Chapter01. R の導入とデータ操作の基本

池田龍也

2025-03-09

目次

1　はじめに

1.1　この資料の使い方

この資料は主に**心理系の大学院生が R で分析できるようになることを目的**に作成した。そのため心理統計については別に学ぶ必要がある。幸い, 日本語または英語で読める資料はオンラインにも書籍にも潤沢に存在している。また臨床心理系の大学院であれば, 統計法や研究法の授業があるはずである。

この資料は R スクリプトと分析結果, およびその解説を記載しているため, 長くなりがちである。各章には, 色付きのボックスが配置されている。**忙しい人や概要だけ掴みたい人は, まずボックスから読むことをおすすめする**。各ボックスの意味は以下の通りである。

> **ノート**
>
> このボックスは情報提供である。書籍やパッケージなど, 読者にとって有益と思われる内容を記載する。

> **ヒント**
>
> このボックスには各章のうち, ここだけ押さえればとりあえず OK という内容を記載する

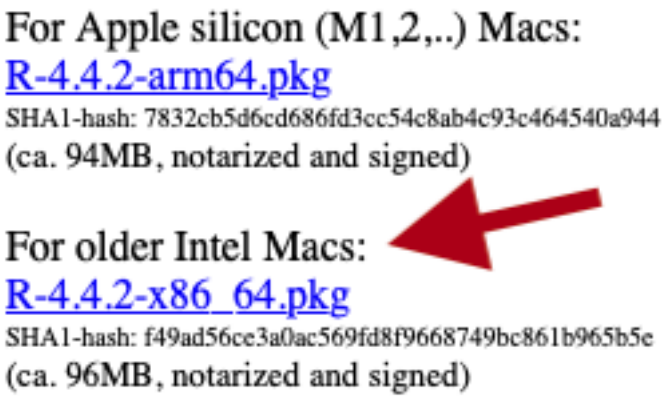
> **重要**
>
> 取り返しのつかない失敗を避けるための注意喚起, 実施にあたって知っておくべきことなど, 注意事項を記載する

2　R・RStudio のダウンロードとインストール

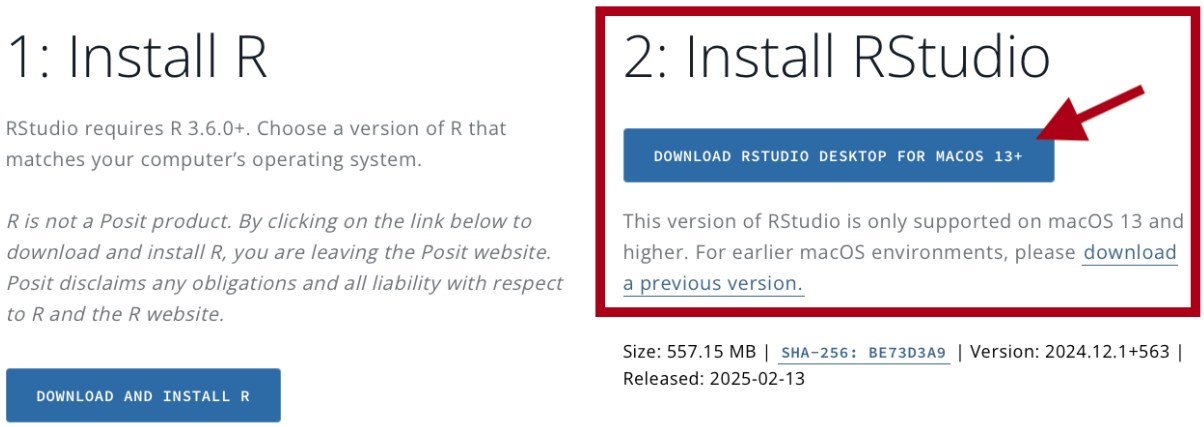
2.1　インストール

R を使用するだけなら R の公式サイト（https://ftp.yz.yamagata-u.ac.jp/pub/cran/）で OS に合ったものをダウンロードし, インストールすれば良い。**R は MacOS，Linux，Windows に対応**している。インストールそのもので困ることはとくにないはずであるが，Windows 版のインストールについては東京経済大学システム課のサイトが丁寧に解説されていて参考になる（URL：https://www.tku.ac.jp/iss/guide/classroom/soft/rrstudio-desktop.html）。

macOS 版の R については「続ける」「インストール」を選択し続ければ良い。macOS は CPU が Apple シリコンか Intel かで対応する R のバージョンが異なる。最新の mac は Apple シリコンが搭載されているはずである。もし自分の mac の CPU がどちらか分からなければ, メニューバーの**リンゴアイコン > このマックについて**を選択し, プロセッサを確認すればどちらか分かる。この項目に Intel と書いてあれば For older Intel Macs のほうをダウンロードする。



RStudio は R を便利に使うための統合開発環境と呼ばれるソフトウェアである。とくに事情がないかぎり，**RStudio も一緒にインストールすることをおすすめする**。公式サイト（https://posit.co/download/rstudio-desktop/）から OS に合ったものをダウンロードしてインストールする。筆者は mac0S を使っているため青いボタンのところに「MACOS 13+」と書かれているが，Windows や Linux でアクセスすれば対応した OS に変わる。



インストールにあたって注意事項がある。**R Studio を用いてパッケージをインストールするとき，WindowsOS の PC のユーザー名に全角文字（ひらがな，カタカナ，漢字，など）が入っていると正常に実行できないことがある**。そのようなケースでは環境変数にパスを指定すると正常に動作するようになる。環境変数にパスを指定する方法は慶應義塾大学の松浦寿幸先生の Note が参考になる（URL：https://note.com/toshi_matsuura/n/n487eecff9632）。

2.2 古いバージョンの入手

古いバージョンの OS を使っている場合，最新の R や RStudio をインストールできないことがある。また，特定の R や R Studio のバージョンで微妙な不具合があることもある。このようなケースでは古いバージョンの R や R Studio を用いる。R はダウンロードサイトにアーカイブへのリンクがあり（macOS：https://ftp.yz.yamagata-u.ac.jp/pub/cran/，Windows：https://ftp.yz.yamagata-u.ac.jp/pub/cran/），

対応するバージョンを入手できる。R Studio は上記の画像で示したボタンの下に「download a previous version」のリンクがあるので, ここから以前のバージョンを入手できる。

なお, 分析で使用するパッケージのなかには「このパッケージを使うには R のバージョンが○○以上でないといけない」のように指定されていることがある。この場合は諦めて最新の OS にしてから R と R Studio をバージョンアップするか，Google Colaboratory のような OS に依存しないサービスを用いるかして対処する。

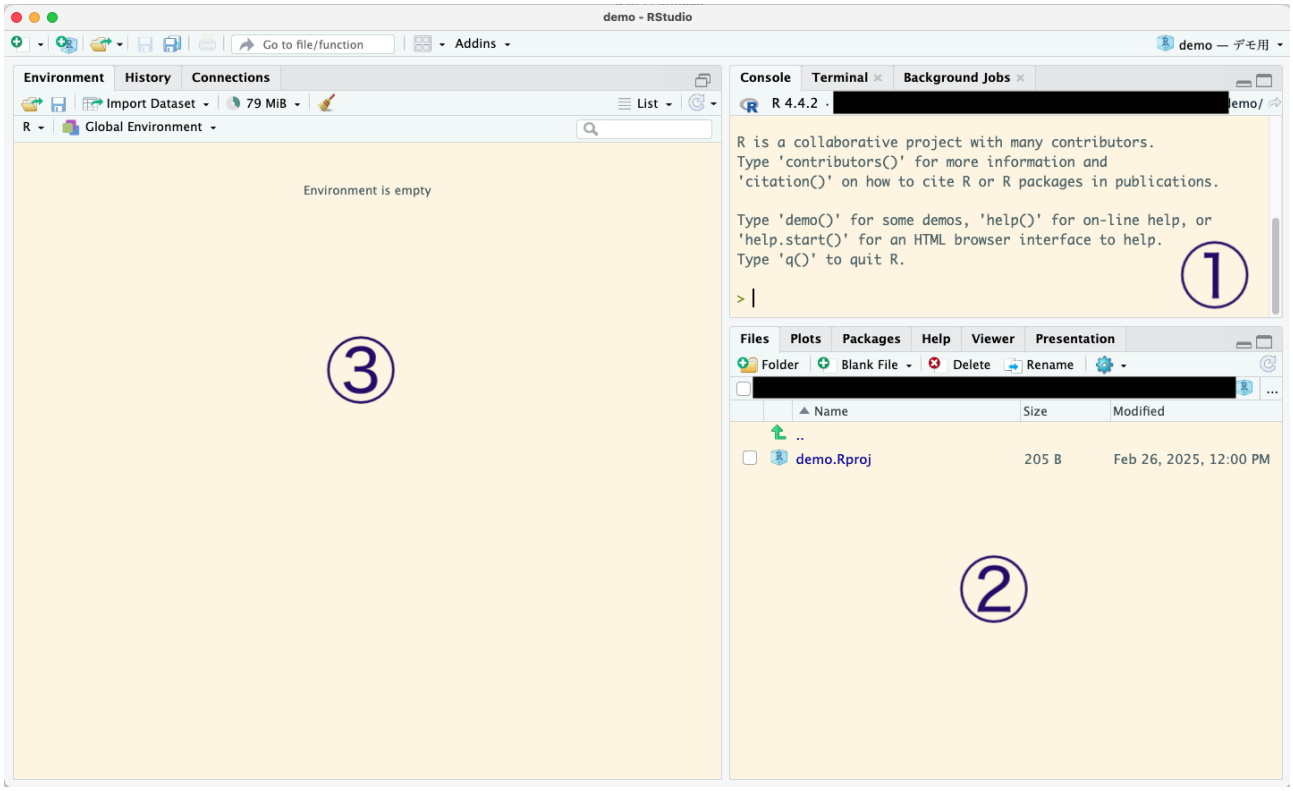
> ここを押さえれば OK
>
> R も RStudio も無料で使える
> R の公式サイト（https://ftp.yz.yamagata-u.ac.jp/pub/cran/）と RStudio の公式サイト（https://posit.co/download/rstudio-desktop/）からダウンロードできる
> R を使うなら一緒に RStudio を使うのがおすすめ

