]https://rmarkdown.rstudio.com/
[4]https://pandoc.org/
[5]https://git-scm.com/

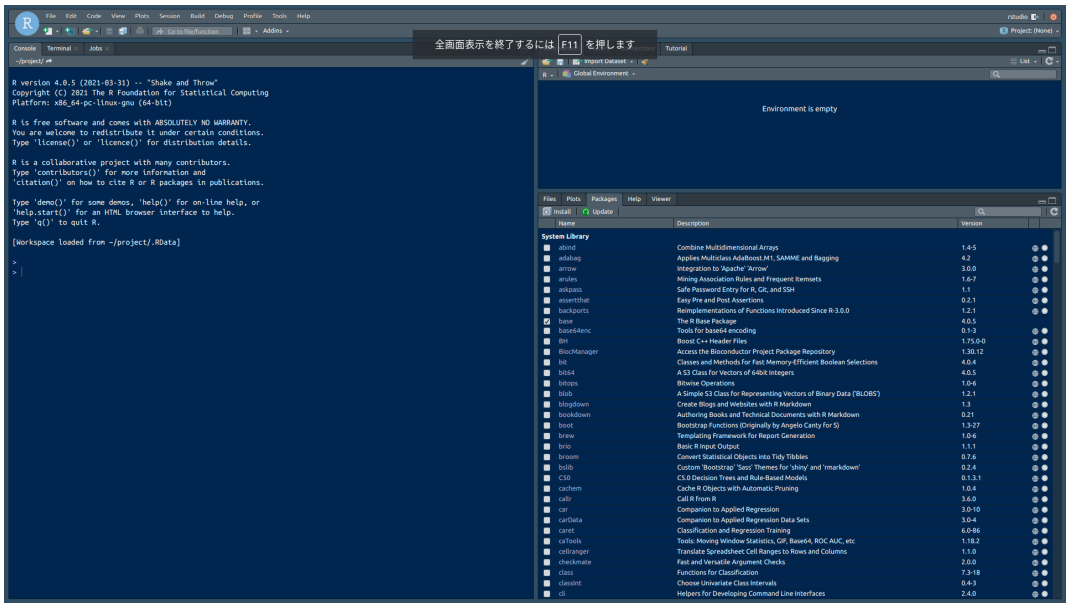


Figure 1.4: RStudio Server, Docker

1.1.4 RStudio Cloud

RStudio Cloud 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** に切り替えて使うことや **RStduio Package Manager** とも連携しています。また、英語版ですがチュートリアル機能が充実しているのも特徴です。無料プランが用意されていますが利用時間が 15 時間/月に限定されていますので、繋げっぱなしでの長時間利用には不向きです。

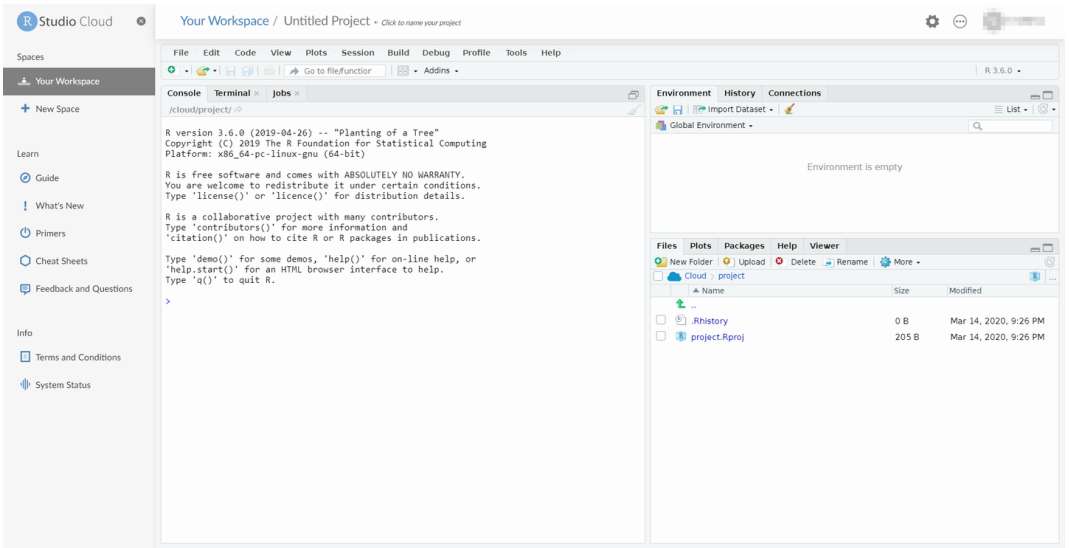


Figure 1.5: RStudio Cloud

1.1.5 Programing 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可能です。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 の VisualStudio Code 用の機能拡張を使えば、お好みのエディタから直接 **R** を使えるようになります。

Chapter 2

R の基本

> 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にあります。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。

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

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

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

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

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] https://www.slideshare.net/shuyo/r-4022379

```
##  [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。

2.1 基本的な演算

最初に基本的な演算です。

2.1.1 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 + 2     # 加算
```

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

```
2 - 3     # 減算
```

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

```
3 * 4     # 乗算
```

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

```
4 / 5     # 除算
```

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

[2] https://github.com/k-metrics/kmetrics

2.1.2 代入演算

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。

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

2.1.3 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。

```
w
```

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

```
x
```

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

```
y
```

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

```
z
```

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

```
print(w)
```

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

```
print(x)
```

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

```
print(y)
```

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

```
print(z)
```

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

2.1.4 合算値

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。

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

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

2.1.5 平均値

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。

```
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。

