Universidade Federal do Ceará (UFC/Sobral)

Aula 04 - Métodos Computacionais Aplicados

Prof. Weligton Gomes

03/04/2023

Instalação e Ativação de Pacotes

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

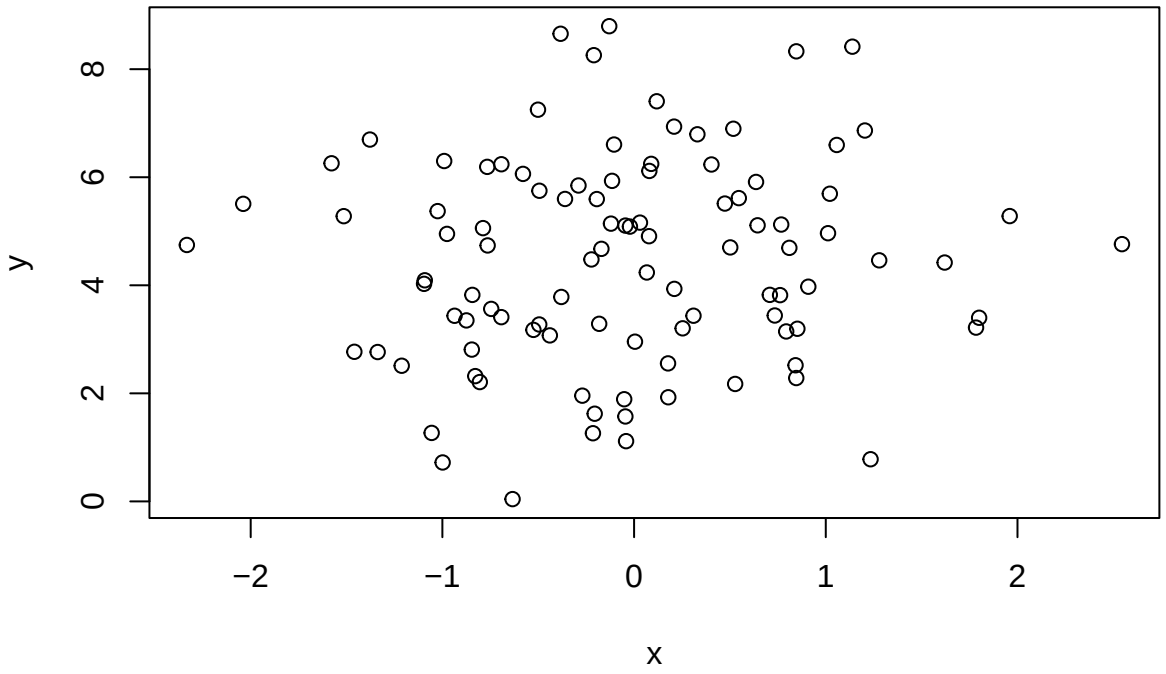
Extrair números aleatórios de uma distribuição normal - rnorm()

Observação: r de random ou aleatório e norm - de distribuição normal

```
x<-rnorm(100)
y<-rnorm(100,mean = 5, sd = 2)
print(x)
```

