Getting Started

-Free and Open Source
-Multiplatform
*Utilizarei o símbolo > para indicar o input do console e >>> para o output
-As funções básicas são o R base, o restante são adicionados pela comunidade com CRAN ou github.
-É possível salvar os scripts para serem usados depois.

Installing Packages

install.packages("namepackage")

Como instalar um pacote:
install.packages("dslabs") //instalando o pacote dslabs
após instalar:
library (dslabs) //carregando o pacote dslabs

Para instalar mais de um pacote:
install.packages(**c**("pacote1", "pacote2"))

Também é possível utilizar o botão **tools - > install packages**

*Portanto, é importante ter um script que instale todos os pacotes que precisa, pois caso seja necessário reinstalar ou instalar uma nova versão do R, será necessário instalar os pacotes novamente.

*You can add the option dependencies = TRUE, which tells R to install the other things that are necessary for the package or packages to run smoothly. Otherwise, you may need to install additional packages to unlock the full functionality of a package.

Running Commands While Editing Scripts

Color and identation are automatically added in R studio.
O editor Rstudio também nos ajuda a testar nosso código enquanto editamos os scripts.
As primeiras linha de código em R geralmente são dedicadas a carregar as bibliotecas que utilizaremos.

Exemplo. (mostrar um gráfico de assassinatos vs população total)

```
library(tidyverse)
library(dslabs)
data(murders)

murders %>%
        ggplot(aes(population, total, label = abb, color = region)) +
        geom_label()
```

*Para testar apenas uma linha por vez se utilizar Ctrl-Enter
*Para testar o código todo Ctrl-Shift-Enter

Objects

Assignar variáveis:
a <- -1
b <- 2 //ou utilizar o simbolo = mas não é recomendado fazer isto

Para ver o **valor de um variável**:
>*a*
>>>1

Print:
print()
ex.
> *print(a)*
>>> 1

Ver as **variáveis salvas no workspace**:
ls()
ex.
>*ls()*
>>>"a" "b" "c"

Code: solving the equation $x^2 + x - 1 = 0$

```
# assigning values to variables
a <-1
b <-1
c <--1

# solving the quadratic equation
(-b + sqrt(b^2 - 4*a*c))/(2*a)
(-b - sqrt(b^2 - 4*a*c))/(2*a)
```

Functions

Funções:
ex. *log()*
>log(8)
caso digite log sem parêntese, a IDE mostrará o código de como usar o log

>>> 2.079
>exp(1) // e^1
>>>2.718
>log(2.718)
>>>1
>log(exp(1))
>>>1

Nested Functions:
Usar a função como argumento para outra função.
Ex.
log(exp(1))

Help system:
ex.
help("log") ou *?log*
ex.
help("+")
?"+" //note que nesse caso é necessário aspas para o operador
>...shows the help file… //arquivo mostra como a função log funciona
>help("+")
Para ver os **argumentos da função**:
> args(log)
>>> function (x, base=exp(1)) //quando se tem o igual temos o valor de default do argumento
Null
Como descobrimos como os argumentos funcionam, se quisermos o log de 8 na base 2:
>log(8,base=2)
>>>3

Ver os **nomes de objetos**:
data()
ex.
> pi
>>>3.1415
Co2
>>>Console mostra a data de pre-built objetcs para Co2

Variable names in R

Starts with a letter
Can't contain spaces
ex.
solution_1
solution_2

Comments:
now commenting

Data Types

class():
ajuda a determinar a classe de um objeto
ex.
> a <- 2
> class(a)
>>> numerical

Integer class
L
ex.
> class(3L)
>>> integer

Data Frame:
*Nele, pode-se combinar vários formatos de dados em um único objeto.
*Pode ser imaginado como tabelas.
>libray(dslabs)
>data("murders") //utiliza-se para acesar o data set
>class(murders)
>>> "data.frame"
str() //mostra a estrutura de um objeto
ex.
> str(murders)
>>> 'data.frame': 51 observations, 51 rows, and five variables. Além do nome de colunas etc.
>head(murders) //mostra as primeiras 6 linhas de murders

