Intervalo de Confiança para o Tempo de Execução

Rafael Tenfen

Data de entrega: 07/05/2021

Descrição da atividade

O objetivo desta atividade é aplicar o conceito de intervalo de confiança à medição do tempo de execução de uma função.

Algumas recomendações:

- Se você não estiver habituado com R Markdown, acostume-se a processar com frequência o documento, usando o botão **Knit**. Isso permitirá que eventuais erros no documento ou no código R sejam identificados rapidamente, pouco depois de terem sido cometidos, o que facilitará sua correção. Na verdade, é uma boa ideia você fazer isso **agora**, para garantir que seu ambiente esteja configurado corretamente. Se você receber uma mensagem de erro do tipo *Error in library(foo)*, isso significa que o pacote `foo` não está instalado. Para instalar um pacote, execute o comando `install.packages("foo")` no Console, ou clique em *Tools -> Install Packages*.
- Após concluir a atividade, você deverá submeter no Moodle um arquivo ZIP contendo:
  - o arquivo fonte .Rmd;
  - a saída processada (PDF ou HTML) do arquivo .Rmd;
  - o(s) arquivo(s) de dados necessário(s) para o processamento do .Rmd, caso você opte por salvar os dados analisados separadamente.

Configuração

Nesta atividade, a única configuração necessária consiste em carregar o arquivo `rand171.R`, que contém a função `rand171()`, cujo tempo de execução será medido.

```
source("rand171.R")         # carrega arquivo que contem a funcao
```

Salvando seus dados

Este é um experimento iterativo com dados que não se repetem: você irá realizar uma primeira rodada de execuções, analisar os dados dessa rodada para determinar quantas execuções adicionais serão necessárias, e analisar os dados da segunda rodada para saber se serão necessárias mais execuções. Nesse processo, você irá trabalhando com os dados à medida em que eles são obtidos. Ao final do processo, você precisa:

- salvar seus dados em um arquivo;
- escrever sua análise dos dados com base nos dados salvos.

O salvamento dos dados é importante porque eles estarão disponíveis no seu ambiente, mas não para quem receber o seu arquivo .Rmd. Assim, os dados precisam ser fornecidos de alguma forma para que seja possível reprocessar o arquivo .Rmd e reproduzir a análise usando os dados originais.

Você pode salvar seus dados de duas formas. A primeira é preservá-los no próprio arquivo .Rmd, e a segunda é usando um arquivo de dados auxiliar.

Salvando os dados no próprio arquivo .Rmd

Supondo que seus tempos de execução estejam armazenados em um vetor `texec`, você pode usar a função `dput()` para mostrar o conteúdo do vetor de modo que possa ser atribuído a uma variável. O código abaixo demonstra isso com um vetor `x`:

```
x <- 2^c(1:8)
dput(x)
```

```
## c(2, 4, 8, 16, 32, 64, 128, 256)
```

Veja que `dput()` enumera os elementos do vetor em um formato que permite a atribuição a uma variável: os valores de `x` poderiam ser preservados colocando `x <- c(2, 4, 8, 16, 32, 64, 128, 256)` em um bloco R no arquivo .Rmd. Você pode usar a mesma estratégia para preservar os valores de `texec`.

Salvando os dados em um arquivo auxiliar

Supondo novamente que seus tempos de execução estejam armazenados em um vetor `texec`, o código abaixo mostra como salvá-los em um arquivo `texec-rand171.dat`.

```
# texec contém os tempos de execução medidos no experimento
df <- data.frame(texec)
write.table(df, "texec-rand171.dat", row.names = FALSE, quote = FALSE)
```

O código abaixo pode ser usado para recuperar `texec` a partir do arquivo salvo pelo código anterior:

```
texec <- read.table("texec-rand171.dat", header = TRUE)$texec
```

Caso você opte por salvar os dados em um arquivo separado, o comando que lê os dados do arquivo deve ser incluído no seu bloco R de análise.

Definição do experimento

A função `rand171()` encontra `noc` ocorrências do número 171 em uma sequência de números aleatórios, e imprime e retorna o seu tempo de execução (em segundos).

```
#qnorm()

#rand171(33)
#rand171(33)
#rand171(33)

rand171(5)
```

```
## tempo de execucao=0.782 s
```

```
## [1] 0.781935
```

1. Determine o número n de execuções necessárias para obter, com 95% de confiança, o tempo médio de execução com uma margem de erro de $\pm 6\%$. Considere uma amostra inicial de 5 execuções. Se necessário, ajuste o parâmetro `noc` para que o tempo de execução fique na ordem de 3 a 5 s (o valor

default é 20, que deve ser adequado para o RStudio Cloud). Para obter n, use a equação 4.21 (pág. 53) do livro; adote $e = 0.06$, e lembre-se que o coeficiente z deve ser obtido da distribuição t (pois a amostra inicial tem 5 elementos).

2. Realize o número de execuções determinadas no passo anterior, e calcule o intervalo de confiança de 95% para a média. Observe que você deve considerar as 5 execuções iniciais: por exemplo, se no passo 1 você descobriu que são necessárias 25 execuções, execute a função mais 20 vezes, e calcule o IC usando todas as 25 execuções.

3. O IC obtido no item anterior fica dentro da margem de erro desejada de $\pm 6\%$? Caso contrário, amplie a amostra até obter a margem pretendida, e informe o novo IC calculado e quantas execuções foram necessárias ao final. A margem de erro é dada por

$$e = \frac{zs}{\bar{x}\sqrt{n}}$$

onde o coeficiente z deve ser obtido da distribuição t (se $n < 30$) ou da distribuição normal (se $n \geq 30$). Para obter uma porcentagem, multiplique e por 100.

4. Compare os histogramas dos tempos de execução obtidos nos passos 1, 2 e 3 (se houver).

Respostas

Como explicado acima, você deve salvar os dados obtidos durante a realização do experimento e depois escrever sua análise com base nos dados salvos. *Trabalhe com os tempos já coletados, não é necessário realizar novas execuções de* `rand171()`.

