

R ビギナーズガイド

Syunsuke Fukuda

2019-07-13

目次

第 1 章	初めての RStudio	5
1.1	何を起動するのか	5
1.2	ペインとタブ	6
1.3	コンソール	7
第 2 章	コンソールとコマンド	9
2.1	コマンド	9
2.2	履歴機能	10
2.3	補完機能	11
2.4	複数行入力	12
2.5	変数と代入	13
2.6	ベクトル	14
2.7	データフレーム	18
第 3 章	パス	23
3.1	ファイルを扱うために必要不可欠な知識	23
3.2	Windows のディレクトリ構成	23
3.3	パスの書き方のルール	24
3.4	Windows のパスの注意点	25
3.5	カレントディレクトリ	26
3.6	カレントディレクトリを意識した作業	27
3.7	RStudio のプロジェクト	29
第 4 章	パッケージ	33
4.1	RStudio でのパッケージのインストール	34
4.2	パッケージの管理	37
4.3	パッケージを読み込む library() 関数	38
4.4	GitHub にあるパッケージのインストール	38
4.5	有名なパッケージ	40
第 5 章	日常作業は、R Notebook で行う	43
5.1	作業の準備はプロジェクトから	43
5.2	R Notebook を新規作成する	44
5.3	R Notebook の構造	45

5.4	R Notebook とマークダウン	46
5.5	マークダウンと R マークダウン	51
5.6	R Notebook とは	54

第 1 章

初めての RStudio

R やプログラミング初心者、自称文系でパソコンに対する知識があまりない方へのガイドです。

コンソール上でのコマンドやファイルパスに関する知識を紹介したうえで、統合開発環境 RStudio に触れてもらい、最終的に R Notebook で作業が出来るようになることが目標です。

1.1 何を起動するのか

初心者向け R のインストールガイド に従って準備作業された場合には、R と RStudio がインストールされていると思います。ですから、Windows のスタートメニューを見ると、R と RStudio の 2 つがあります（図1.1）が、このうち、これから皆さんが使っていくのは **RStudio** の方です。

R は、R 言語を処理するためのプログラムであり、入力されたコマンドを解釈して処理し、その結果を出力するプログラムです。そして、それ以外のことは殆どしてくれません。

プログラミングの経験があまりない方にとっては、「それ以外のことって何？パソコンはただ計算して処理してくれるものだし、それで十分じゃないの？」と思われるかもしれません。

しかし、実際のプログラミング作業では、正確なコマンド入力を行い、プログラミングの流れを正確に把握し、沢山の必要となるファイルを管理する必要があります。そして、この何千行もあるプログラミングコードの中のただの一字を間違えたただけでも、プログラムは正しく動いてくれなくなります。

そこで、このように面倒なプログラミング作業を正確に行うためには、通常、「それ以外のこと」が大いに必要となります。一般的に、プログラミング作業を行う場合には、この「それ以外のこと」もひっくるめて、誰でも正確なプログラミングが簡単にできるようにプログラミング作業をサポートしてくれるソフトを使うことになります。

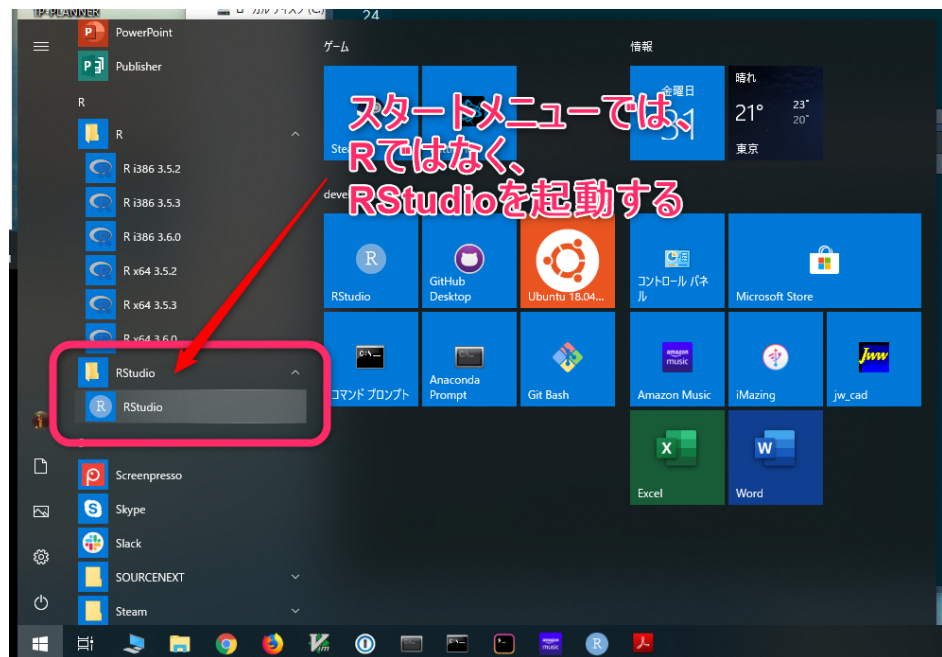


図1.1 スタートメニューの中の RStudio

このプログラミング作業をサポートしてくれるソフトのことを **統合開発環境** と呼びます。英語では、Integrated Development Environment と呼ばれるため、略して **IDE** とも呼ばれます。この IDE には、沢山のものがありますが、R のプログラミングを行う際に定番な IDE は、**RStudio** です。

ですから、R の作業を行う時は、Windows のスタートメニューから **RStudio** を起動します。

1.2 ペインとタブ

RStudio を立ち上げて、まずは、RStudio の見た目に慣れましょう。大雑把に見て3つの部分に分かれていることが把握できるはずです。(図1.2) この区分されている一つ一つの部分を区画やペインと呼びます。

そして、その各ペインの上部に注目してもらくと、どのペインにもタブを見つけることが出来るはずです。(図1.3) タブは、それをクリックすることでペインの内容を切り替えることが出来ます。

RStudio は、小さな画面の中に沢山の機能が詰め込まれているので、タブの文字やその周辺に表示されているアイコン、更にはその他の情報の文字が非常に小さくなっています。ですから、いつも見ている画面なのに、その存在を意識することがないと、それがそこにあることさえ気づかないこともあります。

まずは、RStudio は、いくつかのペインで構成され、各ペインはその内容をタブで切り替えられるということを把握しましょう。

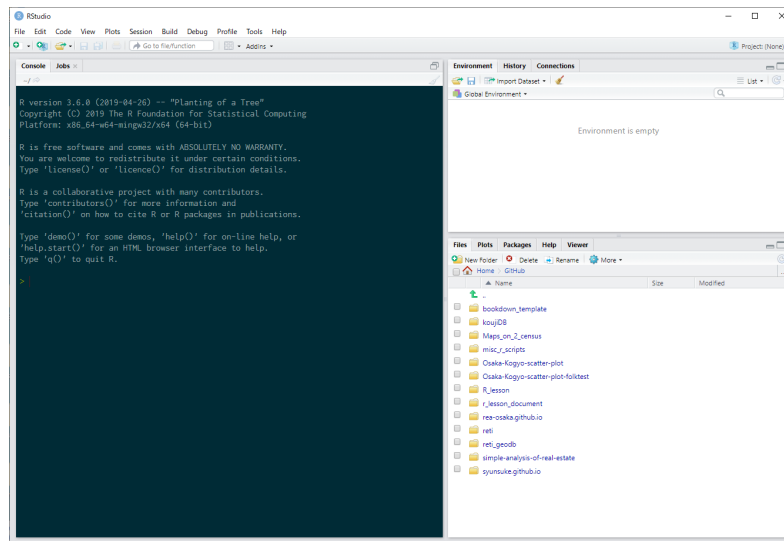


図1.2 ペインで構成される RStudio

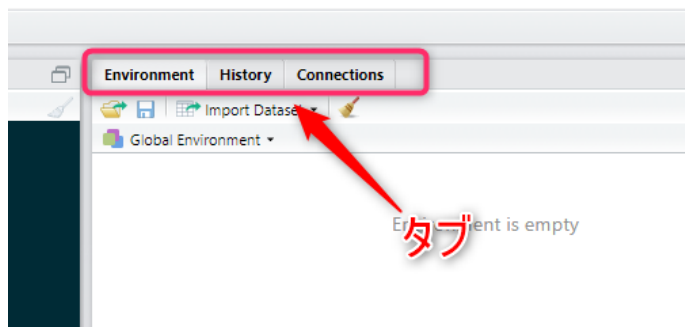


図1.3 ペイン上部の tab

1.3 コンソール

立ち上げたばかりの RStudio（図1.2）は、左側のペインにコンソールが表示されています。タブ名が Console になっていることを確認しましょう。

コンソールは、ユーザーからの R コマンドを受け付け、それを処理して、結果を表示してくれる場所です。

起動したばかりの RStudio のコンソールには、内部で動いている R 処理プログラムのバージョンについて述べられ、続いて、ライセンスや R を支える開発貢献者等、更に、ヘルプ等を見るためのコマンドの紹介がなされ、最後に、R を終了するためのコマンドが記されています。

そして、これらメッセージの後ろに “>”（大なり記号）が行頭にぽつんと表示され、その右側に “|”（縦棒）が点滅しているのが見て取れるはずです。この、“>” の印のことをプ

プロンプトといいます。そして、“|”の印のことをカーソルといいます。

コンソールでの作業は、まず、プロンプトに対して、コマンドとして意味のある文字列を入力し、入力完了したら、キーボードの **Enter** キーを押して、入力したコマンドを R に処理させます。そして、その処理がおわると、R はコンソール上に何らかの結果を表示し、再び、コンソールにはプロンプトが現れ、コマンド入力待ちの状態に戻ります。

では早速、実践してみましょう。プロンプトに対して `1 + 2 + 3` というコマンドを入力し、実行してください。

1. コンソールのプロンプトに `1 + 2 + 3` と書き込む
2. Enter キーを押す
3. 結果をみる

```
1 + 2 + 3
```

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

実は、このプロンプトは、「コマンドの入力を受け付けています」という意味を持っています。

コマンドを実行して処理をしている最中には、このプロンプトは表示されません。普段は、あっというまに処理が終わるので、いつでもプロンプトが表示されているように見えますが、時間のかかる処理をしている時には、なかなかプロンプトが表示され無い事もあります。つまり、コンソールにプロンプトが無い時は、前のコマンドの処理中なので、次のコマンドを入力するためには、前のコマンドの処理が終わり、プロンプトが表示されるのを待つ必要があります。

第 2 章

コンソールとコマンド

2.1 コマンド

プロンプトに対して渡すコマンドの基本形を見てみましょう。

2.1.1 四則演算

プロンプトに対して、通常の四則演算式を入力すれば、電卓変りに使うことが出来ます。

```
# 四則演算
(350 + 120) * (100 - 20) / (10 * 10)
```

```
## [1] 376
```

2.1.2 関数

通常のコマンドは、R の関数を入力して実行します。まずは、現在の時間を教えてくれる `date()` という関数を試してみましょう。

```
# 現在の時刻を表示する関数
date()
```

```
## [1] "Sat Jul 13 11:13:53 2019"
```

関数には決まったパターンがあります。`date()` を見て把握できる通り、関数は、必ず、関数名とそれに続く括弧から成り立っています。上記の例である `date()` は、かっこの中に何も書いてありませんでしたが、括弧の中には引数と呼ばれるものが入ることもあります。引数は、「ひきすう」と読みます。

では、引数をとる関数を試してみましょう。`sum()` 関数は、引数として渡された数字の合計を返してくれます。

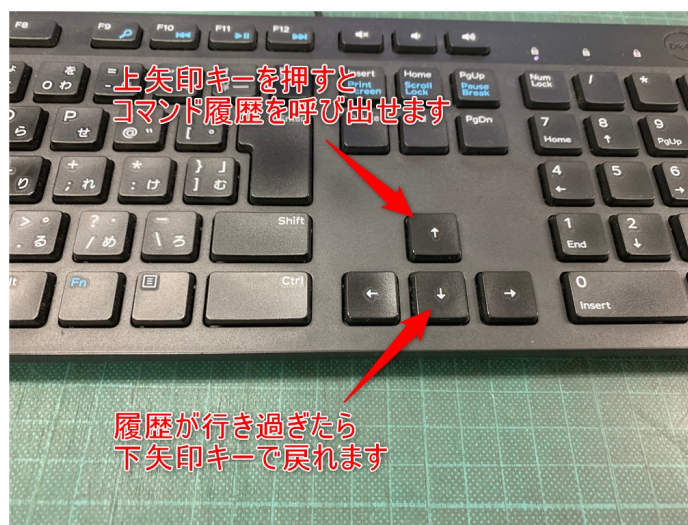


図2.1 キーボードのカーソルキー

```
# 引数として渡された数字を合計する sum() 関数
```

```
sum(1, 2, 3, 4, 5)
```

```
## [1] 15
```

この様にカンマを使って区切ることで、関数には複数の引数を渡すことが出来ます。関数の引数は、その各関数によって、どのような引数を取るのかが決まっています。また、その引数の場所に意味があったり、引数を省略すると自動的に既定値（デフォルト値）をとることもあります。このような引数に関するルールについては、のちに学習することになります。ここでは、まず、関数は引数を取ることがあるという基本的な事を把握できれば十分です。

