

文系のための R セミナー

第 1 回 R をはじめてみよう：ベクトル操作まで(6/23(火), 26(金))

R 概要

環境構築：Posit Cloud の登録

簡単な計算

変数とオブジェクト

ベクトル作成と簡単な操作

スクリプトファイルの保存

演習問題

第 2 回 データの可視化をしてみよう(7/3(金), 7(火))

環境構築：Posit Cloud の登録 5 分

クラスと関数 5 分

- R で扱うデータはすべてクラスに分類される
- 関数のネスト

データフレームの操作 15 分

- データフレームとは
- データフレームを概観する基本的な関数
- 列の操作
- 行の操作

データのビジュアライゼーション 20 分

- 散布図の基本的な書き方
- パッケージの導入
- ggplot2 を使った散布図の描き方

散布図とスクリプトファイルの保存(時間があれば)

演習問題(時間があれば)

R 概要

Rとは、統計解析やデータ可視化に特化したプログラミング言語で、無償で利用できます。現在人気のあるPythonなどの汎用プログラミング言語と比べると、アプリ開発などには向いていないものの、回帰分析や時系列分析、仮説検定など統計の基礎から高度な手法まで幅広く対応しています。エクセルでも基本的な回帰分析はできますが、Rを使うことで、テキストデータや空間データの分析も扱うことができます。またエクセルは表計算ソフトですので、直感的に使うことができますが、自分が行った作業の過程が少し分かりにくいところがあります。他方、Rはコードを書くという行為が挟まる分、手間はかかりますが、分析の過程が記録され作業の工程をあとから確認することができます。さらに同じ作業を別のデータ分析に再利用することができます。

	R	Excel	Python
使用タイミング	統計分析、学術研究	幅広い業務、教育統計	データ分析、Web・AI開発
費用	無料	有料	無料
特徴	・統計分析に特化 ・グラフや図の可視化が簡単 ・文章で残しやすい	・直感的な操作 ・マクロが使える	・ライブラリが豊富 ・機械学習やAI分野で主流
習得難易度	慣れるのに少し時間がかかる	簡単	直感的
汎用性	○ 統計分析のみ	△ 統計や機械学習は苦手	◎ 統計・機械学習・Web開発など幅広く対応
グラフ作成	◎ 簡単・キレイに作成可能	△ 簡単に作図可能。ただし微調整が難しい	○ Rに比べるとやや難しい

環境構築：Posit Cloud の登録

Rはインストールすれば単体でもすぐに分析することができます。しかし、コードが書きにくかったり、統計分析のためのデータの読み込みが大変だったりします。なので、一般的にはRStudioと呼ばれるソフトウェアをダウンロードして利用します。このようなソフトをIDE(Integrated Development Environment、統合開発環境)と呼びます。RStudioはRをより便利に使うためのソフトで、これから行う分析をより簡単に行うことができます。プログラミング言語と聞くと、親しみにくいイメージがある方も多いと思います。しかし、文法や用語は完璧に覚える必要はなく、適宜教科書等を参照することで慣れていくことが大切です。IDEは言語の学習と実践的な利用をサポートしてくれる環境であり、これを活用すればより効率的にRでデータ整理・分析を行うことができます。RStudioを用意するなど、Rを始めとしたプログラミング言語を自分のパソコンで利用できるようにする

ことを環境構築といいますが、使用しているパソコンやネット環境などによっては、とても時間がかかる場合があります。よって、今回は **Posit Cloud** (<https://posit.cloud>) というサービスを利用します。

Posit Cloud とは、RStudio をブラウザ上で利用することができるサービスで、RStudio を管理している団体が運営しています。サービスに多少制限はありますが、無料でも利用できるので、今回はこちらを利用します。まず、Posit Cloud のサイトにアクセスし、画面右上の「Log in」を選択します。

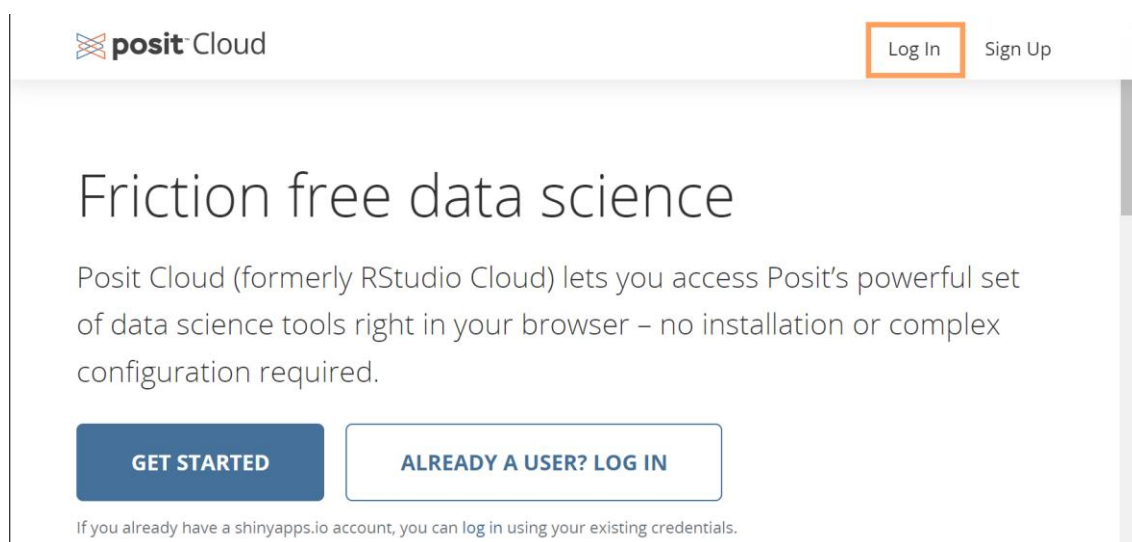


図 1. Posit Cloud のトップページ

次に「Sign Up」を選択し、画面上部に必要情報を入力するか、Google アカウントを持っている方は下の「Sign Up with Google」から手続きを進めてください。Google アカウント以外で登録(Sign up)を行った場合は、差出人”noreply@posit.cloud”から件名”Please verify your email address”というメールが届きます。メール本文中の”Verify your email”と書かれたリンクをクリックすると、承認完了です。

