Chapter02. R による記述統計と可視化

池田龍也

2025-03-10

目次

1　記述統計

1.1　使用するデータ

Chapter 01 で生成したデータセットを分析に使用する。

```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
  )
```

1.2 平均値,中央値,最頻値

ここからはRで統計解析を実施する方法を述べる。ただし本稿の目標はRを使えるようになることであるため，**統計そのものについての詳しい解説は行わない**。統計については日本語で読める優れたテキストがあるため,そちらを参照してほしい。また,筆者の所属コースではM1後期に臨床心理統計研究法という授業があるので,本コースの院生はこの授業で理論面を学んでほしい (逆にこの授業ではソフトの使い方はほとんど解説しない)。

さて,代表値の代表例といえば平均値,中央値,最頻値である。平均値は `mean()`,中央値は `median()` で求められる。最頻値を求める方法はいくつかあり,そのなかでも `modeest` パッケージの `mfv()` を使用するのが簡単である。

```
mean(data$x1)
```

```
[1] 0.09040591
```

```
median(data$x1)
```

```
[1] 0.06175631
```

```
library(modeest)
x <- c(0, 0, 1, 1, 1, 3, 2, 1, 4, 4)
mfv(x)
```

```
[1] 1
```

もし箱の中に欠測値があるとエラーが出て計算できない。欠測値を除いて平均値などを計算するときは引数 `na.rm = TRUE` または `na.rm = T` とする。たとえば平均値であれば `mean(data$x1, na.rm = T)` と書く。

行方向の平均値を求める場合は `apply()` を用いて

```
apply(data[5:8], 1, mean)
```

```
  [1]   8.52649672   3.16072085   8.65937396   7.37625477   9.85829111
  [6]   6.92723804  10.13092226  10.79541940   5.64006080  -1.04082597
 [11]   9.16088573   1.95040404  15.51522142   5.77245164   2.25669602
 [16]   4.49693078   5.25573737   6.35356931   8.91497305   9.34094765
 [21]   3.08397930   8.75015587   7.33244020   5.16368564  -4.77895745
```

```
 [26]   6.26475859   6.02978330   4.33981837   7.93516967   7.18724490
 [31]  -1.46029547   2.80599183   5.79263242   7.69126699  16.29652494
 [36]   0.76696153  14.37567233   2.12222137  -5.76349384  12.21204468
 [41]  -0.07219661   5.94566294  11.73154302  15.21358048  14.65445172
 [46]   7.41960940  12.14316093   1.39543521  -3.12331997  11.39933177
 [51]   1.41733998   2.23572182   4.24761156  11.76266353   6.80236548
 [56]   6.97742742   1.07651294   7.64460109   1.30487542   5.54051281
 [61]   0.73389209   9.91295531  11.38171277 -12.86001834   4.70752835
 [66]   4.88923578   2.98569339   8.34596964   3.21891595   7.01020843
 [71]   4.67304671   2.24069585   6.55769509  -7.51259636   7.75082682
 [76]  12.48532664   5.41573207   1.51873350   2.85132035   7.84794732
 [81]  10.44806107  -0.18781364   6.22630792  10.36394839   5.13598514
 [86]   5.73933968   1.20959418   5.31006591   1.13831368   7.79799191
 [91]   5.11107450   9.18018131   5.70477302   8.70163186  13.39507348
 [96]  -6.16428084   5.54621152  12.12537554   9.06841470  10.46427044
```

とすると簡潔である。

1.3 分散,標準偏差

i 番目のデータ x を x_i と書き, x の平均値を $\bar{x}$ とする。サンプルサイズが N のとき, 標本分散 s^2 は

$$s^2 = \frac{\Sigma(x_i - \bar{x})^2}{N}$$

であり, 標本標準偏差 s は

$$s = \sqrt{s^2} = \sqrt{\frac{\Sigma(x_i - \bar{x})^2}{N}}$$

である。ただし R では分散に不偏分散 u^2 を用い, 標準偏差に不偏分散の平方根 u (不偏標準偏差) を用いている。式は以下の通りである。

$$u^2 = \frac{\Sigma(x_i - \bar{x})^2}{N-1}$$

$$u = \sqrt{u^2} = \sqrt{\frac{\Sigma(x_i - \bar{x})^2}{N-1}}$$

R で不偏分散と不偏分散の平方根を計算するには, それぞれ `var()` と `sd()` を用いる。

```r
# 不偏分散の計算
var(data$x1)
```

```
[1] 0.8332328
```

```r
# 不偏分散の平方根（不偏標準偏差）の計算
sd(data$x1)
```

```
[1] 0.9128159
```

標本分散と標本標準偏差を計算したい場合は, 以下のように計算する。

```r
N <- length(data$x1)# データの長さ=サンプルサイズを求める
uu <- var(data$x1)# 不偏分散を求める
sum_dim <- uu * (N - 1)# 不偏分散から偏差の総和を求める
ss <- sum_dim / N# 偏差の総和をサンプルサイズで割って標本分散を求める
s <- sqrt(ss)# 標本分散の平方根=標本標準偏差を求める
ss
```

```
[1] 0.8249005
```

```r
s
```

```
[1] 0.9082403
```

`length()` はデータの長さを計算する関数である。`data` は 100 個データがあるので, `length(data$x1)` は 100 となる。`x <- c(0, 1, 2, 1, 1, 3, 1)` ならデータは 7 個なので, `length(x)` を実行すると 7 が計算結果として返ってくる。

`sqrt()` は平方根を計算する関数である。`sqrt(2)` なら中学で暗記した通り $\sqrt{2} = 1.41421356\ldots$ であり, これも中学で暗記した通り `sqrt(3)` なら $\sqrt{3} = 1.7320508\ldots$ である。

人力で平方根の計算しなくて良いので計算自体は簡単である。ただいくつも変数がある場合, この処理を繰り返すのは面倒である。そこでどうしても標本分散と標本標準偏差を用いたい場合は, 処理を関数としてまとめると楽である。

```r
var2 <- function(x){
  N = length(x)
  sum_dim = sum((x - mean(x))^2)
  ss = sum_dim / N
```

```
  return(ss)
}

var2(data$x1)
```

```
[1] 0.8249005
```

```
sd2 <- function(x){
  N = length(x)
  sum_dim = sum((x - mean(x))^2)
  ss = sum_dim / N
  s = sqrt(ss)
  return(s)
}

sd2(data$x1)
```

```
[1] 0.9082403
```

ここでは標本分散と標本標準偏差の式に合わせて関数を書いている。`sum()` は総和を求める関数であり, 偏差の総和を求めている。

1.4 標準誤差

標準誤差 SE は不偏分散 u^2 とサンプルサイズ N を用いて

$$SE = \sqrt{\frac{u^2}{N}} = \frac{u}{\sqrt{N}}$$

と計算する。なぜか R には標準誤差を計算する関数が標準搭載されていないので, 自作関数を作るか `plotrix` パッケージの `std.error()` 関数を用いる。ここでは関数を自作して計算する。

```
se <- function(x){
  se = sd(x) / length(x)
  return(se)
}

se(data$x1)
```

```
[1] 0.009128159
```

1.5 四分位数,四分位範囲,四分位偏差

四分位数の計算には quantile(), 四分位範囲の計算には IQR() を用いる。四分位偏差は $\frac{IQR}{2}$ なので, IQR() の結果を 2 で割れば求められる。面倒なら処理を関数にしてしまえば良い。

```
# 四分位数
quantile(data$x1)
```

```
         0%         25%         50%         75%        100%
-2.30916888 -0.49385424  0.06175631  0.69181917  2.18733299
```

```
# 四分位範囲
IQR(data$x1)
```

```
[1] 1.185673
```

```
# 四分位偏差
IQR(data$x1) / 2
```

```
[1] 0.5928367
```

```
# 四分位偏差を求める関数
QD <- function(x){
  qd <- IQR(x) / 2
  return(qd)
}
QD(data$x1)
```

```
[1] 0.5928367
```

1.6 summary() と psych::desrribe()

多くの尺度得点や因子得点を使っている場合, 関数を使って 1 つ 1 つ計算するのは効率が悪い。R にはデータセットに含まれる変数の記述統計を一度に計算する summary() が標準で組み込まれていて, この関数を使うと最小値, 四分位数, 平均値, 最大値を計算できる。使い方はシンプルであ

り,引数にデータセットを指定するだけである。変数名を入力すれば,その変数の記述統計を計算することもできる。

```
summary(data)
```

```
       x1                  x2                  x3                x4
 Min.   :-2.30917   Min.   :-10.2662   Min.   :-3.783   Min.   :-2.493
 1st Qu.:-0.49385   1st Qu.: -4.0055   1st Qu.: 2.343   1st Qu.:-2.146
 Median : 0.06176   Median : -1.1292   Median : 5.180   Median :-2.001
 Mean   : 0.09041   Mean   : -0.5377   Mean   : 5.602   Mean   :-2.007
 3rd Qu.: 0.69182   3rd Qu.:  2.3392   3rd Qu.: 8.818   3rd Qu.:-1.862
 Max.   : 2.18733   Max.   : 16.2052   Max.   :16.465   Max.   :-1.486
       y1                  y2                 y3                y4
 Min.   :-11.8273   Min.   :-63.337   Min.   :-2.1015   Min.   :13.50
 1st Qu.: -2.8641   1st Qu.:-10.338   1st Qu.:-0.5160   1st Qu.:18.91
 Median :  0.7766   Median :  4.095   Median : 0.3302   Median :20.31
 Mean   :  0.5579   Mean   :  2.109   Mean   : 0.4106   Mean   :20.18
 3rd Qu.:  3.4093   3rd Qu.: 15.090   3rd Qu.: 1.2192   3rd Qu.:21.44
 Max.   : 11.3754   Max.   : 43.011   Max.   : 2.9914   Max.   :24.63
      z
 Length:100
 Class :character
 Mode  :character
```

```
summary(data$x1)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-2.30917 -0.49385  0.06176  0.09041  0.69182  2.18733
```

ただ，summary() で出力される結果は論文や学会発表で使用しづらい。そこで psych パッケージの describe() を用いれば,より見慣れた形式で記述統計を一度に計算できる。

```
library(psych)
describe(data)#記述統計を計算する
```

```
   vars   n  mean    sd median trimmed   mad    min   max  range  skew kurtosis
x1    1 100  0.09  0.91   0.06    0.08  0.89  -2.31  2.19   4.50  0.06    -
0.22
x2    2 100 -0.54  4.83  -1.13   -0.83  4.83 -10.27 16.21  26.47  0.63     0.58
x3    3 100  5.60  4.75   5.18    5.46  4.54  -3.78 16.47  20.25  0.32    -
0.59
x4    4 100 -2.01  0.21  -2.00   -2.01  0.21  -2.49 -1.49   1.01 -0.03    -
0.31
y1    5 100  0.56  4.48   0.78    0.57  4.63 -11.83 11.38  23.20 -0.05    -
0.11
y2    6 100  2.11 19.25   4.09    3.17 19.98 -63.34 43.01 106.35 -0.58     0.50
y3    7 100  0.41  1.14   0.33    0.40  1.28  -2.10  2.99   5.09  0.12    -
0.75
y4    8 100 20.18  2.15  20.31   20.25  1.99  13.50 24.63  11.13 -0.33     0.12
z*    9 100  1.50  0.50   1.50    1.50  0.74   1.00  2.00   1.00  0.00    -
2.02
     se
x1 0.09
x2 0.48
x3 0.47
x4 0.02
y1 0.45
y2 1.93
y3 0.11
y4 0.22
z* 0.05
```

出力の内容のうち, 心理学でよく用いるのは

- `n`:サンプルサイズ
- `mean`:平均値 (算術平均)
- `sd`:標準偏差
- `median`:中央値
- `min`/ `max`/ `range`:最小値, 最大値, 範囲
- `skew`/ `kurtosis`:歪度, 尖度
- `se`:標準誤差

であろう。

さらに，psych パッケージの describeBy() を用いるとグループごとに記述統計を計算できる。たとえば data の各データがどのグループか表すデータが z という箱にはいっているとすると，group = data$z とする。

```
describeBy(data, group = data$z)
```

```
 Descriptive statistics by group
group: L
   vars  n  mean    sd median trimmed   mad    min   max range  skew kurtosis
x1    1 50  0.15  0.91   0.15    0.15  0.75  -2.31  2.19  4.50 -0.04     0.05
x2    2 50  0.19  4.65  -0.38   -0.21  4.18  -6.55 16.21 22.76  0.98     1.39
x3    3 50  6.25  4.72   5.52    6.35  4.35  -3.78 16.47 20.25 -0.04    -
0.36
x4    4 50 -2.01  0.21  -2.00   -2.01  0.19  -2.49 -1.49  1.01  0.05     0.10
y1    5 50  1.01  4.57   1.09    1.08  3.33 -11.83 11.38 23.20 -0.21     0.27
y2    6 50 -0.89 18.59   1.80    0.67 18.15 -63.34 26.72 90.06 -0.92     1.19
y3    7 50  0.49  1.12   0.39    0.47  1.30  -1.68  2.99  4.68  0.10    -
0.79
y4    8 50 20.15  2.21  20.41   20.33  1.88  13.50 23.88 10.38 -0.76     0.51
z     9 50  1.00  0.00   1.00    1.00  0.00   1.00  1.00  0.00   NaN      NaN
     se
x1 0.13
x2 0.66
x3 0.67
x4 0.03
y1 0.65
y2 2.63
y3 0.16
y4 0.31
z  0.00
------------------------------------------------------------
group: R
   vars  n  mean    sd median trimmed   mad    min   max range  skew kurtosis
x1    1 50  0.03  0.93  -0.07    0.01  0.92  -1.97  2.17  4.14  0.16    -
0.52
x2    2 50 -1.27  4.95  -1.52   -1.58  4.78 -10.27 10.50 20.77  0.41    -
0.38
```

```
x3    3 50  4.96  4.74   3.88    4.57  4.10  -1.80 15.99 17.79  0.68    -
0.55
x4    4 50 -2.00  0.21  -2.00   -2.00  0.22  -2.41 -1.59  0.82 -0.11    -
0.86
y1    5 50  0.11  4.39  -0.13    0.06  4.81  -9.40 10.64 20.05  0.10    -
0.55
y2    6 50  5.10 19.63   6.83    6.17 19.11 -39.57 43.01 82.58 -0.36    -
0.41
y3    7 50  0.33  1.16   0.31    0.30  1.24  -2.10  2.80  4.91  0.14    -
0.81
y4    8 50 20.21  2.11  20.13   20.18  1.83  15.84 24.63  8.79  0.18    -
0.56
z     9 50  2.00  0.00   2.00    2.00  0.00   2.00  2.00  0.00   NaN      NaN
     se
x1 0.13
x2 0.70
x3 0.67
x4 0.03
y1 0.62
y2 2.78
y3 0.16
y4 0.30
z  0.00
```