2.2 履歴機能

先の `sum()` 関数では、1 から 5 までの数字を合計しました。次は、1 から 6 までの数字を合計したいとしましょう。実際の計算作業においても、一度行った計算の一部を調整して、再度、似たような計算を行うこともあると思います。そんな、作業をサポートしてくれるのが履歴機能です。

コンソールにフォーカスがある状態で、カーソルキーの上矢印“↑”を一を押してください。（図2.1）前回実行したコマンドがプロンプト上に入力された状態になりましたか？何度もこのカーソルの上矢印キーを押すと、どんどん、前に入力して実行したコマンドに変わっていくはずですが。これを履歴機能と呼びます。今度は、下矢印“↓”キーを押して下さい。そうです、行き過ぎたら戻ればよいわけです。

このように、面倒で複雑なコマンドも1度入力してしまえば、いつでも指一本で簡単に呼び出せるようになります。ですから、似たようなコマンドは1から打ち込むことなく、以

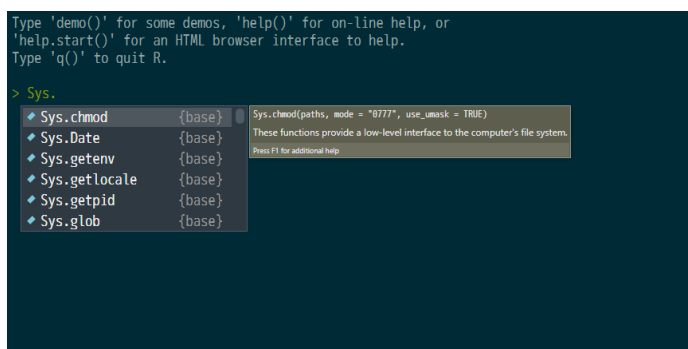


図2.2 補完メニュー

前実行したコマンドを再度呼び出し、必要な部分のみを修正するだけでよくなるので、入力が格段に簡単になります。

では、1 から 6 までの合計値を計算してみてください。

```
# 履歴を活用して 1 から 6 の合計値を計算する
```

```
sum(1, 2, 3, 4, 5, 6)
```

```
## [1] 21
```

2.3 補完機能

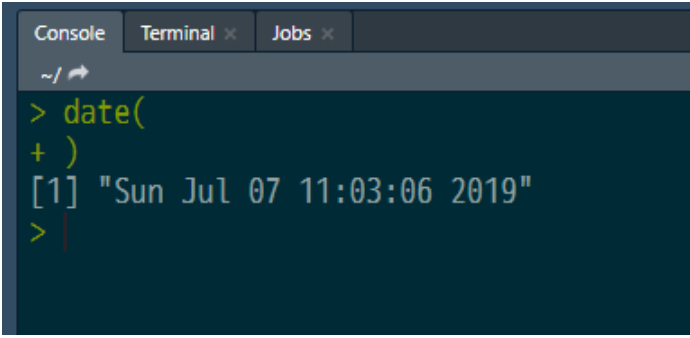
補完機能は、コマンドや変数を途中まで入力しさえすれば、残りの部分を適切に補って入力してくれる機能です。

システムの保持している日時を出力してくれる `Sys.Date()` 関数で実践してみましょう。プロンプトに `Sys.` と入力してください。入力している傍にメニューが現れ、その中に `Sys.Date` を見つけることができるはずです。(図2.2)

カーソルキーを使って、`Sys.Date` を選択してから、Enter キーか Tab キーを押すと、プロンプトに `Sys.Date()` が入力されます。

プロンプトへの入力中、補完メニューが出ていない状態で、Tab キーを押すと、補完メニューを呼び出すことが出来ます。補完メニューを呼び出したまま、入力続けると、補完候補がリアルタイムで絞り込まれていきます。補完候補は、コマンド等の綴りを正確に覚えていない時でも、適当に試しながら探することが出来ます。例えば、よく使う `library()` 関数の正確なつづりに自信がなくても、`lib` さえわかれば、補完で何とか出来ます。

また、一旦、補完メニューを呼び出すと、連続した文字列でなくても絞り込みが効くようになります。例えば、`read.csv()` 関数という、csv ファイルを読み込む関数があります。この関数を補完する際、まず、`r` と一文字書いた時点で、tab キーで補完メニューを呼び出します。この状態では当然、沢山の補完候補が表示されます。しかし、この状態で、続けて `csv`、すなわち、`r``csv` となるように入力を行って絞り込んでみてください。このよう



```
Console Terminal Jobs  
~/  
> date(  
+ )  
[1] "Sun Jul 07 11:03:06 2019"  
>
```

図2.3 複数行入力

な補完検索のコツは、一旦、メニューを開くところにあります。プロンプトに対して、メニューを開かず、いきなり、`rcsv` と入力してもうまくいきません。

さて、補完対象となるのは、関数名をはじめ、変数名、このあと紹介するファイルパス、ライブラリなどがあります。また、RStudio では、プロンプト上に限らず、ダイアログ内での入力でも補完が効くこともあります。とりあえず、よくわからないときは `Tab` キーで補完の問い合わせをしてみましょう。プログラムでは、コードが一文字間違っても動きません。コードの正確性をあげてケアレスミスを防止するため、そして、精神的ストレスを軽減するため補完機能や、履歴機能を活用していきましょう。

2.4 複数行入力

ここまで、コンソール上では、コマンド履歴の呼び出しや補完機能が働く事を紹介しました。更に、コンソール上でのコマンド入力は複数行に分けて行うことができます。

2.4.1 複数行入力と第2プロンプト “+”

例えば、試しに、`date()` とプロンプトに入力してください。この時、通常、補完機能が働いて自動的に `date()` と括弧を閉じて、自動的にその括弧の間に、カーソルが来ますが、その状態からカーソルを動かして右側の括弧を削除して `date()` という状態にします。それから、`Enter` キーを押します。プロンプトがあった次の行の左端に `+` 文字が表示されるはずです。これは、第2プロンプトと呼ばれるもので、「まだ続きのコマンド入力を受け付けていますよ」という事を示すプロンプト記号です。これは加算演算子では無いことに注意しなければなりません。では、ここで閉じ括弧 `)` を入力して `Enter` キーを押してください。`date()` コマンドが実行されます。(図2.3)

コンソール上では、`Enter` キーが押された時に、現在の行内で括弧が閉じられていない時、または、行末に演算子があり、右辺の入力がなされていない時には、コマンド入力が完成していないものと判断し、エラーではなく、次の行に第2プロンプトを出して、さらなる入力を促してくれます。例えば、関数の引数が沢山あったり、演算子を何個もつないだ長い計算式の場合、1行がとて長くなって意味がわかりにくくなりますし、編集作業も大

変になります。このような場合に、コンソールの複数行入力機能は、便利に使えるように思えます。

しかし、残念ながら実際に使ってみると、コマンド履歴が1行毎であったり、前の行にもどって編集作業が出来ない等、それほど便利ではありません。ですから、実際にはこの機能を積極的に使う場面はないとおもいますが、次の項で述べる、複数行入力のキャンセルの仕方は必ず覚えておきましょう。

2.4.2 複数行入力のキャンセル

複数行入力のキャンセルは **Esc** キーで出来ます。途中まで入力したコマンドをすべて放棄して、元のプロンプトに戻ることが出来ます。自分で、意図的に複数行入力を行った場合は、問題ないのですが、初心者のうちは、複雑な式を入力している際に誤って括弧の対が不ぞろいになっている状態で Enter キーを押してもコマンドが実行されず、意図していない複数行入力の状態になってしまい、パニックになることがあります。そして、このように複数行入力になってしまったら、コマンド入力が不完全であるうちは、いくら Enter キーを押してもコマンドが実行されず、次の行に + が表示されるだけでその状態から抜け出せなくなることがあります。

ですから、皆さんはまず、複数行入力というものが存在することを認識し、間違ってもその状態になってしまったら、Esc キーで抜け出すことが出来るということを必ず覚えておきましょう。

2.5 変数と代入

プログラミングをしたことが無い人ならば、まずは単純に、変数とは名前の付いたデータの入れ物だと考えればOKです。

a という変数に、10 という数を入れるならば、次のように書きます。

```
# 変数に値を代入する
```

```
a <- 10
```

<-の部分は、変数にデータを代入する代入記号です。< (小なり記号) と、- (ハイフン) の2文字を使って矢印のような形を書き込むこととなりますが、RStudio 上では、Alt キーと-ハイフンキーを同時押しすれば、一発で書き込むことが出来ます。

変数への代入を行うと、その代入された値はコンピューターの中に保持されます。そして、RStudio は変数とその値についての情報を右上ペインの Environment タブにまとめて表示してくれます。(図2.4)

変数は、この保持する機能をもっているということを意識しましょう。先ほど、1 から 6 までの数字の合計値を計算しました。しかし、実は、計算をして結果を表示させただけであり、その結果を改めて他の計算に使おうと思っても既にそのデータはパソコン上にあり

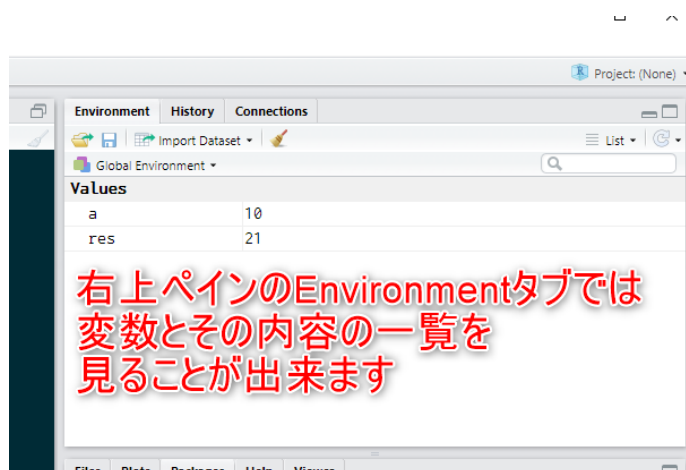


図2.4 Environment タブ

ません。そこで、何らかの計算を行い、その計算結果を後から別の場所で使いたい場合には、変数を使ってデータを保持しておく必要があるのです。

計算結果を代入する

```
res <- sum(1, 2, 3, 4, 5, 6)
```

この変数 `res` の内容は、右上ペインの `Environment` タブに表示されている変数一覧で確認できるはずです。

では、1 から 6 までの数字が合計されているので、それを使って、1 から 6 までの数字の平均を求めてみましょう。合計された数字をその個数 6 で割ればよいので、次のようになります。

計算結果を使って更に計算する

```
res / 6
```

```
## [1] 3.5
```

変数の性質については、これからたくさん学ぶべきものが出てきます。しかし、まずは、代入記号を使って値を代入して、その値を保持させ、これを後から使うことが出来るという点を把握しましょう。

2.6 ベクトル

ベクトルという単語を聞くと、高校数学で習った「方向と大きさを持った量」を思い浮かべるかもしれませんが、**R** というベクトルは、単なるデータの集合であり、高校数学のベクトルとは異なります。もう一つ、ベクトルの発音で悩む必要はありません。日本語ではベクトルで大丈夫です。

プログラミングの世界では、ある一連のデータを扱うことが非常に多いです。このため、Rだけでなくどんなプログラミング言語でも、データの集まりを扱うための仕組みがあり

ます。多くは、配列やリストという名前で、それらの仕組みは呼ばれています。

まずは、プログラミングの世界で扱う一連のデータとは、どんなものなのかを考えてみましょう。例えば、ここで、ある人達にテストを受けてもらい、その結果が次のようになっているとします。

名前	点数
レナード	99
シェルドン	100
ハワード	65
ラジェツシュ	60
ペニー	3

このテストについての、最高点、最低点、平均点、中央値、標準偏差等を知るためには、テスト結果の一つ一つの点数が個別を入手できても計算はできません。一連の点数が全てそろった、まとまりとしてのデータが必要です。このひとまとまりのデータを扱うために、R ではベクトルを使います。

2.6.1 ベクトルの作成

では、実際にベクトルを作りましょう。ベクトルの作成には、`c()` 関数を使います。`c()` 関数の `c` は Combine (結合) の `c` です。`c()` 関数は、`,` カンマを使って複数の引数を渡し、ベクトルを作成します。ここでは、上述のテストの一連の点数結果を複数の引数として `c()` に渡して下さい。

```
# c() 関数を使って一連のデータからベクトルを作成する
c(99, 100, 65, 60, 3)
```

```
## [1] 99 100 65 60 3
```

実行すると、すぐに結果としてベクトルが表示されます。その単純な数字の並び、それがベクトルです。

ここで、先に説明を行った変数の事を思い出して下さい。今、作成したベクトルは、変数に代入していないので、コンピューター上に保持されていません。

ここで、練習問題です。このテスト結果のベクトルを `results` という変数に代入し、このデータを保持して下さい。

```
# ベクトルを変数に代入してデータを保持する
results <- c(99, 100, 65, 60, 3)
```

チェックしてみましょう。

- 履歴機能は使えましたか？

- <-を上手に入力できましたか？

2.6.2 データの型とベクトルの性質

右上ペインの Environment タブを見てみましょう。そこに、`results` 変数が加わっているはずですが、変数の内容は、先ほどの単純な数字の 10 や 21 と異なり、次の様になっているはずです。

```
num [1:5] 99 100 65 60 3
```