図 2. 利用登録(サインアップ)画面

登録が完了し、ログインができれば、図 3 のような画面が表示されるはずです。

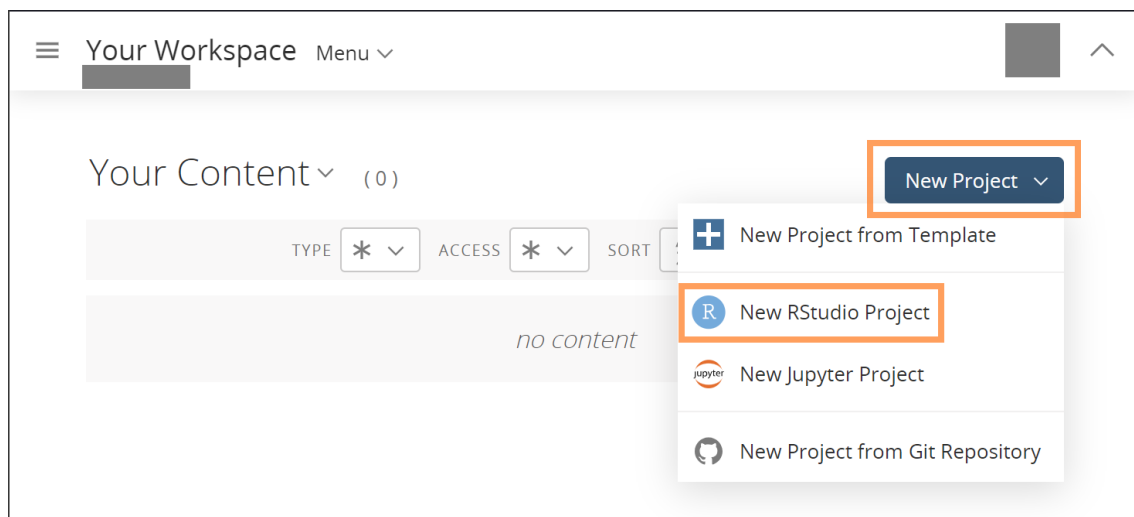


図 3. プロジェクト選択

画面右上の「New Project」を選択し、表示されたリストから「New RStudio Project」を選択してください。(次の画面の立ち上げまで多少時間がかかります)
その後、次のような画面が表示されるはずです。

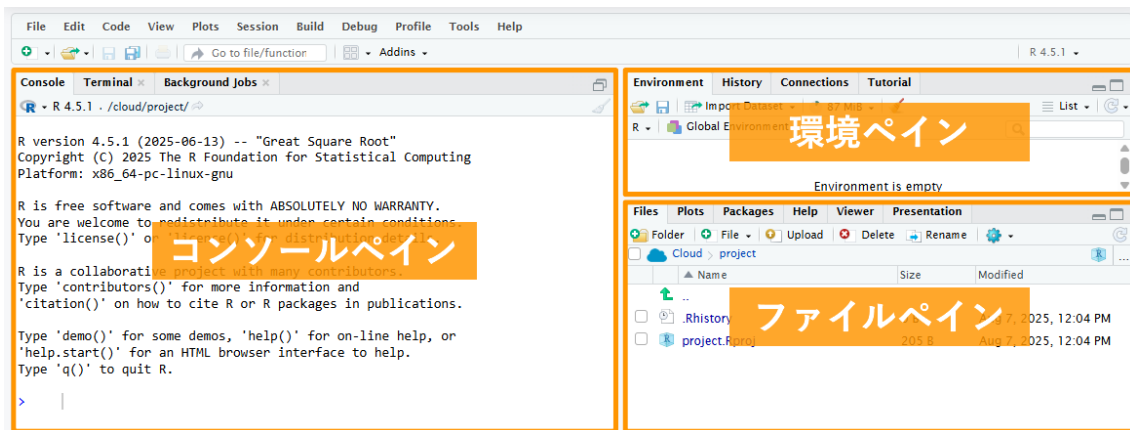


図 4. RStudio の基本画面

RStudio のウィンドウは、ペイン(**pane**)と呼ばれるいくつかの領域から成り立っています。それぞれの役割については都度、解説していきます。

これで今回の環境構築は完了です！

クラスと関数

R で扱う、「1000」という数値や「大阪」という文字列のような値(value)は、それぞれクラス(**class**)に分類されています。クラスごとに扱える関数が異なり、クラスには主に以下のような種類があります。

R の主なクラス

クラス名	説明
character	文字列型（テキストデータ）
numeric	数値型（小数を含む数値データ）
logical	論理型（TRUE または FALSE）
factor	カテゴリデータ型
data.frame	表形式のデータ型

クラスは `class()` 関数を用いることで確認することができます。

```
class("The University of Osaka")
## [1] "character"

class(122)
## [1] "numeric"

a <- "renshu"
class(a) # 変数の場合は中身のクラスが参照される。
## [1] "character"
```

関数(function)とは、入力を受け取り、それに対して何らかの処理を加え、その結果として出力を返す(return)仕組みのことです。関数の括弧内に入力する値を引数(ひきすう、argument)といいます。例えば、class()関数であれば、丸括弧内で受け取った値のクラスを返しますし、c()関数であれば、数値や文字列などの1つ以上の値を引数として受け取り、ベクトルという1つのオブジェクトとしてまとめて返します。このように、関数はさまざまな入力を受け取り、特定の処理を行って結果を返す便利なツールです。関数があらかじめよく使う作業をまとめてくれているおかげで、簡単にデータ分析できます。

