Introdução ao R

Heitor Victor Veiga da Costa
hvvdc1@de.ufpe.br
Departamento de Estatística - UFPE

11 de dezembro de 2017

Sumário

Capítulo 1

Introdução ao R e ambiente RStudio

1.1 Conceito de algoritmo

Algoritmo é uma sequência de instruções bem definidas que descrevem como realizar determinada tarefa. Um algoritmo pode ser realizado por computadores ou seres humanos. Para um melhor entendimento deste conceito vamos analisar dois exemplos.
Exemplo 1: Possível algoritmo para calcular a moda amostral:

Definição: Moda amostral é o valor ou valores que ocorrem com mais frequência em uma amostra.
Algoritmo:

1. Faça uma tabela de frequências para o seu conjunto de dados.

2. Observe sua tabela de frequências e observe quais valores se repetem mais.

3. Se houver somente um valor que se repete mais que todos os outros, então este será a moda amostral desse conjunto de dados.

4. Defina seu conjunto de dados como **unimodal**.

5. Se o passo anterior for satisfeito, então finalize o processo, senão, siga para o passo seguinte.

6. Se dois ou mais valores se repetem com maior frequência, então seu conjunto de dados possui duas ou mais modas amostrais.

7. Defina seu conjunto de dados como **multimodal**.

8. Finalize o processo.

Exemplo 2: Quando vamos cozinhar um macarrão instantâneo e seguimos as instruções da embalagem, estamos seguindo um algoritmo.

Basicamente, um algoritmo é uma sequência de instruções ou tarefas necessárias para alcançar um determinado objetivo.

1.2 Ambientes R e RStudio

O R é um software livre(gratuito) e colaborativo para computação estatística e construção de gráficos. Além disso, o R também é uma linguagem de programação e por isso está em constante atualização, gerado por sua comunidade ativa ao redor do mundo.
Linguagem de Programação colaborativa é quando suas atualizações são feitas pelos próprios usuários. No R essas atualições são denominadas "Pacotes".

Como instalar o R:
O R pode ser baixado diretamente de seu site www.r-project.org e está disponível para as plataformas Linux, Windows e MacOS. Para instalar na plataforma Windows siga os passos a seguir:

1. Acesse o site do R.
2. Clique em CRAN.
3. Escolha o Mirror de sua prefência.
4. Clique na opção "install R for the first time".
5. Escolha a opção "Download R for Windows".
6. Execute o instalador.

RStudio:

O RStudio é um ambiente de desenvolvimento integrado(IDE) para o R e está disponível em duas edições: RStudio Desktop e RStudio Server. Utilizaremos a versão gratuita para desktop. O RStudio diferente do R comum possui um apelo visual muito maior que o R usual, e também busca melhorar a experiência do usuário com o ambiente R. Para entender melhor compare as duas imagens a seguir:

Note que o RStudio é muito mais organizado e com muito mais funcionalidades, tornando a experiência do usuário muito mais rica e menos cansativa. A seguir algumas funcionalidades que o RStudio traz ao R:

- Auto-completar de funções.
- Sugestão de funções ou objetos ao escrever palavras similares nas linhas de código.
- Abas específicas para determinada ação, tal como gráficos, bases de dados etc.
- Histórico de códigos utilizados.
- Executor de linhas de código sequenciais.
- Identificação de erros antes de executa-los.
- Separação entre linhas de código e resultados, facilitando a escrita e execução de comandos.

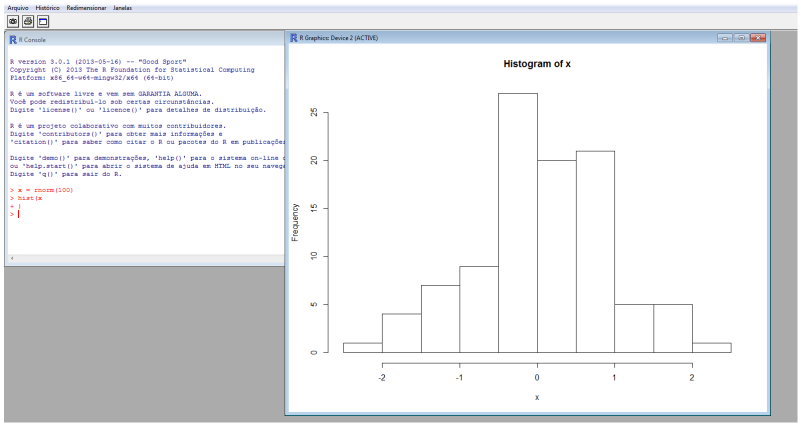


Figura 1.1: R usual

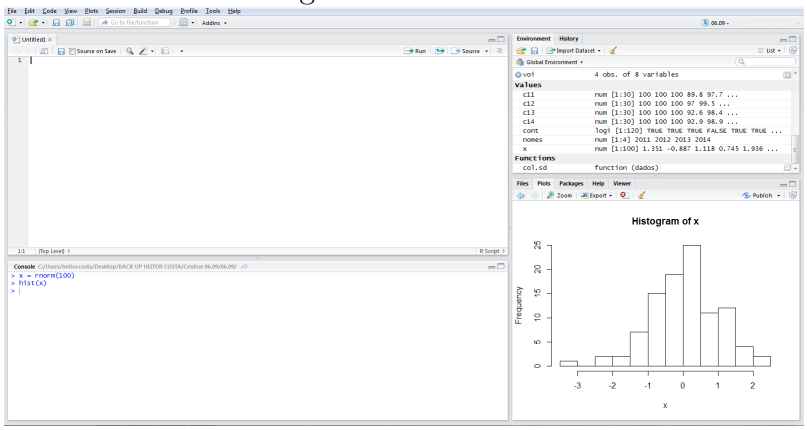


Figura 1.2: RStudio

Então, conseguimos ver que só existem vantagens em usar o RStudio, além disso é gratuito, por isso, use o RStudio!

Como instalar o RStudio para desktop:

1. Acesse o site do RStudio: https://www.rstudio.com
2. Clique na aba "Products"e na parte "RStudio Desktop"escolha a opção "DOWNLOAD RSTUDIO DESKTOP".
3. Escolha o download da aba "FREE".
4. Execute o instalador.

1.3 Cálculos básicos e funções

Para utilizar o R é necessário conhecer e escrever comandos específicos. Inicialmente, o usuário pode não estar acostumado mas, se comparado a softwares "enlatados"como SPSS, STATA, etc o R se mostra muito mais flexível, com mais recursos e mais adaptável. O R também pode ser utilizado como calculadora, e como foi dito anteriormente precisamos conhecer os comandos e sintaxe do R. Abaixo segue alguns comandos de operações matemáticas usuais:

```
Soma +; Subtração -; Multiplicação *; Divisão /; Expoente ^;
```

Como em uma calculadora usual, para cálculos mais complexos é aconselhável o uso de parênteses. Por exemplo, para fazermos a operação 12 + 6 dividido por 3 elevado ao cubo temos:

```
> ((12 + 6)/3)^3
```

O R possui também várias funções pré-definidas. Para utilizar uma função no R você deve conhecer duas coisas: O nome da função e seus argumentos.

função(argumentos)

Abaixo alguns exemplos de funções matemáticas usuais:
Exponencial (e^a):

```
> exp(a) #onde a é o expoente.
```

Raiz quadrada ($\sqrt{a}$):

```
> sqrt(a)
```

No R algumas funções possuem uma padronização de seus argumentos. Isso foi implementado para facilitar o uso de algumas funções que são repetidas com alta frequência. Recomenda-se que o usuário casual também faça isso para funções muito utilizadas. Abaixo algumas funções usuais que possuem argumentos pré-definidos:
Logaritmo (log_b^a):

```
> log(a, base = b)
```

Atenção: Em estatística é comun usar-se o logaritmo neperiano (log_e^a), por conta disto, o padrão da função log vem com b = e.