データの型

この情報は、3つの部分から構成されています。はじめの `num` は、データの型を表している部分です。`num` は、は `number` すなわち、数値の略であり、このベクトルのデータの型が数値型であることを表しています。

プログラミングを行う場合、その扱うデータには型というものが必ずあります。数値型以外の典型的な型として、文字列型があります。

データ型の違いは、データの性質の違いを表しています。ですから、データの型によって、出来る処理も異なります。例えば、数値型のデータを使って、平均値等の計算ができますが、文字列型のデータの平均値の計算は出来ません（意味論的にナンセンスという意味）。

ここで、この型を意識して、データを作成してみます。

まずは、数値型のデータを作製して、変数に代入してみます。また、同時に `str()` 関数を使って、作成したデータの型を確認します。`str` は `structure`（構造）の略で、引数に与えたデータの構造を表示してくれる関数です。

数値型データの作成

```
n <- 123
str(n)
```

```
## num 123
```

何かのコマンドを使用するわけでは無く、式の中で単純に数字を書けば、それが数値型のデータとして作成されます。

次は、文字列型のデータを作製して、変数に代入してみます。

文字列型データの作成

```
s <- "Hello World!"
str(s)
```

```
## chr "Hello World!"
```


文字列を二重引用符で囲むことで、文字列型のデータを作成できます。str() 関数の出力には、このデータの型が chr である書かれていますが、これは character (文字) の略です。

ベクトルの型

データに型があることがわかりました。そして、実は複数データをとるベクトルにはこのデータの型について大きな約束事があります。その約束事は、ベクトルを構成するデータの型は全て同じでなければならないということです。ですから、データの型といってもいいですし、ベクトルの型といっても意味が通じます。また、右上ペインのベクトルの内容表示に型は、データの型でありベクトルの型なのです。

2.6.3 ベクトルと要素

Environment タブの表示には、型の次に [1:5] という表示があります。この表示は、このベクトルには、1 番から 5 番までのデータがありますということを示しています。つまり、この表示により、ベクトルに入っているデータの数を把握することが出来ます。

ベクトルは、ベクトルを構成する個別のデータは順序をもつという性質があります。ですから、ベクトルの何番目のデータという表現を使うことで、ベクトルを構成する個別のデータを一意に取り出すことが可能になっています。

このベクトルの要素を取り出すためには [] 角括弧演算子を使います。例えば、変数 results に入っているベクトルの 1 番目の要素を取り出すには次のように書きます。

```
# ベクトルにアクセスする [] 演算子
results[1]
```

```
## [1] 99
```

変数名の後ろに [] 角括弧のペアを書き、その中に何番目のデータにアクセスしたいのか、その数値を書き込みます。

2.6.4 ベクトルを使って計算

では、テストの点数がベクトルにまとまったので、テストに関する平均を計算しましょう。平均の計算には mean() 関数を使います。この mean() 関数の引数として、点数の入ったベクトルを渡す事で、平均の計算が出来ます。

```
# 平均を計算する
mean(results)
```

```
## [1] 65.4
```

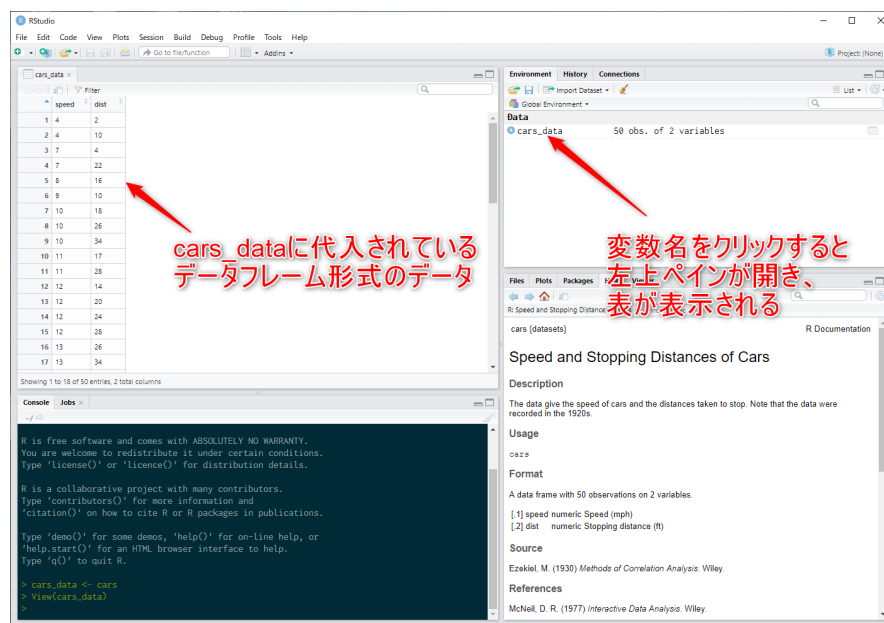


図2.5 dataframe の表示

2.7 データフレーム

ベクトルは、順序のある一連のデータの集まりで1次元的数据構造でした。これに対して、データフレームとは、2次元的なデータ構造をした、表形式のデータの集まりです。これは、エクセルの表のようなものと認識してもらえれば結構です。

Rの大きな特徴の一つが、このデータフレームを簡単に処理できるところにあります。

Rには、沢山のサンプルデータがデータフレームの形で付随しています。このサンプルの一つである `cars` データを見てみましょう。`cars` データは、1920年代の自動車のスピードと停車できるまでの距離の関係の実験結果です。

コンソール上でプロンプトから `cars` と呼び出せば、データが表示されます。しかし、ここでは、`cars_data` という変数を作ってそこに代入し、このデータを保持しましょう。

データフレームのサンプルデータ cars

```
cars_data <- cars
```

2.7.1 データを View() でみる

右上ペインの Environment タブに `cars_data` が表示されます。ここで、その Environment タブで `cars_data` と書かれている部分をクリックして下さい。そうすると、左上ペインに表形式のデータフレームが表示されます。(図2.5)

行

列

Row

Col

横向きの行を表している

縦向きの列を表している

横向きの隙間が横の行を表している

縦向きの隙間が縦の列を表している

図2.6 行と列の覚え方

コンソールを見てみると `View(cars_data)` というコマンドが書かれているのを見つけることが出来ます。実は、変数名をクリックすることで、RStudio が自動的にフレームデータを見るためのコマンドを実行してくれているのです。`View()` 関数は、RStudio でデータを見るための関数です。また、この関数は、Environment タブの右端にある表を表すアイコンをクリックして呼び出すことも出来ます。

2.7.2 行と列

表形式のデータを扱うには、まず、行と列、その英語の省略形である `row` と `col` を正確に暗記しましょう。覚え方は図2.6

R のデータ分析で扱う表形式のデータは整然データと呼ばれるルールに従ったデータになっています。整然データでは、行と列はその役割が決まっており見やすさのために行と列を入れ替えたりすることは在りません。

`cars` データを例に見ると、行はそれぞれの観測した車を表しており、列は個々の観測で得られたスピードと距離という個々のデータの内容を表しています。ですから、データフレームでは、行数がデータの件数、列数が各観測で観測したデータの種類の数を表します。

右上ペインの Environment タブに表示されている情報は次のようなものでした。

```
cars_data    50 obs. of 2 variables
```

50 obs. は、50 件のデータであり、50 行あること。2 variables は、2 列のデータの種類があり、すなわち、2 列であることを示しています。つまり、この表示を確認することでデータフレームの大きさを把握することが出来ます。

2.7.3 整然データとキーワード

variables は、変数という意味の英語です。obs. は、observation の略で、観測という意味の英語です。これらは、整然データについて勉強するとよく出てくるキーワードです。

行列	対応するキーワード
行	observation 観測
列	variable 変数

データ分析を始める段階においては、整然データの定義を精密に理解までしなくても、キーワードとして耳慣れておけば十分です。

2.7.4 列の名前と列へのアクセス

データフレームの列については、通常、その列データがなんであるかを示す列名が付けられています。左上ペインに表示されている `View(cars_data)` の結果を見てください。列の一番上に、speed と dist と書かれているのを見つける事が出来るはずですよ。(図2.5) これが、それぞれの列の列名です。今度は、`str()` 関数を使って、cars_data の構造を確認してみましょう。

```
# データフレーム cars_data の構造を確認する
str(cars_data)
```

```
## 'data.frame':    50 obs. of  2 variables:
## $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
## $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

1 行目で、このデータが `data.frame` であることとその行と列の大きさの情報が示されます。2 行目以降は、各列についての情報が順に表示されています。各列の情報は、\$ 記号の後ろに、列名が表示され、その後ろにデータの型と具体的な内容の一部が表示されます。すなわち、この方法でも、データフレームの列名を確認することが出来ます。

この cars_data は、先に述べた通り、自動車のスピードと停車距離についての実験結果を表しています。そして、今、確認した通り、スピードは speed 列に、停車距離は dist 列にそれぞれ数値として入っています。

ここで、この実験が行われた際の平均スピードを求めたい場合、データフレーム cars_data から、speed の列データだけを取り出す必要が生じます。これを行う一つの方法が \$ 演算子です。

```
# cars_data から speed 列を取り出す
```

```
cars_data$speed
```

```
## [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14
## [24] 15 15 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24
## [47] 24 24 24 25
```

データフレームの後ろに `$` 記号を書き、その後ろに列名を書きます。50 個の数値が表示されていますが、取り出された結果が何かを改めて `str()` 関数で確認しましょう。

```
# 取り出されたデータの構造を確認する
```

```
str(cars_data$speed)
```

```
## num [1:50] 4 4 7 7 8 9 10 10 10 11 ...
```

型名と要素の大きさが表示されました。これは、先に勉強したベクトルです。これで、データフレームの列をベクトルとして取り出すことが出来るようになりました。

練習問題として、`cars_data` で示される実験での、平均スピードを求めてみましょう。