A forma mais fácil de fazer isso é usando um bloco separado para cada uma das questões 1 a 4. Você pode adaptar a estrutura abaixo para isso.

```
# código para recuperar os tempos de execução mensurados aqui

randData <- c()
noc3to5s <- 33
initialExec <- 5

# generate 5 initial executions
for (i in 1:initialExec) {
  randData <- c(randData, rand171(noc3to5s))
}
```

```
## tempo de execucao=3.794 s
## tempo de execucao=3.180 s
## tempo de execucao=3.444 s
## tempo de execucao=4.128 s
## tempo de execucao=3.731 s
```

```
df5 <- data.frame(randData)
write.table(df5, "randData5-rand171.dat", row.names = FALSE, quote = FALSE)
dput(randData)
```

```
## c(3.7939670085907, 3.17998504638672, 3.44360852241516, 4.12793350219727,
## 3.73074555397034)
```

Questão 1

```
# código para analisar os dados da questão 1 aqui

xbar <- mean(randData)
s <- sd(randData)
e <- 0.06
ic <- 0.95
alfa <- 1 - ic
p <- 1 - alfa/2


z <- qnorm(p)
if (initialExec < 30) {
  z <- qt(p, initialExec - 1)
}

nest <- (z*s/(e*xbar))^2
nestCeil <- ceiling(nest)
nestCeil
```

```
## [1] 21
```

Resposta para a questão 1: São necessários 21 amostras

Questão 2

```
# código para analisar os dados da questão 2 aqui
randDataN <- randData

if (nestCeil < initialExec) { # to avoid negative numbers on for
  # enter here and it would be 1:0, generating + 1 to randDataN
  nestCeil <- initialExec
}

for (i in 1:(nestCeil - initialExec)) {
  randDataN <- c(randDataN, rand171(noc3to5s))
}
```

```
## tempo de execucao=2.887 s
## tempo de execucao=4.838 s
## tempo de execucao=3.485 s
## tempo de execucao=4.135 s
## tempo de execucao=4.123 s
## tempo de execucao=4.138 s
## tempo de execucao=3.117 s
## tempo de execucao=5.850 s
## tempo de execucao=2.912 s
## tempo de execucao=4.126 s
## tempo de execucao=4.045 s
## tempo de execucao=3.548 s
## tempo de execucao=3.426 s
## tempo de execucao=4.626 s
## tempo de execucao=2.788 s
## tempo de execucao=4.081 s
```

```
dput(randDataN)
```

```
## c(3.7939670085907, 3.17998504638672, 3.44360852241516, 4.12793350219727,
## 3.73074555397034, 2.88685870170593, 4.83771896362305, 3.48523330688477,
## 4.13506436347961, 4.12303185462952, 4.13787031173706, 3.1173050403595,
## 5.85017323493958, 2.91249775886536, 4.12634587287903, 4.04493975639343,
## 3.54810905456543, 3.42579507827759, 4.62636780738831, 2.78827404975891,
## 4.08114314079285)
```

```
dfN <- data.frame(randDataN)
write.table(dfN, "randDataN-rand171.dat", row.names = FALSE, quote = FALSE)
```

```
randDataN.ic <- t.test(randDataN, conf.level=0.95)
#randDataN.ic

#str(randDataN.ic)
#randDataN.ic$conf.int
```

Resposta para a questão 2: IC 3.4979603, 4.1594652

```
# código para analisar os dados da questão 3 aqui
randDataErrorMargin <- randDataN
xbar <- mean(randDataErrorMargin)
s <- sd(randDataErrorMargin)

ic <- 0.95
alfa <- 1 - ic
p <- 1 - alfa/2

z <- qnorm(p)
if (length(randDataErrorMargin) < 30) {
  z <- qt(p, length(randDataErrorMargin) - 1)
}

e = ((z * s) / (xbar * sqrt(length(randDataErrorMargin)))) * 100

e
```

```
## [1] 8.638737
```

```
while (e > 6) {
  randDataErrorMargin <- c(randDataErrorMargin, rand171(noc3to5s))

  xbar <- mean(randDataErrorMargin)
  s <- sd(randDataErrorMargin)
  z <- qnorm(p)
  if (length(randDataErrorMargin) < 30) {
    z <- qt(p, length(randDataErrorMargin) - 1)
  }
  e = ((z * s) / (xbar * sqrt(length(randDataErrorMargin)))) * 100
  e
}
```

```
## tempo de execucao=4.498 s
## tempo de execucao=3.726 s
## tempo de execucao=4.234 s
## tempo de execucao=3.488 s
## tempo de execucao=4.813 s
## tempo de execucao=3.849 s
## tempo de execucao=4.319 s
## tempo de execucao=4.484 s
## tempo de execucao=2.903 s
## tempo de execucao=2.665 s
## tempo de execucao=2.541 s
## tempo de execucao=3.415 s
## tempo de execucao=2.897 s
## tempo de execucao=3.518 s
## tempo de execucao=3.417 s
## tempo de execucao=3.307 s
## tempo de execucao=3.747 s
```

```
dput(randDataErrorMargin)
```

```
## c(3.7939670085907, 3.17998504638672, 3.44360852241516, 4.12793350219727,
## 3.73074555397034, 2.88685870170593, 4.83771896362305, 3.48523330688477,
## 4.13506436347961, 4.12303185462952, 4.13787031173706, 3.1173050403595,
## 5.85017323493958, 2.91249775886536, 4.12634587287903, 4.04493975639343,
## 3.54810905456543, 3.42579507827759, 4.62636780738831, 2.78827404975891,
## 4.08114314079285, 4.49752879142761, 3.72599411010742, 4.23373007774353,
## 3.48757672309875, 4.81343960762024, 3.84853553771973, 4.31925082206726,
## 4.4844024181366, 2.90342998504639, 2.66492319107056, 2.54065275192261,
## 3.41519689559937, 2.89678478240967, 3.51816201210022, 3.41658353805542,
## 3.30744075775146, 3.74662017822266)
```

```
dfN <- data.frame(randDataErrorMargin)
write.table(dfN, "randDataErrorMargin-rand171.dat", row.names = FALSE, quote = FALSE)
```

```
randDataErrorMargin.ic <- t.test(randDataErrorMargin, conf.level=0.95)
randDataErrorMargin.ic
```

```
##
##  One Sample t-test
##
## data:  randDataErrorMargin
## t = 32.977, df = 37, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  3.512751 3.972682
## sample estimates:
## mean of x
##  3.742716
```

Resposta para a questão 3: Foram necessário 38 amostras para ficar dentro da margem de erro desejada de 6% e o novo IC é 3.512751, 3.9726816

```
# código para buscar os dados da questão 4 aqui

q1 <- read.table("randData5-rand171.dat", header = TRUE)$randData
q1
```

```
## [1] 3.793967 3.179985 3.443609 4.127934 3.730746
```

