

EDA

Danilo A C Souto

10/11/2020

Exploratory Data Analysis in R - EDA

Linguagem R

História

R foi construído tomando como referência a linguagem S desenvolvida em 1960 e 70 pelos pesquisadores da Bell Laboratories. Com foco em ser open source, Ross Ihaka e Robert Gentleman lançaram em 1990 na Universidade de Auckland na Nova Zelândia em 1990 sob licença pública GNU.

A sua popularidade cresceu bastante devido a sua flexibilidade para análise de dados e ferramentas gráficas poderosas.

Qualquer pessoa pode contribuir com pacotes do R e mais detalhes sobre os colaboradores principais R Core Team podem ser encontradas em <http://www.r-project.org/>.

Recentemente com o crescimento da importância dada computação voltada aos dados as empresas tem dado mais importância a linguagens focadas na análise de dados como o R. Outro ponto importante a ressaltar é que pelo fato de ser Open Source e possuir um número muito grande de pacotes disponíveis tornam os projetos de análises com um custo baixo, análises avançadas além de possuir fácil integração com diversos outros softwares como SPSS, Power BI, etc.

Instalando

Para instalar o R, deve-se começar instalando o r-base que é o pacote principal da linguagem R. Depois iremos instalar um ambiente de desenvolvimento. Neste curso iremos utilizar o RStudio, mas pode ser utilizado qualquer outro pois existem vários.

Instalando R

Será abordada a instalação em 3 sistemas: Windows, Linux Ubuntu e Mac OS

Instalando R no Windows

Vá na pasta onde fez o Download e execute o arquivo R-4.0.2.exe - Faça o processo de instalação como o de qualquer outro aplicativo [Next, Next, ..., Finish]

Instalando R no Ubuntu

Abra um terminal e digite

```
sudo apt update
sudo apt -y install r-base
sudo apt -y install r-base-dev
sudo apt -y install libcurl4-openssl-dev libssl-dev libxml2-dev
```

Instalando R no Mac OS

Similar como no Windows, mas escolha os pacotes para mac OS <https://cran.r-project.org/> - escolha R base - faça o download da versão de acordo com sua versão do sistema operacional. MacOS - o arquivo vai ter a extensão .pkg

Instalando R Studio

Como IDE iremos utilizar o RStudio, mas fique à vontade para utilizar outra IDE caso prefira.

Baixando Rstudio Windows

Vá para o link de download <https://rstudio.com/products/rstudio/download/#download> - Vá na pasta onde fez o Download e execute o arquivo. - Faça o processo de instalação como o de qualquer outro aplicativo [Next, Next, ..., Finish]

Baixando Rstudio Ubuntu

Vá para o link de download <https://rstudio.com/products/rstudio/download/#download>


OS	Download	Size	SHA-256
Windows 10/8/7	RStudio-1.2.5033.exe	149.83 MB	7fd3bc1b
macOS 10.12+	RStudio-1.2.5033.dmg	126.89 MB	b67c9875
Ubuntu 14/Debian 8	rstudio-1.2.5033-amd64.deb	96.18 MB	89dc2e22
Ubuntu 16	rstudio-1.2.5033-amd64.deb	104.14 MB	a1591ed7
Ubuntu 18/Debian 10	rstudio-1.2.5033-amd64.deb 	105.21 MB	08eaa295
Fedora 19/Red Hat 7	rstudio-1.2.5033-x86_64.rpm	120.23 MB	38cf43c6
Fedora 28/Red Hat 8	rstudio-1.2.5033-x86_64.rpm	120.87 MB	452bc0d0
Debian 9	rstudio-1.2.5033-amd64.deb	105.45 MB	27c59722
SLES/OpenSUSE 12	rstudio-1.2.5033-x86_64.rpm	98.87 MB	9c1e200c
OpenSUSE 15	rstudio-1.2.5033-x86_64.rpm	106.91 MB	98fd2258

Figure 1: Baixe o Rstudio

depois que o download concluir abra um terminal de comando do Ubuntu e vá para a pasta onde o arquivo foi salvo

```
cd Downloads
sudo apt update

sudo apt install r-base
sudo apt install r-base-dev
sudo apt install libcurl4-openssl-dev libssl-dev libxml2-dev

sudo dpkg -i rstudio-1.2.5033-amd64.deb
sudo apt -f install
```

Baixando Rstudio Mac OS

Vá para o link de download <https://rstudio.com/products/rstudio/download/#download>












OS	Download	Size	SHA-256
Windows 10/8/7	 RStudio-1.2.5033.exe	149.83 MB	7fd3bc1b
macOS 10.12+	 RStudio-1.2.5033.dmg 	126.89 MB	b67c9875
Ubuntu 14/Debian 8	 rstudio-1.2.5033-amd64.deb	96.18 MB	89dc2e22
Ubuntu 16	 rstudio-1.2.5033-amd64.deb	104.14 MB	a1591ed7
Ubuntu 18/Debian 10	 rstudio-1.2.5033-amd64.deb	105.21 MB	08eaa295
Fedora 19/Red Hat 7	 rstudio-1.2.5033-x86_64.rpm	120.23 MB	38cf43c6
Fedora 28/Red Hat 8	 rstudio-1.2.5033-x86_64.rpm	120.87 MB	452bc0d0
Debian 9	 rstudio-1.2.5033-amd64.deb	105.45 MB	27c59722
SLES/OpenSUSE 12	 rstudio-1.2.5033-x86_64.rpm	98.87 MB	9c1e200c
OpenSUSE 15	 rstudio-1.2.5033-x86_64.rpm	106.91 MB	98fd2258

Figure 2: Baix o Rstudio

Instalando pacotes/bibliotecas no R

Depois de terminada a instalação do RStudio, abra-o e instale alguns pacotes.

