

医療データ科学実習

Practice of Biomedical Data Science

第4回

前回のSlidoの質問に対する回答

Q1. `breaks = c(0, 20, 40, 60, Inf)` だと、区切り値の default では例えば 20 の所は `20]`, `(20` となるのだと思うのですが、`20)`, `[20` という区切り方にしたい時の指定方法はどうなるのでしょうか？

A1. `cut()` 関数には `right` オプションがあり, これを `FALSE` にすると左閉じ右開きの区間分けが得られます: `cut(x, breaks = c(0, 20, 40, 60, Inf), right = FALSE)`

Q2. `breaks = seq(4, 8, 0.5)` という文法は `dplyr` や `janitor` をインストール・インポートする前の R でも通用しますか？

A2. `cut`関数, `seq`関数ともにbase Rの関数なので通用します。

前回のSlidoの質問に対する回答

Q3. 今行っている処理のどの部分を dplyr が担っていて、どの部分を janitor が担っているのかは、不可分(もしくはあまり意識することに意味はない)のでしょうか？

A3. それぞれのパッケージにどのような関数が含まれるかは公開されているパッケージのマニュアルから確認できます。(<https://cran.r-project.org/web/packages/dplyr/dplyr.pdf>, <https://cran.r-project.org/web/packages/janitor/janitor.pdf>)

Q4. %>%での処理について、あまりにも%>%でコードをつなげていくと処理が重くなるなどのデメリットはあるのでしょうか。

A4. 膨大な量のパイプ演算子を使用することがあまりないためわかりませんが、実際の解析で使用する範囲では変わらない印象です。

Q5. adorn_pct_formatting()のカッコの中にdigits =1を入れる場合と入れない場合があるのはなぜですか？

A5. adorn_pct_formatting関数のdigits引数のデフォルト設定がdigits =1になっているため、省略している場合があります。

Slidoで質疑応答に参加しよう

医療データ科学実習第2回

医療データ科学実習第2回
2025/04/15~2025/04/22
#2974 065

ライブインタラクシ...

Slido を切り替え

ダークモード ☐

Slido について

Slido を無料で試す

Q&A

Polls

質問を入力

人気 最近

1 個の質問

匿名
6 日前

0 0

テスト質問です


Moderator
6 日前

テスト質問了解です

質問を投稿部分

人の質問に投票できる
人気の質問は上位に表示される


参加
URL : <https://app.sli.do/event/c3tMS5GsSGjpX49jYKgP4Q>



- 匿名で質疑応答に参加できるプラットフォーム
- スマホからでも簡単に利用できます
- 講義後1週間解放しておくので自由に質問やコメントを投稿してください

主催者としてログイン -
プレゼンテーション モード
許可可能な使用 - Slido のプライバシー
Cookie の設定
© 2012-2025 Slido - 67.29.3

slido



Chap1. Rの組み込み関数を用いた グラフィックス

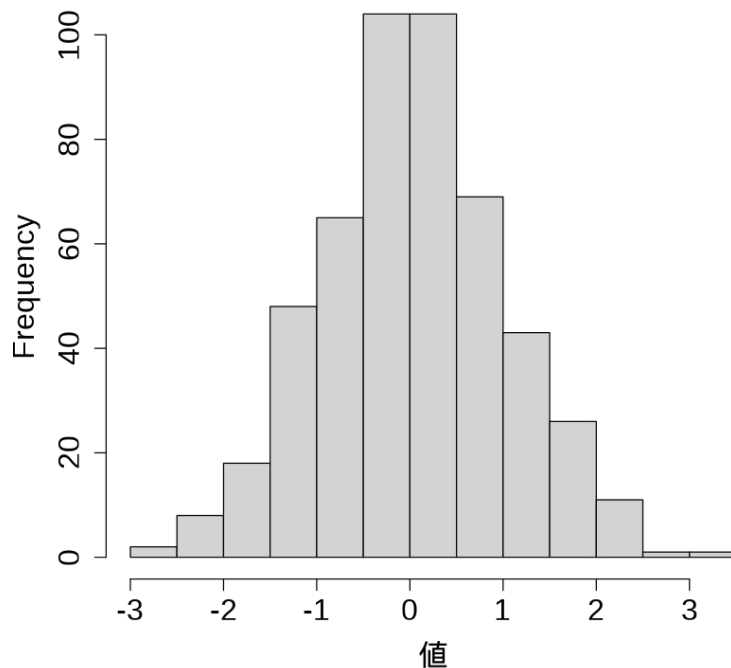
- 1次元の連続変数の経験分布
(相対頻度, 累積頻度, 生存頻度)
- 箱ひげ図
- 複数のプロットの重ね描き

hist によるデータのヒストグラム作成

ヒストグラム (histogram)

- 連続的な数値データの分布(頻度)を視覚的に表現する棒グラフの一種
- データがどの範囲に多く存在するか(分布の形)を把握するためによく使われる
 - 棒の幅=ビンの幅(通常は等間隔)
 - 棒の高さ=ビンに含まれるデータの数

dataのヒストグラム

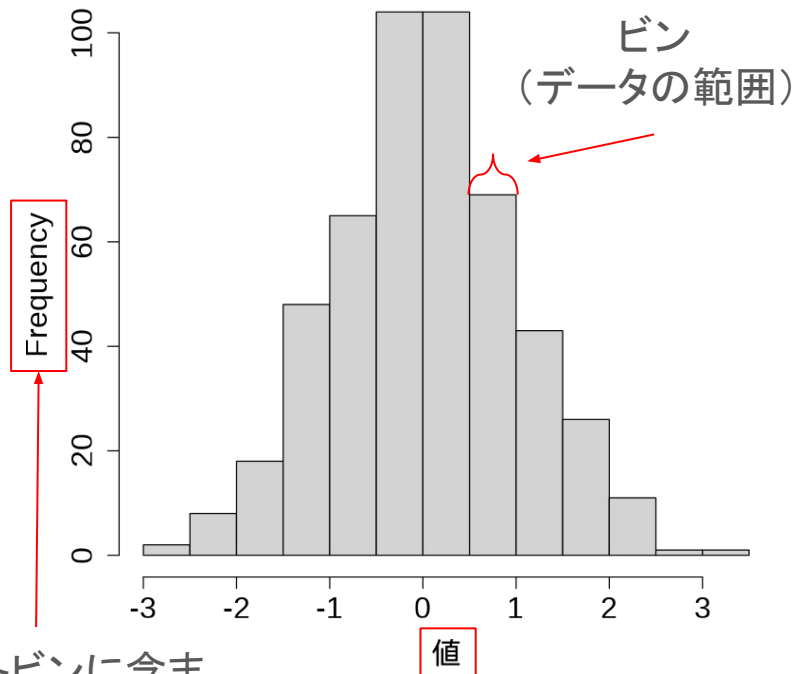


hist によるデータのヒストグラム作成

ヒストグラム(histogram)

- 連続的な数値データの分布(頻度)を視覚的に表現する棒グラフの一種
- データがどの範囲に多く存在するか(分布の形)を把握するためによく使われる
 - 棒の幅=ビンの幅(通常は等間隔)
 - 棒の高さ=ビンに含まれるデータの数

dataのヒストグラム



縦軸は各ビンに含まれるデータの件数

横軸はデータの値

hist によるデータのヒストグラム作成

ヒストグラム描画のサンプルコード

```
set.seed(123)
data <- rnorm(500)
par(cex.main = 3, cex.lab = 2, cex.axis = 2)
hist(data, breaks=10, main='dataのヒストグラム', xlab='値')
```


hist によるデータのヒストグラム作成

ヒストグラム描画のサンプルコード

```
set.seed(123)
```

→ 乱数シードを固定

```
data <- rnorm(500)
```

→ 標準正規分布からの乱数を500個生成

```
par(cex.main = 3, cex.lab = 2, cex.axis = 2)
```

```
hist(data, breaks=10, main='dataのヒストグラム', xlab='値')
```