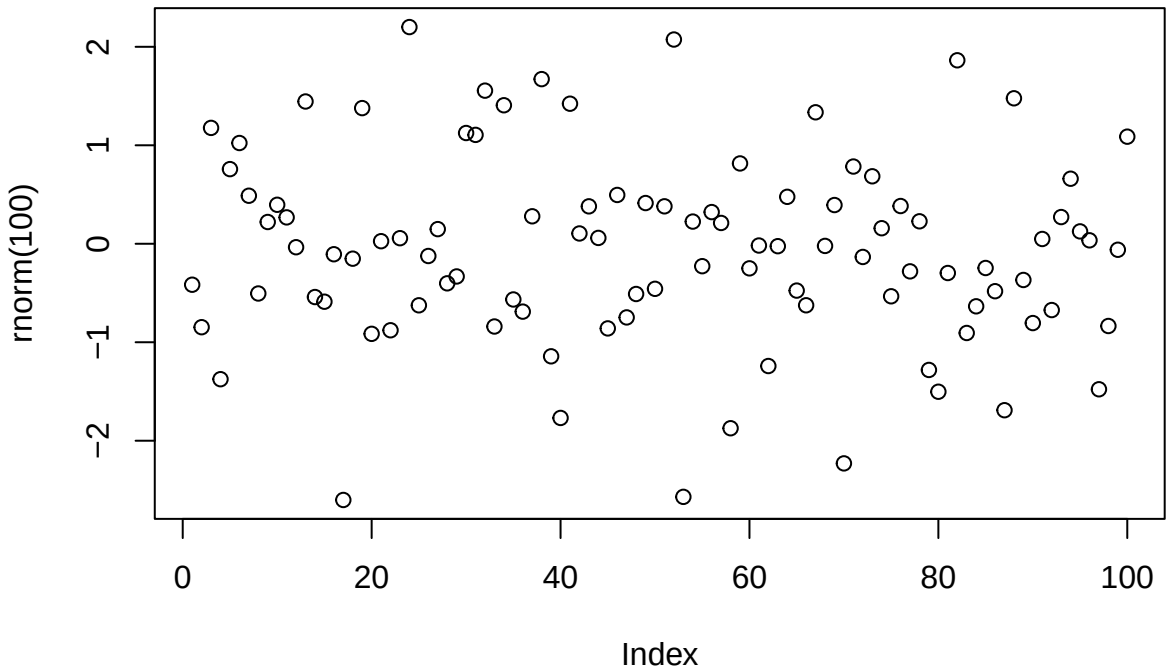
```
##   [1] -0.129549865  0.846305946 -1.337595085 -0.936455037 -0.692531696
##   [6] -0.804907173  0.636212059  0.842224484 -0.360248468  1.011703353
##  [11] -1.024772935 -0.692199319  1.057186444 -0.222516071  0.767512492
##  [16]  0.077919394  1.021194723 -0.494800138 -0.045430382 -0.210007781
##  [21]  0.845944310  1.800045151 -0.214578917 -1.091677407 -0.493914796
##  [26] -0.990564966 -0.041786179  0.209993296 -0.975955279  0.908926666
##  [31] -1.055875812 -0.501289499  0.501720149 -1.514626037  0.809958853
##  [36] -0.270010410 -0.998673599 -0.766019112  0.089386997 -0.289949187
##  [41] -0.379620892  1.278771361 -0.182025398 -0.763972633  0.708062982
##  [46]  0.733924840  0.517832245  1.959426217 -0.114785301 -0.439417398
##  [51]  1.783851365  0.851814410 -2.038962228 -0.170347244 -0.828542415
##  [56] -0.633931561 -0.525021409 -1.212132154  0.066403735 -0.045364219
##  [61]  0.253373908  0.030690254  0.793588637  0.644274454  0.309883016
##  [66] -1.095603238  1.138742829  0.473442367  0.527263705  0.178423180
##  [71]  0.079635242  1.203953087  0.177155210  0.330295052  2.545131234
##  [76] -0.194512327 -1.578141708  0.118153404  0.208569388  0.761586068
##  [81] -0.745187869 -2.333021337  1.620068062  0.546382324 -0.843757656
##  [86] -0.846411722 -1.378036555 -0.205437528 -0.021713982  1.233730320
##  [91] -0.104527674 -0.383271555 -1.459156402 -0.051592450 -0.579599849
##  [96]  0.004632402 -0.120530334 -0.788119235  0.403575456 -0.874725041
```

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



#Visualização de dados pelo comando View

```
View(rnorm(100))
plot(rnorm(100))
```



Os dois caracteres <- devem ser lidos como um único símbolo: uma seta apontando para a variável à qual o valor é atribuído. Isso é conhecido como operador de atribuição.

```
x <- 2
x + x
```

```
## [1] 4
```

Operação com Vetores:

Um ponto forte do R é que ele pode manipular vetores de dados inteiros como objetos únicos. Um vetor de dados é simplesmente uma matriz de números e vetor pode ser construído assim:

```
weight <- c(60, 72, 57, 90, 95, 72)
height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
bmi <- weight/height^2
bmi
```

```
## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

```
mean(bmi)
```

```
## [1] 23.13262
```

```
summary(bmi)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   19.59   20.04   21.58   23.13   24.25   31.38
```

Considere, por exemplo, o cálculo da média e do desvio padrão da variável weight de forma manual

Média = Somatório de weight/número de observações

```
sum(weight)
```

```
## [1] 446
```

```
sum(weight)/length(weight)
```

```
## [1] 74.33333
```

Desvio padrão = A raíz quadrada da variância. Onde a variáncia é o somatório dos desvios médio quadrado / n-1