> 気をつけよう
>
> ユーザー名に全角文字が入っているとライブラリのインストールにトラブルが生じることがある。そのようなケースでは松浦寿幸先生の Note を参考に解決できる。

3　R と RStudio の使い方

3.1　R Studio 画面の説明

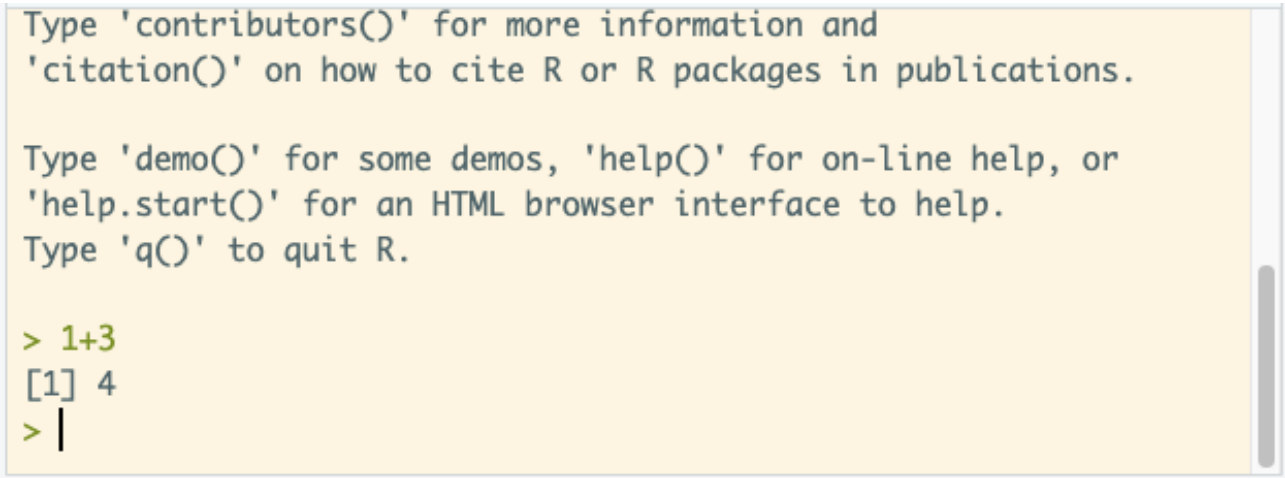
RStudio を開くと, 画像のような画面が表示される(筆者は使い勝手の都合でパネルの配置を変更しているので, インストール初期の配置とは違うかもしれない)。ちなみに背景色や文字色はメニューバーから **Tools > Global Options... > Appearance** で変更できる。修士論文や博士論文などで R Studio の画面を四六時中見つめる必要があるときは, 目に優しいカラーリングにしたほうが何となく疲れが少ない気がする。



画面を説明する。最初に開いたときに表示されるパネルは 3 つである。

① ここに入力された内容が計算される。
②色々なタブがあり, いま開いているフォルダ(Files), 図(Plots), パッケージの管理(Packages), パッケージのマニュアル(Help)などを表示できる。
③ いまあるオブジェクト(データや関数など)が一覧で表示される。

① に表示されている `>` のあとに `1+3` と入力してエンターを押すと, 画像のように `[1] 4` と表示される。

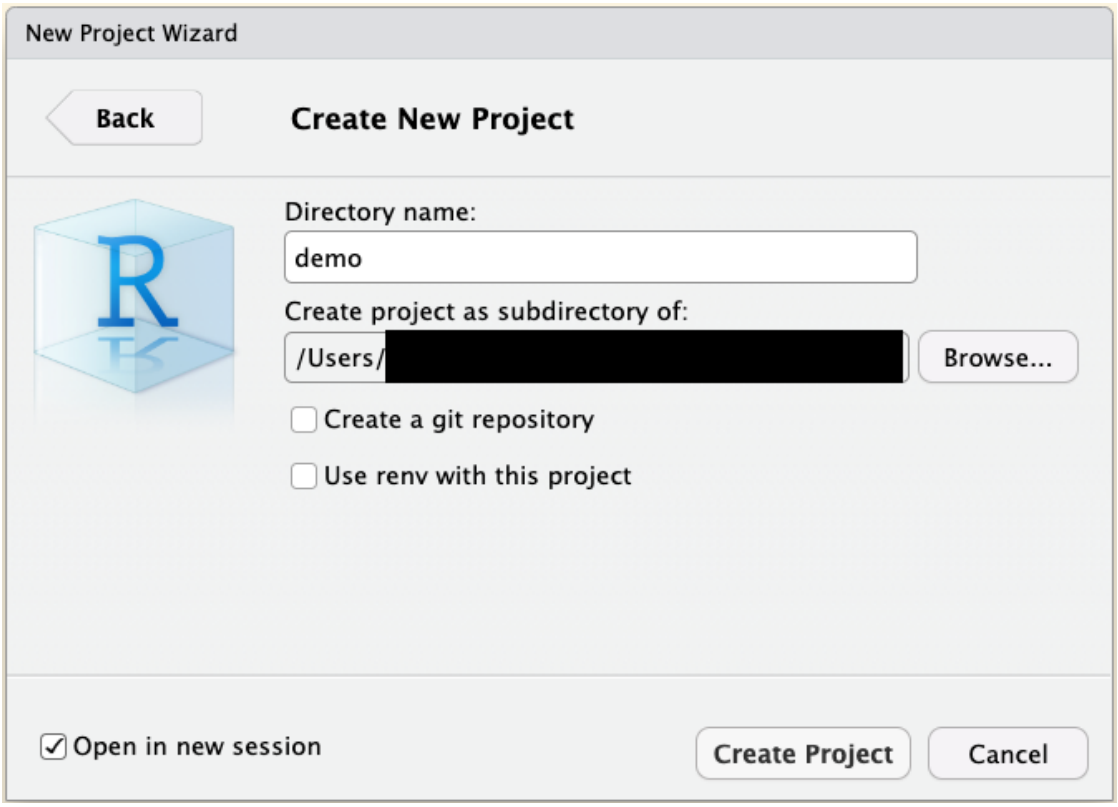


R では基本的に 1 行 1 行スクリプトを書いていき,順番に実行することで計算する。このようなユーザーインターフェースを **CUI** と呼ぶ。これに対して SPSS や JASP のようにボタンを押していくタイプのものは GUI と呼ぶ。それぞれ一長一短あり, どちらかが絶対的に優れているというわけではない。

R は python ほど**導入の敷居が高くなく,無料で,ユーザー数も多いため日本語・英語の資料が充実している**。その気になればかなりマニアックな手法も用いることができる。勉強すれば美しいグラフや論文に掲載可能な図を作ることもでき (池田 (2024) や Ikeda & Urano (2025) の図はすべて R で描画した), 分析結果のレポートを PDF で作ることもできる (この PDF も R で書いている)。そのため統計解析に限れば R で十分な場面が多い。もし将来的に機械学習をやってみたいなら python でも良いかもしれないし, ベイズを手軽に実施したいなら JASP でも良い。有償ソフトウェアの JMP はアカデミックライセンスが無償化されたため, 教育研究機関に所属している間は申請すれば無料で使える。JASP と JMP はいずれも GUI であり, どうしても CUI に抵抗がある学生や教職員におすすめできる。