→ タイトル, 縦軸, 横軸の文字サイズ

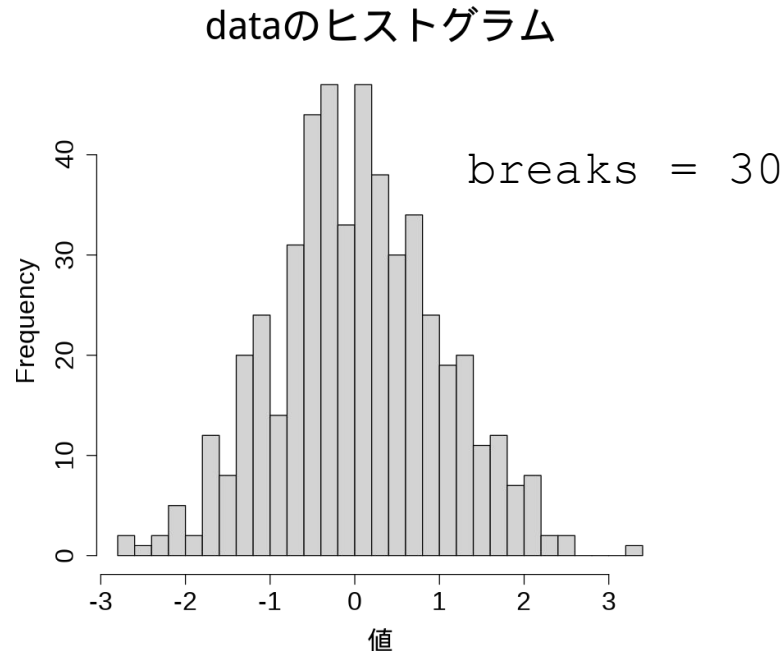
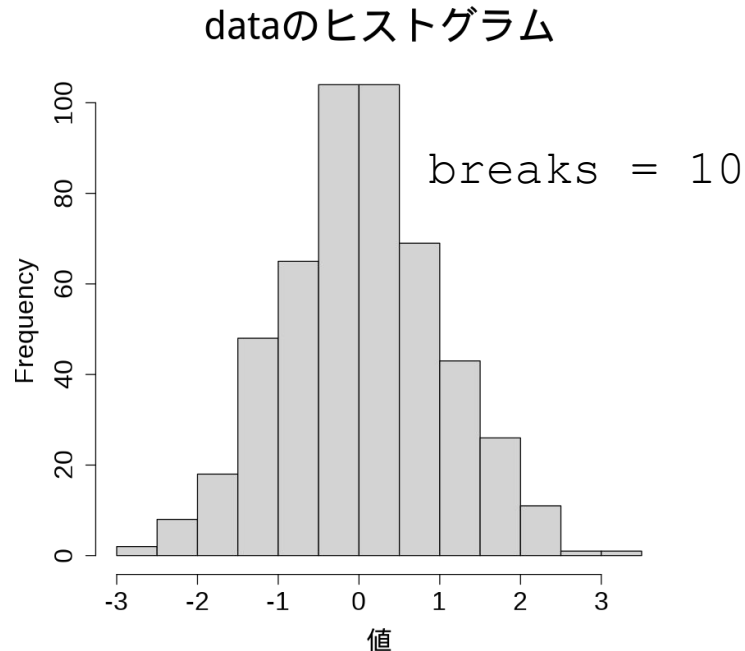
→ ヒストグラムを描画するRのベース関数

- data: ヒストグラムを作りたいデータ集合
- breaks: ビンの数
- main: プロットのタイトル
- xlab: 横軸のラベル

演習: RStudioで上記のコードを実行してヒストグラムを描画してみよう

hist によるデータのヒストグラム作成

ビンの違いによるヒストグラムの違い



binを細かくするほど局所的な形状が正確に表現されるようになる

演習 : ビンの数を増やすとヒストグラムの形状がどう変化していくか見てみよう

hist によるデータのヒストグラム作成

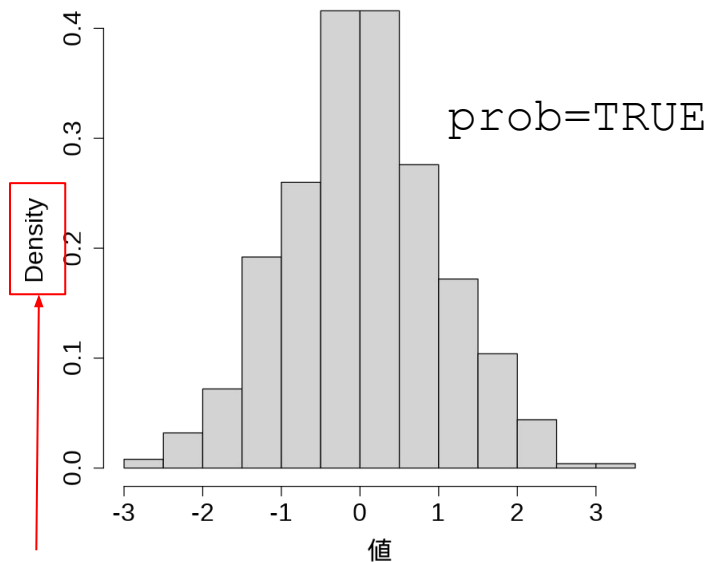
prob オプションによる確率密度の描画

```
hist(data, breaks=10, prob=TRUE, main='dataのヒストグラム', xlab='値')
```

dataのヒストグラム

prob オプションで縦軸を頻度↔確率密度と切り替えられる

- prob=FALSE: 縦軸が頻度に(デフォルト)
- prob=TRUE: 縦軸が確率密度に



縦軸が Density に変わる

演習: prob オプションを切り替えてヒストグラムを描画してみよう

hist によるデータのヒストグラム作成

The Old Faithful Geyser(間欠泉)データ

- Rの組み込みデータセット

```
print(faithful)
```

	eruptions	waiting
1	3.600	79
2	1.800	54
3	3.333	74
4	2.283	62
5	4.533	85
6	2.883	55
7	4.700	88
8	3.600	85
9	1.950	51



1列目: 噴出の継続時間, 2列目: 間欠泉の噴出の間隔(いずれも分)

hist によるデータのヒストグラム作成

ヒストグラム描画のサンプルコード

```
faithful_data <- faithful$waiting  
par(cex.main = 3, cex.lab = 2, cex.axis = 2)  
hist(faithful_data, breaks=20, prob=FALSE, main='Old Faithfulデータのヒストグラム', xlab='待ち時間')
```

hist によるデータのヒストグラム作成

ヒストグラム描画のサンプルコード

```
faithful_data <- faithful$waiting  
par(cex.main = 3, cex.lab = 2, cex.axis = 2)  
hist(faithful_data, breaks=20, prob=FALSE, main='Old Faithfulデータのヒストグラム', xlab='待ち時間')
```

→ faithfulデータのwaiting列(2列目を取り出す)