```
xbar <- sum(weight)/length(weight)
weight - xbar
```

```
## [1] -14.333333  -2.333333 -17.333333  15.666667  20.666667  -2.333333
```

```
(weight - xbar)^2
```

```
## [1] 205.444444   5.444444 300.444444 245.444444 427.111111   5.444444
```

```
sum((weight - xbar)^2)
```

```
## [1] 1189.333
```

```
sqrt(sum((weight - xbar)^2)/(length(weight) - 1))
```

```
## [1] 15.42293
```

De forma mais fácil e ágil, tem-se a função mean():

```
mean(height)
```

```
## [1] 1.791667
```

```
sd(height)
```

```
## [1] 0.1002829
```

```
mean(weight)
```

```
## [1] 74.33333
```

```
sd(weight)
```

```
## [1] 15.42293
```

Como um exemplo um pouco mais complicado do que R pode fazer, considere o seguinte: A regra prática é que o IMC (BMI) para um indivíduo com peso normal deve estar entre 20 e 25, e queremos saber se nossos dados se desviam sistematicamente deste. Você pode usar um teste t de uma amostra para avaliar se o IMC das seis pessoas pode ser considerado como tendo uma média de 22,5, dado que eles vêm de uma distribuição normal. Para isso, você pode usar a função t.test.

(Você pode não conhecer a teoria do teste t ainda. O exemplo é incluído aqui principalmente para dar alguma indicação de como é a saída estatística "real").