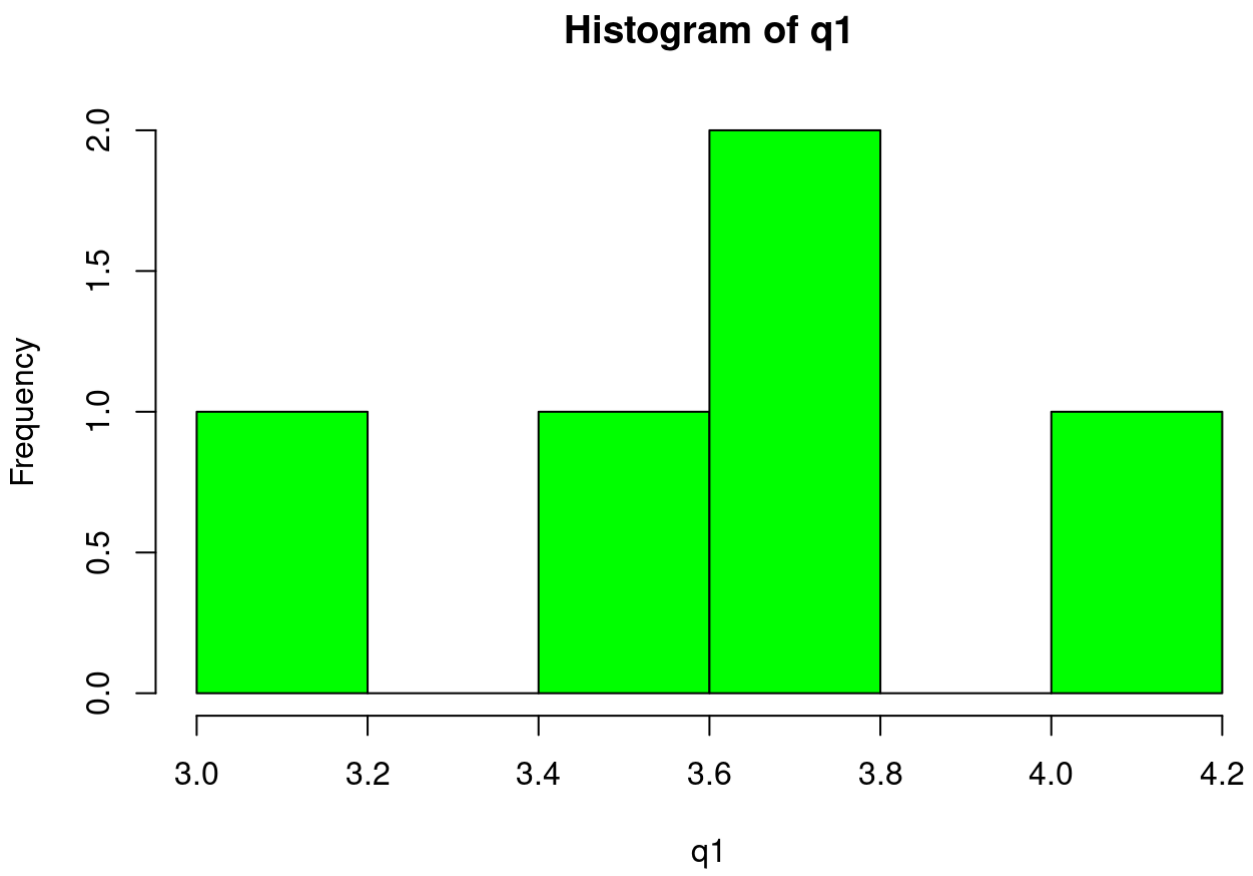
```
q2 <- read.table("randDataN-rand171.dat", header = TRUE)$randDataN
q2
```

```
##  [1] 3.793967 3.179985 3.443609 4.127934 3.730746 2.886859 4.837719 3.485233
##  [9] 4.135064 4.123032 4.137870 3.117305 5.850173 2.912498 4.126346 4.044940
## [17] 3.548109 3.425795 4.626368 2.788274 4.081143
```

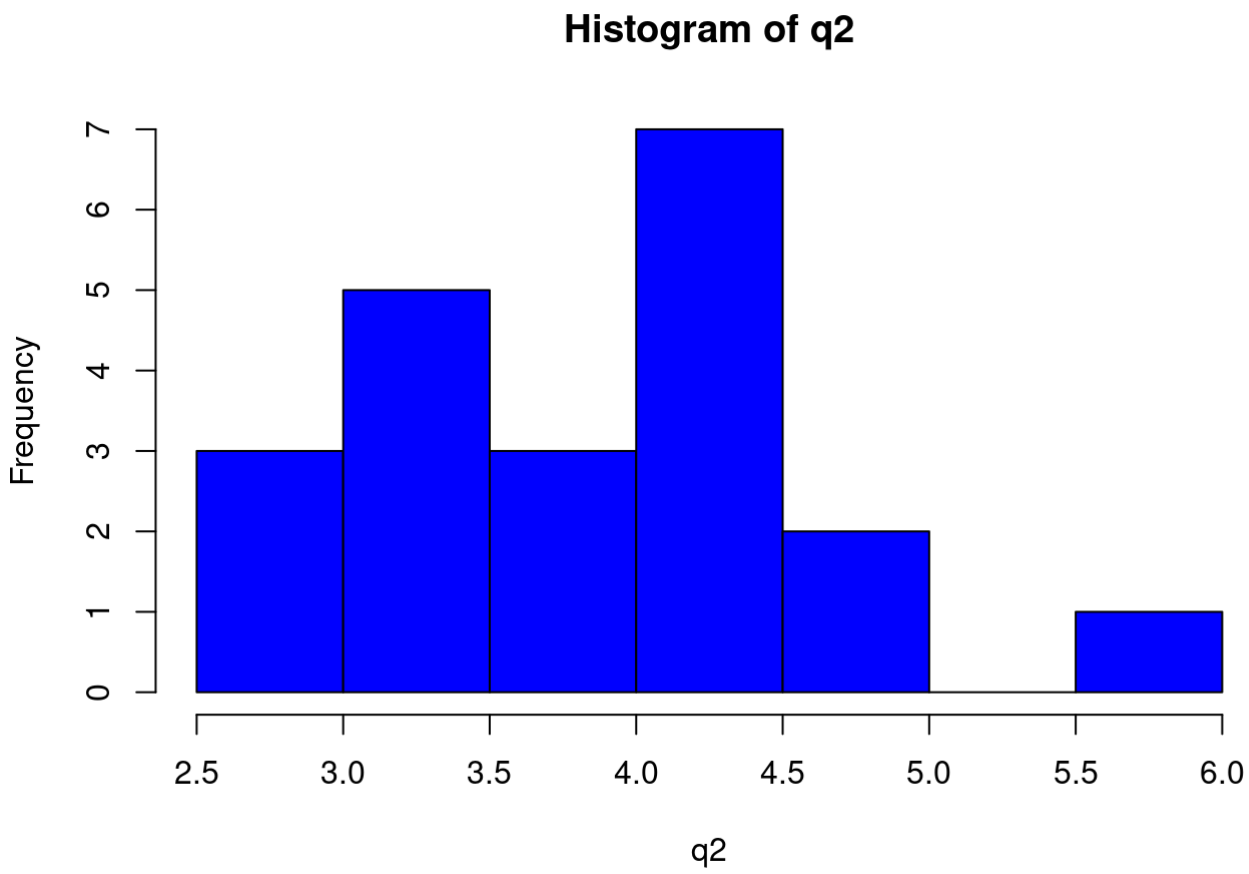
```
q3 <- read.table("randDataErrorMargin-rand171.dat", header = TRUE)$randDataErrorMargi
n
q3
```

```
##  [1] 3.793967 3.179985 3.443609 4.127934 3.730746 2.886859 4.837719 3.485233
##  [9] 4.135064 4.123032 4.137870 3.117305 5.850173 2.912498 4.126346 4.044940
## [17] 3.548109 3.425795 4.626368 2.788274 4.081143 4.497529 3.725994 4.233730
## [25] 3.487577 4.813440 3.848536 4.319251 4.484402 2.903430 2.664923 2.540653
## [33] 3.415197 2.896785 3.518162 3.416584 3.307441 3.746620
```