Arredondamento:

```
> round(a, digits = d) #Por padrão d = 0
```

O RStudio possui quatro janelas:

1. **Script:** É onde escrevemos os códigos. Para executar os códigos escritos no script basta seleciona-los e clicar no botão *Run* ou usar ctrl+r que serão executados de forma sequencial. Para salvar o script, clique no botão do disquete e após isso selecione um diretório e nome para seu script.
   Exemplo(faça em seu script):

   ```
   round(11.87779, 2)
   sqrt(25); 2^3
   ```

   Note que existem dois comandos na mesma linha. Isso é possível pois inserimos um **;**, este símbolo indica o fim de um comando.

2. **Console:** É onde são executados os comandos. No exemplo anterior todos os comando que executamos apareceram aqui com seus respectivos resultados. Nesta janela também podemos executar os comandos de forma direta, porém este procedimento não é aconselhável.

3. **Memória RAM:** É onde ficam todos os objetos e histórico de execuções. (Será discutido um pouco mais a frente).

4. **Arquivos gerados e arquivos externos:** Onde ficam os outputs de gráficos, função help, pacotes e pastas de trabalho.

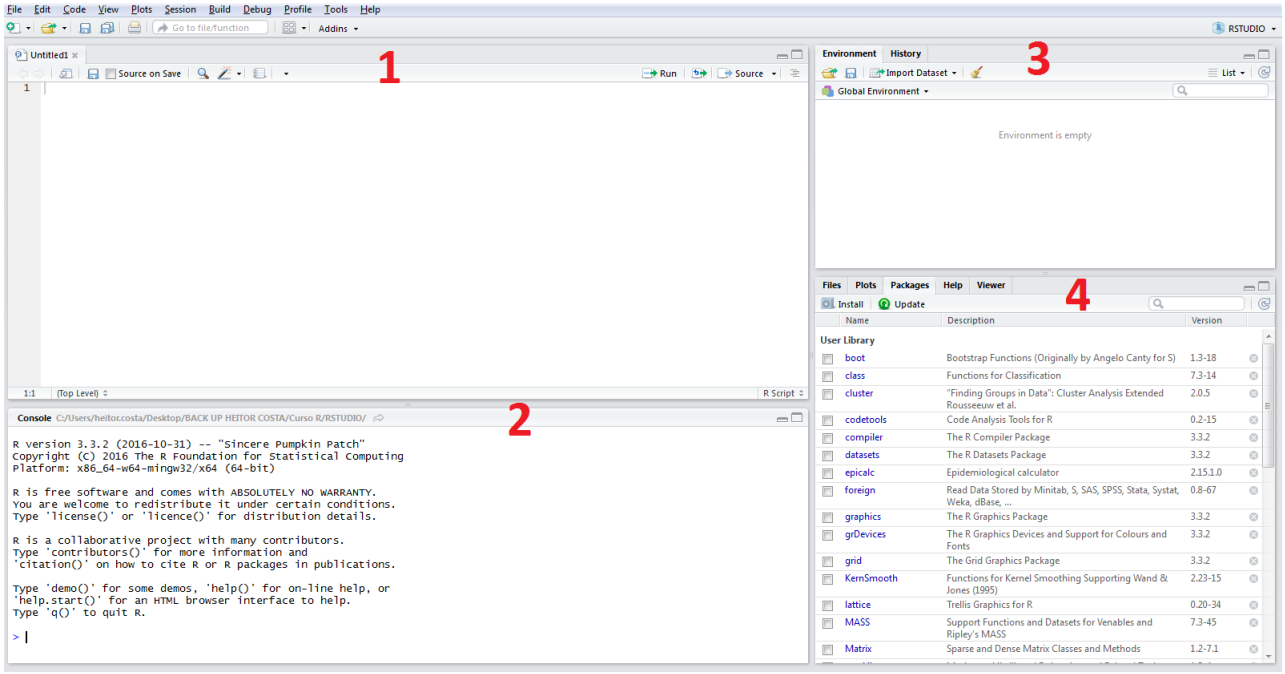


Figura 1.3: Janelas do RStudio

Se estiver escrito corretamente, a função irá retornar o resultado esperado, caso contrário, exibira algum tipo de aviso em seu console, veja o exemplo:
Resultado correto:

```
> log(1)
[1] 0
```

Resultado errado ou que não existe:

```
log(-1)
[1] NaN
Warning message:
In log(-1) : NaNs produced
```

1.4 Pacotes no R

Como dito anteriormente, para utilizar as funções no R é necessário saber escrever seus comandos específicos. Contudo, o R possui milhares de funções e se todas as vezes que fossemos utiliza-lo fosse necessário o carregamento destas funções seria preciso utilizar um longo tempo de inicialização. Por conta disto, foram criados os packages(pacotes),que são um compilado de funções.

O RStudio já possui alguns pacotes pré-instalados que podem ser encontrados na aba "Packages", para serem utilizados basta seleciona-los na lista ou utilizar o comando abaixo:

library("nome do pacote")

Caso o pacote que você deseja não esteja disponível ao instalar o RStudio, você pode baixar novos pacotes, basta conhecer o nome do mesmo. Para um exemplo prático vamos instalar o pacote **xlsx**.

1. Na aba "Packages"clique no botão "Install"
2. Caso você não tenha o arquivos .rar do pacote em seu computador, na opção "install from"escolha a opção: Repository(CRAN, CRANextra)
3. Na opção Packages insira o nome do pacote desejado , neste caso **xlsx**
4. Clique no botão install sem alterar as opções padrão.

Após o procedimento anterior o RStudio fará o download do pacote escolhido e o instalará em sua máquina automaticamente. Pronto, instalamos o pacote xlsx, mas e agora? Para que serve este pacote? Quais suas funções? Para responder estas perguntas precisamos entrar na página do pacote em questão. Para tal, na aba "Packages"clique em cima do nome do pacote e entao será aberta a janela "help", com várias informações sobre o pacote e suas funções. Para a descrição de alguma função específica utilize o comando abaixo:

help(nome da função) ou ?nome da função

1.5 Mémoria RAM e objetos

Como o R é uma linguagem de programação, precisamos entender um pouco oque é mémoria RAM. No computador existem dois tipos de mémoria, a de execução e a de armazenagem. Para executar determinada ação, o computador precisa utilizar um espaço finito de mémoria de execução, que é nossa mémoria RAM. Por exemplo, se a função exp(a) necessita de 8bits de mémoria RAM e só temos disponível 4bits, esta função não será executada.

Existem vários tipos de linguagens de programção, o R por exemplo é uma linguagem voltada à objetos. Para entender melhor este conceito vamos estudar o exemplo a seguir: No seu console escreva os seguintes códigos:

```
> x = 10
> y = 20
```

Ao fazer os comando acima estamos dando valores a x e y e salvando-os na memória RAM. Você vai ver que na janela "Environment"no espaço values estarão x e y com seus respectivos valores. Podemos chamar x e y de objetos e estes estão armazenados na memória RAM. Com estes objetos podemos fazer várias operações como:

```
> x + y
[1] 30
> x*y
[1] 200
> x/y
[1] 0.5
> (x + y)^(x/y)
[1] 5.477226
```

Existem vários tipos de objetos e esses serão discutidos em outra sessão.
Atenção: No R existe diferença entre letras maiúsculas e minúsculas, por isso muito cuidado ao definir objetos. O separador de decimais no R é dado por um ponto(.), por exemplo: 10,2 é escrito no R como 10.2.

