Comparação de Alternativas

Rafael Tenfen

Data de entrega: 15/05/2021

Descrição da atividade

O objetivo desta atividade é aplicar as técnicas de comparação de alternativas. A atividade é dividida em três partes:

1. Comparação usando ICs vs. teste t
2. Comparação de duas alternativas
3. Comparação de múltiplas alternativas

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;
  - os arquivos de dados referentes às Partes 2 e 3, que são necessários para o processamento do .Rmd.

Configuração

Nesta atividade, a única configuração necessária consiste em carregar o pacote `ggplot2` e o arquivo `compar-altern.R`, que são usados na Parte 1 da atividade.

```
library(ggplot2)
source("compar-altern.R")
```

Parte 1: Comparação usando ICs vs. teste t

Uma das formas de determinar se duas variáveis são estatisticamente diferentes é observando os seus intervalos de confiança. Existem três resultados possíveis para essa comparação:

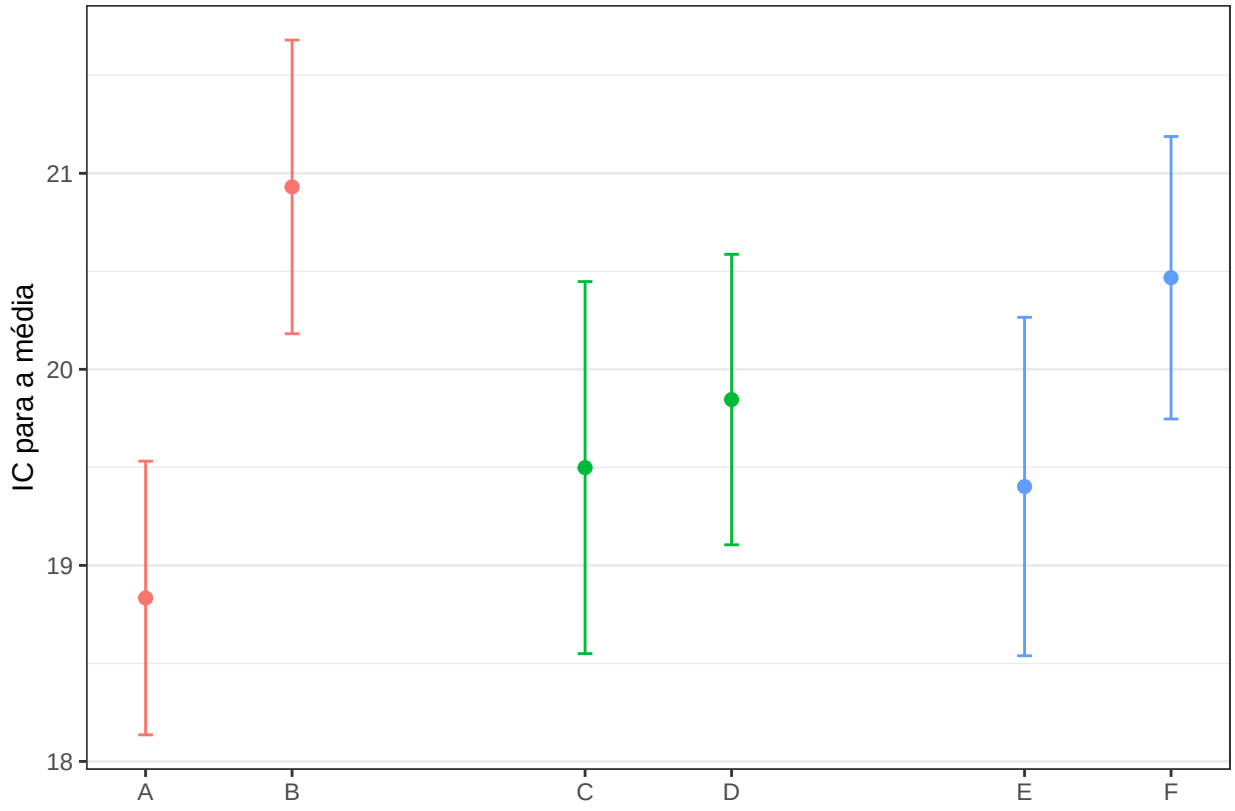
1. *Não existe sobreposição entre os ICs.* Nesse caso, as variáveis são estatisticamente diferentes.

