Brief Introduction to R

どんぐり研究所　孫 建強

Brief Introduction to R

Contens

可視化

- 作図関数
- データ可視化

可視化

- 作図関数
- データ可視化

作図関数

R のグラフ作成関数には、高水準作図関数と低水準作図関数がある。高水準作図関数は、プロットエリアを初期化してからグラフを描く機能が備えられている。これに対して、低水準作図関数は、すでに存在しているプロットエリアに追加する形でグラフを描く機能が備えられている。

```
x <- c(1, 2, 3, 4, 5)
y <- c(1.2, 2.5, 3.4, 3.3, 2.8)
```

プロットエリアを初期化し線グラフを描く。▶

```
plot(x, y, type = 'l')
```

高水準作図関数

プロットエリアを初期化し、何も描かない。▶

```
plot(x, y, type = 'n')
```

高水準作図関数

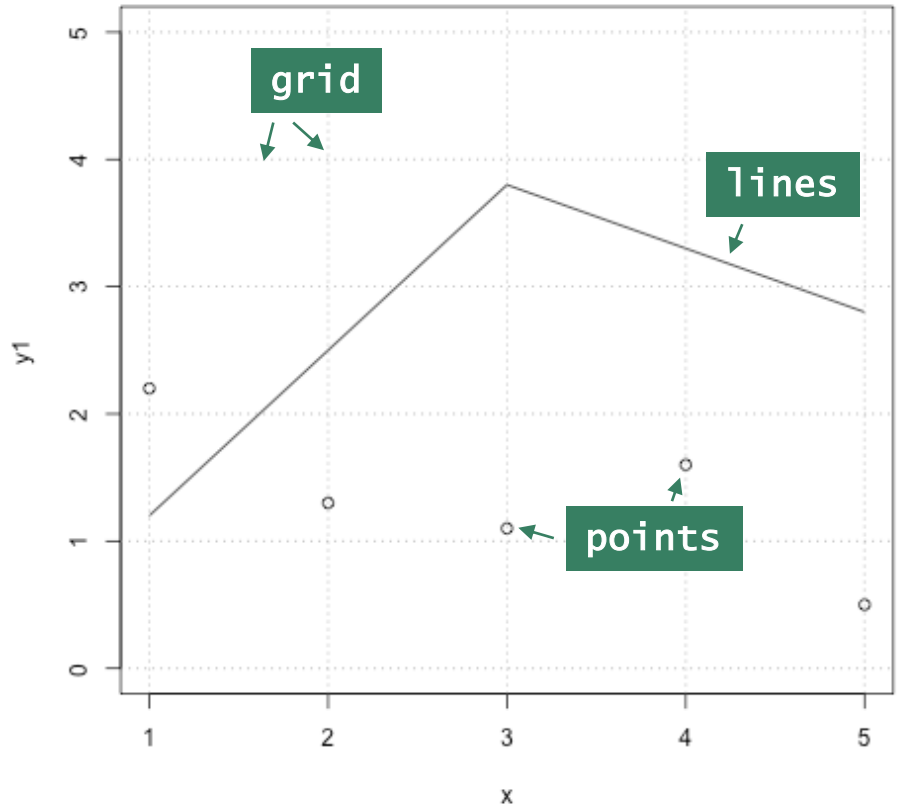
線グラフを追加する。▶

```
lines(x, y)
```

低水準作図関数

低水準作図関数

点や線など複数タイプのグラフを 1 つの図に描き入れるとき、高水準作図関数で空白のプロットエリアを作成してから、低水準関数で点や線を追加するとよい。




```
x <- c(1, 2, 3, 4, 5)
y1 <- c(1.2, 2.5, 3.8, 3.3, 2.8)
y2 <- c(2.2, 1.3, 1.1, 1.6, 0.5)

plot(x, y1, type = 'n',
     ylim = c(0, 5))

lines(x, y1)

points(x, y2)

grid(col = 'darkgray')
```

グラフ保存

グラフをファイルに書き出す場合は、プロットを行う関数群の前後に、ファイルを開く関数とファイルを閉じる関数を実行する。よく使う関数として、png、pdf、tiff 等がある。

```
x <- c(1, 2, 3, 4, 5)
y <- c(1.2, 2.5, 3.4, 3.3, 2.8)
```

PNG 描画デバイスを開く。▶

```
png('linepolot.png', 600, 500)
```

作図関数

```
plot(x, y)
lines(x, y)
grid()
```

ファイルを閉じる。▶

```
dev.off()
```

グラフ保存

グラフを画像として保存する場合、画像サイズを大きくすると、グラフの目盛りや凡例などの文字が著しく小さくなる。この際、解像度（res）を大きめな値に設定しておくと、それらの文字も大きくなり、見やすくなる。

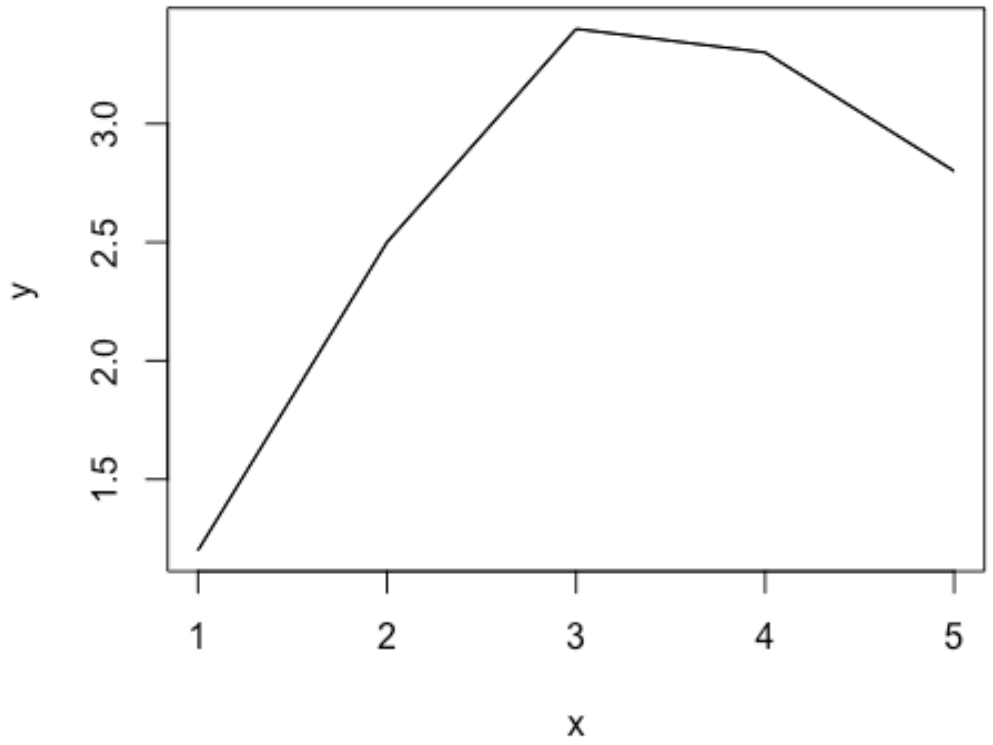
```
x <- c(1, 2, 3, 4, 5)
y <- c(1.2, 2.5, 3.4, 3.3, 2.8)




png('linepolot.png',
    1200, 800, res = 226)

plot(x, y)
lines(x, y)
grid()

dev.off()
```

可視化

- 作図関数
- データ可視化

線グラフ

高水準作図関数の plot 関数の type オプションに l（小文字エル）を指定することで、線グラフを描くことができる。低水準作図関数を使用する場合は、lines 関数を使用する。

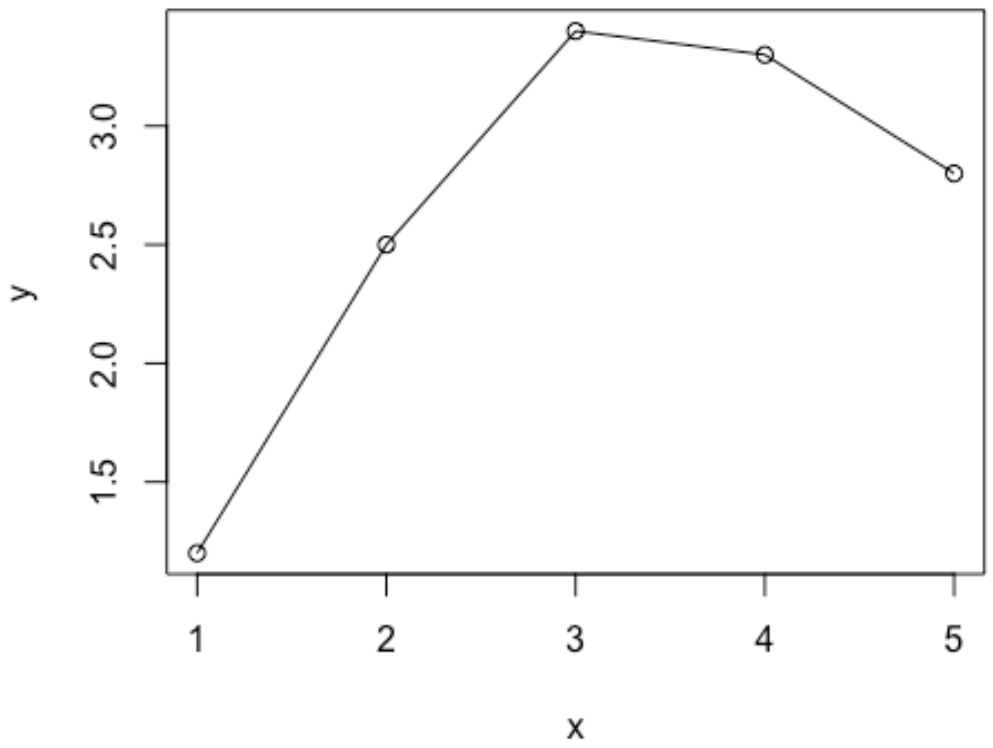



```
x <- c(1, 2, 3, 4, 5)
y <- c(1.2, 2.5, 3.4, 3.3, 2.8)
plot(x, y, type = 'l')
```

```
plot(x, y, type = 'n')
lines(x, y)
```

線グラフ

高水準作図関数の plot 関数の type オプションに l（小文字エル）を指定することで、線グラフを描くことができる。低水準作図関数を使用する場合は、lines 関数を使用する。

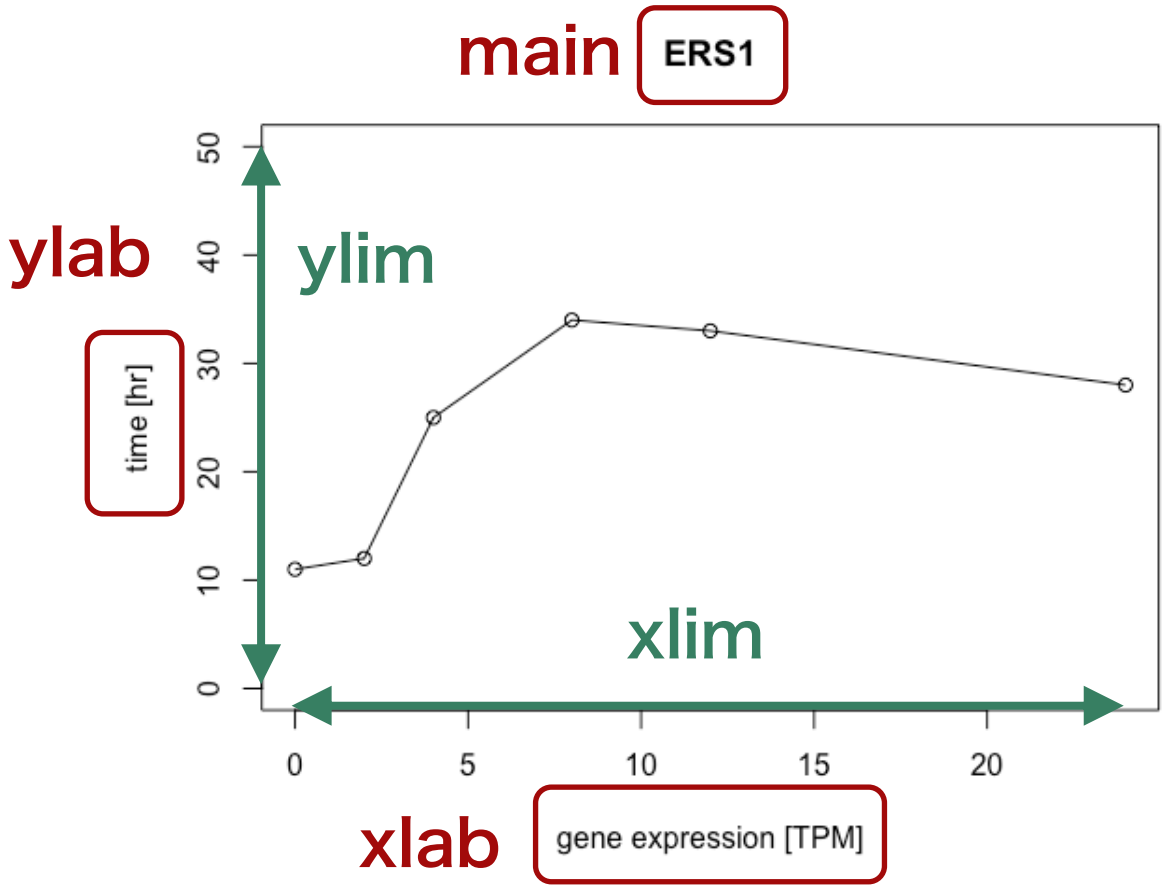



```
x <- c(1, 2, 3, 4, 5)
y <- c(1.2, 2.5, 3.4, 3.3, 2.8)
plot(x, y, type = 'o')
```

```
plot(x, y, type = 'n')
lines(x, y)
points(x, y)
```