1.7 信頼区間

信頼区間を計算する方法はいくつかある。ここではブートストラップ法を用いる方法を紹介する。

ブートストラップ法はデータから復元抽出を繰り返し, 各サンプルのデータを使って分析する。したがって復元抽出の回数が 100 回であれば, 分析結果も 100 個手に入る。たとえば 1 回目の抽出では $\bar{x}_1 = 2.11$, 2 回目の抽出では $\bar{x}_2 = 3.05$, 3 回目の抽出では $\bar{x}_3 = 0.97$…… K 回目の抽出では $\bar{x}_K = 2.07$ のように, $\bar{x}$ を中心にある範囲で分析結果がばらつくことが多い (もとのデータの分布によってはそうならないこともある)。ブートストラップ法ではこのばらつきを利用して信頼区間を求める。

R では `boot` パッケージを用いて計算するため, 人間が何度も計算する必要はない。

```r
library(boot)

boot_mean <- function(data, i) {
  mu = mean(data[i])# i 番目の data の平均値を計算
  return(mu)# 計算結果を返す
}

# ブートストラップ法により復元抽出を実施する
boot_res <- boot(data$x1, boot_mean, R = 1000)

# ブートストラップ法で得たデータから信頼区間を求める
boot_ci <- boot.ci(boot_res, type = "perc")

# x1 の 95% 信頼区間
boot_ci
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot_res, type = "perc")

Intervals :
Level     Percentile
95%   (-0.0800,  0.2715 )
Calculations and Intervals on Original Scale
```

おそらく 3 行目〜6 行目で関数を定義する箇所が難所であろう。

3 行目では関数の名前と引数を定義している。引数は 2 つあり，1 つは data, もう 1 つは i である。data は元データであり，i は元データの上から何番目を取り出すかを意味する。
4 行目ではブートストラップに用いるサンプルを取り出して平均値を計算し，5 行目では平均値を出力している。

もし x1 と x2 の相関係数の信頼区間を求めたいなら, 次のように計算する。

```r
boot_r <- function(data, i){
  boot_sample = data[i, ]
  corr = cor(boot_sample$x1, boot_sample$x2)
```

```r
  return(corr)
}

dat <- data.frame(#相関を求める変数だけ別の箱に入れる
  x1 = data$x1,
  x2 = data$x2
)

boot_res <- boot(data, boot_r, R = 1000)

boot_ci <- boot.ci(boot_res, type = "perc")

boot_ci
```

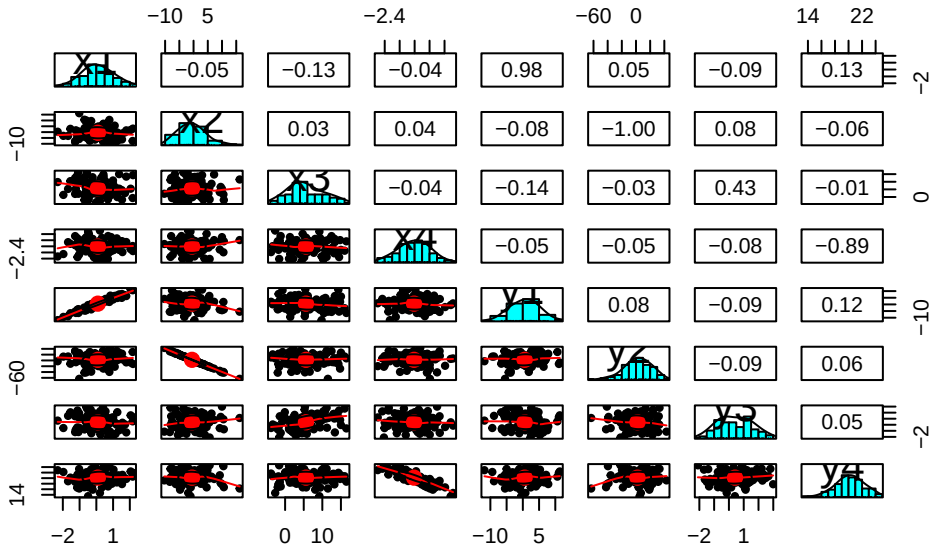
```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot_res, type = "perc")

Intervals :
Level     Percentile
95%   (-0.2190,  0.1418 )
Calculations and Intervals on Original Scale
```

自作関数の 1 行目〜2 行目は平均値のときと同じである。3 行目では平均値を求める代わりに相関を求めている。もし `x1` と `x3` の相関の信頼区間を求めたいなら, ここを `cor(boot_sample$x1, boot_sample$x3)` のように書き換える

2　psych による可視化

以降で触れる R に標準搭載された base パッケージによる関数や ggplot2 パッケージによる可視化は, 可視化そのものを目的にしている。しかし探索的データ分析ではざっくりとデータを可視化し, 全体的な傾向を確認して以降の分析に活用することが目的である。そこで活躍するのが psych パッケージの pairs.panels() である。引数にデータセットを渡すと, 相関係数, 密度曲線付きのヒストグラム, 回帰曲線付きの散布図を表示してくれる。

```
1 pairs.panels(data[1:8])#Z は可視化から除外する
```



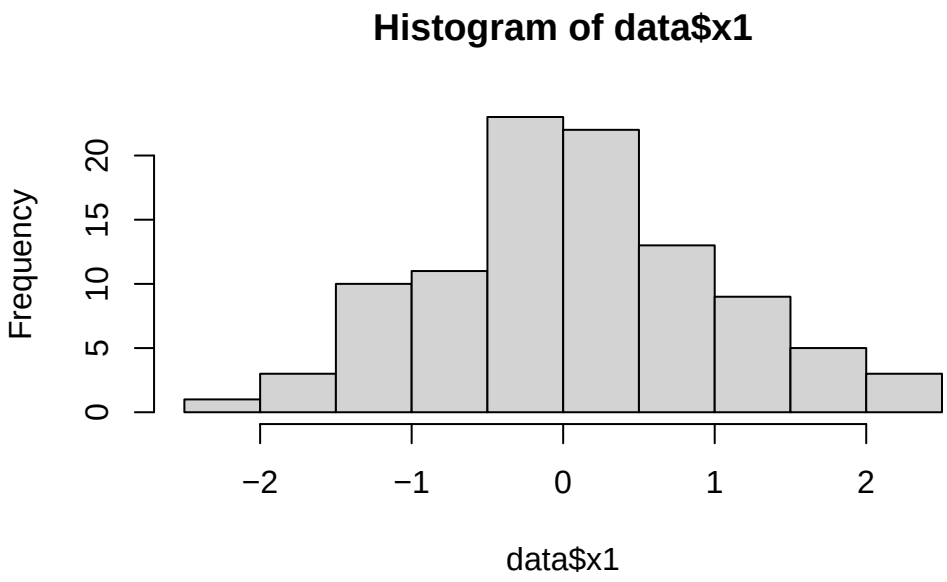
学会発表や投稿論文に使えるクオリティではないものの, 収集したデータの特徴を手軽に確認することはできる。

3　R 標準関数による可視化

3.1　ヒストグラム

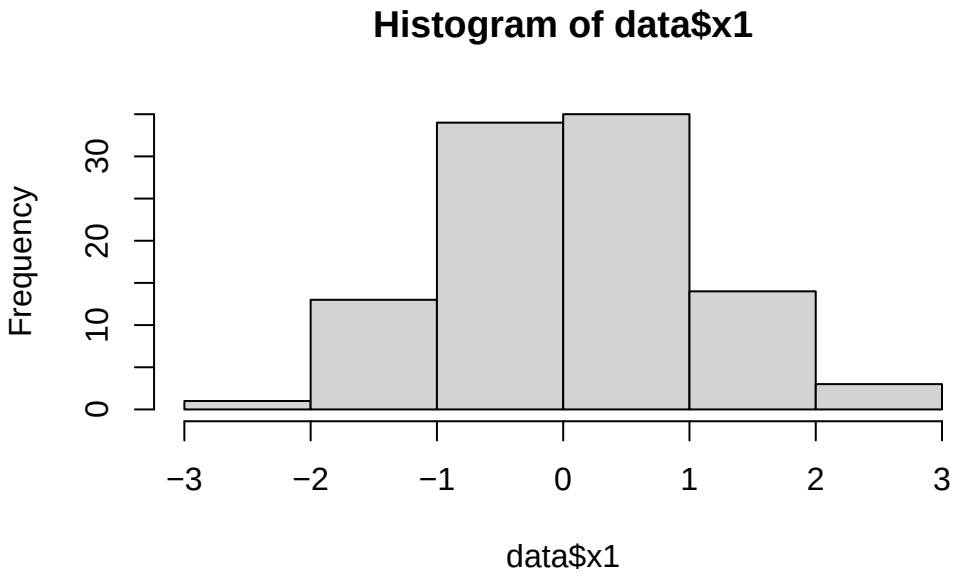
R には `hist()` 関数が標準搭載されており, 簡単にヒストグラムを可視化できる。

```r
# ヒストグラムの可視化
hist(data$x1)
```



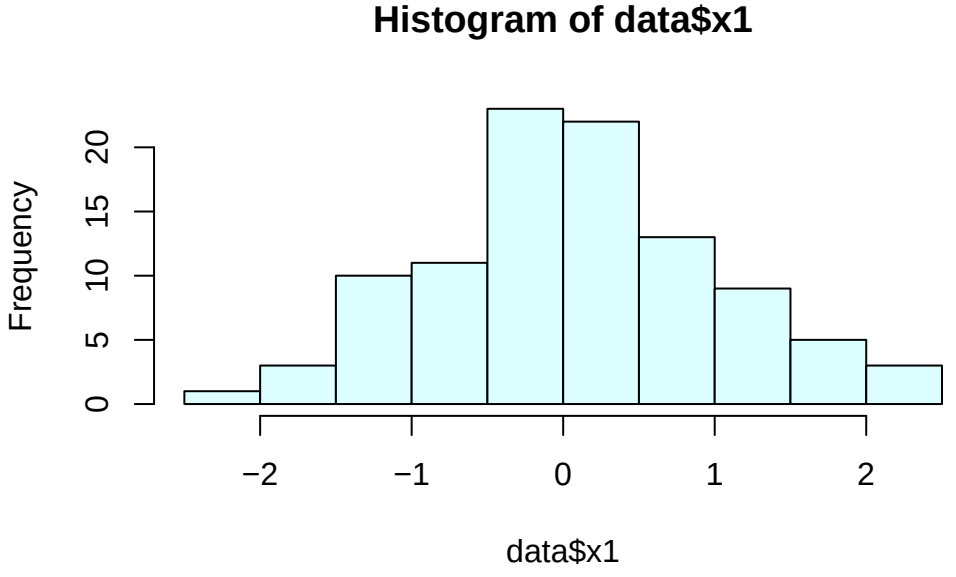
データをいくつに区切るかを指定して，**ヒストグラムの棒 (bin) の数を変える**こともできる。ここでは 5 つに区切ることにして, 棒の数が 6 つになるようにしてみよう。区切り数は `breaks` で決められる。今回は 5 つに区切るので，`breaks = 5` と書く。

```r
hist(data$x1, breaks = 5)
```



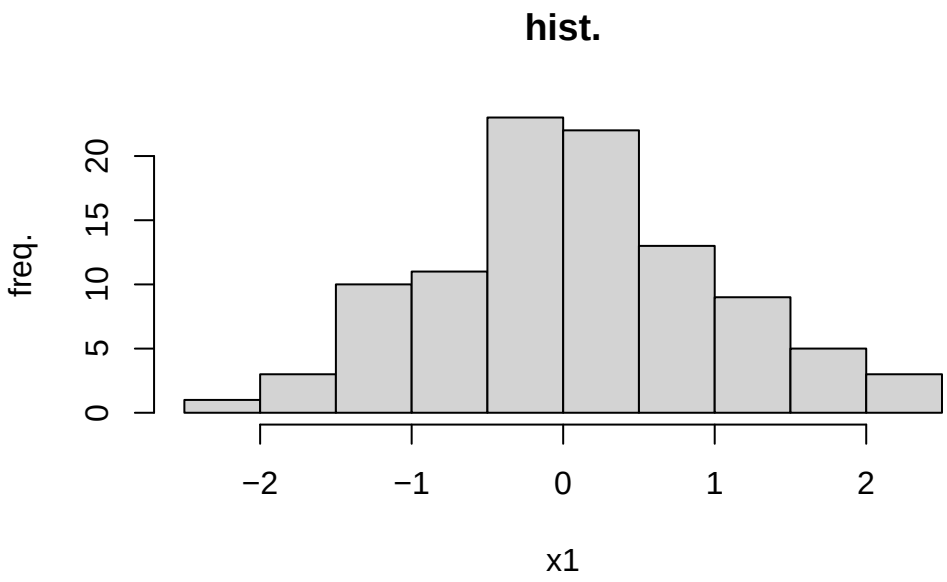
色を変えることもできる。メジャーな色なら色の名前 (red, green, gray, black など) を書けば良いこともあるし, もっと細かく指定したいなら 16 進法 (たとえば白なら`#ffffff`, 黒なら `#000000`) で指定することもできる。色は `col` で決められる。今回は薄い青色にしたいので, `col = "#ddffff"`と書く。

```
hist(data$x1, col = "#ddffff")
```