1.6 Importação de base de dados

Vamos agora aprender como importar uma base de dados do excel em .xlsx e para tal utilizaremos o pocote **xlsx** e a função abaixo:

```
read.xlsx("nome do arquivo.xlsx", sheetName = "janela do arquivo excel")
```

Atenção: Para utilizar a função acima com somente estes dois argumentos, a base de dados deve estar na pasta onde estamos trabalhando.

1.7 Exercícios

Procurar na internet o nome e estudar os argumentos das seguintes funções no R:

- Valor absoluto $|x|$
- Somátorio $\sum$
- Produtório $\prod$
- Combinação $\binom{a}{b}$
- Fatorial $x!$

Use os resultados abaixo para testar as funções:
$|-10| = 10$
$\sum x_i = 20$ onde $x_i = 1, 2, 3, 4, 10$
$\prod x_i = 240$ onde $x_i = 1, 2, 3, 4, 10$
$\binom{5}{3} = 10$
$5! = 120$

Capítulo 2

Objetos e Variáveis

Como dito na sessão anterior vimos que o R é uma linguagem de programação voltada a objetos e que estes objetos ficam guardados na memória RAM. Nesta sessão iremos estudar alguns tipos de objetos.

2.1 Tipos de objetos

O R possui 6 tipos de objetos: vetores, matrizes, data-frames, listas, funções e arrays. Iremos focar nos cinco primeiros.

2.1.1 Vetores

Vetores são o tipo mais básico e simples de objeto para armanezar dados no R. Para criar um vetor usaremos a função c()(concatenação). Por exemplo:

```
> x = c(4,5,6,8,5.2)
> x
[1] 4.0 5.0 6.0 8.0 5.2
> letras = c("a","b","c","d","e")
> letras
[1] "a" "b" "c" "d" "e"
```

Para acessar os elementos de um vetor utilizaremos [**i**] onde i é o índice do elemento do vetor. Por exemplo, se quisermos saber o elemento da posição 2 no vetor x:

```
> x[2]
[1] 5
> letras[3]
[1] "c"
```

Se quisermos criar um vetor onde exista alguma lei de formação, podemos usar algumas das funções a seguir:

rep(e,n): Cria um sequência períodica onde "e"é o elemento de repetição e "n"é a quantidade de períodos(n = 1 por padrão). Veja o exemplo:

```
> y = rep(1,5)
> y
[1] 1 1 1 1 1
> z = rep(x,3) #Estamos repetindo o vetor x 3 vezes.
```

```
> z
 [1] 4.0 5.0 6.0 8.0 5.2 4.0 5.0 6.0 8.0 5.2 4.0 5.0 6.0 8.0 5.2
> p = rep(x) #Como por padrão n = 1, x é repetido uma única vez.
> p
[1] 4.0 5.0 6.0 8.0 5.2
```

seq(from,to,by): Cria um intervalo de valores onde "from"é primeiro elemento, "to"é o elemento que delimita o fim do intervalo e "by"é a lei de formação da sequência. Por exemplo:

```
> seq(0,12,2)
[1]  0  2  4  6  8 10 12
> seq(12,0,-2)
[1] 12 10  8  6  4  2  0
> seq(x[1],x[4],1)
[1] 4 5 6 7 8
```

Um caso partícular dessa função é dado pelo comando "a:b"que pode ser utilizado diretamente na função c(). Onde "a"é o inicio da sequência e "b"o final e sua lei de formação é dado sempre pelo valor 1. Veja o exemplo:

```
> x2 = c(1:7)
> x2
[1] 1 2 3 4 5 6 7
> x3 = c(7:1)
> x3
[1] 7 6 5 4 3 2 1
> c(x[2]:10)
[1]  5  6  7  8  9 10
```

Podemos também criar vetores sequenciais de caracteres utilizando a função letters[a:b](letras minúsculas) e LETTERS[a:b](letras maiúsculas). Por exemplo:

```
> alfabeto = letters[1:26]
> alfabeto
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r"
  "s" "t" "u" "v" "w" "x" "y" "z"
> ALFABETO = LETTERS[1:26]
> ALFABETO
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R"
  "S" "T" "U" "V" "W" "X" "Y" "Z"
```

Atenção: Vetores só podem ser formados por números ou caracteres, nunca os dois ao mesmo tempo.

2.1.2 Matrizes

Uma matriz é a reorganização de elementos de um vetor em linhas e colunas. Para criar uma matriz no R utilizaremos a função **matrix(e,i,j)** onde "e"são os elementos da matriz, "i"o número de linhas e "j"o número de colunas. Vale ressaltar que cada coluna de uma matriz deve possuir o mesmo nº de linhas. A seguir alguns exemplos:

```
> matrix(x,5,1)
     [,1]
[1,]  4.0
[2,]  5.0
[3,]  6.0
[4,]  8.0
[5,]  5.2
> matrix(c(x,1,2,3,10,11),5,2) #Estamos definindo um vetor diretamente
                                  na função matrix.
     [,1] [,2]
[1,]  4.0    1
[2,]  5.0    2
[3,]  6.0    3
[4,]  8.0   10
[5,]  5.2   11
```

Uma outra maneira de fazer uma matriz é unindo vetores através da função **cbind()**. Por exemplo:

```
> m = cbind(x,c(1,2,3,10,11)) #Temos a mesma matriz do exemplo anterior.
> m
       x
[1,] 4.0  1
[2,] 5.0  2
[3,] 6.0  3
[4,] 8.0 10
[5,] 5.2 11
> ml = cbind(letras,c("f","g","h","i","j"))
> ml
     letras
[1,] "a"    "f"
[2,] "b"    "g"
[3,] "c"    "h"
[4,] "d"    "i"
[5,] "e"    "j"
```

Para acessar os elementos de uma matriz ou obter uma submatriz usaremos o comando **[i,j]** onde i é o índice da linha e j é o índice da coluna. Segue o exemplo:

```
> m[1,2]

1
> m[4,2]

10


> m[c(3:5),c(1:2)] #Estamos acessando a submatriz das linhas 3 a 5
                    e das colunas 1 a 2.
       x
[1,] 6.0  3
[2,] 8.0 10
```

```
[3,] 5.2 11
> ml[5,2]

"j"
> ml[c(2:4),2] #Submatriz das linhas 2 a 4 e coluna 2
[1] "g" "h" "i"
```

Atenção: Tal como os vetores, as matrizes só podem ser formadas por números ou caracteres, o que é muito natural dado que matrizes são formadas por vetores.