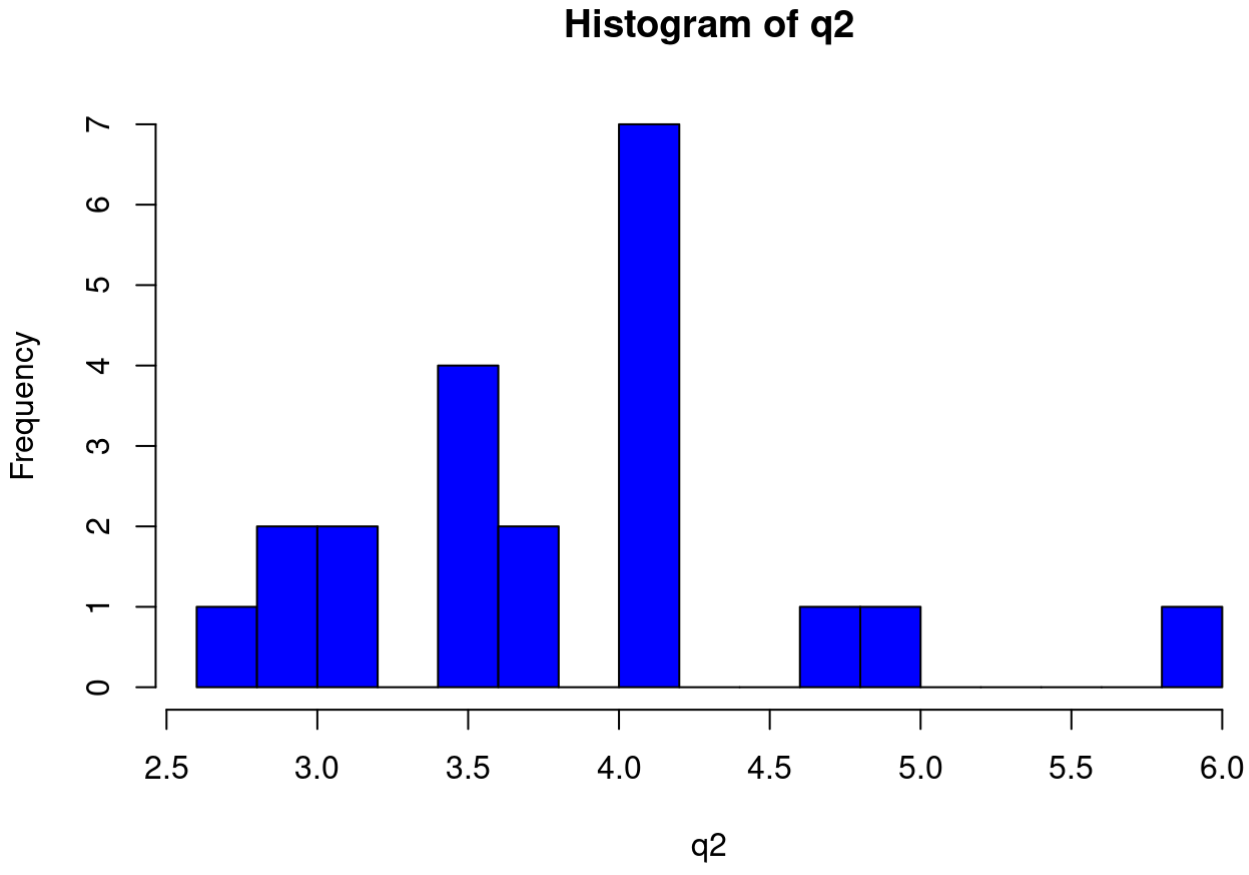
```
hist(q1, col = "GREEN", breaks=5)
```



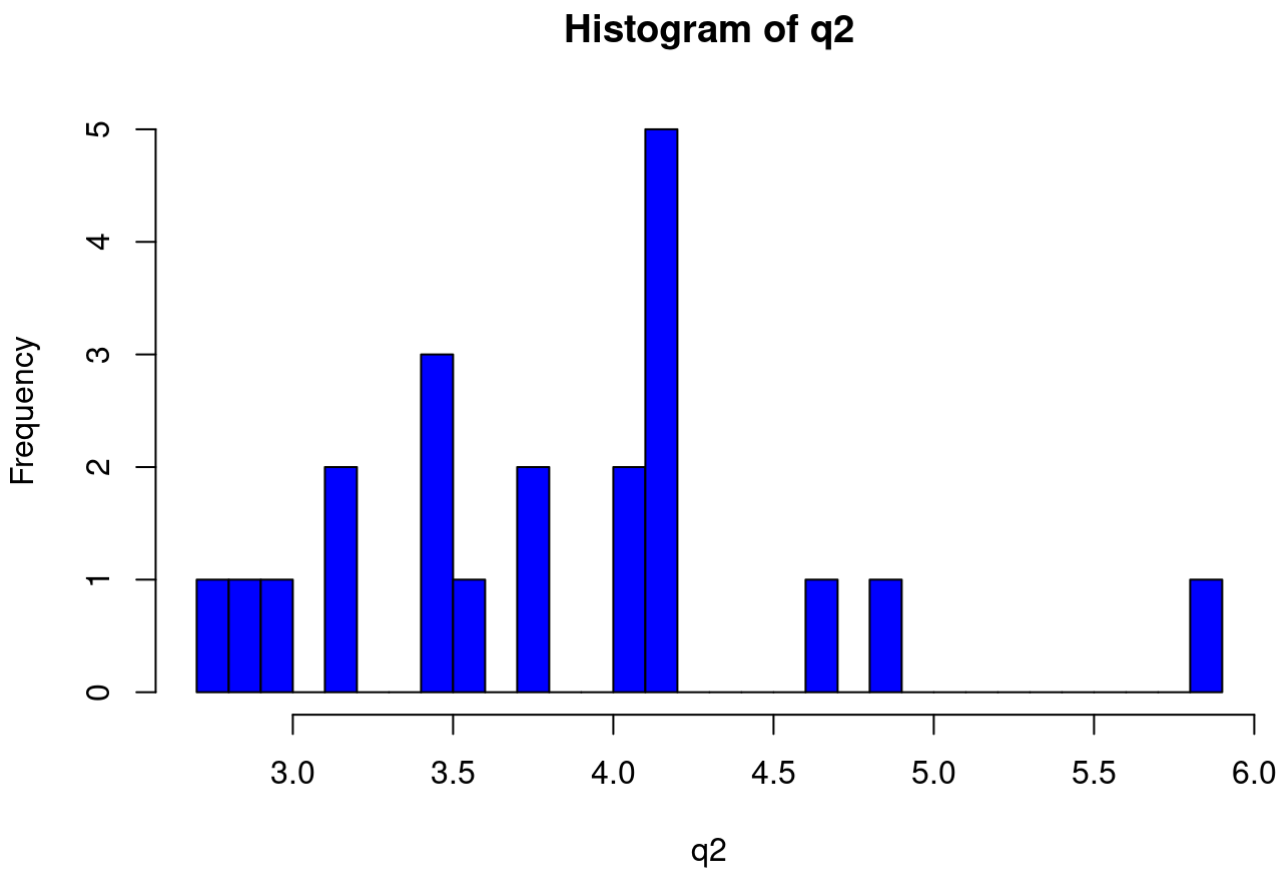

```
hist(q2, col = "BLUE", breaks=5)
```