軸や図のテキストも変更できる。横軸のテキストは `xlab`, 縦軸のテキストは `ylab`, 図のタイトルは `main` を指定する。

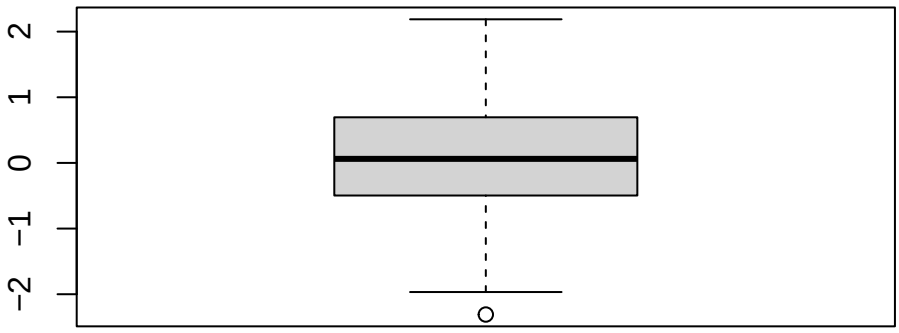
```
hist(data$x1, xlab = "x1", ylab = "freq.", main = "hist.")
```



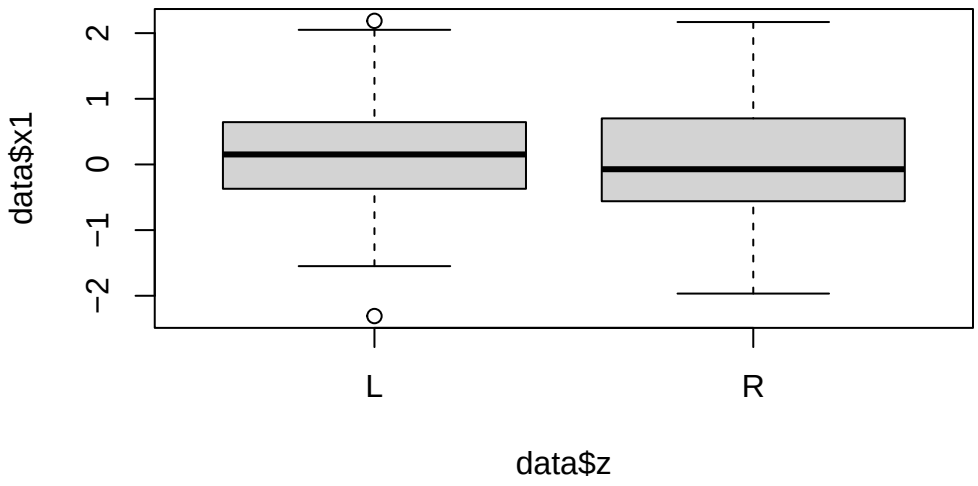
3.2 箱ひげ図

四分位数をまとめた箱ひげ図を作成するには `boxplot()` を用いる。色やテキストの変更についてはヒストグラムと同様に変更できる。グループにわけて箱ひげ図を作成するには，`~`を用いて**変数 ~ グループ**のように書く (`~`はチルダと読む。`shift+ へ`を入力すると出てくる)。`data` という大きな箱にある `z` ごとに `x1` の箱ひげ図を描きたいなら `boxplot(data$x1 ~ data$z)` とする。

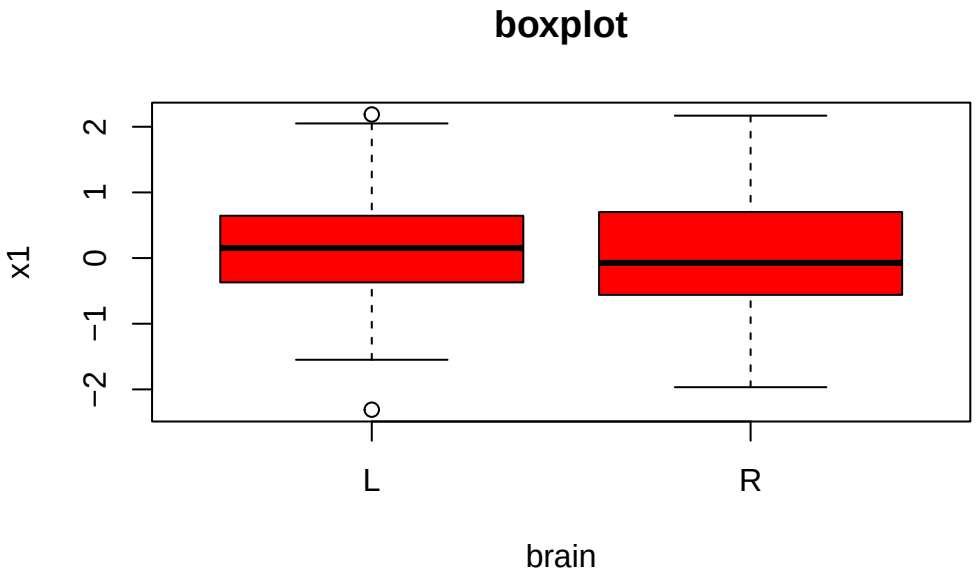
```
# x1 の箱ひげ図
boxplot(data$x1)
```




```
# z ごとに x1 の箱ひげ図
boxplot(data$x1 ~ data$z)
```




```r
#色とテキストを変更
boxplot(data$x1 ~ data$z,
        col = "red",
        xlab = "brain", ylab = "x1", main = "boxplot")
```

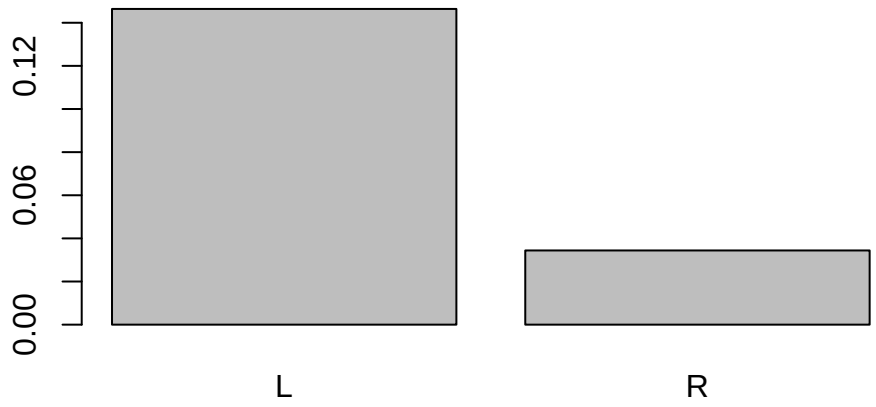


3.3 棒グラフ

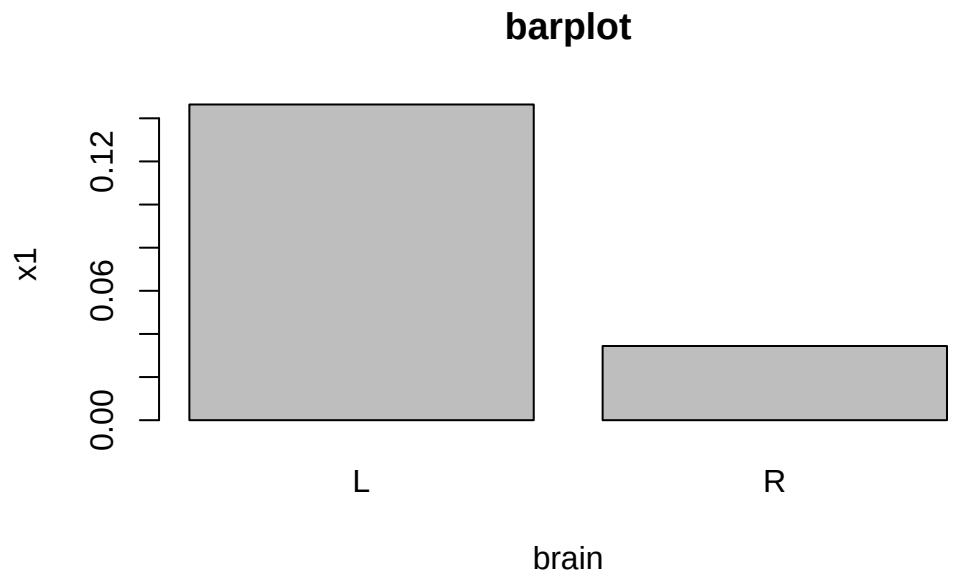
棒グラフを作成するのはひと手間かける必要がある。R で棒グラフを作成する関数は `barplot()` であるが, 引数には代表値を用いる必要がある。よってグループごとの代表値を計算し, その平均値を用いるという 2 段階の処理を要する。棒グラフでないといけない理由がないなら, 可視化が容易な箱ひげ図で十分であろう。

```r
# グループごとに x1 の平均値を計算する
x1_desctiptive_stat <- describeBy(data[1], group = data$z)

```

```r
# 平均値の部分だけ取り出す
x1_mean <- c(x1_desctiptive_stat$L$mean, x1_desctiptive_stat$R$mean)

# 棒グラフを作成する
barplot(x1_mean, names.arg = c("L", "R"))
```



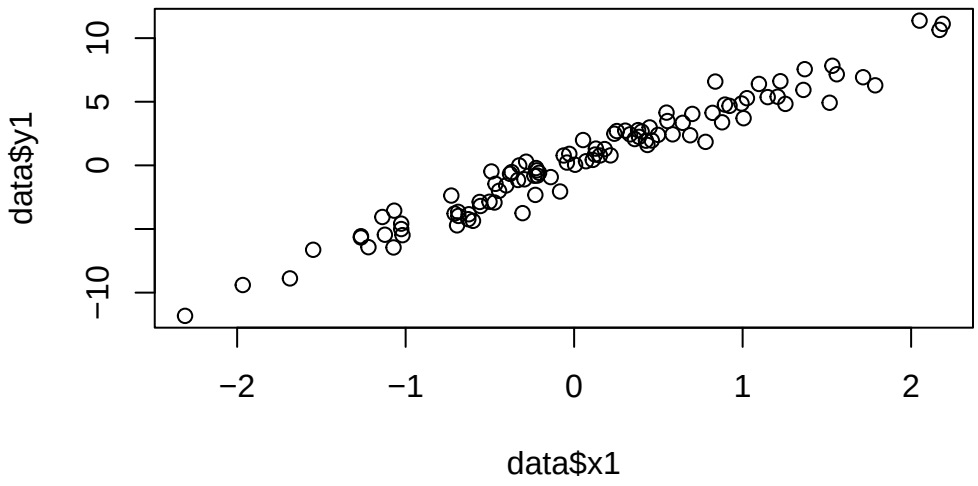

```r
# テキスト（軸ラベル）を追加する
barplot(x1_mean, names.arg = c("L", "R"),
        xlab = "brain", ylab = "x1", main = "barplot")
```



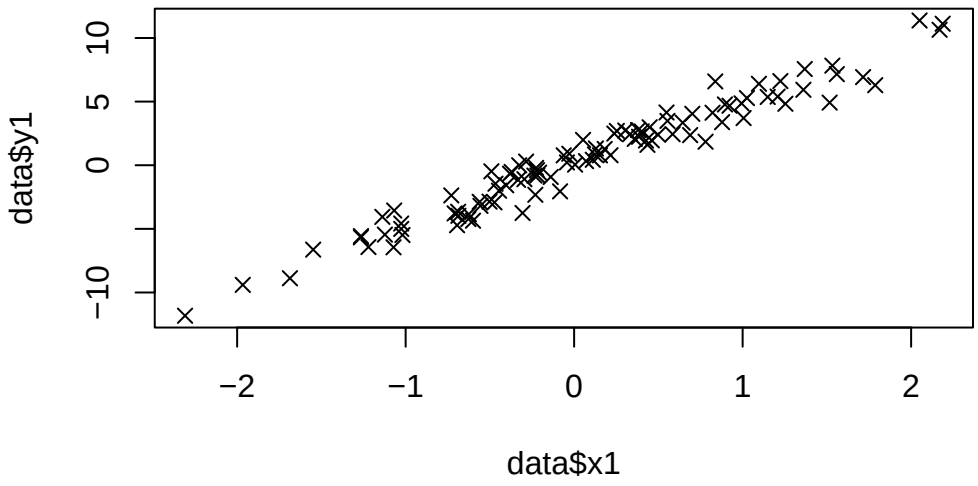
3.4 散布図

散布図を描くには plot() を用いる。たとえば data という大きな箱に入っている x1 と y1 を使って散布図を描くなら，plot(data$x1, data$y1) と書く。デフォルトではデータ点を◯で表現するが，pch で値を指定することで × や△で表現することもできる。またこの記号のサイズも cex で変更できる。