```
x <- c(w, x, y, z)
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。

2.1.6 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。

```
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。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ので注意してください。

2.2 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。

2.2.1 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。

2.2.2 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	

[3] https://style.tidyverse.org/syntax.html#object-names

型	クラス	タイプ	モード	格納モード	備考
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

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。

2.2.3 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	factor, ordered	インデックスを持つ変数型
マトリクス型（行列型）	matrix	二次元のベクトル型
アレイ型（配列型）	array	多次元のベクトル型
データフレーム型	data.frame	**等長**なベクトル型
リスト型	list	自由度が最も高い変数型

2.2.3.1 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。

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

```
##  int 2
```

```
x <- c(1, 2, 3)         # 実数の 1 から 3 の三つの値が代入
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ォート（ダブルまはたシングル）で囲みます。

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

2.2.3.2 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。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）という属性がつくベクトル型変数です。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"
```

2.2.3.3 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。

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。

2.2.4 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
```

2.2.5 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, TRUE,
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。

2.2.6 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"), x)
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
```

2.3 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
```

2.3.1 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

2.4 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。

2.4.1 [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
```

2.4.2 $ 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。

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

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

```
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。

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

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

```
##### Operators ##############################################################
```

リスト型を $ 演算子で参照するの場合は要素が 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。

```
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"
```

2.5 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	is.logical	
整数型	is.integer	
実数型	is.double	
数値型	is.numeric	整数型または実数型の場合 TRUE
複素数型	is.complex	
文字型	is.character	

同様に変数型を検査するための is. 関数軍も用意されています。

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

定数用の is. 関数群もあります。

定数	関数	備考
NULL	id.null	NULL 値か否か
NA	is.na	欠損値か否か
NaN	is.nan	非数か否か
inf	is.infinit	無現値か否か（有限値か否かを確認する is.finit 関数もあり）

2.6 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)
```

```
##  [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
```

2.6.1 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
```

2.6.2 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
```

2.6.3 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

2.6.4 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）厳密にい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
%o%	外積（ベクトル積）
%x%	クロネッカー積

2.6.5 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）	

演算子	説明	順位
==, !=	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）	
&, &&, |, ||	論理積、論理和（論理演算子）	
~	フォーミュラ	
<-	代入	低

2.7 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。

2.7.1 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
`if else`	基本的な条件分岐（予約語）
`switch`	条件が多数に分岐する場合に便利（予約語）
`ifelse`	Excel の IF 関数に似た条件分岐（関数）

2.7.2 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)"
```

2.7.3 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。

2.7.3.1 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
```

2.7.3.2 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"
```

2.7.4 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)
```

2.7.5 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 [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
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

2.7.6 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
```

[4] http://masato-613.hatenablog.com/entry/2017/04/25/064632

```
  } else {
    print(as.character(i))
  }
}
```

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

2.7.7 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 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

Program

Chapter 13

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ゝめします。

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 Dekstop（以降、RStudio）を利用しても構いません。

13.1 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）の利用をおすゝめします。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
5. アップロードしたファイルを Google ドライブに保存する

[1] https://colab.research.google.com/?hl=ja
[2] https://rstudio.cloud/

13.1.1 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 Google [3] 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。

13.1.2 Open Google Colab

Google で Google Colab を検索して Colaboratory - Google Colab [4] のリンクをくと以下のような画面が表示されます。