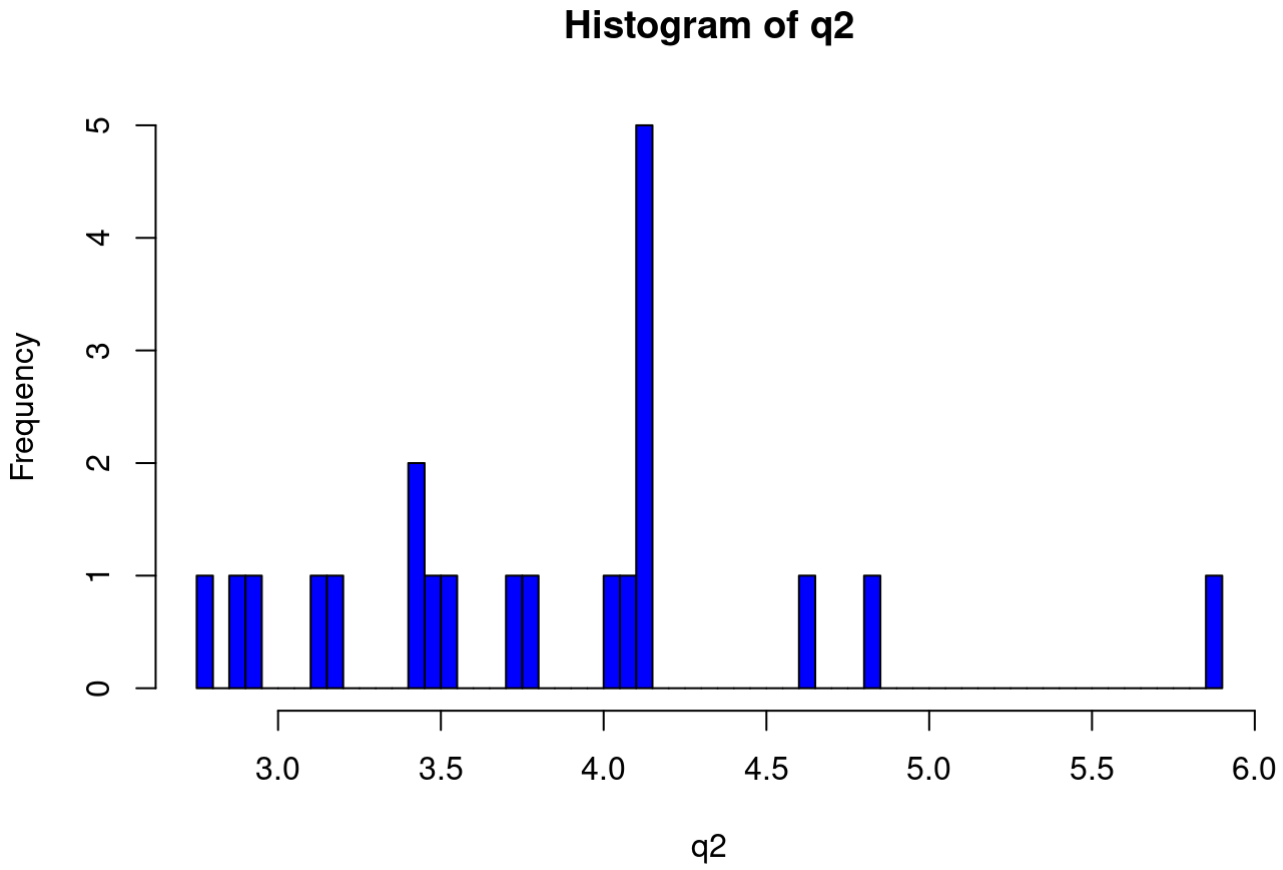

```
hist(q2, col = "BLUE", breaks=15)
```