Acessar nome das colunas:
>names(murders)
>>> Mostrará os nomes das colunas

Acessar as variáveis das colunas:
> murders$population //mostra a coluna population
Outra maneira é utilizando-se [[
Ex.
> *b <- murders[["population"]]*

Vetores:
Valores da tabela

length:
>pop <- murders$population

>length(pop)
>>>51 //tamanho de populations em murders (ou número de vetores)

Characteres Vectors:
> a <- 1
> a
>>> 1
Caso queira saber o **valor da string** a:
>*"a"*
>>>"a"
ex.
> class(murders$state)
>>> "character"

Logical Vectors:
TRUE or FALSE
ex.
> z <- 3 == 2
> z
>>> false
> class(z)
>>> logical

Comparar se variáveis são iguais:
> a <- 1
> b <- 1
> identical(a, b)
>>> TRUE

Factor: //útil para guardar data categórica
> class(murders$region) //cada região é uma categoria categórica
>>> "factor" //*** não confundir com characters
Para ver as categorias:
levels()
> levels(murders$region)
>>> "North" "South" "west" "North Central"
*salvar "categoric datas" desse modo, é mais eficiente para a memória.

Vectors

-Datasets complexos podem ser quebrados em componentes vetores. //cada coluna pode ser um vetor.

Concatenete

c

ex.

> *codes <- c(380, 124, 818)* //concatena os dados em um vetor codes

Character vector

ex.

country <- (“italy”,“canada”,”egypt”) //caso não use as aspas, R procurará por variáveis //com esses nomes.

ex. nomeando as entradas do vetor:

> *codes <- (italy=380,canada=124,egypt=818)* //pode-se usar aspas, ex. codes<-(“ab”=3)

```
> codes
 italy canada egypt
   380    124   818
```

> *class(codes)*

>>> *numeric*

Names para assimilar nomes às entradas do vetor:

> *codes <- c(380,124,818)*

> *country <- c(“italy”,”canada”,”egypt”)*

> *names(code) <- country*

```
> codes
 italy canada egypt
   380    124   818
```

Função que gera sequências

seq

ex.

> *seq(1,10)* //ou 1:10

ex. pulando em 2

>>> *1 2 3 4 5 6 7 8 9 10*

> *seq(1,10,2)*

>>> *1 3 5 7 9*

-É possível gerar sequências com lenght.out

> *x <- seq(0, 100, lenght.out = 5)*

>>> 0 25 50 75 100

Subsetting

-Permite acessar partes específicas de um vetor

[ ]

ex.

> *codes [2]*

>>> canada
124
ex.
> *codes[c(1,3)]* //ou codes[c("italy","egypt")]
>>> italy egypt //codes[1:3] dará os 3 primeiros vetores
380 818
ex.
> *codes["canada"]*
>>> canada
124

Vector Coercion

-Quando uma entrada não coincide com o esperado, R tenta adivinhar o que queremos dizer antes de apontar um erro.
ex.
> *x <- c(1,"canada",e)*
> *x*
>>> "1" "canada" "3" //ou seja, ele converteu 1 e 3 em strings.
> *class(x)*
>>> "character"

Forçar Coercão

as.tipo //as.character() as.numeric()
ex.
> *x <- 1:5*
> y <- as.character(x)
> *y*
>>> "1" "2" "3" "4" "5"

Missing Data

NA //not available
-Quando R falha em tentar coerção
ex.
> *x <- c("1","b","3")*
> *as.numeric(x)*
>>> 1 NA 3
is.na()
retorna quais entradas são NA
> *is.na(na_example)*

Sorting

sort()
-Ordena os vetores em ordem crescente
ex.
> libray(dslabs)
> data(murders)
> sort(murders$total)
>>> 2 4 5 5 7 8 11 ….