O R é composto de diversas bibliotecas úteis para a mais diversas funções. Elas estão disponíveis gratuitamente no cran-r e podem ser instaladas facilmente apenas utilizando o comando `install.packages("nome-da-biblioteca")` no próprio R

Exemplo: a biblioteca tidyverse é um conjunto grande de bibliotecas muito úteis. No console digite e confirme com conforme descrito abaixo:

```
install.packages("tidyverse")
```

O processo de instalação deve ser disparado procurando alguns pacotes e instalando.

Executando linha no RStudio

Coloque o cursor na linha que deseja executar e pressione Ctrl + R

```
ls()
```

Início

O R

Funções

Uma função é definida com ou sem argumentos

```
funcao(arg1, arg2, ...)
```

```
funcao(arg1, arg2)
```

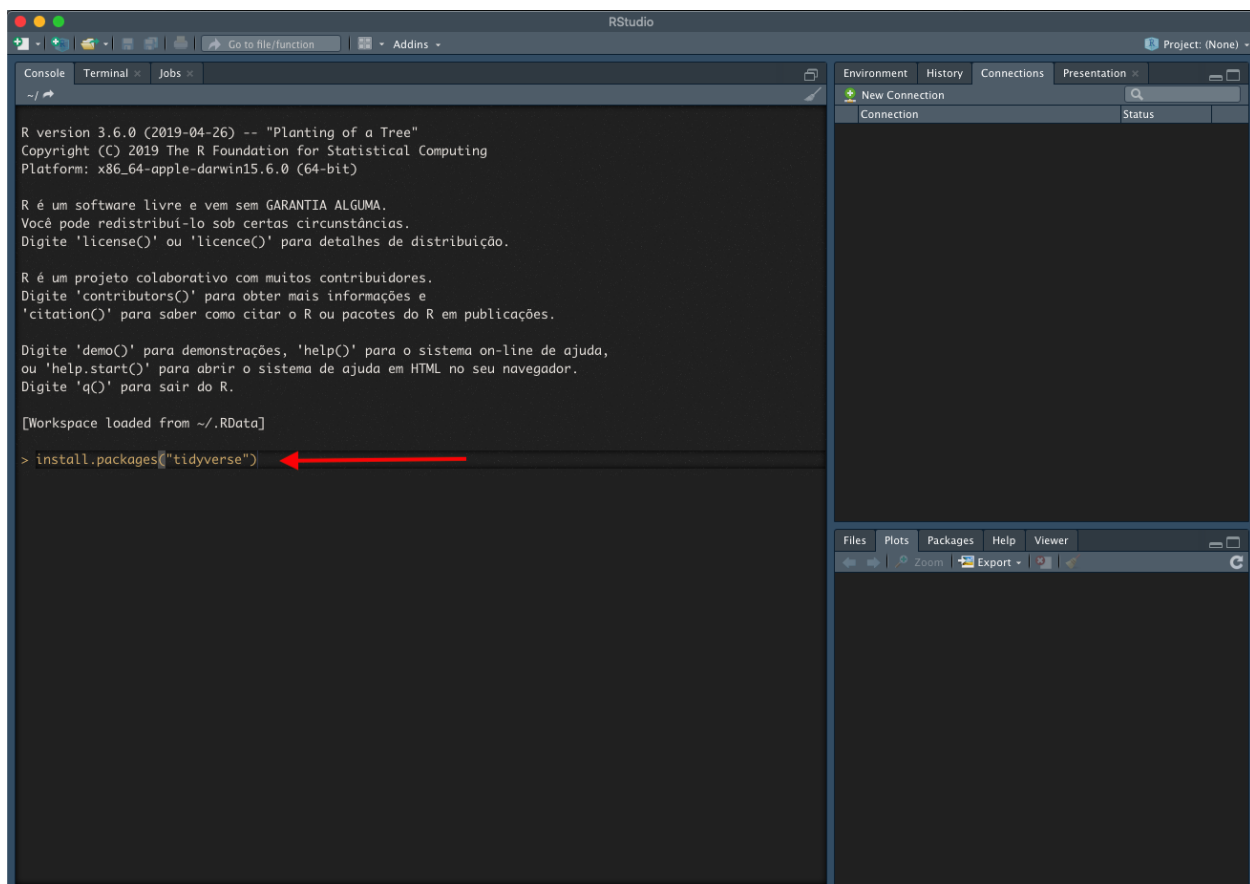


Figure 3: instalando pacotes

```
print("Olá Mundo")
```

Algumas funções podem possuir valores de argumentos pré definidos

```
runif(100)
```

```
runif(n = 100, min = 10, max = 20)
```

Ajuda

Uma função muito importante do R é a função `help`. Ela exibe a documentação da função onde diz como usar e os argumentos obrigatórios e opcionais da função

```
help(runif)
```

Outra forma de acessar a ajuda é usar o ?

```
?runif
```

A documentação contém vários campos

- Description
- Usage
- Arguments
- Details
- Value
- References
- See Also
- Examples

Portanto não apenas exibe a descrição como documentação de referência para o uso da função bem como alguns exemplos

Outra função de ajuda importante é o `apropos()` que retorna as funções que contêm a palavra procurada

```
apropos("runif")
```

Carregar Bibliotecas e Instalar pacotes

Para carregar bibliotecas a serem utilizadas no R

```
library(agricolae)
```

Agora, se uma biblioteca necessária não estiver instalada, você pode utilizar o comando

```
install.packages("ggplot2")
```

Este comando busca o pacote nos mirrors do CRAN e o instala. Bastando apenas utilizar o comando `library(pacote)` para carregar e utilizar as funções do pacote.