座標軸

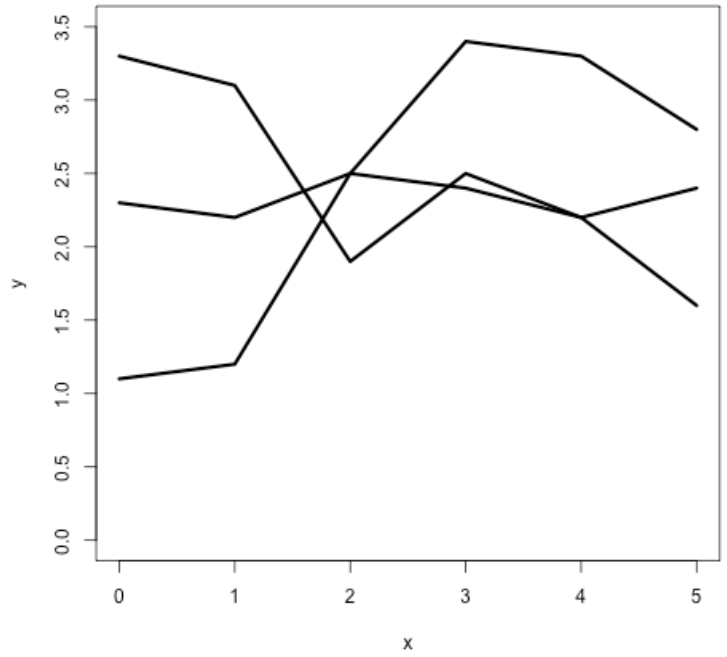
plot 関数にオプションを指定することで、縦軸および横軸の範囲や軸ラベルを調整することができる。




```
x <- c(0, 2, 4, 8, 12, 24)
y <- c(11, 12, 25, 34, 33, 28)

plot(x, y, type = 'n',
     main = 'ERS1',
     xlab = 'gene expression [TPM]',
     ylab = 'time [hr]',
     xlim = c(0, 24),
     ylim = c(0, 50))

lines(x, y)
points(x, y)
```

線グラフ

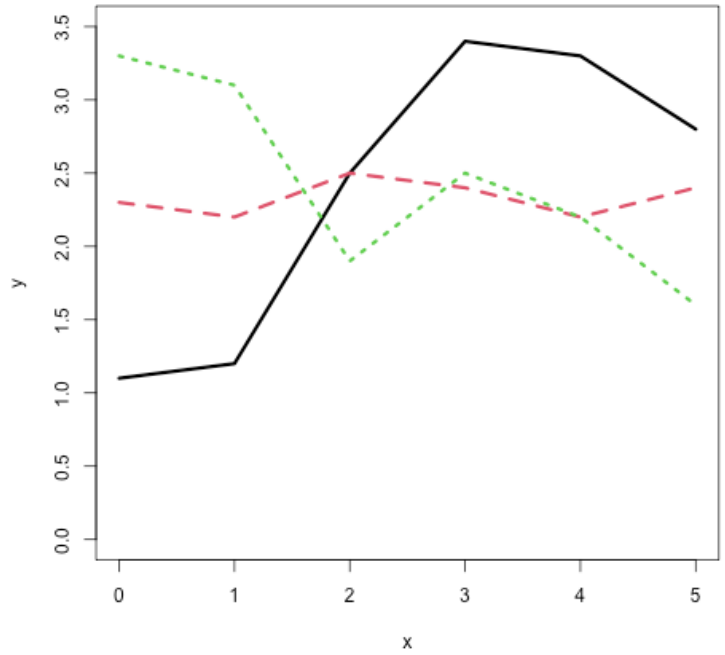
複数グラフを描く場合は、plot 関数でプロットエリアを初期化してから、lines 関数と points 関数などでグラフを必要な数だけ追加していく。




```
x <- c(0, 1, 2, 3, 4, 5)
y1 <- c(1.1, 1.2, 2.5, 3.4, 3.3, 2.8)
y2 <- c(2.3, 2.2, 2.5, 2.4, 2.2, 2.4)
y3 <- c(3.3, 3.1, 1.9, 2.5, 2.2, 1.6)

plot(x, y1, type = 'n',
     xlim = c(0, 5), ylim = c(0, 3.5),
     xlab = 'x', ylab = 'y')

lines(x, y1)
lines(x, y2)
lines(x, y3)
```

線グラフ

複数グラフを描く場合は、plot 関数でプロットエリアを初期化してから、lines 関数と points 関数などでグラフを必要な数だけ追加していく。

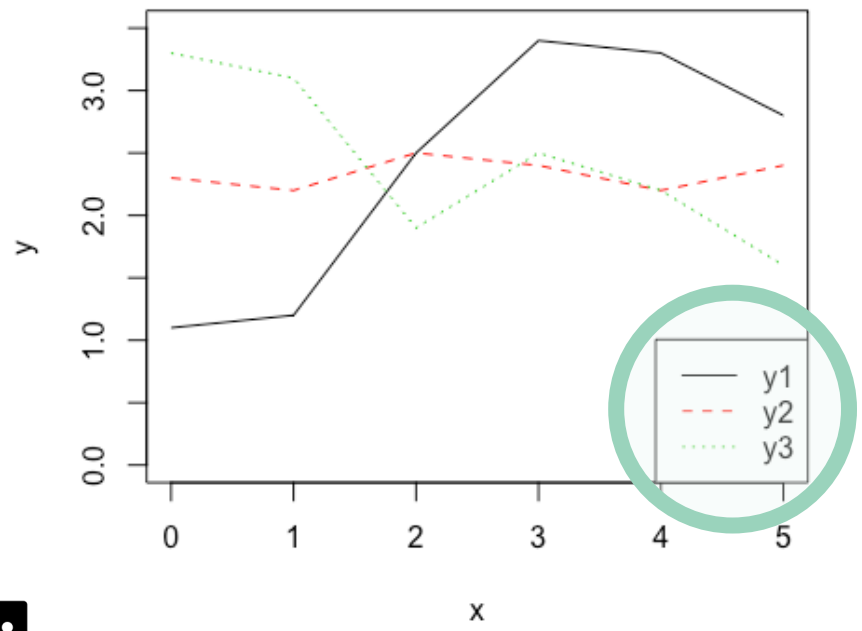



```
x <- c(0, 1, 2, 3, 4, 5)
y1 <- c(1.1, 1.2, 2.5, 3.4, 3.3, 2.8)
y2 <- c(2.3, 2.2, 2.5, 2.4, 2.2, 2.4)
y3 <- c(3.3, 3.1, 1.9, 2.5, 2.2, 1.6)

plot(x, y1, type = 'n',
     xlim = c(0, 5), ylim = c(0, 3.5),
     xlab = 'x', ylab = 'y')

lines(x, y1, lty = 1, col = 1)
lines(x, y2, lty = 2, col = 2)
lines(x, y3, lty = 3, col = 3)
```

lines 関数の lty オプションで線の種類、col オプションで色の種類を指定できる。

線グラフ

複数グラフを描く場合は、plot 関数でプロットエリアを初期化してから、lines 関数と points 関数などでグラフを必要な数だけ追加していく。




```
x <- c(0, 1, 2, 3, 4, 5)
y1 <- c(1.1, 1.2, 2.5, 3.4, 3.3, 2.8)
y2 <- c(2.3, 2.2, 2.5, 2.4, 2.2, 2.4)
y3 <- c(3.3, 3.1, 1.9, 2.5, 2.2, 1.6)

plot(x, y1, type = 'n',
     xlim = c(0, 5), ylim = c(0, 3.5),
     xlab = 'x', ylab = 'y')

lines(x, y1, lty = 1, col = 1)
lines(x, y2, lty = 2, col = 2)
lines(x, y3, lty = 3, col = 3)

legend('bottomright',
       legend = c('y1', 'y2', 'y3'),
          lty = c( 1,    2,    3),
          col = c( 1,    2,    3))
```

legend 関数を使用することで、グラフ凡例を追加することができる。この際に、lines 関数に指定した lty と col オプションも凡例と同じ順序で指定する必要がある。

線の種類

lty = 1 or 'solid'

lty = 2 or 'dashed'

lty = 3 or 'dotted'

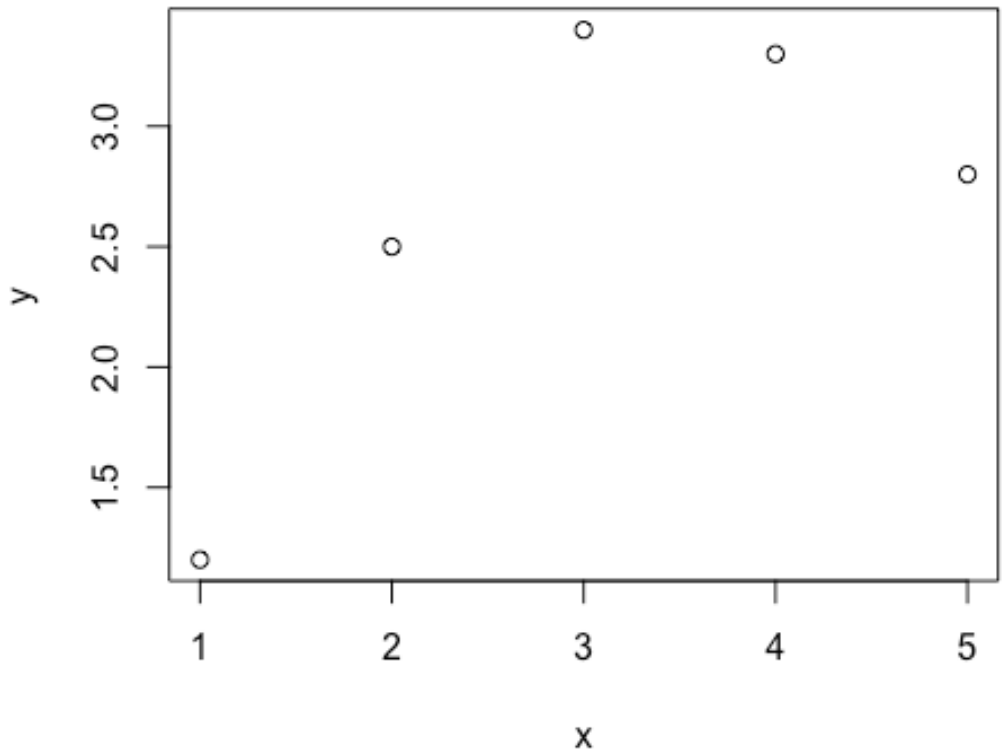
lty = 4 or 'dotdash'

lty = 5 or 'longdash'

lty = 6 or 'twodash'

散布図

高水準作図関数の plot 関数の type オプションに p を指定することで、点グラフを描くことができる。低水準作図関数を使用する場合は、points 関数を使用する。