2.1.3 Data-frames

Os data-frames são muito parecidos com matrizes, diferindo basicamente em uma coisa, os data-frames podem ser formados por números e caracteres. Como as matrizes, cada coluna de um data-frame deve possuir o mesmo nº de linhas. Em geral, podemos pensar em um data-frame como uma planilha do excel onde cada coluna representa uma variável e as linhas os dados pertencentes a esta variável. Para criarmos um data-frame devemos utilizar o seguinte comando:

$$\textbf{data.frame}(a_1,a_2,a_3,...,a_k)$$

Onde cada a_i é um vetor(de números ou caracteres). A seguir um exemplo:

```
> altura = c(1.75,1.5,1.85,1.55,1.7)
> classificacao = c("alto","baixo","alto","baixo","alto")
> dados = data.frame(altura,classificacao)
> dados
  altura classificacao
1   1.75          alto
2   1.50         baixo
3   1.85          alto
4   1.55         baixo
5   1.70          alto
```

Para acessar os elementos de um data.frame, basta utilizar **[i,j]**, como nas matrizes. Uma peculiaridade dos data-frames é que podemos acessar uma coluna inteira a partir do comando **$**.

```
> dados$altura
[1] 1.75 1.50 1.85 1.55 1.70
> dados$classificacao
[1] alto  baixo alto  baixo alto
Levels: alto baixo
```

Note que como estamos utilizando o RStudio, ao chamar o objeto dados$ o programa informa as variáveis que o data-frame possui.

2.1.4 Listas

Listas são basicamente vetores sem nenhuma restrição, isto é, cada elemento de uma lista pode ser qualquer coisa, inclusive outra lista. Imagine um varal onde você possa

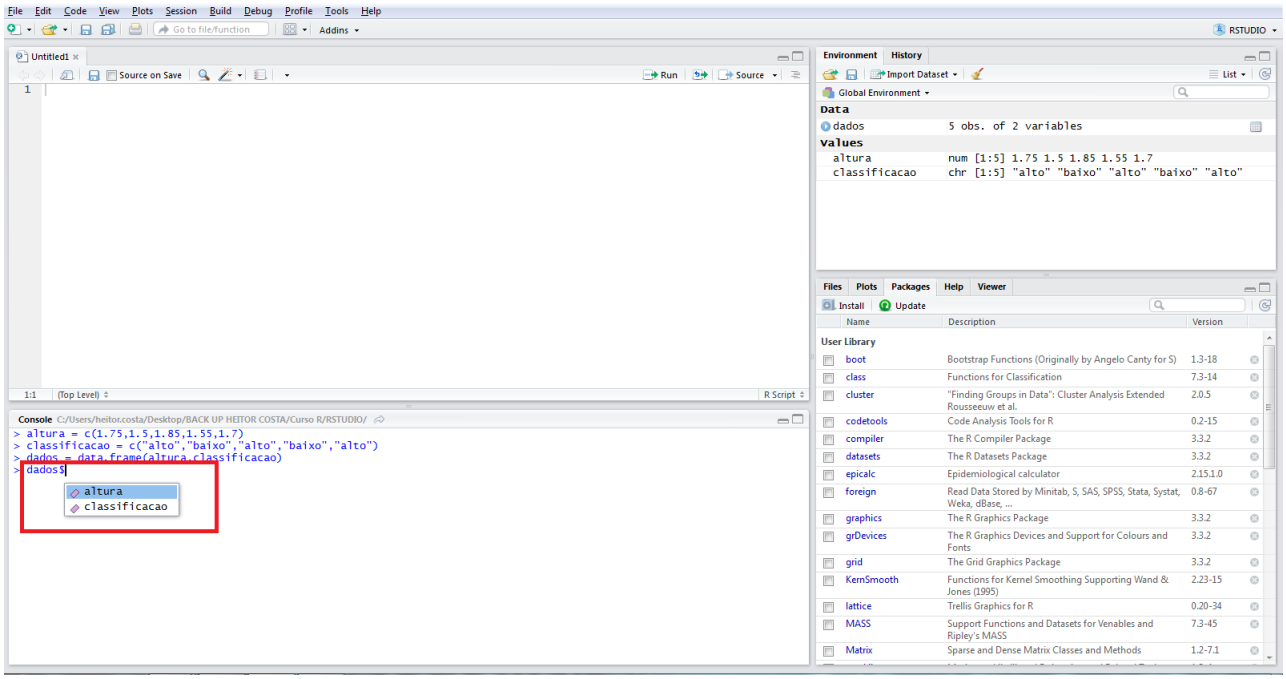


Figura 2.1: Comando $ para acessar variáveis em um data-frame

pendurar tudo o que você quiser, isso é uma lista. Dentre todos os objetos já citados, as listas são de longe as mais flexíveis, contudo, são as que dão mais trabalho para fazer manipulações. Para criar uma lista utilize o comando:

$$\textbf{list}(a_1,a_2,a_3,...,a_k)$$

Onde cada a_i é um objeto qualquer. Exemplos:

```
> lista = list(altura,classificacao,dados)
> lista
[[1]]
[1] 1.75 1.50 1.85 1.55 1.70

[[2]]
[1] "alto"  "baixo" "alto"  "baixo" "alto"

[[3]]
  altura classificacao
1   1.75          alto
2   1.50         baixo
3   1.85          alto
4   1.55         baixo
5   1.70          alto
```

Para acessar os elementos de uma lista utiliza-se o comando [[**i**]] onde i é índice do elemento da lista. Por exemplo:

```
> lista[[2]]
[1] "alto"  "baixo" "alto"  "baixo" "alto"
> lista[[3]]
  altura classificacao
1   1.75          alto
```

```
2   1.50         baixo
3   1.85          alto
4   1.55         baixo
5   1.70          alto
> lista[[3]]$altura #Como o elemento 3 da lista é um data-frame,
                     podemos ultilizar o comando $.
[1] 1.75 1.50 1.85 1.55 1.70
```

2.1.5 Funções

Funções são os objetos mais peculiares na linguagem R. Imagine uma máquina de sorvete, onde você insere os ingredientes como leite, açúcar, essências, etc e depois de um tempo a máquina te prepara um delicioso sorvete. Uma função é a mesma coisa, o usuário insere argumentos(ingredientes) e espera que a função retorne algo criado a partir destes argumentos. Para criarmos uma função utilizamos o seguinte comando:

$$\textbf{nomedafunção} = \textbf{function}(a_1,a_2,...,a_k)\{\bullet\}$$

Onde cada a_i é um argumento e $\bullet$ é oque a sua função vai executar. Como exemplo vamos implementar uma função que soma dois valores:
Atenção: Escreva os códigos a seguir em um script.

```
somar2 = function(a1,a2)
  {
  resultado = a1 + a2
  return(resultado)
  }
```

Para utilizar a função que nós criamos basta escrever seu nome e utilizar como argumentos os valores que você quer somar.

```
> somar2(1,2)
[1] 3
> somar2(1,5)
[1] 6
> somar2(1,5.2293)
[1] 6.2293
```

Vamos agora implementar uma função para calcular a média amostral. Onde dados é o vetor que o usuário deve fornecer e n é o tamanho da amostra. Lembre-se da expressão da média amostral:

$$\overline{X} = \frac{\sum x_i}{n}$$

```
media = function(dados,n)
  {
  resultado = sum(dados)/n #sum() é função do somatório
  return(resultado)
  }
> media(x,5)
[1] 5.64
```

Depois de implementar e testar sua função de calcular a média amostral, compare o resultado obtido a partir dela com o da função padrão do R mean(). Analise também os argumentos.

2.2 Tipos de Classes

Agora que já conhecemos os tipos de objetos no R, vamos estudar os tipos de classes de um objeto. Muitas vezes quando usamos uma função e ela retorna algo inesperado ou um erro, atrelamos este fato ao tipo de objeto, sendo que muitas vezes o resultado de determinada função depende do tipo de classe do objeto e não do tipo de objeto. Isso acontece com bastante frequência pois o tipo de objeto é definido na implementação da função, mudando somente o tipo de classe do objeto que vamos informar como argumento. A seguir definiremos os tipos de classes e daremos alguns exemplos para fixação do conteúdo.