```
# cars_data で示される実験での平均スピード
```

```
mean(cars_data$speed)
```

```
## [1] 15.4
```

では、平均停車距離はどうでしょうか？ 停車距離データが入っている列の列名が何かということ把握しましょう。

```
# cars_data で示される実験での平均停車距離
```

```
mean(cars_data$dist)
```

```
## [1] 42.98
```


第 3 章

パス

3.1 ファイルを扱うために必要不可欠な知識

データ分析の対象となる多くのデータは、**CSV** ファイルと呼ばれる形式のファイルとして配布されています。そして、RStudio でデータ分析を行うためには、必ず、この CSV ファイルのような分析の対象となるデータをもっているファイルを **R** に読み込む必要があります。また、R を使ったより複雑な処理を行う時、その処理をファイルに記述して保存します。これら保存された処理は、他の処理からそのファイルを読み込むことで利用したりします。このように、R の処理を行う場合に、ファイルを扱う作業は必須となります。

ここで、普段使っている Windows の環境では、何かのファイルを指定する時、画面を見て目の前にあるファイルのアイコンにマウスカーソルを持って行き、それをクリックするだけで済みます。もしくは、何かのアプリのメニューからファイル選択用のダイアログを呼び出し、それを使ってファイルを選択します。しかし、コンソール上での作業では、アイコンやメニューやダイアログはありません。ファイルの指定もコマンドとして文字を書く事でその作業を行っていきます。この時、パソコン上にあるファイルを特定するために必要となるのがパスです。

3.2 Windows のディレクトリ構成

Windows 上のファイルは、ドライブの中にあるディレクトリの中、若しくは、ディレクトリの中のディレクトリ（又は、そのディレクトリの中のディレクトリの中の、、、）に存在しています。尚、ディレクトリは Windows ではフォルダと呼ばれることもあります。普段、デスクトップや自分のドキュメントフォルダ等へデータを保存していると思いますが、これらの場所も全て、ドライブから始まるディレクトリ構造の一部になっています。

Windows のエクスプローラーを使って実際に見てみましょう。PC の中にある C ドライブをエクスプローラーで開いて下さい。（図3.1）

もちろん各人によって内容は異なりますが、最低限、次の様なフォルダが見られるはずです。

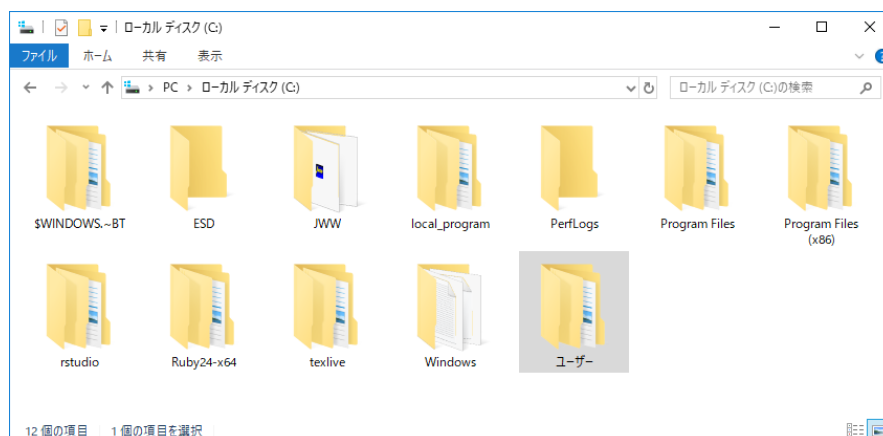


図3.1 Cドライブのルートディレクトリ

- Windows
- Program Files
- Users、若しくは、ユーザー

Windows フォルダは、Windows 自体の色々なファイルやフォルダが入っています。また、**Program Files** フォルダの中には、あなたの Windows にインストールされているプログラムの実行ファイル等が入ったフォルダが入っています。そして、**ユーザー**フォルダの中には、各アカウント名のフォルダが在り、その中を覗くと自分がいつもデータを整理しているであろう見慣れたフォルダ名が並んでいると思います。

少し探索してどんなフォルダの中にどんなフォルダが入っているのか、Cドライブからどんな順番であなたのデスクトップやドキュメントフォルダが位置しているのか、Windows のディレクトリ構造の大雑把なイメージを持って下さい。

3.3 パスの書き方のルール

エクスプローラーでディレクトリ構造の探索を行ってみると、例えば私（アカウント名 shunsk）のデスクトップにある sample.txt というファイルは、「Cドライブの中のユーザーディレクトリの中の shunsk ディレクトリの中のデスクトップディレクトリの中の sample.txt」という風に一意に言葉で説明することができるのが分かります。そして、これを次のルールに基づいて表現することで、コンソール上で利用できるファイルパスになります。

- 先頭をドライブ名（C とか D とか）＋コロンではじめる
- ディレクトリを表すためにディレクトリ名を “/”（スラッシュ）で区切る

私のデスクトップ上の sample.txt のファイルパスは次のようになります。

`C:/Users/shunsk/Desktop/sample.txt`

もう少し、別の例も見ましょう。

パス	説明
C:/Users/shunsk/	C ドライブの Users ディレクトリの中の shunsk ディレクトリ
C:/	C ドライブの直下のディレクトリ
D:/data/text/hoge.txt	D ドライブの data ディレクトリの中の text ディレクトリの中の hoge.txt ファイル

3.4 Windows のパスの注意点

ここで、Windows のパスを R で扱う時の注意点を述べておきます。

3.4.1 ディレクトリを表す記号

まず、Windows のシステム上でのパス表記において、ディレクトリを表す記号には円マーク、若しくは、これと同じ文字を意味するバックスラッシュが使われています。しかし、R の文字列においては、バックスラッシュは特別の意味を持つ文字であるため、単独でバックスラッシュの文字を使うことが出来ません。そこで、R のコンソール上では、ディレクトリ区切りを表現するためには、必ず、スラッシュ記号を使います。

3.4.2 ディレクトリ名称

次に、Windows のディレクトリ名は、エクスプローラーで表示されるものと、ファイルシステム上の名称で異なるものがあります。上記の例では、エクスプローラーでユーザーやデスクトップと日本語で表示されていても、システム上のディレクトリ名は、Users や Desktop となっている場合があります。RStudio でパスを入力する時は、補完機能が使えるので真のディレクトリ名をわざわざ調べる必要はありません。補完候補の中には、真のディレクトリ名があるので、そこから選択すればよいだけです。

3.4.3 間違いがちな所

パスの初めは、ドライブ名+コロンの後にまずスラッシュが入りますまた、コンソール上でのパスは文字列として扱うので、必ず、引用符で囲みます。

正誤	パス
正解	“C:/Users/”
間違い	C:Users

3.4.4 パス入力の実習

では、デスクトップに `sample.txt` という空のファイルを作成し、このファイルパスを `sample_txt_path` という変数に代入してください。ファイルパスの入力には必ず、補完機能を使います。

ファイルパス補完のコツは、まず、ドライブのルートディレクトリとして `"C:/"` と書き込み、そのスラッシュの後ろでタブキーを押すとディレクトリ構造に従った補完候補が呼び出せます。

```
# ファイルパスを補完機能で入力
sample_txt_path <- "C:/Users/shunsk/Desktop/sample.txt"
```

3.5 カレントディレクトリ

Windows を使っている場合には、ほとんど意識することがありませんが、実は、パソコンの操作をしている時、操作をしている人は必ずどこかの場所 (ディレクトリ) にいることになっています。そして、その場所をカレントディレクトリ (current directory: 現在のディレクトリ) やワーキングディレクトリと呼びます。

実際に、あなたが今どこにいるのか？あなたのカレントディレクトリを調べてみましょう。カレントディレクトリのパスを出力する `getwd()` 関数を実行してください。

```
# カレントディレクトリを出力するコマンド
getwd()

## [1] "C:/Users/shunsk/Documents/GitHub/r_beginners_guide"
```

出力は、実行した人ごとに異なります。

さて、まずは、この「カレントディレクトリ」という単語を覚えて、「自分のいるディレクトリ」という概念があるということを意識しましょう。また、カレントディレクトリは、コンソール毎にあります。複数の R のコンソールを立ち上げている時には、そのコンソール毎にカレントディレクトリは独立しています。

3.5.1 カレントディレクトリとパスの種類

さて、先に説明したパスの表現は以下の様なものでした。

```
"C:/Users/shunsk/Desktop/sample.txt"
```

このようにドライブ名から表示したパスの書式を絶対パス、完全パス、フルパス等と呼びます。

これに対して、カレントディレクトリからの相対的な位置で表したパスの書式を相対パスといいます。

例えば、カレントディレクトリ内の `csvdata` ディレクトリにある `hoge.txt` というファイルなら、相対パスでの表現の仕方は、`"csvdata/hoge.txt"` となります。

相対パスを表現する文字列の先頭には、ドライブ名やディレクトリを意味するスラッシュがなく、カレントディレクトリ直下にある ファイル名やディレクトリ名そのものから始まります。

表現の仕方をあらためて次にまとめておきます。

絶対パスで `"C:/Users/shunsk/Desktop/sample.txt"` となっているファイルについてのカレントディレクトリと相対パスの関係は次の通りです。

カレントディレクトリ	相対パス
<code>"C:/Users/"</code>	<code>"shunsk/Desktop/sample.txt"</code>
<code>"C:/Users/shunsk/"</code>	<code>"Desktop/sample.txt"</code>
<code>"C:/Users/shunsk/Desktop/"</code>	<code>"sample.txt"</code>

3.6 カレントディレクトリを意識した作業

みなさんも、普段、パソコンを使って何かの作成作業を行っていると思います。その作業では、作業ごとにフォルダを作成し、その中にワードファイルやエクセルファイル、写真等のイメージファイル、その他のデータファイル等を入れて管理しているのではないのでしょうか？そして、作業をする時はこれらファイルの入っているフォルダを開き、その中にあるファイルアイコンをダブルクリック等して作業をしていることと思います。

R をコンソールで使う場合でも同様です。R の作業をする時に必要となるファイルをフォルダに入れて管理することになります。ただ、R での作業はコンソールで行うので、視覚的にフォルダを開きませんが、代わりに、そのフォルダをカレントディレクトリに設定して、各ファイルへのアクセスを容易にします。

ここから、カレントディレクトリを意識して、実際の R での作業を始めるための準備を体験してもらいます。

3.6.1 デスクトップでの事前準備

まず、デスクトップに `r_work` というフォルダを作ってください。次に、作業対象となる CSV ファイルを用意します。適切な整然データの csv ファイルの例として、次の web ページから国土交通省が発表している不動産取引価格情報データの CSV ファイルを入手できます。適当な csv ファイルを入手してください。（手元に適切なデータがある場合は、そ

れを使っても結構です。)

`http://www.land.mlit.go.jp/webland/download.html`

準備した csv ファイルを `r_work` ディレクトリの中にコピーし、コピーしたファイル名は `my_data.csv` というファイル名に変更しておいてください。

3.6.2 カレントディレクトリの変更

デスクトップ上での準備が完了したら、RStudio のコンソールのカレントディレクトリを今、作成した `r_work` フォルダに変更する作業を行います。

カレントディレクトリの変更は、`setwd()` 関数を使います。`setwd()` 関数には、カレントディレクトリにしたいディレクトリのパスの文字列を引数として渡して使います。各自のデスクトップへのパスはアカウント名が異なるため下の例とは異なるので、コピーアンドペーストでは失敗します。必ず、補完機能を使って各自のデスクトップ上の `r_work` ディレクトリへのパスを入力しましょう。

```
# デスクトップの r_work ディレクトリをカレントディレクトリにする。  
setwd("c:/Users/shunsk/Desktop/r_work/")
```

3.6.3 ファイルの読み込み作業

整然データとしての csv ファイルを読み込む関数の一つに `read.csv()` 関数があります。この関数は、読み込みたいファイルのファイルパスを引数として取ります。さて、この引数として渡すファイルパスは、上記で勉強した通り、絶対パスと相対パスの両方で書けます。

まず、相対パスでデータを読み込んでみましょう。読み込んだデータを保持するために、`data_a` という変数に代入してください。

```
# 相対パスを使ってファイルを読み込む  
data_a <- read.csv("my_data.csv")
```

このように、カレントディレクトリを適切に設定するとファイル読み込みを指示するコマンドのタイピング量は減り、作業が便利になるように思えます。

では、作業をする時には、いつでも相対パスで指示すればよいのでしょうか？

3.6.4 相対パスと絶対パス

相対パスを使うには、前述の通り前提条件が必要です。上記の作業では、ファイルを読み込む前に、カレントディレクトリの変更を行い、カレントディレクトリが `my_data.csv` を入れている `r_work` になっていることが 確実な状態です。

この様に、カレントディレクトリと対象となるファイルの 相対的位置関係をきちんと把握できている場合、相対パスでファイルを正しく指定することが出来ます。逆を言えば、カレントディレクトリが分からなければ、相対パスでパスの指定をすることは出来ません。

次に、絶対パスでデータを読み込んでみましょう。今度は、`data_b` という変数に代入します。

絶対パスを使ってファイルを読み込む

