

## T3 – Operaciones basicas en R

### Nociones de programación en R

Os comandos (scripts em linguagem R) devem ser explicitados na linha de comando (console) ou por meio de um editor de texto (a ser comentado depois). Veja os exemplos de operações matemáticas (Quadro 2):

Quadro 2: R como super-calculadora.

%Carregando pacote para tabela knitr & kableExtra

```
require(knitr)
```

```
## Loading required package: knitr
```

```
require(kableExtra)
```

```
## Loading required package: kableExtra
```

```
if(!require(data.table)){install.packag  
es("data.table"); require(data.table)}
```

```
## Loading required package: data.table
```

```
#Criando conjunto de dados
```

```
#OperacoesR <- data.table(Operador = c( "+", "-", "*", "/", "^", "abs", "exp", "log", "sqrt", "rnorm"),  
#                               Descricao = c( "Adicao", "Subtracao", "Multiplicacao", "Divisao", "Exponenciacao",  
#                               Exemplos = c( "5+4", "5-4", "5*4", "5/4", "5^4", "abs(-5)", "exp(5)", "log(exp(5))", "sqrt(64)", "rnorm(100000)" )  
Tabela <- data.table(Operador = c("+", "-", "*", "/", "^", "abs", "exp", "log", "sqrt", "rnorm" ),  
                      Descricao = c("Adicao", "Subtracao", "Multiplicacao", "Divisao", "Exponenciacao",  
                      Exemplos = c("5+4", "5-4", "5*4", "5/4", "5^4", "abs(-5)", "exp(5)", "log(exp(5))", "sqrt(64)", "rnorm(100000)"),
```

```
## Warning in data.table(Operador = c("+", "-", "*", "/", "^", "abs", "exp", :  
## Item 2 is of size 9 but maximum size is 10 (recycled leaving remainder of 1  
## items)
```

```
kable(Tabela, booktabs = T)
```

Operador	Descricao	Exemplos
+	Adicao	5+4
-	Subtracao	5-4
*	Multiplicacao	5*4
/	Divisao	5/4
^	Exponenciacao	5^4
abs	Valor Absoluto	abs(-5)
exp	log logaritmo (default 'e log natural)	exp(5)
log	Raiz quadrada	log(exp(5))
sqrt	sorteia 100.000 valores de uma Normal	sqrt(64)
rnorm	Adicao	rnorm(100000)

### O R como calculadora

O forma de uso mais básica do R é usá-lo como calculadora. Os operadores matemáticos básicos são:

+ para soma, - subtração, \* multiplicação, / divisão e ^ exponenciação. Digite as seguintes operações na linha de comandos do R:

```
%Carregando pacote para tabela knitr & kableExtra
```

```
2+2
```

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

```
2*2
```

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

```
2/2
```

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

```
2-2
```

```
## [1] 0
```

```
2^2
```

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

Use parênteses para separar partes dos cálculos, por exemplo, para fazer a conta  $4+16$ , dividido por 4, elevado ao quadrado:

```
((4+16)/4)^2
```

```
## [1] 25
```

## Funções do R

O R tem diversas funções que podemos usar para fazer os cálculos desejados. O uso básico de uma função é escrever o nome da função e colocar os argumentos entre parênteses, por exemplo: função(argumentos). função especifica qual função irá usar e argumentos especifica os argumentos que serão avaliados pela função. Não se assuste com esses nomes, com um pouco de pratica eles se tornarão triviais.

Antes de usar uma função precisamos aprender como usá-la. Para isso vamos aprender como abrir e usar os arquivos de ajuda do R.

## Objetos do R (O que são?):

O que são os Objetos do R? Existem muitos tipos de objetos no R que só passamos a conhecê-los bem com o passar do tempo. Por enquanto vamos aprender os tipos básicos de objetos.

a) vetores: uma seqüência de valores numéricos ou de caracteres (letras, palavras).

b) matrizes: coleção de vetores em linhas e colunas, todos os vetores dever ser do mesmo tipo (numérico ou de caracteres).

c) dataframe: O mesmo que uma matriz, mas aceita vetores de tipos diferentes (numérico e caracteres).