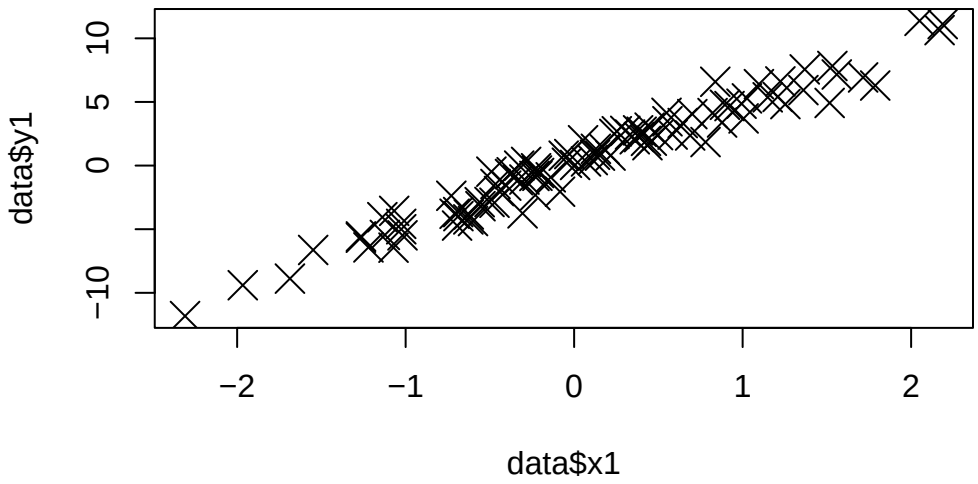
```r
# x1 と y1 の散布図
plot(data$x1, data$y1)
```




```r
# データ点を「×」に変更
plot(data$x1, data$y1, pch = 4)
```

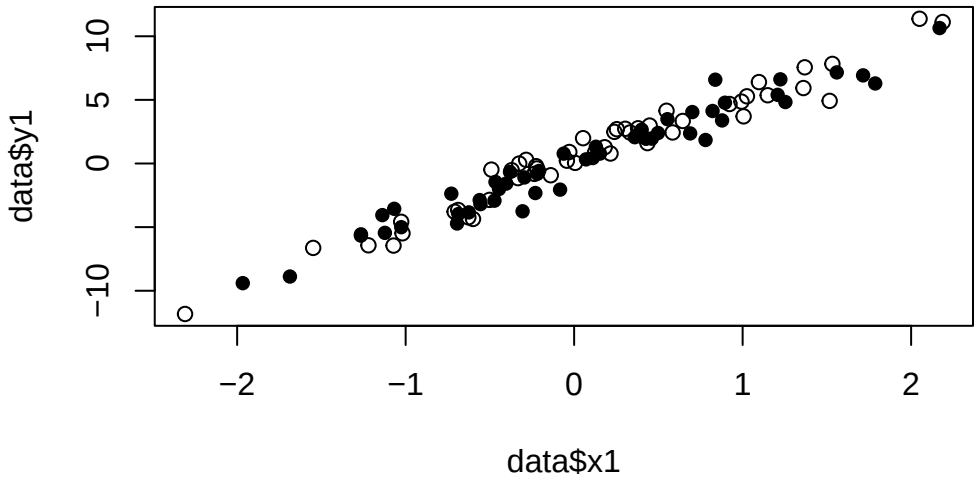



```r
# データ点の大きさを変更（デフォルトの 2 倍）
plot(data$x1, data$y1, pch = 4, cex = 2)
```



少し手間があるものの, グループごとに記号を変えて散布図を作ることもできる。グループ間で関連性に偏りがないか可視化するときに役立つ。

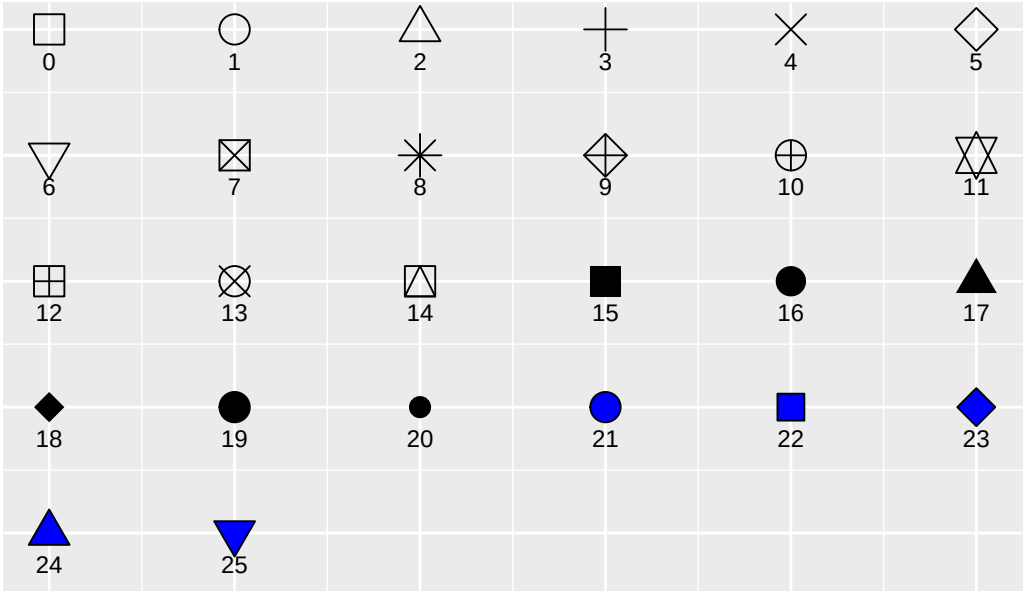
```
# 記号を指定する。z が L のときは◯で，R のときは●で表現する
pch_value <- c("L" = 1, "R" = 16)
# z の値によって記号が変わるようにする
plot(data$x1, data$y1, pch = pch_value[z])
```



データ点はほかにも色々な形状を選択できる。値と記号の対応関係については以下の通りである。記号の下にある数字を pch に指定すると, その記号に変えられる。

```
ggpubr::show_point_shapes()
```

Point shapes available in R



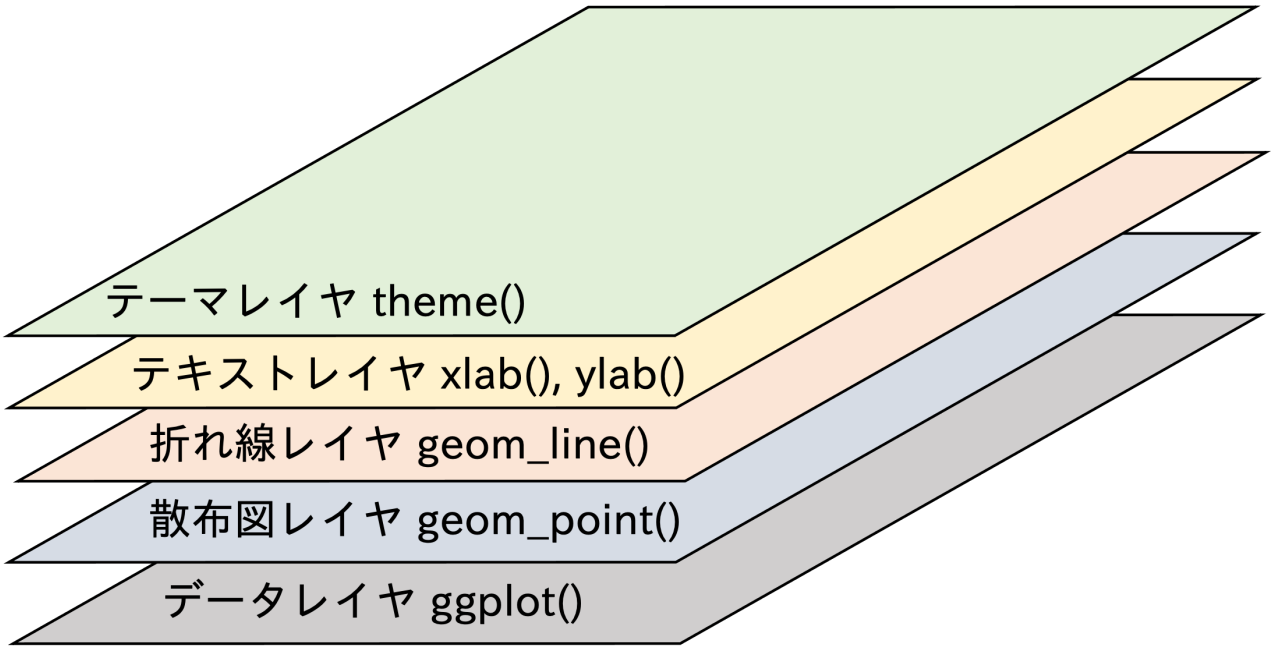
4　ggplot2 による可視化

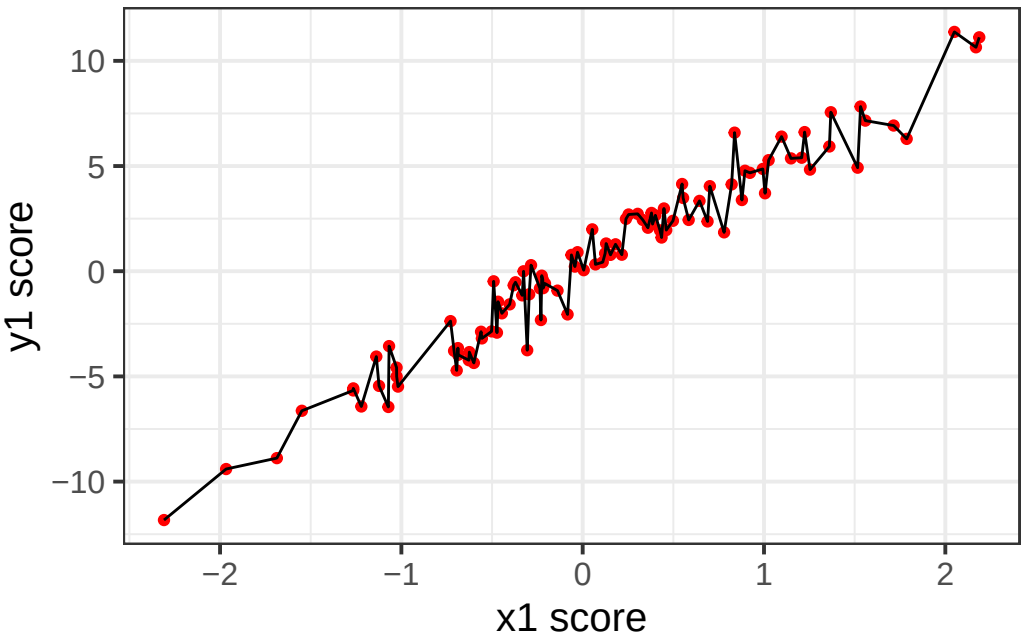
R に標準搭載された関数では, とくに学術論文に掲載する水準の図を作るのが難しい。ggplot2 パッケージを用いると, より柔軟で美しい図を作ることができる。

```
#install.packages("ggplot2")
library(ggplot2)
```

4.1　ggplot2 の仕組み

ggplot2 では図をレイヤとして捉え, レイヤを重ねていく。油絵のように重ね描くため, より上層のスクリプト (よりあとから実行されたスクリプト) が上に来る。よって下図であれば散布図の上に折れ線グラフが描画される。





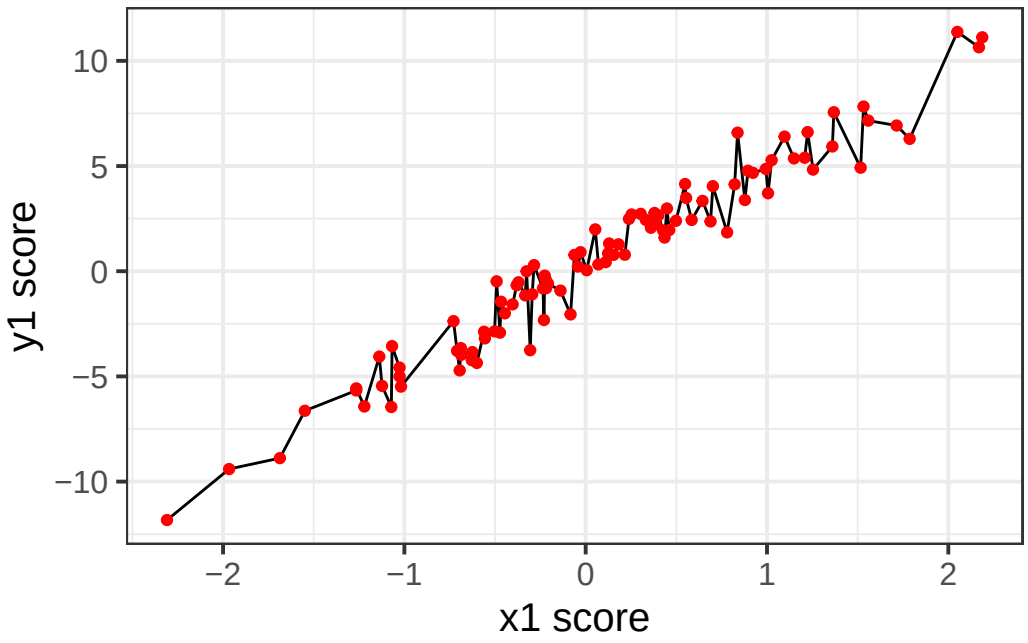
上記の関数 1 つ 1 つがレイヤになっている。最初の関数は `ggplot()` であり, データセットや図の x 軸と y 軸などを指定する。ここでは x1 と y1 の散布図と折れ線グラフを描画するために，`aes(x = x1, y = y1)` と書いて x 軸に `x1`，y 軸に `y2` を設定している。

2 つ目の関数である `geom_point()` は散布図を描く関数である。`color` (または `colour`) でデータ点の色を変えることができる。色名以外にも，16 進法で指定できる。細かな色使いや配色バリアフリーを実現したい場合，16 進法で指定するのがおすすめである。ほかにも設定できる箇所があるため, 詳細は該当箇所で述べる。