```
data_b <- read.csv("c:/Users/shunsk/Desktop/r_work/my_data.csv")
```

絶対パスは、今使っているパソコン上のファイルの場所を一意に表しています。ですから、カレントディレクトリが何処であったとしても、正しくファイルの場所を指定することが出来ます。

上記の様にファイルの読み込み作業において、絶対パスと相対パスの両方のコードで実践してみました。この両パス表記は、どちらかが優れているというものではなく、それぞれ、その性質に応じた使い方をします。そして、どちらの表記であっても、自分で読み書きできる必要があります。

まず、絶対パスを使う時は、現在のカレントディレクトリを気にせず、ダイレクトにパスを指定したい場合です。例えば、カレントディレクトリのすぐ近くのディレクトリにカレントディレクトリを移動させたい場合には、相対的な関係を直感的に把握できますが、構造的に離れたディレクトリへ移動する時には、ディレクトリルートから辿る方が簡単です。

次に、絶対パスが使えない場合があることを把握しましょう。それは、絶対パスの入ったコードは他人の環境で動かないという事です。前述しましたが、私のデスクトップ上のファイルへの絶対パスは、あなたのデスクトップ上のファイルへの絶対パスになりません。ですから、絶対パスを使ったコードをあなたがコピー&ペーストして実行してもうまく動きません。

一方、他人とコードを共有する場合、若しくは、自分のパソコンの中であっても、そのコードの置き場所を何処にでも移動できるようにする場合、パス表記全てをベースとなるディレクトリからの相対表記にしておき、作業の時に、カレントディレクトリをそのベースディレクトリに設定することで、ベースディレクトリの絶対パスが何処であっても、相対パスの表記は文字通り同じになります。つまり、あなたが、適切なカレントディレクトリさえ設定すれば、私の相対パスで書かれたコードをコピー&ペーストするだけで、あなたのパソコンでもそのコードが動くようになります。

以上のパスに関する基礎的な知識を把握していれば、R の教科書でのファイルの読み込み作業で失敗することは無くなります。

3.7 RStudio のプロジェクト

先ほどは、コマンドを使ってカレントディレクトリを変更することを覚えましたが、R で作業をする毎にわざわざカレントディレクトリを設定するのは面倒です。そこで、RStudio

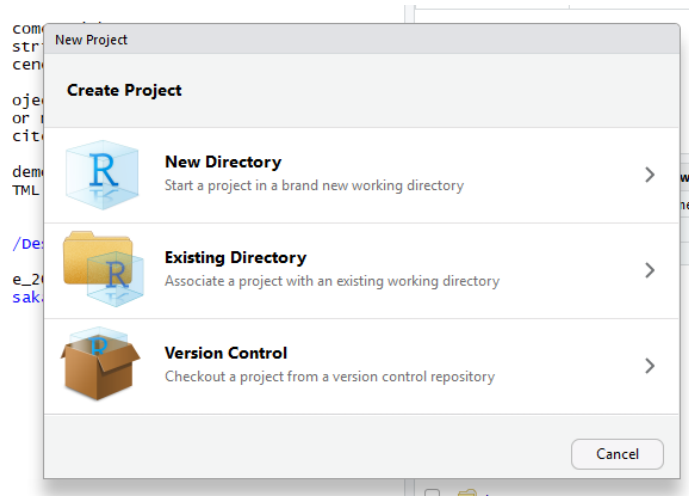


図3.2 プロジェクト作成ダイアログ

は、このような煩雑な作業をしなくてすむように、プロジェクトという機能を備えています。

RStudio では、作業ディレクトリ毎に RStudio のプロジェクトファイルを作成しておきます。そうすることで、そのプロジェクトファイルを開くだけで、そのディレクトリがカレントディレクトリであるコンソールが開くようになります。また、別のプロジェクトファイルを開くと、その別の作業用のディレクトリをカレントディレクトリとする別のコンソールが開きます。このように、RStudio では、作業ディレクトリ毎のコンソールを別々に開いて複数の作業を同時に行う事が簡単にできます。

3.7.1 プロジェクトの作成

では、早速プロジェクトを作成しましょう。RStudio を立ち上げ、メニューの File から New Project を選択します。^{*1} そうすると、プロジェクト作成ダイアログが表示されます。(図3.2)

ここでは、選択肢が3つありますが、今回は、一番手軽にプロジェクトを作成できる2番目の方法「既に存在する作業フォルダをプロジェクトにする (Existing Directory)」を紹介します。この2番の方法さえ覚えておけば、新規にフォルダを作ってプロジェクトにする時も、自分で新しいフォルダを先に作りさえすれば、この2番の方法を使って同じようにプロジェクトを作成することが出来ます。

では、ダイアログの Existing Directory の部分をクリックして下さい。ダイアログが切り替わるので、ディレクトリ選択ダイアログ (Browse ボタンから) 等を使って、プロジェクトにしたい既存のディレクトリのパスを指定します。ここでは、先に作成したデ

^{*1} 既にコンソールで何かの作業をしていると、作業中の変数等の状況を保存しておくかどうかの確認ダイアログが出ますが、特に必要が無い限り作業は保存しませんので、Don't Save ボタンをクリックすればOKです。

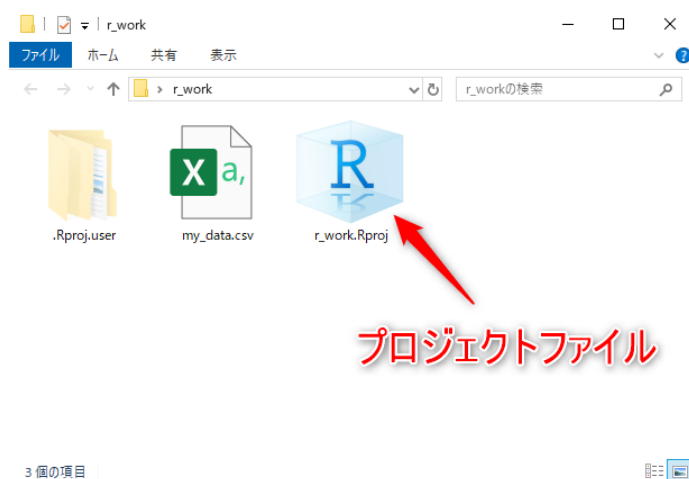


図3.3 プロジェクトファイル

デスクトップ上にある `r_work` のパスを指定して下さい。パスの指定が出来たら、ダイアログ右下の **Create Project** ボタンをクリックするとプロジェクトの作成作業は完了です。そのまま、デスクトップの `r_work` ディレクトリをカレントディレクトリとした RStudio のコンソールが開きますが、ここで、一旦、RStudio を終了しましょう。右上の `×` ボタン等を使って終了して下さい。

さて、今まで RStudio の起動は、Windows のスタートメニューから行っていたと思います。しかし、これからの作業は RStudio のプロジェクトファイルから起動することが出来ます。デスクトップの `r_work` フォルダを開いて下さい。ファイル名の拡張子が `.Rproj` となっていて、R の文字がついたアイコンのファイルが新たに作られているはずです。これが、RStudio のプロジェクトファイルです。(図3.3)

このディレクトリでの作業を開始する時には、このプロジェクトファイルをダブルクリックする事で、そのディレクトリをカレントディレクトリとしたコンソールを持つ RStudio を起動することが出来ます。

3.7.2 RStudio に小さく表示される情報

RStudio はプロジェクトごとに立ち上げることが出来ると述べましたが、複数の RStudio が立ち上がると、どの RStudio がどのプロジェクトが見わけがつかなくなるかもしれません。

そんな時、右上ペインのその右上をよく見て下さい。その他、カレントディレクトリを知るために `getwd()` 関数を使っていたと思いますが、コンソールの上部、Console とタブに書かれているところのすぐ下をよく見て下さい。RStudio には、小さなところにいろんな情報が隠れています。

第 4 章

パッケージ

R のパッケージとは、関数やデータセットを集めたものです。R では、パッケージをインストールすることで、機能を拡張することが出来る仕組みになっています。R の参考書等ではよく、その説明の中で使われる関数を含んでいるパッケージのインストールが要求されます。

このパッケージは、世界各国の現場の研究者達や RStudio チームをはじめとするオープンソース開発者達が独自に開発して公開しています。このパッケージには、**CRAN** を通じて公式に配布されているものと、**GitHub** 等を通じてプログラマが独自に配布しているものがあります。

まず、CRAN は、The Comprehensive R Archive Network（包括的 R アーカイブネットワーク）の略称で R に関するコードとドキュメントを配布しているサーバー群です。（図4.1）R の本体もこの CRAN で配布されています。CRAN は、シーランやクランと発音されているようです。CRAN は世界各国にミラーサーバーがあり、それらが常に同期されています。

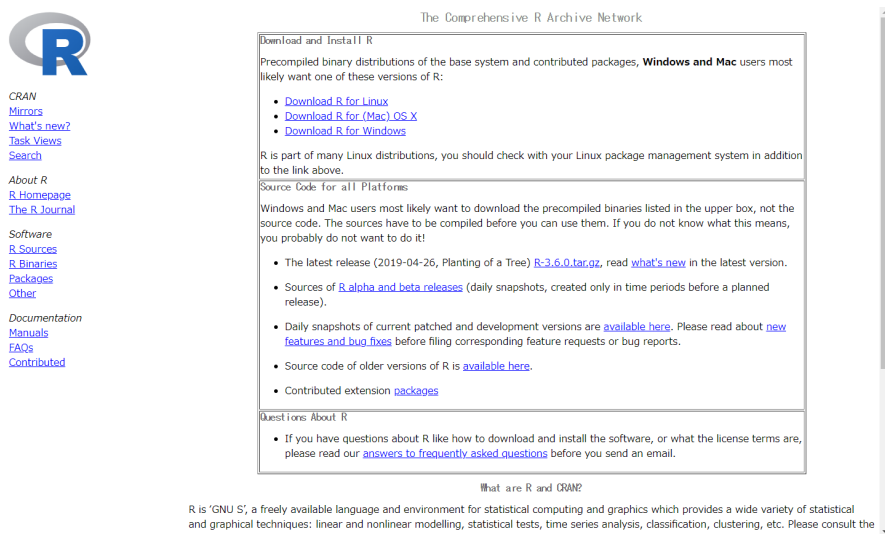
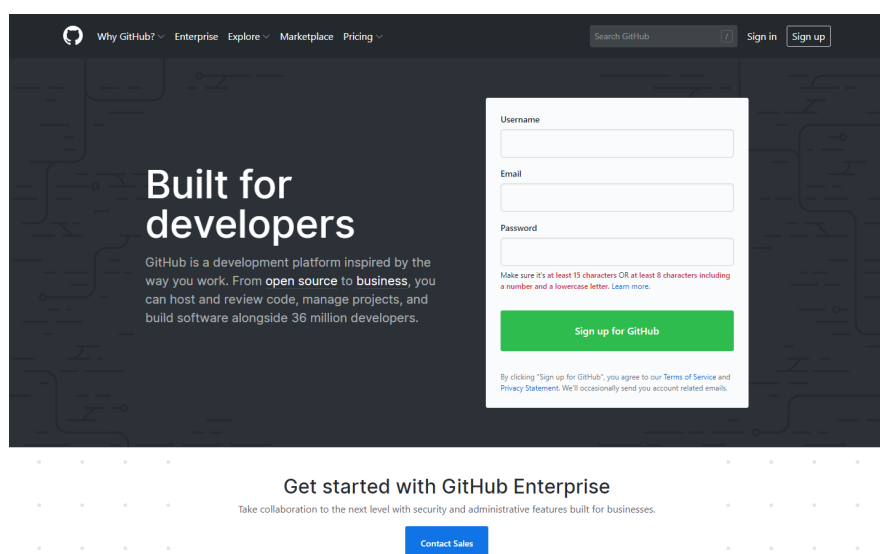


図4.1 <https://cran.r-project.org/>

図4.2 <https://github.com/>

次に、GitHub は、世界中のプログラマが沢山のプログラムのソースコードを公開している有名な Web サービスです。(図4.2) プログラムの開発をサポートするサービスなので、CRAN に登録されている R のパッケージも、その開発段階の最新のものは GitHub 上にもあったりします。

CRAN に登録されているパッケージは、2019/5/31 時点で 14,307 あります。しかし、R のパッケージは CRAN に登録されずに GitHub 上のみで公開されているものもあるので、世の中にある R パッケージは上記の数よりもずっと多く存在します。R では、これらの全てのパッケージをインストールしなければならないわけではなく、これらのうちから自分の作業に必要なもののみをインストールします。一般的に、R で新しいことをはじめようとする場合、参考書等でほぼ初めに述べられるのが、その作業に必要なパッケージのインストールについてです。また、R のバージョンアップを行ったり、再インストールを行った場合には、いつも使っていたパッケージについて、再度インストールする必要があります。

以上のことから、R 言語を利用する場合、パッケージの管理は必須の作業になります。

4.1 RStudio でのパッケージのインストール

R には、パッケージをインストールするための **R** の関数が用意されています。ここで上記の通り R のパッケージの配布先は主に CRAN と GitHub の 2 つがありますが、この CRAN にあるパッケージと GitHub にあるパッケージで インストールするための関数が異なることに注意が必要です。

ダウンロード先	インストール用関数
CRAN	<code>install.packages()</code>

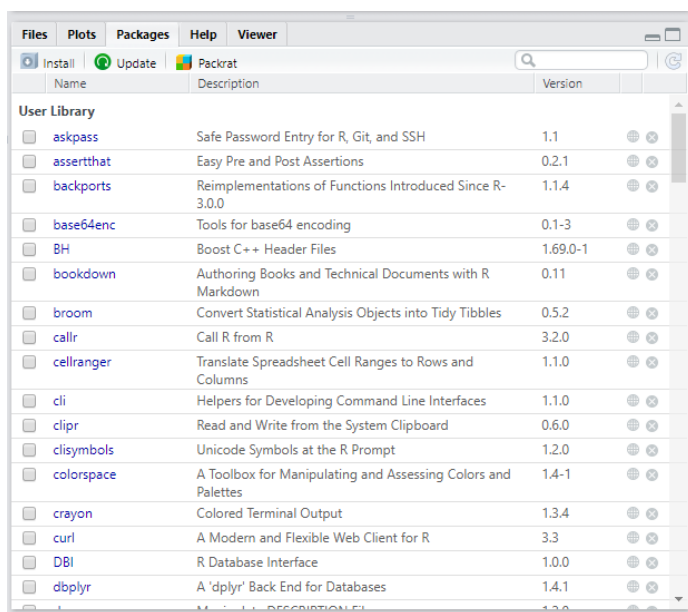


図4.3 RStudio 右下ペインの Packages タブ

ダウンロード先 インストール用関数

GitHub install_github()

このように、R には本来的にインストール用の関数が用意されていますが、日常の R 作業においては、**RStudio** の機能を使ってパッケージの管理を行う方が便利です。RStudio の右下ペイン、**Packages** タブを見てください。(図4.3)

一覧には、既にインストールされているパッケージのパッケージ名、概略の説明、バージョンが順に記されています。このタブからパッケージに関する色々な操作が出来ますが、まずは、パッケージのインストールの仕方を覚えましょう。但し、この方法でインストールできるのは、CRAN で配布されているパッケージのみです。また、パッケージのインストールは、RStudio 内部で自動的に CRAN 等からパッケージをダウンロードしてからインストールを行うためインターネットに接続された状態で行う必要があります。

Packages タブのタブの下をよく見ると、Install と Update と書かれた部分がみつかります。(図4.4)

このうち **Install** と書かれている部分をクリックしてください。**Install Packages** ダイアログが出てきます。(図4.5)

このダイアログの Packages(separate multiple with space or comma): と書かれている入力フォームにインストールしたいパッケージ名を入力しダイアログ下部の Install ボタンを押すことでパッケージのインストールが出来ます。

この RStudio でのインストール機能については、まず、この入力フォームでは、パッケージ名の補完機能が働いています。パッケージ名の綴りがうろ覚えであっても、補完メニュー

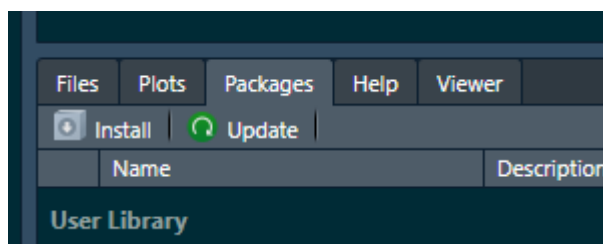


図4.4 RStudio 右下ペインの Packages タブ

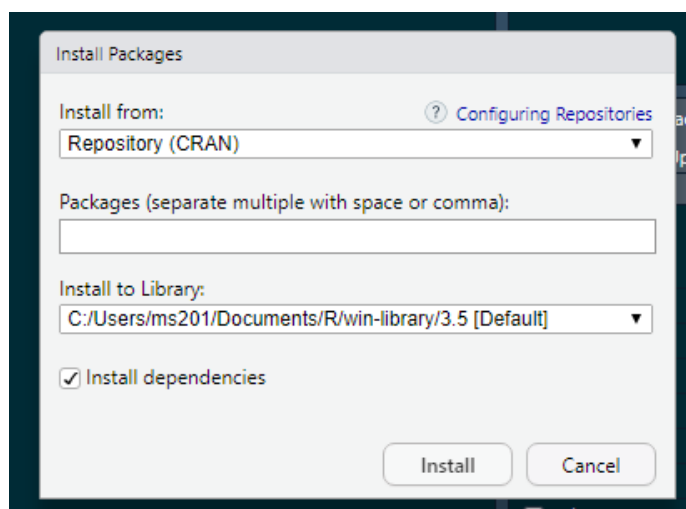


図4.5 Install Packages ダイアログ

ーが支援してくれるので便利です。次に、入力フォームには、スペースもしくはカンマ区切りを使うことで、複数のパッケージ名を書き込んで、一度に複数のパッケージをインストールすることが出来ます。更に、ダイアログの下部にある Install dependencies と書かれている部分にあるチェックボックスにチェックがあることを確認しましょう。パッケージは、それをインストールするためには、先に別のパッケージをインストールしておく必要がある場合があります。（こういう関係は、依存関係と呼ばれる）Install dependencies のチェックを入れておけば、自分がインストールしたいパッケージに必要なパッケージがある場合、依存関係が解決されるように自動的に他のパッケージをインストールしてくれます。

尚、RStudio のインストール機能を使っても実際には、先に説明したパッケージのインストール関数が使われて作業が行われます。Install Packages ダイアログの Install ボタンを押すと、左側ペインの Console に沢山の出力が現れているはずです。先に述べた通り、インストール作業では、自分の指定したパッケージ以外にも依存関係を解決するために沢山のパッケージがインストールされる事があり、この場合、インストール作業が終了するまで数分かかることもあります。インストール作業中なのか、終了したのかは、Console にプロンプトがあるかないかで判断できます。Console 画面をよく見て、作業が終了するのをのんびり待ってください。

また、パッケージのインストール作業は、ネットワーク回線の状態が悪い時などパッケー

ジのダウンロードに失敗して、インストール作業自体が失敗することがあります。「インストールしたはずなのに、」という場合、Packages タブの一覧に、パッケージ名があるかどうか確認してください。Packages タブ上部右端の検索フォームを使うと簡単です。よくわからない場合、再度重複してインストールしたとしても問題ありません。もう一度、目的のパッケージをインストールしてみてください。

4.2 パッケージの管理

再度、右下ペインの Packages タブに着目してください。(図4.3) このパッケージタブでは、インストールだけでなくパッケージについて様々なことが出来ます。

4.2.1 アップデート

一覧上部にある、Update と書かれている部分をクリックすると、Update Packages ダイアログが出てきます。アップデート可能なパッケージがある場合、ここに表示されるので、アップデートするパッケージにチェックをいれて Install Update ボタンをクリックすれば、アップデートをおこなってくれます。

4.2.2 マニュアルをみる

パッケージ一覧のパッケージ名をクリックすることで、右下ペインは Help タブに切り替わり、クリックしたパッケージのマニュアルを表示することが出来ます。

4.2.3 パッケージの読み込みとデタッチ

一覧の左端にあるチェックボックスにチェックをいれると、Colnsole の環境にそのパッケージをロードします。Console に目を移すと、`library()` 関数が呼ばれていることがわかります。逆に、チェックを外すと、現在の Console 環境からそのパッケージを取り除きます。これも Console に目を移すと `detach()` 関数が呼ばれていることを確認できます。

4.2.4 パッケージに関する Web サイトをみる

一覧の右端に小さな丸いアイコンが二つ並んでいます。そのうち、左側の十字マークのアイコンをクリックすると、Web ブラウザが起動し、そのパッケージの開発元の GitHub のリポジトリページや CRAN の Web ページなどを閲覧することが出来ます。

4.2.5 パッケージアンインストール

一覧の右端に小さな丸いアイコンが二つ並んでいるうちの、右側の × マークのアイコンからパッケージをアンインストールすることが出来ます。× マークアイコンをクリックすると、そのパッケージをアンインストールして良いかの確認ダイアログが出ます。アンインストールする場合は、Yes を選択してアンインストールを実行します。但し、基本的でシステムに必須のパッケージは、ここからアンインストールしてもアンインストールが出来ない旨が表示されて、アンインストールは行われません。

4.3 パッケージを読み込む library() 関数

パッケージに含まれている関数は、パッケージをインストールしただけでは使えません。ここで、GitHub で公開されているパッケージをインストールするための関数と、この関数を含む devtools パッケージの関係を例にパッケージの使い方を紹介します。

通常、利用したい関数を含んだパッケージを、上記の手順で R にインストールします。既存パッケージを確認して、devtools パッケージがインストールされていない場合、ここでインストールしてみてください。

そして、パッケージ内の関数を利用する場合、まず、その環境 (console 等) に、library() 関数を使ってパッケージを読み込みます。