3.2 プロジェクトファイルの作成

最初に画面上部の**メニューバーから File > New Project...**と選択するとポップアップ画面が表示されるので, **New Directory → New Project** の順に選択する。最後にプロジェクトの名前(≒フォルダ名)と場所(＝保存先)を決める。**「Browse...」ボタン**を押せば見慣れたウィンドウが表示されて, どこにプロジェクトフォルダを作るか決められる。今回はどこでも良いので, 好きな場所にプロジェクトファイルを作成しよう。なお, まっさらなプロジェクトファイルを作成するために, **左下の「Open in new session」のチェックボックスにはチェックを入れておくことをおすすめする**。

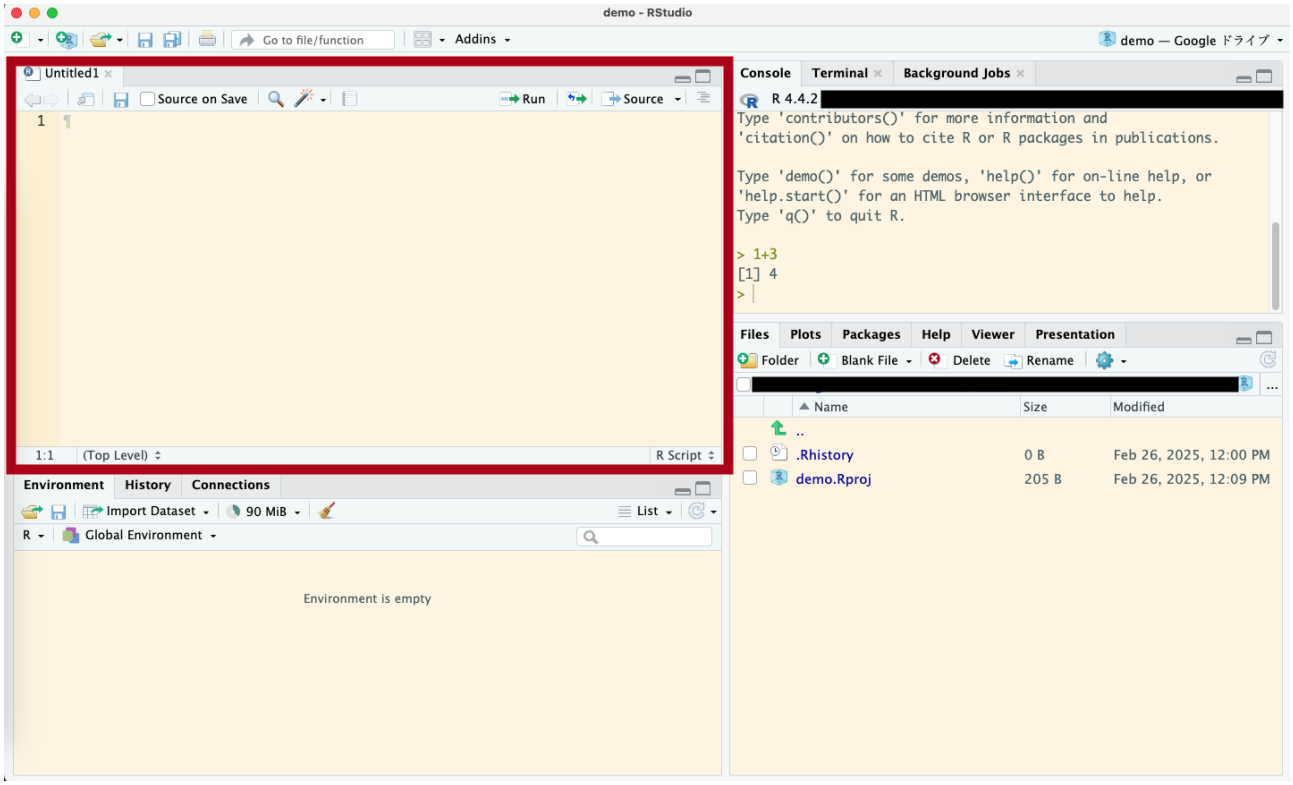


ディレクトリというのはフォルダのようなもので, ここでは R で分析するための新しいフォルダとプロジェクトファイルを作成している。ではなぜ直接新規フォルダの作成をしないのかというと, ただ新規フォルダを作ってもプロジェクトファイルがないため, 色々と不都合であるためである。たとえば後述するように, R に csv データを読み込む場合, プロジェクトファイルと csv ファイルの位置関係が重要になる。また, これも後述するように, データセットや画像を出力する場合も, プロジェクトファイルと保存先の位置関係が重要である。おそらく, ピンとこない人も多いと思われるが, ここでは**どこで作業しているかはっきりさせるため**にプロジェクトファイルを作っている, という理解で十分である。

プロジェクトフォルダを見ると，demo.Rproj（拡張子を非表示にする設定なら demo）というファイルができている。一旦 RStudio を閉じても, このファイルを開けば**どこで作業しているかはっきりした状態で作業を再開**できる。

3.3 R スクリプトを書く

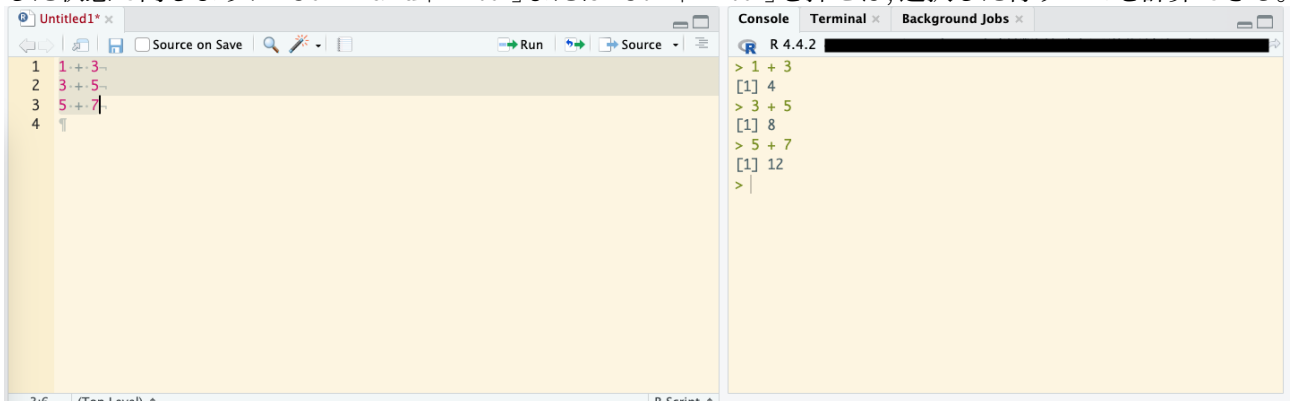
CUI で分析するうえで，①に書いていくのは効率が悪い。そこで分析用のコードを書く場所と計算する場所を分ける。画面上部のメニューバーから **File > New File > R Script と選択すると,新しい R スクリプトを作成できる**。



R に計算させる内容は，**原則としてここ（エディタ）に書くようにするほうが良い**。R ファイルを保存しておけば分析結果を再現しやすく，公開もしやすい。

R スクリプトを保存するにはエディタパネル上部のフロッピーディスクのアイコンを押すか，他のソフトと同じように**「Command+s」または「Ctrl+s」で保存**できる。最初に保存するときはどんな名前で保存するか聞かれるので，わかりやすい名前を付ける。例えば前処理用のスクリプトなら cleaning.R や transform.R，t 検定なら t_test.R のように，ファイル名を見て何をするために書いたスクリプトなのか分かるようにしておくほうが良い。**博士論文用の R スクリプトだから D_themes.R とか hakuron.R にすると，あとから理由が分からなくなるためおすすめできない**。

さっそく何か書いてみよう。エディタに 1 + 3 と書き，その行で「Command+Enter」(macOS) または「Ctrl+Enter」(Linux / Windows) を押すと，コンソールで計算が始まる。また，画像のように複数行選択した状態で同じように「Command+Enter」または「Ctrl+Enter」を押せば，選択した行すべてを計算できる。



3.4 四則演算 + α

四則演算はそれぞれ以下のように書く。いずれも一般的な表現である。

```r
# 和
1 + 2
```

```
[1] 3
```

```r
# 差
2 - 1
```

```
[1] 1
```

```r
# 積
2 * 3
```

```
[1] 6
```

```r
# 商
10 / 2
```

```
[1] 5
```

その他のメジャーな計算は以下のように書く。記述統計については別に示す。

```r
#累乗
2^3

#三角関数
sin(0)
cos(0)
tan(0)

#対数
log(10)

#自然対数
exp(10)
```