```
t.test(bmi, mu=22.5)
```

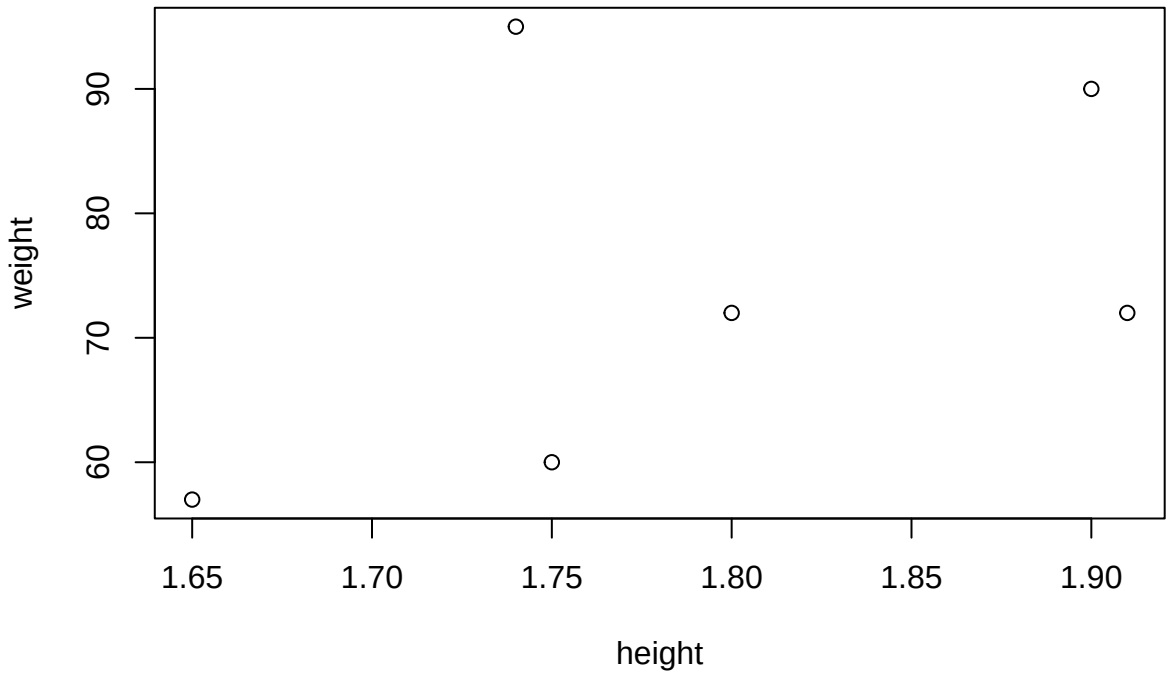
```
##
##  One Sample t-test
##
## data:  bmi
## t = 0.34488, df = 5, p-value = 0.7442
## alternative hypothesis: true mean is not equal to 22.5
## 95 percent confidence interval:
##  18.41734 27.84791
## sample estimates:
## mean of x
##  23.13262
```

O argumento $\mu = 22.5$ atribui um valor ao argumento formal mu, que representa a letra grega μ convencionalmente usada para a média teórica. Se isso não for fornecido, o teste t usaria o valor padrão $mu = 0$.

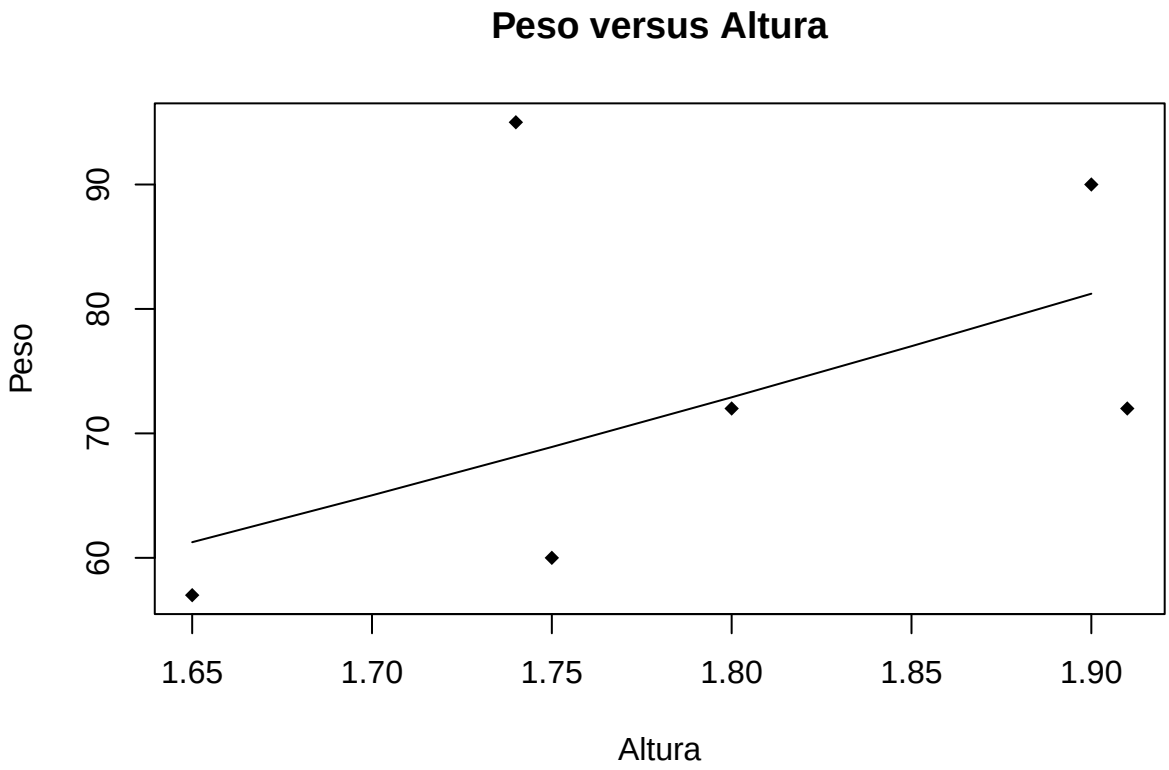
Para um teste como este, obtemos uma impressão mais extensa do que nos exemplos anteriores.Você pode se concentrar no valor de p, que é usado para testar a hipótese de que a média é $22,5$. O valor de p não é pequeno, indicando que não é improvável obter dados como os observados se a média fosse de fato $22,5$. (Falando de forma simplificada; na verdade, p é a probabilidade de obter um valor t maior que $0,3449$ ou menor que $-0,3449$). No entanto, você também pode observar o intervalo de confiança de 95% para a verdadeira média. Este intervalo é bastante amplo, indicando que realmente temos muito pouca informação sobre a verdadeira média.

Gráficos no R

```
plot(height,weight)
```




```
plot(height, weight, pch=18, main = "Peso versus Altura", xlab = "Altura", ylab = "Peso")
#pch = plotting Character (0:18)
hh <- c(1.65, 1.70, 1.75, 1.80, 1.85, 1.90)
lines(hh, 22.5 * hh^2)
```




```
args(plot.default)
```

```
## function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
##     log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
```

```
##     ann = par("ann"), axes = TRUE, frame.plot = axes, panel.first = NULL,
##     panel.last = NULL, asp = NA, xgap.axis = NA, ygap.axis = NA,
##     ...)
## NULL
```

Funções e Argumentos

Muitas coisas em R são feitas usando chamadas de função, comandos que se parecem com uma aplicação de uma função matemática de uma ou várias variáveis; por exemplo, log (x) ou plot (altura, peso).