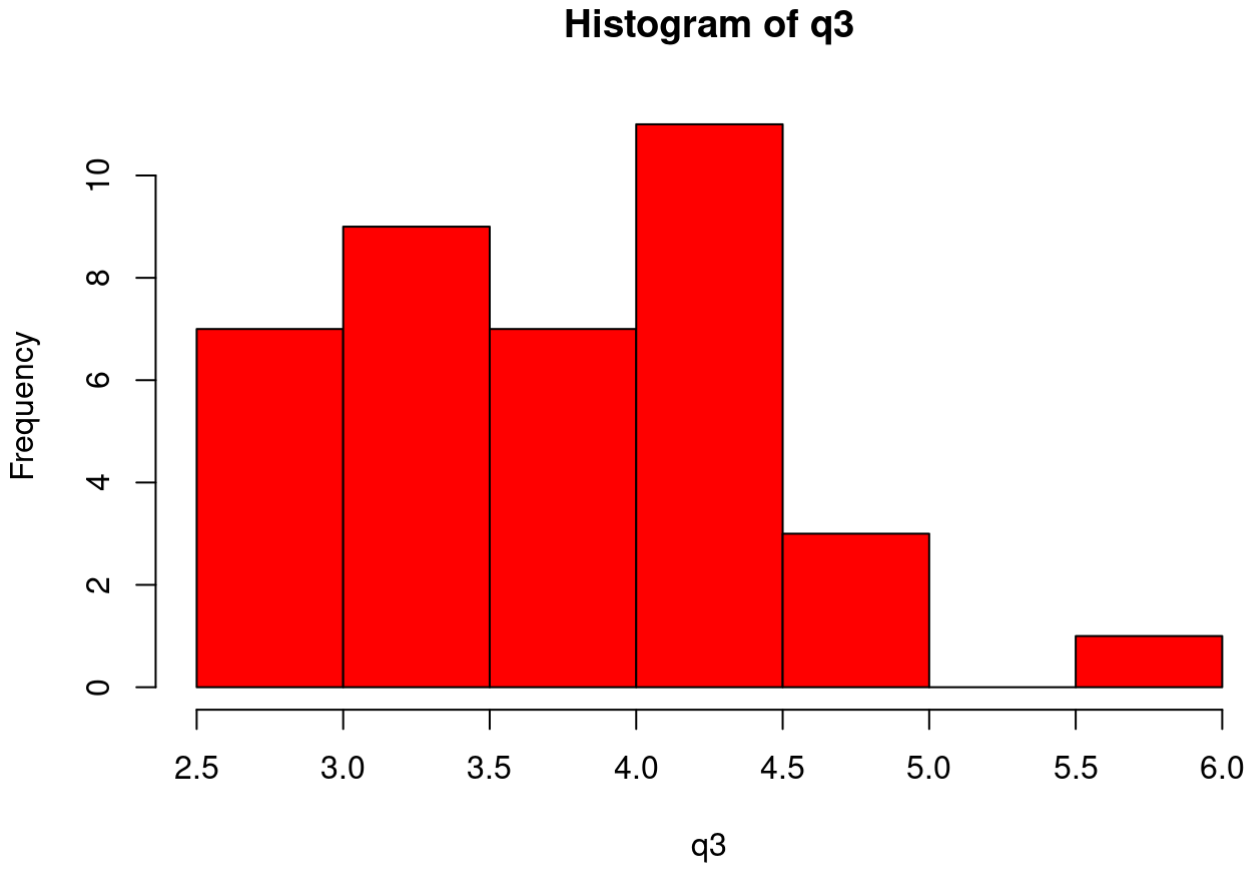
```
hist(q2, col = "BLUE", breaks=30)
```



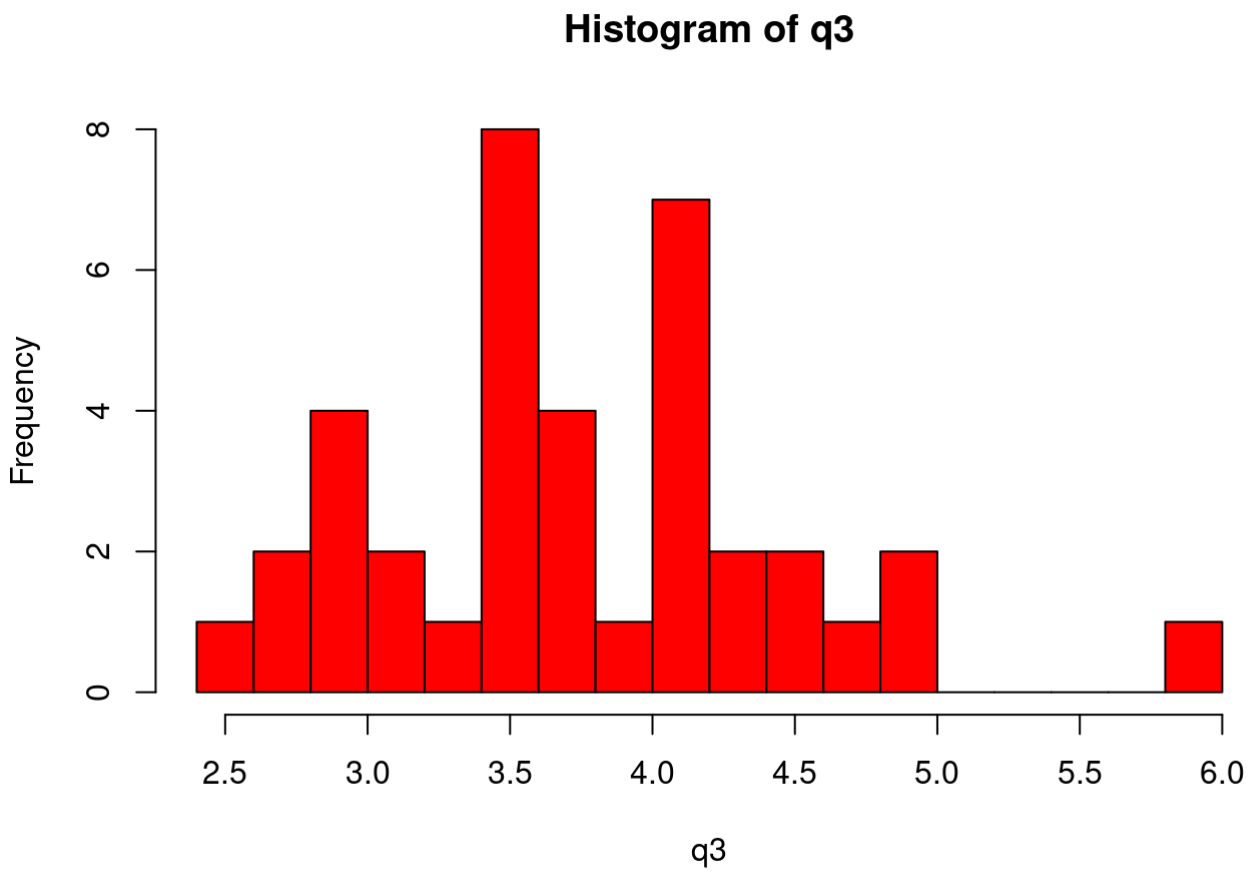

```
hist(q2, col = "BLUE", breaks=50)
hist(q2, col = "BLUE", breaks=75)
```