それでは、データの要約を見るのに使用する基本的な関数をいくつか紹介します。

```
vec1 <- c(1, 2, 3, 4, 5)
length(vec1) # ベクトルの長さ、すなわちベクトル内の要素の数を返す。
## [1] 5
range(vec1) # ベクトル内の値の範囲(range)を返す。
## [1] 1 5
mean(vec1) # 平均値(mean)を返す。
## [1] 3
```

これらの関数に引数を渡す場合、関数側では引数に名前を割り当てており、この引数名を明示して引数を渡すことができます。

```
mean(x = vec1) # 左側が引数名。関数側であらかじめ名前が決まっている。
## [1] 3
length(x = vec1)
## [1] 5
```

引数が1つだけのときは省略しても問題ありませんが、後で出てくる関数では引数が複数あるものもあり、明示して書くと読みやすくなります。

また、関数はネスト(入れ子)して使うことができます。関数の処理は内側から進んでいきます。

```
length(c(1, 3, 5, 7, 9)) # 始めに、(1, 3, 5, 7, 9)のベクトルを作成(c()関数)し、
                           次にそのベクトルの長さを出力(length()関数)する。
## [1] 5
sqrt(length(c(1, 3, 5, 7, 9))) # 上記処理に加えて、最後にベクトルの長さの平方根
                               を取っている(sqrt()関数)。
## [1] 2.236068
```

データフレームの操作

それでは、エクセルで扱うようなデータフレームの簡単な操作とグラフ化を行いましょう。今回は、mtcars という、1974 年に発行されたアメリカの「Motor Trend」誌に記載された自動車の性能に関する情報をまとめたものを扱ってみましょう。

##	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
## Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
## Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
## Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
## Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
## Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
## Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
## Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
## Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
## Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4

mtcars の変数

変数名	内容
mpg	1 ガロンあたりの走行距離（マイル）
cyl	シリンダー数
disp	排気量（立方インチ）
hp	馬力
drat	リアアクスル比
wt	車体重量（1000 ポンド単位）
qsec	1/4 マイルの加速時間（秒）
vs	エンジン形状（0 = V 型，1 = 直列）
am	トランスミッション（0 = 自動，1 = 手動）
gear	ギア数
carb	キャブレター数

R は統計分析を目的としたプログラミング言語ですので、実は最初から統計分析を勉強するためのデータフレームが組み込まれています。プログラミング「言語」に、データセットが入っているのは奇妙な感じがしますが、あくまで R は統計分析をするための機能を提供するものなので、それを学習するためのデータも同梱されていますということです。ではデータフレームを変数に入れてオブジェクトにし、クラスを確かめてみましょう。オブ

ジェクト名は何でもいいですが、ここでは df(data frame の略)としましょう。

```
df <- mtcars # 変数 df に mtcars データセットを入れる。
```

```
class(df)
```

```
## [1] "data.frame"
```

クラスを確認すると「**data.frame(データフレーム)**」と表示されました。データフレームとは、「同じ長さの複数のベクトルを列として一つにまとめた**2次元データ構造**」です。要するに、エクセルやスプレッドシートで作った表みたいなものです。分析に移る前に、まず読み込んだデータフレームがどのようなものか、関数を使って概観しましょう。

dim()関数 : データフレームの行数と列数をベクトル形式で返す。(行は横のまとまり、列は縦のまとまり)

```
dim(df)
```

```
## [1] 32 11
```

head()関数 : データフレームの冒頭 6 行を返す。

```
head(df)
```

```
##           mpg  cyl  disp  hp drat    wt  qsec vs  am  gear  carb
## Mazda RX4      21.0   6  160 110 3.90 2.620 16.46  0   1    4    4
## Mazda RX4 Wag  21.0   6  160 110 3.90 2.875 17.02  0   1    4    4
## Datsun 710     22.8   4  108  93 3.85 2.320 18.61  1   1    4    1
## Hornet 4 Drive  21.4   6  258 110 3.08 3.215 19.44  1   0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0   0    3    2
## Valiant        18.1   6  225 105 2.76 3.460 20.22  1   0    3    1
```

summary()関数: 各列の最小値、第 1 四分位数、中央値、第 3 四分位数、最大値を返す。

```
summary(df)
```

```
##           mpg           cyl           disp           hp
##  Min.    :10.40  Min.    :4.000  Min.    : 71.1  Min.    : 52.0
## 1st Qu.:15.43  1st Qu.:4.000  1st Qu.:120.8  1st Qu.: 96.5
##  Median :19.20  Median :6.000  Median :196.3  Median :123.0
##   Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
## 3rd Qu.:22.80  3rd Qu.:8.000  3rd Qu.:326.0  3rd Qu.:180.0
```

```
## Max. :33.90 Max. :8.000 Max. :472.0 Max. :335.0
##      drat      wt      qsec      vs
## Min. :2.760 Min. :1.513 Min. :14.50 Min. :0.0000
## 1st Qu.:3.080 1st Qu.:2.581 1st Qu.:16.89 1st Qu.:0.0000
## Median :3.695 Median :3.325 Median :17.71 Median :0.0000
## Mean   :3.597 Mean   :3.217 Mean   :17.85 Mean   :0.4375
## 3rd Qu.:3.920 3rd Qu.:3.610 3rd Qu.:18.90 3rd Qu.:1.0000
## Max.   :4.930 Max.   :5.424 Max.   :22.90 Max.   :1.0000
##      am      gear      carb
## Min. :0.0000 Min. :3.000 Min. :1.000
## 1st Qu.:0.0000 1st Qu.:3.000 1st Qu.:2.000
## Median :0.0000 Median :4.000 Median :2.000
## Mean   :0.4062 Mean   :3.688 Mean   :2.812
## 3rd Qu.:1.0000 3rd Qu.:4.000 3rd Qu.:4.000
## Max.   :1.0000 Max.   :5.000 Max.   :8.000
```