2. *Existe sobreposição entre os ICs, e ao menos um deles inclui a média da outra variável.* Nesse caso, as variáveis são estatisticamente equivalentes.
3. *Existe sobreposição entre os ICs, mas nenhum deles inclui a média da outra variável.* Nesse caso não é possível afirmar nada, sendo necessário realizar um teste t (ou equivalente) para determinar se a diferença é estatisticamente significativa.

O gráfico abaixo ilustra os três resultados. As variáveis comparadas são as colunas A–F do conjunto de dados contido no arquivo `comparacao-ic.dat`, e os ICs têm um nível de confiança de 95%. As conclusões visuais são as seguintes:

1. As variáveis A e B são estatisticamente diferentes, e A $<$ B.
2. As variáveis C e D são estatisticamente equivalentes.
3. Não é possível afirmar se E $<$ F ou não, é preciso realizar um teste t para ver se a diferença é estatisticamente significativa.

```
dados <- read.table("comparacao-ic.dat", head=TRUE)
dados.ic <- geraIC(dados)
plotaIC(dados.ic)
```



Para esta primeira parte, você deve comparar os pares de variáveis representados no gráfico (A/B, C/D, E/F) usando o teste t com um nível de confiança de 95% (o mesmo usado para gerar os ICs). Para cada par de variáveis, indique claramente (a) o resultado da comparação (ou seja, se as variáveis são ou não estatisticamente diferentes) e (b) se esse resultado é idêntico ao obtido pela comparação visual dos ICs. Considere que as observações não são pareadas.

Análise e respostas

```
# seu código R aqui
dados
```

```
##           A        B        C        D        E        F
## 1  21.67424 21.48488 17.58587 21.26818 18.64442 21.78631
## 2  17.89704 24.32349 20.55486 20.01863 18.66001 19.56320
## 3  18.21695 20.44786 22.16888 20.11890 16.25540 20.91498
## 4  19.07907 19.49555 15.30860 21.91918 18.65243 19.49381
## 5  18.00849 20.48859 20.85825 19.61256 20.70046 18.68500
## 6  18.04723 23.16563 21.01211 18.10359 20.39522 21.16747
## 7  17.83902 18.79209 18.85052 22.14951 20.09999 18.50934
## 8  17.62202 22.55174 18.90674 18.95269 18.19454 21.17046
## 9  18.71426 22.49435 18.87110 20.96972 18.61681 21.63742
## 10 18.75473 21.00471 18.21992 19.12810 16.61094 20.61974
## 11 16.79095 20.51034 19.04561 23.20460 18.89368 20.26016
## 12 18.62689 19.81680 18.00323 20.04881 19.51039 19.12733
## 13 17.83667 19.95021 18.44749 19.58112 22.41193 17.97558
## 14 17.97756 21.47243 20.12892 19.99748 21.00303 22.87319
## 15 19.25654 23.60541 21.91899 17.74181 18.00883 20.80591
## 16 20.34458 20.26990 19.77943 18.66476 19.71110 22.82785
## 17 21.97173 18.41395 18.97798 16.63992 16.73078 17.63928
## 18 18.33997 19.41463 18.17761 18.31801 20.75641 21.17257
## 19 21.90886 20.88739 18.32566 20.41141 20.94583 22.43292
## 20 17.76329 20.02441 24.83167 20.06820 23.24223 20.68431
```

```
nc = 0.95
alfa = 1 - nc
(dados.A.shap = shapiro.test(dados$A))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  dados$A
## W = 0.82545, p-value = 0.002124
```

```
dados.A.shap$p.value > alfa
```

```
## [1] FALSE
```

```
(dados.B.shap = shapiro.test(dados$B))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  dados$B
## W = 0.95462, p-value = 0.4428
```

```r
dados.B.shap$p.value > alfa
```

```
## [1] TRUE
```

```r
# Mesmo o shapiro apontando que alguns nados não possuem a distribuição normal, será adotado o test t p
# P < alfa significa que tem diferença
# P > alfa significa que não tem diferença

(dados.AB.test = t.test(dados$A, dados$B, conf.level=nc, paired=FALSE))
```

```
##
##  Welch Two Sample t-test
##
## data:  dados$A and dados$B
## t = -4.2872, df = 37.812, p-value = 0.0001202
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.087673 -1.106755
## sample estimates:
## mean of x mean of y
##  18.83350  20.93072
```

```r
dados.AB.test$p.value < alfa
```

```
## [1] TRUE
```

```r
(dados.CD.test = t.test(dados$C, dados$D, conf.level=nc, paired=FALSE))
```

```
##
##  Welch Two Sample t-test
##
## data:  dados$C and dados$D
## t = -0.60358, df = 35.889, p-value = 0.5499
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.5139020  0.8195255
## sample estimates:
## mean of x mean of y
##  19.49867  19.84586
```

```r
dados.CD.test$p.value < alfa
```

```
## [1] FALSE
```

```r
(dados.EF.test = t.test(dados$E, dados$F, conf.level=nc, paired=FALSE))
```

```
##
##  Welch Two Sample t-test
##
```

```
## data:  dados$E and dados$F
## t = -1.9827, df = 36.821, p-value = 0.0549
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.15377092  0.02353298
## sample estimates:
## mean of x mean of y
##  19.40222  20.46734
```

```
dados.EF.test$p.value < alfa
```

```
## [1] FALSE
```

```
dados.EF.test$conf.int
```

```
## [1] -2.15377092  0.02353298
## attr(,"conf.level")
## [1] 0.95
```