`sin()` や `exp()` のように英語 +() で書かれたものを関数と呼ぶ。関数は複数の処理をまとめたもので, ほかにも t 検定を実施する `t.test()` 関数や相関係数を計算する `cor()` 関数がある。平均値を例にとると, 平均値を求めるにはサンプルサイズがいくつかるか数え, 総和を求め, 総和をサンプルサイズで割る必要がある。この処理をまとめたものが `mean()` 関数であり, カッコ内にデータを入れることで平均値を返す (カッコ内に入る数字や文字を**引数**と呼ぶ)。さらに自前の関数を作ることもできる。

```r
# 自作関数
my_function <- function(x){
  result = log(x) + exp(x)
  return(result)
}
my_function(x = 5)
```

```
[1] 150.0226
```

```r
# 検算
log(5) + exp(5)
```

```
[1] 150.0226
```

関数の引数は x であり，`my_function(x = 5)` を実行すると `log(5) + exp(5)` を計算した結果を返す。分析中に同じような計算を繰り返す場合, その処理を関数にしてしまったほうがスクリプトがすっきりする。

3.5 変数という名の箱

計算結果をどこかに置いておいて, あとから再利用したいことがる。たとえば t 検定で得た t 値を使って効果量を計算したいとき, いちいち計算し直すのは面倒くさい。そこで**計算結果を入れておく箱を用意する**。このような箱のことを**変数**と呼び, 箱に入れる操作を**代入**と呼ぶ。R では<-や=で代入する。**<-の左側が箱,右側が中身である。**

```r
result <- 10 + 5
result
```

```
[1] 15
```

上記の例では,「10 + 5」の計算結果を「result」という箱に保存し (1 行目), その中身を表示している (2 行目)。さらにこの箱は計算にも使える。

```r
result / 3
```

```
[1] 5
```

result の中身は 15 なので，3 で割ると答えは 5 になる。もちろんこの計算結果を別の箱に入れることもできる。

```r
result2 <- result / 3
result2
```

```
[1] 5
```

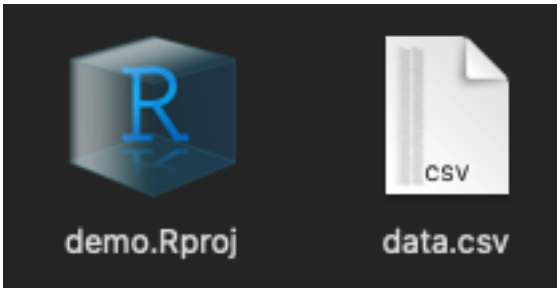
ほかにもデータセットを箱に入れておくこともできる。欠測値を処理したデータと処理前のデータを別々の箱に入れておくこともできるし，Excel や csv ファイルを直接書き換えることがない。そのため「間違って素データに上書き保存してしまった」のような事故を防ぐことができる。

> ここを押さえれば OK
>
> R はコンソールに結果が出てくる。
> 新しく分析をはじめるときは RStudio でプロジェクトファイルを作ろう。
> スクリプトはコンソールでなくエディタに書こう。
> 関数は処理を 1 つにまとめた便利アイテム。
> 変数はただの入れ物。

4　データの読み込みと書き出し

4.1　データの読み込み

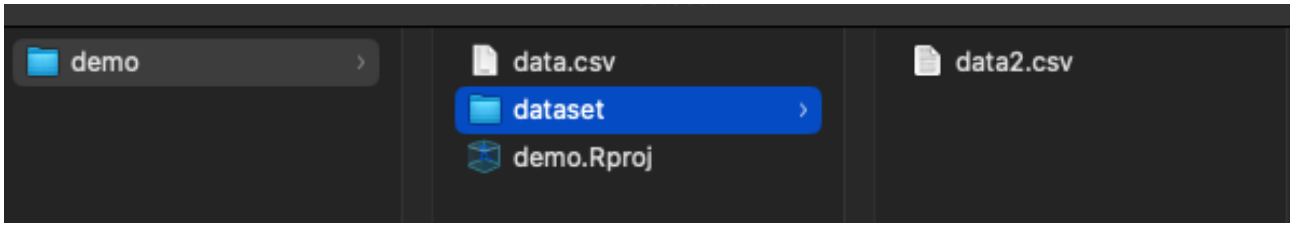
先に触れた通り，**csv や Excel などからデータを読み込むこともできる**。ここでは下図のように同じフォルダにある data.csv を読み込んでみよう。




```
dat <- read.csv("data.csv")#data.csv を読み込んで，dat という箱(変数)に入れる
head(dat, 6)#読み込めたか確認するために,最初の 6 行を表示する
```

```
  X     group_a     group_b
1 1 -0.56047565 -0.71040656
2 2 -0.23017749  0.25688371
3 3  1.55870831 -0.24669188
4 4  0.07050839 -0.34754260
5 5  0.12928774 -0.95161857
6 6  1.71506499 -0.04502772
```

では,以下の画像のように同じフォルダに dataset フォルダがあり,その中に data2.csv がある場合はどうであろうか。



さきほどと同じように読み込んでみよう。

```
dat2 <- read.csv("data2.csv")
```

今回は以下のようにエラーが表示された。

```
> dat2 <- read.csv("data2.csv")
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :
  cannot open file 'data2.csv': No such file or directory
```

エラー内容を翻訳すると「data2.csv を開きたいって言ってるけどさ, そんなフォルダもファイルもないよ？」である。このタイプのエラーは**ファイル名のスペルミスや存在しないファイルを読もうとしたときに生じる**。read.csv("data2.csv") は「このフォルダ内にある data2.csv を読んでね」という指示であり, R は忠実に指示に従い, その結果 No such file or direcroty と言っているのである。

同じフォルダにファイルがない場合,どこにファイルがあるのか教えてあげる必要がある。今回は dataset フォルダにあるので, 以下のように書く。

```
dat2 <- read.csv("dataset/data2.csv")
head(dat2, 6)
```

```
  X     group_a     group_b
1 1 -0.56047565 -0.71040656
2 2 -0.23017749  0.25688371
3 3  1.55870831 -0.24669188
4 4  0.07050839 -0.34754260
5 5  0.12928774 -0.95161857
6 6  1.71506499 -0.04502772
```

"dataset/data2.csv" が data2.csv のある場所である。きちんとどこにファイルがあるか教えてあげたため, R もきちんとファイルを読み込んでくれる。融通を利かせてほしいところではあるが, きちんと指示をしていないこちらに非がある。
あなたが食卓についているとき,「塩をとって」と頼まれたとしよう。食卓の上にも手の届く範囲にも塩がなければ, あなたは「ないよ」または「どこにあるの」と返すであろう。ここで相手が「いや, キッチンの右側の上から二番目の引き出しにあるから。ちゃんと探してよ」と言ったら,「それならせめてキッチンにあるって言ってよ」となるはずである。R もきっと同じようなことを思っているはずである。

4.2 データの書き出し

データを分析していると, 色々な処理を施したデータを保存しておきたくなることがある。R では write.csv() でデータを csv ファイルとして書き出すことができる。

```
write.csv(dat, "dat2.csv")
write.csv(dat, "dataset/dat3.csv")
```

write.csv(dat, "dat2.csv") を翻訳すると,「dat という箱に入ったデータを dat2.csv という名前の

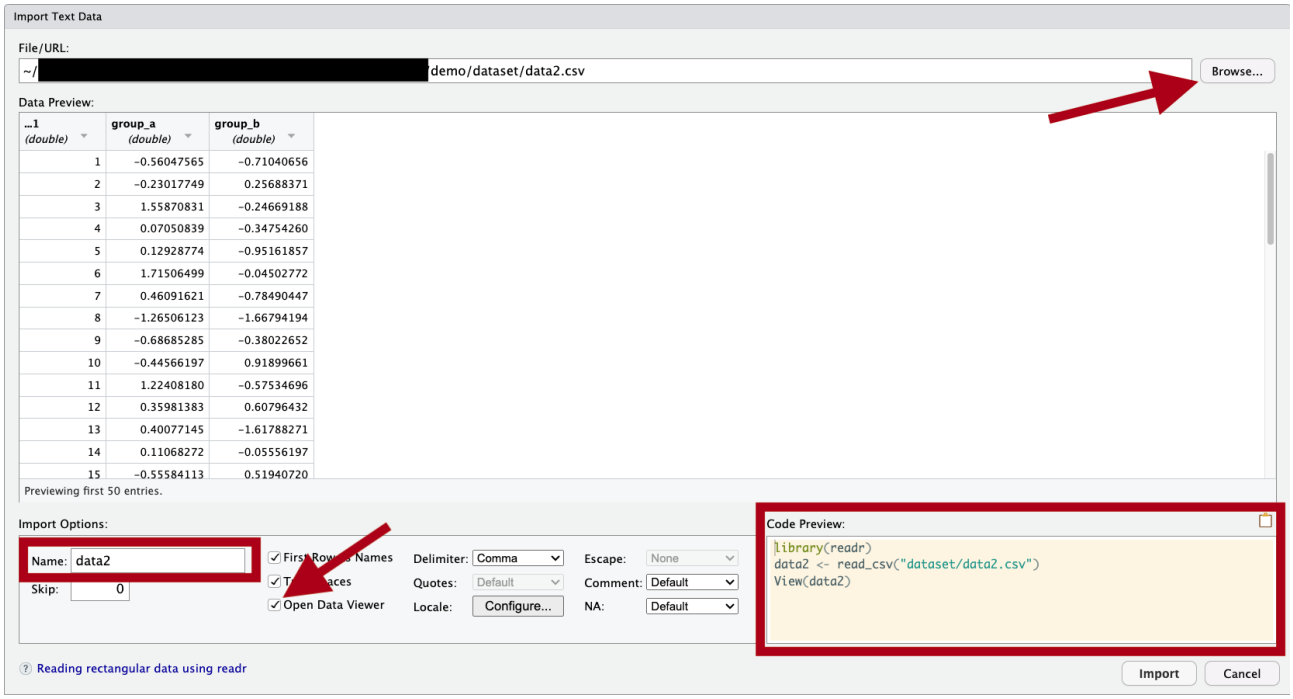
csv ファイルとして保存してね」である。実行するとプロジェクトファイルのあるフォルダに `dat2.csv` が現れる。