order()
-Retorna o indice do vetor que o ordena
ex.
> x <- c(31, 4, 15, 92, 65)
> order(x)
>>> 2 3 1 5 4 //[2] é o menor número, [3] é o segundo menor e assim em diante
ex.2 é possível utilizar o order com outras funções
sort()
> sort(x)
> index <- order(x)
> x[index]
>>> 4 15 31 65 92
note que:
> index <- order(murders$abb) *//*retorna as abreviações em ordem alfabetica
é diferente de:
> index <- order(murders$total)
> murders$abb[index] *//retorna as abreviações em ordem crescente de assassinatos por estado*

max()
-mostra o maior valor entre os índices
ex.
> max(murders$total)
>>> 1257
which.max() //também há o min() e which.min()
-mostra o índice de onde está o maior valor
ex.
> i_max <- which.max(murder$total)
>>> 5
> murders$state[i_max]
>>> “California”

rank()
-diz o ranking em ordem crescente de cada índice de x
> x <- c(31, 4, 15, 92, 65)
> rank(x)
>>> 3 1 2 5 4 //neste exemplo, 31 é o terceiro menor número
-Caso se queira do maior para o menor:
> rank (-x)

Data Frame

data.frame()
ex.
> temp <- c(35, 88, 42, 84, 81, 30)
> city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro", "San Juan", "Toronto")
> city_temps <- data.frame(name = city, temperature = temp)

-Por padrão, o data frame transforma os characters em factor. Para evitar isso:

grades <- data.frame(names=c(“John”,”Juan”,”Jeans”,”Yao”),

exam_1 = c(95, 60, 80, 85),

exam_2 = c(90, 85, 85, 95),

stringsAsFactors = False)

Média

mean()
ex.
> x <- 1:100
> mean(x)
>>> 50.5

Vector Arithmetic

> heights <- c(69,62,66,70,70,73,67,73,67,70) //em inches
*> heights * 2.54*
>>> 175.26 157.48 167.64 177.80 177.80…. //agora mudados para cm

Se soubermos que a média é de 69, podemos tirar 69 de todos os vetores e saber a diferença deles para a média:

> heights - 69
>>> 0 -7 -3 1 1 4 -2 4 -2 1

Se tivermos 2 vetores do mesmo comprimento, podemos somar entrada por entrada, o mesmo vale para operações como “*”, “-” e “/”.
> heights_1 <- c(1,2,3,4,5)
> heights_2 <- c(2,3,4,5,6)
> heights_1 + heights_2
>>> 3 5 7 9 11

Extra:
Obter a taxa de morticidade:
*> murder_rate <- murders$total / murders$population * 100000*
ver os dados em ordem decrescente
> murders$state[order(murder_rate, decreasing = TRUE)]

Indexing

Logical operators to index vectors

Se compararmos um vetor com um número, ele irá comparar todas as entradas.

Caso queira encontrar o índice de assassinatos menor ou igual a 0.71:
> index <- murder_rate < = 0.71
> index
>>> FALSE FALSE FALSE TRUE TRUE FALSE ....

Para mostrar os valores verdadeiros:
> muders$state[index]
>>> “hawaii” “iowa” “new hampshire” …

Para saber quantos são os valores verdadeiros:
> sum(index)
>>> 5

Operadores:
< <= > >= == != ! (not) | (or) & (and)

Vetores que satisfazem duas condições:
> index <- safe & west
> murder$state[index]
>>> “hawaii” “idaho” “oregon” “utah” “wyoming”

Indexing function

which()
-Nós as entradas do vetor que são verdadeiras.
ex.
> x <- c(FALSE, TRUE, FALSE, TRUE, TRUE, FALSE)
> which(x)
>>> 2 4 5
ex2.
> index <- which(murders$state == "Massachusetts")
> index
>>> index
> murder_rate[index]
>>> 1.802
ou simplesmente:
> index <- murder$state == "Massachusetts")
>>> 1.802
porém, do primeiro modo, o objeto index é muito menor.