- *Respostas aqui*
- a) Foi identificado que o par A/B é estatiscamente diferente, pois não inclui o valor 0 no seu intervalo de confiança e o valor p-value 10^{-4} é menor que o alfa 0.05. Já os pares C/D e E/F são estatisticamente equivalentes, pois o valor 0 está contido no intervalo de confiança e o p-value de C/D é 0.55 maior que o alfa 0.05 e também o p-value do par E/F é 0.055 maior que o alfa também.
- b) No caso dos pares A/B e C/D de fato é identico ao que é possível visualizar na comparação visual dos ICs, A/B são estatisticamente diferentes, C/D são estisticamente equivalentes e no caso do par E/F é possível ter conclusões mais precisas com o test t e afirmar que eles também são estisticamente equivalentes pois o seu intervalo de confiança -2.15, 0.02 contém o valor 0.

Parte 2: Comparação de dois algoritmos de ordenação

Nesta segunda parte iremos comparar o tempo de execução de dois algoritmos de ordenação, ordenação por inserção (*InsertionSort*) e ordenação por seleção (*SelectionSort*). Esses dois algoritmos têm complexidade $O(n^2)$, e são considerados ineficientes.

Para essa comparação iremos usar tempos de execução medidos pelo script Python `sortcomp2.py`. Esse script mede o tempo que cada algoritmo leva para ordenar **o mesmo vetor** de `n` elementos. O número de rodadas pode ser passado como parâmetro na linha de comando (por *default* são realizadas 3 rodadas). A cada rodada os elementos do vetor sofrem uma permutação aleatória; logo, é possível (mas pouco provável) que o vetor esteja (quase) em ordem (de)crescente.

O script pode ser executado no RStudio Cloud. Na janela inferior esquerda, normalmente usada para o console, há uma aba Terminal, na qual você pode executar comandos do Linux.