→ ヒストグラムの各種文字サイズ変更

→ ヒストグラム描画. bin数は20, 縦軸は頻度

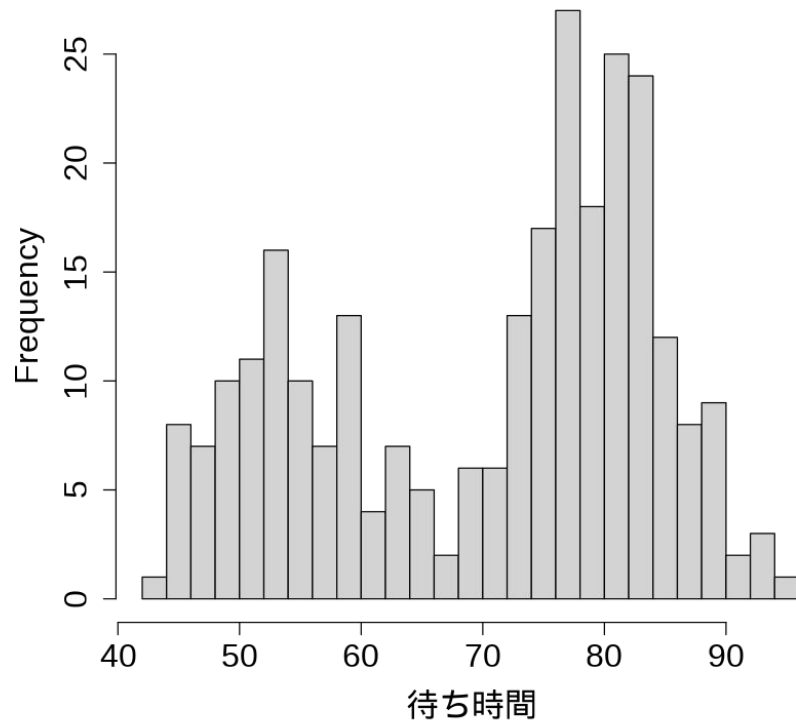
hist によるデータのヒストグラム作成

ヒストグラム描画の結果

40～60分の間と70～90分の間に頻度の山
ができていることが分かる
→ いわゆる二峰性の分布になっている

演習 : faitufunデータのeruptions列(1列
目)に対して同様のヒストグラムを描いて
みよう

Old Faithfulデータのヒストグラム



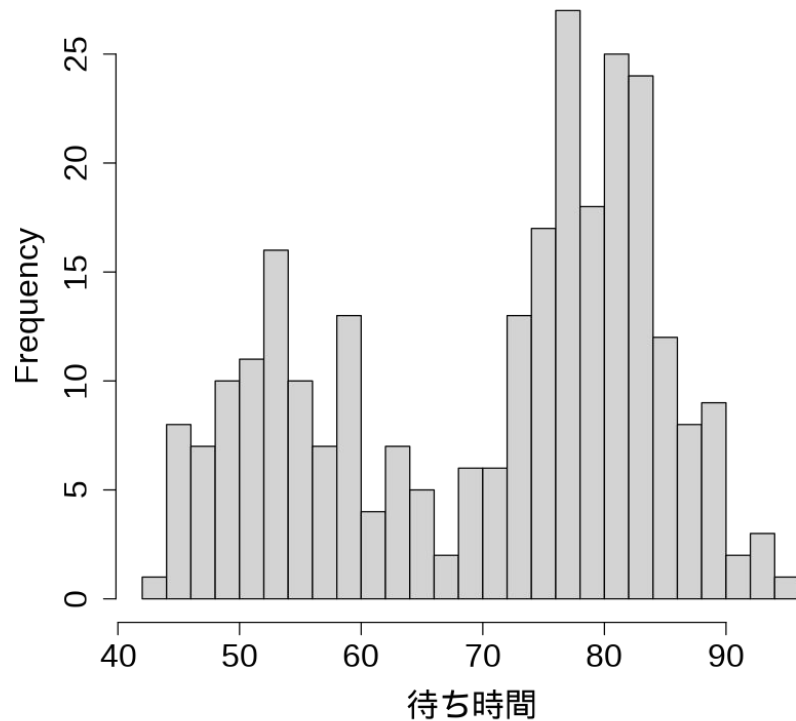
hist によるデータのヒストグラム作成

ヒストグラム描画の結果

40～60分の間と70～90分の間に頻度の山
ができていることが分かる
→ いわゆる二峰性の分布になっている

演習 : faitufunデータのeruptions列(1列
目)に対して同様のヒストグラムを描いて
みよう

Old Faithfulデータのヒストグラム



hist によるデータのヒストグラム作成

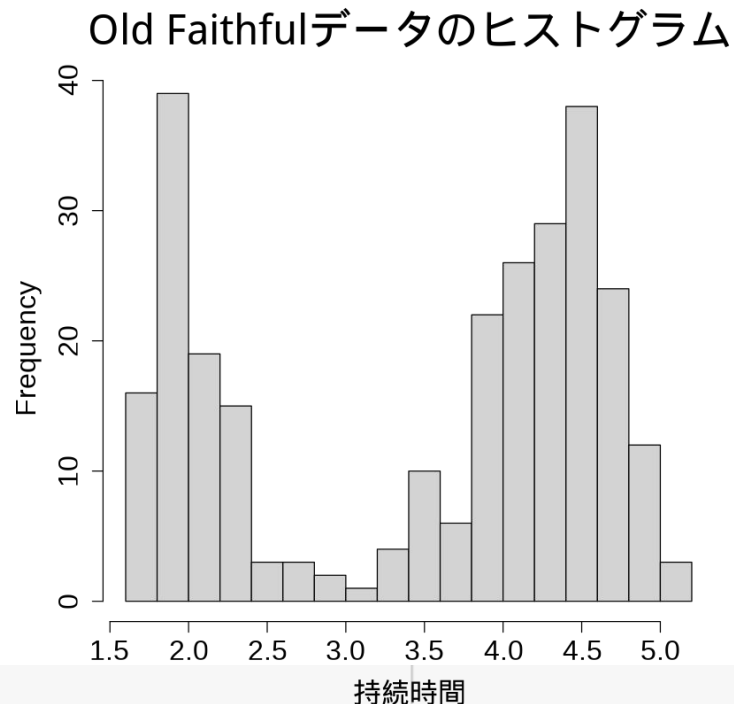
演習: ヒストグラム描画の結果

1.5～2.5分の間と3.5～5.0分の間に頻度の山ができていることが分かる

→ こちらも二峰性の分布になっている

サンプルコード提示:

```
faithful_data <- faithful$eruptions
par(cex.main = 3, cex.lab = 2, cex.axis = 2)
hist(faithful_data, breaks=20, prob=FALSE, main='Old Faithfulデータのヒストグラム', xlab='持続時間')
```



ベース関数によるヒストグラム

演習 : RStudioで実行してみよう

1. パラメータ 1 の指数分布の乱数1000個を生成し相対頻度のヒストグラムを作成

ヒント

- 指数分布に従う乱数の生成は `rexp()` 関数で
- オプション `n=1000` で個数を, `rate = 1` でパラメータを指定

2. `airquality` データセットの `Ozone` 変数について, 欠損値を除いて相対頻度のヒストグラムを作成

ヒント

- 対象のデータは組み込みデータなので `airquality$Ozone` で得られる
- 欠損値を除いた配列は `na.omit(元データ)` で得られる

* サンプルコードは講義後に公開します

☕ ベース関数によるヒストグラム

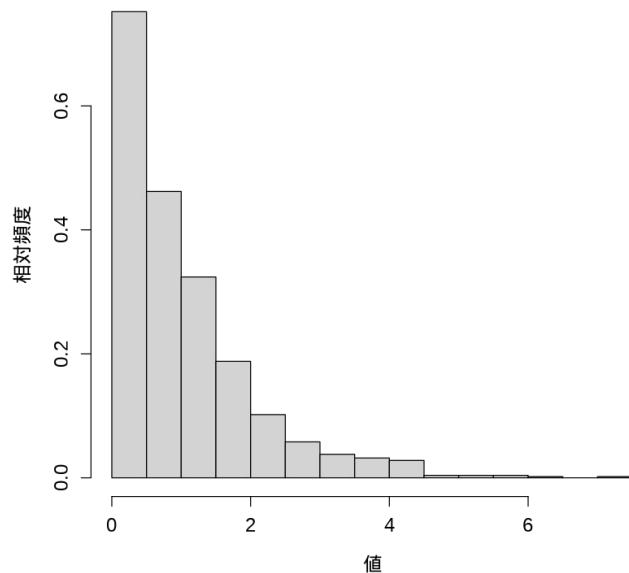
演習 : RStudioで実行してみよう

1. のサンプルコードと実行結果

```
set.seed(123) # 乱数シードを固定
data <- rexp(n = 1000, rate = 1)

hist(data,
      prob = TRUE,
      main = "指数分布の相対頻度ヒストグラム",
      xlab = "値",
      ylab = "相対頻度")
```

指数分布の相対頻度ヒストグラム



*コードの改行に注意

(提示しているサンプルコードはスライドに収めるために改行している)

☕ ベース関数によるヒストグラム

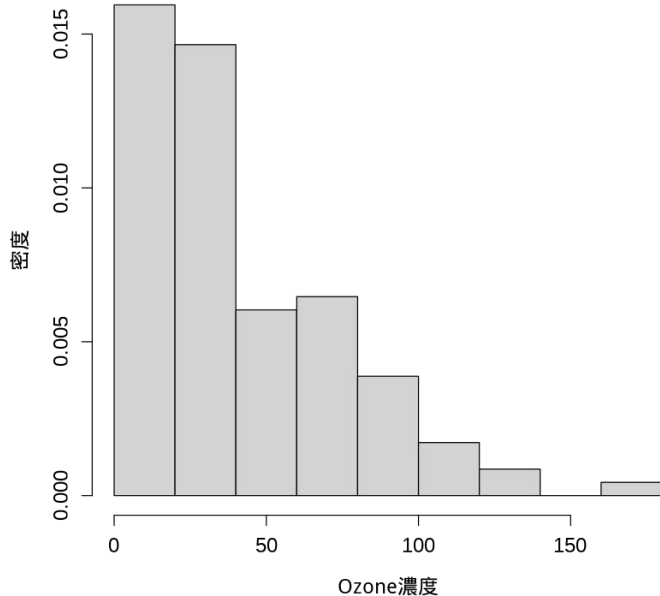
演習 : RStudioで実行してみよう

2. のサンプルコードと実行結果

```
# Ozoneの欠損値を除いたベクトルを作成
ozone_clean <- na.omit(airquality$Ozone)

# 相対頻度ヒストグラムを作成
hist(ozone_clean,
     prob = TRUE,
     main = "Ozoneの相対頻度ヒストグラム",
     xlab = "Ozone濃度",
     ylab = "密度")
```