Geralmente nós guardamos nossos dados em objetos do tipo data frame, pois sempre temos variáveis numéricas e variáveis categóricas (por exemplo, largura do rio e nome do rio, respectivamente).

d) listas: conjunto de vetores, dataframes ou de matrizes. Não precisam ter o mesmo comprimento, é a forma que a maioria das funções retorna os resultados.

e) funções: as funções criadas para fazer diversos cálculos também são objetos do R.

No decorrer da apostila você verá exemplos de cada um destes objetos.

## Pastas

A função setwd faz com que você consiga ir a pasta desejada.

```
setwd("Documentos/Cursos/SAR-PolSAR-Course/Code/")
```

A função `getwd` serve para saber em que pasta você se encontra.

```
getwd()
```

```
## [1] "/Users/lacccan/Documents/Cursos/SAR-PolSAR-Course/Code"
```

## Como criar objetos

### Objetos vetores com valores numéricos

Vamos criar um conjunto de dados de contém o número de espécies de aves (riqueza) coletadas em 10 locais. As riquezas são 22, 28, 37, 34, 13, 24, 39, 5, 33, 32.

```
aves<-c(22,28,37,34,13,24,39,5,33,32)
```

O comando `<-` (sinal de menor e sinal de menos) significa assinalar (assign). Indica que tudo que vem após este comando será salvo com o nome que vem antes. É o mesmo que dizer “salve os dados a seguir com o nome de aves”.

A letra `c` significa concatenar (colocar junto). Entenda como “agrupe os dados entre parênteses dentro do objeto que será criado” neste caso no objeto `aves`.

Para ver os valores (o conteúdo de um objeto), basta digitar o nome do objeto na linha de comandos. `aves`

A função `length` fornece o número de observações (`n`) dentro do objeto.

```
length(aves)
```

```
## [1] 10
```

Objetos vetores com caracteres (letras, variáveis categóricas).

Também podemos criar objetos que contém letras ou palavras ao invés de números. Porém, as letras ou palavras devem vir entre aspas `"`.

```
letras<-c("a","b","c","da","edw")
```

```
7
```

```
## [1] 7
```

```
letras
```

```
## [1] "a" "b" "c" "da" "edw"
```

```
palavras<-c("Manaus","Boa Vista","Belém","Brasília")
```

```
palavras
```

```
## [1] "Manaus" "Boa Vista" "Belém" "Brasília"
```

Crie um objeto “misto”, com letras e com números. Funciona? Esses números realmente são números?

Note a presença de aspas, isso indica que os números foram convertidos em caracteres. Evite criar vetores “mistos”, a menos que tenha certeza do que está fazendo.

## Operações com vetores

Podemos fazer diversas operações usando o objeto `aves`, criado acima.

```
max(aves) #valor máximo contido no objeto aves
```

```
## [1] 39
```

```
min(aves) #valor mínimo
```

```
## [1] 5
```

```
sum(aves) #Soma dos valores de aves
```

```
## [1] 267
```

```
aves^2 #...
```

```
## [1] 484 784 1369 1156 169 576 1521 25 1089 1024
```

```
aves/10
```

```
## [1] 2.2 2.8 3.7 3.4 1.3 2.4 3.9 0.5 3.3 3.2
```

Agora vamos usar o que já sabemos para calcular a média dos dados das aves.

```
n.aves<-length(aves) # número de observações (n)  
media.aves<-sum(aves)/n.aves #média
```

Para ver os resultados basta digitar o nome dos objetos que você salvou

```
n.aves # para ver o número de observações (n)
```

```
## [1] 10
```

```
media.aves # para ver a média
```

```
## [1] 26.7
```

Você não ficará surpreso em saber que o R já tem uma função pronta para calcular a média.

```
mean(aves)
```

```
## [1] 26.7
```