```
# devtools パッケージを読み込む
library(devtools)
```

この様にパッケージを読み込むことで、このパッケージに含まれる install_github() 関数がこのコンソール上で使えるようになります。

パッケージのインストールをちゃんとしていても、library() 関数によるパッケージの読み込みを忘れているとそのパッケージに含まれる関数を呼び出しても、その関数を見つけることが出来ない旨のメッセージが添えられたエラーが起こります。忘れずにパッケージの読み込みをしましょう。

4.4 GitHub にあるパッケージのインストール

先に紹介した通り、GitHub で公開されているパッケージは、install_github() 関数を使ってインストールします。そして、この関数を利用するためには、devtools パッケージをインストールした上で、library() 関数を使ってパッケージをコンソールに読み込んで置くことを忘れてはいけません。

install_github() 関数に渡す引数は、インストールしたいパッケージのある github 上

のリポジトリを表す文字列です。

GitHub には、沢山の開発者のアカウントがあり、そのアカウントの中に複数のリポジトリが入っています。リポジトリとは貯蔵庫という意味ですが、単純に開発している物毎にまとめられたディレクトリと考えればOKです。

ここでは、GitHub 上のパッケージの例として **reti** パッケージを例としてインストールします。reti(Real Estate Transaction-price Infomation data) パッケージ^{*1}は、国土交通省が公開している不動産取引価格情報のデータを R で扱いやすくするためのパッケージで、筆者の所属する大阪府不動産鑑定士協会不動産取引価格情報活用小委員会^{*2}が GitHub 上で開発して公開しています。

このパッケージあるリポジトリを GitHub の URL で表すと次の通りになっています。

```
https://github.com/rea-osaka/reti
```

`https://github.com/`の後ろにある **rea-osaka** の部分がアカウント名で、スラッシュを挟んで、その次の **reti** の部分がリポジトリ名です。

リポジトリは各アカウントの人たちが自由に付けたり、他人の物をコピーして持ってきたり出来るのでリポジトリ名は GitHub 全体では重複することが出来ます。但し、各アカウント内では同名のリポジトリは存在することが出来ません。そして、GitHub 上では、アカウント名は重複しません。そこで、GitHub 上でのリポジトリは、アカウント名/リポジトリ名という形式を使って一意に選択することが出来ます。rea-osaka が開発している reti は、**rea-osaka/reti** と表記することが出来ます。

そして、`install_github()` 関数には、このリポジトリを特定する文字列を引数として渡します。

```
# GitHub 上の reti パッケージをインストールする
library(devtools)
install_github("rea-osaka/reti")
```

GitHub 上で公開されているパッケージ等は、GitHub のリポジトリページをみると大体英語で書かれていますが、**Install** という項目があります。そこには、上記で述べた通り、devtools パッケージをインストールしていないなら、先にインストールして下さいということや、それを実際に行ってくれるコードが書かれています。そして、そのパッケージをインストールするための `install_github()` 関数のコードも書かれています。そこで、注意深い人は、次のような表示になっていることに気づくと思います。例えば、reti のページでは次のようなコードが書かれています。

```
devtools::install_github("rea-osaka/reti")
```

`install_github` の前に `devtools::` が付いています。この表記を使うと、関数名が一意に決定できるため `library()` 関数でパッケージを読み込む必要がありません。

^{*1} <https://github.com/rea-osaka/reti>

^{*2} <https://rea-osaka.github.io/>

実は、関数名は、パッケージ間では、重複する可能性があります。そこで、どの関数を使っているのかをはっきりさせるためには、どのパッケージのどの関数という長い表記になります。しかし、実際のプログラム上では、イチイチ長い名前を書くのは見づらくなるので、`library()` 関数を使ってこの環境で同名の関数が複数あるのにパッケージ名が省略されている場合は、`library()` で指定したものをを使うという仕組みが作られています。逆に、正式な関数の名称は、パッケージ名と関数名を`::`で繋いで表すことが出来、その具体例が上記の `devtools::install_github()` という表記なのです。

このような理解があれば、安心して github 上のパッケージのインストールにも挑戦できると思います。

4.5 有名なパッケージ

4.5.1 tidyverse

`tidyverse`^{*3}はデータサイエンス作業で必須となるパッケージ群です。

実は、`tidyverse` はパッケージではなく、パッケージの集まりですが、インストール作業や `library()` 関数での読み込みは、通常のパッケージと同じく `tidyverse` でOKです。このパッケージ群は、CRAN で配布されているので、RStudio の Packages タブからインストールできます。更に、CRAN で配布されているものよりも最新の開発版については、各パッケージについて GitHub でも公開されており、これを `install_github()` 関数を使ってインストールすることも出来ます。

`tidyverse` にはデータサイエンス作業で必須となる以下のパッケージが含まれています。

`ggplot2`

グラフの描画を行うための関数をあつめたパッケージ

`dplyr`

整然データに対して、フィルタリングしたり、並べ替えをしたり、列を抜き出したり、新たな列を作ったり、要約計算をしたりするための関数が集まったパッケージ。ディープレイヤーと読めばOK。プレイヤーはベンチです。

`tidyr`

`tidy` はきちんとしたという意味の英語ですが、データサイエンス上では整然データを差します。そして、`tidyr` は一般データを扱いやすい整然データに加工するための関数があつまったパッケージです。タイディアーと読みめばOKです。

^{*3} <https://www.tidyverse.org/>

readr

R 外部にあるデータファイルを R に読み込むための関数が集まったパッケージです。

purrr

関数型プログラミングを行うための関数セットです。

tibble

data.frame 型のデータを改良したデータ型を使うための関数が集まったパッケージです。

stringr

文字列処理を行うための関数が集まったパッケージです。

forcats

ファクタ型のデータを扱う際の一般的な問題を解決してくれる関数が集まったパッケージです。

4.5.2 devtools

devtools^{*4}は、パッケージ開発を支援するための関数を集めたパッケージです。先に紹介した通り、GitHub からのパッケージのインストールには、このパッケージのインストールは必須です。CRAN で配布されていると共に、GitHub 上でも開発版が公開されています。

4.5.3 leaflet

Leaflet^{*5}という、対話的地図環境を構築するための javascript ライブラリを R から使うための関数を集めたパッケージです。これを使うことで、データをオープンストリートマップ等の上に簡単にプロットできるようになります。CRAN で配布されていると共に、GitHub 上でも開発版が公開されています。

^{*4} <https://devtools.r-lib.org/>

^{*5} <https://rstudio.github.io/leaflet/>

第 5 章

日常作業は、R Notebook で行う

実際の R によるデータ作業では、その分析作業の流れ自体にも意味があり、それらを一つ一つを形として残すことによって作業の意味が分かり易くなります。

いままで、R の作業はコンソールのプロンプトにコマンドを入力し実行することで行うと説明してきました。確かに、一つ一つのコマンド実行と、その結果を見た上で、次のコマンドを何にするか決めて実行するというインタラクティブな操作は行えます。しかし、全体的な流れを俯瞰したり、後からまとめて把握して検討したりということには適していません。また、全体をみて検討するだけならば、一般的に、スクリプトを書く事によって、複数コマンドを作業の流れの通りに記載し、一気に実行する方法もありますが、この場合、インタラクティブな操作ではありません。

そこで、R の様にインタラクティブなデータ作業に適しているのが、**R Notebook** です。このノートというものを聞いたことが無い人がイメージするなら、マイクロソフトのワードの文章を作っていて、計算部分のみエクセルを埋め込んで実行できるようにしてあるという感じのものです。また、その成果を保存したノートファイルは、RStudio や R の処理系が無くても見ることのできる html 形式のようなファイルなので便利に扱えます。

5.1 作業の準備はプロジェクトから

RStudio で作業する場合、作業用のフォルダを用意して、そこにプロジェクトを作る癖を付けましょう。ノート等を扱う場合、作業の成果をファイルとして保存することになりますし、データ分析作業は、取り込み用の CSV ファイルを管理する事も多く、ファイル読み込み等、ファイル間の位置関係に気を配る必要が多くあります。そこで、コード内ではファイルパスとして相対パスを使えるように、関連するファイルを同じディレクトリ内で管理することが重要になります。

では、新しいフォルダを用意してプロジェクトを作り、そのプロジェクトで RStudio を開いてください。

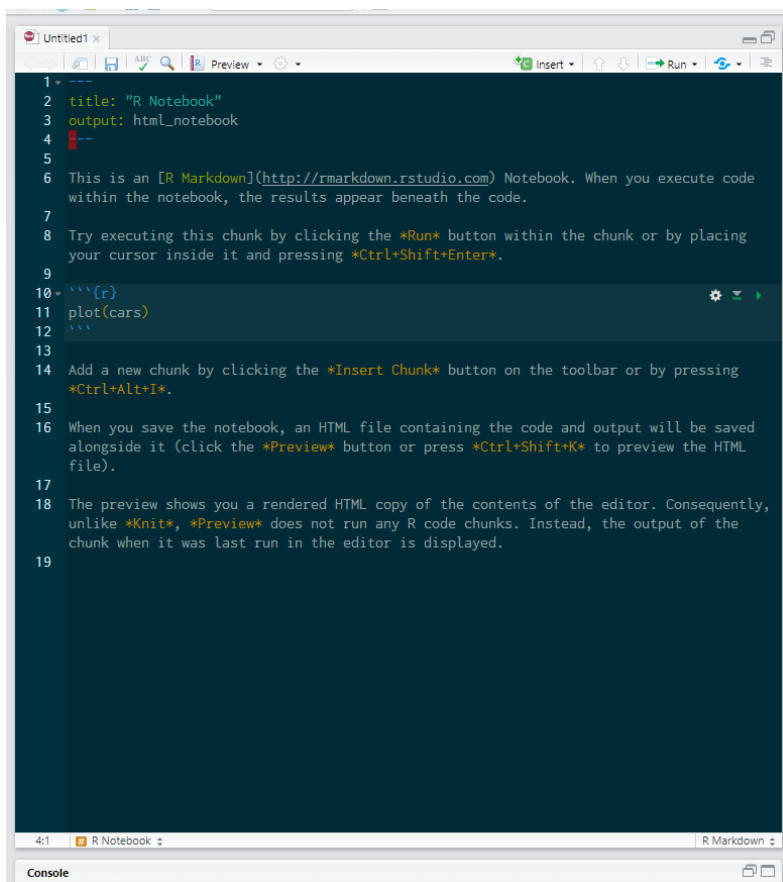


図5.1 R Notebook を新規作成

5.2 R Notebook を新規作成する

RStudio のメニューから、File > New File > R Notebook と辿って選択して下さい。RStudio の左側が一つのペインになったはずです。(図5.1)

但し、よく見ると、一番下に Console のペインが小さく縮んでいるのが分かります。各ペインの右上にある二つのアイコンを操作することによって、各ペインは伸び縮ませることができます。ですから、コンソールが必要になったときは、それらアイコンを使って Console ペインを復元すれば OK です。

この今、新しく開いたペインが R Notebook の編集領域です。この部分はメモ帳と同じでカーソルを好きな所に動かして、なんでも好きな事を書くことができます。

英語がたくさん書かれていますが、この殆どはプログラムのコードではありません。内容は、R Notebook の簡単な使い方の説明ですが、今は、特に気にしなくて結構です。また、ここに書かれている文書は、編集して消してしまっても、全然平気です。R Notebook を新規作成すれば、いつでも同じものが出てきます。



図5.2 チャンク

5.3 R Notebook の構造

R Notebook の説明を理解するためには、まず最低限、R Notebook の構造を覚えておかなければなりません。

5.3.1 設定を行う YAML ヘッダ

ファイルの先頭に注目してください。