Números inteiros:

É a classe de objetos que representa os números inteiros. Na linguagem R é donotada por **"integer"**.
Exemplos:

$$\mathbb{Z} = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$$

```
> Z = c(-3:3)
> Z
[1] -3 -2 -1  0  1  2  3
```

Números reais:

É a classe de objetos que representa os números reais. Em linguagem R é chamada de **"numeric"**.
Exemplos:

$$\mathbb{R} = \{..., -0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3, ...\}$$

```
> R = seq(-1,1,0.2)
> R
 [1] -1.0 -0.8 -0.6 -0.4 -0.2  0.0  0.2  0.4  0.6  0.8  1.0
```

Caracteres:

Classe de objetos que representa letras e palavras em geral. No R é indicada por **"character"**.
Exemplos:

```
> c = letters[1:5]
> c
[1] "a" "b" "c" "d" "e"
```

Fatores:

Classe de objetos que define categorias de classificação aos respectivos elementos. Podem ser númerais ou palavras (não confundir com as classes "numeric"e "character"), e em R é representado por **"factor"**.

Diferente das outras classes, precisamos explicitar para o R que este objeto é um fator, para isso utilizaremos a função **as.factor(objeto)** onde seu argumento será o objeto que queremos definir como um fator. Após transformarmos um objeto em fator ele irá fazer com que cada representação gráfica dos elementos de um objeto se tornem "Levels"(categorias). Muito cuidado nessa parte, já que o R diferencia letras maiúsculas de minúsculas.

```
> cf = as.factor(c) #Transformamos o objeto c que era da classe "character"
> cf                em um fator.
[1] a b c d e
Levels: a b c d e
> zf = as.factor(z) #Transformamos o objeto z que era da classe "integer"
> zf                em um fator.
[1] -3 -2 -1 0  1  2  3
Levels: -3 -2 -1 0 1 2 3
> dados$classificacao = as.factor(dados$classificacao)
> dados$classificacao
[1] alto  baixo alto  baixo alto
Levels: alto baixo
#Transformamos o objeto dados$classificacao que era da classe "character"
 em um fator.
```

Atenção: Caso se queira transformar um objeto da classe "numeric"em um fator, devemos estudar bem o objeto que queremos transformar pois isso poderia gerar uma infinidades de levels, oque não seria interessante.

Lógicos:

A classe de objetos do tipo lógico é definida por elementos do tipo booleano. São objetos formados por elementos do tipo 1 ou 0, onde 1 é TRUE e 0 é FALSO. Iremos estudar esta classe com mais detalhes na sessão de operadores lógicos. Essa classe é definida como "logical"em R. Para criarmos um objeto desse tipo seguiremos com o mesmo raciocínio da classe "factor"utilizando a função **as.logical(objeto)**.

Exemplos:

```
> l = c(1,1,1,1,0,0,0,1)
> l
[1] 1 1 1 1 0 0 0 1
> l = as.logical(l)
> l
[1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE
```

Observação: Caso o usuário queira transformar um objeto de uma classe para outra, basta utilizar o comando **as.nomedaclasse(objeto)**.

2.2.1 Funções que identificam tipos de classe

Como dito anteriormente é muito importante saber com que tipo de classe estamos trabalhando, e é muito natural que queiramos identificar o tipo de classe do objeto que estamos usando. Para isso, o R possui algumas funções de identificação de classes. Essas funções sempre retornam os valores TRUE ou FALSE. A seguir algumas funções para esta tarefa:

class(objeto): Identifica o tipo de objeto requerido.

```
> class(c)
[1] "character"
> class(z)
[1] "integer"
> class(r)
[1] "numeric"
> class(l)
[1] "logical"
```

Vale frizar que quando utilizamos **class()** em objetos compostos (data-frames, listas) a função retorna o tipo do objeto e não a classe a que ele pertence.

is.tipodaclasse(objeto)

Pergunta se o objeto é do tipo de classe desejado retornando TRUE se verdadeiro e FALSE caso contrário.

```
> is.integer(z)
[1] TRUE
> is.integer(r)
[1] FALSE
> is.integer(c)
[1] FALSE
> is.integer(l)
[1] FALSE
```

Note que como os inteiros pertencem ao conjunto dos reais a função **is.numeric()** retorna TRUE caso o objeto investigado seja formado por números inteiros.

```
> is.numeric(z)
[1] TRUE
```

Um outro identificador é a função **is.na(objeto)** que pergunta se um objeto ou um elemento de algum objeto não possui nada dentro dele, ou seja, se é nulo.

```
> na = c(1,2,3,NaN,5)
> is.na(na)
[1] FALSE FALSE FALSE  TRUE FALSE
```

2.3 Tipos de variáveis

Do ponto de vista da Estatística existem dois tipos de variáveis principais, as discretas e contínuas. Contudo, antes de nos aprofundarmos neste tópico será feita uma leve explanação sobre variáveis aleatórias que é de suma inportância para entender bem os cálculos estatísticos que irão surgir ao longo do curso.

Variável aleatória:
Denotamos como variável aleatória toda variável onde suas características são bem definidas(espaço amostral) porém, de caratér aleatório, ou seja, sabemos os valores que a variável pode assumir mas eles ocorrem de forma não controlada.

2.3.1 Variáveis aleatórias discretas

Variável aleatória discreta é toda aquela que pode ser contada ou catalogada em um espaço finito de estados, ou seja, que possam ser indexadas. Sendo assim, as variáveis do tipo qualitativa se encaixam muito bem como variáveis aleatórias discretas.

Exemplos:

- **Lançamento de uma moeda**
  Ao lançarmos uma moeda sabemos que os possíveis valores serão **{cara,coroa}**, contudo, não sabemos qual será esse valor a não ser que façamos um experimento.

- **Sexo dos moradores de um determinado bairro**
  Os indivíduos podem ser do sexo masculino ou feminino. Sendo assim nossa variável aleatória pode assumir os valores **masculino** e **feminino**.

- **Dias da semana**
  Sabemos que uma semana tem sete dias, portanto nossa variável aleatória assume os valores **{domingo, segunda-feira, ... ,sábado}**. Ainda podemos criar outra variável a partir dessa que são os dias úteis da semana como **{útil,não-útil}**.

Em R as classes de objetos a que este tipo de varável pode pertencer são os inteiros(integer), caracteres(character), fatores(factor) e lógicos(logical).

2.3.2 Variáveis aleatórias contínuas

Variável aleatória contínua é toda aquela que assume valores em um intervalo, ou seja, variáveis contínuas são variáveis quantitativas.

Exemplos:

- **Peso:**
  Geralmente o peso de uma pessoa varia entre zero e 300kg, portanto nossa varável aleatória pode assumir valores no intervalo **{0,300}** em kg ou **{0,300000}** em gramas.

- **Altura:**
  Comumente uma pessoa tem altura entre zero e 2,3 metros, então, a variável Altura assume valores no intervalo **{0,230}** em centímetros ou **{0,2.3}** em metros.

Ao utilizarmos uma variável do tipo contínua devemos ter muito cuidado ao defini-la pois, como vimos nos exemplos podemos ter vários intervalos diferentes para uma mesma variável. Em R este tipo de variável só pode pertencer a classe númerica(numeric).