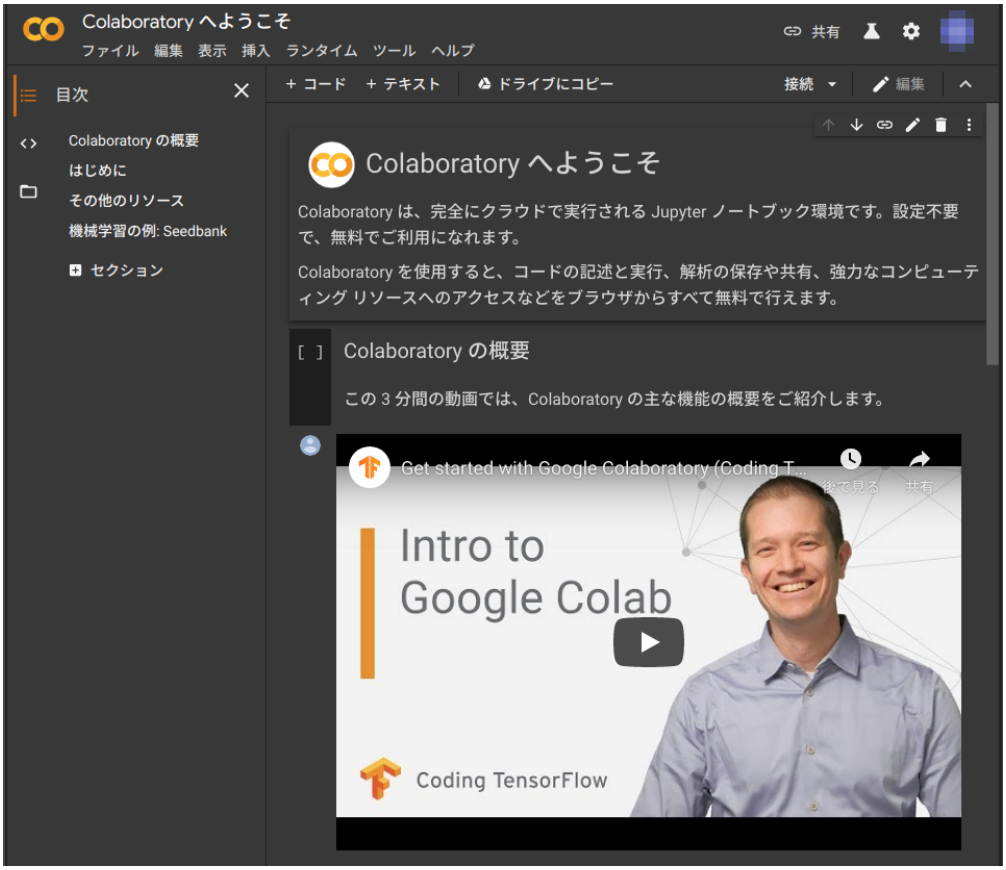


Figure 13.1: Google Colab, Theme: dark

画面テーマは右上の歯車ボタンから変更できます。

[3] https://www.google.co.jp
[4] https://colab.research.google.com/notebooks/welcome.ipynb?hl=ja