Os passos deste experimento são os seguintes:

1. Execute o script usando o comando `python sortcomp2.py 2`. O número de rodadas (2, no exemplo) fica a seu critério.

2. Verifique se os dados obtidos satisfazem as premissas do teste *t*. Use o teste de Shapiro-Wilk como teste de normalidade.[1]
3. Analise os dados obtidos, e identifique qual dos algoritmos tem melhor desempenho, considerando níveis de confiança de 95% e 99%.
4. O que mudaria na análise dos dados caso os vetores ordenados pelos dois algoritmos tivessem o mesmo número de elementos mas não fossem idênticos? Modifique o script (você pode usar o editor do próprio RStudio), alterando a linha

```
v2 = v[:]
```

para

```
v2 = v[:]
random.shuffle(v2)
```

Refaça os passos 1, 2 e 3 e veja se as conclusões são as mesmas obtidas inicialmente.

Lembre-se que os tempos de execução dos algoritmos devem ser salvos em um arquivo de dados para que sua análise seja reproduzível. Para facilitar essa tarefa, o script já gera a saída em um formato apropriado; você pode redirecionar a saída do script para um arquivo (por exemplo, `python sortcomp2.py 2 >parte2.dat`) ou simplesmente criar o arquivo de dados no próprio editor do RStudio Cloud (crie um novo arquivo texto e cole a saída do script).

Análise e respostas

```
# seu código R aqui
nc = 0.95
alfa = 1 - nc

p2.dados.2 <- read.table("parte2.2.dat", head=TRUE)
p2.dados.2
```

```
##   insertion selection
## 1   4.26571   2.65676
## 2   4.23612   2.84006
```

```
#shapiro.test(p2.dados.2$insertion)

p2.dados.5 <- read.table("parte2.5.dat", head=TRUE)
p2.dados.5$insertion
```

```
## [1] 4.21064 4.07841 4.08664 4.13663 4.08064
```

```
(p2.dados.5.insertion.shap = shapiro.test(p2.dados.5$insertion))
```