str()関数：データフレームの行数、列数、列ごとのクラスと含まれるデータの冒頭 10 行を示す。

str(df)

```
## 'data.frame': 32 obs. of 11 variables:
## $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num 160 160 108 258 360 ...
## $ hp : num 110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
## $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num 16.5 17 18.6 19.4 17 ...
## $ vs : num 0 0 1 1 0 1 0 1 1 1 ...
## $ am : num 1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num 4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num 4 4 1 1 2 1 4 2 2 4 ...
```

obs.(observations):「観測値」という意味でここではデータフレームの行数を示します。

variables:「変数」という意味で、データフレームの列数を指します。

実際にデータ分析をするときにも、以上のような関数を用いて、分析したいデータフレ

ームが本当に読み込めているか、数値クラスが文字列クラスとして読み込まれていないかなどを確認めます。関数を使う以外に、エクセルやスプレッドシートのようにデータフレーム全体を確認する方法もあります。

`View(df)` # 「V」は大文字! 画面右上の *environment* ペインから直接見ることもできる。

先述したように、データフレームはベクトルを列としてまとめた集合であり、今回読み込んだデータフレームは 11 個のベクトルが 1 つの表形式にまとめられたものです。ここで、\$演算子を使うことで、データフレーム内の個別の変数(列)を抜き出し、ベクトルとして扱うことができます。それでは、データフレームから mpg 列(mile per gallon, 1 ガロン当たり何マイル進めるか = 燃費)を抜き出し、1 リットル当たり何キロメートル進めるかという値に変換しましょう。

データフレーム名\$変数名

```
df$mpg
## [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 1
0.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 1
9.7
## [31] 15.0 21.4
mpg <- df$mpg
kmpl <- (mpg * 1.60934) / 3.78541 # kilometers per liter
kmpl
## [1] 8.928000 8.928000 9.693257 9.098057 7.950171 7.695086 6.079543
## [8] 10.373486 9.693257 8.162743 7.567543 6.972343 7.354971 6.462171
## [15] 4.421486 4.421486 6.249600 13.774628 12.924343 14.412343 9.140571
## [22] 6.589714 6.462171 5.654400 8.162743 11.606400 11.053714 12.924343
## [29] 6.717257 8.375314 6.377143 9.098057
```

それでは作成した kmpl ベクトルを df に追加してみましょう。列を抜き出した際と同じように、新たに列を追加するには、ベクトル名を指定し、そこに代入します。コードではベクトル名と同じ「kmpl」としていますが、他のベクトル名でも構いません。ただし、既存の列名にすると上書きしてしまうので、気を付けてください。

```
df$kmpl <- kmpl
str(df)
## 'data.frame': 32 obs. of 12 variables:
## $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...
```

```
## $ disp: num 160 160 108 258 360 ...
## $ hp : num 110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
## $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num 16.5 17 18.6 19.4 17 ...
## $ vs : num 0 0 1 1 0 1 0 1 1 1 ...
## $ am : num 1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num 4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num 4 4 1 1 2 1 4 2 2 4 ...
## $ kmp1: num 8.93 8.93 9.69 9.1 7.95 ...
```

また行を抜き出す場合、df[抜き出したい行,]のように指定します。

df[1,] *#1 行目を抜き出す。*

```
##           mpg cyl disp  hp drat   wt  qsec vs am gear carb  kmp1
## Mazda RX4  21    6  160 110   3.9 2.62 16.46  0  1    4    4 8.928
```

df[1] *# コンマを入れ忘れると列の抜き出しになってしまいます。*

```
##           mpg
## Mazda RX4    21.0
## Mazda RX4 Wag 21.0
## Datsun 710    22.8
## Hornet 4 Drive 21.4
## Hornet Sportabout 18.7
## Valiant      18.1
## Duster 360   14.3
## Merc 240D    24.4
## Merc 230     22.8
## Merc 280     19.2
## Merc 280C    17.8
## Merc 450SE   16.4
## Merc 450SL   17.3
## Merc 450SLC  15.2
## Cadillac Fleetwood 10.4
## Lincoln Continental 10.4
## Chrysler Imperial 14.7
## Fiat 128     32.4
## Honda Civic  30.4
```

```
## Toyota Corolla      33.9
## Toyota Corona       21.5
## Dodge Challenger    15.5
## AMC Javelin         15.2
## Camaro Z28          13.3
## Pontiac Firebird    19.2
## Fiat X1-9           27.3
## Porsche 914-2       26.0
## Lotus Europa        30.4
## Ford Pantera L      15.8
## Ferrari Dino        19.7
## Maserati Bora       15.0
## Volvo 142E          21.4
```

df[2:4,] # 連続した行を抜き出す場合は「:(コロン)」で範囲指定します。

```
##                mpg cyl disp  hp drat   wt  qsec vs am gear carb    kmp1
## Mazda RX4 Wag  21.0   6  160 110 3.90 2.875 17.02  0  1    4    4 8.928000
## Datsun 710      22.8   4  108  93 3.85 2.320 18.61  1  1    4    1 9.693257
## Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1 9.098057
```