3 つ目の関数は `geom_line()` であり, 折れ線グラフを描く。

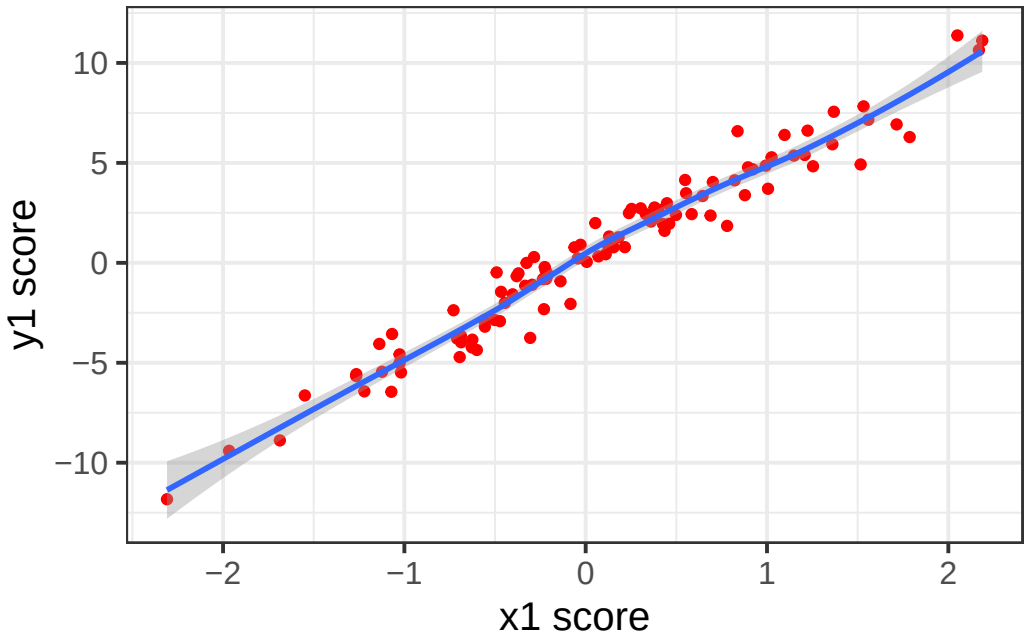
4 つ目の関数は `xlab()`，5 つ目の関数は `ylab()` であり, それぞれ x 軸と y 軸のテキストを指定する。

この図をよく見ると, 赤いポイントのうえに黒い線が乗っているのが見える。これは散布図を作る `geom_point()` を実行し, そのあとに折れ線グラフを作る `geom_line()` を実行したためである。関数の順を逆にすると, 以下のように黒い線のうえに赤いポイントが乗る。



ggplot2 の関数のうち，geom_***() は図を作る関数である。ほかにも統計的な処理を施す stat_***() もある。たとえば stat_smooth() はデータから近似曲線を描いてくれる。

```
ggplot(data, aes(x = x1, y = y1)) + geom_point(color = "red") + stat_smooth() +
  xlab("x1 score") + ylab("y1 score") + theme_bw(base_size = 15)
```

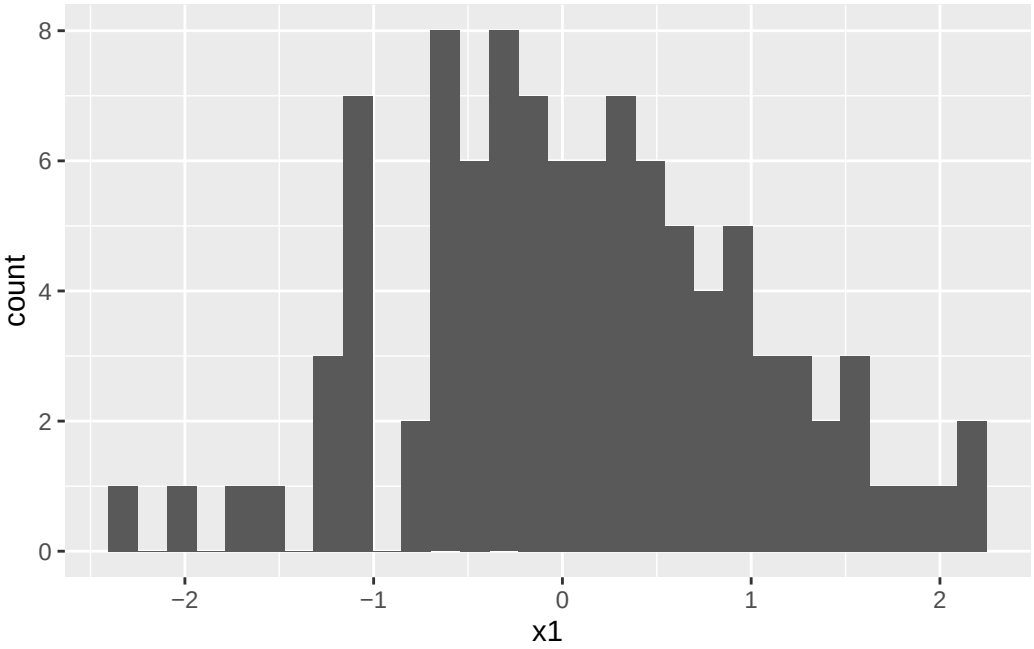


このようなレイヤ構造と様々な処理が可能な関数により，ggplot2 パッケージを用いることで複数の情報を圧縮した複雑な図を生み出すことができる。以降では比較的単純な可視化について説明する。詳しい操作法については日本語で読める良書があるので，詳細は書籍に譲る (たとえば『R グラフィッククックブック第 2 版』)。

4.2 ヒストグラム

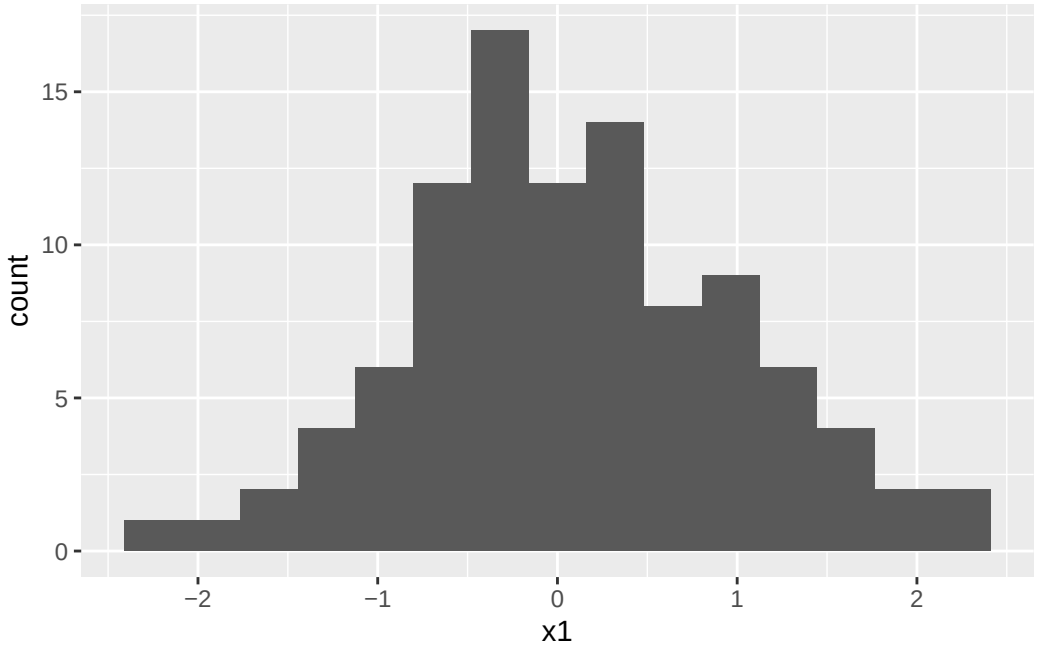
ggplot2 でヒストグラムを描くには，geom_histogram() を用いる。

```
ggplot(data, aes(x = x1)) + geom_histogram()
```



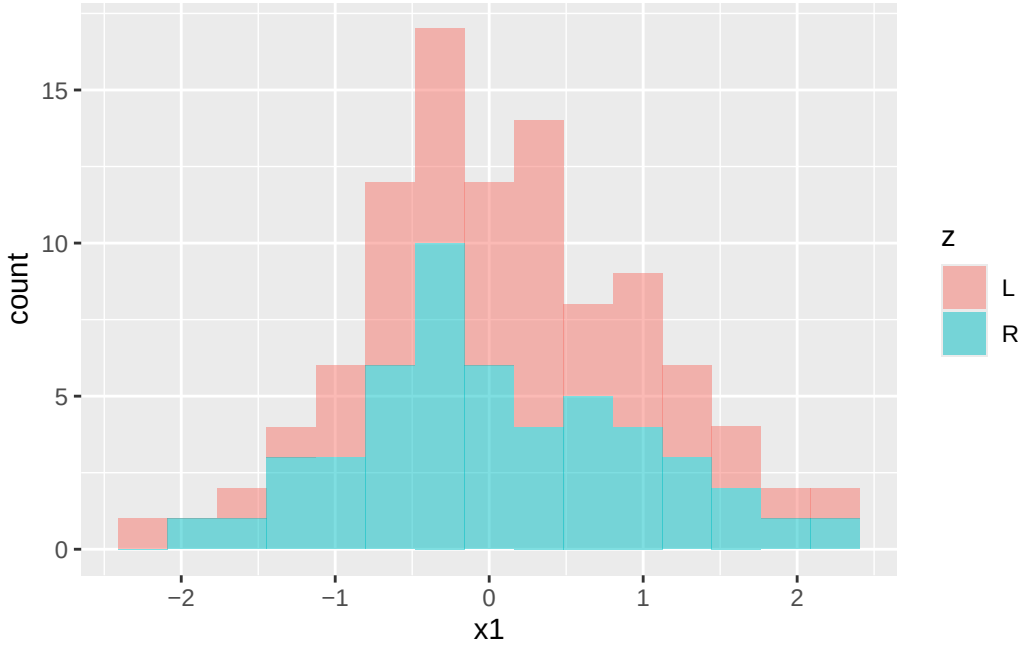
上記のスクリプトだけでヒストグラムを作れるものの, 棒 (bin) の数がデフォルトで 30 になっており, データによっては歯の抜けたクシのようになる。そこで引数に `bins` を用いて微調整し, 棒の数を変更する。

```
ggplot(data, aes(x = x1)) + geom_histogram(bins = 15)
```



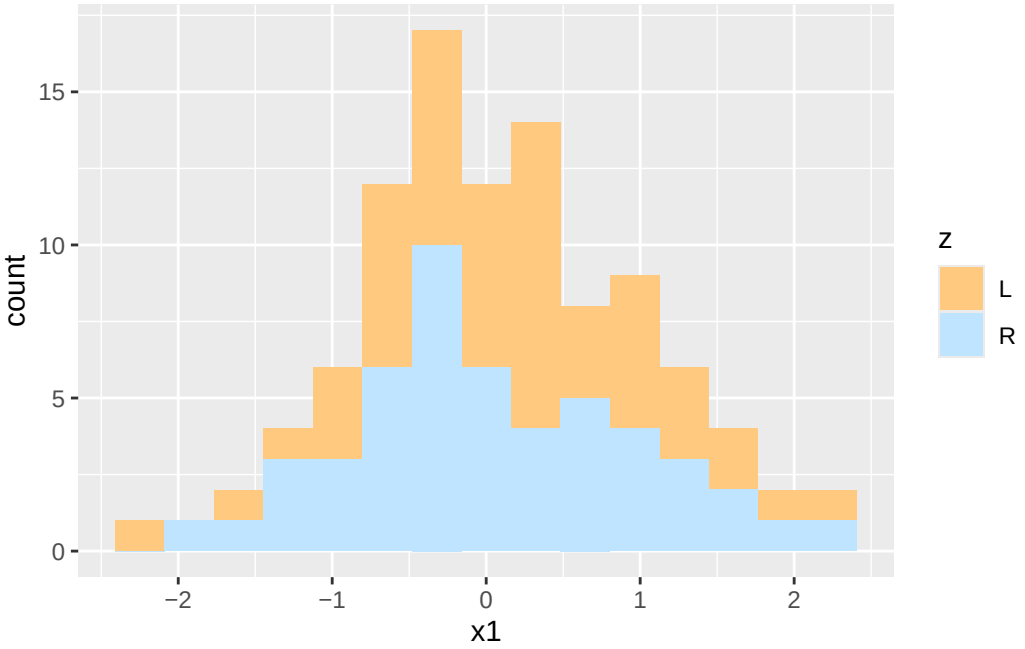
2 つのヒストグラムを重ねることもできる。z が R のケースと L のケースで x1 のヒストグラムを作り重ねるなら，aes() のなかで fill を指定すると色分けしてくれる。また, ヒストグラムが重なった部分も表示するために，geom_histogram() のなかで alpha で色を透過させることもできる (今回はほとんど同じ分布なので効果は実感しづらいが)。

```
ggplot(data) +
  geom_histogram(aes(x = x1, fill = z), bins = 15, alpha = 0.5)
```