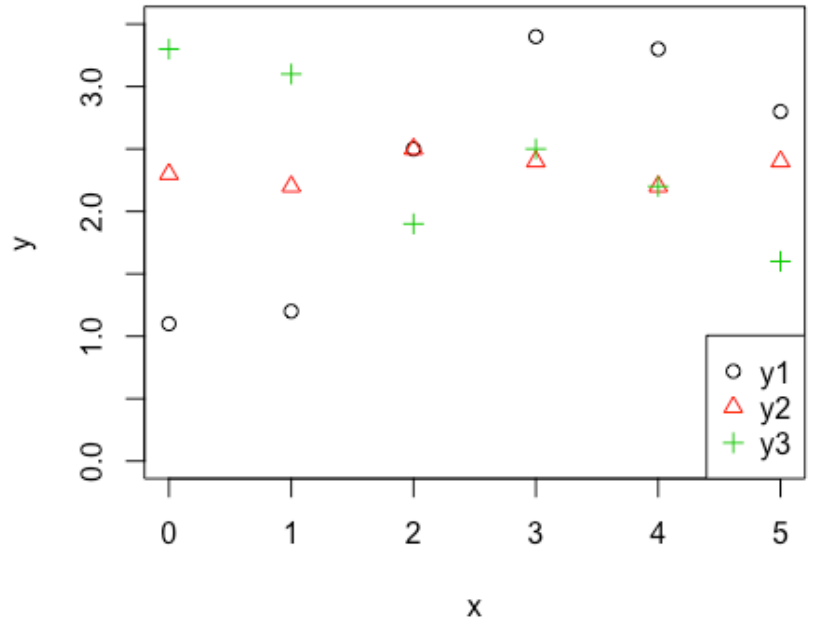



```
x <- c(1, 2, 3, 4, 5)
y <- c(1.2, 2.5, 3.4, 3.3, 2.8)
plot(x, y, type = 'p')
```

```
plot(x, y, type = 'n')
points(x, y)
```

散布図

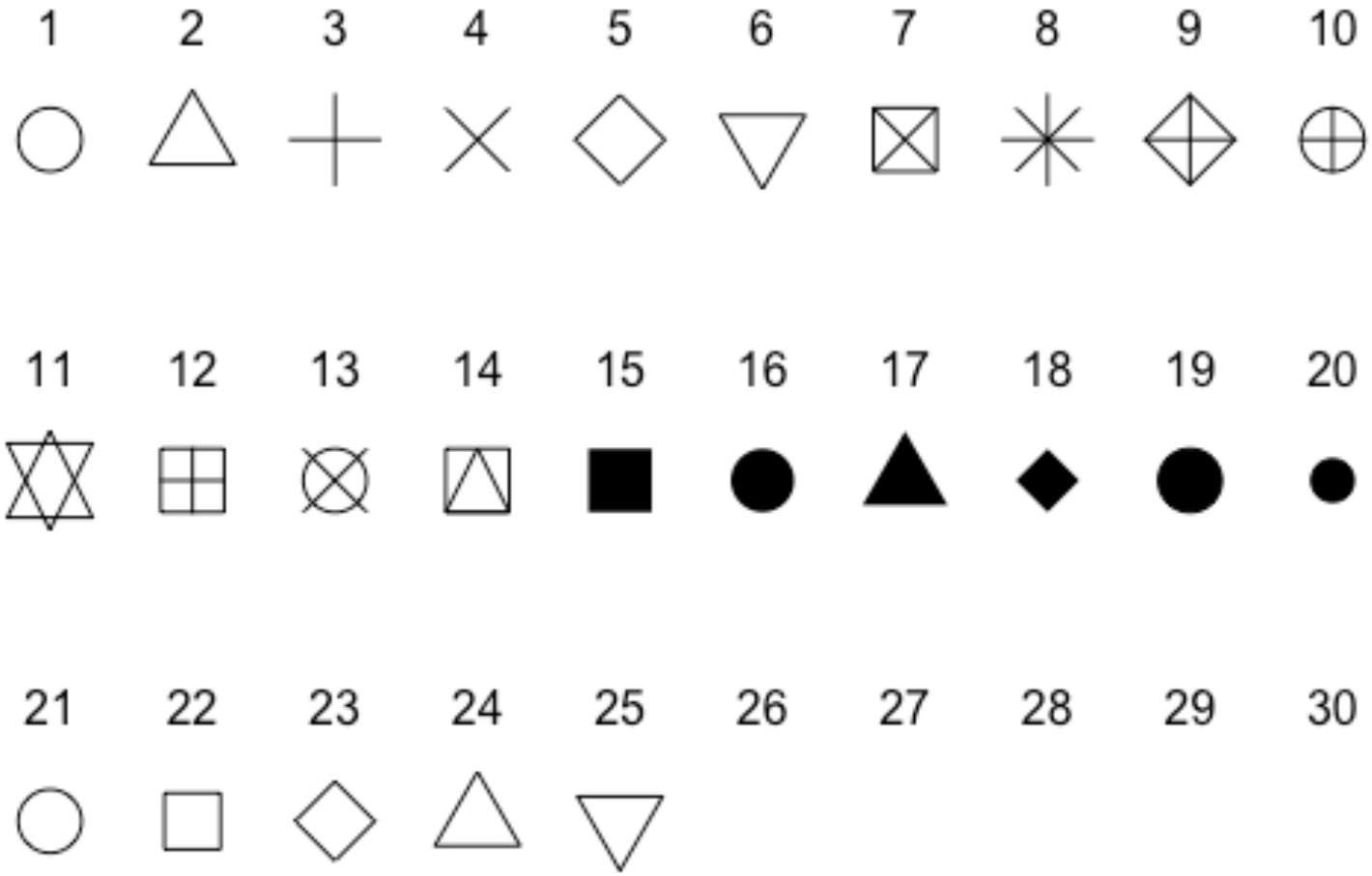
複数グラフを描く場合は、plot 関数でプロットエリアを初期化してから、points 関数などでグラフを必要な数だけ追加していく。




```
x <- c(0, 1, 2, 3, 4, 5)
y1 <- c(1.1, 1.2, 2.5, 3.4, 3.3, 2.8)
y2 <- c(2.3, 2.2, 2.5, 2.4, 2.2, 2.4)
y3 <- c(3.3, 3.1, 1.9, 2.5, 2.2, 1.6)

plot(x, y1, type = 'n',
     xlim = c(0, 5), ylim = c(0, 3.5),
     xlab = 'x', ylab = 'y')
points(x, y1, pch = 1, col = 1)
points(x, y2, pch = 2, col = 2)
points(x, y3, pch = 3, col = 3)

legend('bottomright',
       legend = c('y1', 'y2', 'y3'),
          pch = c(  1,    2,    3),
          col = c(  1,    2,    3))
```

legend 関数を使用することで、グラフ凡例を追加することができる。この際に、points 関数に指定した pch と col オプションも凡例と同じ順序で指定する必要がある。

点の種類


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30


0
10
1
11
2
12
3
13
4
14
5
15
6
16
7
17
8
18
9
19

問題 V1-1

10 min

xy 平面上に、-1 ≤ x ≤ 1 および -1 ≤ y ≤ 1 の区間に 1 万個の点をランダムにばらまいたとき、単位円内側に含まれる点のみを散布図で示せ。

```
x <- runif(10000, min = -1, max = 1)
y <- runif(10000, min = -1, max = 1)
```

問題 V1-2

5 min

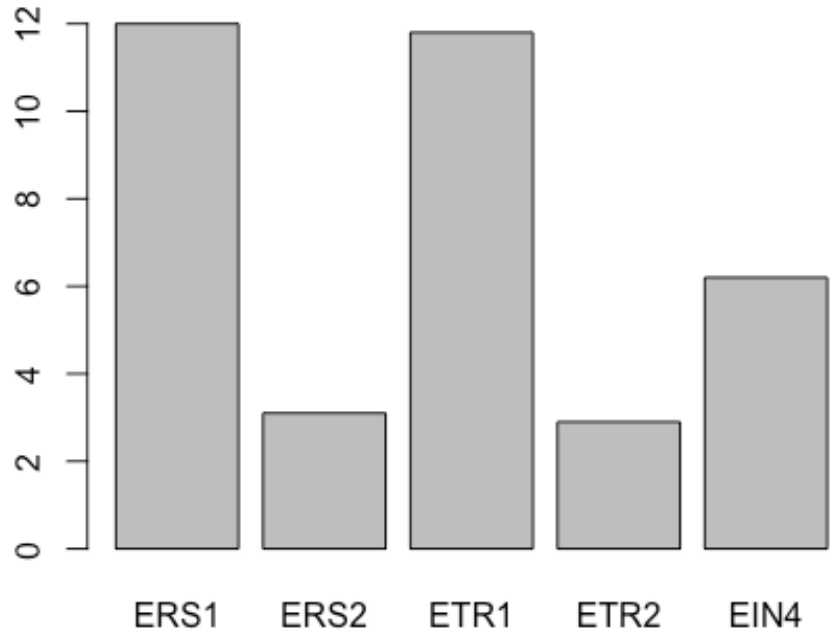
diversity_galapagos.txt ファイルを読み込んで、面積 Area を横軸の座標とし、種の数 Species を縦軸の座標とし、 面積と種数の関係をわかりやすく図示せよ。

```
filename <- 'diversity_galapagos.txt'
```

https://aabbdd.jp/data/diversity_galapagos.txt

棒グラフ

棒グラフは barplot 関数で描く。この際、横軸のラベルは names.arg オプションで与える。また、ベクトル y に names 属性が含まれている場合は、names.arg オプションを指定しなくても横軸のラベルが描かれる。

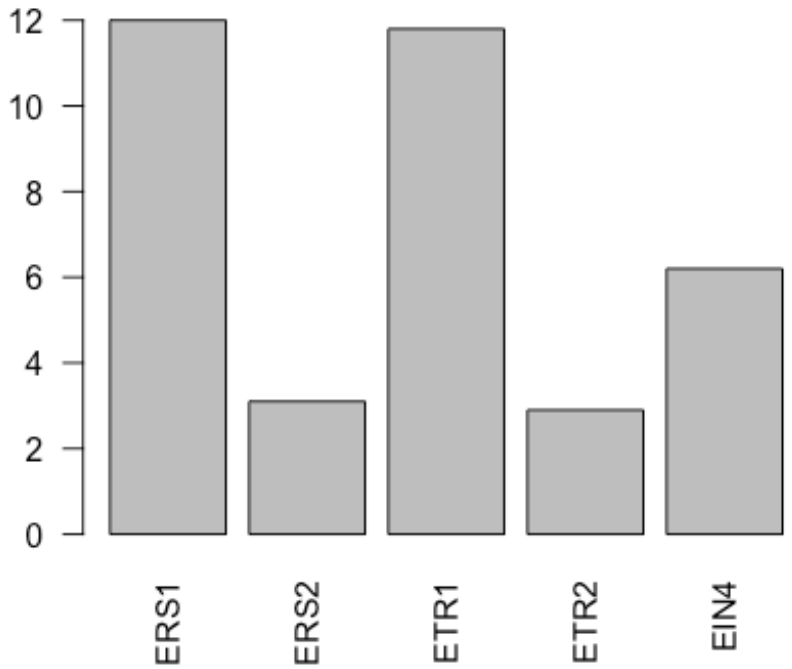



```
x <- c('ERS1', 'ERS2', 'ETR1',
       'ETR2', 'EIN4')
y <- c(12.0, 3.1, 11.8, 2.9, 6.2)
barplot(y, names.arg = x)
```

```
x <- c('ERS1', 'ERS2', 'ETR1',
       'ETR2', 'EIN4')
y <- c(12.0, 3.1, 11.8, 2.9, 6.2)
names(y) <- x
barplot(y)
```

棒グラフ

棒グラフは barplot 関数で描く。この際、横軸のラベルは names.arg オプションで与える。また、ベクトル y に names 属性が含まれている場合は、names.arg オプションを指定しなくても横軸のラベルが描かれる。