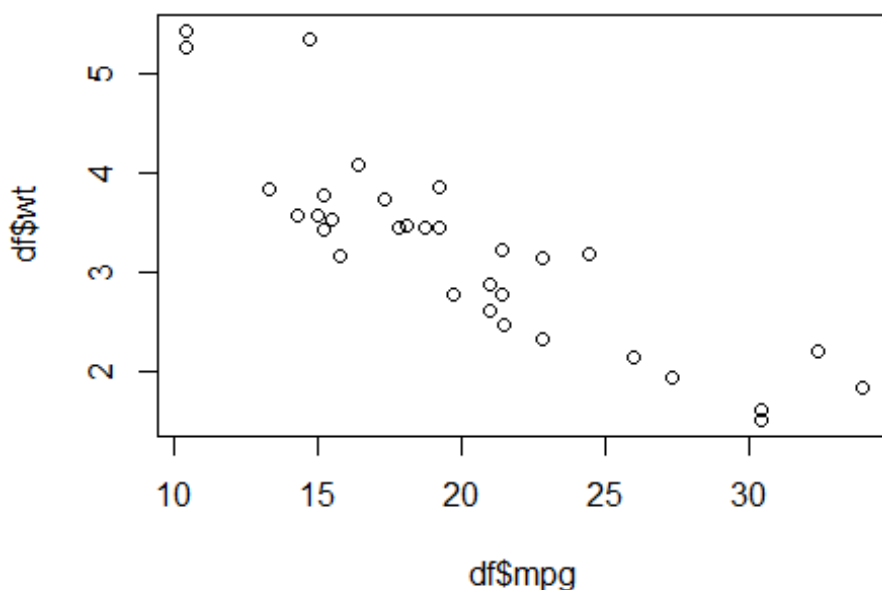
df[df\$cyl == 6,] # 今回の講習内では扱いませんが、条件を指定して抜き出すことも可能です。

```
##                mpg cyl  disp  hp drat   wt  qsec vs am gear carb    kmp1
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4 8.928000
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4 8.928000
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1 9.098057
## Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1 7.695086
## Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4 8.162743
## Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4 7.567543
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6 8.375314
```

データのビジュアライゼーション

では最後に、このデータフレームを使って散布図を作成してみましょう！下のコードを入力してみてください。

```
plot(x = df$mpg, y = df$wt) # コンマで区切るよう注意してください。
```



これは散布図(scatter plot)で、x 軸に燃費(1 ガロンあたりの距離(マイル))、y 軸に車体重量(単位：1000 ポンド)の値が入っています。ここまで関数に入れる引数は基本的に 1 種類だけでしたが、plot()関数のように複数種類の引数を受け取ることができるものがあります¹。引数が複数ある場合、上記コードのように書くことで、「x(軸)には mpg ベクトル、y(軸)には wt ベクトル」と、渡した引数がどこに代入したいものかを明示して渡すことができます。引数を明示するのは面倒かもしれませんが、結果的にコード作成しているときや後から読むときに混乱せずに済みます。

同じように散布図を描くにしても、より便利に、より柔軟に描く方法が R には備わっています。R では、これまでに使ってきた基本機能以外はパッケージという形で

CRAN(Comprehensive R Archive Network)というサイトに置いてあります。また、パッケージとは、あるテーマに沿った関数を箱詰めになっているセットのことだと考えてください。install.packages()関数を使うことで、引数に入力したパッケージをそのサイトから

¹ c()関数は複数の要素を受け取りましたが、いずれも「ベクトルの要素」という意味では 1 種類です。対して、plot()関数は x 軸のデータ、y 軸のデータ、グラフのタイトルのように複数種類の引数を受け取ることができます。

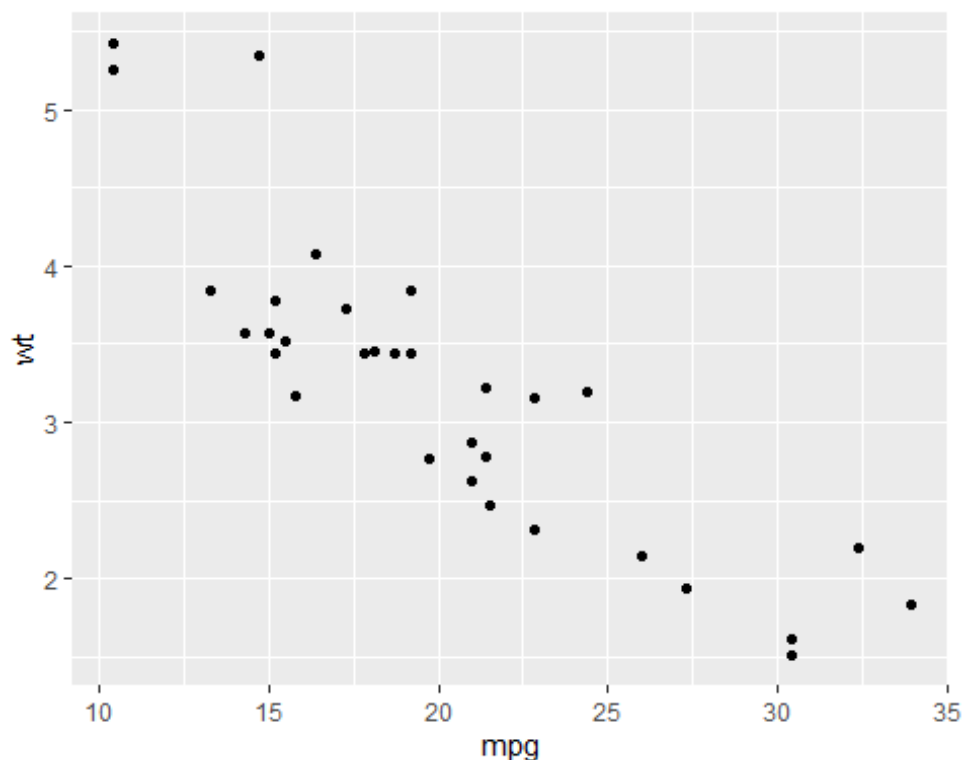
取ってきてインストールすることができます。そして **library()**関数でパッケージを使えるように読み込むことができます。パッケージを読み込むと、そのパッケージに収録されている関数を R セッションで使うことができるようになります。なお、**install.packages()**関数を用いてのパッケージダウンロードは、基本的に一度行えばそれ以降はする必要がありません。しかし、**library()**関数でのパッケージの呼び出しは、各 R セッションにて行う必要があるため、注意が必要です。