Vale salientar que uma variável contínua pode ser facilmente transformada em uma variável discreta. A partir do intervalo original podemos criar sub-intervalos e categoriza-los como bem quisermos. Veja os exemplos:

Exemplo:
Altura:
Antes tinhamos em metros, que a altura pertencia ao intervalo **{0,2.3}**. Uma possível categorização seria:

Sub-intervalo	Categoria
{0,1.6}	Baixo
{1.6,1.85}	Regular
{1.85,2.3}	Alto

2.4 Comandos lógicos no R

Muitas vezes queremos fazer comparações entre objetos ou elementos de um objeto, para isso existem os operadores lógicos. Esse operadores são muito úteis ao análisar dados já que com eles podemos fazer comparações e condicionar alguns objetos a partir de outros objetos. A seguir a lista de operadores lógicos que podem ser utilizados no R:

Operador	Descrição
==	igual
!=	diferente
>	maior
<	menor
>=	maior ou igual
<=	menor ou igual
&&	e
||	ou
!	não

Atenção: Cuidado para não confundir o operador lógico **igual (==)** com o de **atribuição (=)**.
Sempre que fizermos este tipo de operação será retornado um valor lógico TRUE ou FALSE.

Exemplos:

```
> dados$altura>1.7
[1]  TRUE FALSE  TRUE FALSE FALSE
> dados$altura>=1.7
[1]  TRUE FALSE  TRUE FALSE  TRUE
> dados$classificacao=="baixo"
[1] FALSE  TRUE FALSE  TRUE FALSE
> !(dados$altura>1.75)
[1]  TRUE  TRUE FALSE  TRUE  TRUE
> dados$classificacao==dados$altura
```

```
[1] FALSE FALSE FALSE FALSE FALSE
> dados$classificacao!=dados$altura
[1] TRUE TRUE TRUE TRUE TRUE
> (dados$altura>1.75)||(dados$classificacao=="alto")
[1] TRUE
> (dados$altura>1.75)&&(dados$classificacao=="alto")
[1] FALSE
> (dados$altura>1.75)||(dados$classificacao!="alto")||(dados$altura<1.8)
[1] TRUE
```

Note que ao utilizarmos os operadores || ou && eles só retornam uma resposta, ou seja, a comparação só está sendo feita no primeiro elemento dos dois vetores. Isto ocorre pois, para estes operadores devemos explicitar o índice de cada elemento a ser testado. Para resolver este problema, precisamos acessar todos os elementos de um vetor para que as comparações sejam feitas em todos os elementos. Veremos a seguir como resolver este problema.

2.4.1 Comandos de decisão - if e else

O comando if (se) é utilizado para tomar decisão, ou seja, se a condição que impusermos for verdadeira será executada uma série de tarefas, senão, não fazemos nada ou tomamos outra decisão. Este último é o comando else (senão), que é ocional. Em outras palavras, estamos criando um algoritmo para que o computador faça a operação indicada por nós. A síntaxe dos comandos de decisão são muito fáceis e intuítivas, veja:

```
if(isso aconte){
  faça isso
}
else{
  faça aquilo
}
```

A condição deve sempre vir entre parênteses e a ação dentro das chaves.

```
resposta = character()
if(dados$altura[1]>=1.70){
  resposta[1] = "Alto"
}
else{resposta = "Baixo"}
```

Note que ao executarmos o else aparece uma mensagem de erro. Isso ocorre pois o else só deve ser usado na implementação do objeto função(function) ou em comando de repetição. Iremos discutir essa parte em um exemplo mais a frente, por hora continue a leitura.
Para criar o vetor resposta com 5 elementos faça os códigos a seguir:

```
resposta = character()
if(dados$altura[1]>=1.70){
  resposta[1] = "Alto"
}
if(dados$altura[1]<1.70){
  resposta[1] = "Baixo"
}
```

```
if(dados$altura[2]>=1.70){
  resposta[2] = "Alto"
}
if(dados$altura[2]<1.70){
  resposta[2] = "Baixo"
}
if(dados$altura[3]>=1.70){
  resposta[3] = "Alto"
}
if(dados$altura[3]<1.70){
  resposta[3] = "Baixo"
}
if(dados$altura[4]>=1.70){
  resposta[4] = "Alto"
}
if(dados$altura[4]<1.70){
  resposta[4] = "Baixo"
}
if(dados$altura[5]>=1.70){
  resposta[5] = "Alto"
}
if(dados$altura[5]<1.70){
  resposta[5] = "Baixo"
}
```

Observe o trabalho que nos deu criar um vetor com 5 elementos. Para que não tenhamos sempre este problema ao fazermos operações similares foram criados os comandos de repetição, os famosos comandos de "loop".

2.4.2 Comandos de repetição - for e while

O comando for (para) é uma estrutura de repetição onde o usúario indica o número de repetições a serem feitas a partir de um valor inicial. É muito utilizado caso queiramos averiguar os elementos de um vetor.
Estrutura do comando for:

```
for(i in 1:n){
  Execute as intruções}
```

Onde:
i é o objeto de iteração.
n é o número de repetições.

Em outras palavras, oque o código nos diz é: Para i indo de 1 até n, faça as seguintes intruções n vezes. Por exemplo, caso queiramos fazer o vetor resposta do exemplo do comando if faremos o seguinte código:

```
resposta2 = character()
i = 1
for(i in 1:5){
  if(dados$altura[i]>=1.70){
```

```
    resposta2[i] = "Alto"
  }
  else{
    resposta2[i] = "Baixo"
  }
}
```

Ótimo não é? Fizemos a mesma operação de antes com $\frac{1}{6}$ do número de linhas. Veja também que o comando else pôde ser utilizado, diminuindo ainda mais o número de linhas escritas. Genial!

Agora que já sabemos utilizar o comando for, estudaremos o comando while. O comando while (enquanto) é muito parecido com o comando for. Ele gera uma estrutura de repetição porém o número de repetições é imposto por uma condição lógica. Sendo assim ele pode gerar infinitas repetições ou nenhuma, tudo depende da condição imposta pelo usúario.

Estrutura do comando while:

```
while(isso for verdader){
  Execute as intruções
}
```

Como exemplo vamos fazer o mesmo vetor do exemplo do comando if. Veja:

```
resposta3 = character()
i = 1
while(i<5){
  if(dados$altura[i]>=1.70){
    resposta3[i] = "Alto"
  }
  else{
    resposta3[i] = "Baixo"
  }
  i = i + 1
}
```

Observe que o vetor resposta3 só possui 4 elementos, isso ocorreu por que a condição lógica do while é de que i¡5. Para que fique correto devemos usar a condição lógica de i¡6 ou i¡=5.

```
resposta3 = character()
i = 1
while(i<=5){
  if(dados$altura[i]>=1.70){
    resposta3[i] = "Alto"
  }
  else{
    resposta3[i] = "Baixo"
  }
  i = i + 1
}
```

Fizemos a mesma operação de 3 formas diferentes. Como a linguagem R é muito facilitada ainda temos o comando condicionante, que funciona como um atalho ao comando if. E graças a ele podemos fazer a mesma operação já realizada, agora em apenas 2 linhas.

Estrutura do comando condicionante:

```
objeto[condição]
```

Onde:
"objeto"é oque será condicionado.
"condição"é a condicão imposta a este objeto.

```
resposta4 = character()
resposta4[dados$altura>=1.7] = "Alto"
resposta4[dados$altura<1.7] = "Baixo"
```

Agora teste usando o comando == (igual) se todos os vetores resposta criados são iguais. E então, são iguais? Incrível não é?

Capítulo 3

Estatística

Estatística é a ciência onde se estudam métodos apropriados para obtenção, representação e análise de dados. Nesta sessão serão apresentados os métodos de como fazer análise descritiva de dados e de inferência estatística.