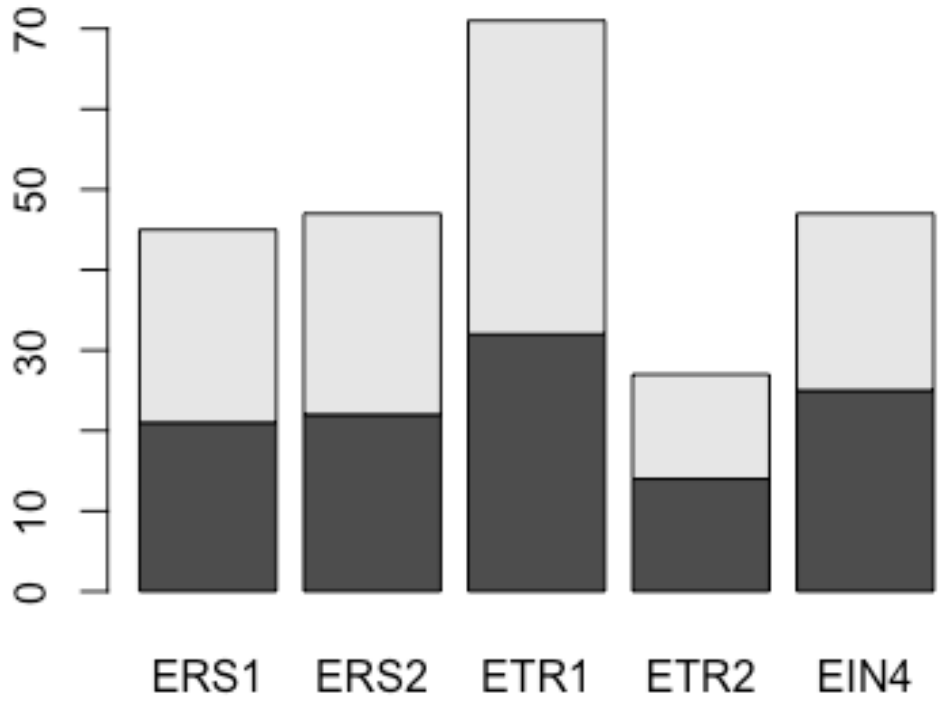



```
x <- c('ERS1', 'ERS2', 'ETR1',
       'ETR2', 'EIN4')
y <- c(12.0, 3.1, 11.8, 2.9, 6.2)
barplot(y, names.arg = x, las = 2)
```

```
x <- c('ERS1', 'ERS2', 'ETR1',
       'ETR2', 'EIN4')
y <- c(12.0, 3.1, 11.8, 2.9, 6.2)
names(y) <- x
barplot(y, las = 2)
```

棒グラフ

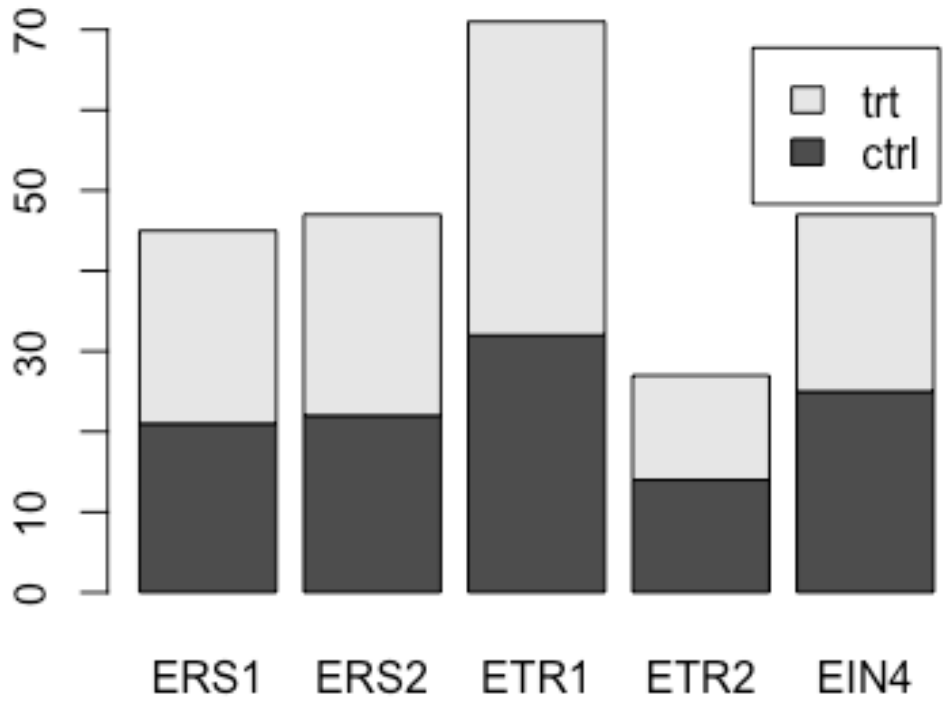
barplot 関数に行列を与えることで積み上げ棒グラフを描くことができるようになる。




```
x <- c('ERS1', 'ERS2', 'ETR1',
       'ETR2', 'EIN4')
ctrl <- c(21, 22, 32, 14, 25)
trt  <- c(24, 25, 39, 13, 22)

d <- rbind(ctrl, trt)
colnames(d) <- x
d
#      ERS1 ERS2 ETR1 ETR2 EIN4
# ctrl   21   22   32   14   25
# trt    24   25   39   13   22

barplot(d)
```

棒グラフ

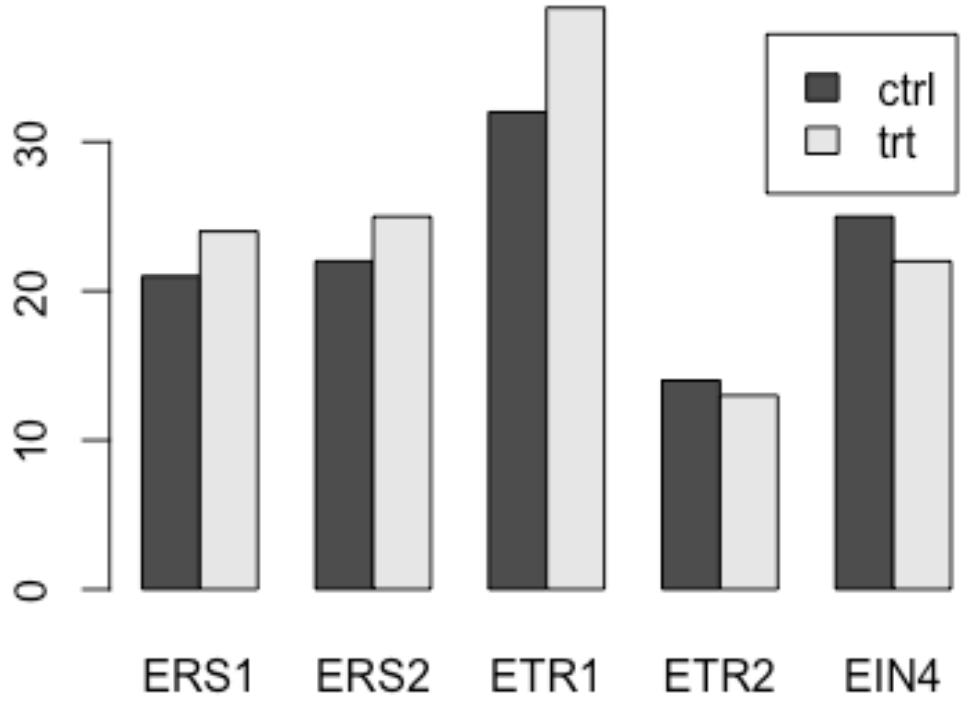
barplot 関数に行列を与えることで積み上げ棒グラフを描くことができるようになる。




```
x <- c('ERS1', 'ERS2', 'ETR1',
       'ETR2', 'EIN4')
ctrl <- c(21, 22, 32, 14, 25)
trt  <- c(24, 25, 39, 13, 22)

d <- rbind(ctrl, trt)
colnames(d) <- x
d
#      ERS1 ERS2 ETR1 ETR2 EIN4
# ctrl   21   22   32   14   25
# trt    24   25   39   13   22

barplot(d, legend = TRUE)
```

棒グラフ

barplot 関数に行列を与えることで積み上げ棒グラフを描くことができるようになる。beside = TRUE オプションを指定することで、横並びの棒グラフにすることができる。




```
x <- c('ERS1', 'ERS2', 'ETR1',
       'ETR2', 'EIN4')
ctrl <- c(21, 22, 32, 14, 25)
trt  <- c(24, 25, 39, 13, 22)

d <- rbind(ctrl, trt)
colnames(d) <- x
d
#      ERS1 ERS2 ETR1 ETR2 EIN4
# ctrl   21   22   32   14   25
# trt    24   25   39   13   22

barplot(d, beside = TRUE, legend = TRUE)
```

問題 V1-3

10 min

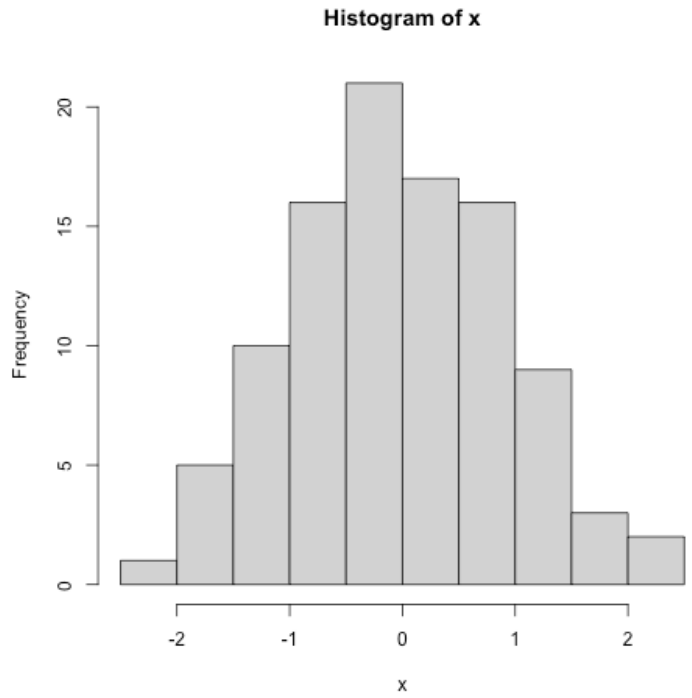
iris.txt ファイルを読み込んで、setosa, versicolor, virginica の Sepal.Length, Sepal.Width, Petal.Length, Petal.Width の平均値を棒グラフで示せ。

https://aabbdd.jp/data/iris.txt

```
filename <- 'iris.txt'
```

ヒストグラム

ヒストグラムは、R で標準実装された hist 関数で描くことができる。ヒストグラムの幅（個数）は、デフォルトでは、スタージェスの公式により決定されている。階級の区間は (a, b] のように決められている。



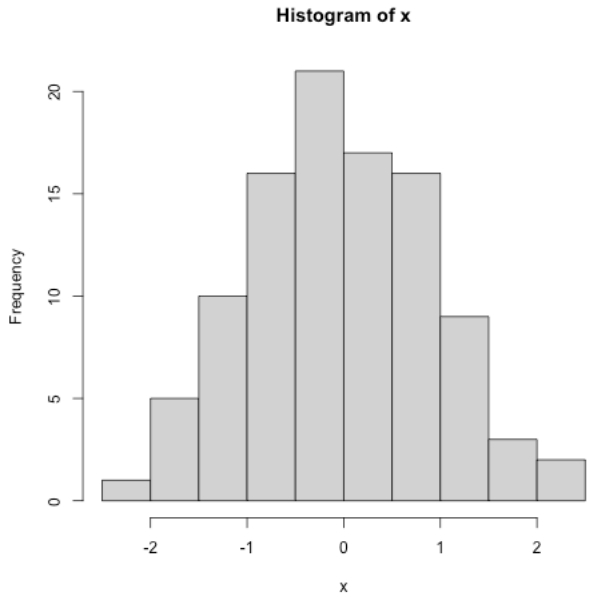

```
x <- rnorm(100)
hist(x)
```

ヒストグラム

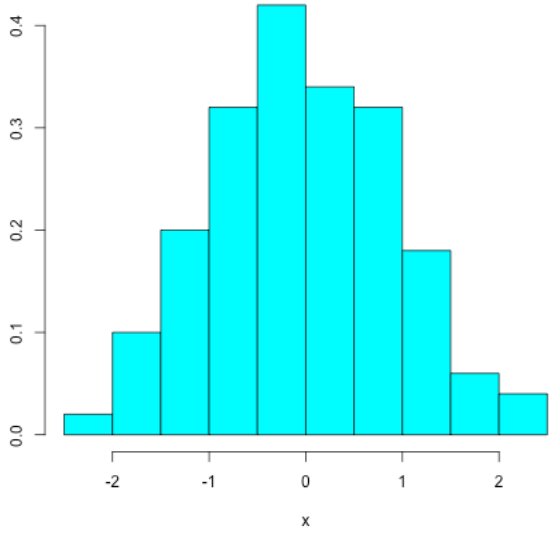
ヒストグラムを描く関数として hist 関数のほかに truehist 関数もよく使われる。 truehist 関数は、ヒストグラムの幅はスコットの公式によって決められる。また、階級の区間は [a, b) のように決められている。

```
x <- rnorm(100)
hist(x)


library(MASS)
truehist(x)
```