## Acessar valores dentro de um objeto [colchetes]

Caso queira acessar apenas um valor do conjunto de dados use colchetes []. Isto é possível porque o R salva os objetos como vetores, ou seja, a sequência na qual você incluiu os dados é preservada. Por exemplo, vamos acessar o quinto valor do objeto aves.

```
aves[5] # Qual o quinto valor de aves?
```

```
## [1] 13
```

```
palavras[3] # Qual a terceira palavra?
```

```
## [1] "Belém"
```

Para acessar mais de um valor use c para concatenar dentro dos colchetes [c(1,3,...)]:

```
aves[c(5,8,10)] # acessa o quinto, oitavo e décimo valores
```

```
## [1] 13 5 32
```

Para excluir um valor, ex: o primeiro, use:

```
aves[-1] # note que o valor 22, o primeiro do objeto aves, foi excluído
```

```
## [1] 28 37 34 13 24 39 5 33 32
```

Caso tenha digitado um valor errado e queira corrigir o valor, especifique a posição do valor e o novo valor. Por exemplo, o primeiro valor de aves é 22, caso estivesse errado, ex: deveria ser 100, basta alterarmos o valor da seguinte maneira.

```
aves[1]<-100 # O primeiro valor de aves deve ser 100
aves
```

```
## [1] 100 28 37 34 13 24 39 5 33 32
```

```
aves[1]<-22 # Vamos voltar ao valor antigo
```

## Gerar seqüências (usando : ou usando seq)

: (dois pontos) Dois pontos " : " é usado para gerar seqüências de um em um, por exemplo a seqüência de 1 a 10:

```
1:10 # O comando : é usado para especificar seqüências.
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
5:16 # Aqui a seqüência vai de 5 a 16
```

```
## [1] 5 6 7 8 9 10 11 12 13 14 15 16
```

seq A função seq é usada para gerar seqüências especificando os intervalos. Vamos criar uma seqüência de 1 a 10 pegando valores de 2 em 2.

```
seq(1,10,2) #seq é a função para gerar seqüências, o default é em intervalos de 1.
```

```
## [1] 1 3 5 7 9
```

A função seq funciona assim:

seq(from = 1, to = 10, by = 2 ), seqüência(de um, a dez, em intervalos de 2)

```
seq(1,100,5) #seqüência de 1 a 100 em intervalos de 5
```

```
## [1] 1 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96
```

```
seq(0.01,1,0.02)
```

```
## [1] 0.01 0.03 0.05 0.07 0.09 0.11 0.13 0.15 0.17 0.19 0.21 0.23 0.25 0.27
```

```
## [15] 0.29 0.31 0.33 0.35 0.37 0.39 0.41 0.43 0.45 0.47 0.49 0.51 0.53 0.55
```

```
## [29] 0.57 0.59 0.61 0.63 0.65 0.67 0.69 0.71 0.73 0.75 0.77 0.79 0.81 0.83
```

```
## [43] 0.85 0.87 0.89 0.91 0.93 0.95 0.97 0.99
```

## Gerar repetições (rep)

rep Vamos usar a função rep para repetir algo n vezes.

```
rep(5,10) # repete o valor 5 dez vezes
```

```
## [1] 5 5 5 5 5 5 5 5 5 5
```

A função rep funciona assim:

rep(x, times=y) # rep(repita x, y vezes) # onde x é o valor ou conjunto de valores que deve ser repetido, e times é o número de vezes

```
rep(2,5)
```

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

```
rep("a",5) # repete a letra "a" 5 vezes

## [1] "a" "a" "a" "a" "a"

rep(1:4,2) # repete a sequência de 1 a 4 duas vezes

## [1] 1 2 3 4 1 2 3 4

rep(1:4,each=2) # note a diferença ao usar o comando each=2

## [1] 1 1 2 2 3 3 4 4

rep(c("A","B"),5) # repete A e B cinco vezes.

## [1] "A" "B" "A" "B" "A" "B" "A" "B" "A" "B"

rep(c("A","B"),each=5) # repete A e B cinco vezes.

## [1] "A" "A" "A" "A" "A" "B" "B" "B" "B" "B"

rep(c("Três","Dois","Sete","Quatro"),c(3,2,7,4)) # Veja que neste

## [1] "Três" "Três" "Três" "Dois" "Dois" "Sete" "Sete"
## [8] "Sete" "Sete" "Sete" "Sete" "Sete" "Quatro" "Quatro"
## [15] "Quatro" "Quatro"
```

caso a primeira parte do comando indica as palavras que devem ser repetidas e a segunda parte indica quantas vezes cada palavra deve ser repetida