Para esta sessão iremos utilizar um banco de dados composto pelo índice de desenvolvimento humano municipal(IDHM) do Brasil dos anos de 1990, 2000 e 2010 acompanhado do resultado do 1º turno das eleições para presidente no ano de 2014. Para download deste conjunto de dados utilize o link a seguir:
https://1drv.ms/x/s!AvVenL2qYu22gQlq2sAJFC7KwSAK

Para esta sessão espera-se que o leitor já possua uma base teórica em estatística básica, comumente apresentada em cursos de graduação para que seja possível o bom entendimento dos códigos a serem apresentados.

3.1 Estatística descritiva

As estatísticas descritivas são resultados que resumem e descrevem um conjunto de dados. Essas técnicas são muito importantes, são graças a elas que podemos construir uma "base"para fazermos inferências sobre a populção de estudo.

3.1.1 Medidas de posição:

São as medidas que representam os dados orientando-os quanto à a sua posição

- Média amostral:

  mean(objeto, na.rm)

  onde o argumento na.rm indica que o cálculo será feito utilizando os dados faltantes.
  Exemplos:

```
> mean(idhm$IDHM_2010, na.rm = TRUE) #IDHM médio do Brasil no ano de 2010
[1] 0.6591574
> mean(idhm$Dilma.Rousseff) #Média de votos por cidade de Dilma
[1] 7763.611
> mean(idhm$Levy.Fidelyx) #Média de votos por cidade de Levy
[1] 80.15117
> mean(idhm$Aécio.Neves[idhm$IDHM_2010>0.55]) #Média de votos de Aécio
[1] 6621.834                      nas cidades com idhm acima de 0.55
```

- Mediana amostral:

median(objeto, na.rm)

onde o argumento na.rm indica que o cálculo será feito utilizando os dados faltantes. **Exemplos:**

```
> median(idhm$IDHM_2010, na.rm = TRUE) #Mediana do IDHM do Brasil em 2010
[1] 0.665
> median(idhm$Dilma.Rousseff) #Mediana dos votos de Dilma
[1] 3138
> median(idhm$Aécio.Neves[idhm$IDHM_2010>0.55], na.rm = TRUE)
[1] 1729    #Mediana dos votos de Aécio nas cidades com idhm acima de 0.55
```

- Quantis e valores extremos de um conjunto de dados:

Um quantil é a posição (x) do eixo horizontal (eixo x) que representa a frequência observada (probabilidade) de um conjunto de dados.

$$P(X \leq x)$$

A mediana é o quantil 50%, ou seja, é a posição do eixo horizontal que representa a probabilidade de 50%. O código para encontrar os quantis de um conjunto de dados é dado por:

quantile(objeto, probs, na.rm)

onde probs é o vetor de quantis que se deseja calcular e na.rm expresa que o cálculo será feito utilizando os dados faltantes. **Exemplos:**

```
> quantile(idhm$IDHM_2010,c(0.25,0.5,0.75), na.rm = T)
  25%   50%   75%  #Quartis
0.599 0.665 0.718
> quantile(idhm$IDHM_2010,seq(0.1,1,0.1), na.rm = T) #Decis
  10%   20%   30%   40%   50%   60%   70%   80%   90%  100%
0.562 0.588 0.611 0.638 0.665 0.688 0.708 0.727 0.750 0.862
```

Dada a definição acima é fácil ver que os valores máximos e mínimos são respectivamente os quantis para 0% e 100%. Os comandos são dados como segue:

min(objeto,na.rm) e max(objeto,na.rm

```
> quantile(idhm$IDHM_2010,c(0,1), na.rm = T)
   0%  100%
0.418 0.862
> min(idhm$IDHM_2010, na.rm = T)
[1] 0.418
> max(idhm$IDHM_2010,na.rm = T)
[1] 0.862
```

- Sumário dos dados:

  O R possui uma função que faz o sumário de uma variável, ou seja, se a variável for contínua serão feitos os cálculos de medidas de posição e se forem discretas serão calculadas as frequências absolutas.

  summary(objeto)

  Exemplos:

```
> summary(idhm$IDHM_2010) #Medidas de dispersão para o IDHM do Brasil em 2010
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
 0.4180  0.5990  0.6650  0.6592  0.7180  0.8620       5
 > summary(idhm$Estado) #Freq. absoluta para os estados do Brasil
 AC  AL  AM  AP  BA  CE  DF  ES  GO  MA  MG  MS  MT  PA  PB  PE  PI  PR  RJ  RN
 22 102  62  16 417 184   1  78 246 217 853  79 141 144 223 185 224 399  92 167
 RR  RS  SC  SE  SP  TO
 15 497 295  75 645 139
```

  Se o objeto utilizado como argumento dessa função for um data-frame será feita a sumarização de cada variável contida no data-frame.

3.1.2 Medidas de dispersão

As medidas de dispersão medem o grau de variabilidade dos elementos de um conjunto de dados. Essas medidas são muito importantes, já que a todo momento estamos resumindo os dados através de uma medida de posição. Geralmente as medidas de dispersão mais utilizadas são o desvio padrão, amplitude e coeficiente de variação.

- Variância e desvio padrão amostral:

  A variância é soma ao quadrado dos desvios dos elementos em relação a sua média divididos por n-1. Sintetizando, ela mede em média, o quanto os dados desviam da média.

$$S^2 = \frac{\sum(x_i - \overline{x})^2}{n-1}$$

  Como a variância gera uma unidade de medida elevada ao quadrado, é comum usar o desvio padrão afim de normalizar a unidade de medida utilizada. O desvio padrão é a raiz quadrada da variância.

$$S = \sqrt{S^2}$$

  No R as funções para o cálculo destas medidas são:

  Variância: **var(objeto,na.rm)**
  Desvio padrão: **sd(objeto, na.rm)**

  Exemplos:

```
> var(idhm$IDHM_2010, na.rm = T)
[1] 0.005183609
> sd(idhm$IDHM_2010, na.rm = T) #Desvio padrão do IDHM BR em 2010
[1] 0.07199728
> sd(idhm$IDHM_2010, na.rm = T)==sqrt(var(idhm$IDHM_2010, na.rm = T))
[1] TRUE
> sd(idhm$IDHM_2000,na.rm = T) #Desvio padrão do IDHM BR em 2000
[1] 0.1043961
```

- Amplitude amostral:

  A amplitude é o intervalo onde a amostra assume valores. É calculada fazendo a diferença entre o maior e o menor valor assumido por nossa amostra. Não existe uma função específica em R para este cálculo, porém ele é muito simples e fácil de ser feito. A seguir um possível método de cálculo da amplitude utilizando o R:

  amplitude = max(objeto) - min(objeto)

  Exemplos:

```
>max(idhm$IDHM_2010, na.rm = T) - min(idhm$IDHM_2010, na.rm = T)
[1] 0.444 #Amplitude do IDHM do Brasil no ano de 2010
> max(idhm$IDHM_2000, na.rm = T) - min(idhm$IDHM_2000, na.rm = T)
[1] 0.612 #Amplitude do IDHM do Brasil no ano de 2000
```

- Coeficiente de variação:

  Muitas vezes cálculamos o desvio padrão, contudo, não conseguimos identificar se o resultado obtido foi alto ou baixo. Para isso foi criado o coeficiente de variação que consiste em ver o quanto o desvio padrão está variando da média.

  $$CV = \frac{S}{\overline{X}} * 100$$

  Em R também não temos uma função específica para esse cálculo, mas podemos utilizar o seguinte código:

  cv = (sd(objeto)/mean(objeto))*100

  Exemplos:

```
>(sd(idhm$IDHM_2010, na.rm = T)/mean(idhm$IDHM_2010, na.rm = T))*100
[1] 10.92262  #Coef. var. do IDHM do Brasil no ano de 2010
> sd(idhm$IDHM_2000,na.rm = T)/mean(idhm$IDHM_2000,na.rm = T)
[1] 0.1994268 #Coef. var. do IDHM do Brasil no ano de 2000
```

Vale frizar que as medidas de posição e de dispersão expostas anteriormente só funcionam para variáveis contínuas.