ここでは、**ggplot2** という、グラフを書くためのパッケージを使ってみます。

```
# パッケージのインストール。パッケージ名は引用符で囲む。  
# 「Warning～」となにやら不穏当な表示が出ますが、単なる注意喚起ですので、気にしないで大丈夫です。  
install.packages("ggplot2")  
library(ggplot2) # パッケージの読み込み。クォーテーションで囲まなくて良い  
##  
## Attaching package: 'ggplot2'  
## The following object is masked _by_ '.GlobalEnv':  
##  
##      mpg
```

ggplot2 を使って、もう一度散布図を書いてみましょう。

```
# “scatter1” という名前のオブジェクトとして散布図を保存します。  
# ggplot()関数内の引数や関数のネストが複雑になっているので、注意して入力してください。  
# 見やすさのために「+」で改行しています。「+」まで入力した後、Enter を押すことで、  
# 自動的に少し字下げ(インデント)した状態でコードのつづき が入力できるはずです。  
scatter1 <- ggplot(data = df, mapping = aes(x = mpg, y = wt)) +  
  geom_point()  
  
# 保存したグラフを表示する  
print(scatter1)
```



先ほどの散布図のコードよりも長くなりましたが、これが `ggplot2` で散布図を書くための最低限のコードです。1 つずつ解説します。

まず、**`ggplot()`**関数はグラフを書くためのベースを用意する関数です。絵のキャンバスを用意するようなイメージです。`data` 引数はグラフ描画対象のデータセットを指定します。`mapping` 引数では指定したデータセット内の引数で `x` 軸と `y` 軸に割り当てる列を選びます。その際、**`aes()`**関数(**`aesthetic`**、美的の意)を用いて指定します。`ggplot()`関数でグラフの基本情報が設定出来たら、次にどのように描画するかを決めます。今回は散布図を描きたいので、**`geom_point()`**関数を用いました。このとき、両関数を `+` でつなぎます。これ以外にもヒストグラムを描く **`geom_histogram()`**関数や、折れ線グラフを描くための、**`geom_bar()`**関数といった関数など様々用意されています。

`ggplot()`関数を使うと、作成するグラフをオブジェクトとして保存することが可能です。保存した図を表示するには **`print()`**関数を使います。

このグラフをもう少し分析チックにするために、次に車のシリンダー数(cyl)毎に点を塗り分けてみましょう。次のコードのように、aes()関数に color 引数を追加し、塗り分けの基準とする cyl 変数を追加します。ただし、このままだと cyl 変数は連続的に変化する変数とみなされてしまい、データセットに存在しないシリンダー数 7 まで凡例に表示されてしまいます(左図)。ですので、**factor()**関数を使い、変数を離散的な値(カテゴリー型)に変換しましょう(右図)。

左図(scatter2)

```
scatter2 <- ggplot(data = df,
                    mapping = aes(x = wt, y = mpg, color = cyl)) +
  geom_point()
```

```
print(scatter2)
```

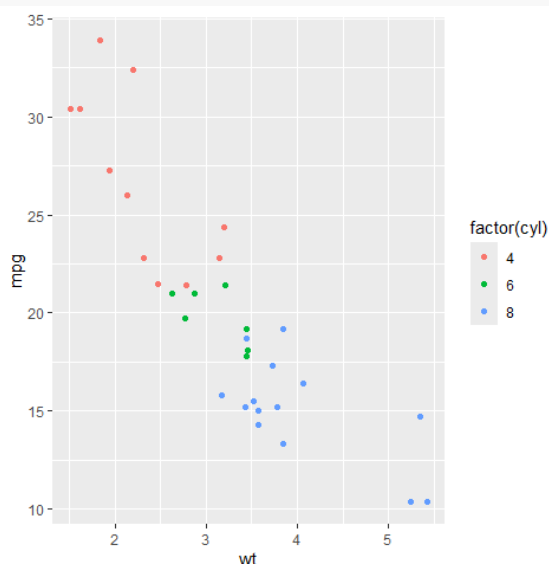
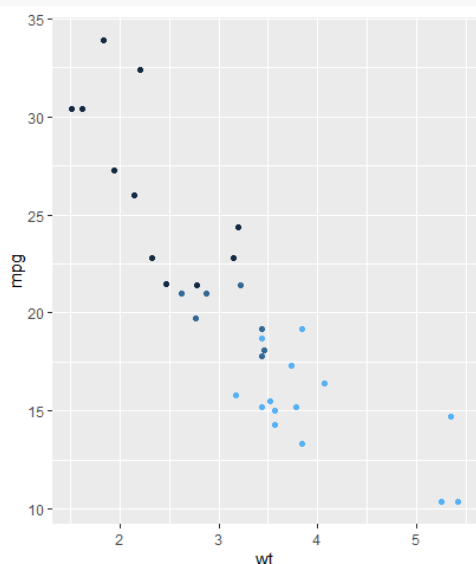
右図(scatter3)

左図のコードの color 引数が cyl→factor(cyl)となっています。

コードのネストが深くなっているので入力ミスが無いか注意しましょう。

```
scatter3 <- ggplot(data = df,
                    mapping = aes(x = wt, y = mpg, color = factor(cyl))) +
  geom_point()
```