Arrumando a casa

O comando `ls()` lista todos os objetos disponíveis na sessão e o comando `rm()` remove um objeto.

```
a <- 1
b <- 2
c <- a + b
ls()
rm(a)
ls()
```

Para remover todos os objetos da sessão use `rm(list = ls())`

Funções

Uma função sem Argumento. Define a função assim

```
funcao <- function(){  
  print("Esta é minha Função")  
}
```

E a invoca desta maneira

```
funcao()
```

Agora, funções podem possuir um ou mais argumentos e ter retorno

```
soma <- function(a,b){  
  return(a+b)  
}
```

```
soma(1,2)
```

Outra função

```
quadrado <- function(x){  
  return(x^2)  
}  
quadrado(2)
```

Tipos básicos

- double
- integer
- numeric
- character
- logical
- complex
- row

Estrutura lógica

```
if(condicao){  
  #código a ser executado se a "condição" lógica for válida  
  print("a condição é verdadeira")  
}
```

```
variavel <- 1  
if(variavel == 1){  
  print("o valor da variável é 1")  
}
```

```
## [1] "o valor da variável é 1"
```

```
variavel <- 1  
if(variavel > 1){  
  print("o valor da variável é maior que 1")  
}else{  
  print("o valor da variável é menor ou igual a 1")  
}
```

Else

```
## [1] "o valor da variável é menor ou igual a 1"
```

Vetores

Utiliza a função `c()` para criar vetores

```
vet <- c(0,1,2,3,4,5,6,7,8,9)
```

A função `length()` retorna a quantidade de elementos

```
length(vet)
```

A função `seq()` cria uma sequência de dígitos

```
sequencia <- seq(from=-1, to= 1, by = 0.1)
```

Laços de repetição

while Os laços de repetição executam trechos de código de maneira repetida até que o critério seja satisfeito

```
variavel <- 1
while(variavel < 10){
  # este trecho será executado enquanto o valor da variável for menor que 10
  print(paste("Valor:", variavel))
  variavel <- variavel + 1
}
```

```
## [1] "Valor: 1"
## [1] "Valor: 2"
## [1] "Valor: 3"
## [1] "Valor: 4"
## [1] "Valor: 5"
## [1] "Valor: 6"
## [1] "Valor: 7"
## [1] "Valor: 8"
## [1] "Valor: 9"
```

```
print("Saiu do laço de repetição")
```

```
## [1] "Saiu do laço de repetição"
```

```
print(paste("Valor:", variavel))
```

```
## [1] "Valor: 10"
```

Percorrendo vetores

```
vetor <- LETTERS
i <- 1
while(i <= length(vetor)) {
  print(vetor[i])
  i <- i + 1
}
```

```
## [1] "A"
## [1] "B"
## [1] "C"
## [1] "D"
## [1] "E"
## [1] "F"
## [1] "G"
```

```
## [1] "H"
## [1] "I"
## [1] "J"
## [1] "K"
## [1] "L"
## [1] "M"
## [1] "N"
## [1] "O"
## [1] "P"
## [1] "Q"
## [1] "R"
## [1] "S"
## [1] "T"
## [1] "U"
## [1] "V"
## [1] "W"
## [1] "X"
## [1] "Y"
## [1] "Z"
```

FOR Mas como pode ver, ter que ficar manipulando os índices diretamente e a condição de parada pode ser trabalhoso. Para este caso existe outra estrutura muito mais interessante. **FOR**

```
vetor <- LETTERS
for (variavel in vetor) {
  print(variavel)
}
```

```
## [1] "A"
## [1] "B"
## [1] "C"
## [1] "D"
## [1] "E"
## [1] "F"
## [1] "G"
## [1] "H"
## [1] "I"
## [1] "J"
## [1] "K"
## [1] "L"
## [1] "M"
## [1] "N"
## [1] "O"
## [1] "P"
## [1] "Q"
## [1] "R"
## [1] "S"
## [1] "T"
## [1] "U"
## [1] "V"
## [1] "W"
## [1] "X"
## [1] "Y"
## [1] "Z"
```


Como pode ver o critério de parada é o próprio tamanho dos vetores e o valor pode ser acessado diretamente sem ter que trabalhar com o índice diretamente.

Operações com vetores

Aqui o R começa a se diferenciar das outras linguagens pois ele realiza operações vetoriais com facilidade.

Ele pode multiplicar um vetor por um número sem a necessidade de um laço de repetição `for`

```
vet <- c(0,1,2,3,4,5,6,7,8,9)
vet * 2
q
```

Aplicar uma função sobre um vetor também é possível. Lembra da nossa função de soma e quadrado? Sem nenhuma modificação é possível fazer operações com vetores

```
soma(vet, vet)
```

```
quadrado(vet)
```

Multiplicar vetores e somar vetores também é possível, mas tome cuidado, pois vetores de tamanhos diferentes podem gerar resultados inesperados devido a reciclagem.

```
vet + vet
vet * vet
vet * c(1,2)
```

**** Um vetor só pode conter elementos do mesmo tipo****

Mudando o tipo do vetor/objeto

```
as.character(vet)
```

```
as.numeric(c("1","2","3"))
```

Matriz

São estruturas bidimensionais

Assim como os vetores podem conter somente um tipo de informação

```
matriz <- matrix(1:12, nrow = 3, ncol = 4)
matriz
matriz <- matrix(1:12, nrow = 3, ncol = 4, byrow = TRUE)
dim(matriz)
```

função `cbind()` A função `cbind()` adiciona uma nova coluna na matriz

```
matriz <- matrix(1:12, nrow = 3, ncol = 4, byrow = TRUE)
dim(matriz)
matriz
matriz <- cbind(matriz, c(5,5,5))
matriz
dim(matriz)
```

A função `rbind()` adiciona uma nova linha na matriz

```
matriz <- matrix(1:12, nrow = 3, ncol = 4, byrow = TRUE)
dim(matriz)
matriz
matriz <- rbind(matriz, c(5,5,5,5))
matriz
```

```
dim(matriz)
```

Acessar um elemento de uma matriz

```
matriz[2,2]  
matriz[2,]  
matriz[,1]
```

Listas

A forma mais simples de descrever uma lista talvez seja que ela seja equivalente ao vetor mas pode conter tipos diferentes de dados

```
lista <- list(1,2,"a",1:20)  
lista  
lista[[1]]  
lista[[3]]  
lista[[4]][19]  
length(lista)
```