## Gerar dados aleatórios

runif (Gerar dados aleatórios com distribuição uniforme)

runif(n, min=0, max=1) # gera uma distribuição uniforme com n valores,

começando em min e terminando em max.

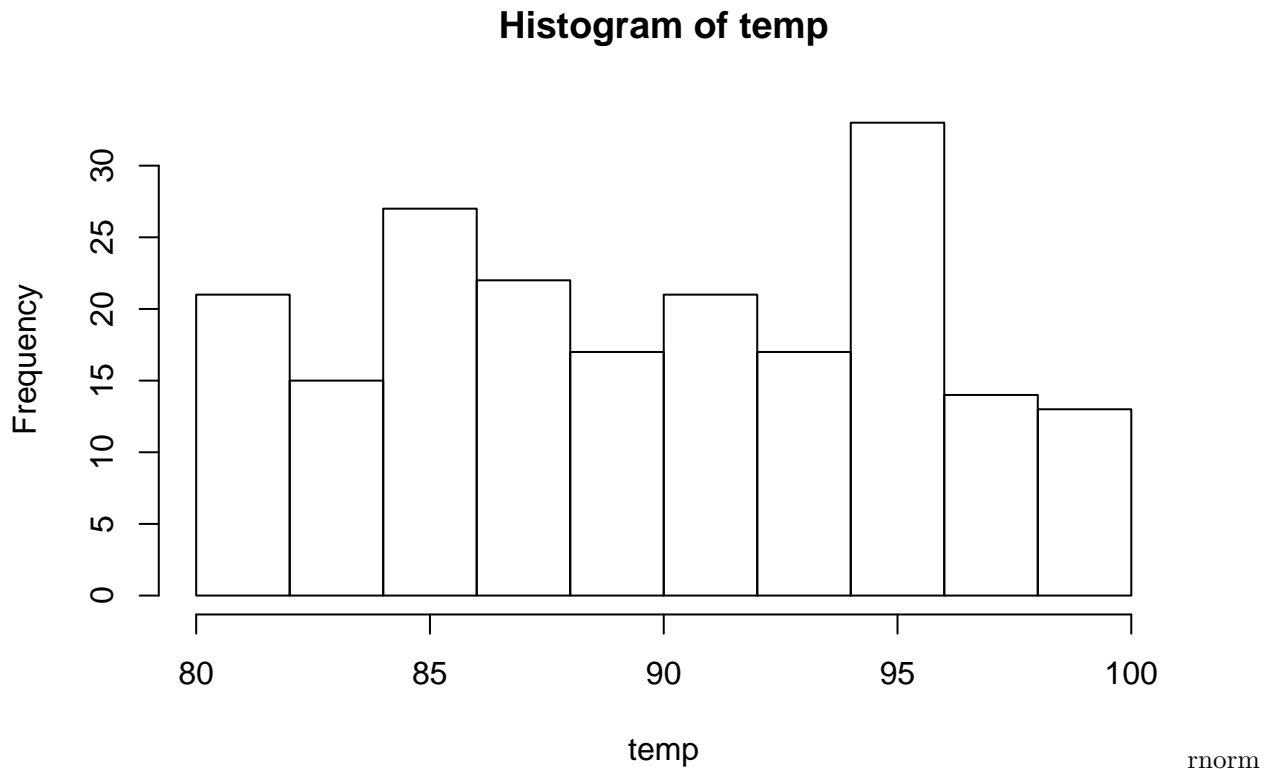
```
runif(200,80,100) # gera 200 valores que vão de um mínimo de 80 até um máximo de 100
```

```
## [1] 80.80485 99.33529 92.33512 93.76267 84.79765 91.95422 89.44312
## [8] 92.74013 98.50015 98.00227 99.51293 80.32075 89.30781 97.66968
## [15] 88.16295 95.50054 94.58753 83.37712 89.20889 87.11982 92.76406
## [22] 99.07759 97.26286 80.16995 87.67566 85.33465 81.42354 90.70265
## [29] 94.92362 94.87479 97.37631 94.75199 82.81590 92.48305 99.40479
## [36] 98.56597 92.34922 82.42402 96.58534 94.94930 96.91671 86.00248
## [43] 86.23256 91.97557 99.71200 95.61648 88.45721 95.76839 94.92371
## [50] 96.57338 86.08831 92.50982 85.89608 80.04148 92.79142 86.06121
## [57] 92.71534 84.15385 99.65836 81.93567 90.68484 81.75932 97.40334
## [64] 95.19540 87.96606 82.41996 80.59291 85.43628 82.61103 85.04545
## [71] 84.41059 96.77943 96.05409 84.49127 94.89464 94.34221 94.70682
## [78] 88.44678 84.16818 82.38116 96.47838 96.18471 84.36906 96.94512
## [85] 83.99889 80.93765 98.92134 90.08448 92.39755 83.11648 87.36146
## [92] 99.82820 90.89309 85.52363 82.72096 81.67199 89.44327 87.48837
## [99] 88.44301 98.53803 99.05008 91.51738 85.35204 93.80999 91.32652
## [106] 81.67454 90.97616 90.04492 94.23434 85.32264 95.12889 98.56625
## [113] 92.20393 87.44812 83.80522 97.81554 92.90767 99.71090 81.72145
## [120] 81.32916 88.02167 93.39759 92.46212 93.41471 86.69800 99.35609
## [127] 83.65526 85.80480 80.83177 98.50813 81.11265 88.37444 95.53269
## [134] 90.21780 80.74258 97.16832 99.21220 88.92490 89.27994 83.62531
```

```
## [141] 82.10364 91.98956 93.66340 96.80133 83.51402 88.37050 90.82869
## [148] 80.62918 93.41374 82.49270 81.40193 97.03987 88.92652 95.59544
## [155] 81.54795 98.05173 85.77475 96.30630 91.67502 83.24963 87.21579
## [162] 95.63616 80.74499 83.04339 81.54956 89.40181 89.57453 96.11980
## [169] 80.57688 94.51788 83.23494 92.61749 93.16525 80.68270 90.46231
## [176] 87.97275 98.69406 89.53406 80.96254 84.04375 93.60195 80.38573
## [183] 83.64781 87.44366 95.85487 94.26372 99.08094 98.13475 84.01337
## [190] 97.15959 93.73693 99.94233 97.98177 81.45379 93.72629 96.60136
## [197] 93.90411 83.55189 84.26580 85.69085
```

```
temp<-runif(200,80,100)
```

```
hist(temp) # Faz um histograma de frequências dos valores
```