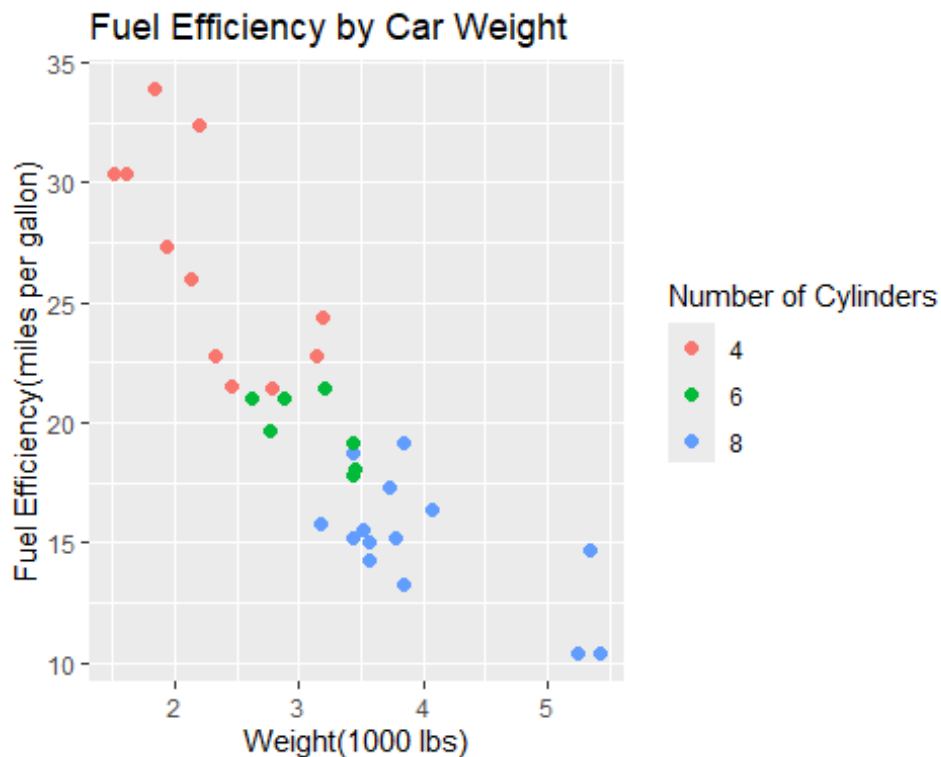
```
print(scatter3)
```



ここに更に手を加えて、グラフを分析結果として見やすくまとめましょう。まず、散布図の点が少し小さくて見づらいので、`geom_point()`関数で `size` 引数を指定して、点を少し大きくします。次に、散布図自体のタイトルと、x 軸と y 軸のラベルを付けて、グラフの内容がすぐ分かるようにします。また、凡例部分の名称も分かりやすいものにしましょう。これらの作業は、**`labs()`**関数を用いることで実行できます。

```
scatter4 <- ggplot(data = df,
                    mapping = aes(x = wt, y = mpg, color = factor(cyl))) +
  geom_point(size = 2) +
  labs(title = "Fuel Efficiency by Car Weight", # グラフのタイトル
       x = "Weight(1000 lbs)",                # x 軸のラベル
       y = "Fuel Efficiency(miles per gallon)", # y 軸のラベル
       color = "Number of Cylinders")         # 凡例のラベル

print(scatter4)
```



このように、`ggplot2` パッケージにはグラフをキレイに描くための多くの機能が備わっています。ちなみに、同じようなものを先ほど散布図を作ったデフォルトの R の機能でやろうとすると、次のようなやや煩雑なコードになってしまいます。

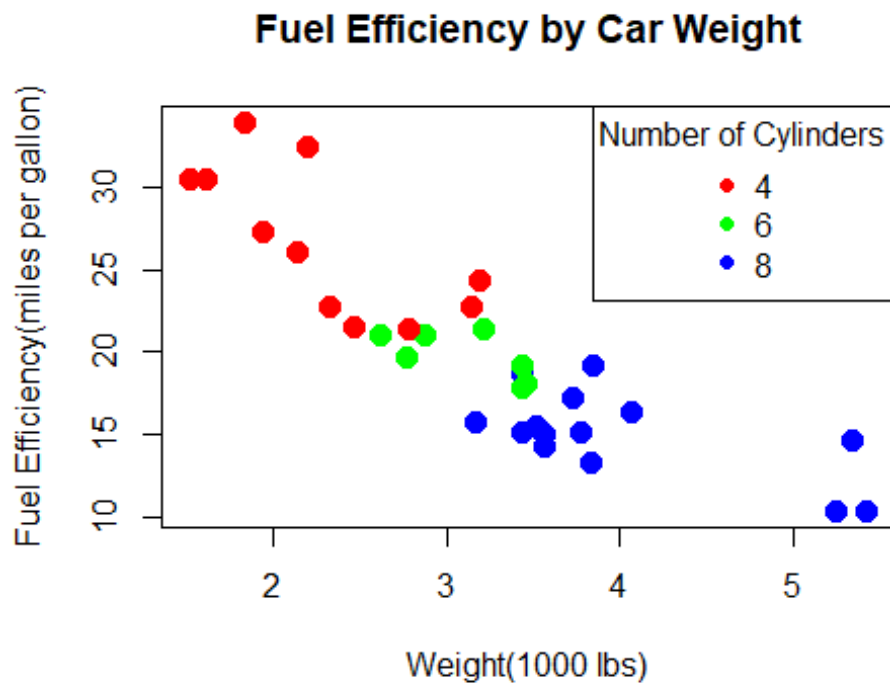
```

df <- mtcars
colors <- c("red", "green", "blue") # シリンダー数ごとの色を指定
cyl_factors <- as.factor(df$cyl)    # シリンダー数を因子型に変換
color_mapping <- colors[cyl_factors]

# 散布図の作成
plot(x = df$wt, y = df$mpg,
     col = color_mapping,           # 色をシリンダー数に基づいて設定
     pch = 16,                     # 点の形状を設定
     cex = 1.5,                    # 点のサイズ
     main = "Fuel Efficiency by Car Weight", # タイトル
     xlab = "Weight(1000 lbs)",      # X軸ラベル
     ylab = "Fuel Efficiency(miles per gallon)") # Y軸ラベル

# 凡例を追加
legend("topright",
     legend = levels(cyl_factors), # シリンダー数のラベル
     col = colors,                 # 色を対応付け
     pch = 16,                     # 点の形状
     title = "Number of Cylinders")