hist



truehist



問題 V1-4

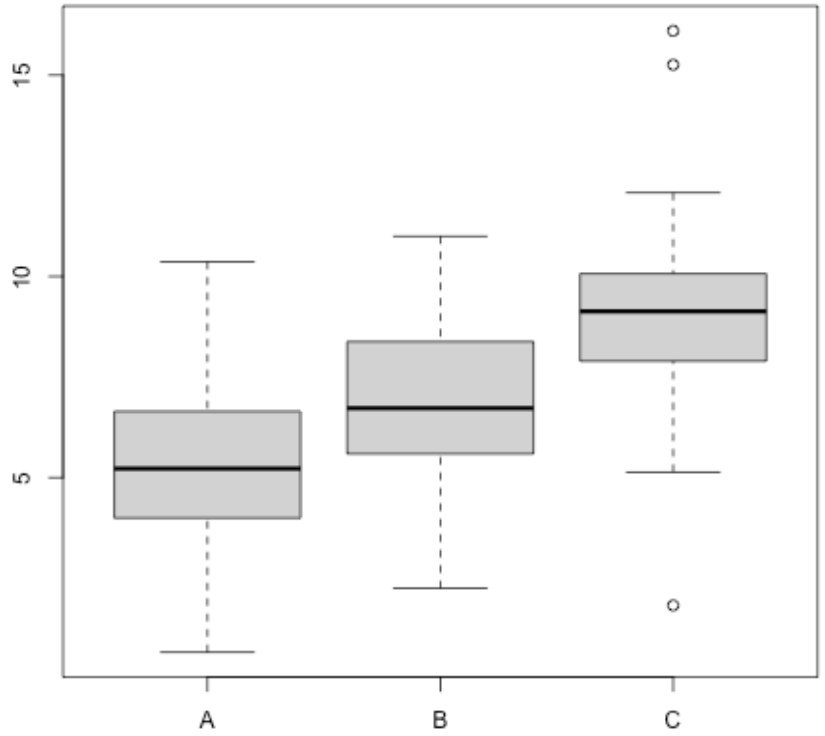
10 min

乱数を利用して、0 以上 1 未満の一様分布の最大値と最小値の差をヒストグラム を示せ。

```
x <- runif(1000, min = 0, max = 1)
M <- max(x)
m <- min(x)
d <- M - m
```

ボックスプロット

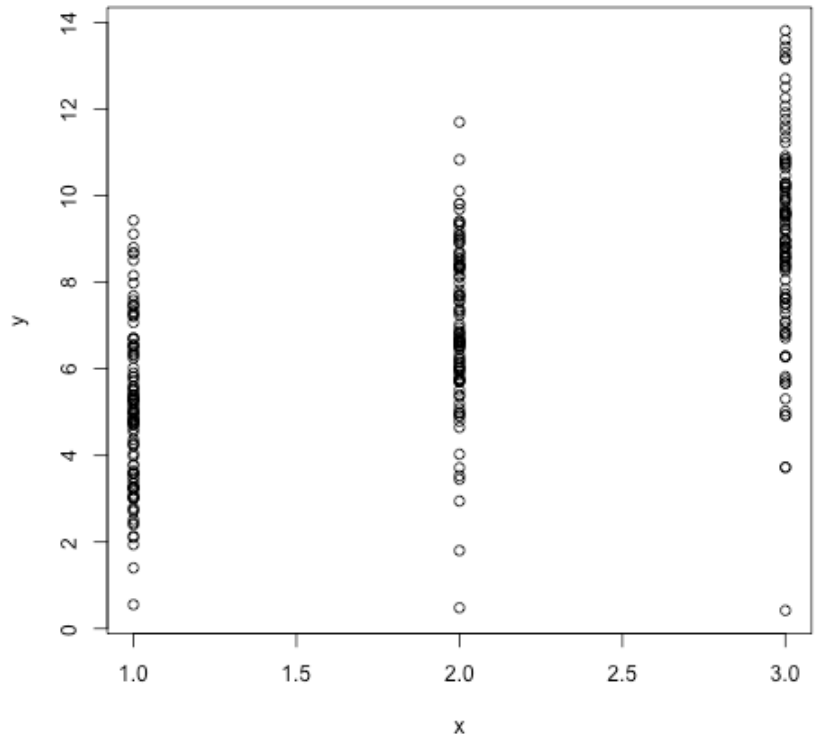
ボックスプロットは、boxplot 関数に複数のベクトルをあるいはデータフレームを代入して描く。




```
a <- rnorm(100, 5, 2)
b <- rnorm(100, 7, 2)
c <- rnorm(100, 9, 2)

boxplot(a, b, c,
        names = c('A', 'B', 'C'))


m <- cbind(A = a, B = b, C = c)
boxplot(m)
```

jitter

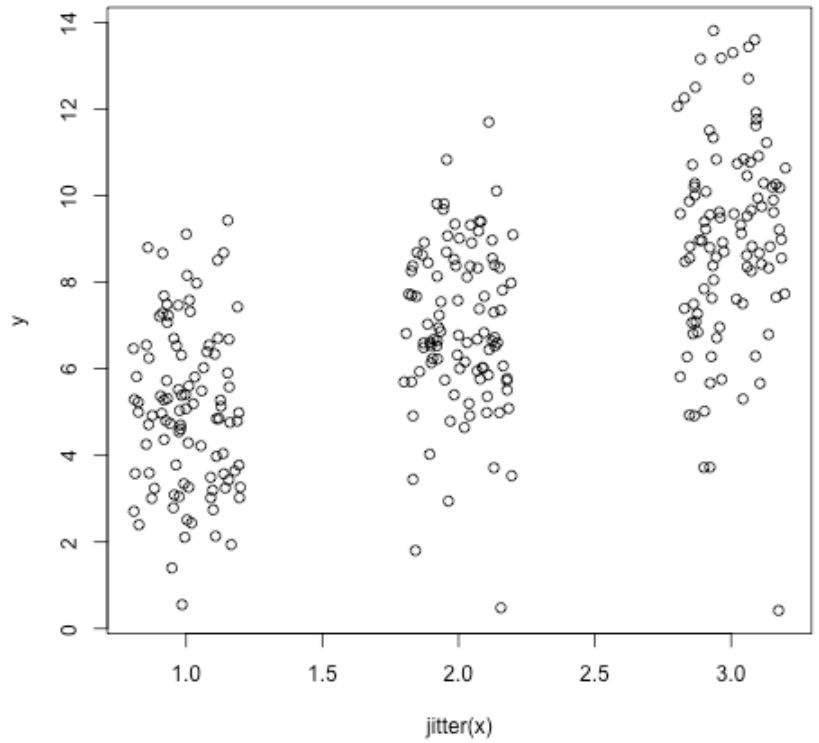
実際の値を jitter 関数に代入することで、実際の値にノイズが加わる。このことを利用して横軸に乱数を加えて jitter プロットを描くことができる。




```
a <- rnorm(100, 5, 2)
b <- rnorm(100, 7, 2)
c <- rnorm(100, 9, 2)

x <- rep(1:3, each = 100)
y <- c(a, b, c)

plot(x, y)
```

jitter

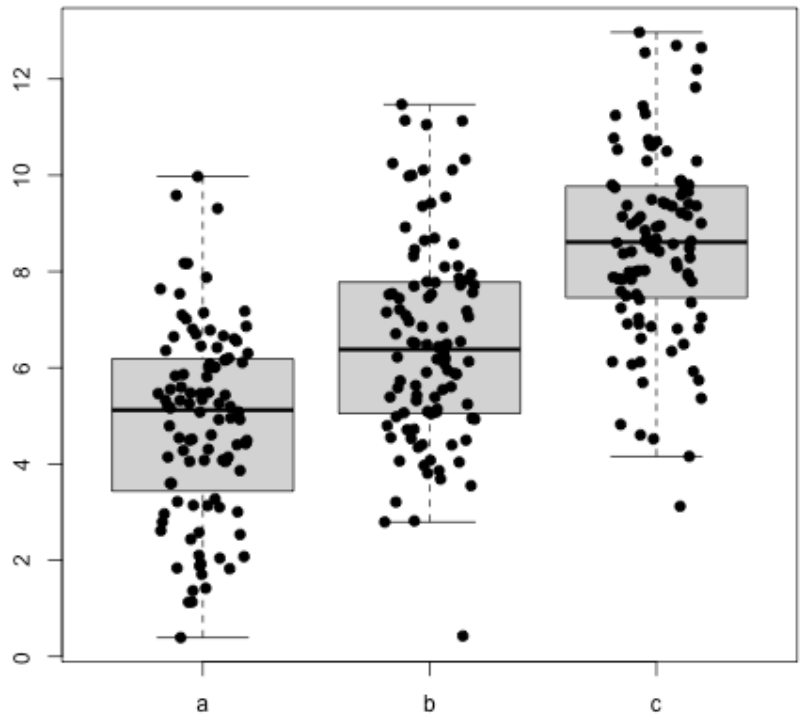
実際の値を jitter 関数に代入することで、実際の値にノイズが加わる。このことを利用して横軸に乱数を加えて jitter プロットを描くことができる。




```
a <- rnorm(100, 5, 2)
b <- rnorm(100, 7, 2)
c <- rnorm(100, 9, 2)

x <- rep(1:3, each = 100)
y <- c(a, b, c)

plot(jitter(x), y)
```

jitter

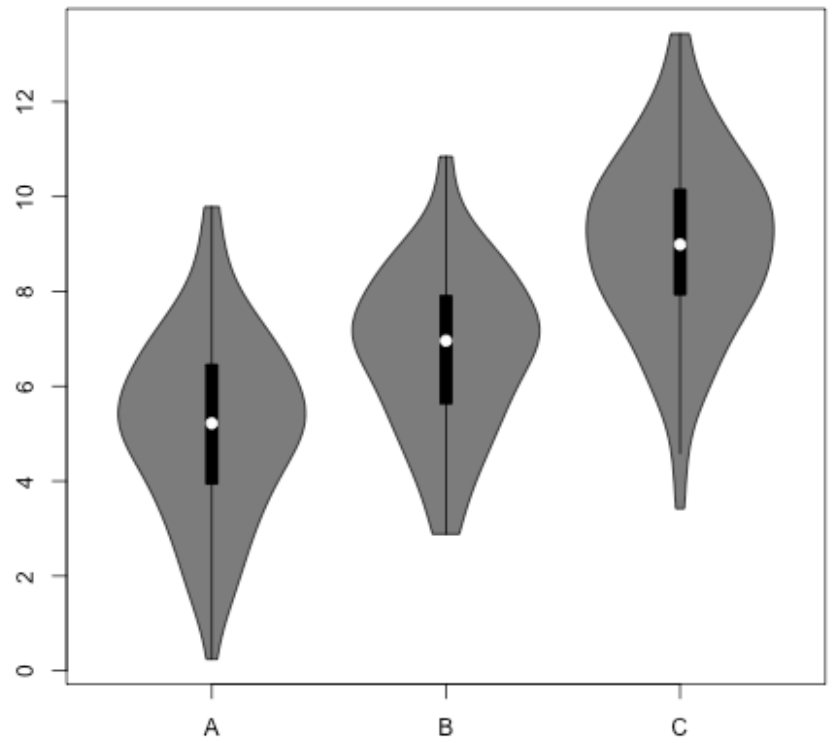
ボックスプロットを描いてから jitter を描くと、次のようなグラフが得られる。




```
a <- rnorm(100, 5, 2)
b <- rnorm(100, 7, 2)
c <- rnorm(100, 9, 2)

y <- cbind(a, b, c)

boxplot(y, outline = F)

x <- rep(1:3, each = 100)
x2 <- jitter(x)
y2 <- c(a, b, c)
points(x2, y2, pch = 19)
```