Data Frame

Uma das estruturas de dados mais interessantes do R. É uma estrutura de duas dimensões, listas de listas, mas com a seguinte obrigatoriedade: **Elas devem conter o mesmo comprimento**

```
df <- data.frame(nome = c("Luis", "João", "Arnaldo", "Ana"),  
                 sexo = c("M", "M", "M", "F"),  
                 idade = c(12, 34, 35, 18)  
                 )  
df
```

```
##      nome sexo idade  
## 1    Luis    M    12  
## 2   João    M    34  
## 3 Arnaldo    M    35  
## 4     Ana    F    18
```

```
dim(df)  
length(df)  
ncol(df)  
nrow(df)
```

```
colnames()
```

As funções `rbind()` e `cbind()` também funcionam com Data Frame para adicionar uma nova linha e coluna respectivamente.

função `str()` exibe a estrutura dos dados.

```
str(df)
```

Já a função `summary()` mostra um resumo das informações contidas

```
summary(df)
```

Em alguns casos pode ser conveniente, ou até exigido, que um data frame seja convertido para uma matriz

```
as.matrix(df)
```

Para exibir os nomes de colunas ou linhas temos a função

```
colnames(df)
rownames(df)
```

Também pode ser útil alterar o nome de uma coluna

```
colnames(df) <- c("nome", "genero", "idade")
colnames(df)
```

Dados

É possível entrar com dados no R através de várias formas

Função `scan()`

A função `scan()` le dados digitados pelo usuário para variável até que seja apertada a tecla ENTER duas vezes.

```
x <- scan()
x
```

Importar um arquivo cvs

Para importar arquivos temos funções específicas como o `read.table()` e mais especificamente o `read.csv()`.

```
dados <- read.table(file.choose(), header = TRUE, sep = ";", quote = "\"", dec = ",")
dados <- read.csv("caminho para o arquivo", header = TRUE, stringsAsFactors = FALSE)
str(dados)
head(dados)
tail(dados)
```

Também é possível importar dados no formato excel. Para isso temos o pacote `gdata`

```
library(gdata)
read.xls("arquivo.xls", sheet = "Plan1", header = TRUE, dec = ",")
```

Gravar Dados

Os dados gerados e trabalhos no R podem ser exportados para um arquivo facilmente utilizando

```
write.table(dados, file = "caminho/arquivo.csv")
write.table(dados, file = "caminho/arquivo.csv", sep = ";", dec = ",", row.names = FALSE)
write.csv2(dados, file = "caminho/arquivo.csv", row.names = FALSE)
```

A linguagem R também permite salvar objetos, funções e dados obtidos durante o trabalho para que sejam utilizados posteriormente

`dput()` salva um objeto em formato textual.

```
quadrado
dput(quadrado)
dput(quadrado, file = "qq.R")
qq <- dget(file = "qq.R")
```

A função `dump()` permite salvar mais de 1 dado e a função `source()` recupera todos.

```
quadrado
x
dump(c("quadrado", "x"), file = "dados.R")
rm( quadrado)
rm( x)
source("dados.R")
```

Já `save()` e `save.image()` salvam os objetos em formato binário. `save.image()` salva todos os objetos na sessão

```
save(iris, file = "iris.rda")
load("iris.rda")
```

```
save.image(file = "area_trabalho.RData")
load("area_trabalho.RData")
```

Selecionando Valores

Ao trabalhar com Vetores, Matrizes e Data Frames, é possível selecionar os valores desejados, filtrar intervalos e remover dados nulos, (NA no R)

```
set.seed(123)
x <- runif(20, min = 10, max = 20)
x
```

```
## [1] 12.87578 17.88305 14.08977 18.83017 19.40467 10.45556 15.28105 18.92419
## [9] 15.51435 14.56615 19.56833 14.53334 16.77571 15.72633 11.02925 18.99825
## [17] 12.46088 10.42060 13.27921 19.54504
```

```
# [1] 12.87578 17.88305 14.08977 18.83017 19.40467 10.45556 15.28105 18.92419
# [9] 15.51435 14.56615 19.56833 14.53334 16.77571 15.72633 11.02925 18.99825
# [17] 12.46088 10.42060 13.27921 19.54504
x[3]
```

```
## [1] 14.08977
# [1] 14.08977
```

```
x[x > 17]
```

```
## [1] 17.88305 18.83017 19.40467 18.92419 19.56833 18.99825 19.54504
# [1] 17.88305 18.83017 19.40467 18.92419 19.56833 18.99825 19.54504
```

```
x[x > 15 & x < 17]
```

```
## [1] 15.28105 15.51435 16.77571 15.72633
#[1] 15.28105 15.51435 16.77571 15.72633
```

```
x <- 1:10
x
```

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

```
l <- LETTERS[1:10]
l
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
x[l == "D"]
```

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

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

```
x[1 %in% c("A","E","I")]
```

```
## [1] 1 5 9
```

```
# [1] 1 5 9
```

A função which retorna os índices dos elementos os elementos

```
which(1 %in% c("C","B"))
```

```
## [1] 2 3
```

Seleção em Data Frames

Com o nosso Data Frame

```
df
```

```
##      nome sexo idade
## 1    Luis    M    12
## 2   João    M    34
## 3 Arnaldo    M    35
## 4     Ana    F    18
```

Podemos selecionar apenas linhas

```
df[2,]
```

```
##      nome sexo idade
## 2 João    M    34
```

```
df[1:2,]
```

```
##      nome sexo idade
## 1 Luis    M    12
## 2 João    M    34
```

E realizar a seleção condicional

```
df[df$genero == "F",]
```

```
## [1] nome sexo idade
## <0 rows> (or 0-length row.names)
```

Podemos também selecionar por colunas

```
df[,c(1,3)]
```

```
##      nome idade
## 1    Luis    12
## 2   João    34
## 3 Arnaldo    35
## 4     Ana    18
```