match()
> index <- match(c("New York", "Florida","Texas"),muders$state)
> index
>>> 33 10 44 //index que correspondem aos estados das cidades

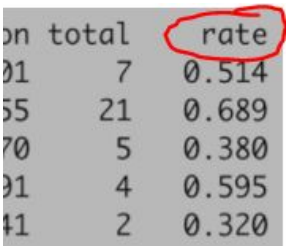
%in%
-Confere se os elementos de um vetor estão em outro.
ex.
> x <- c("a","b","c","d","e")
> y <- c("a","d","f")
> y %in% x
>>> TRUE, TRUE, FALSE
ex prático. (saber se os estados estão no vetor)
> c("Boston", "Dakota", Washington") %in% murders$state
>> FALSE, FALSE, TRUE

Basic Data Wrangling

dplyr //pacote para trabalhar com tabelas

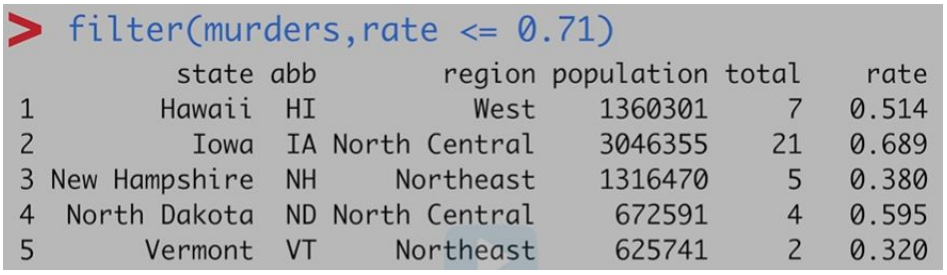
> library(dplyr)

Adicionar ou mudar uma coluna:
mutate()
-Usa o data frame como primeiro argumento e o nome e valor da variável como segundo.
ex.
> *murders <- mutate(murders, rate = total/population*100000)*



on	total	rate
01	7	0.514
55	21	0.689
70	5	0.380
01	4	0.595
41	2	0.320

Filtrar os dados em subconjuntos de linhas:
filter()
-"filter é um select para linhas"
-Toma a tabela de dados como primeiro argumento e a condicional como segundo.
ex.
> *filter(murders, rate <= 0.71)*



```
> filter(murders,rate <= 0.71)
          state abb        region population total    rate
1        Hawaii  HI          West    1360301     7   0.514
2          Iowa  IA North Central    3046355    21   0.689
3 New Hampshire  NH     Northeast    1316470     5   0.380
4  North Dakota  ND North Central     672591     4   0.595
5       Vermont  VT     Northeast     625741     2   0.320
```

-filtrar resultados do top 5 do ranking.
> *rank <- -rank*
> *filter (murders, rank <=5)*
-ver os dados de mais de uma linha:
> *filter(murder, state %in% c("New York", "Texas")*

Para subdividir os dados em colunas específicas:
select()
> *new_table <- select(murders, state, region, rate)* //selecionará apenas estas 3 colunas

Remover linhas

!=
ex.

no_florida <- filter(murders, state != “Florida”)

Calcular número de linhas

nrow()
ex.
> nrow(no_florida)

Pipe operator

%>%
-Selecionar o que se quer dos dados e os filtrar por exemplo:
> murders %>% select(state, region, rate) %>% filter(rate <= 0.71)