`write.csv(dat, "dataset/dat3.csv")` を翻訳すると,「`dat` という箱に入ったデータを `dataset` フォルダの中に `dat3.csv` という名前の csv ファイルとして保存してね」である。データの読み込みと同じく, データの書き出しでも場所を指定する必要がある。もし明示しないと，R は現在のフォルダに書き出す。もし存在しないフォルダを指定すると，`No such file or directory` とエラーメッセージが出る。翻訳すると「保存してねって言われてた場所, ないんですけど？」である。

読み込んだ csv と同じ名称で書き出したら, 元の csv ファイルは上書き保存される。しかも確認や警告は一切ない。そのため書き出しの際は注意する必要がある。

4.3 GUI を用いたデータの読み込み

もっと複雑な場所にファイルが存在する場合, 住所を全部書くのは大変かもしれない。そのようなケースでは GUI を使うのも良い選択である。R Studio のメニューバーから **File > Import Dataset >From Text(readr)** を選択すると, 以下のような画面がポップアップする。



この画面で「Browse...」から目当てのファイルを選択し, 画面左下の「Name」に好きな名称を入力し,「Open Data Viewer」のチェックを外す。「Import」を押すとデータを読み込める。なお画面右下の R スクリプトをコピーしてから「Import」を選択するのをおすすめする。R スクリプトを保存しておけば, 次回はこの操作をスキップしてコードを実行するだけで同じことができるためである。
「Open Data Viewer」にチェックを入れたままにしていると, データを読み込んだときにデータセットが表示される。よって目的のファイルを読み込めたか確認したい場合は, チェックを入れたままにしておく。

なお, 初回にこの操作を実行するとき，`readr` というパッケージを自動的にインストールするので, インターネットに接続した状態で実行する必要がある。パッケージの使い方については次に説明する。

ここを押さえれば OK

csv を読み込む関数は `read.csv()`。
csv を書き出す関数は `write.csv()`。
どこにあるファイルなのかはきちんと示そう。
GUI でデータを読み込むこともできる。
GUI でデータを読み込んだときも R スクリプトはコピーしてエディタに貼り付けておこう。
パッケージを使えば csv だけでなく Excel や SPSS も読み込め, 多くのデータ形式に対応できる。

気をつけよう

データを書き出すときに同じ名前のファイルがあると, 警告も確認もなく上書き保存される。不幸な事故を避けるために, 可能なら出力したデータを保存するフォルダを作り `write.csv(data, "output/data.csv")` のように別フォルダに書き出そう。

5　パッケージのインストールと読み込み

5.1　パッケージのインストール

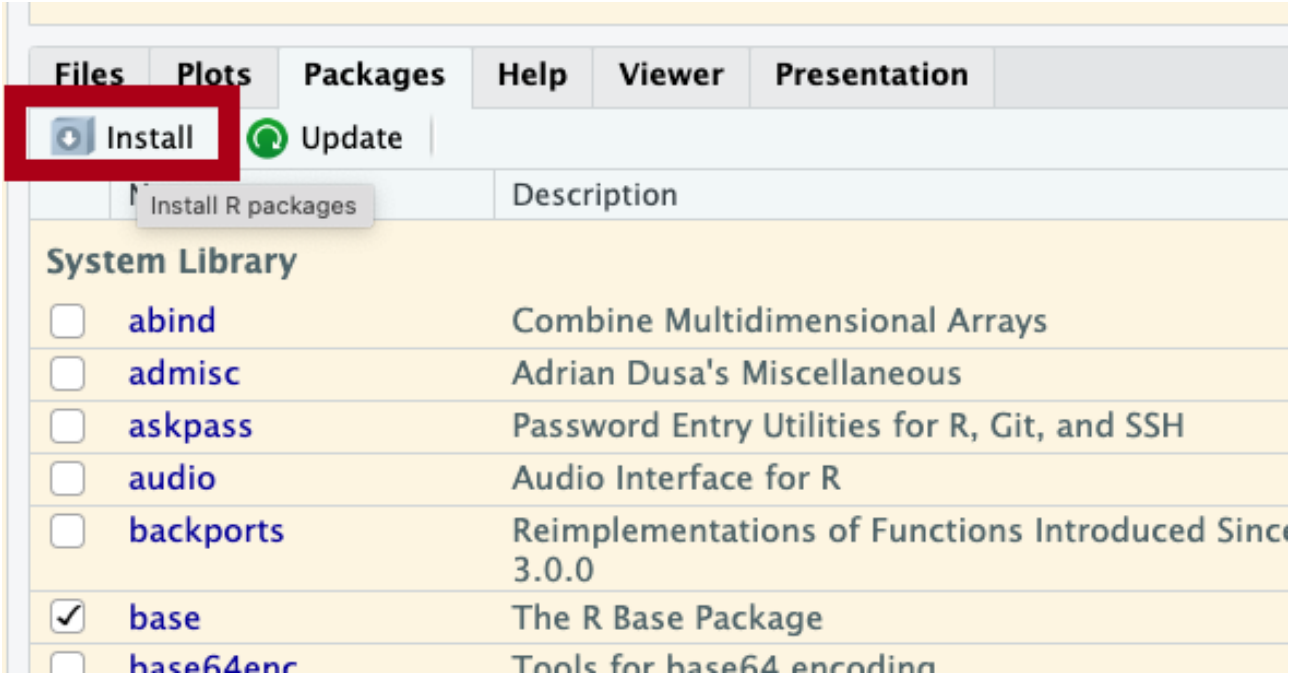
R では様々な分析が可能である。t 検定や相関係数など標準的な計算は, パッケージを追加しなくても実施可能である。共分散構造分析や項目反応理論など複雑な分析もパッケージなしでできなくはない。しかし非常に計算が面倒であるため, パッケージを利用したほうが圧倒的に便利である。そこでパッケージの入手について説明する。

パッケージをインストールするのは簡単であり, 原則として `install.packages("パッケージ名")` を実行すれば良い。ここでは例として tidyverse というパッケージをインストールする。