Ozoneの相対頻度ヒストグラム



*コードの改行に注意

(提示しているサンプルコードはスライドに収めるために改行している)

ecdf によるデータの経験累積分布関数の可視化

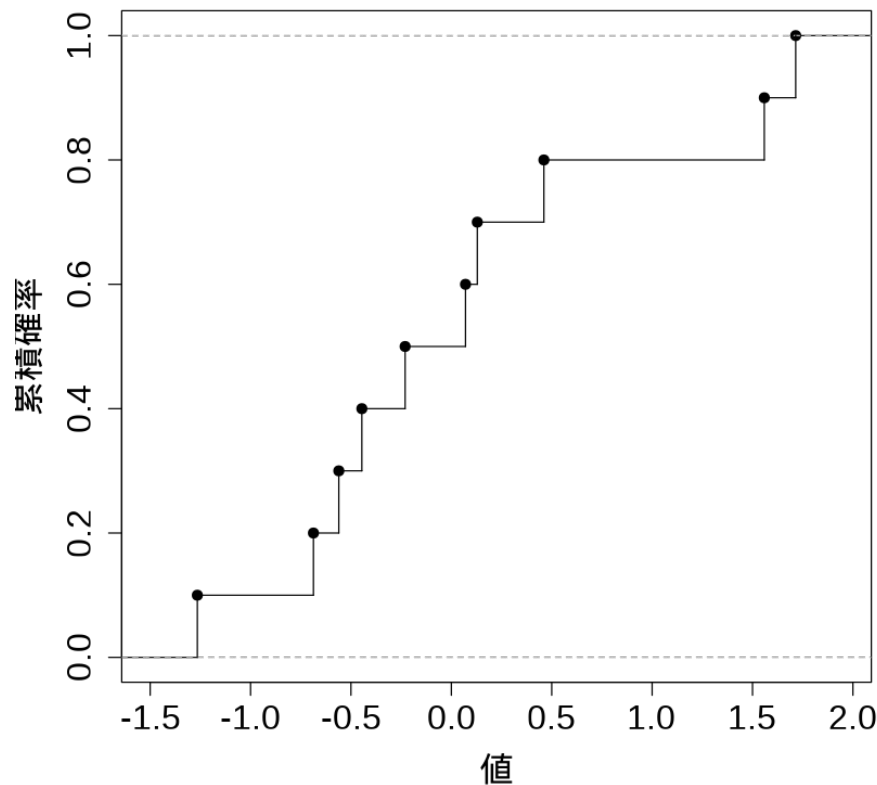
累積分布関数(cumulative distribution function, cdf)

確率変数 X の累積分布関数 $F(X)$ は、ある実現値 x に対して

$$F(x) = \Pr(X \leq x)$$

で定義される

データのECDF



ecdf によるデータの経験累積分布関数の可視化

累積分布関数(cumulative distribution function, cdf)

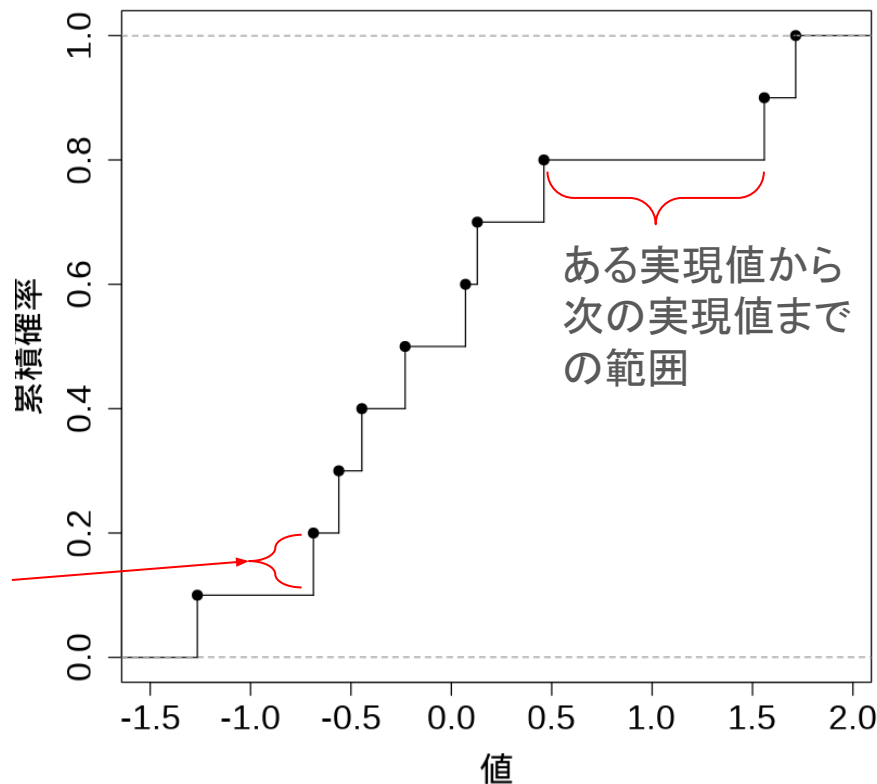
確率変数 X の累積分布関数 $F(X)$ は、ある実現値 x に対して

$$F(x) = \Pr(X \leq x)$$

で定義される

観測値が現れたとき、その点より小さいか等しい値のデータの割合(累積確率)が増える → 結果として、cdfの値(縦軸)が「ジャンプ」する

データのECDF



ecdf によるデータの経験累積分布関数の可視化

cdf描画のサンプルコード

```
data <- rnorm(10)
par(cex.main = 3, cex.lab = 2, cex.axis = 2)
plot(ecdf(data), main='データのECDF', xlab='値', ylab='累積確率')
```

ecdf によるデータの経験累積分布関数の可視化

cdf描画のサンプルコード

```
data <- rnorm(10)
```

→ 標準正規分布から乱数を10個生成

```
par(cex.main = 3, cex.lab = 2, cex.axis = 2)
```

```
plot(ecdf(data), main='データのECDF', xlab='値', ylab='累積確率')
```

→ data の経験累積分布関数 (ecdf) を計算

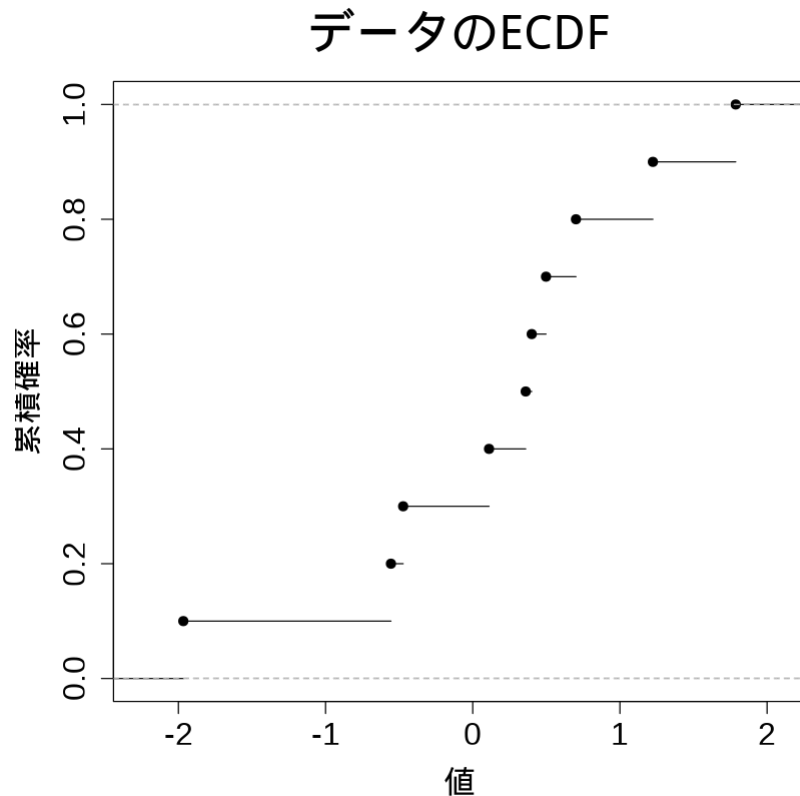
→ 計算された ecdf をプロット

ecdf によるデータの経験累積分布関数の可視化

cdf描画のプロット結果

演習 : RStudioでコードを実行して標準正規分布の ecdf をプロットしてみよう

*デフォルトではジャンプを表す縦の線は入らないことに注意



ecdf によるデータの経験累積分布関数の可視化

ecdfの縦線を自動で入れる方法

```
plot(ecdf(data), main = "データのECDF", xlab = "値", ylab = "累積確率",  
verticals = TRUE,  
do.points = TRUE)
```