(Gerar dados aleatórios com distribuição normal)

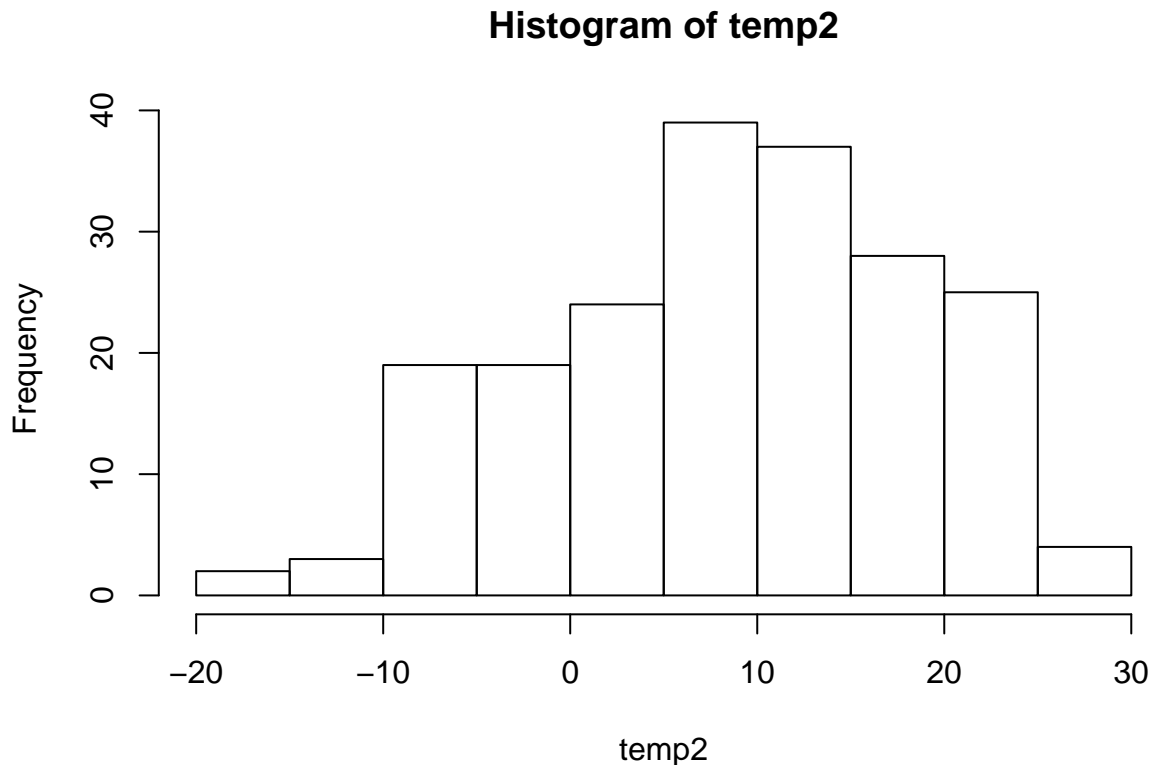
`rnorm(n, mean=0, sd=1)` # gera n valores com distribuição uniforme, com média 0 e desvio padrão 1.

```
rnorm(200,0,1) # 200 valores com média 0 e desvio padrão 1
```

```
## [1] 0.02612159 -0.77318349 0.43246300 0.32959423 0.08966190
## [6] 1.74130854 -2.14484051 -1.25560219 -0.16147979 -1.09557142
## [11] 0.85222347 0.01047381 -1.01961442 -0.52079985 1.81536882
## [16] -0.68022016 0.50453975 -0.66606674 0.78447047 -0.76188860
## [21] -0.38270744 -1.36602247 -1.06133851 0.24123412 -0.27234717
## [26] 2.54652112 -1.39376253 2.72947325 -1.66833865 1.78873714
## [31] -0.80701290 -0.86170625 0.30505438 -1.14108290 -0.56703616
## [36] -0.04488842 0.78178636 0.88258449 0.44027724 -0.31324703
## [41] 0.66714235 1.94748471 0.49716149 0.39084612 -0.38109581
## [46] -1.58471526 0.74561183 0.66780058 1.96885729 0.03647606
## [51] 0.82685828 -1.91582369 -0.59394782 0.20033885 -0.15805471
## [56] -0.23094259 0.31334661 -0.68332039 -0.26485915 -1.73854819
## [61] -0.16904856 1.03716496 -0.16070636 1.27860721 0.28200323
```

```
## [66] -0.93584914  0.61119821 -1.18791932  0.88000395 -0.35565806
## [71]  0.86986421 -0.10448968 -1.55446335 -1.85306474 -0.99209728
## [76] -0.92265051 -0.08336320 -0.09629422  2.51837294 -2.16116946
## [81]  0.61600394  0.09619309 -0.25319887  0.34153589  0.03454265
## [86] -0.46738820 -0.16938963  1.10722618  1.95804482  0.38644457
## [91]  1.22685481  0.15370231  0.24667389  1.17145271  0.46630568
## [96] -1.58212806  1.22310604  