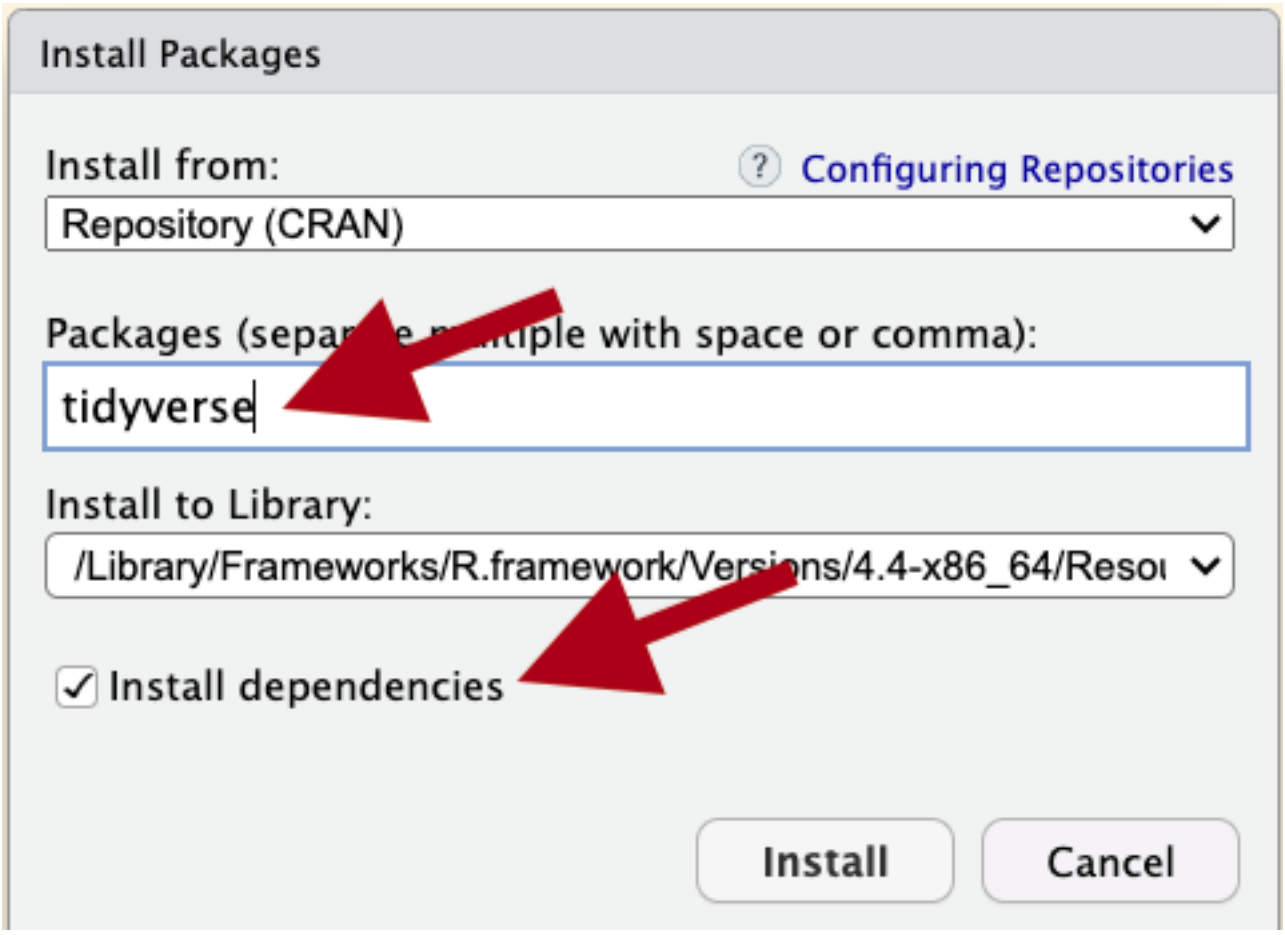
```
1 install.packages("tidyverse")
```

パッケージの一覧は CRAN の公式サイト (https://cran.r-project.org/web/packages/available_packages_by_date.html)から確認できる。また, どのようなパッケージがあるか調べたいなら, Google や Yahoo などで「○○分析 R パッケージ」のように調べると良い。マイナーなパッケージは日本語の検索にヒットしないことがあるため,「**analysis R package」のように英語で検索するとヒットすることがある。

GUI でインストールすることもできる。
「Files」や「Plots」のタブがあるパネルから**「Packages」タブを選択**する。パネル上部の**「Install」ボタンを選択**し, ポップアップした画面にパッケージの名前を入力してインストールする。



このとき,**「Install dependencies」のチェックボックスにチェック**を入れておく。パッケージを使うとき, 別のパッケージが必要になることがある。このチェックボックスを入れておくと, その必要なパッケージも一緒にインストールしてくれる。



5.2 パッケージの読み込み

パッケージをインストールしただけではパッケージを使えない。きちんと R に「このパッケージを使うよ」と教えてあげる必要がある。具体的には以下の通りである。

```
library(tidyverse)
```

原則として, この処理は 1 回の分析につき 1 回で良い。しかし R や PC を再起動した場合は改めて `library()` を実行する必要がある。別のプロジェクトファイルで分析するときも同様であり, プロジェクト A で読み込んだパッケージが, 自動的にプロジェクト B で読み込まれることはない。
`install.packages()` は原則として 1 回だけで良い。ただし R のバージョンをアップデートした場合は改めてインストールする必要がある。

ほかにもパッケージを使う方法がある。たとえば `psych` パッケージに含まれる `alpha()` がある。この関数は引数に指定した項目群の信頼性係数 α を計算する。data に抑うつ尺度の項目得点が入っているとして, α を計算するには

```
# library() を使わず alpha() を使う
psych::alpha(data)

# library() を使う
```

```
5 library(psych)
6 alpha(data)
```

のように書く。どちらで書いても結果は同じである。パッケージ名と関数名を:: で繋げて書けば，library() でパッケージを読み込む必要はない。ただしそれぞれに長所と短所があるため，原則として library() でパッケージを読み込み，例外的に:: で繋げて書くのが穏当であろう。関数のなかには，同じパッケージに含まれている別の関数を内部で呼び出しているものもある。このような関数は library() でパッケージを読み込んでおかないと，「そんな関数ないですけど？」とエラーが返ってくることになる。

ここを押さえれば OK

install.packages()：パッケージのインストール。原則としてインストールは 1 回だけで OK。
library()：パッケージの読み込み。R を起動するたびに読み込む。使わなくても読み込めることがあるが，R に慣れてから。

心理系でおすすめするパッケージ

以下は筆者が実際によく使っているパッケージである。

- tidyverse　可視化に特化した ggplot2 をはじめとしてデータ分析によく用いるパッケージがまとまっている。
  - ggplot2　綺麗な図を描画できる。複数のグラフを重ね合わせたり，複数のグラフをパネル上に並べたりなど，色々なことができる。学術論文に掲載できる水準の図を作ることもできる。使いこなすのは大変かもしれないが，うまく表示できるととても気持ち良い。
  - deplyr　データセットの操作に便利な関数は何でも揃っている。たとえばグループごとに平均や分散を計算してデータセットに追加したり，条件ごとに処理を変えたり，やりたいことはだいたい何でもできる。上手く処理できるととても気分がいい。
- psych　記述統計や相関係数，因子分析など心理学でよく用いる分析を簡単に実行できる。
- effectsize　効果量の計算に用いる。
- readxl　Excel を読み込むときに必要なパッケージ。オンライン調査で納品されるデータが Excel であった場合，csv に変換する手間なく読み込めるのがありがたい。
- mirt　項目反応理論を適用するときに使用する。
- lavaan　共分散構造分析に用いる。係数や適合度の計算だけでなく，測定不変性の検証や確認的因子分析もできる。
- devEMF　図をメタファイルとして保存するときに用いる。メタファイルで保存すると，画質がガビガビにならないのが便利である。

そのほかにもマルチレベル分析に lme4 を用いたり，ベイズ推定に rstan を用いたり，パッケージには大変助けられている。

6　データ操作

6.1　データ生成

R にはデータを生成する関数が標準搭載されている。架空のデータセットを作るメリットは様々であるが, 背景にある確率分布が明らかであるためデータ分析の練習にちょうど良い。また分布のパラメタ (正規分布でいえば平均と分散) を変えて, 分布の形状がどのように変化するか学ぶこともできる。さらに架空のデータであるため, 公開しても問題ない。ここでは**データを生成する関数を用い, 架空のデータセットを作成**する。

```r
set.seed(123)
n = 100
mean1 <- 0
mean2 <- 5
mean3 <- -2
sigma1 <- 1
sigma2 <- 5
sigma3 <- 0.2
x1 <- rnorm(n = n, mean = mean1, sd = sigma1)
x2 <- rnorm(n = n, mean = mean1, sd = sigma2)
x3 <- rnorm(n = n, mean = mean2, sd = sigma2)
x4 <- rnorm(n = n, mean = mean3, sd = sigma3)

y1 <- x1 * 5 + rnorm(n = n, mean = 0, sd = 1)
y2 <- x2 * -4 + rnorm(n = n, mean = 0, sd = 1)
y3 <- x3 * 0.1 + rnorm(n = n, mean = 0, sd = 1)
y4 <- x4 * -10 + rnorm(n = n, mean = 0, sd = 1)

z <- c(rep("R", 50), rep("L", 50))

data <- data.frame(
  x1 = x1,
  x2 = x2,
  x3 = x3,
  x4 = x4,
  y1 = y1,
  y2 = y2,
  y3 = y3,
  y4 = y4,
  z = z
  )
```

```
32
33 head(data)
```

```
           x1         x2        x3        x4         y1        y2          y3
1 -0.56047565 -3.5520328 15.994052 -2.143048 -2.8759343 13.606238  2.67341743
2 -0.23017749  1.2844185 11.562065 -2.150538 -2.3195389 -6.131373  1.12885952
3  1.55870831 -1.2334594  3.674275 -2.187708  7.1587933  5.960623  0.33409713
4  0.07050839 -1.7377130  7.715970 -2.210503  0.3237004  7.701913 -0.74447059
5  0.12928774 -4.7580928  2.928300 -2.087432  1.3171346 17.523205  1.08321537
6  1.71506499 -0.2251386  2.618766 -1.933764  6.9247784  0.805407  0.05114237
        y4 z
1 20.70227 R
2 19.96494 R
3 21.18398 R
4 22.22388 R
5 19.50961 R
6 19.92762 R
```