[1] O procedimento correto em caso de falha das premissas seria usar um outro teste para comparação das observações, como o teste de postos sinalizados de Wilcoxon. **Nesta atividade**, porém, para simplificar, use o teste *t*, independentemente do resultado do teste de normalidade.

```
##
##  Shapiro-Wilk normality test
##
## data:  p2.dados.5$insertion
## W = 0.79864, p-value = 0.07899
```

```
p2.dados.5.insertion.shap$p.value > alfa
```

```
## [1] TRUE
```

```
(p2.dados.5.selection.shap = shapiro.test(p2.dados.5$selection))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  p2.dados.5$selection
## W = 0.87484, p-value = 0.2866
```

```
p2.dados.5.selection.shap$p.value > alfa
```

```
## [1] TRUE
```

```
# p-value de shapiro é > alfa, então é possível assumir que a distribuição é normal

p2.dados.100 <- read.table("parte2.100.dat", head=TRUE)
#p2.dados.100$insertion
#p2.dados.100

# P < alfa significa que tem diferença
# P > alfa significa que não tem diferença

(p2.dados.100.insertion.shap = shapiro.test(p2.dados.100$insertion))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  p2.dados.100$insertion
## W = 0.79168, p-value = 1.386e-10
```

```
p2.dados.100.insertion.shap$p.value > alfa
```

```
## [1] FALSE
```

```
(p2.dados.100.selection.shap = shapiro.test(p2.dados.100$selection))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  p2.dados.100$selection
## W = 0.98151, p-value = 0.1739
```

```r
p2.dados.100.selection.shap$p.value > alfa
```

```
## [1] TRUE
```

```r
(p2.dados.100.test95 = t.test(p2.dados.100$insertion, p2.dados.100$selection, conf.level=0.95))
```

```
##
##  Welch Two Sample t-test
##
## data:  p2.dados.100$insertion and p2.dados.100$selection
## t = 163.13, df = 189.96, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1.524328 1.561643
## sample estimates:
## mean of x mean of y
##  4.232962  2.689976
```

```r
(p2.dados.100.test99 = t.test(p2.dados.100$insertion, p2.dados.100$selection, conf.level=0.99))
```

```
##
##  Welch Two Sample t-test
##
## data:  p2.dados.100$insertion and p2.dados.100$selection
## t = 163.13, df = 189.96, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 99 percent confidence interval:
##  1.518375 1.567597
## sample estimates:
## mean of x mean of y
##  4.232962  2.689976
```

```r
str(p2.dados.100.test95)
```

```
## List of 10
##  $ statistic  : Named num 163
##   ..- attr(*, "names")= chr "t"
##  $ parameter  : Named num 190
##   ..- attr(*, "names")= chr "df"
##  $ p.value    : num 4.05e-206
##  $ conf.int   : num [1:2] 1.52 1.56
##   ..- attr(*, "conf.level")= num 0.95
##  $ estimate   : Named num [1:2] 4.23 2.69
##   ..- attr(*, "names")= chr [1:2] "mean of x" "mean of y"
##  $ null.value : Named num 0
##   ..- attr(*, "names")= chr "difference in means"
##  $ stderr     : num 0.00946
##  $ alternative: chr "two.sided"
##  $ method     : chr "Welch Two Sample t-test"
##  $ data.name  : chr "p2.dados.100$insertion and p2.dados.100$selection"
##  - attr(*, "class")= chr "htest"
```

```r
round(p2.dados.100.test95$p.value, 4) < 0.05
```

```
## [1] TRUE
```

```r
round(p2.dados.100.test99$p.value, 4) < 0.01
```

```
## [1] TRUE
```

```r
p2.dados.100.shuffle <- read.table("parte2.shuffle.100.dat", head=TRUE)
#p2.dados.100.shuffle

(p2.dados.100.shuffle.insertion.shap = shapiro.test(p2.dados.100.shuffle$insertion))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  p2.dados.100.shuffle$insertion
## W = 0.96245, p-value = 0.006032
```

```r
p2.dados.100.shuffle.insertion.shap$p.value > alfa
```

```
## [1] FALSE
```

```r
(p2.dados.100.shuffle.selection.shap = shapiro.test(p2.dados.100.shuffle$selection))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  p2.dados.100.shuffle$selection
## W = 0.98125, p-value = 0.1662
```

```r
p2.dados.100.shuffle.selection.shap$p.value > alfa
```

```
## [1] TRUE
```

```r
(p2.dados.100.shuffle.test95 = t.test(p2.dados.100.shuffle$insertion, p2.dados.100.shuffle$selection, c
```

```
##
##  Welch Two Sample t-test
##
## data:  p2.dados.100.shuffle$insertion and p2.dados.100.shuffle$selection
## t = 148.62, df = 191.33, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1.397588 1.435183
## sample estimates:
## mean of x mean of y
##  4.130530  2.714145
```

```
(p2.dados.100.shuffle.test99 = t.test(p2.dados.100.shuffle$insertion, p2.dados.100.shuffle$selection, c
```

```
##
##  Welch Two Sample t-test
##
## data:  p2.dados.100.shuffle$insertion and p2.dados.100.shuffle$selection
## t = 148.62, df = 191.33, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 99 percent confidence interval:
##  1.391590 1.441181
## sample estimates:
## mean of x mean of y
##  4.130530  2.714145
```

```
#str(p2.dados.100.shuffle.test95)
round(p2.dados.100.shuffle.test95$p.value, 4) < 0.05
```

```
## [1] TRUE
```

```
round(p2.dados.100.shuffle.test99$p.value, 4) < 0.01
```

```
## [1] TRUE
```