Basic Plots

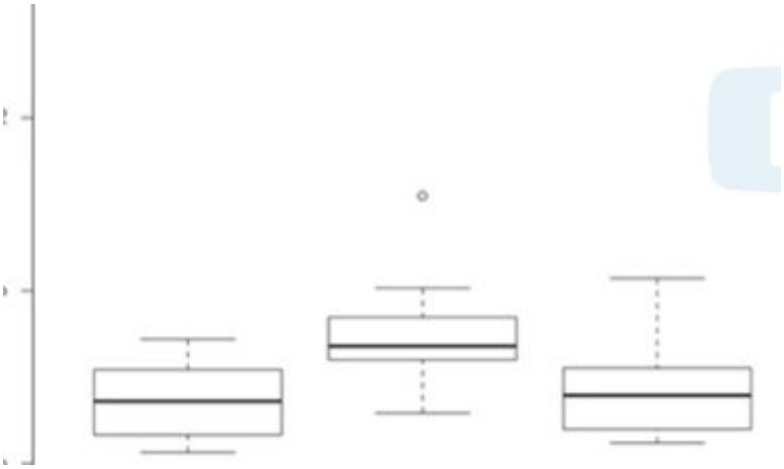
plot()
> population_in_millions <- murders$population/10^6
> total_gun_murders <- murders$total
> plot(population_in_millions, total_gun_murders)

Histogram

hist()
ex.
hist(murders$rate)

Boxplot

boxplot()
> boxplot(rate~region, data = murders)



Introdução da programação em R

Condicionais básicas

if-else:
ex.
> if(a!=0){
> print(1/a)
> } else{
> print("No reciprocal for 0.")
> }

ifelse:
-retorna um valor para true e outro caso seja false
ifelse()
ex.
> a <- 0
> ifelse(a > 0, 1/a, NA) //1/a para true e NA para false
ex. em vetores
> a <- c(0,1,2,-4,5)
> result <- ifelse(a > 0, 1/a, NA)
> result
>>> NA 1 0.5 NA 0.20
ex. *trocando valores inválidos(NA) do vetor por 0
> data(na_example)
> sum(is.na(na_example))
>>> 145
> no_nas <- ifelse(is.na(na_example), 0, na_example)
> sum(is.na(no_nas))
>>> 0

any and all

-any retorna true se existir pelo menos uma entrada true
-all retorna true se todas entradas forem true
any()
all()
ex.
> z <- (TRUE, TRUE, FALSE)
> any(z)
>>> TRUE

FUNCTIONS

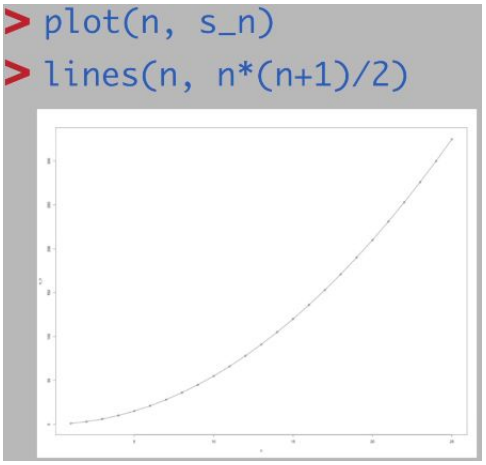
ex.
> avg <- function(x){ //podendo ter mais de um argumento
> s <- sum(x)
> n <- length(x)
> s/n
> }
> x <- 1:100
> avg(x)
>>> 50.5
Note que:
> identical (mean(x), avg(x)) //pois a função faz o mesmo de mean()
>>> True
*Também é necessário ficar atendo ao escopo da variável (global ou local)

FOR LOOPS

ex.
> m <- 25
> s_n <- vector(length = m) //cria um vetor vazio
> for(n in 1:m){
> s_n[n] <- compute_s_n(n)
> }

ex. de plot utilizando linhas para essa função for:

```
> plot(n, s_n)
> lines(n, n*(n+1)/2)
```



outras funções:

-funções que são utilizadas em R por serem mais eficazes que o for:
apply(x,y,z)
lapplay()
	resultado na forma de lista
sapply()
	resultado na forma de vetor
tapply()
mapply()
-outras são:
split()
cut()
quantile()
reduce()
identical()
unique()