ヴァイオリンプロット

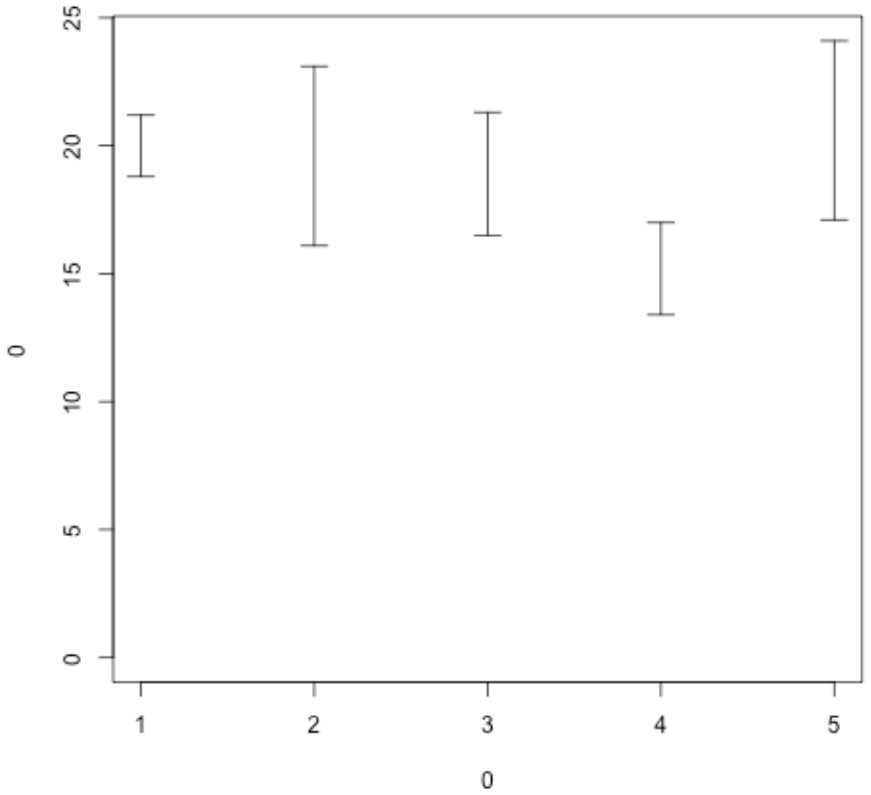
ボックスプロットの上にデータの推定密度を描き入れることもできる。このようなグラフはヴァイオリンプロットと呼ばれている。R の vioplot パッケージ中の関数を利用して描くことができる。




```
library(vioplot)

a <- rnorm(100, 5, 2)
b <- rnorm(100, 7, 2)
c <- rnorm(100, 9, 2)

vioplot(a, b, c,
        names = c('A', 'B', 'C'))
```

エラーバー

エラーバーを描くには矢印を描く arrows 関数を利用する。この際に、矢印の矢の部分の角度を 90 度にするとエラーバーのように見える。




```
m <- c(20.0, 19.6, 18.9, 15.2, 20.6)
s <- c( 1.2,  3.5,  2.4,  1.8,  3.5)

plot(0, 0, xlim = c(1, 5),
           ylim = c(0, max(m + s)),
     type = 'n')

arrows(1:5, m - s,
       1:5, m + s,
       code = 3,
       lwd = 1, angle = 90, length = 0.1)
```

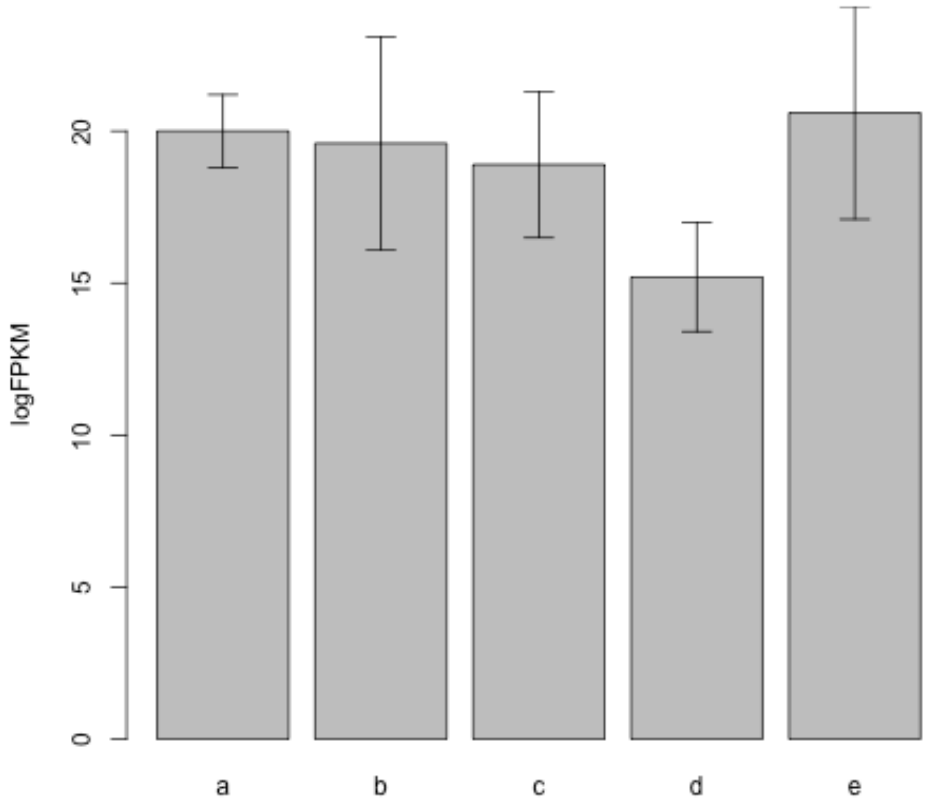
code = 1 ⟵

code = 2 ⟶

code = 3 ⟷

エラーバー

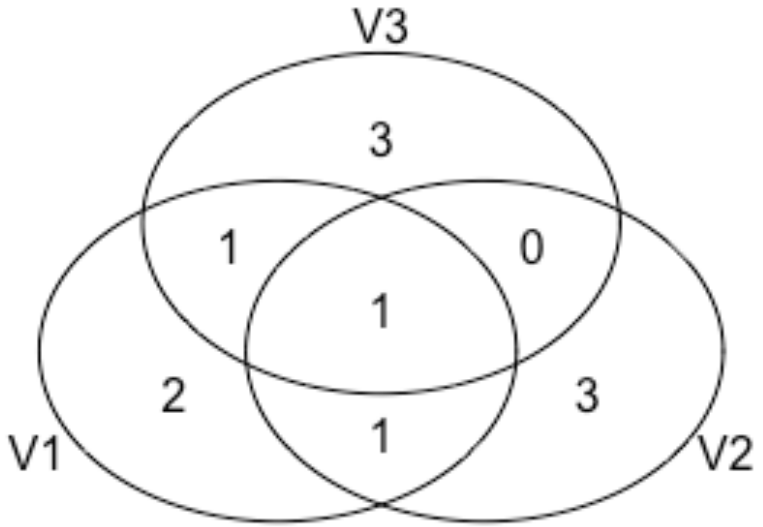
棒グラフを描いた後にエラーバーを描けば、エラーバー付きの棒グラフを作成することができる。




```
label <- c("a", "b", "c", "d", "e")
m <- c(20.0, 19.6, 18.9, 15.2, 20.6)
s <- c( 1.2,  3.5,  2.4,  1.8,  3.5)
names(m) <- label

b <- barplot(m, ylab = "logFPKM",
             ylim = c(0, max(m + s)))
b
# [,1]
# [1,]  0.7
# [2,]  1.9
# [3,]  3.1
# [4,]  4.3
# [5,]  5.5

arrows(b, m - s,
       b, m + s,
       code = 3,
       lwd = 1, angle = 90, length = 0.1)
```

ベン図

ベン図を描く関数は、R に標準搭載されていない。そのため、ベン図を描くに gplots パッケージ中の venn 関数を使用する。

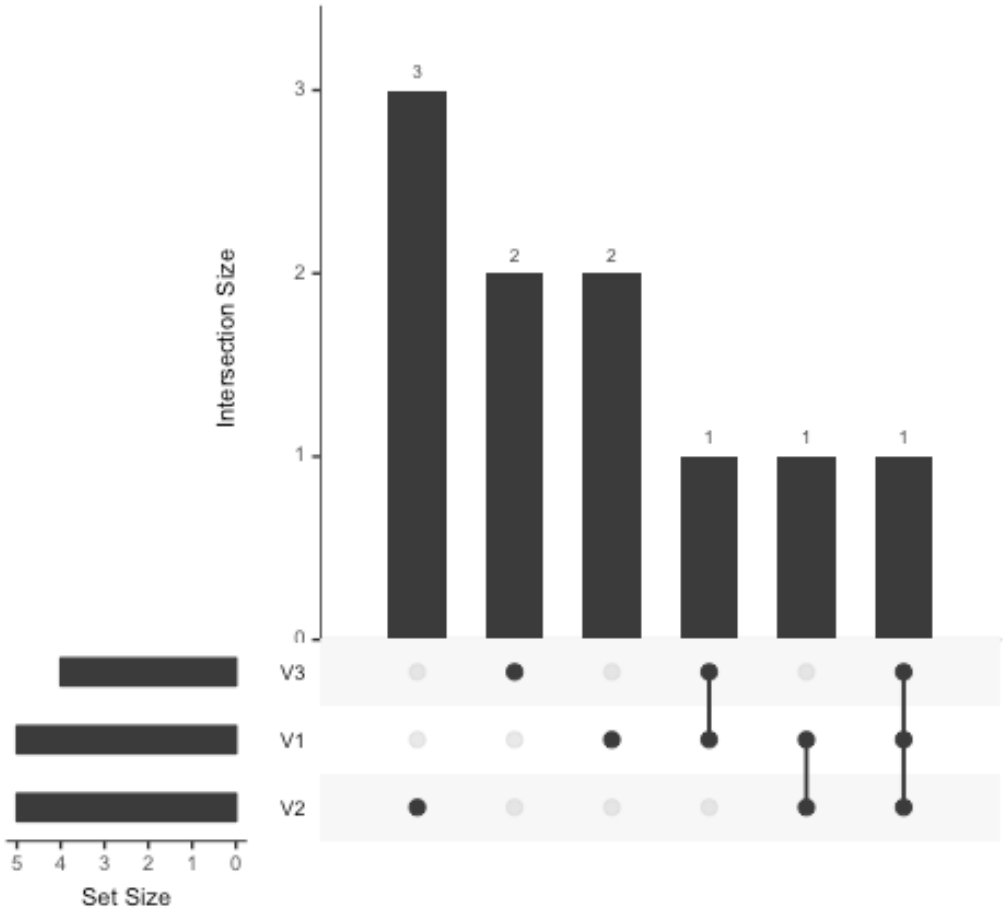



```
library(gplots)
x1 <- c("AA", "BB", "CC", "DD", "EE")
x2 <- c("AA", "BB", "XX", "YY", "ZZ")
x3 <- c("AA", "PP", "QQ", "RR", "EE")

data <- list(V1 = x1, V2 = x2, V3 = x3)
venn(data)
```

ベン図を描くとき、venneuler や VennDiagram などのパッケージを使うことで、自由度の高いベン図を描くことができる。

UpSet 図

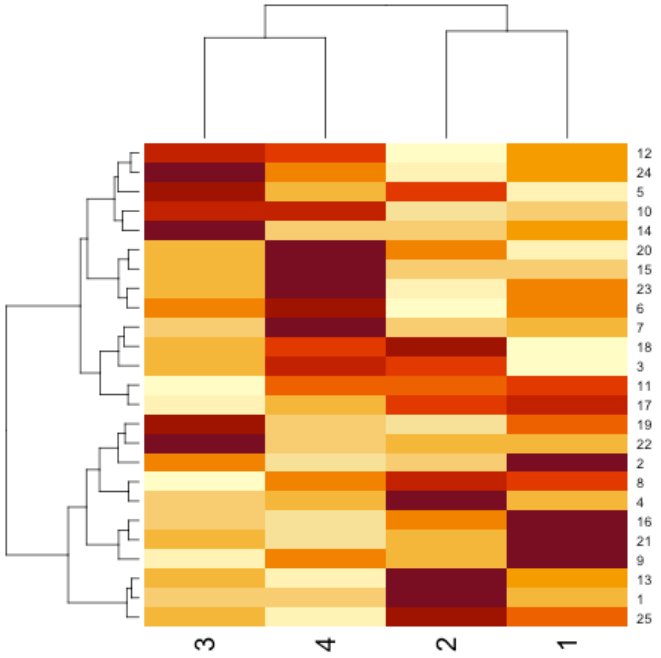
集合同士の重なり具合をプロットするために、ベン図の代わりに UpSet プロットを使用することもある。




```
library(UpSetR)

x1 <- c("AA", "BB", "CC", "DD", "EE")
x2 <- c("AA", "BB", "XX", "YY", "ZZ")
x3 <- c("AA", "PP", "QQ", "EE")
data <- list(V1 = x1, V2 = x2, V3 = x3)

upset(fromList(data), order.by = "freq")
```