13.1.3 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。

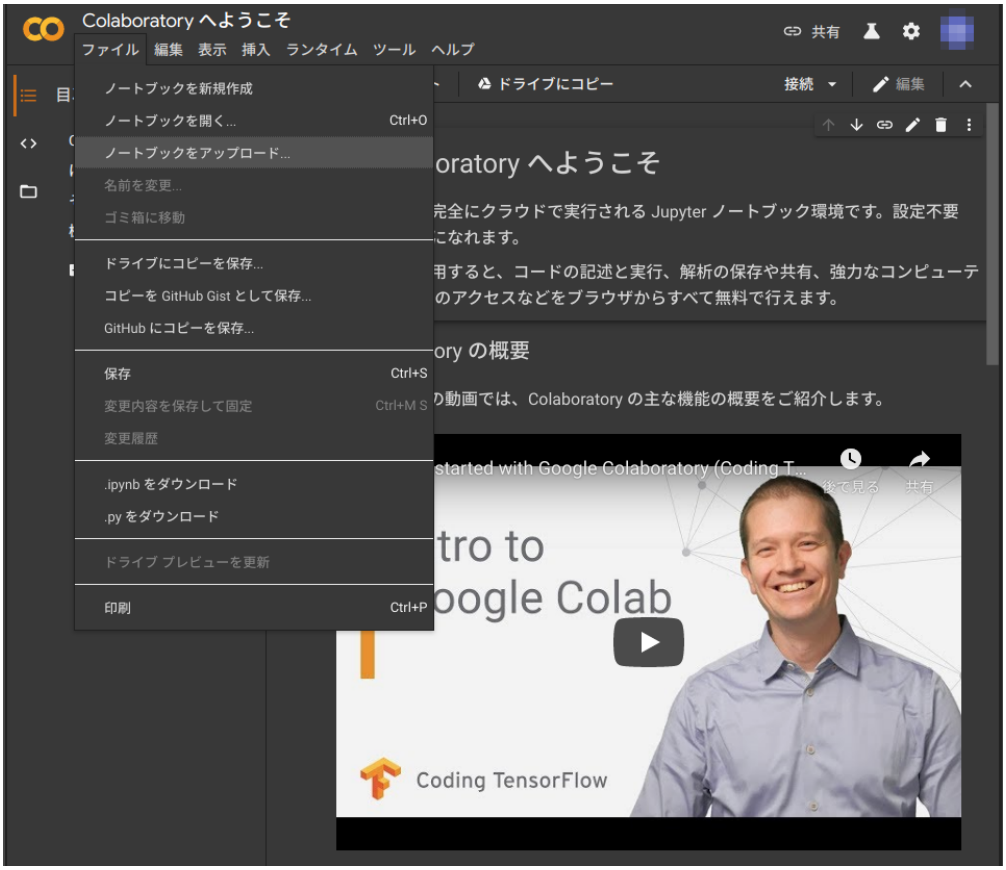


Figure 13.2: Upload notebook file

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
```

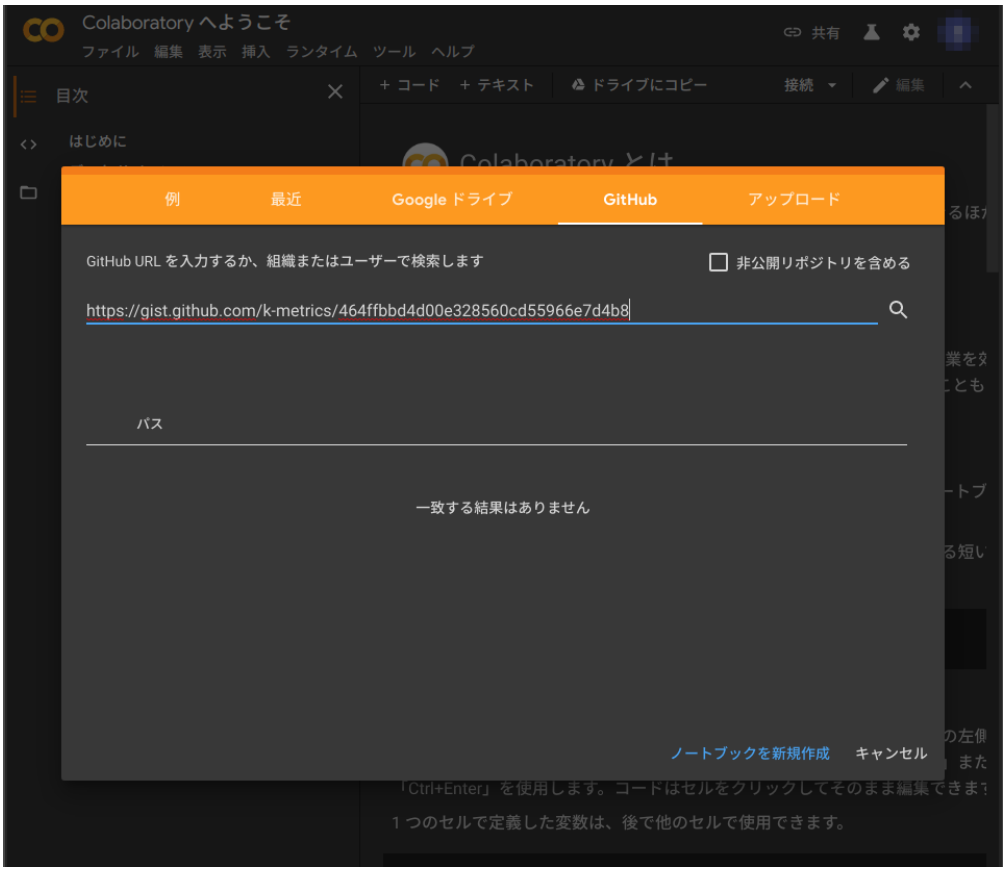


Figure 13.3: Upload from GitHub

テンプレートがアップロードされ表示されます。

13.1.4 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。

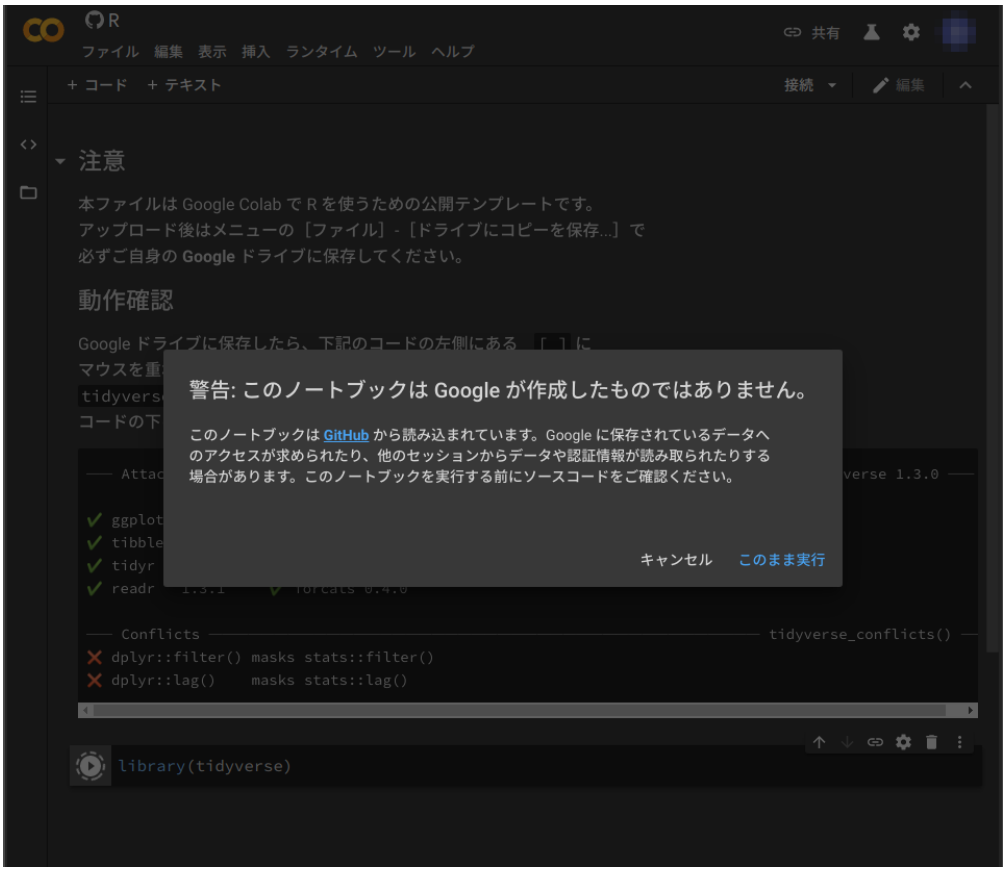


Figure 13.4: Warning dialog

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。

13.1.5 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。

13.2 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。

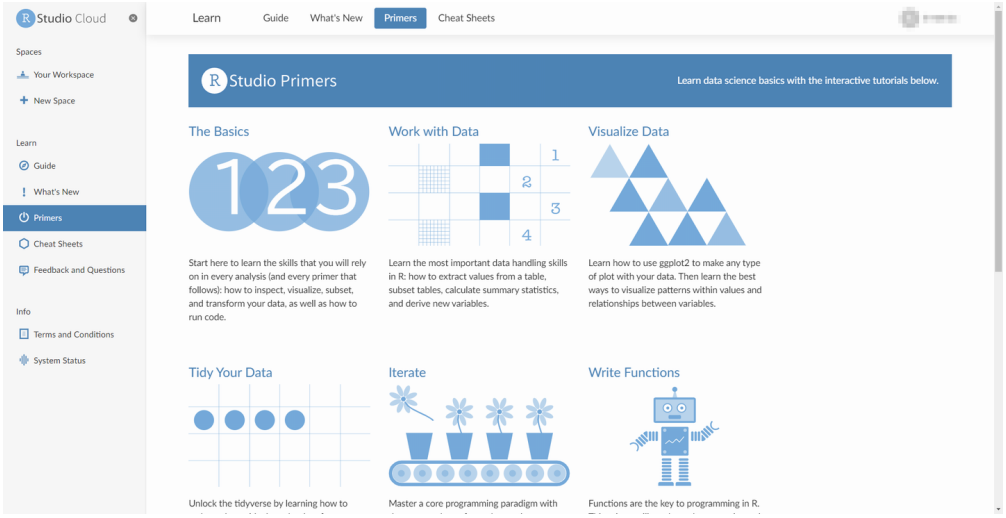


Figure 13.5: RStudio Cloud, beta

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

13.2.1 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/

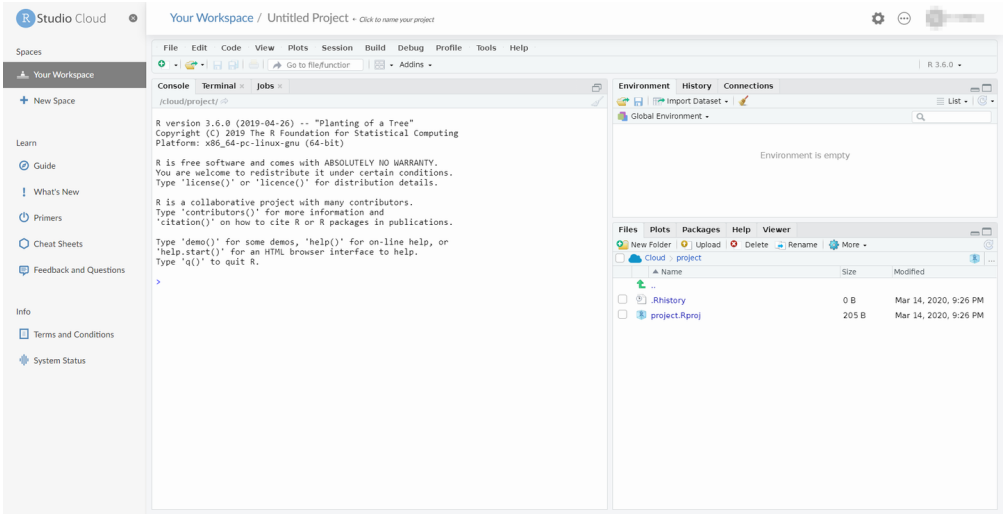


Figure 13.6: Initial View

13.2.2 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。

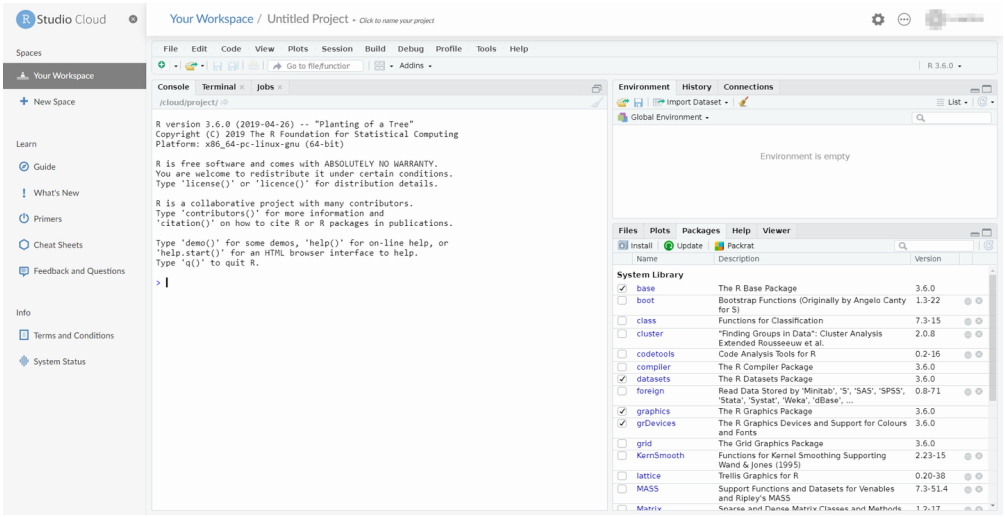


Figure 13.7: Packages Manager

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。

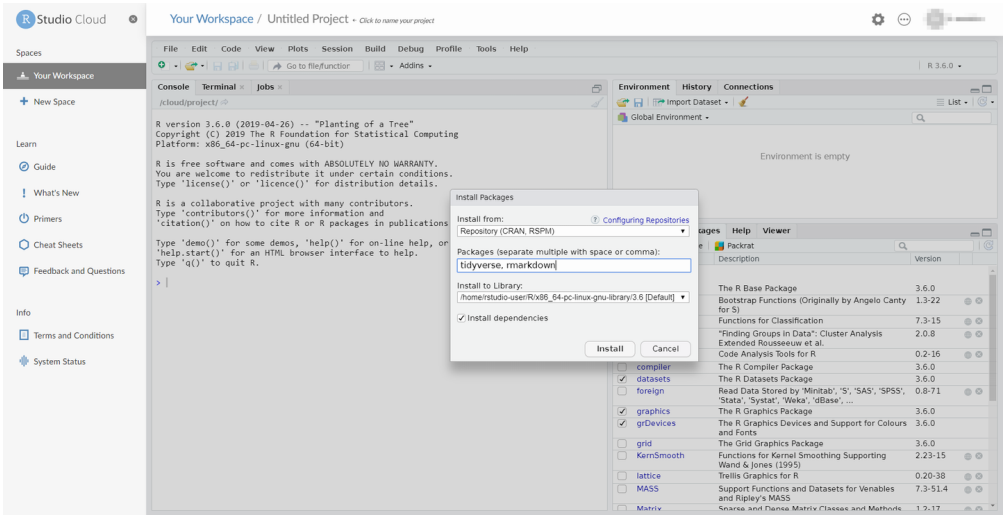


Figure 13.8: Install Dialog

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/[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	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

Git [2] 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)[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

[1] https://www.tidyverse.org/
[2] https://git-scm.com/
[3] https://cran.r-project.org/
[4] https://cran.r-project.org/mirrors.html

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。

A.1.1 Windows

Winodws では特段の理由がない限り CRAN [6] 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 [7] 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。

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 [9] 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 [10] 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 [11] 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。

[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/
[11] https://cran.r-project.org/bin/linux/redhat/README

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 [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。

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)
```