Ao utilizar o sinal “-” seleciona-se tudo menos o intervalo marcado

```
df[, -2]
```

```
##      nome idade
## 1    Luis    12
## 2   João    34
```

```
## 3 Arnaldo      35
## 4      Ana      18
```

e, juntando tudo:

```
df[df$idade >= 18 & df$genero == "F", c("nome", "idade")]
```

```
## [1] nome  idade
## <0 rows> (or 0-length row.names)
```

Agora um último comando de utilidades, comando `merge()`

```
l <- data.frame(id=1:26 , minusculas = letters[1:26])
l <- l[-c(1,10,15,20),]
```

```
l
```

```
##      id minusculas
## 2      2          b
## 3      3          c
## 4      4          d
## 5      5          e
## 6      6          f
## 7      7          g
## 8      8          h
## 9      9          i
## 11     11         k
## 12     12         l
## 13     13         m
## 14     14         n
## 16     16         p
## 17     17         q
## 18     18         r
## 19     19         s
## 21     21         u
## 22     22         v
## 23     23         w
## 24     24         x
## 25     25         y
## 26     26         z
```

```
L <- data.frame(indice = 26:1, maiusculas = LETTERS[26:1])
L <- L[-c(2,11,16,21),]
```

```
L
```

```
##      indice maiusculas
## 1         26          Z
## 3         24          X
## 4         23          W
## 5         22          V
## 6         21          U
## 7         20          T
## 8         19          S
## 9         18          R
## 10        17          Q
## 12        15          O
## 13        14          N
```

```
## 14      13      M
## 15      12      L
## 17      10      J
## 18       9      I
## 19       8      H
## 20       7      G
## 22       5      E
## 23       4      D
## 24       3      C
## 25       2      B
## 26       1      A
```

```
# inner join
merge(L, l , by.x = "indice", by.y = "id")
```

```
##      indice maiusculas minusculas
## 1         2          B           b
## 2         3          C           c
## 3         4          D           d
## 4         5          E           e
## 5         7          G           g
## 6         8          H           h
## 7         9          I           i
## 8        12          L           l
## 9        13          M           m
## 10       14          N           n
## 11       17          Q           q
## 12       18          R           r
## 13       19          S           s
## 14       21          U           u
## 15       22          V           v
## 16       23          W           w
## 17       24          X           x
## 18       26          Z           z
```

```
# left join
merge(L, l , by.x = "indice", by.y = "id", all.x = T)
```

```
##      indice maiusculas minusculas
## 1         1          A      <NA>
## 2         2          B           b
## 3         3          C           c
## 4         4          D           d
## 5         5          E           e
## 6         7          G           g
## 7         8          H           h
## 8         9          I           i
## 9        10          J      <NA>
## 10       12          L           l
## 11       13          M           m
## 12       14          N           n
## 13       15          O      <NA>
## 14       17          Q           q
## 15       18          R           r
## 16       19          S           s
## 17       20          T      <NA>
```

```
## 18      21      U      u
## 19      22      V      v
## 20      23      W      w
## 21      24      X      x
## 22      26      Z      z
```

right join

```
merge(L, l , by.x = "indice", by.y = "id", all.y = T)
```

```
##      indice maiusculas minusculas
## 1         2          B          b
## 2         3          C          c
## 3         4          D          d
## 4         5          E          e
## 5         6        <NA>         f
## 6         7          G          g
## 7         8          H          h
## 8         9          I          i
## 9        11        <NA>         k
## 10        12         L          l
## 11        13         M          m
## 12        14         N          n
## 13        16        <NA>         p
## 14        17         Q          q
## 15        18         R          r
## 16        19         S          s
## 17        21         U          u
## 18        22         V          v
## 19        23         W          w
## 20        24         X          x
## 21        25        <NA>         y
## 22        26         Z          z
```

outer join

```
merge(L, l , by.x = "indice", by.y = "id", all.y = T, all.x = T)
```

```
##      indice maiusculas minusculas
## 1         1          A        <NA>
## 2         2          B          b
## 3         3          C          c
## 4         4          D          d
## 5         5          E          e
## 6         6        <NA>         f
## 7         7          G          g
## 8         8          H          h
## 9         9          I          i
## 10        10         J        <NA>
## 11        11        <NA>         k
## 12        12         L          l
## 13        13         M          m
## 14        14         N          n
## 15        15         O        <NA>
## 16        16        <NA>         p
## 17        17         Q          q
## 18        18         R          r
## 19        19         S          s
```



```
## 20      20      T      <NA>
## 21      21      U      u
## 22      22      V      v
## 23      23      W      w
## 24      24      X      x
## 25      25      <NA>     y
## 26      26      Z      z
```

Funções Úteis

função replicate: rep()

```
? rep
rep(1,10)
```

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

```
rep(c(1,2,3),times = 3)
```

```
## [1] 1 2 3 1 2 3 1 2 3
```

```
rep(c(1,2,3),each = 10)
```

```
## [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3
```

Função Summary

```
summary(iris)
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width
## Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100
## 1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300
## Median :5.800 Median :3.000 Median :4.350 Median :1.300
## Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199
## 3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800
## Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500
## Species
## setosa :50
## versicolor:50
## virginica :50
##
##
##
```

Função Estrutura str()

```
str(iris)
```

```
## 'data.frame': 150 obs. of 5 variables:
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

Função para amostra

```
sample(1:100, 10 )
```

```
## [1] 57 92 9 93 72 26 7 42 98 83
```

Função wich

```
vetor <- letters
```

```
vetor=="j"
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE  
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [25] FALSE FALSE
```