ヒートマップ

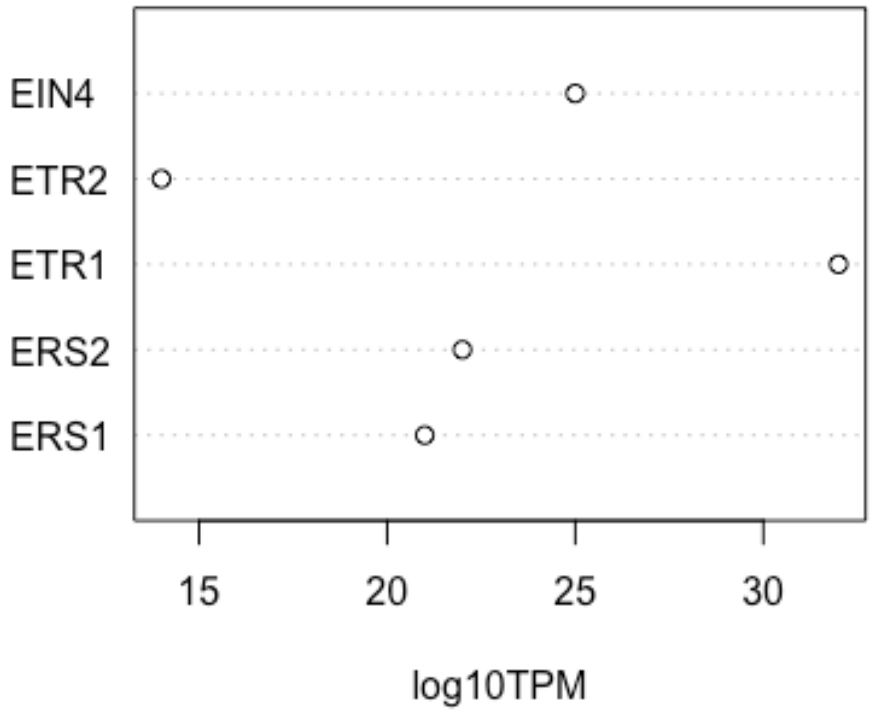
ヒートマップは、行列型データを heatmap 関数に代入することで描くことができる。ヒートマップでクラスタリング結果を示す際に、クラスタリングを行うための距離計算やクラスタリング計算は、distfun と hclustfun 関数で指定することができる。




```
r <- matrix(runif(100), ncol = 4)
heatmap(r)



d <- function(x) {
          dist(x, method = 'euclidean')
}
h <- function(x) {
          hclust(x, method = 'ward')
}
heatmap(r, distfun = d, hclustfun = h)
```

dotchart

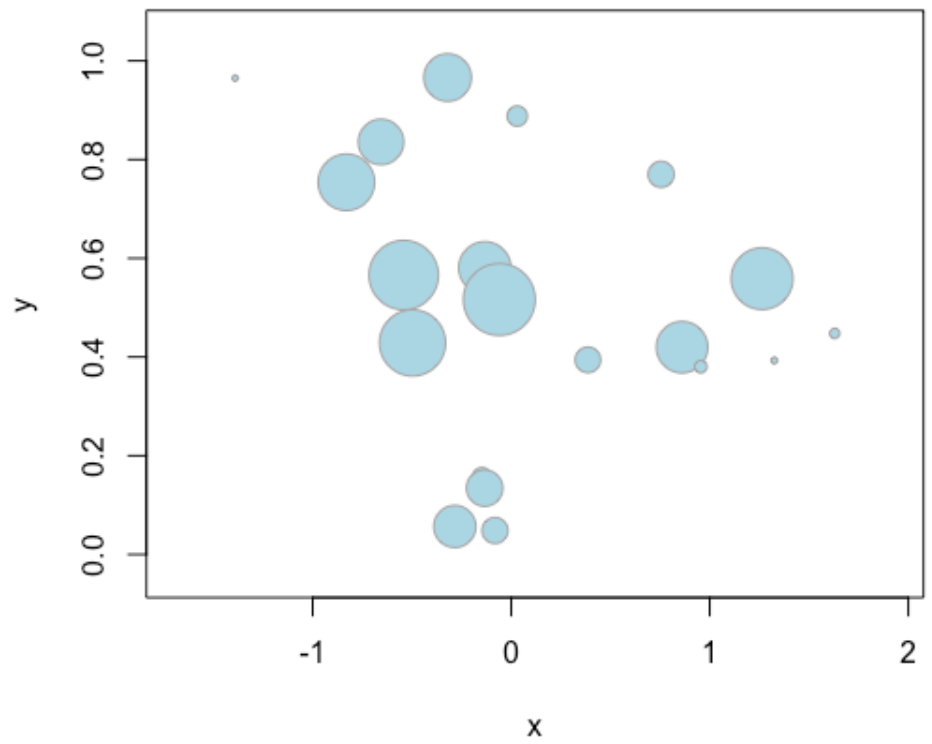
dotchart 関数を使用することで、横軸が連続量、縦軸が離散量であるようなデータをわかりやすくプロットすることができる。




```
y <- c('ERS1', 'ERS2', 'ETR1',
       'ETR2', 'EIN4')
x <- c(21, 22, 32, 14, 25)
names(x) <- y

x
# ERS1 ERS2 ETR1 ETR2 EIN4
#   21   22   32   14   25

dotchart(x, xlab = 'log10TPM')
```

symbols

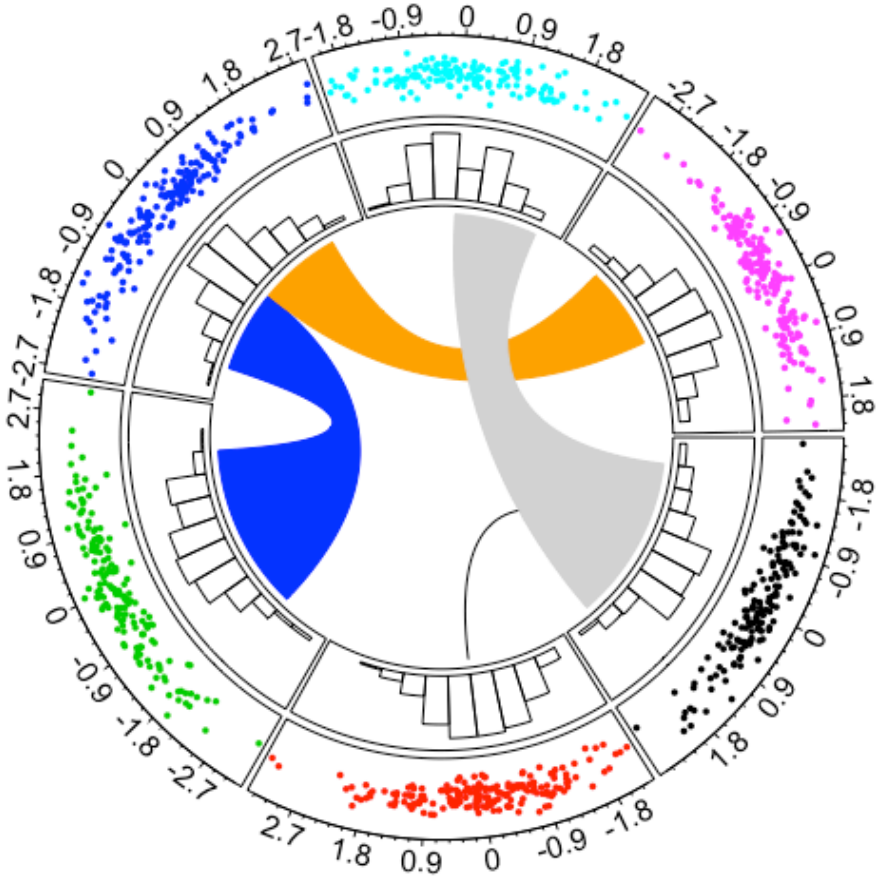
散布図は、symbols 関数を使用しても描くことができる。右は x 座標と y 座標を symbols 関数に与えて散布図を描く例である。この際に、点の大きさを変数 r の値に応じて描くことにする。




```
x <- rnorm(20)
y <- runif(20)
r <- runif(20)

symbols(x, y,
        circles = r,
        inches = 0.2,
        fg = 'darkgray',
        bg = 'lightblue',
        xlab = 'x', ylab = 'y')
```

circos

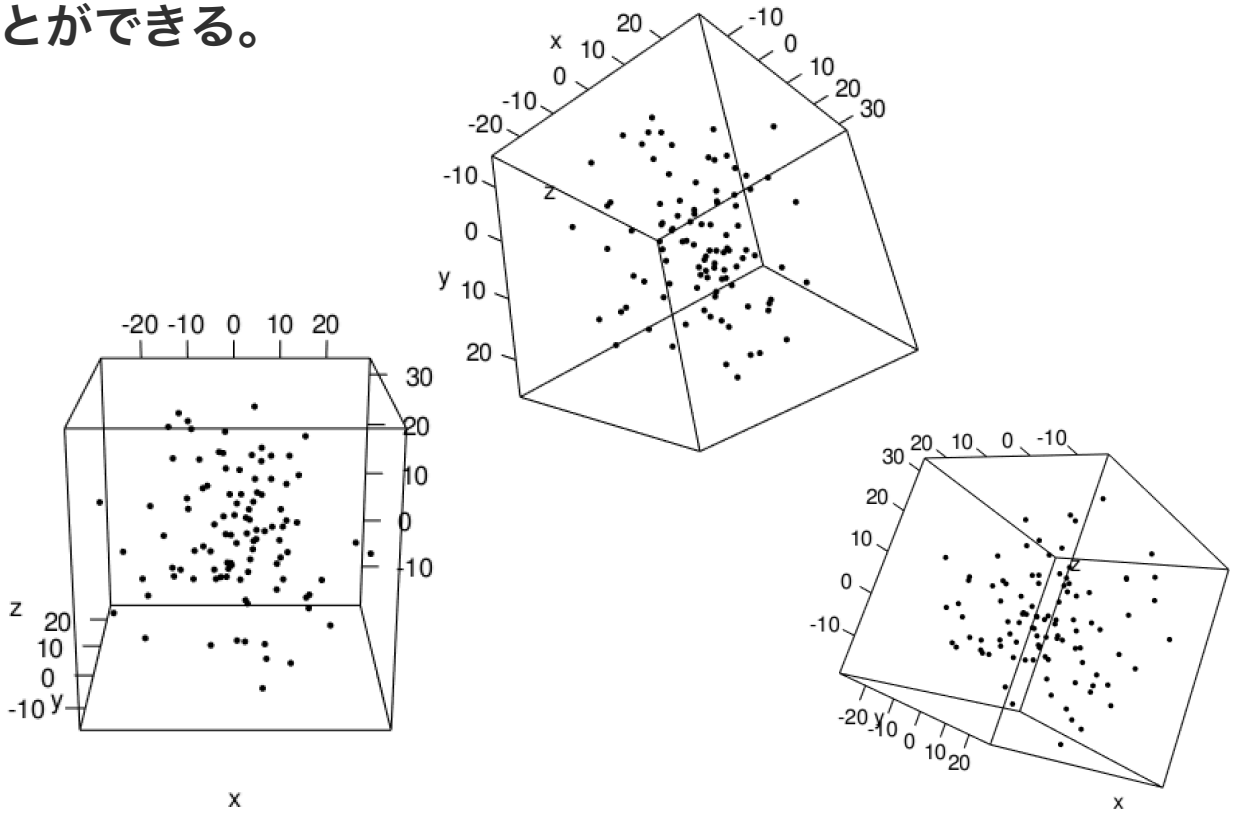
ゲノム構造の可視化によく使われている circos プロットを描くには circlize パッケージを利用する。使い方は少し複雑であるので、ここでは詳細な説明を割愛する。



```
library(circlize)
n <- 1000
d <- data.frame(
 chr = sample(c("chr1",
      "chr2", "chr3", "chr4",
      "chr5", "chr6"),
 n, replace = TRUE),
 x = rnorm(n), y = rnorm(n))

head(d)
## chr x y
## 1 chr4 0.1361047 -0.9054468
## 2 chr6 1.3881900 0.2430469

circos.initialize
circos.trackPlotRegion
circos.trackPoints
circos.trackHist
circos.link
  :
```