*コードの改行に注意(提示しているサンプルコードはスライドに収めるために改行している)

ecdf によるデータの経験累積分布関数の可視化

ecdfの縦線を自動で入れる方法

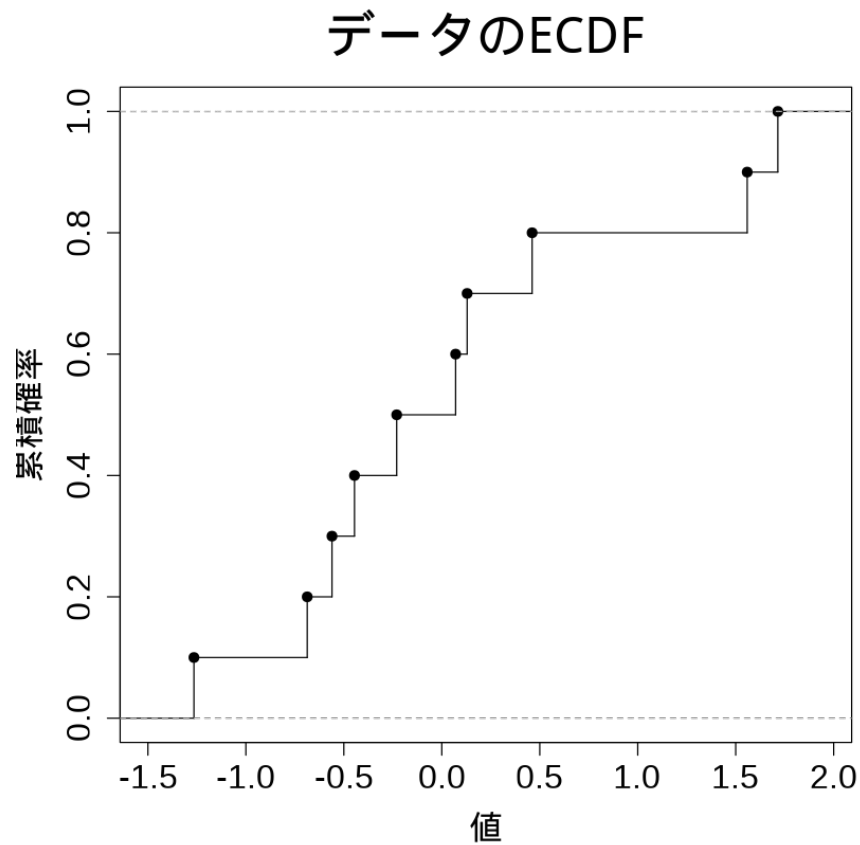
```
plot(ecdf(data), main = "データのECDF", xlab = "値", ylab = "累積確率",  
verticals = TRUE, do.points = TRUE)
```

→ plot 関数の verticals オプションを TRUE にする

→ do.points オプションでデータ点の表示も指定できる

ecdf によるデータの経験累積分布関数の可視化

実行結果





ベース関数による経験累積分布関数

演習 : RStudioで実行してみよう

iris データの Sepal.Length に対してECDFを描き中央値に垂直線を引く

ヒント

- データの中央値の計算は `median()` 関数で
- 中央値に垂線を引く操作は以下で実行できる (`med` が計算した中央値)

```
abline(v = med, col = "red", lwd = 2, lty = 2)
```

* サンプルコードは講義後に公開します



ベース関数による経験累積分布関数

演習: サンプルコードと実行例

```
# Sepal.Length のベクトルを取得
x <- iris$Sepal.Length

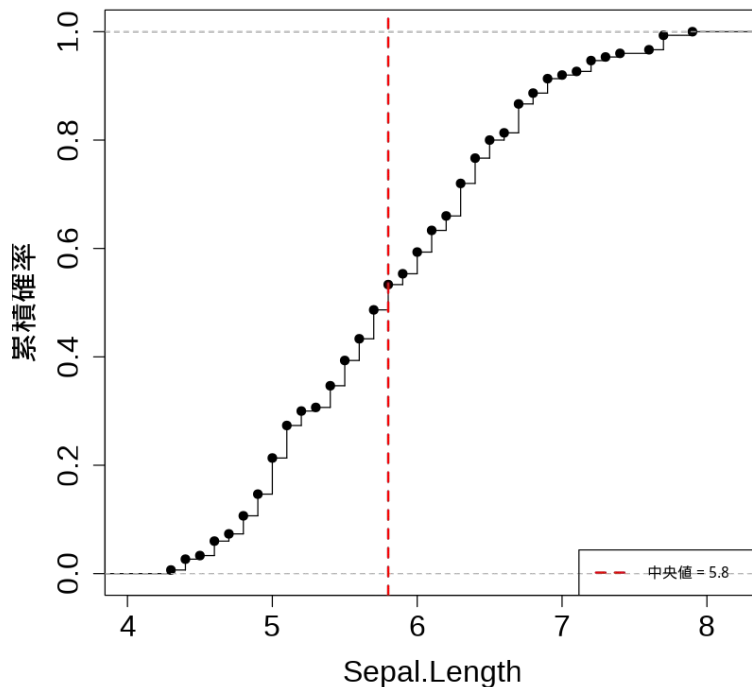
# ECDFを計算
F <- ecdf(x)

# ECDF のプロット
plot(F,
     main = "Sepal.Lengthの経験累積分布関数",
     xlab = "Sepal.Length",
     ylab = "累積確率",
     verticals = TRUE,
     do.points = TRUE)

# 中央値の計算
med <- median(x)

# 中央値に垂直線を追加
abline(v = med, col = "red", lwd = 2, lty = 2)
```

Sepal.Lengthの経験累積分布関数



ecdf によるデータの生存頻度の可視化

生存関数(survival function)

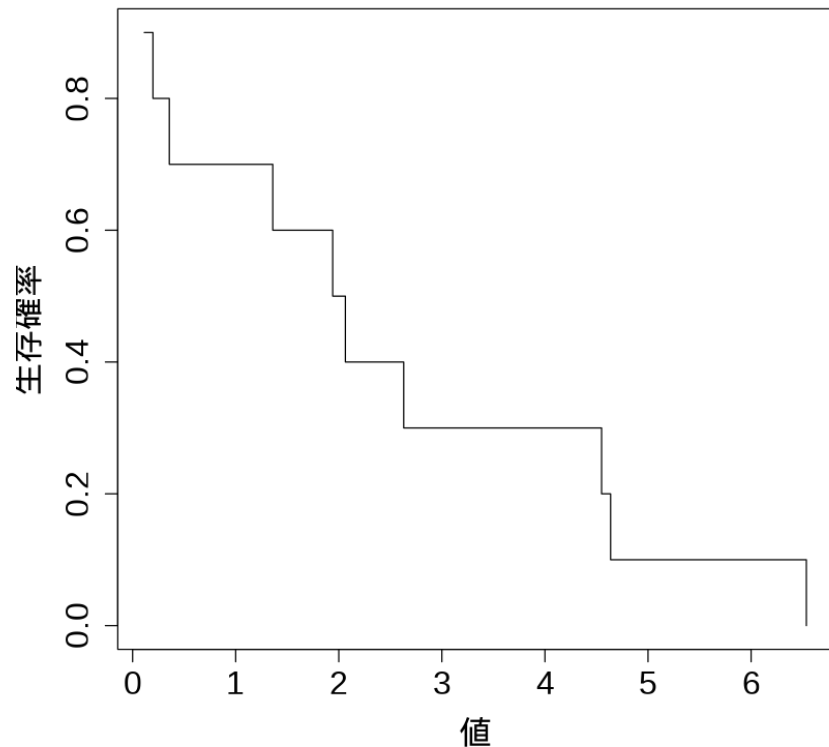
確率変数 X に対する生存関数
とは、ある実現値 x に対して

$$\begin{aligned} S(x) &= \Pr(X > x) \\ &= 1 - F(x) \\ &= 1 - \Pr(X \leq x) \end{aligned}$$