```
---  
title: "R Notebook"  
output: html_notebook  
---
```

R Notebook の先頭には必ず、この形式の表記があります。この部分を **YAML ヘッダ** (YAML はヤムルと読みます) と呼びます。YAML は、データ記述方法の一つで、---に囲まれた部分に、あるルールに基づいてデータを記載する方法で、R Notebook では、ここにそのノートのための設定を記載します。しかし、R Notebook を触り始める時には、この部分に何を書くのかを覚えるのではなく、まず、ファイル先頭の---に囲まれた部分が、**YAML ヘッダ**であるということを把握すれば十分です。

5.3.2 コードが実行できる場所、チャンク

真ん中あたりに ```{r} と ``` に挟まれた部分があります。このような部分を **チャンク** と呼びます。チャンクはよく見ると、ほかの部分と少し色が変わっています。また、右端に歯車、下三角、右三角の小さなアイコンがあります。(図5.2)

このチャンクにコマンドを書いて、右端の右三角アイコンをクリックすると、チャンク内のコマンドが実行され、チャンクのすぐ下に結果が出力されます。試しに、右三角アイコンをクリックしてみてください。cars データに関する散布図が描画されます。

チャンクは、ノートの中に幾つでも作れます。チャンクは、```{r} と ``` に挟まれた部分なので、基本は、これを直接自分で書き込むことで作成できますが、キーボードショートカット (Ctrl + Alt + I) やツールバーの **Insert** アイコンをクリックすると出てくるメニューから **R** を選ぶことでカーソル位置に空のチャンクを挿入できます。

5.3.3 それ以外の部分

YAML ヘッダとチャンク以外の部分は、メモ帳に字を書いているのと同じです。この部分に、コードに関する説明や、コードを実行した結果に対するコメントや考え等文章ならなんでも書くことが出来ます。

5.4 R Notebook とマークダウン

R Notebook を新規作成しただけの状態では、編集ペインに表示されているだけで、実体のファイルがありません。編集ペインのタブの部分を見てください。新規作成された R Notebook は、**Untitled1** となっているはずです。この名称は、編集ペインの内容がまだ保存されていなくて、ファイルの実態が無い事を表しています。

そこで、今から、編集ペインの内容を保存して、ファイルの実態を作成しましょう。ツールバーにある、フロッピーディスクのアイコンをクリック、若しくは、キーボードショートカット (Ctrl + S) で、ファイルの保存が出来ます。ファイル保存ダイアログが出るので、保存する場所が、プロジェクトのディレクトリであることを確認して、適当なファイル名、例えば、`my_first_note` と入力します。この時、拡張子は付けなくて結構です。

ファイルを保存すると、まず、タブの部分が `Untitled1` から保存したファイル名、例えば `my_first_note.Rmd` 等に変わっていることが確認できます。

RStudio の右下ペインを Files タブに切り替えて下さい。そこには、プロジェクトディレクトリ内に有るファイルの一覧が表示されていますが、その中に先ほど決定したファイル名に `.Rmd` という拡張子、及び、`.nb.html` という拡張子が付いた2つのファイルを見つけることが出来ます。(図5.3)

今、編集ペインで編集しているファイルは、`.Rmd` の拡張子が付いているファイルです。今まで、「R Notebook を新規作成します」等という風に説明していましたが、実は、この編集領域で書いてきた書式は、R マークダウンと呼ばれるルールに従って書かれている書式です。そして、この R マークダウン形式で書かれているファイルには、`.Rmd` という拡張子が付けられます。

5.4.1 マークダウン

パソコンを使った文書作成を行う時、真っ先にイメージするのは、マイクロソフトのワード等に代表されるワープロソフトです。これらのソフトは、文書作成時に、文章としての内容だけでなく、印刷される時のイメージ、例えば、文章が何処で折り返されるとか、ここの文字の大きさや太字にするかどうか等の文書の体裁を画面上で見た目通りに指定して、作製していきます。

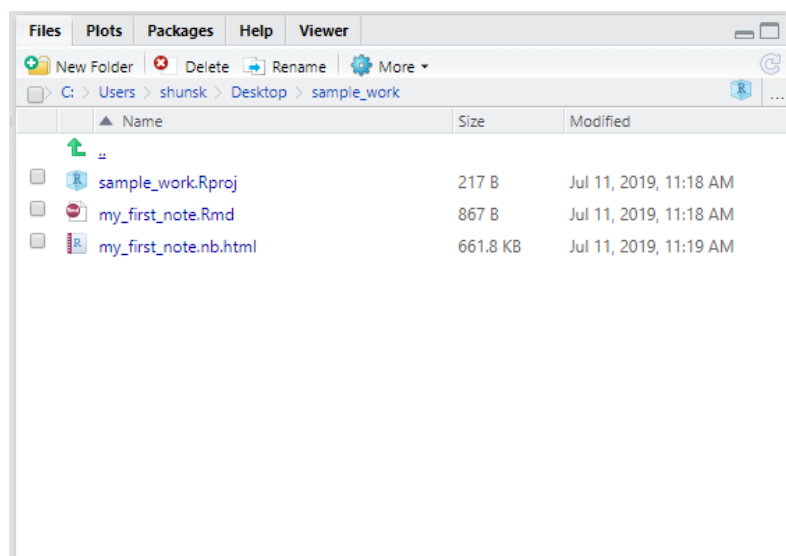


図5.3 右下ペイン Files タブ

しかし、実は、コンピューターを使った文書作成には、別の方法もあります。その一つがマークダウンです。マークダウンでの文書作成では、文書の内容と体裁を切り離して処理します。まず、文書の内容については、タイトル、段落、文、といった構造を意識して文書の内容を記述するだけです。一方、体裁部分は、文書の意味的な内容は関知せず、その構造等のみに基づいて体裁を決めて処理します。

こうすることで、まず、文書の内容を作成する時には、デザインに気を使うことなく、その文書の意味的な内容の作成に集中が出来ます。そして、体裁は、特に気を使わなくても統一的な体裁を適用しやすくなると共に、違う体裁に切り換えれば、文書全体の体裁を一気に変更することが簡単に出来るようになります。

5.4.2 マークダウンの出力ファイル

ここで、マークダウンという言葉が初耳の方にとっては、あまりイメージできないと思いますので、実物を見てみましょう。実は、文書の作成部分というのが、`.Rmd` ファイルの内容です。そして、マークダウンについて体裁を処理したものが、`.nb.html` ファイルです。

このように、マークダウンを使った文書作成では、まず、`.Rmd` ファイルに文書の中身を作製し、これを処理して、体裁の整えられた出力ファイルが別のファイルとして作製されます。尚、RStudio上で、**R Notebook** を扱う場合、出力について特別な事をする必要はなく、単に、`.Rmd` ファイルを保存するたびに`.nb.html` ファイルが自動で出力されます。(既にファイルがある場合上書きされて、保存された.Rmdに対応する最新のものになります)

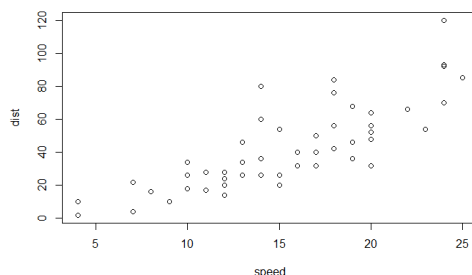
出力されたファイルの拡張子が`.nb.html`で、htmlファイルの一種であることが分かります。htmlファイルは、いわゆるインターネットで見ているページのもとなっているhtml書式で書かれたファイルであり、この種類のファイルは、インターネットエクス

R Notebook

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the Run button within the chunk or by placing your cursor inside it and pressing `Ctrl+Shift+Enter`.