```
args(plot.default)
```

```
## function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
##     log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
##     ann = par("ann"), axes = TRUE, frame.plot = axes, panel.first = NULL,
##     panel.last = NULL, asp = NA, xgap.axis = NA, ygap.axis = NA,
##     ...)
## NULL
```

Vetores:

Existem mais dois tipos, vetores de caracteres e vetores lógicos. Um vetor de caracteres é um vetor de strings de texto, cujos elementos são especificados e impressos entre aspas:

```
c("Huey","Dewey","Louie")
```

```
## [1] "Huey"  "Dewey" "Louie"
```

Não importa se você usa símbolos de aspas simples ou duplas, desde que a aspa esquerda seja igual à direita:

```
a<-c('Huey','Dewey','Louie')
c(T,T,F,T)
```

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

```
bmi > 25
```

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

Se você imprimir um vetor de caracteres, geralmente ele sairá com aspas adicionadas a cada elemento. Existe uma maneira de evitar isso, ou seja, usar a função cat Por exemplo,

```
cat(c("Huey","Dewey","Louie"))
```

```
## Huey Dewey Louie
```

Observação: Valores Missing no R são representados por NA.

Funções que criam vetores

Aqui, apresentamos três funções, c, seq e rep, que são usadas para criar vetores em várias situações. O primeiro deles, c, já foi introduzido. É a abreviação de "concatenar", juntando itens de ponta a ponta, que é exatamente o que a função faz:

```
c(42,57,12,39,1,3,4)
```

```
## [1] 42 57 12 39  1  3  4
```

```r
x <- c(1, 2, 3)
y <- c(10, 20)
c(x, y, 5)
```

```
## [1]  1  2  3 10 20  5
```

A segunda função, seq (“sequência”), é usada para séries de números equidistantes.

```r
seq(4,9)
```

```
## [1] 4 5 6 7 8 9
```

```r
4:9
```

```
## [1] 4 5 6 7 8 9
```

Se você quiser uma sequência em saltos de 2.

```r
seq(4,10,2)
```

```
## [1]  4  6  8 10
```

A terceira função, rep (“replicar”), é usada para gerar valores repetidos. Ele é usado em duas variantes, dependendo se o segundo argumento é um vetor ou um único número:

```r
oops <- c(7,9,13)
rep(oops,3)
```

```
## [1]  7  9 13  7  9 13  7  9 13
```

```r
rep(oops,1:3)
```

```
## [1]  7  9  9 13 13 13
```

```r
rep(1:2,c(10,15))
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```r
rep(1:2,each=10) #Igual ao comando abaixo.
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

```r
rep(1:2,c(10,10))
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

Em R, a noção de matriz é estendida a elementos de qualquer tipo, então você poderia ter, por exemplo, uma matriz de caracteres (string). Matrizes e arranjos são representadas como vetores com dimensões:

Exemplo de matrizes nas notas de aula

```r
x <- 1:12
dim(x) <- c(3,4)

y<-matrix(c(2,4,5,8,3,2,1,12,9),nrow=3,byrow=TRUE)

x <- matrix(1:12,nrow=3,byrow=T)
rownames(x) <- LETTERS[1:3]
colnames(x) <- LETTERS[4:7]
x
```

```
##   D  E  F  G
## A 1  2  3  4
## B 5  6  7  8
## C 9 10 11 12
```

```
colnames(x)<-c("Cajá","Melão", "Abacaxi", "Maçã")
x
```

```
##   Cajá Melão Abacaxi Maçã
## A    1     2       3    4
## B    5     6       7    8
## C    9    10      11   12
```

Matriz Transposta

```
t(x)
```

```
##         A B  C
## Cajá    1 5  9
## Melão   2 6 10
## Abacaxi 3 7 11
## Maçã    4 8 12
```

Inversa de Matriz

```
solve(y)
```

```
##             [,1]        [,2]        [,3]
## [1,]  0.01570681  0.12565445 -0.03664921
## [2,] -0.36649215  0.06806283  0.18848168
## [3,]  0.48691099 -0.10471204 -0.13612565
```

Você pode "colar" vetores ou juntar coluna ou linha, usando as funções cbind (coluna) e rbind (linha).

```
cbind(A=1:4,B=5:8,C=9:12)
```

```
##      A B  C
## [1,] 1 5  9
## [2,] 2 6 10
## [3,] 3 7 11
## [4,] 4 8 12
```

```
rbind(A=1:4,B=5:8,C=9:12)
```

```
##   [,1] [,2] [,3] [,4]
## A    1    2    3    4
## B    5    6    7    8
## C    9   10   11   12
```