で定義される

→ データが x を超えて
存在する確率

データの生存関数



ecdf によるデータの生存頻度の可視化

生存関数(survival function)

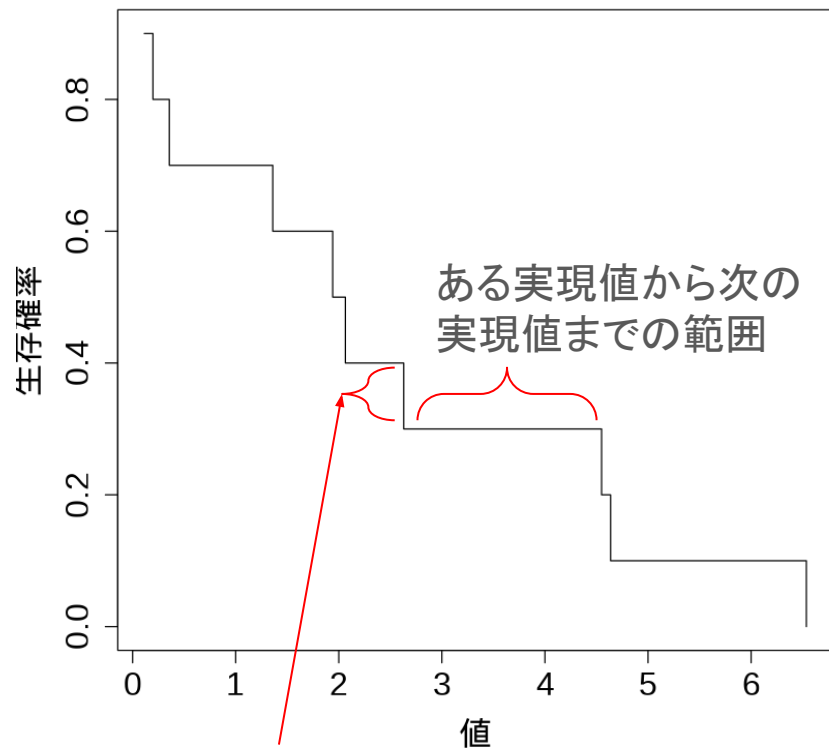
確率変数 X に対する生存関数
とは、ある実現値 x に対して

$$\begin{aligned} S(x) &= \Pr(X > x) \\ &= 1 - F(x) \\ &= 1 - \Pr(X \leq x) \end{aligned}$$

で定義される

→ データが x を超えて
存在する確率

データの生存関数



観測値が減ったとき、その点より大きいか等しい値のデータの割合(生存確率)が減る → cdfと同様のジャンプが起こる

ecdf によるデータの生存頻度の可視化

経験生存関数描画のサンプルコード

```
data <- rchisq(10, df=3)
ecdf_func <- ecdf(data)
x_vals <- sort(unique(data))
survival_vals <- 1 - ecdf_func(x_vals)
par(cex.main = 3, cex.lab = 2, cex.axis = 2)
plot(x_vals, survival_vals, type='s', main='データの生存関数', xlab='値', ylab='生存確率')
```

ecdf によるデータの生存頻度の可視化

経験生存関数描画のサンプルコード

```
data <- rchisq(10, df=3) → 自由度3のカイ二乗分布から乱数を10個生成
ecdf_func <- ecdf(data) → データの経験累積分布関数値を計算
x_vals <- sort(unique(data)) → データを小さい順に並べ替え（横軸の作成）
survival_vals <- 1 - ecdf_func(x_vals)
par(cex.main = 3, cex.lab = 2, cex.axis = 2)
plot(x_vals, survival_vals, type='s', main='データの生存関数', xlab='値', ylab='生存確率')
```

ecdf によるデータの生存頻度の可視化

経験生存関数描画のサンプルコード

```
data <- rchisq(10, df=3)
ecdf_func <- ecdf(data) → データの経験累積分布関数値を計算
x_vals <- sort(unique(data))
survival_vals <- 1 - ecdf_func(x_vals) → ecdf からデータの生存関数値を計算
par(cex.main = 3, cex.lab = 2, cex.axis = 2)
plot(x_vals, survival_vals, type='s', main='データの生存関数', xlab='値', ylab='生存確率')
```

→ プロット線のスタイルを指定

- `type="s"`: steps (階段関数状のプロット)
- これ以外にもいくつかオプションがある

→ プロットの縦軸

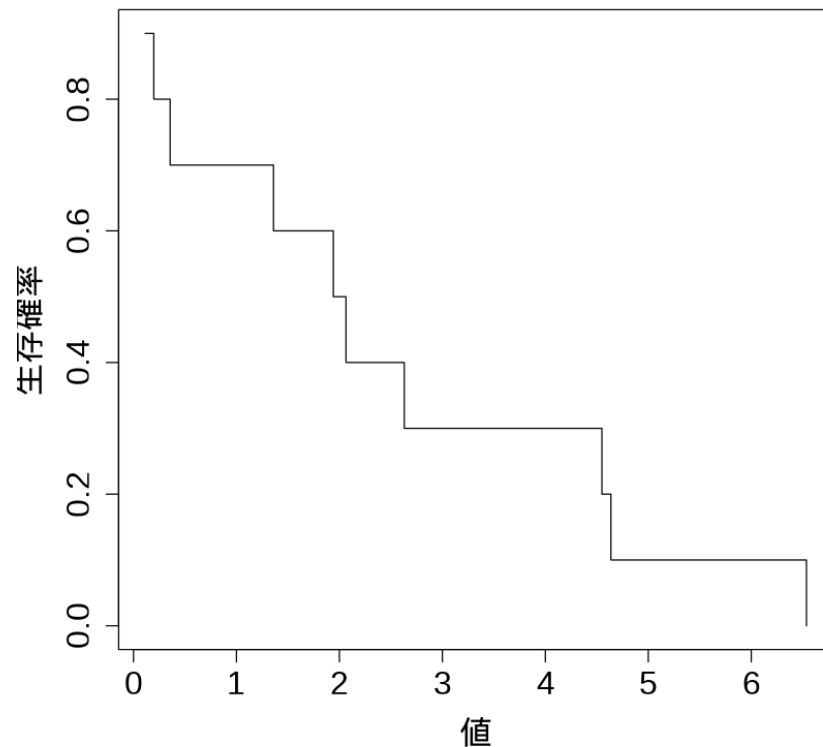
→ プロットの横軸

ecdf によるデータの生存頻度の可視化

経験生存関数描画の実行結果

演習 : RStudioでコードを実行して生存関数をプロットしてみよう

データの生存関数



☕ ベース関数による生存関数

演習 : RStudioで実行してみよう

faithful\$eruptions の生存関数を描画し, 0.2の生存確率に対応する時間を調べる

ヒント

- まずデータの生存関数を描画し, 生存確率0.2の基準線を引いてみよう
- 次に, 生存確率0.2に対応する横軸上の点を見つけよう(次のコード)

```
x_02 <- x_vals[which(survival_vals <= 0.2)[1]]
```

* サンプルコードは講義後に公開します

→ 配列の中で survival_vals が
0.2 以下であるもののうち最初の
要素を取り出す

☕ ベース関数による生存関数

演習 : サンプルコード実行例

```
# 対象データ : 噴火時間
```

```
eruption_times <- faithful$eruptions
```

```
# 経験的累積分布関数
```

```
F <- ecdf(eruption_times)
```

```
# 生存関数 (1 - F(x))
```

```
x_vals <- sort(unique(eruption_times))
```

```
survival_vals <- 1 - F(x_vals)
```

```
# 生存関数の描画
```

```
plot(x_vals, survival_vals, type = "s",  
     main = "Old Faithful の噴火時間に対する生存関数",  
     xlab = "噴火時間 [分]",  
     ylab = "生存確率")
```

```
# 生存確率0.2の基準線
```

```
abline(h = 0.2, col = "red", lty = 2)
```

```
# 対応する x (生存確率が0.2を下回る最初のx) を探す
```

```
x_02 <- x_vals[which(survival_vals <= 0.2)[1]]
```