[12] https://rstudio.com/products/rstudio/download/#download

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 [13] 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 [14] 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

[13] https://rmarkdown.rstudio.com/
[14] https://demo.rstudiopm.com/client/#/

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 の利用をおすゝめします。

A.4.1 Git

Windows および macOS は Git の ダウンロードページ [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。

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 [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

[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
[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 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 [1] 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 [2] 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 [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/

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 [4] 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 [5] 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 [6] 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 [7] 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
kmetrics/jverse [11]	Japanized rocker/verse

[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/

rocker [12] 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 [13] 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

[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 C

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）を利用をおすゝめします。

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
- 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

[1] https://www.rcommander.com/
[2] https://www.juse.or.jp/sqip/workshop/outline/index.html
[3] https://rstudio.com/products/rstudio/

  - 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。

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。

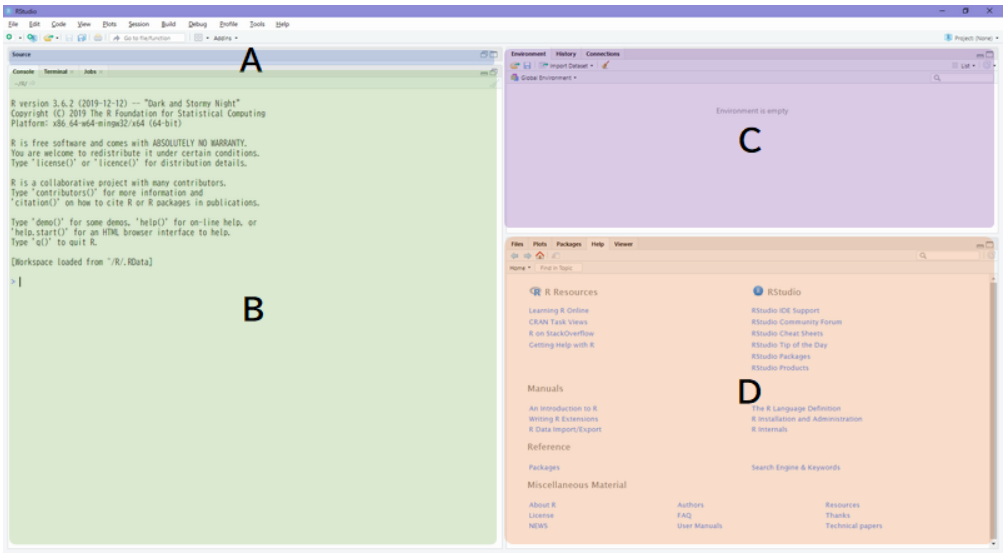


Figure C.1: RStudio Desktop, Windows

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。

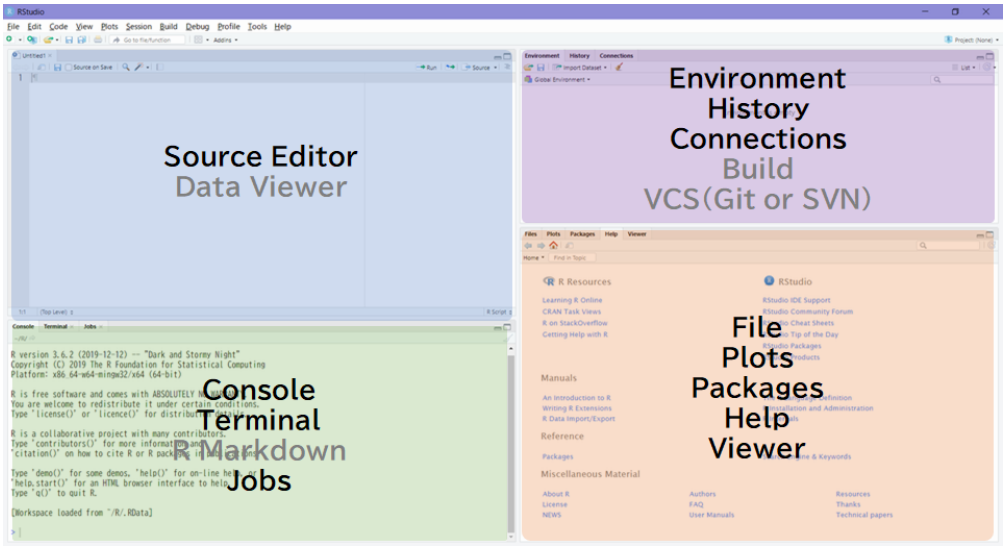


Figure C.2: RStudio Pane Layout, Windows

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するこ
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
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 を有効にしてい
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
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
[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
[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
[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。