- *Respostas aqui*
- 1: Executado com 2, 5 e 100
- 2: Bom, com 2 dados não é possível executar, com 5 os foi registrado que tanto o 'selection' quanto o 'insertion' tem a distribuição normal, porém com 100 dados as amostras possuem distribuição normal para o 'selection' mas não é possível dizer o mesmo para o 'insertion', de todo modo as amostras com 100 dados serão utilizados por terem maior quantidade de dados
- 3: Para 95% de nivel de confiança e 100 amostras é possível observar que os dados estatisicamente possuem diferença, pois o intervalo de confiança 1.52, 1.56 não contem o valor 0 e o p-value: 0 é menor que o alfa: 0.05. Para 99% de nivel de confiança e 100 amostras é possível observar que os dados estatisicamente possuem diferença, pois o intervalo de confiança 1.52, 1.57 não contem o valor 0 e o p-value: 0 é menor que o alfa: 0.01. A média de 'insertion' 4.23 é maior que a média de 2.69 e como estamos com a métrica de tempo de execução, quanto menor melhor, então o algoritmo de 'selection' é melhor.
- 4: A análise continou a mesma adicionando o shuffle ao script, até executei novamente pra certificar... Entao tanto para 95, quanto 99% de nivel de confiança para as 100 amostras e possivel observar estatisticamente que os dados possuem diferença e a média do 'insertion' é maior que a média do 'selection' e como estamos com a métrica de tempo de execução, quanto menor melhor, então o algoritmo de 'selection' é melhor.

Parte 3: Comparação de três algoritmos de ordenação

Na terceira parte iremos comparar o tempo de execução de três algoritmos de ordenação, *QuickSort*, *MergeSort* e *HeapSort*. Esses três algoritmos têm complexidade $O(n \log n)$ no caso médio, e são considerados eficientes. Para essa comparação iremos usar tempos de execução medidos pelo script Python `sortcomp3.py`, que é análogo ao script usado na segunda parte da atividade.

Neste experimento, primeiro execute o script usando o comando `python sortcomp3.py 2`. O número de rodadas (2, no exemplo) fica a seu critério.

A seguir, faça uma análise de variância adotando um nível de confiança de 95%, e responda aos seguintes itens:

1. Qual a porcentagem de variação que pode ser explicada pelas alternativas e qual a porcentagem explicada pelo ruído das medições?
2. Mostre a tabela ANOVA (conforme o esquema abaixo) e determine se existem diferenças estatisticamente significativas entre os tempos médios de resposta de cada algoritmo.

Fonte de variação	Alternativas	Erros	Total
Soma de quadrados			
Graus de liberdade			
Média quadrática			
F calculado			
F crítico			

3. Caso a ANOVA indique que há diferenças estatisticamente significativas, ranqueie os algoritmos de acordo com o seu tempo médio de resposta (use o teste de Tukey).

Novamente, lembre-se que os tempos de execução dos algoritmos devem ser salvos em um arquivo de dados para que sua análise seja reproduzível. Caso necessário, reveja as orientações ao final da Parte 2.

Análise e respostas

```
# seu código R aqui
p3.dados <- read.table("parte3.100.dat", head=TRUE)
#p3.dados
p3.stack = stack(p3.dados)
#p3.stack
(p3.aov = aov(values~ind, p3.stack))
```

```
## Call:
##    aov(formula = values ~ ind, data = p3.stack)
##
## Terms:
##                       ind Residuals
## Sum of Squares  2.5348710 0.0079581
## Deg. of Freedom         2       297
##
## Residual standard error: 0.005176369
## Estimated effects may be unbalanced
```

```
summary(p3.aov)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## ind           2  2.535   1.267   47302 <2e-16 ***
## Residuals   297  0.008   0.000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#str(p3.aov)

ssa = 2.535
sse = 0.008
sst = ssa + sse
(alt = ssa/sst * 100)
```

```
## [1] 99.68541
```

```r
(ruido = sse/sst * 100)
```

```
## [1] 0.3145891
```

```r
ssa2 = ssa/2
sse2 = sse/(2*297)

(hsd = TukeyHSD(p3.aov, conf.level=0.95))
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = values ~ ind, data = p3.stack)
##
## $ind
##                     diff        lwr        upr p adj
## heap-merge   0.1037652  0.1020408  0.1054896     0
## quick-merge -0.1211713 -0.1228957 -0.1194469     0
## quick-heap  -0.2249365 -0.2266609 -0.2232121     0
```

```r
(Fcrit = qf(0.95, p3.aov$rank - 1, p3.aov$df.residual))
```

```
## [1] 3.026153
```