```
plot(cars)
```



Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing `Ctrl+Alt+I`.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press `Ctrl+Shift+K` to preview the HTML file).

図5.4 R Notebook 出力

ローラー等の Web ブラウザで開いて見る事が出来ます。この RStudio の右下ペイン Files タブの中で、`.nb.html` という拡張子がついたファイル名の部分をクリックすると、メニューがポップアップするので、その中から **View in Web Browser** を選択してください。これが、体裁を整えたマークダウンの出力ファイルです。(図5.4)

RStudio の編集ペインで書き込んでいる内容が、まあまあ綺麗な体裁で、表示できていることが分かると思います。

もう少しだけ、マークダウンの話を続けます。マークダウンは、色々なところで、色々なものに使われます。例えば、あなたが、今読んでいるこの文書ですが、これもマークダウンで書かれてた文書に体裁を整える処理が行われてファイルとして出力され、それをあなたが読んでいます。

もし、Web 上で読んでいるならば、html ファイルとして出力されたものを読んでいますし、電子ブックでなら、epub ファイルとして、紙ベースなら pdf として出力し、それを印刷したものかもしれません。あなたが、どの読み方をしていたとしても今僕が書いているのは、文書の体裁等を全く気にせず構造と内容のみを書き綴っただけの R マークダウンファイルなのです。(図5.5)

5.4.3 出力形式の変更

markdown は、その処理によってさまざまな出力をすることが出来るというお話をしました。そして、R Notebook も、その markdown によるさまざまな出力のひとつなのです。さて、では、さまざまな出力の中から、どうやって出力したい形式を選ぶのでしょうか？実際には、色々な方法がありますが、R Notebook を RStudio で扱う場合には、ファイル先頭の YAML ヘッダ内の、`output:` 項目に `html_notebook` という値を与えます。RStudio では、この様に設定されている Rmd ファイルを読み込めば、R Notebook で出力してくれます。

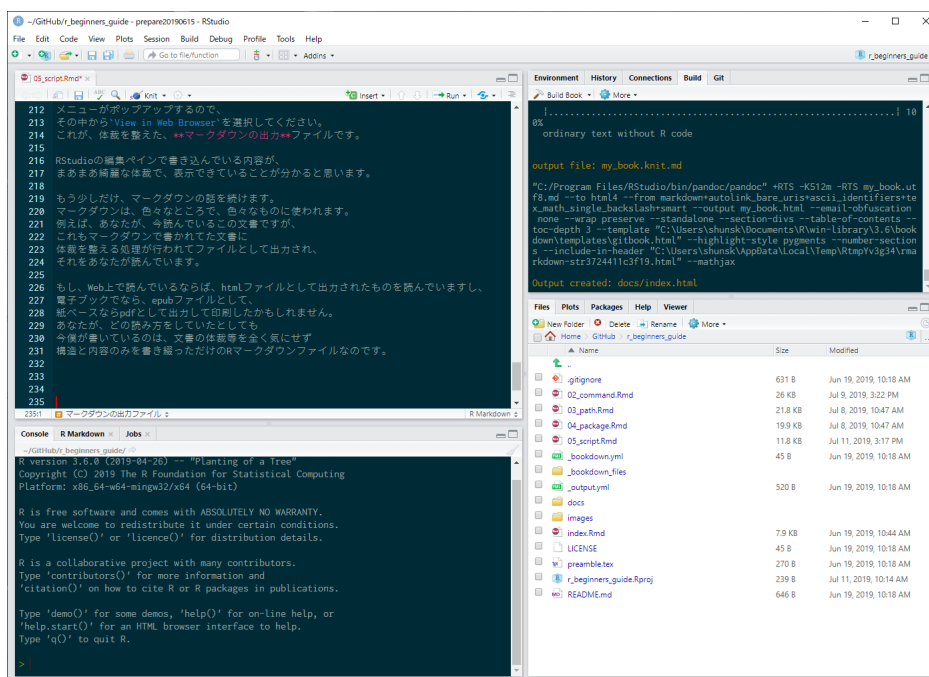


図5.5 この文書の作成風景

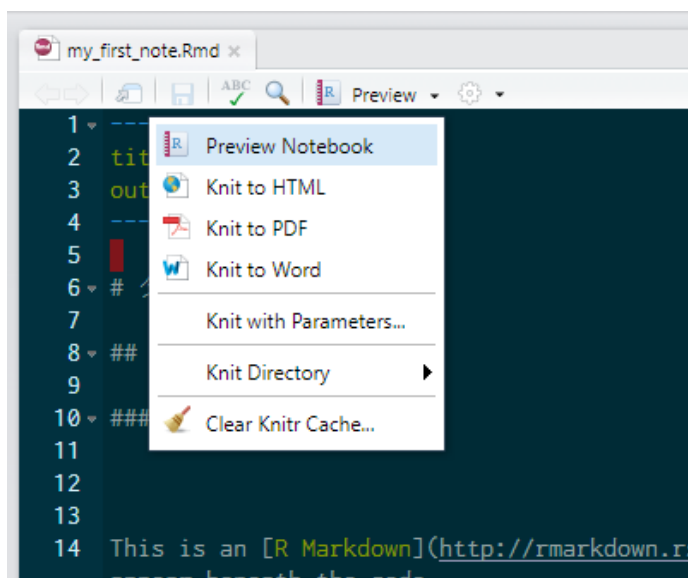


図5.6 出力形式の選択

すなわち、この部分を他の値に書き換えると、別の形式で出力してくれるのです。

編集ペインのツールバーに R と書かれたノートの絵が描かれたアイコンとその横には Preview と書かれている部分（ここをクリックすると、R Notebook の出力のプレビューをしてくれます。）がありますが、その Preview と書かれた文字の右隣に小さな下三角のアイコンがあります。そこを、クリックするとメニューが現れます。（図5.6）

メニューの中に、次の選択肢がみられます。

- Knit to HTML
- Knit to PDF
- Knit to Word

これらを選択すれば、それぞれ、HTML、PDF^{*1}、Word の出力を行ってくれます。Knit は、ニットと読みます。ニットは編み物をするという意味の英語ですが、ここでは、マークダウンの処理をする R パッケージ名です。つまりは、「ニットする」とは、編み物を編み上げるように、マークダウンファイルを処理して出力ファイルを作るということを示しています。

では試しに、HTML ファイルの出力を行ってみましょう。メニューから Knit to HTML を選択してください。右下ペインの Filse タブに .html ファイルが新たに作成され、プレビューウィンドウも立ち上がるかもしれません。しかし、一番注目して欲しいのは、編集ペインのメニューバーに Preview と書かれた部分がなくなり、その代わりに、青い小さな毛玉のアイコンとその隣に Kint と書かれた部分が現れる事です。そして、Kint と書かれた部分の右隣の下三角をクリックして現れるメニューに R Notebook の文字が無い事も確認してください。

この様にメニュー操作をしていただけなのに、意図せず、R Notebook としての出力が出来なくなることがあります。

5.4.4 R Notebook として出力するための設定

上述のように、色々な操作をしているうちに R Notebook での出力が出来なくなった時、YAML ヘッダの output: 項目を確認してください。

先の html 出力を選択した場合、

```
output:  
  html_document:  
    df_print: paged
```

という風になっているかもしれません。RStudio は、Rmd ファイルのこの部分を読み取って、それにふさわしいツールバーの内容を表示します。上記の例だと、html ファイルとして出力する設定になっているので、通常、html ファイル出力をするための処理を行う Knit を呼び出すメニューにツールバーが変更されたのでした。

そこで、R Notebook で出力したい時には、output: の内容を次のように書き換えて保存します。

```
output: html_notebook
```

^{*1} PDF 出力に関しては、RStudio や R 以外に TeX と呼ばれる組版システムのインストールが必要であったり、日本語については別途設定が必要であったりするので、選択してもエラーが出て上手く出力できないかもしれません。

書き換える時は、`output:` の行に続いて字下げされている行も全て消して書き換えます。
(先の例では、`df_print: paged` の行まで)

まずは、この方法だけ覚えておけば、予期せず出力設定が変わってしまった時に戻すことが出来ます。

5.5 マークダウンと R マークダウン

5.5.1 マークダウン

上述した通り、マークダウンでは文章の中身を記述するわけですが、その時、いくつかの簡単なルールに従って文章を書きます。また、この文章を書く部分というのは、先に述べた YAML ヘッダとチャンク以外の部分に書きます。以下に、基礎的なマークダウンのルールを少しだけ紹介します。

改行と段落

マークダウンで文章を書いていく時、文章の折り返しはその表示される媒体に任されるので、マークダウンファイル内での改行は無視されます。また、文章の区切りは段落を変えろという構造で表現します。マークダウンでは文書の間に空行を入れることで段落を変えることが出来ます。

では、次の文を `my_first_note.Rmd` 等のマークダウンファイルに書き込むか、コピー&ペーストしてみましょう。

これは、一つ目の段落です。
マークダウンファイルの中で改行しても
その改行位置に意味は無く、
出力ファイルではその位置で改行されません。

これが、二つ目の段落です。
文と文の間にからの行を入れると、
そこが段落の区切りになります。

書き込んだら、保存してプレビューを見てみてください。Rmd に書き込んだ形と、実際に出力される形の違いを確認してください。

見出し

段落の集まりには、見出しを付けることが出来ます。見出しには、章のように大きな見出し、節のように中くらいの見出し、その次の段落の集まりにつける小見出し等、段階を付けることが出来ます。

見出しは、行頭に # を並べて書くことで表します。# が多くなるほど、見出しが小さくなっていきます。次の例を、マークダウンファイルに書き込んで試してみてください。

大きな見出し

中くらいの見出し

中身の文書

小さな見出し

中身の文書

箇条書き

例えば、次のような箇条書きを書くことが出来ます。

- 果物
 - みかん
 - すいか
- 野菜
 - きゅうり
 - キャベツ

マークダウンでの箇条書きは、次のように行頭に-を付け一文字空白を開けて内容を書きます。更に、字下げを使うことで、箇条書きの入れ子を作ることも出来ます。上の箇条書きは、下のようなマークダウンで表現します。

- 果物
 - みかん
 - すいか
- 野菜
 - きゅうり
 - キャベツ

5.5.2 R マークダウン

マークダウンはここまでお話しした通り、文書を作成するための仕組みですが、これに R コマンドの処理や結果を融合させられるようにしたものが、**R マークダウン**です。

R Notebook を使って分析作業を行っていく場合、Rmd ファイル内でその分析についての説明等、文章で表現する部分については上述のマークダウンを活用して表現を行います。

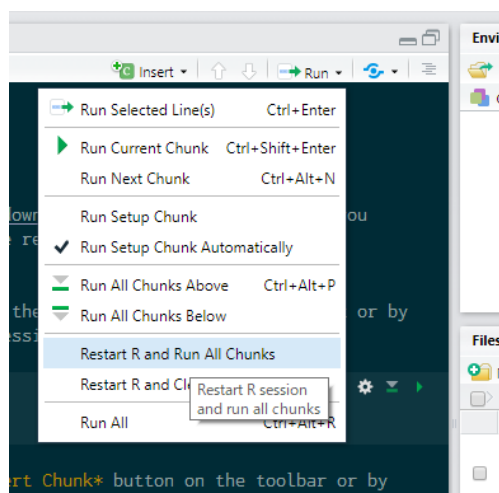


図5.7 Run メニュー

す。そして、実際のデータの扱いについては、チャンク内で、R のコマンドで処理の流れを表現し、それを実行すると、その処理で得られる結果が Rmd ファイルに埋め込まれます。これに加えて、R のコマンドを直接文書の一部に埋め込むインライン処理を使うことも出来ます。

チャンク

チャンクの部分に書き込んだコードは自動的に実行されません。チャンクの右上の右三角アイコンをクリックしたり、“Ctrl + Shift + Enter” のショートカットキーを押して実行する必要があります。また、文書内にチャンクが複数ある場合、或る実行結果によって保存された変数の内容は、それ以前にその変数を使って計算したチャンクについて自動的に再計算されない事に注意が必要です。

文書内の複数チャンクの実行のさせ方は、編集ペインのツールバーにある Run の右にある小さな下三角をクリックすると出てくるメニューの中にたくさん用意されています。最終的に、文書内の計算に整合性を持たせるには **Restart R and Run All chunks** を選択するとよいでしょう。(図5.7) これを実行すれば、新しい環境のもと、チャンクをはじめから最後まで順番に実行してくれます。

インライン

また、チャンク以外に直接 R のコマンドを埋め込み、出力の時には自動的にコマンドの実行結果に置き換えることが出来る書式があります。次のように、文章の途中にバッククオートで囲んだコマンドを書きこみます。一番初めの `r` の文字を忘れないようにしましょう。

今の時刻は、``r date()``です。

プレビューを確認すると、コマンドの部分が保存した時刻になっているはずです。

今の時刻は、Sat Jul 13 11:14:02 2019 です。

5.6 R Notebook とは

さて、あらためて、R Notebook とは、どういうものかを見ていきます。

RStudio で、R Notebook 用の Rmd ファイルを編集ペインで開いた状態にします。まず、編集ペインのツールバーから **Preview** と書かれている部分を押してください。別途、体裁を整える処理がなされた文書が別ウィンドウで開きます。

5.6.1 保存すると、自動で出力される

先に述べましたが、R Notebook は、Rmd ファイルが保存されるとすぐさま、自動的に出力されます。ですから、編集ペインで、文書の何処かに少し文書を付け足して、保存してみてください。付属のプレビューウィンドウは、元のファイルが変更されると表示が更新されるのでプレビューウィンドウ内の表示が自動的に書き換えたものになっているはずです。

5.6.2 ファイルを持ち運べる

R Notebook とは結局、Rmd ファイルに基づいて出力された **.nb.html** ファイルのことを言います。

このファイルは、RStudio や R の環境に関係なく、誰でも見ることが出来ます。例えば、RStudio を終了し、そのプロジェクトファイルの中にある **my_first_note.nb.html** をデスクトップ等にコピーし、そのコピーしたファイルをダブルクリック等で開いてください。自分がいつも使っている Web ブラウザが起動して、先ほどまで見ていた体裁を整える処理がなされた文書が表示されるはずです。

このファイルをそのままメールに添付して送信すれば、受け取った人も、そのまま同じものを見ることが出来ます。その時、その人が R や RStudio を持っている必要はありません。

5.6.3 只の html ファイルではない

R Notebook は、単に表示するだけならば html ファイルと同じなのですが、そのもとのデータである **Rmd** データがファイルの中に含まれています。

RStudio で、新しいフォルダを作成し、そこに空のプロジェクトを作製してください。そして、そのフォルダに先ほどコピーした **my_first_note.nb.html** を入れてください。準備が出来たら、この新しいプロジェクトを開きます。

右下ペインの Files タブの中の `my_first_note.nb.html` をクリックするとメニューがポップアップするので、`Open in Editor` を選択します。すると、`.nb.html` ファイルから `Rmd` ファイルが抜き出され、プロジェクトディレクトリの中に `my_first_note.Rmd` が作成されます。同時に、左上の編集ペインが開いて、このマークダウンファイルを編集出来るようになります。