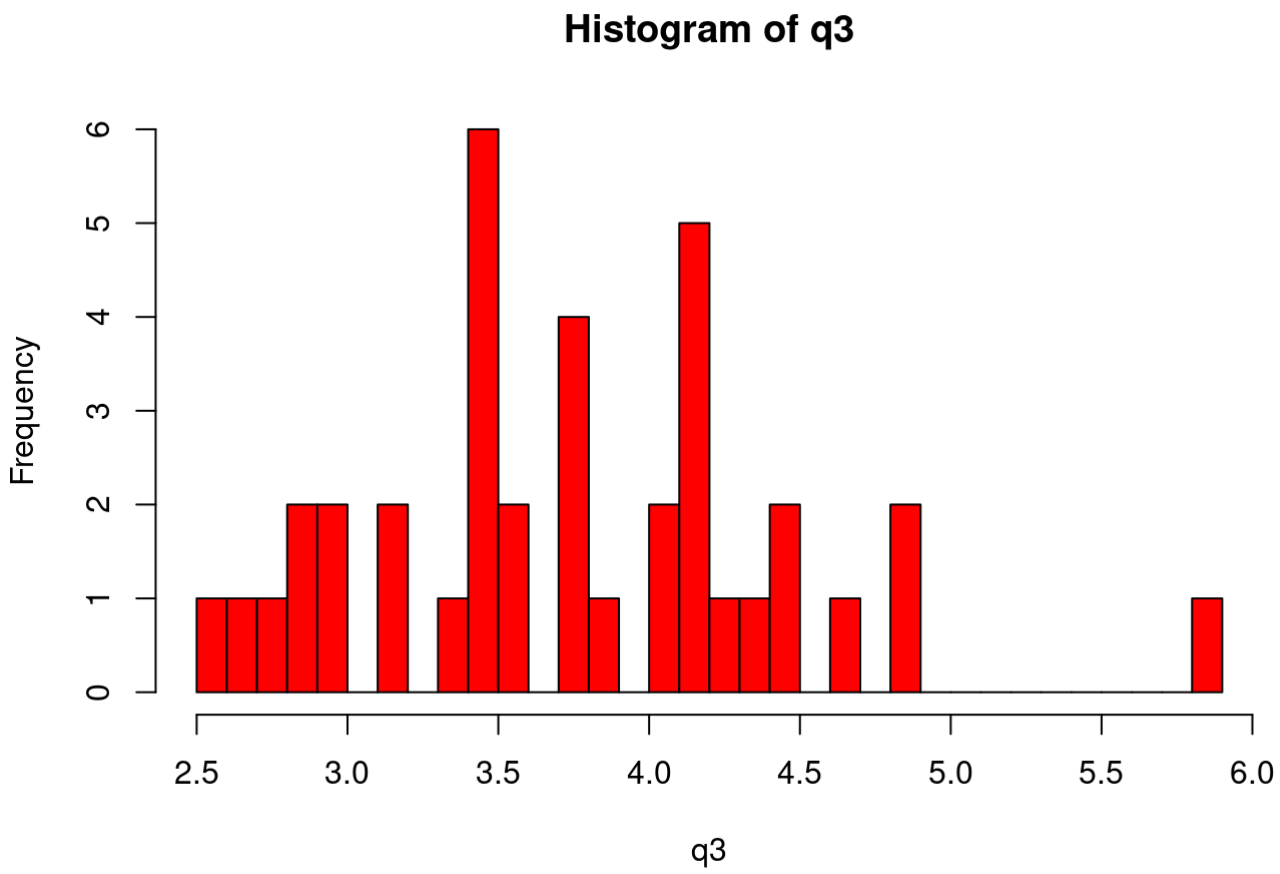

```
hist(q3, col = "RED", breaks=5)
```



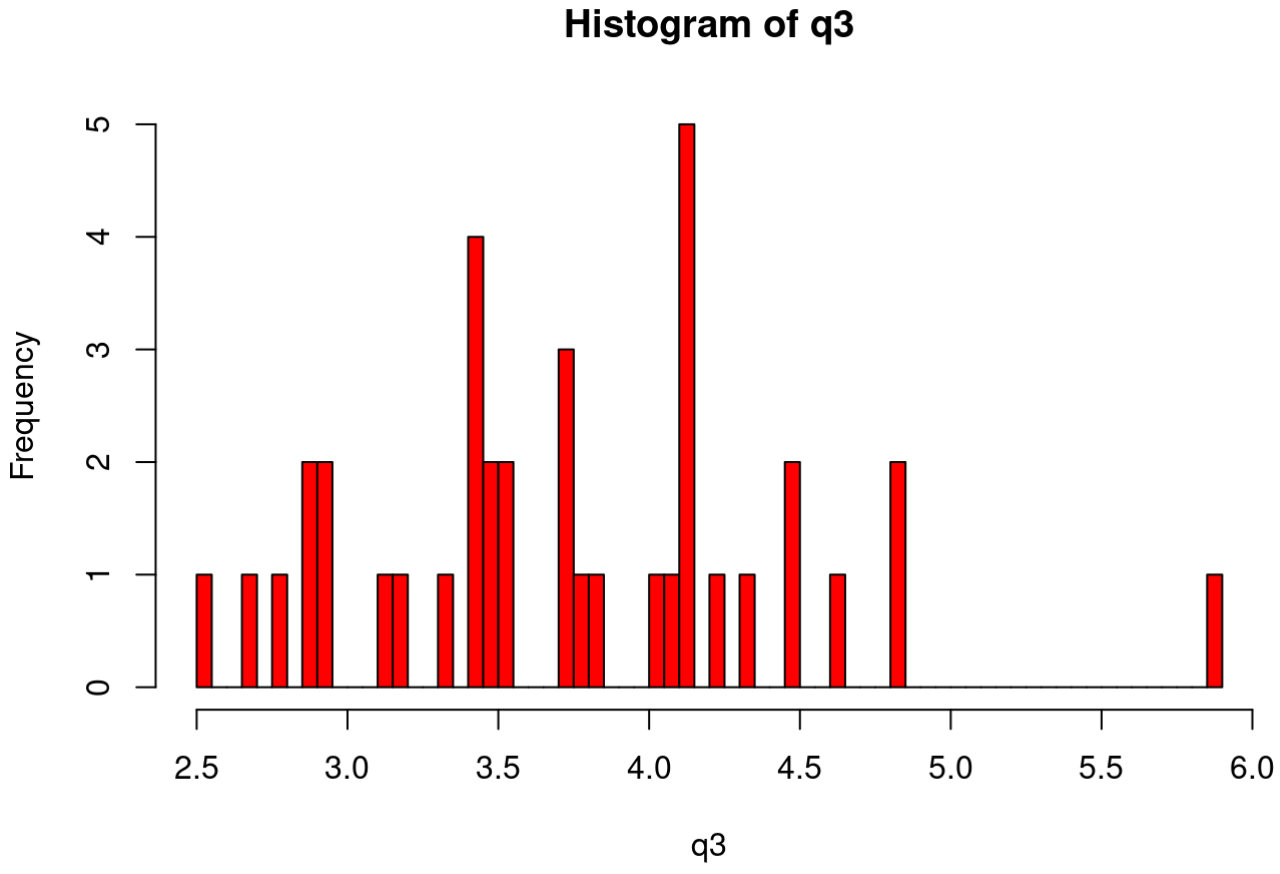
```
hist(q3, col = "RED", breaks=15)
```