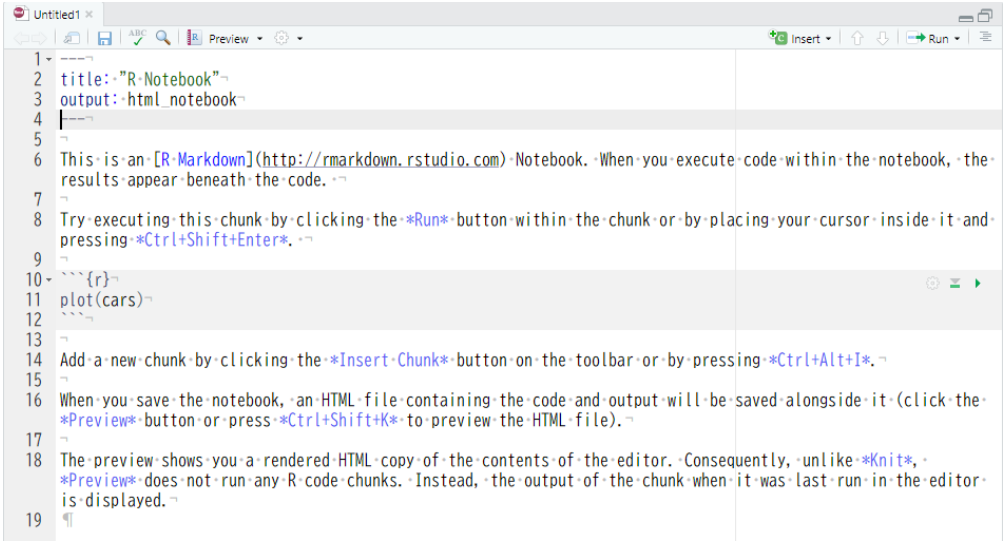


Figure C.3: R Notebook file

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。

Figure C.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。

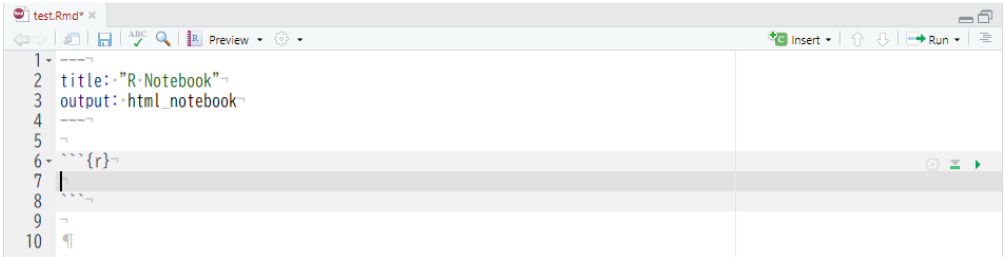


Figure C.5: R Notebook insert chunk

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。

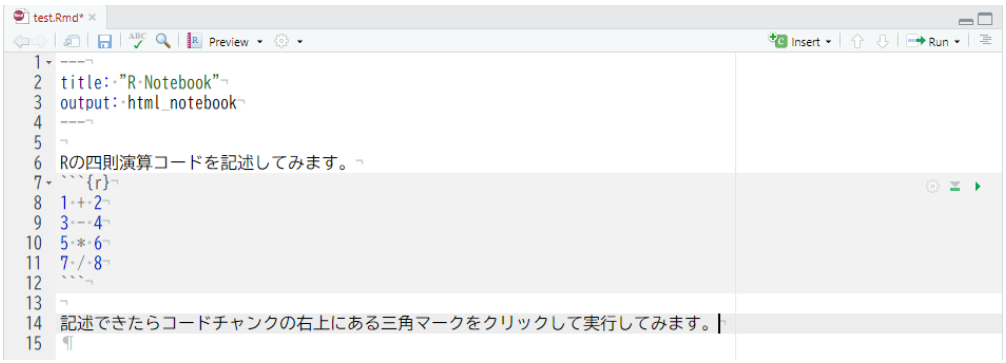


Figure C.6: R Notebook first code

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。

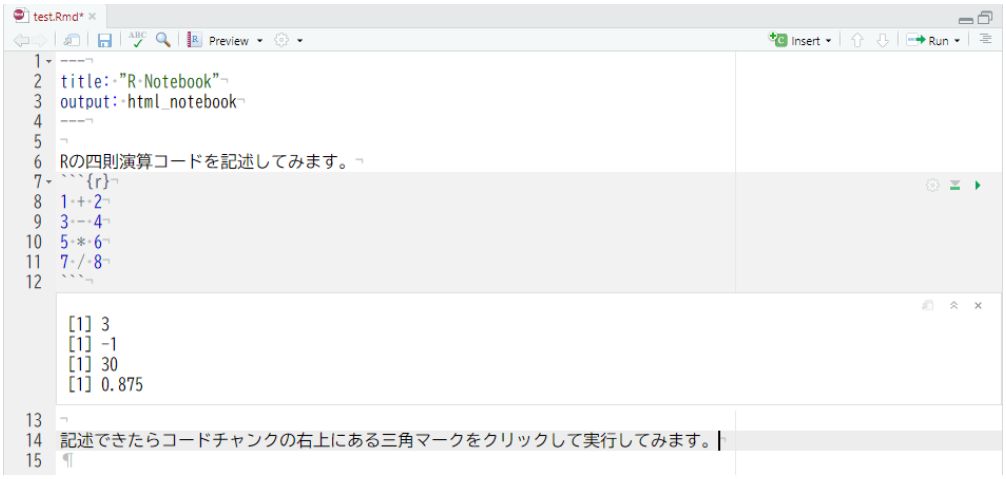


Figure C.7: R Notebook run code

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。

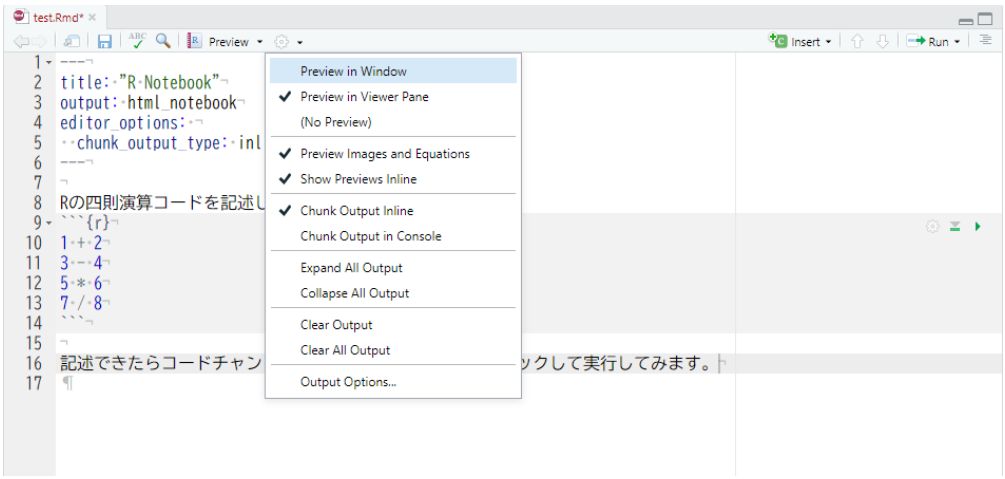


Figure C.8: R Notebook option

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

Genelal 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 or
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。

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）[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 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

[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ゝめです。

- 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。