```
vetor[vetor=="j"]
```

```
## [1] "j"
```

```
which(vetor == 'j')
```

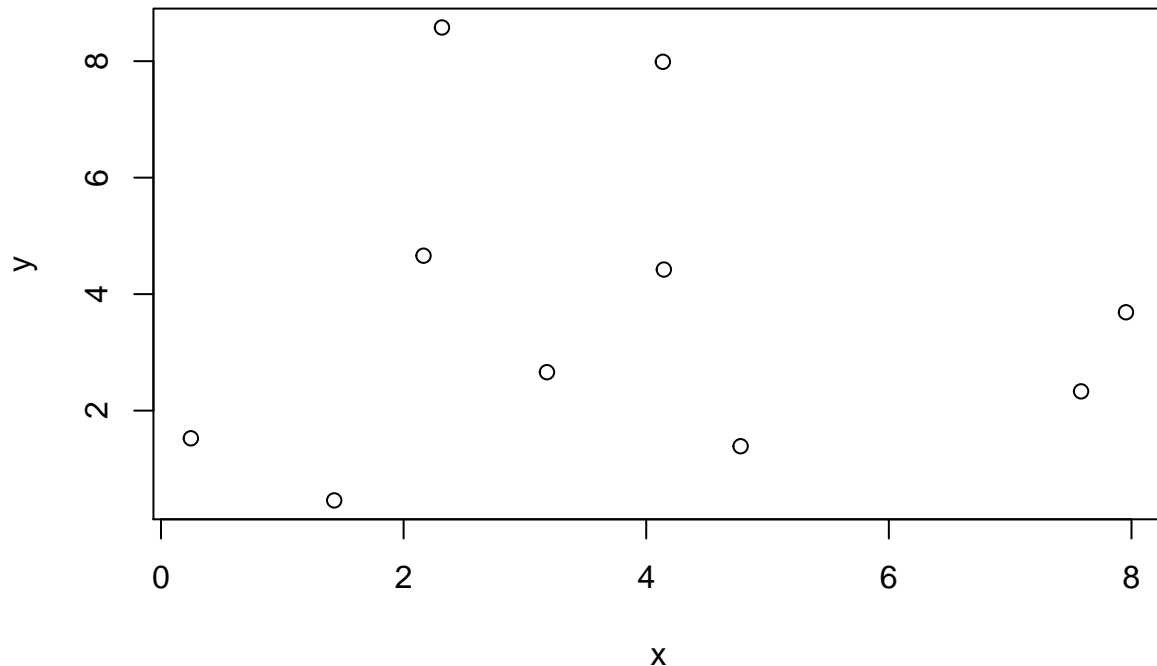
```
## [1] 10
```

Plotar Gráficos

Plotar gráficos de dispersão, basta utilizar a função `plot()`

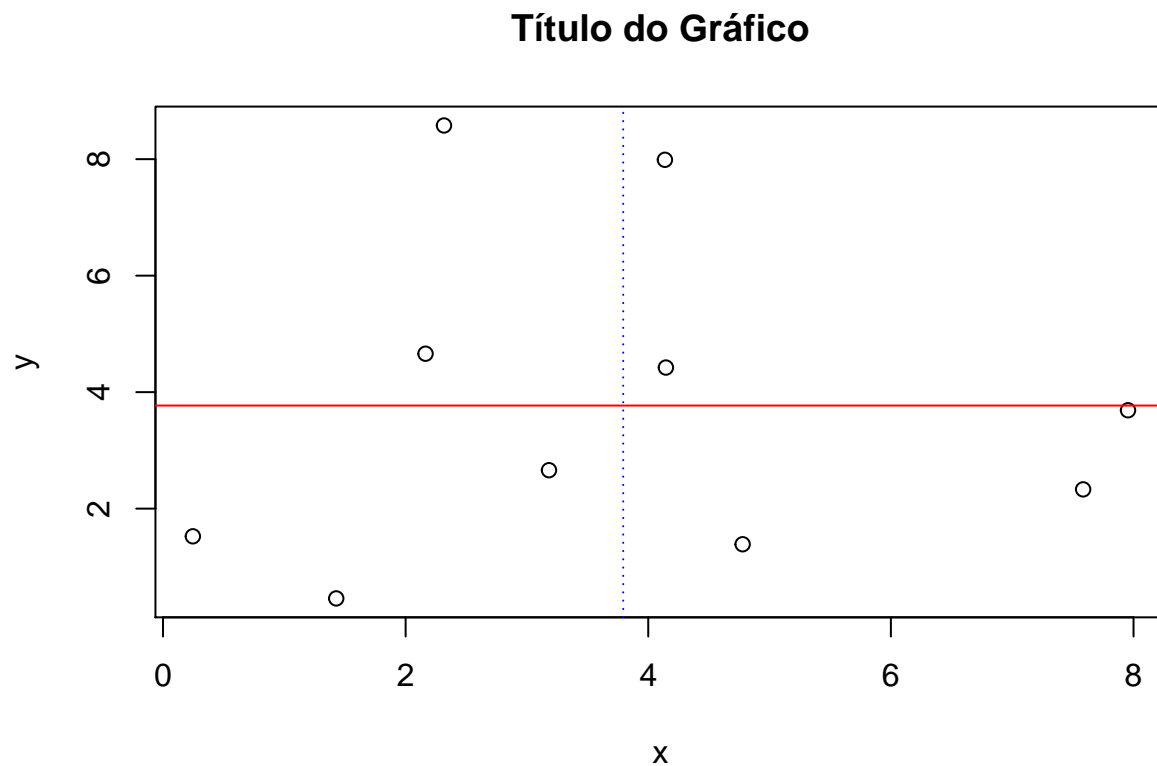
veja `help(plot)` para maiores informações