着色は ggplot2 が自動で実行してくれるが, 色を自分で指定したいなら scale_fill_manual() を用いる。

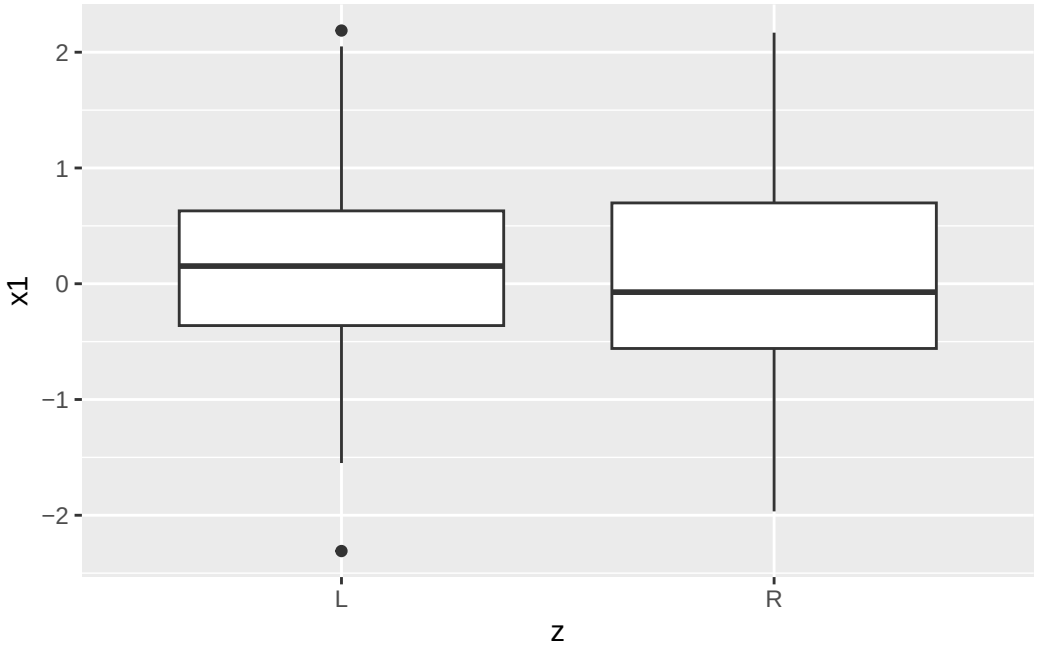
```r
ggplot(data) +
  geom_histogram(aes(x = x1, fill = z), bins = 15) +
  scale_fill_manual(values = c("L" = "#ffca80", "R" = "#bfe4ff"))
```



4.3 箱ひげ図

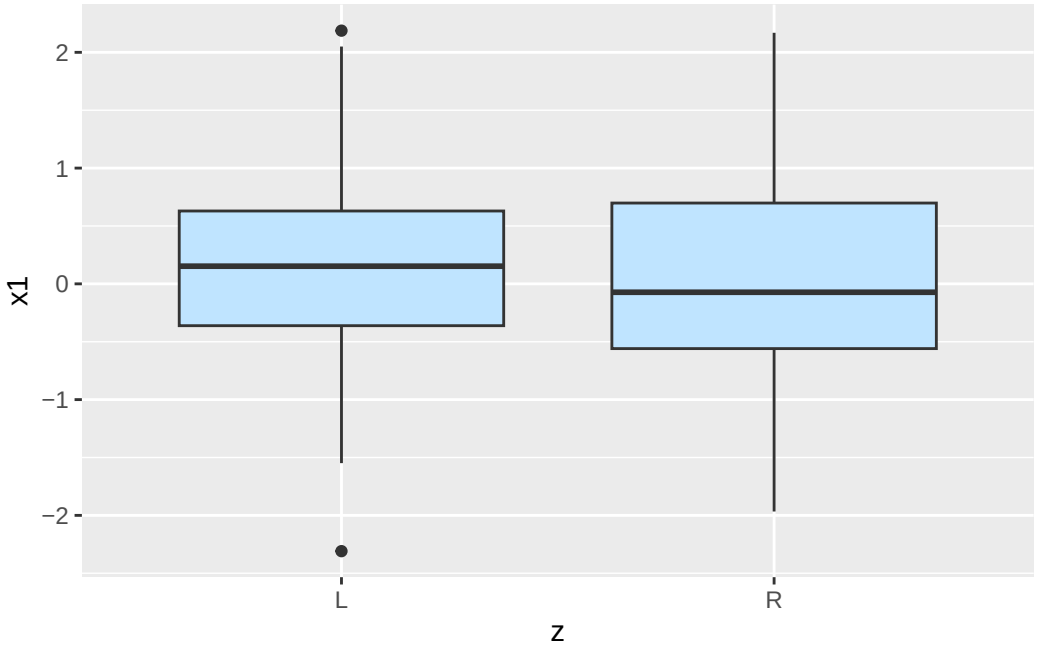
ggplot2 で箱ひげ図を描くには，geom_boxplot() を使う。z が R の者と z が L の者で箱ひげ図を作る場合，ggplot() の引数に aes() を用い，x と group にそれぞれ z を指定する。x は x 軸に何を使うかを指定しており，group はグループを示す変数はどれかを指定している。今回はいずれも z であるため，aes(x = z, y = x1, group = z) と書く。

```r
ggplot(data, aes(x = z, y = x1, group = z)) +
  geom_boxplot()
```



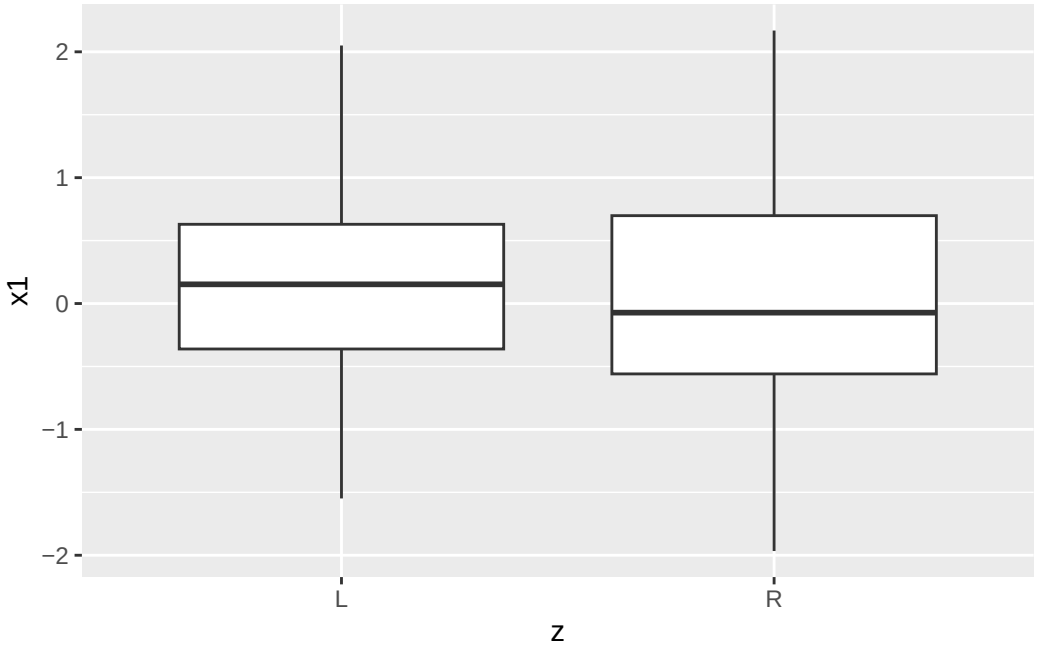
塗りつぶすには geom_boxplot() の引数で fill を指定する。

```
ggplot(data, aes(x = z, y = x1, group = z)) +
  geom_boxplot(fill = "#bfe4ff")
```



なお geom_boxplot() の引数を outliers = FALSE とすると，

```
ggplot(data, aes(x = z, y = x1, group = z)) +
  geom_boxplot(outliers = FALSE)
```



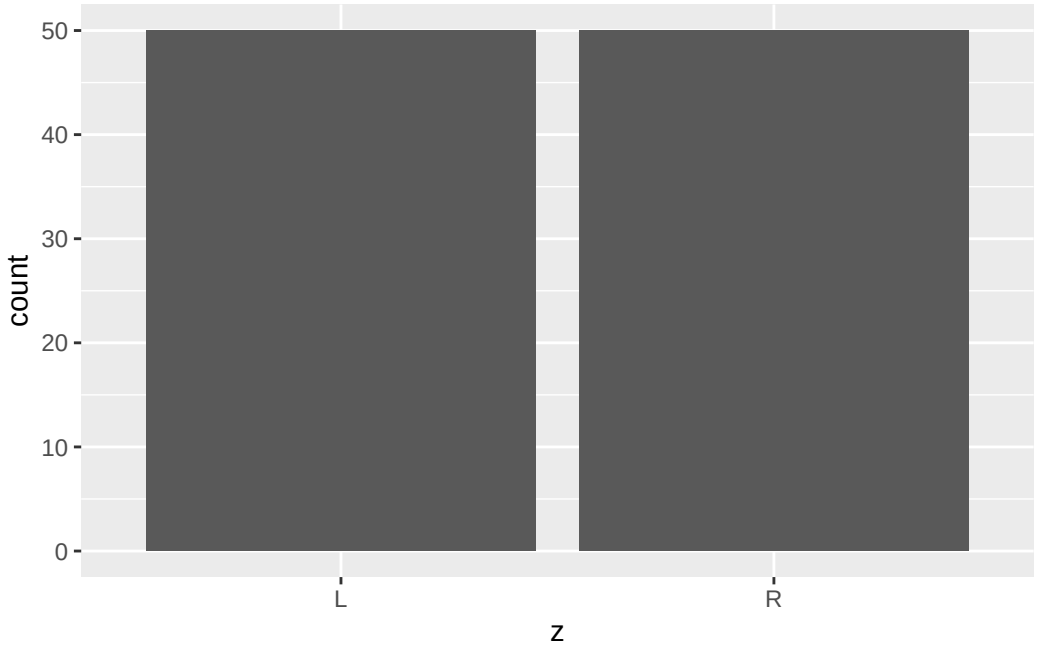
黒丸で示されていた外れ値が非表示になる。

4.4 棒グラフ

ggplot2 で棒グラフを描画する場合は geom_bar() を用いる。ただし geom_bar() はカテゴリ変数の度数 (個数) を可視化するため, 心理学でよく使われているような平均値の可視化には工夫が必要である。

まず, 度数の可視化について示す。z の度数を可視化するためには, aes() の引数に x = z を渡すだけで良い。そのうえで geom_bar() を適用して実行すると, 度数を可視化できる。

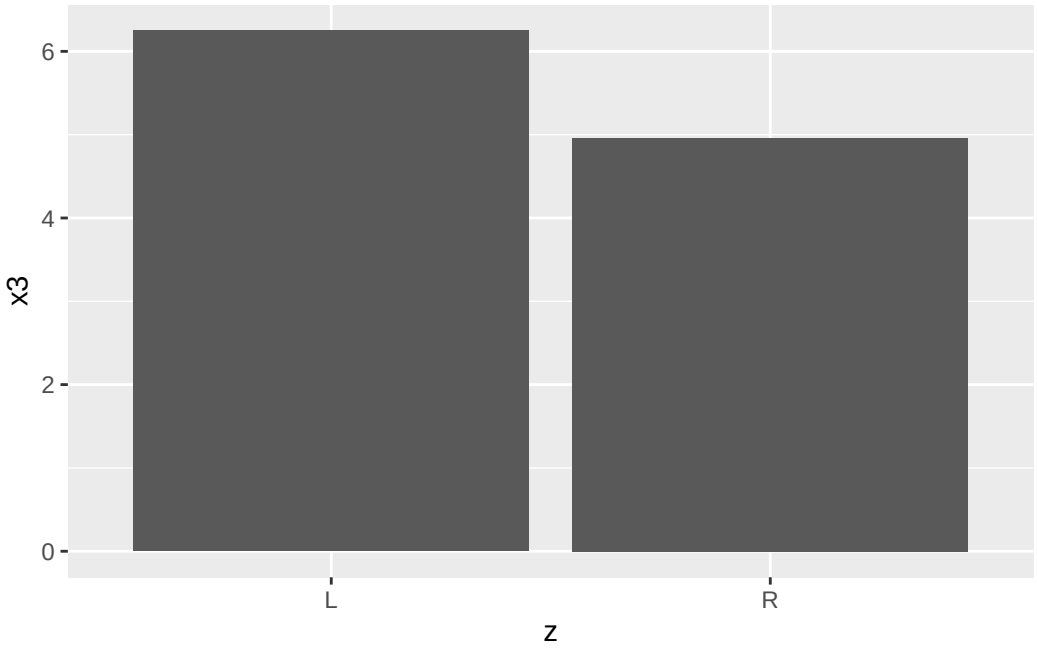
```
ggplot(data, aes(x = z)) +
  geom_bar()
```



次に平均値を可視化する。geom_bar() の引数に stat = "summary"および fun = "mean"を渡すと平均値を可視化できる。stat はデータの処理法を指定する引数であり，stat = "summary" で記述統計を計算することができる。どの記述統計を用いるかは fun で指定する。fun = "mean" は平均値を，fun = "sum"は総和を，fun = "median"は中央値を計算する。

z ごとの x3 の平均値を可視化する場合，aes() の引数を x = z および y = x3 とする。なお，これを逆にすると，横向きの棒グラフになる。

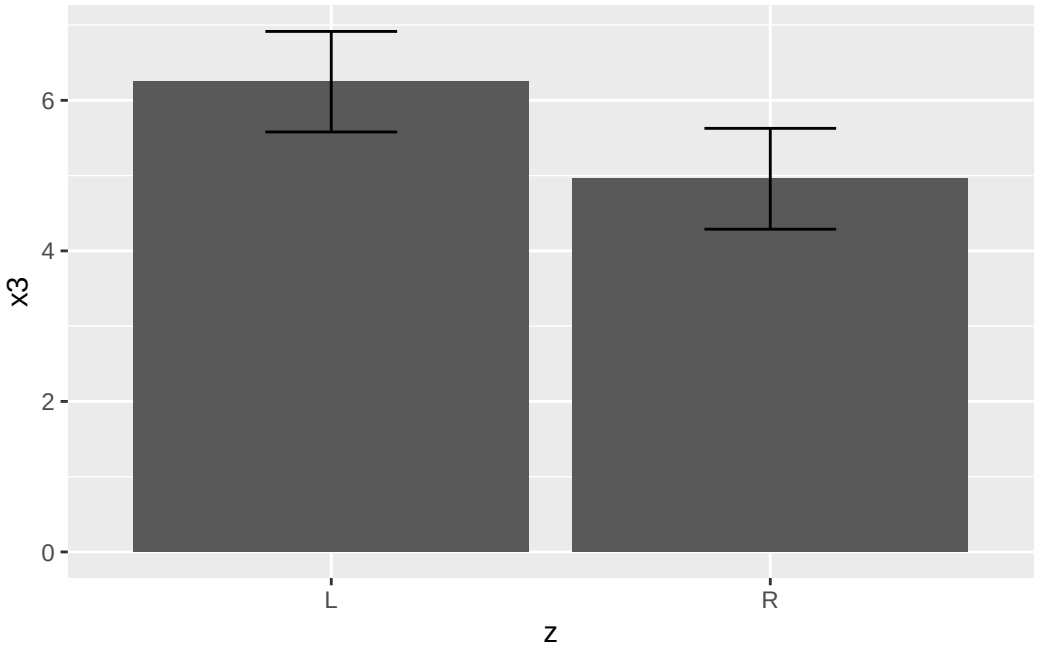
```
ggplot(data, aes(x = z, y = x3)) +
  geom_bar(stat = "summary", fun = "mean")
```