3D 散布図

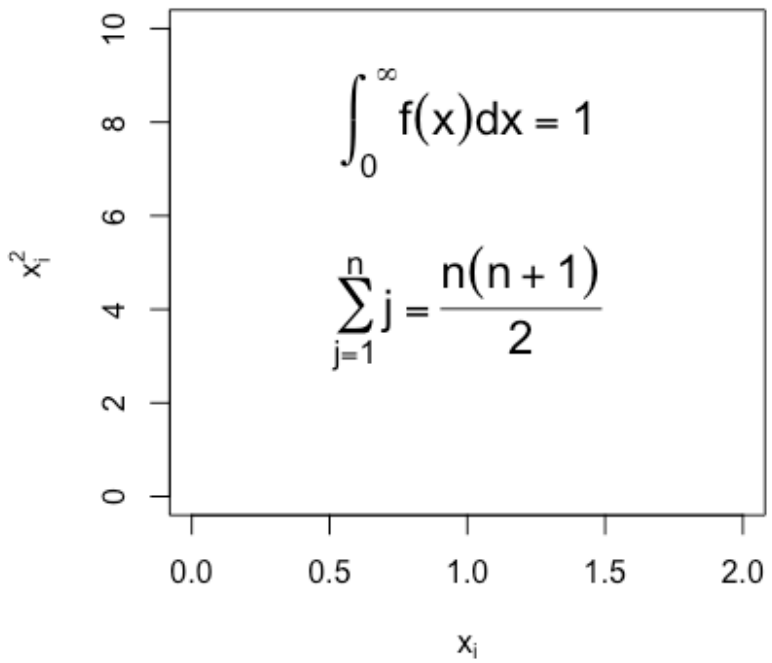
R で 3 次元の散布図を描くには、rgl パッケージの plot3d 関数や scatterplot3d パッケージの scatterplot3d 関数を使う。このうち、plot3d 関数を使用して描いたグラフはマウスで回転させたりすることができる。




```
library(rgl)

x <- rnorm(100, 2, 10)
y <- rnorm(100, 4, 10)
z <- rnorm(100, 6, 10)

plot3d(x, y, z)
```

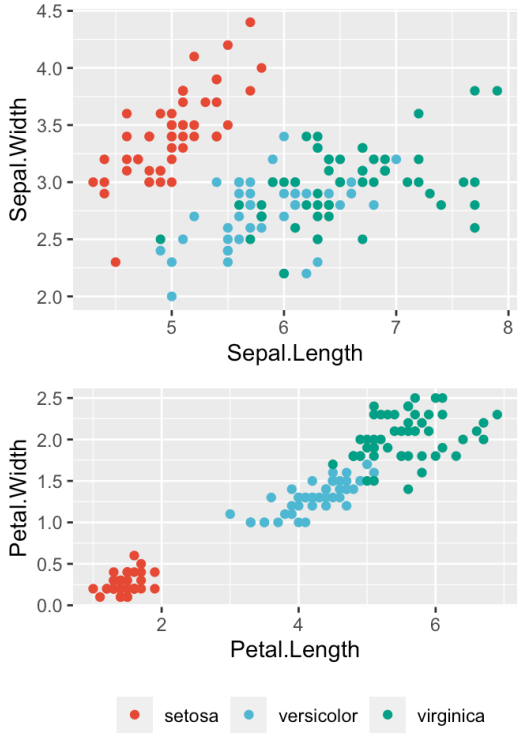
数式

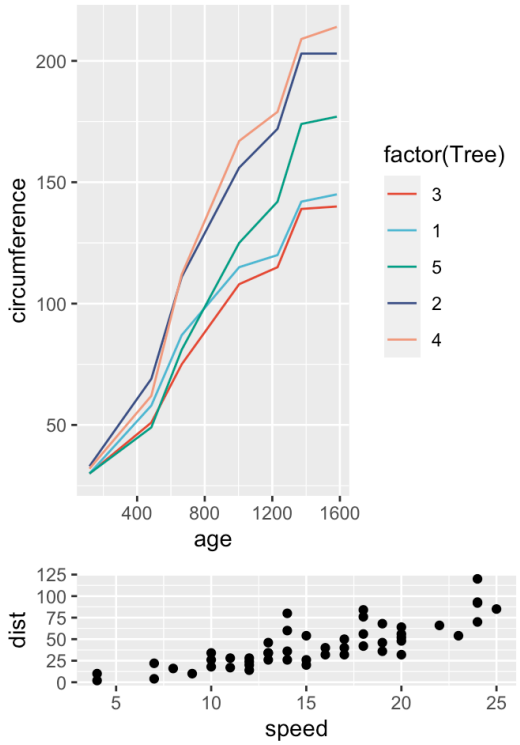
R のグラフ上に数式などを表示させたい場合は expression 関数を使用する。軸ラベルに数式を表示させたい場合は、plot などの関数の xlab や ylab に expression 関数を代入する。グラフの中に数式を表示させたい場合は、text 関数などを使う。

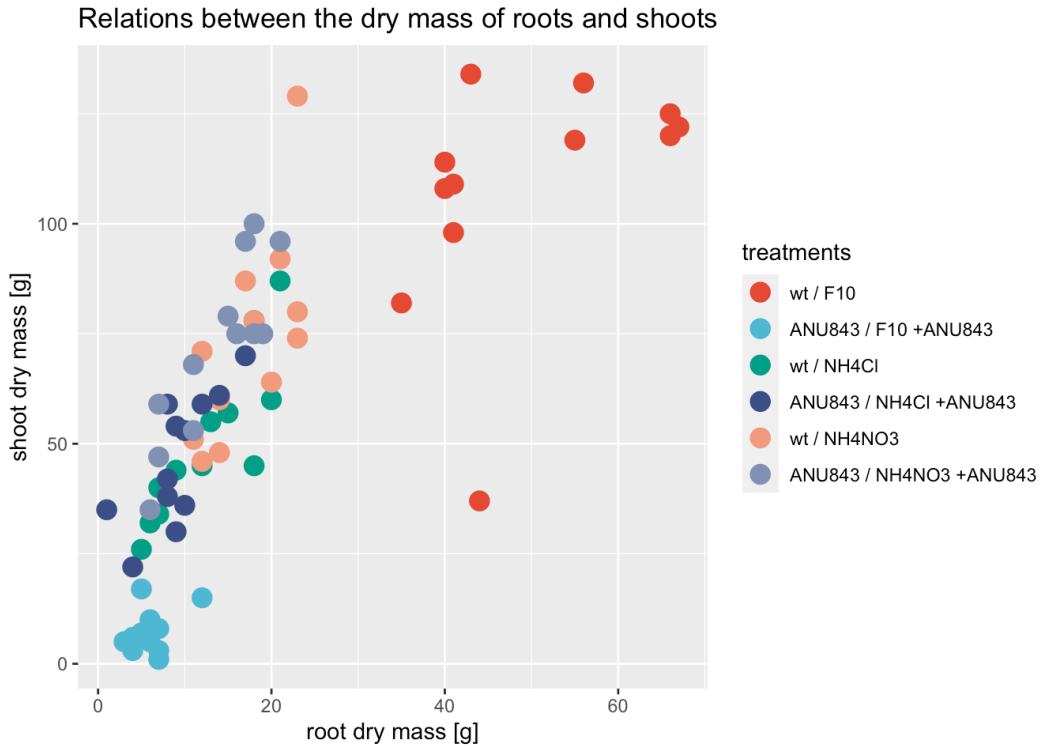


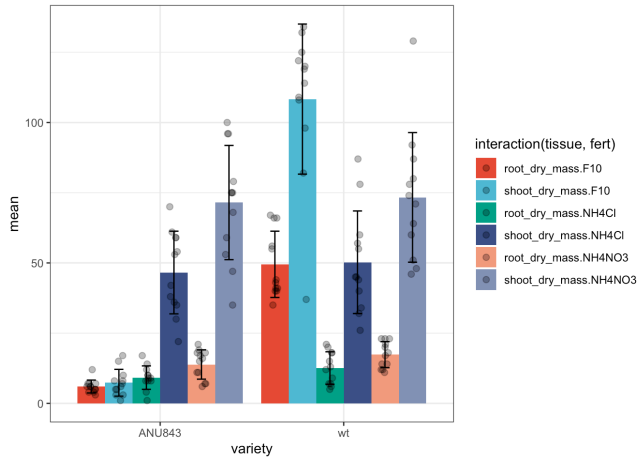

```
xlab <- expression(x[i])
ylab <- expression(x[i]^2)
par(mar = c(4.2, 4.8, 0.5, 0.5))
plot(0, 0, type = "n",
     xlim = c(0,2), ylim = c(0, 10),
     xlab = xlab, ylab = ylab)


text(1, 8, expression(paste(
integral(f(x), 0, infinity), dx == 1 )),
cex = 1.5)

text(1, 4, expression( sum(j, j == 1, n)
== over(n(n+1), 2) ), cex = 1.5)
```

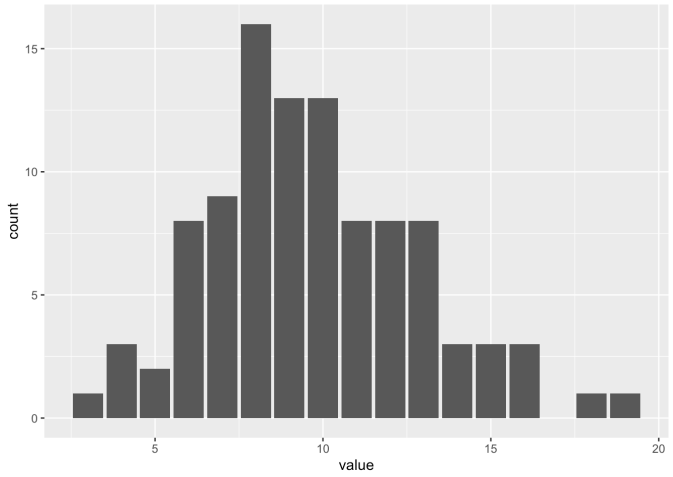
ggplot

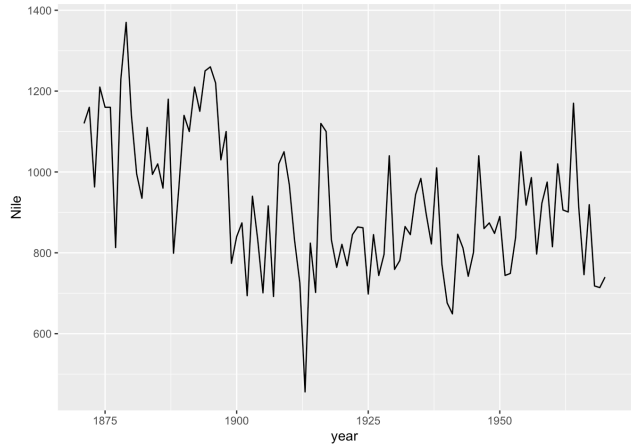

Sepal.Width
Sepal.Length
Petal.Width
Petal.Length
setosa
versicolor
virginica


circumference
age
factor(Tree)
3
1
5
2
4
dist
speed


Relations between the dry mass of roots and shoots
shoot dry mass [g]
root dry mass [g]
treatments
wt / F10
ANU843 / F10 +ANU843
wt / NH4Cl
ANU843 / NH4Cl +ANU843
wt / NH4NO3
ANU843 / NH4NO3 +ANU843


mean
variety
ANU843
wt
interaction(tissue, fert)
root_dry_mass.F10
shoot_dry_mass.F10
root_dry_mass.NH4Cl
shoot_dry_mass.NH4Cl
root_dry_mass.NH4NO3
shoot_dry_mass.NH4NO3


count
value


Nile
year