`rnorm()` は正規分布に従う乱数を生成する関数である。引数は `n` (サンプルサイズ), `mean` (母平均), `sd` (母分散) の 3 つである。乱数を再現するために `set.seed()` を使っている。もし `set.seed()` を使わずに乱数を生成すると, データ生成のたびにデータが変化してしまい, 同じ結果を得られなくなる。

`c()` は数字や文字を 1 つにまとめる関数である。ここでは最頻値を計算するために, 新しく `x` という箱に整数を 10 個入れた。文字を入れる場合は`" "`で囲って `c("Orange", "Apple", "Banana")` のように書く。

`rep()` は数字や文字列をたくさん生成するための関数である。ここでは `rep("R", 50)` で R を 50 個, `rep("L", 50)` で L を 50 個生成している。それぞれの文字を c() でまとめることで, R と L が計 100 個入った箱 `z` というを作っている。

`data.frame()` は複数の変数を束ねてデータフレームという形にする関数である。データフレームの詳細は省くが, 箱がバラバラの状態では分析しづらいため, 整理整頓して棚に収納したような状態だと理解すれば良い。棚のこの列には `x1` が入っていて, 別の列には `y1` が入っていて……という状態になっている。

6.2 要素へのアクセス

いま `data` という大きな箱には `x1`〜`z` まで 9 つの小さな箱が入っている。可視化したいときや記述統計を求めたいときは, 小さな箱のほうに用事がある。小さな箱に用事があるときは`$`を使って呼び出すことができる (例：`data$x1`)。また, 何列目かを指定することで呼び出すこともできる (例：`data[, 1]`)。さらに列番号を用いれば, 複数列にアクセスできる (例：`data[, 1:4]`)。

この方法では面倒だということであれば, `attach()` を用いることもできる。`attach(data)` と書けば,「data の変数に用事があるんですけど」と R に教えることができる。対になる関数は `detach()` であり, 用事がなくなり次第 `detach(data)` と書き,「もう大丈夫です」と教えてあげる必要がある。意外と `detach()` を忘れることが多いので, 注意してほしい。

```r
# 「$」を使ってアクセス
head(data$x1, 3)#先頭の 3 行だけ表示。以下同じ。
```

```
[1] -0.5604756 -0.2301775  1.5587083
```

```r
head(data$y3, 3)
```

```
[1] 2.6734174 1.1288595 0.3340971
```

```r
# 列番号を使ってアクセス
head(data[, 1], 3)
```

```
[1] -0.5604756 -0.2301775  1.5587083
```

```r
head(data[, 7], 3)
```

```
[1] 2.6734174 1.1288595 0.3340971
```

```r
# 列番号を使って複数行にアクセス
head(data[, 1:4], 3)
```

```
          x1        x2        x3        x4
1 -0.5604756 -3.552033 15.994052 -2.143048
2 -0.2301775  1.284419 11.562065 -2.150538
3  1.5587083 -1.233459  3.674275 -2.187708
```

```r
head(data[, 5:8], 3)
```

```
         y1        y2        y3       y4
1 -2.875934 13.606238 2.6734174 20.70227
2 -2.319539 -6.131373 1.1288595 19.96494
3  7.158793  5.960623 0.3340971 21.18398
```

```r
# attach() を使ってアクセス
attach(data)
head(x1, 3)
```

```
[1] -0.5604756 -0.2301775  1.5587083
```

```r
head(y3, 3)
```

```
[1] 2.6734174 1.1288595 0.3340971
```

```
detach(data)
```

attach(data) は detach(data) を実行するまで継続する。そのため次のようなケースでは, データの取り違えが生じる可能性がある。結果がどうなるか確かめてほしい。

```
A <- 3
x <- data.frame(
  A = 10,
  B = 11
)

A

attach(x)
A
detach(x)

A
```

最初に A を表示すると，3 と表示される。しかし attach(x) を実行してから A を表示すると，x の A 列にある数字の 10 が表示される。使い終わったら detach(x) を実行しておかないと，3 のつもりで A を用いているのに内部では 10 として処理され続けることになる。

さらに以下のようなミスもよくある。

```
x <- data.frame(
  A = 10,
  B = 11
)
attach(x)
x$new_number <- 13

new_number
detach(x)
```

このスクリプトを実行すると，4 行目の new_number を表示しようとしたときに Error: object 'new_number' not found とエラーメッセージが出力される。attach(x) を実行したあとに x へ追加した変数は, 変数の名前だけでは呼び出せない。この場合は以下のように書き換えると正しく機能する。

```
x <- data.frame(
  A = 10,
  B = 11
```

```
)
attach(x)
x$new_number <- 13
detach(x)

attach(x)
new_number
```

```
[1] 13
```

```
detach(x)
```

上記のスクリプトでは x に変数を追加したあとに一旦 detach(x) を実行し, 改めて attach(x) を実行している。こうすることで新しく追加した変数を変数名だけで呼び出せる。

6.3 条件に一致するデータの抽出

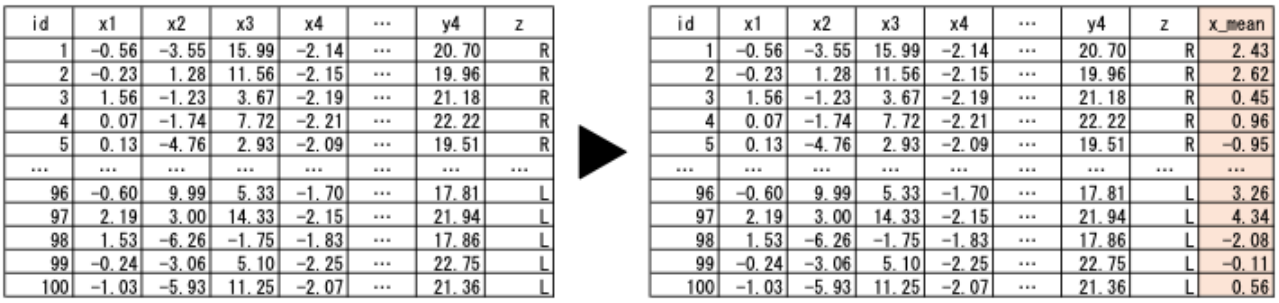
有効対象者のみを分析対象にしたい場合，**データセットの中から条件に一致するデータだけを抽出**する必要がある。このようなときは subset() を用いる。subset() はデータセットから条件に一致する行を抜き出す関数である。1 つ目の引数で抽出元のデータセットを指定し (上記のスクリプトでは data)，2 つ目の引数で抜き出す条件を指定する (上記のスクリプトでは data$z == "R")。
以下のように，2 つ目の引数には文字以外も用いることができる。

式	例	例の意味
>=	a >= 2	a が 2 以上である
<=	a <= 2	a が 2 以下である
>	a > 2	a が 2 よりも大きい
<	a < 2	a が 2 未満である
==	a == 2	a が 2 と等しい
!=	a !=	a が 2 でない

さらに複数の条件を組み合わせることもできる。たとえば z が R (data$z == "R") で x1 が 0 未満 (data$x1 < 0) である行を抽出したければ, 2 つの式を&でつないで subset(data$z == "R" & data$x1 < 0) とする。もし z が L (data$z == "L") または x2 が 1 以上 (data$x2 >= 1) の行を抽出したいなら, 2 つの式を|でつないで subset(data$z == "L" | data$x2 >= 1) とする。

6.4 列の追加

質問紙を使った研究では, 項目得点の平均値を尺度得点とすることがある。このようなケースではデータに変数 (列) を追加する。また, 同じ人を対象に 2 回調査を実施した場合，1 回目と 2 回目で別のデータセットを作っていることがある。このようなケースでも**列方向にデータを追加する**作業が必要になる。



id	x1	x2	x3	x4	…	y4	z
1	−0.56	−3.55	15.99	−2.14	…	20.70	R
2	−0.23	1.28	11.56	−2.15	…	19.96	R
3	1.56	−1.23	3.67	−2.19	…	21.18	R
4	0.07	−1.74	7.72	−2.21	…	22.22	R
5	0.13	−4.76	2.93	−2.09	…	19.51	R
…	…	…	…	…	…	…	…
96	−0.60	9.99	5.33	−1.70	…	17.81	L
97	2.19	3.00	14.33	−2.15	…	21.94	L
98	1.53	−6.26	−1.75	−1.83	…	17.86	L
99	−0.24	−3.06	5.10	−2.25	…	22.75	L
100	−1.03	−5.93	11.25	−2.07	…	21.36	L

id	x1	x2	x3	x4	…	y4	z	x_mean
1	−0.56	−3.55	15.99	−2.14	…	20.70	R	2.43
2	−0.23	1.28	11.56	−2.15	…	19.96	R	2.62
3	1.56	−1.23	3.67	−2.19	…	21.18	R	0.45
4	0.07	−1.74	7.72	−2.21	…	22.22	R	0.96
5	0.13	−4.76	2.93	−2.09	…	19.51	R	−0.95
…	…	…	…	…	…	…	…	…
96	−0.60	9.99	5.33	−1.70	…	17.81	L	3.26
97	2.19	3.00	14.33	−2.15	…	21.94	L	4.34
98	1.53	−6.26	−1.75	−1.83	…	17.86	L	−2.08
99	−0.24	−3.06	5.10	−2.25	…	22.75	L	−0.11
100	−1.03	−5.93	11.25	−2.07	…	21.36	L	0.56