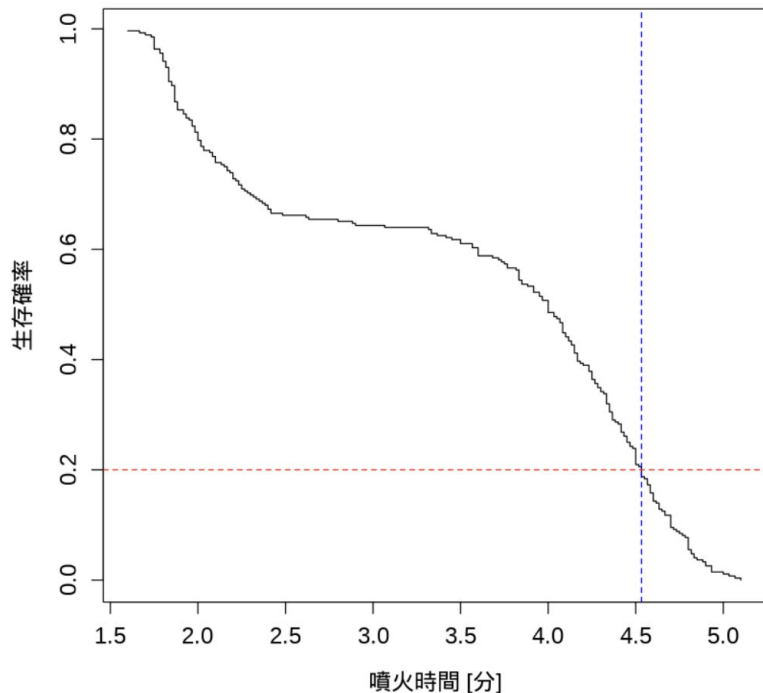
```
# 結果を表示してプロットに追加
```

```
cat("生存確率 0.2 に対応する噴火時間 (最小の x) :", x_02, "分\n")
```

```
abline(v = x_02, col = "blue", lty = 2)
```

生存確率 0.2 に対応する噴火時間 (最小の x) : 4.533 分

Old Faithful の噴火時間に対する生存関数

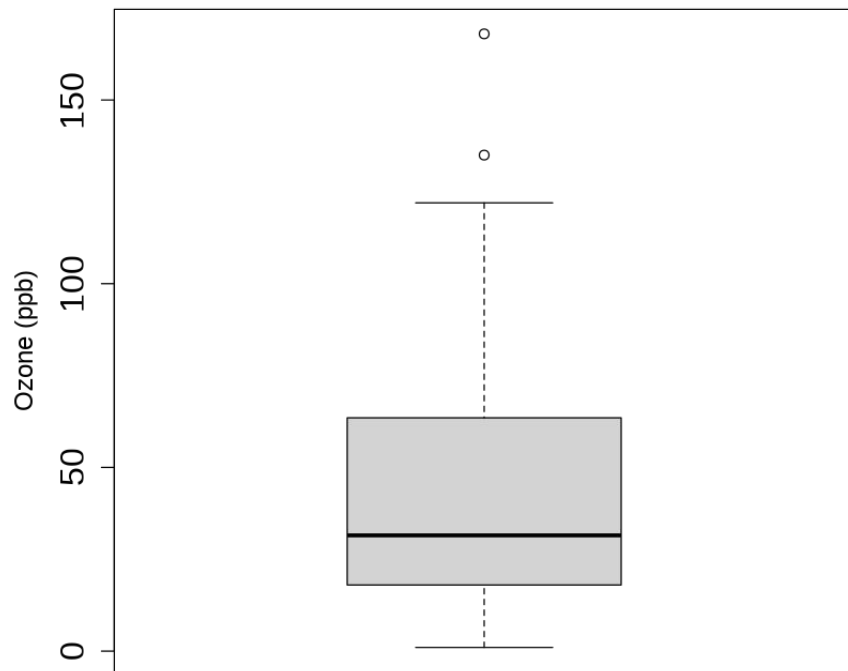


boxplot によるデータの箱ひげ図の描画

箱ひげ図 (box plot)

- データの分布・ばらつき・外れ値を視覚的に要約するためのグラフ
- **5つの要約統計量**をもとに構成され、データの中心・広がり・非対称性・外れ値などを直感的に把握できる

Ozone濃度のボックスプロット

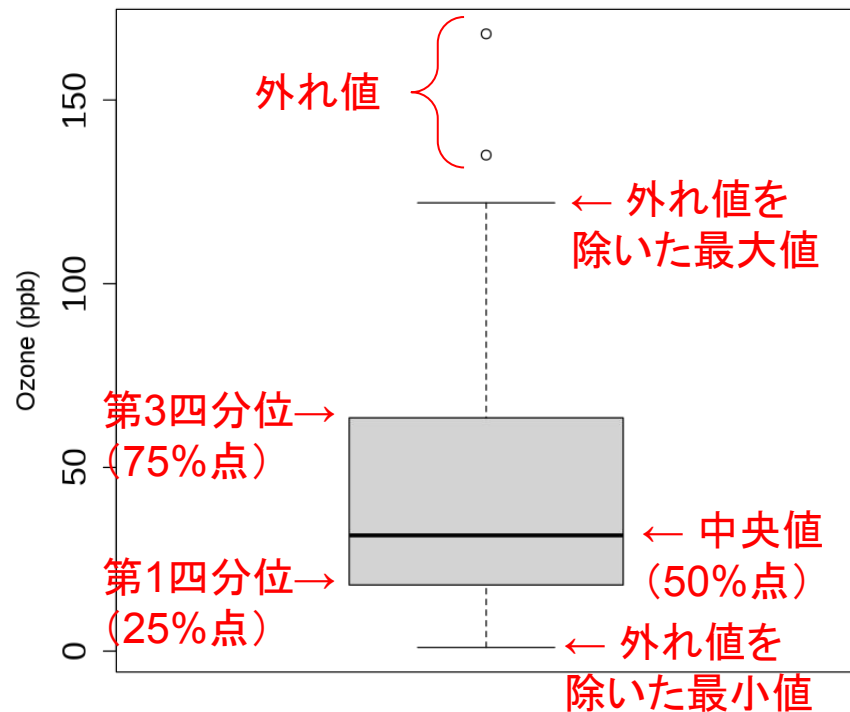


boxplot によるデータの箱ひげ図の描画

箱ひげ図 (box plot)

- データの分布・ばらつき・外れ値を視覚的に要約するためのグラフ
- **5つの要約統計量**をもとに構成され、データの中心・広がり・非対称性・外れ値などを直感的に把握できる

Ozone濃度のボックスプロット



boxplot によるデータの箱ひげ図の描画

箱ひげ図描画のサンプルコード

```
# Ozone列の欠損値を除去
```

```
ozone <- na.omit(airquality$Ozone)
```

```
# ボックスプロットを描画
```

```
par(cex.main = 2, cex.lab = 1.5, cex.axis = 2)
```

```
boxplot(ozone, main = "Ozone濃度のボックスプロット", ylab = "Ozone (ppb)")
```

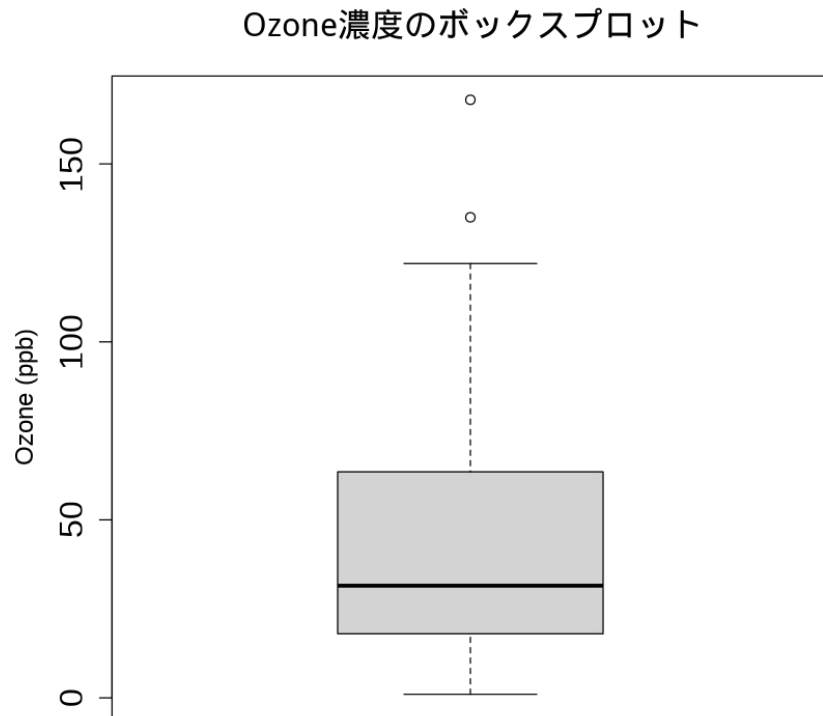
実は、boxplot には na.action オプションがあり、デフォルトでは欠測値は自動的に無視される（つまり最初の欠損値除去はなくても良い）

boxplot によるデータの箱ひげ図の描画

箱ひげ図描画の実行結果

演習 : RStudioでコードを実行して箱ひげ図をプロットしてみよう

演習 : 欠損値の処理を入れた場合と入れない場合でプロットが変わるかどうかを確認しよう



複数のプロットを重ねて表示する

- `plot` 関数を用いる場合の重ね描き
 - `par(new = TRUE)` で次の描画を前の上に重ねる
 - `lines()` や `points()` を使って追加描画
 - `matplot()` を使って複数系列を一括で描画
- `plot` 以外の関数(e.g. `hist`, `boxplot`, ...)
 - 各関数の `add = TRUE` オプションを用いる