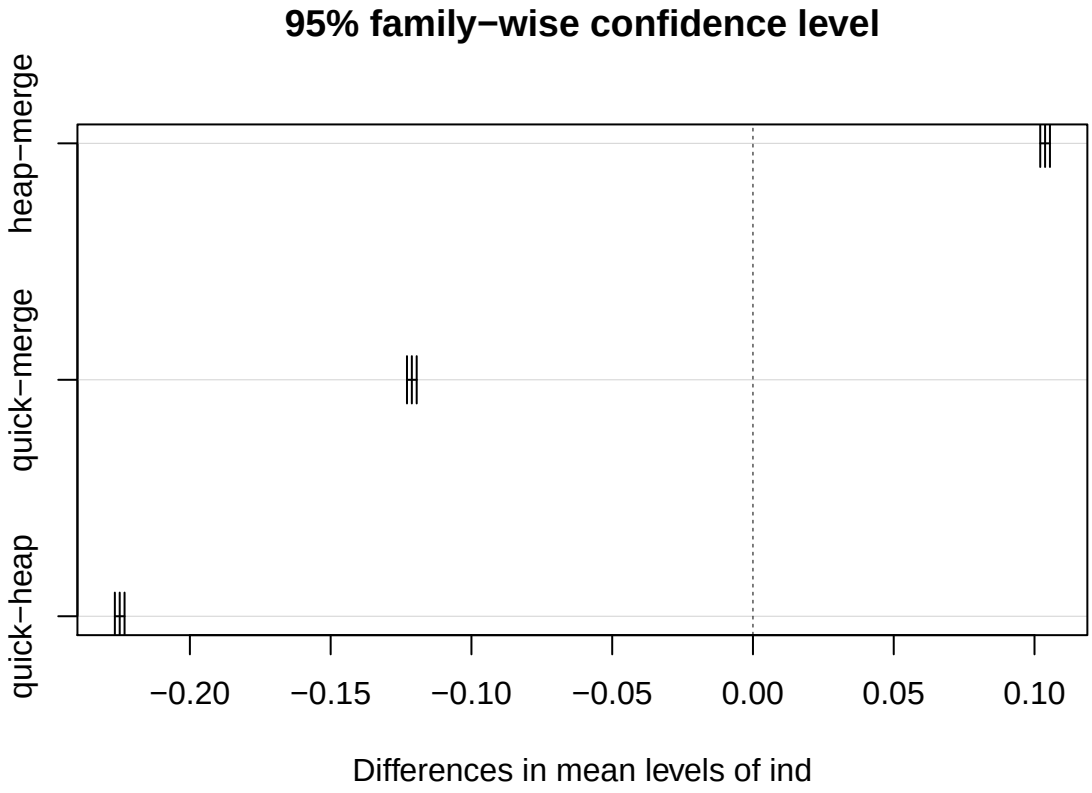
```r
Fcalc = 47302
Fcalc > Fcrit
```

```
## [1] TRUE
```

```r
apply(p3.dados, 2, mean)
```

```
##     merge      heap     quick
## 0.2089795 0.3127447 0.0878082
```

```r
plot(hsd)
```



- *Respostas aqui*
- 1: A porcentagem de variação explicada devido a diferença entre as alternativas é de 99.69%. Já a porcentagem de variação explicada devido a ruido nas medições é de 0.31%.
- 2:

Fonte de variação	Alternativas	Erros	Total
Soma de quadrados	2.535	0.008	2.543
Graus de liberdade	2	297	299
Média quadrática	1.2675	10^{-5}	
F calculado	47302		
F crítico	3.03		

- 3: Fcalc > Fcrit: TRUE, entao aov indica que existe uma diferença significativa. Utilizando TukeyHSD para verificar a diferença entre os pares p adj é sempre 0 que é menor que o alfa 0.05. Assim, de acordo com a métrica de tempo de execução, ou seja quanto menor melhor, a ordem fica 'quick' sendo o melhor, 'merge' o próximo e 'heap' o pior.