```
x = runif(n = 10, min = 0, max = 10 )  
y = runif(n = 10, min = 0, max = 10 )  
plot(x = x, y = y )
```



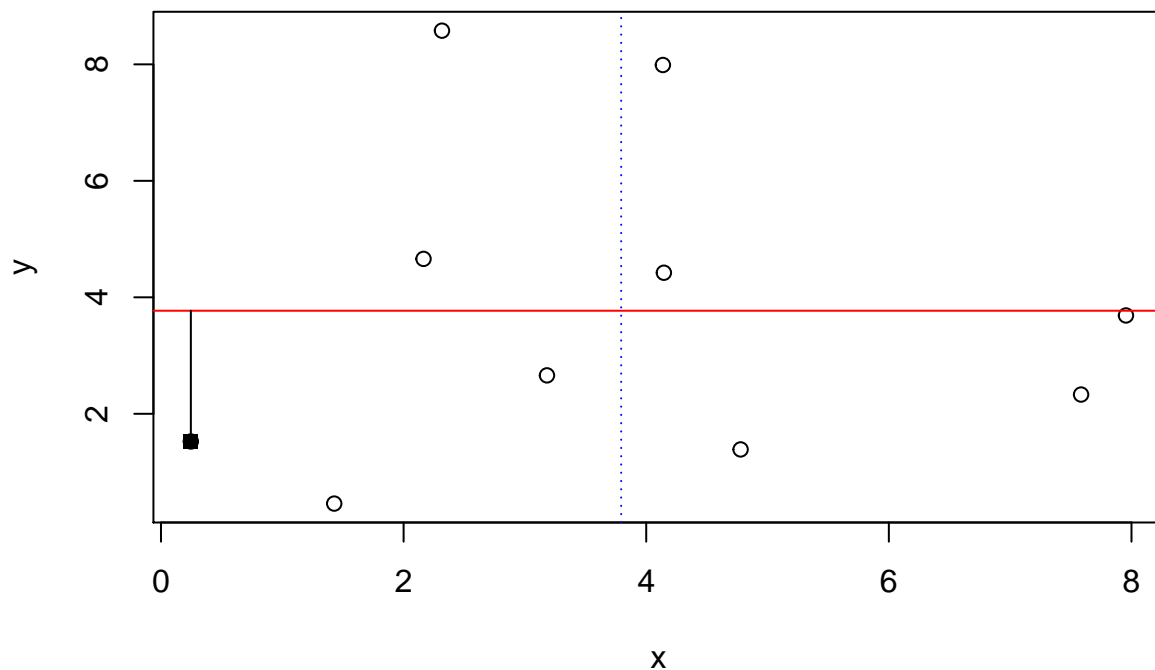
É possível adicionar outras linhas no gráfico com a função `abline()` onde os parâmetros `h` indicam a medida em `y` e `v` a de `x`

```
plot(x = x, y = y )
abline(h=mean(y) , col = "red")
abline(v=mean(x) , col = "blue", lty=3)
title("Título do Gráfico")
```



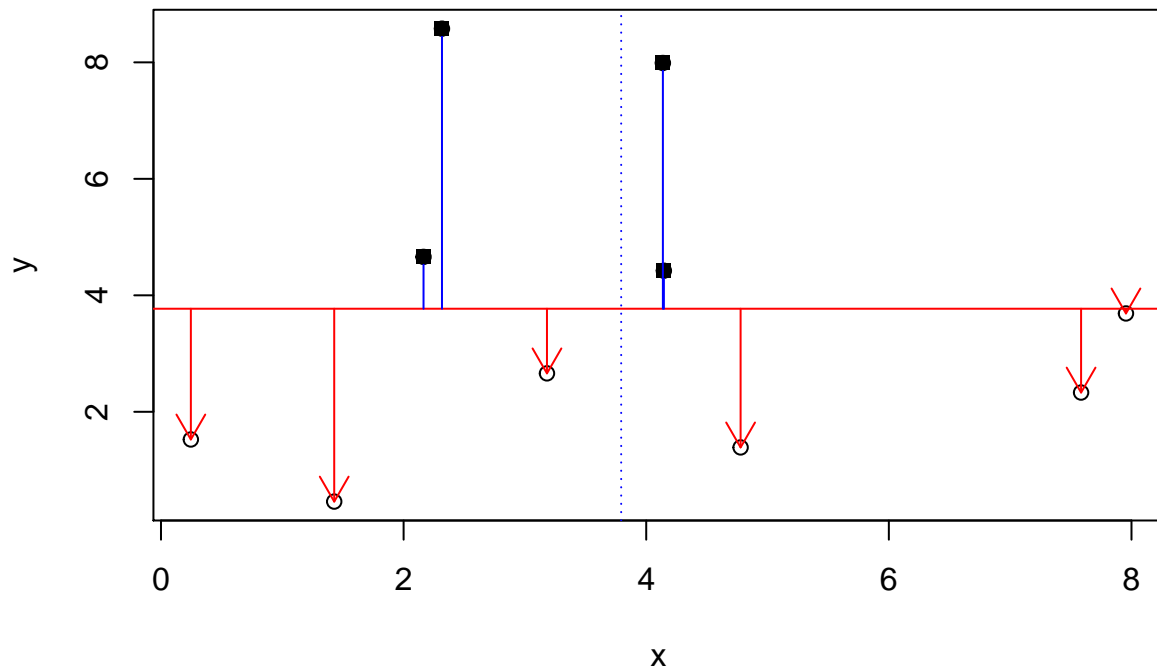
Assim como p configurar pontos e segmentos.