`par(new = TRUE)` で次の描画を前の上に重ねる

プロットの重ね合わせのサンプルコード

```
# 基本のプロット（例：sinカーブ）
```

```
x <- seq(0, 2 * pi, length.out = 100)
```

```
par(cex.main = 2, cex.lab = 1.5, cex.axis = 2)
```

```
plot(x, sin(x), type = "l", col = "blue", lwd = 2,  
      ylim = c(-1.5, 1.5), main = "sin と cos の重ね描き")
```

`par(new = TRUE)` で次の描画を前の上に重ねる

プロットの重ね合わせのサンプルコード

基本のプロット (例: sinカーブ)

```
x <- seq(0, 2 * pi, length.out = 100)
```

→ 横軸の生成: $[0, 2\pi]$

```
par(cex.main = 2, cex.lab = 1.5, cex.axis = 2)
```

```
plot(x, sin(x), type = "l", col = "blue", lwd = 2,  
      ylim = c(-1.5, 1.5), main = "sin と cos の重ね描き")
```

横軸データと
縦軸データ

プロット線のスタイル
("l"は線だけ)

線の色

線の太さ

`par(new = TRUE)` で次の描画を前の上に重ねる

プロットの重ね合わせのサンプルコード

```
# 基本のプロット (例: sinカーブ)
```

```
x <- seq(0, 2 * pi, length.out = 100)
```

```
par(cex.main = 2, cex.lab = 1.5, cex.axis = 2)
```

```
plot(x, sin(x), type = "l", col = "blue", lwd = 2,  
      ylim = c(-1.5, 1.5), main = "sin と cos の重ね描き")
```

→ 縦軸のプロット範囲 (-1.5~1.5の範囲でプロットを作成)

`par(new = TRUE)` で次の描画を前の上に重ねる

プロットの重ね合わせのサンプルコード

```
# 新しいプロットを重ねる  
par(new = TRUE)
```

直前のプロットコードと直後のプロットコードの間に
上をはさむ

`par(new = TRUE)` で次の描画を前の上に重ねる

プロットの重ね合わせのサンプルコード

重ねるプロット（例：cosカーブ）

```
plot(x, cos(x), type = "l", col = "red", lwd = 2,  
     axes = FALSE, xlab = "", ylab = "", ylim = c(-1.5, 1.5))
```

→ 二つ目以降のプロットでは、軸ラベルなどはつけない
（つけると文字が重なって表示されてしまう）

`par(new = TRUE)` で次の描画を前の上に重ねる

プロットの重ね合わせのサンプルコード(全体)

```
# 基本のプロット (例: sinカーブ)
```

```
x <- seq(0, 2 * pi, length.out = 100)
```

```
par(cex.main = 2, cex.lab = 1.5, cex.axis = 2)
```

```
plot(x, sin(x), type = "l", col = "blue", lwd = 2,  
      ylim = c(-1.5, 1.5), main = "sin と cos の重ね描き")
```

```
# 新しいプロットを重ねる
```

```
par(new = TRUE)
```

```
# 重ねるプロット (例: cosカーブ)
```

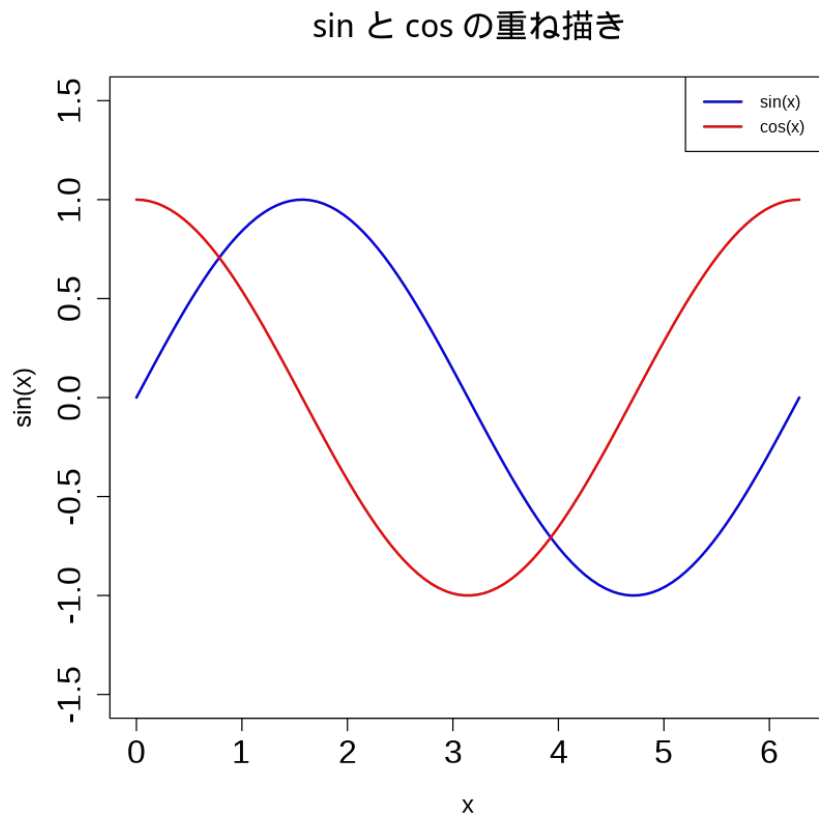
```
plot(x, cos(x), type = "l", col = "red", lwd = 2,  
      axes = FALSE, xlab = "", ylab = "", ylim = c(-1.5, 1.5))
```

`par(new = TRUE)` で次の描画を前の上に重ねる

プロットの重ね合わせの実行結果

演習 : RStudioでコードを実行してプロットを重ねて表示してみよう

*サインカーブのプロットとコサインカーブのプロットを順番に実行すると「プロットが重なる」感じが分かる



hist 関数で add = TRUE オプションを用いる

サンプルコード

```
# サンプルコード1: 2つの正規分布の密度を重ね描き
set.seed(1)
x1 <- rnorm(30, mean=0)
x2 <- rnorm(30, mean=2)
par(cex.main = 3, cex.lab = 2, cex.axis = 2)
hist(x1, breaks=10, prob=TRUE, xlim=c(-4,6), col=rgb(1,0,0,0.3), main='2つの正規分布', xlab='値')
hist(x2, breaks=10, prob=TRUE, add=TRUE, col=rgb(0,0,1,0.3))
```

hist 関数で add = TRUE オプションを用いる

サンプルコード

サンプルコード1: 2つの正規分布の密度を重ね描き

```
set.seed(1)
```

```
x1 <- rnorm(30, mean=0)
```

```
x2 <- rnorm(30, mean=2)
```

→ 2種類の正規分布から2種類のデータセットを生成

```
par(cex.main = 3, cex.lab = 2, cex.axis = 2)
```

```
hist(x1, breaks=10, prob=TRUE, xlim=c(-4,6), col=rgb(1,0,0,0.3), main='2つの正規分布', xlab='値')
```

```
hist(x2, breaks=10, prob=TRUE, add=TRUE, col=rgb(0,0,1,0.3))
```

1つ目のヒストグラム

2つ目のヒストグラム

2つ目のヒストグラム中で add=TRUE
オプションを使って重ね描きする

hist 関数で add = TRUE オプションを用いる

実行結果

演習 : RStudioでコードを実行してプロットを重ねて表示してみよう