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すゝめします。

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
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 WebSockts	Terminal が起動しない場合はチェックを外

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 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 Colud [1] 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。

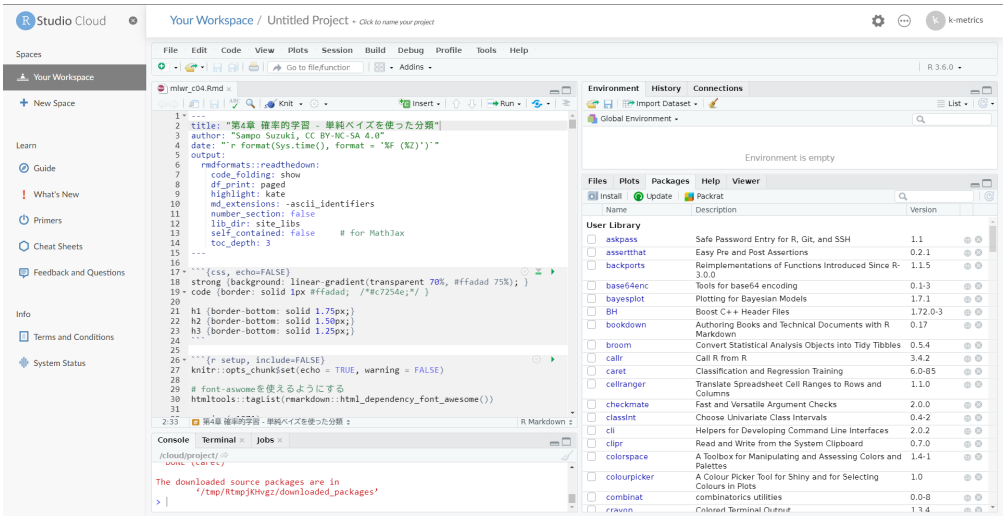


Figure D.1: RStudio Cloud, beta

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向いて

[1] https://rstudio.cloud/

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。

D.2 Exploratory

Exploratory [2] は BI（Business Intelligence）BI（Business Intelligence）ツールのような操作で R を持ち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。

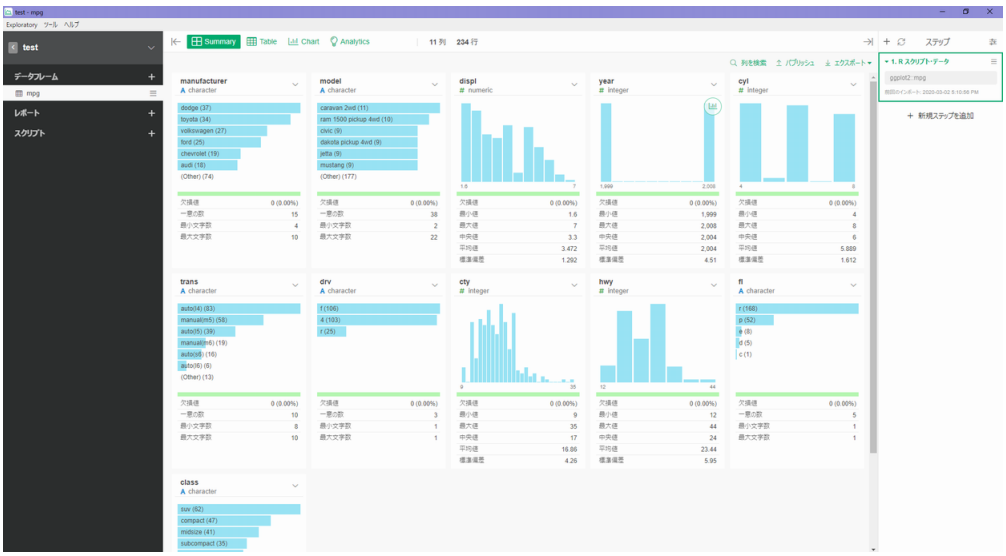


Figure D.2: Exploratory Public

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 またた 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

D.3 binder

binder [6] 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あ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/