```



散布図とスクリプトファイルの保存(時間があれば)

作成した綺麗な散布図(scatter4)を保存しましょう。作成したオブジェクトは `ggsave()` 関数を用いることで保存できます。ggsave 関数は拡張子(.png, .pdf, .jpg など)を変えるだけで、自動でその形式に合わせて保存してくれる便利な関数です。

```
ggsave(filename = "my_plot.png", # ファイル名と保存形式
        plot = scatter4) # どのグラフを保存するかを指定
```

最後にスクリプトファイルも保存しましょう。最初に述べたように、Rのメリットの1つは分析した手順をスクリプトファイルとして保存し、それを再利用できることです。のちに見返したときにわかるようにコメントをつけて残しておきましょう。ファイルの先頭に次の内容を書き込んでおくと、見返したときに便利です。

- ファイル名
- ファイルを作った目的、分析内容
- 作成者
- 作成日
- 最終更新日(または更新日のリスト)と更新者の名前



入力出来たら Ctrl(Command) + S 保存し、実際のファイル名とコメントのファイル名を合わせておきましょう。今回、Posit Cloud で保存したファイルはクラウド上にあるので、今回作った散布図やファイルが欲しい場合は、ファイルペイン(右下の **Files**)でファイルにチェックを入れ、「(歯車)More」→「Export…」でダウンロードしてください。

演習問題(時間があれば)

問 1.

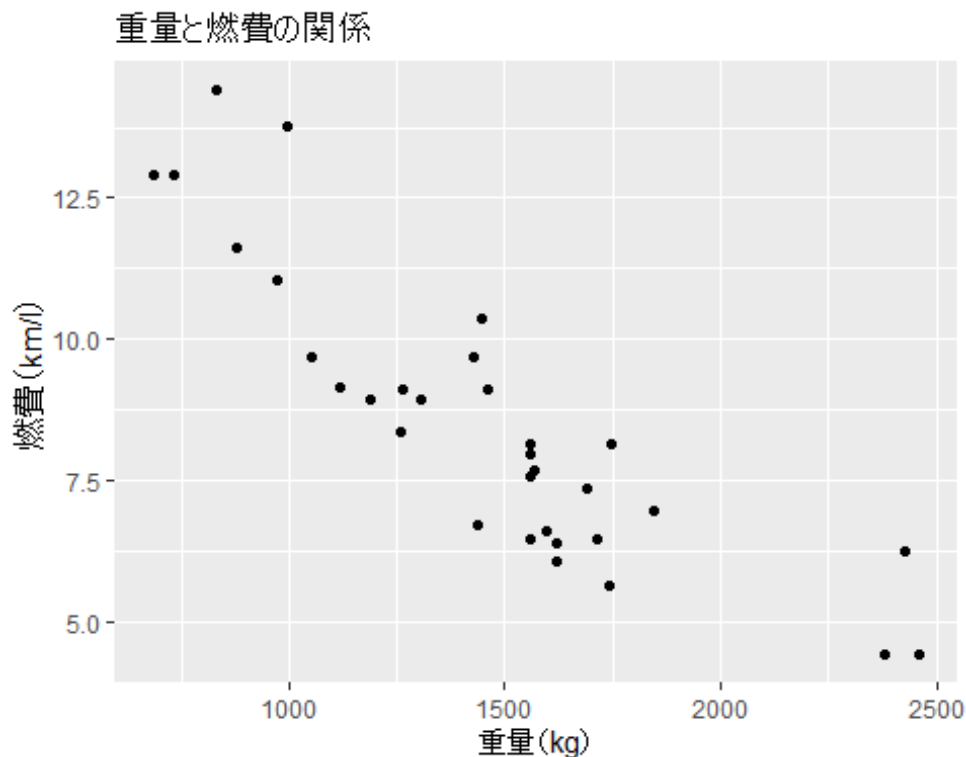
df の wt 列は 1000 ポンド単位です。これをキログラム (1 ポンド $\doteq$ 0.453592kg) に変換した列(wt_kg)を作成しましょう。

```
df$wt_kg <- df$wt * 1000 * 0.453592
```

問 2.

問 1 で作成した wt_kg 列と講習内で作成した kmp1 列を使って散布図を作成してみましょう。その際、下記コードのようにデータフレームや列の指定が ggplot()関数でなくてもできることを確認しましょう。

```
# 先ほど data の指定などを ggplot()関数で行いましたが、  
# 下記のように、geom_point()関数で行うこともできます。  
# グラフを重ねて描画する時にこういった指定が生きてきます  
scatter_p2 <- ggplot() +  
  geom_point(data = df, aes(x = wt_kg, y = kmp1)) +  
  labs(title = "重量と燃費の関係", # 全角と半角の切り替えに注意  
        x = "重量 (kg)",  
        y = "燃費 (km/l)")  
print(scatter_p2)
```



問 3.

次のコードを書いて、ヒストグラムを描いてみましょう。ヒストグラムとはデータをいくつかの区間(bin)に分けて、その区間に含まれるデータの個数を棒グラフで表したものです。

```
histogram_p3 <- ggplot() +  
  geom_histogram(data = df, mapping = aes(x = mpg))  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.  
  
print(histogram_p3)
```