まず前者のパタンについて R で実施する。**平均値や尺度得点のような計算結果を列に追加するには，$を用いる**。たとえば上図のように x1〜x4 の平均を x_mean として data に追加する場合，以下のようにする。

```
data$x_mean <- (data$x1 + data$x2 + data$x3 + data$x4) / 4
head(data, 5)
```

```
           x1         x2        x3         x4          y1         y2          y3
1 -0.56047565 -3.552033 15.994052 -2.143048 -2.8759343 13.606238  2.6734174
2 -0.23017749  1.284419 11.562065 -2.150538 -2.3195389 -6.131373  1.1288595
3  1.55870831 -1.233459  3.674275 -2.187708  7.1587933  5.960623  0.3340971
4  0.07050839 -1.737713  7.715970 -2.210503  0.3237004  7.701913 -0.7444706
5  0.12928774 -4.758093  2.928300 -2.087432  1.3171346 17.523205  1.0832154
        y4 z     x_mean
1 20.70227 R  2.4346237
2 19.96494 R  2.6164420
3 21.18398 R  0.4529540
4 22.22388 R  0.9595658
5 19.50961 R -0.9469842
```

head() で data の先頭 5 行まで表示すると，たしかに追加されているのが確認できる。

その他，**2 つのデータセットを列方向に追加するには cbind() を用いる**。

```
# データセットを分割
data_x <- data[1:4]# data から x1〜x4 だけ抜き出す
data_y <- data[5:8]# data から x1〜x4 だけ抜き出す

# データセットを列方向にくっつける
data_xy <- cbind(data_y, data_x)
head(data, 5)
```

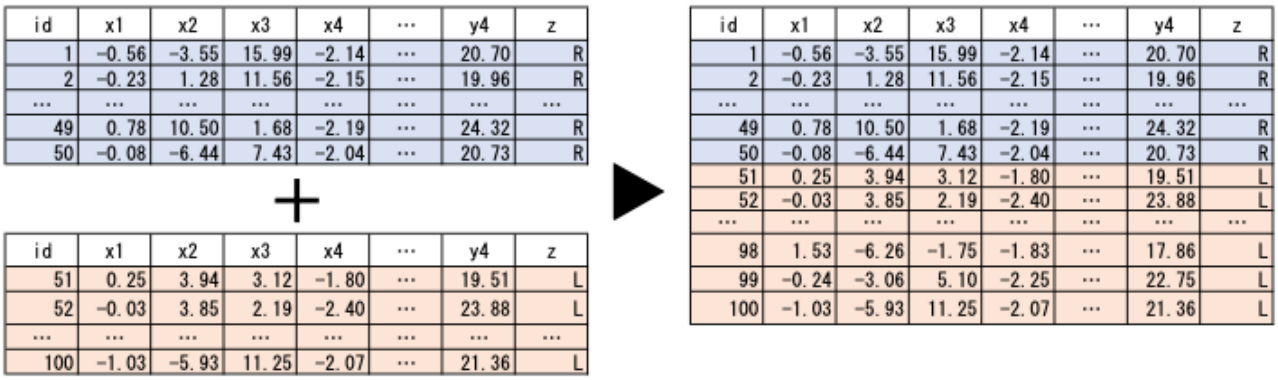
```
           x1         x2        x3         x4          y1         y2          y3
1 -0.56047565 -3.552033 15.994052 -2.143048 -2.8759343 13.606238  2.6734174
2 -0.23017749  1.284419 11.562065 -2.150538 -2.3195389 -6.131373  1.1288595
```

```
3  1.55870831 -1.233459  3.674275 -2.187708  7.1587933  5.960623  0.3340971
4  0.07050839 -1.737713  7.715970 -2.210503  0.3237004  7.701913 -0.7444706
5  0.12928774 -4.758093  2.928300 -2.087432  1.3171346 17.523205  1.0832154
        y4 z      x_mean
1 20.70227 R  2.4346237
2 19.96494 R  2.6164420
3 21.18398 R  0.4529540
4 22.22388 R  0.9595658
5 19.50961 R -0.9469842
```

cbind() は 2 つのデータセットを列方向に結合する関数であり, 引数で指定した 2 つのデータセットを横向きに並べて 1 つにまとめることができる。そのため, 行数が同じ必要がある。

6.5 行の追加

学校 A と学校 B で同じ質問紙を配布・回収したとき, 学校別にデータセットを作ることがある。このように 2 つに分かれたのデータセットを 1 つにまとめるときは, **行方向にデータを追加する**操作が必要になる。



id	x1	x2	x3	x4	…	y4	z
1	−0.56	−3.55	15.99	−2.14	…	20.70	R
2	−0.23	1.28	11.56	−2.15	…	19.96	R
…	…	…	…	…	…	…	…
49	0.78	10.50	1.68	−2.19	…	24.32	R
50	−0.08	−6.44	7.43	−2.04	…	20.73	R

+

id	x1	x2	x3	x4	…	y4	z
51	0.25	3.94	3.12	−1.80	…	19.51	L
52	−0.03	3.85	2.19	−2.40	…	23.88	L
…	…	…	…	…	…	…	…
100	−1.03	−5.93	11.25	−2.07	…	21.36	L

▶

id	x1	x2	x3	x4	…	y4	z
1	−0.56	−3.55	15.99	−2.14	…	20.70	R
2	−0.23	1.28	11.56	−2.15	…	19.96	R
…	…	…	…	…	…	…	…
49	0.78	10.50	1.68	−2.19	…	24.32	R
50	−0.08	−6.44	7.43	−2.04	…	20.73	R
51	0.25	3.94	3.12	−1.80	…	19.51	L
52	−0.03	3.85	2.19	−2.40	…	23.88	L
…	…	…	…	…	…	…	…
98	1.53	−6.26	−1.75	−1.83	…	17.86	L
99	−0.24	−3.06	5.10	−2.25	…	22.75	L
100	−1.03	−5.93	11.25	−2.07	…	21.36	L

上図のようにデータセットを結合するときは, rbind() を用いる。

```r
# z の値でデータセットを 2 つに分ける
data_R <- subset(data, data$z == "R")#z が R の行だけ抜き出す
data_L <- subset(data, data$z == "L")#z が L の行だけ抜き出す

# データセットを行方向にくっつける
data_RL <- rbind(data_L, data_R)
```

rbind() は 2 つのデータセットを行方向に結合する関数であり, **引数で指定した 2 つのデータセットを縦向きに並べて 1 つにまとめる**ことができる。そのため列の名前 (変数の名前=小さい方の箱の名前) が同じである必要がある。

ここを押さえれば OK

正規分布に従うデータは `rnorm()` で生成できる
変数名でデータセットの要素にアクセスするには`$`用いる
列番号を使ってデータセットの要素にアクセスするには**データセット**`[,`**行番号**`]`のように書く
`subset()` を使えば条件に一致した行を抽出できる
列を追加するには**データセット`$`新しい変数** `<-` **計算**のように書く
データセットを融合するには `cbind()` (横方向に融合) または `rbind()` (縦方向に融合) を使う

7　練習問題

12 × 13 と 12÷13 を計算しよう。

20 * 10 の計算結果を calc 変数に格納しよう。

psych パッケージをインストールしよう。

psych パッケージを読み込もう。

適当な場所に data.csv を保存して, その場所からデータを読み込んで atad という箱に入れてみよう。

group_a を 4 倍した値を group_a4 という箱に入れて，atad に追加してみよう。

atad という箱に入ったデータを dat4.csv という名前で書き出そう。

8　回答例

```r
# 12 × 13 と 12÷13 を計算しよう。
12 * 13
12 / 13

# 20 * 10 の計算結果を calc 変数に格納しよう。
calc <- 20 * 10

# psych パッケージをインストールしよう。
install.packages("psych")

# psych パッケージを読み込もう。
library(psych)

# 適当な場所に data.csv を保存して,その場所からデータを読み込んで atad という箱に入れてみよう。
atda <- read.csv("myfile/data.csv")

# group_a を 4 倍した値を group_a4 という箱に入れて，atad に追加してみよう。
atad$group_a4 <- atad$group_a * 4

# atad という箱に入ったデータを dat4.csv という名前で書き出そう。
write.csv(atad, "dat4.csv")
```