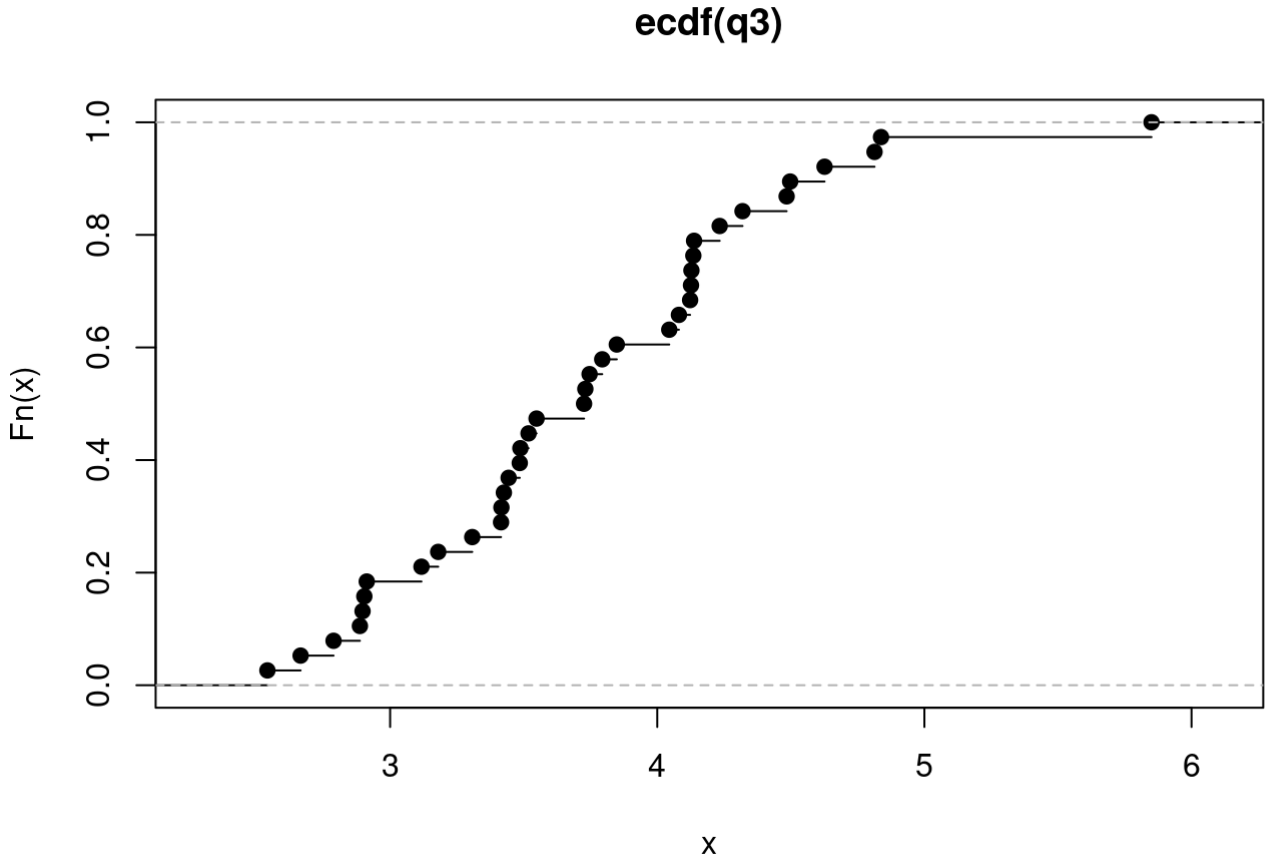

```
hist(q3, col = "RED", breaks=30)
```



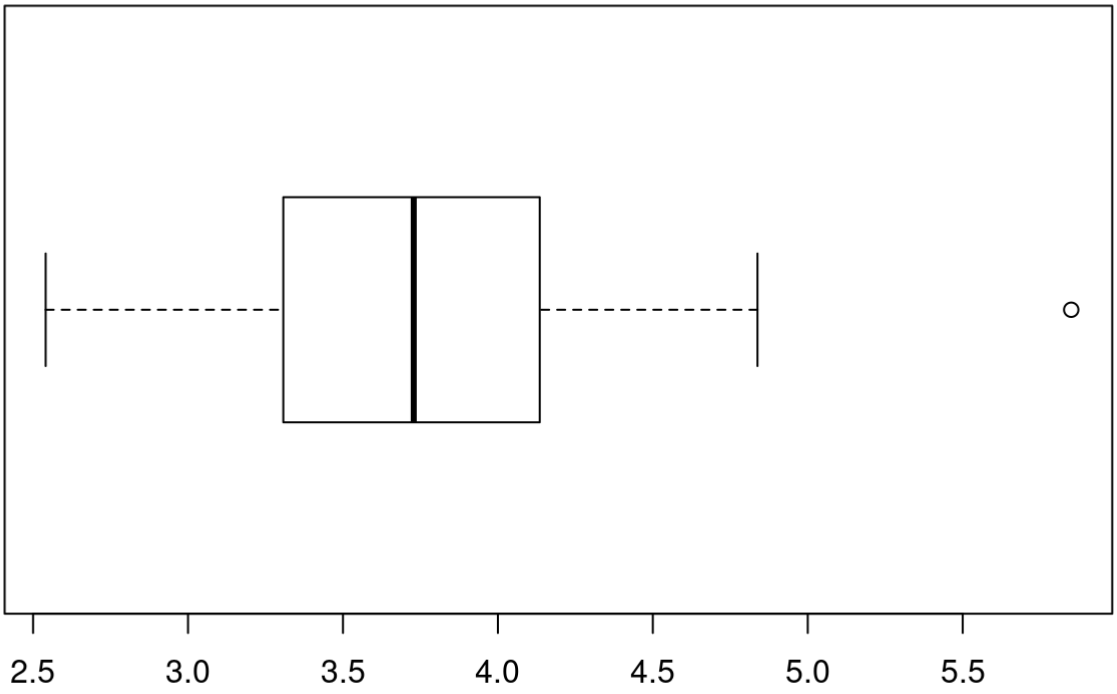

```
hist(q3, col = "RED", breaks=50)
hist(q3, col = "RED", breaks=75)
```



```
plot(ecdf(q3))
```




```
boxplot(q3, horizontal = TRUE)
```



- Resposta para a questão 4:
- Histograma da questão 1 não é possível concluir quase nada sobre ele pela falta de amostras
- Já no Histograma da questão 2 e 3 quanto maior o número de breaks, mais próximo de um gráfico uniforme fica
- Contudo, ao utilizar o boxplot é possível indicar uma tendência de unimodal simétrica
- Talvez, aumentando absurdos a quantidade de amostras, seja possível observar melhor a uniformidade no histograma

…