一般的に，棒グラフにはエラーバーをつけることが多い。そこで，この棒グラフにもエラーバーをつけていく。エラーバーを追加するには stat_summary() を用い，引数を fun.data = mean_se お

よび geom = "errorbar"とする。fun.data はデータをどのように処理するか決めるための引数であり，mean_se は平均値 + 標準誤差，mean_sdl は平均値 +2SD，mean_ci_normal は平均値 + 正規分布を仮定した 95% 信頼区間を計算する。geom はデータをどのように表示するか決める引数であり，geom = "errorbar"でエラーバーとして表示する。エラーバーの幅は width で制御でき, ここでは棒グラフの幅の 30% になるよう width = 0.3 とした (何も指定しないと棒グラフと同じ幅になり見栄えが悪い)。

```
ggplot(data, aes(x = z, y = x3)) +
  geom_bar(stat = "summary", fun = "mean") +
  stat_summary(fun.data = mean_se, geom = "errorbar", width = 0.3)
```



4.5 散布図

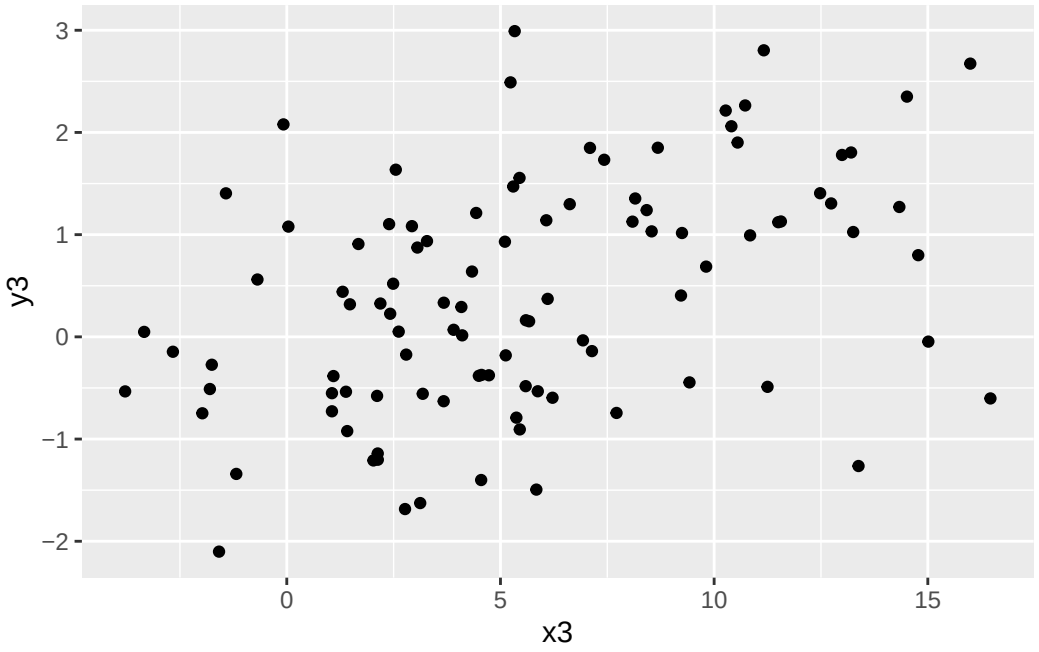
ggplot2 で散布図を描くための関数は 2 つある。1 つ目は geom_point()，2 つ目は geom_jitter() である。両者の違いはデータ点の重なり方にあり，geom_point() はデータ点をそのまま描画するため, 同じ座標に位置するデータ点は重なる。一方で geom_jitter() はデータ点をランダムに少しだけずらして描画するため, 本来同じ座標に位置するデータであっても重なりにくい。よって散布図から実際の数値を読み取るのは困難になる。

以上を踏まえると, サンプルサイズが多くないなら geom_point() が正確であり, サンプルサイズが多いなら geom_jitter() のほうがデータ点が潰れにくくて良い。とくにリッカートスケールで評価する心理尺度の得点は同じ座標に位置することが多いため，geom_jitter() の長所が活かされやすいと思われる。現実的には上記の特徴と手元のデータの特徴を踏まえて, 目的に即したものを使うべきである。

今回使用するデータは正規分布から生成されたデータであるため, ここでは geom_point() によ

り散布図を描画する。単純な x3 と y3 の散布図を描画するには，ggplot() の引数である aes() で x = x3 と y = y3 を指定するだけで良い。

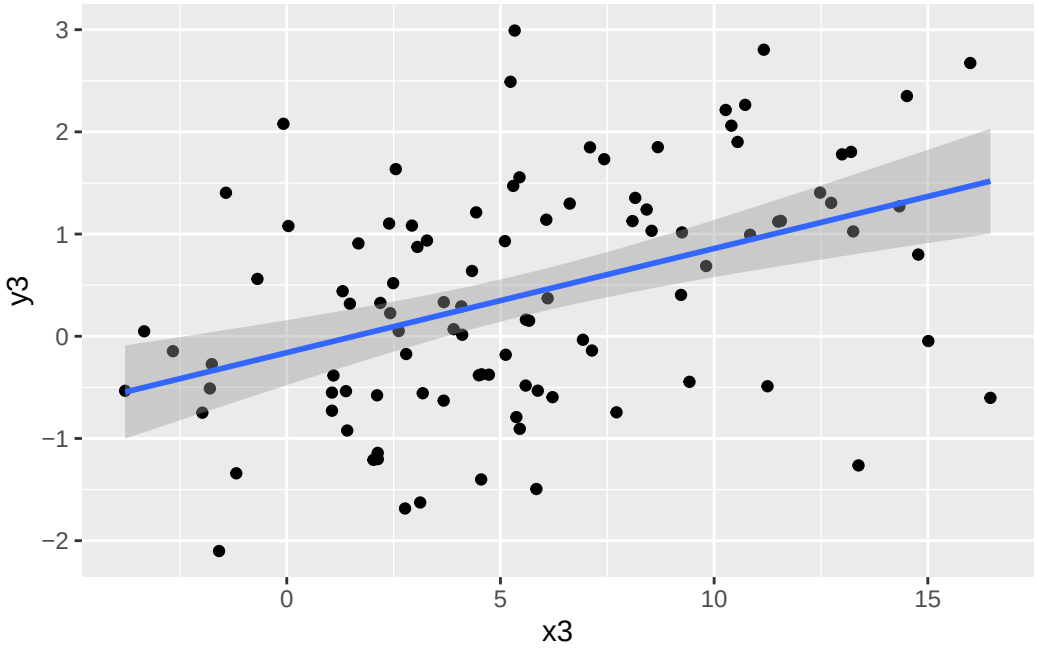
```
ggplot(data, aes(x = x3, y = y3)) +
  geom_point()
```



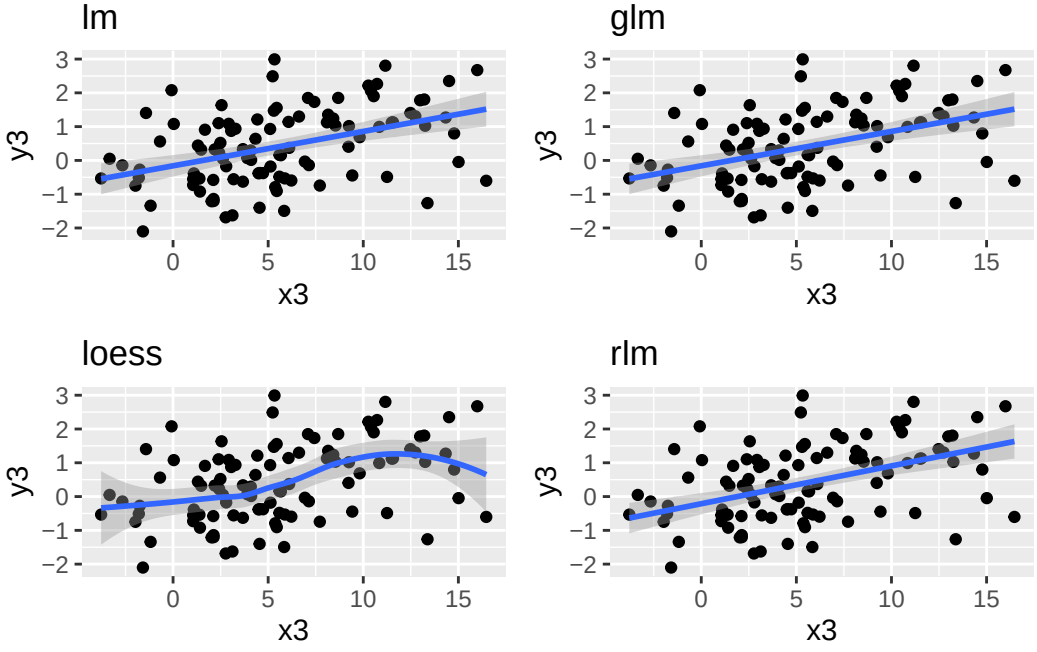
ここに回帰直線と信頼帯を追加するには，stat_smooth() を用いる。引数は method = "lm"および se = TRUE である。もし信頼帯が不要なら，se = FALSE とすれば回帰直線だけを描画できる。

```
ggplot(data, aes(x = x3, y = y3)) +
  geom_point() +
  stat_smooth(method = "lm", se = TRUE)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

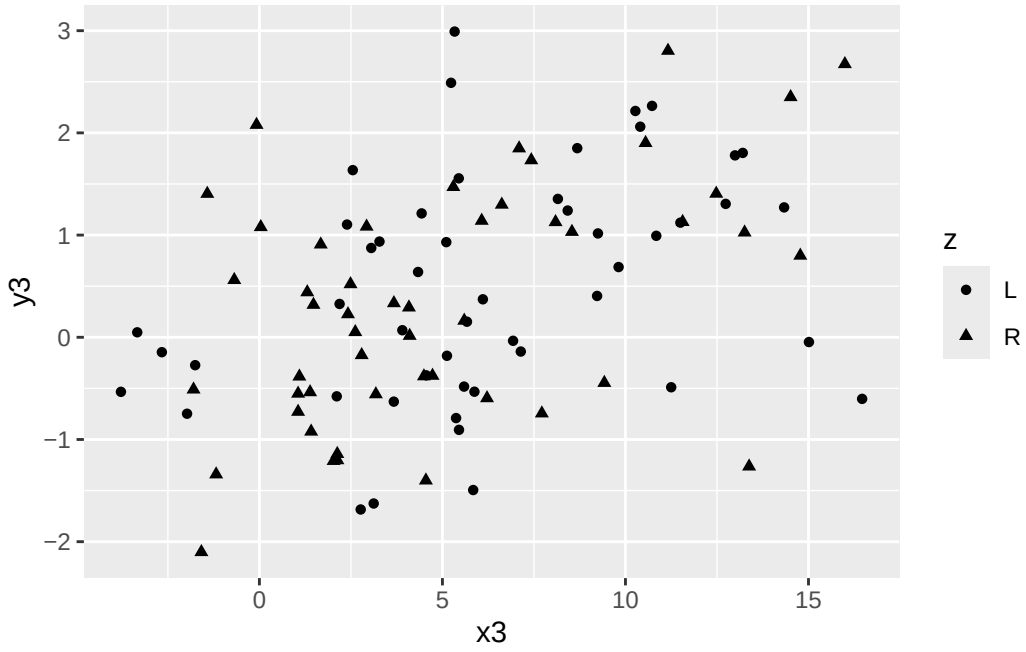


2 変数の関連が直線であれば `method = "lm"`を用いるが, ほかにも様々な回帰曲線を用いることができる。デフォルトでは `method = "loess"`であり, 何も指定しないと局所回帰曲線が追加される。心理系ではあまり馴染みのない曲線であり, 多くのケースで線形回帰にもとづく `method = "lm"`または一般化線形回帰にもとづく `method = "glm"`を用いることになるであろう。`MASS` パッケージを用いれば, 外れ地の影響を抑制したロバスト回帰にもとづく `method = "rlm"`を使える。データの性質や研究仮説と相談して決めるのが一般的である。



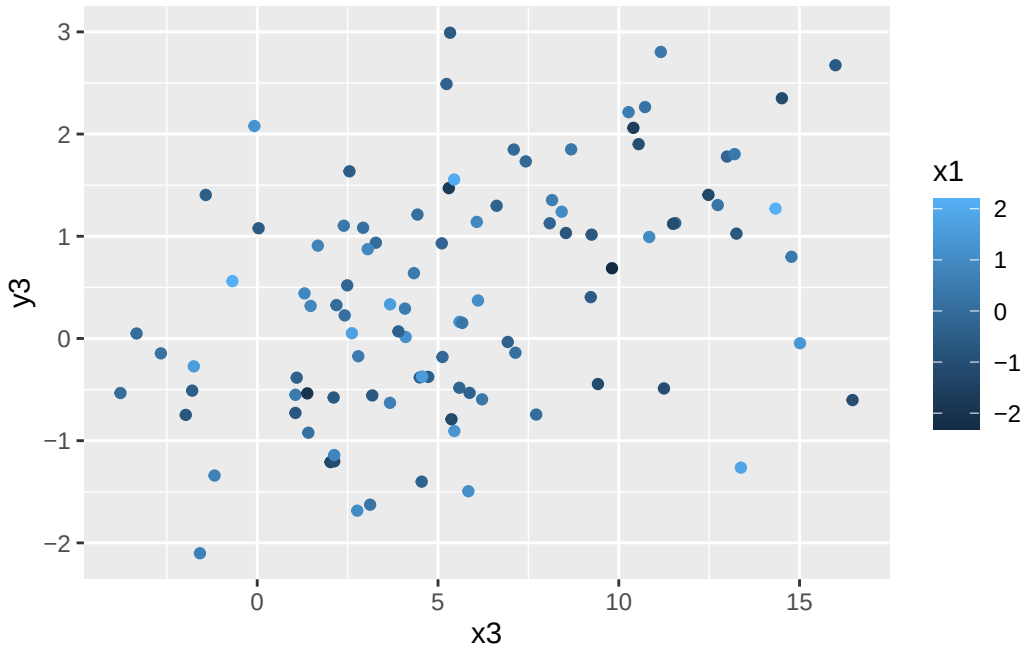
R 標準搭載の `plot()` でやったように, グループごとにデータ点を示す記号を変えることもできる。たとえば `z` が `R` の人と `L` の人で記号を変えたいなら, 以下のように `aes()` に `shape = z` を追加する。