2.06542374  1.30650912 -1.09562007
## [101]  0.29545749  0.44003514 -0.49912182 -1.23350819 -1.01519772
## [106] -0.69811308 -1.00004512  1.86731065 -0.08787564  1.73764256
## [111] -0.77574232  0.24370362 -1.56692083 -1.40809085 -0.53052808
## [116] -0.95971858  0.90993939  0.61867021 -1.88595623  0.79397000
## [121] -0.56730998  0.36868508  0.12332668 -0.24934373 -0.30684857
## [126]  2.29368797  0.53952000  0.87415787  2.02980599 -0.64048724
## [131]  1.01141813  0.79323270  0.44102527 -0.31203056 -0.60292426
## [136]  0.32417964  0.10003129  1.65888953 -0.55437494 -0.19279112
## [141] -1.04931250  1.10234862 -2.81887995 -0.95897533  1.27953037
## [146] -0.63763232 -0.47219547  3.27915496  1.66450402 -0.53105229
## [151] -2.07483052  0.32866965  1.26566528 -0.47493906 -1.57846914
## [156] -3.31063842 -1.49570978 -0.22994650 -0.23998302  0.18755850
## [161]  0.47651121 -0.92719263 -0.79247919  0.69329008  1.12057762
## [166]  0.27384236 -1.62400057  0.05063985  0.43700241  1.24201537
## [171] -0.27588642  1.15776553  1.60785911  0.75757187 -0.48130819
## [176]  0.95703138 -0.12808341  0.13197318 -0.01149562  0.88194905
## [181]  2.32554085  0.06010820  0.17777131 -1.27691503 -0.89043529
## [186]  0.56314356 -0.01717095  1.93578818 -0.06382317  1.46613659
## [191]  1.20285349 -0.26825022  0.83944976 -0.72136436  0.10803116
## [196]  1.74739089 -0.21814851 -0.99919436  0.48896996  0.74409738
```

```
temp2<-rnorm(200,8,10) # 200 valores com média 8 e desvio padrão 10
hist(temp2) # Faz um histograma de frequências dos valores
```