```
plot(x = x, y = y )
abline(h=mean(y) , col = "red")
abline(v=mean(x) , col = "blue", lty=3)
segments(x0 = x[2],x1 = x[2], y0 = mean(y),y1 = y[2])
points(x = x[2],y = y[2], pch = 15)
```



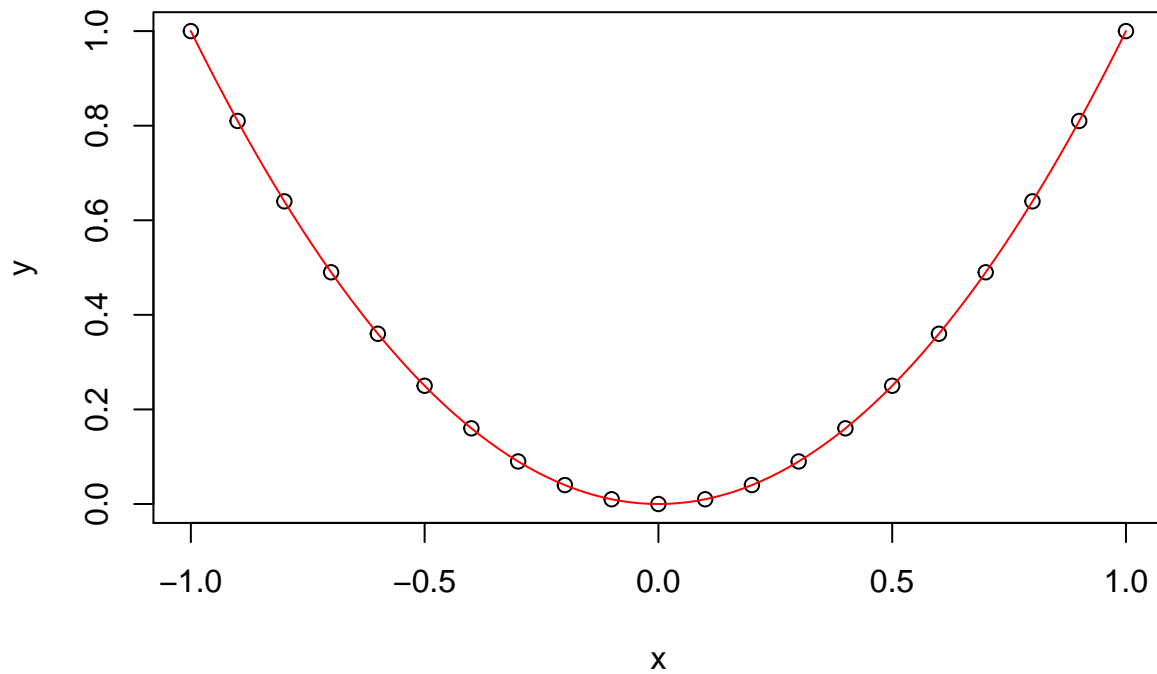
E podemos deixar as coisas mais interessantes

```
plot(x = x, y = y )
abline(h=mean(y) , col = "red")
abline(v=mean(x) , col = "blue", lty=3)
for (i in 1:length(x)) {
  if(y[i] > mean(y)){
    segments(x0 = x[i],x1 = x[i], y0 = mean(y),y1 = y[i], col = "blue")
    points(x = x[i],y = y[i], pch = 15)
  }else{
    arrows(x0 = x[i],x1 = x[i], y0 = mean(y),y1 = y[i], length = .15, col = "red")
  }
}
```



Outra função muito importante para gráficos é a `curve()`

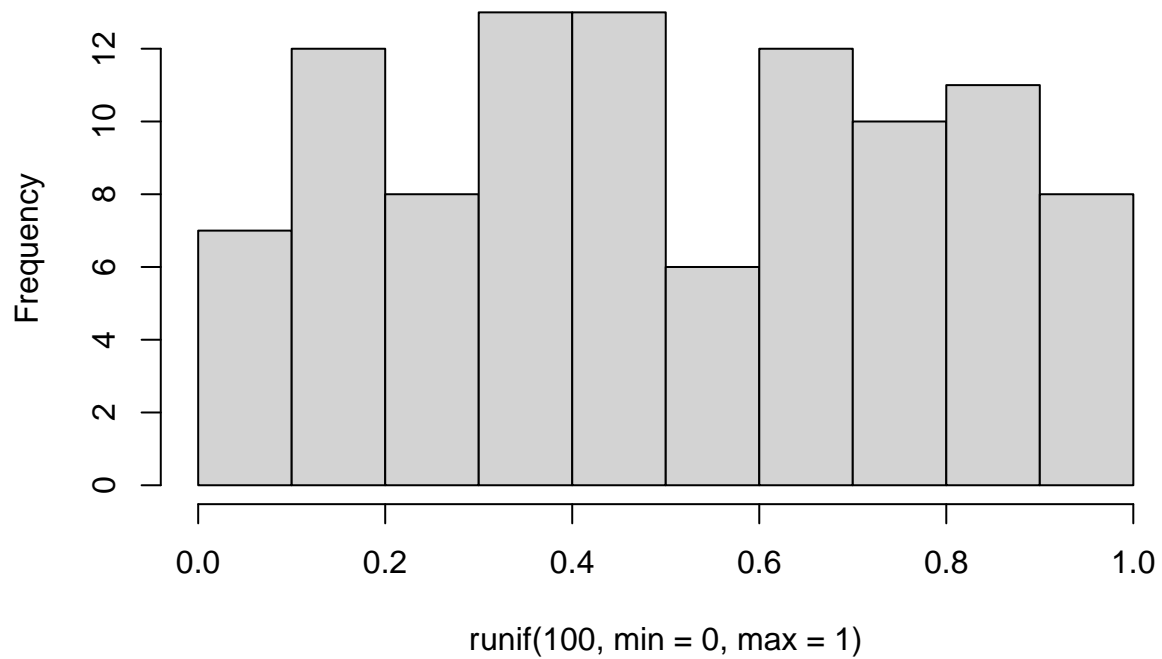
```
x = seq(from = -1, to = 1 , by = 0.1)
y = x^2
plot(x,y)
curve(x^2, from = -1 , to = 1, add = TRUE, col = "red" )
```



Histograma

```
hist(runif(100, min = 0, max = 1))
```

Histogram of runif(100, min = 0, max = 1)



```
set.seed(123)
x = rnorm(n = 10000, mean = 0, sd = 1)
hist(x, freq = F)
abline(v = 0, col = "red")
abline(v = 1.96, col = "blue", lty = 3 )
abline(v = -1.96, col = "blue", lty = 3 )
curve(expr = dnorm(x, mean= 0 , sd = 1), add = TRUE, col = "red" )
```