```
ggplot(data, aes(x = x3, y = y3, shape = z)) +
  geom_point()
```



また, ほかの変数の値によって色を変えることもできる。x1 の値によって記号の色にグラデーションをかける場合，aes() に color = x1 を追加する。第 3 の変数で色合いにグラデーションをかけることにより, 交絡変数の存在を調べることができる。

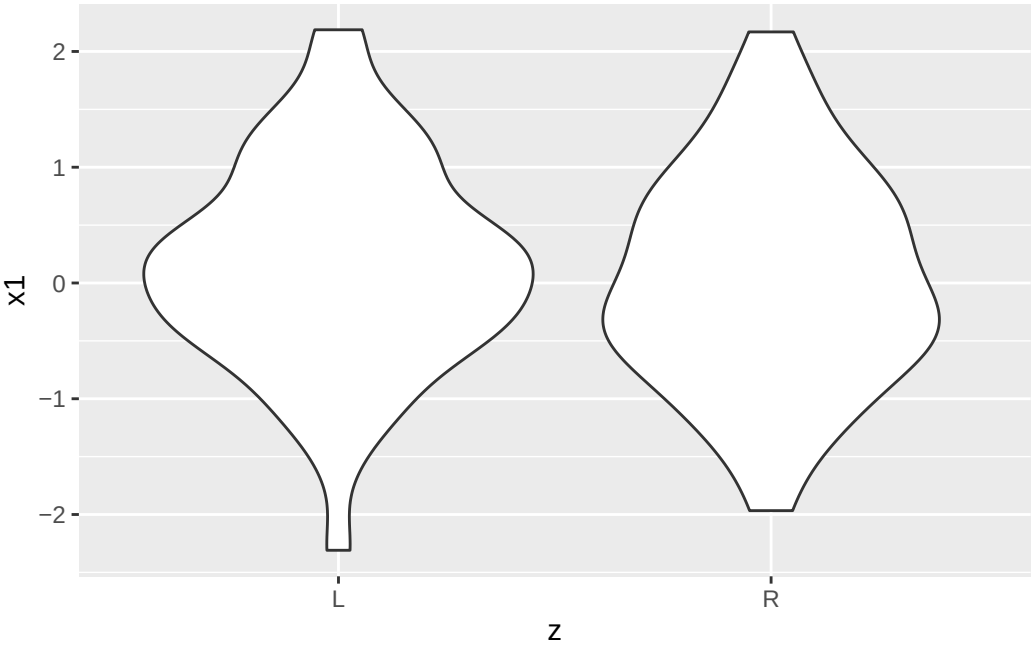
```
ggplot(data, aes(x = x3, y = y3, color = x1)) +
  geom_point()
```



4.6 バイオリンプロット

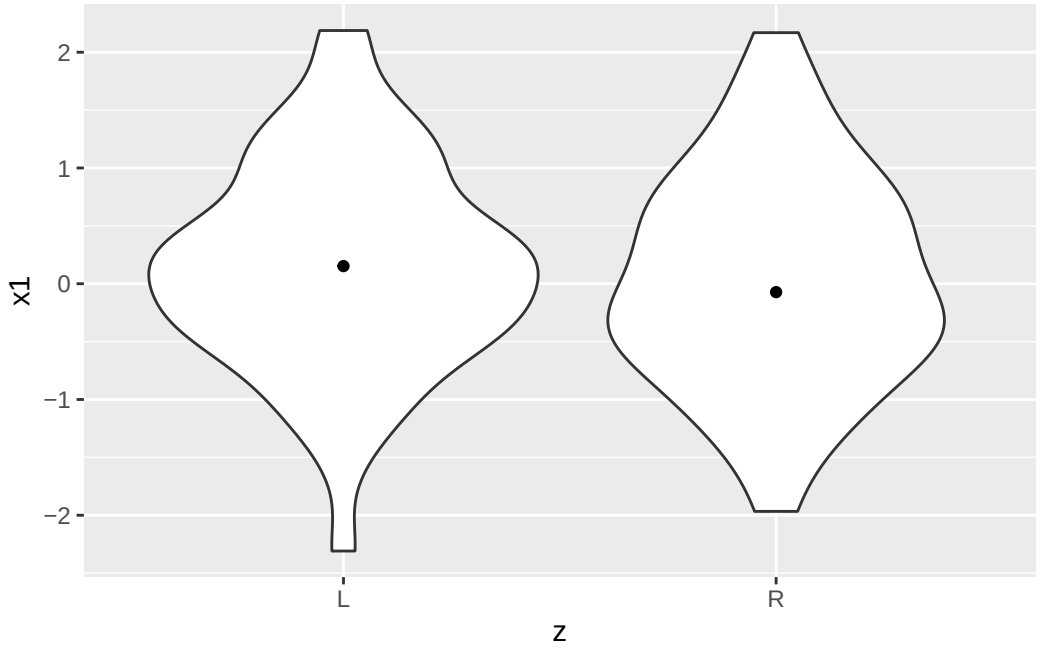
箱ひげ図は分布の形状が分からない。箱ひげ図に似たことがしたいし，分布の形状も示したいというときはバイオリンプロットを用いる。ggplot2 では geom_violin() を用いる。

```
ggplot(data, aes(x = z, y = x1)) +
  geom_violin()
```



ツボのような図を縦向きに分割して 90° 回転させると密度曲線となり，z ごとの分布の形状を確認できる。L は R よりも凹凸が大きく，R は L よりもフラットであることが分かる。ただし分布の形状だけではわかりにくいため，ここに代表値を加えたい。**平均値や中央値を追加したい場合，stat_summary() を用いて引数に fun = "mean"や fun = "median"を渡す。geom = "point" とすれば平均値や中央値の位置を点で示すことができる。**

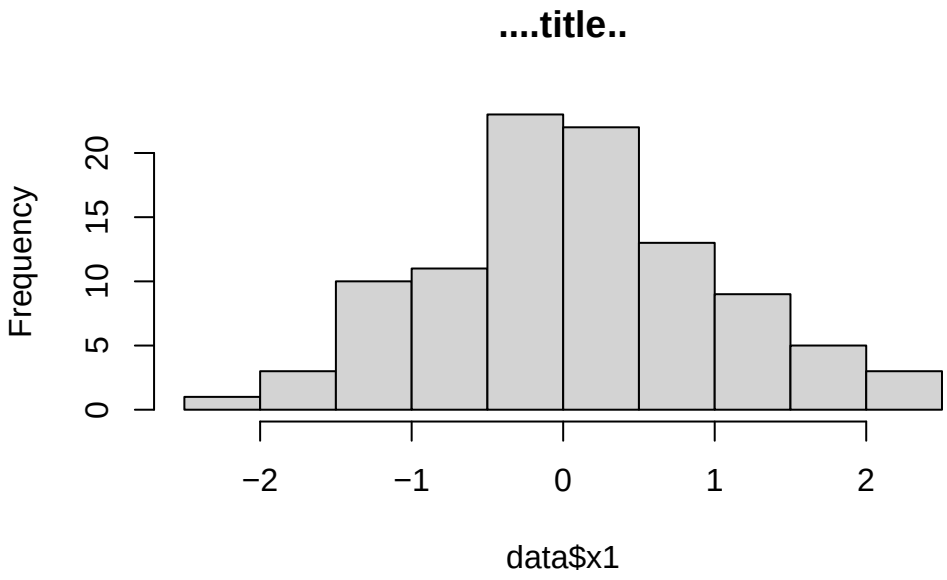
```
ggplot(data, aes(x = z, y = x1)) +
  geom_violin() +
  stat_summary(fun = median, geom = "point")
```


2
1
0
−1
−2
x1
L
R
z

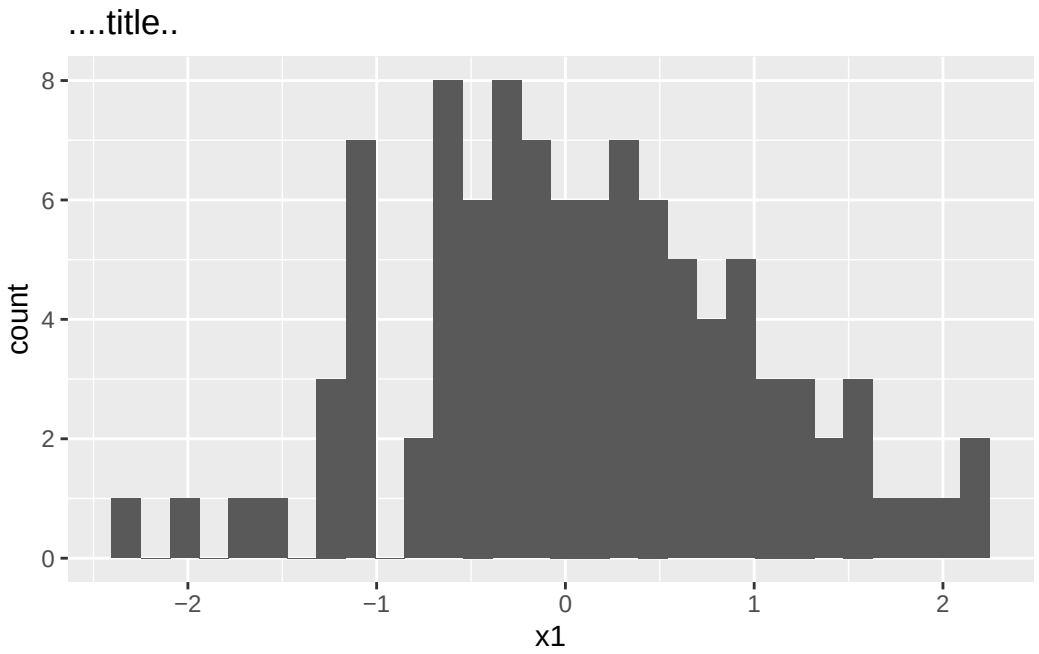
5　日本語の使用による豆腐化現象

macOS で軸ラベルや図のタイトルなどに日本語を使用すると, その部分は以下のように白い四角になる。

```
# の hist() の場合
hist(data$x1, main = "日本語の title です")
```




```
# ggplot2 の場合
ggplot(data = data, aes(x = x1)) +
  geom_histogram() +
  labs(title = "日本語の title です")
```



図のテキストは英語で書くことが多いため大した問題ではないが, どうしても日本語を使いたい場面もあるかもしれない。たとえば英語を使うと「格好つけるな」とか「わざわざ英語で書く意味が分からない」とか言われ, そういった意見を無碍にできないことがある (実話)。

そこで R で生成した図中に日本語を使うための方法を解説する。ただしこのあたりは情報の流れが早いため, この PDF の更新が追いついていない可能性がある。また OS のバージョンによって対応が異なるため, 特別な事情がなければ日本語フォントは使わないほうが賢明である。

具体的な対処法については, 片桐先生のこの記事がよくまとまっている。どうしても日本語を使いたくて困ったら参照してほしい。

6　練習問題

x2 の平均値と標準偏差を求めよう。

y3 の中央値を求めよう。

R 標準搭載の関数で x3 のヒストグラムを描いてみよう。

R 標準搭載の関数で x2 と y2 の散布図を描いてみよう。

psych の describe() を使って data の記述統計を計算してみよう。

ggplot2 を使って x4 のヒストグラムを描いてみよう。

ggplot2 を使って x4 と y4 の散布図を描いてみよう。

7　回答例

```r
# x2 の平均値と標準偏差を求めよう。
mean(data$x2)
sd(data$x2)

# y3 の中央値を求めよう。
median(data$y3)

# R 標準搭載の関数で x3 のヒストグラムを描いてみよう。
hist(data$x3)

# R 標準搭載の関数で x2 と y2 の散布図を描いてみよう。
plot(data$x2, data$y2)

# ggplot2 を使って x4 のヒストグラムを描いてみよう。
ggplot(data, aes(x = x4)) +
  geom_histogram()

# ggplot2 を使って x4 と y4 の散布図を描いてみよう。
ggplot(data, aes(x = x4, y = y4)) +
  geom_point()
ggplot(data, aes(x = x4, y = y4)) +
  geom_jitter()
```