Veja o



help da função ?Distributions para conhecer outras formas de gerar dados aleatórios com diferentes distribuições.

### Fazer amostras aleatórias

A função sample A função sample é utilizada para realizar amostras aleatórias e funciona assim: sample(x, size=1, replace = FALSE) onde x é o conjunto de dados do qual as amostras serão retiradas, size é o número de amostras e replace é onde você indica se a amostra deve ser feita com reposição (TRUE) ou sem reposição (FALSE).

```
sample(1:10,5) # tira 5 amostras com valores entre 1 e 10
```

```
## [1] 2 6 1 9 8
```

Como não especificamos o argumento replace o padrão é considerar que a amostra é sem reposição (= FALSE).

sample(1:10,15) # erro, amostra maior que o conjunto de valores, temos 10 valores (1 a 10)

portanto não é possível retirar 15 valores sem repetir nenhum!

```
sample(1:10,15,replace=TRUE) # agora sim!
```

```
## [1] 7 9 1 8 7 2 1 6 7 4 5 8 7 9 4
```

Com a função sample nós podemos criar vários processos de amostragem aleatória. Por exemplo, vamos criar uma moeda e “jogá-la” para ver quantas caras e quantas coroas saem em 10 jogadas.

```
moeda<-c("CARA","COROA") # primeiro criamos a moeda
```

```
sample(moeda,10)
```

ops! Esqueci de colocar replace=TRUE

```
sample(moeda,10,replace=TRUE) # agora sim
```

```
## [1] "COROA" "CARA" "CARA" "CARA" "COROA" "COROA" "COROA" "COROA"
```

```
## [9] "COROA" "CARA"
```