3.1.3 Gráficos

Gráficos são representações visuais criadas afim de tentar representar informações obtidas através de dados (quantitativos ou qualitativos). Em estatística são essênciais para a melhor visualização dos resultados das análises.

Gráfico de Barras

barplot(objeto)

Exemplo 1:

Vamos fazer um gráfico de barras com a proporção de votos válidos para cada candidato. Para isso vamos criar a nova variável prop.votos para os três primeiros colocados.

```
#Proporção de votos válidos para Dilma
prop.votos.total.dilma = sum(idhm$Dilma.Rousseff)/sum(idhm$Total)
#Proporção de votos válidos para Aécio
prop.votos.total.aecio = sum(idhm$Aecio.Neves)/sum(idhm$Total)
##Proporção de votos válidos para Marina
prop.votos.total.marina = sum(idhm$Marina.Silva)/sum(idhm$Total)
```

Agora que já possuimos as variáveis de interesse vamo criar o gráfico:

```
#grafico simples
barplot(c(prop.votos.total.aecio,prop.votos.total.dilma,prop.votos.total.marina))
```

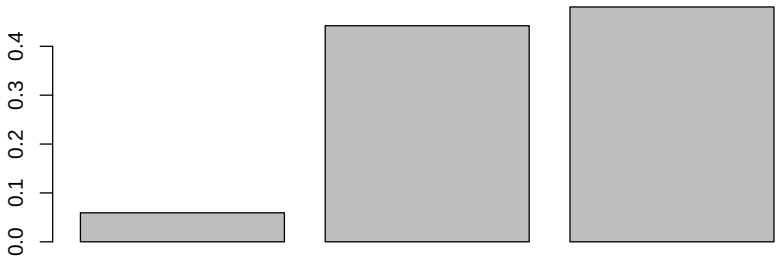


Figura 3.1: Gráfico simples sem nenhum ajuste

Note que o gráfico ficou muito feio e sem informação nenhuma. Contudo, podemos melhorar e muito este gráfico, basta utilizar alguns argumentos na função barplot. Veja:

```
#Grafico simples na horizontal
barplot(c(prop.votos.total.aecio,prop.votos.total.dilma,prop.votos.total.marina),
          horiz = TRUE)
```

Se o argumento horiz = TRUE as barras do gráfico ficam na horizontal, caso contrário, as barras ficam na vertical. Por padrão horiz = FALSE.

```
#Grafico simples na horizontal com título
barplot(c(prop.votos.total.aecio,prop.votos.total.dilma,prop.votos.total.marina),
        horiz = TRUE, main = "Proporção de votos válidos dos 3 primeiros colocados
        nas eleições para presidente em Pernambuco no ano de 2014")
```

O argumento main cria um título para o gráfico e deve ser sempre escrito entre aspas.

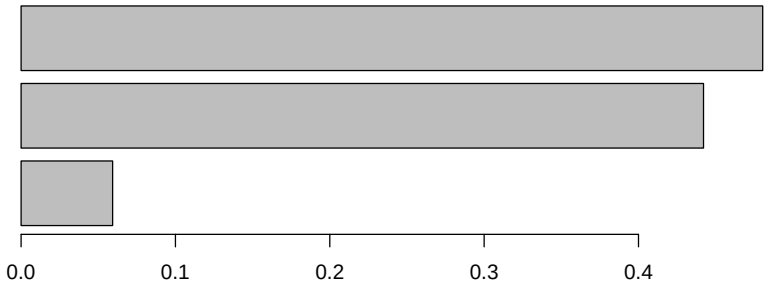


Figura 3.2: Gráfico simples sem nenhum ajuste com as barras na horizontal

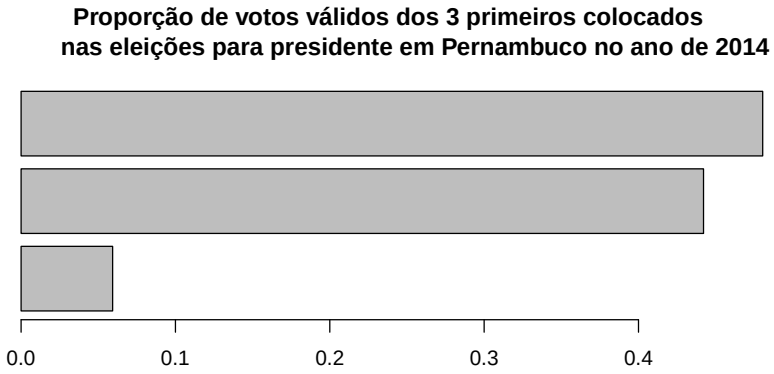


Figura 3.3: Gráfico simples sem nenhum ajuste com as barras na horizontal e com título

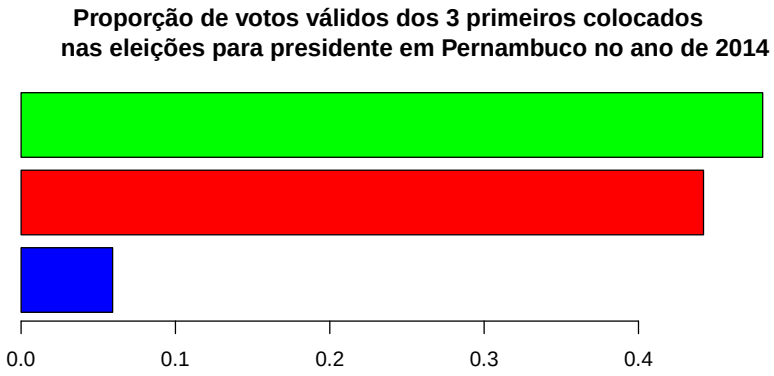


Figura 3.4: Gráfico com as barras coloridas, na horizontal e com título

```
#Grafico na horizontal, com titulo e cores para cada candidato
barplot(c(prop.votos.total.aecio,prop.votos.total.dilma,prop.votos.total.marina),
        horiz = TRUE, main = "Proporção de votos válidos dos 3 primeiros colocados
        nas eleições para presidente em Pernambuco no ano de 2014",
        col = c("blue","red","green"))
```

O argumento col simboliza as cores para os respectivos elementos do vetor dado como argumento, respeitando a ordem de cada elemento. Neste caso temos: azul, vermelho e verde representando respectivamente os candidatos Aécio, Dilma e Marina.

```
#Grafico na horizontal, com titulo e cores para cada candidato e legenda
#para cada cor.
barplot(c(prop.votos.total.aecio,prop.votos.total.dilma,prop.votos.total.marina),
        horiz = TRUE, main = "Proporção de votos válidos dos 3 primeiros colocados
        nas eleições para presidente em Pernambuco no ano de 2014",
        col = c("blue","red","green"),
        legend.text = c("Aecio Neves","Dilma Roussef","Marina Silva"))
```

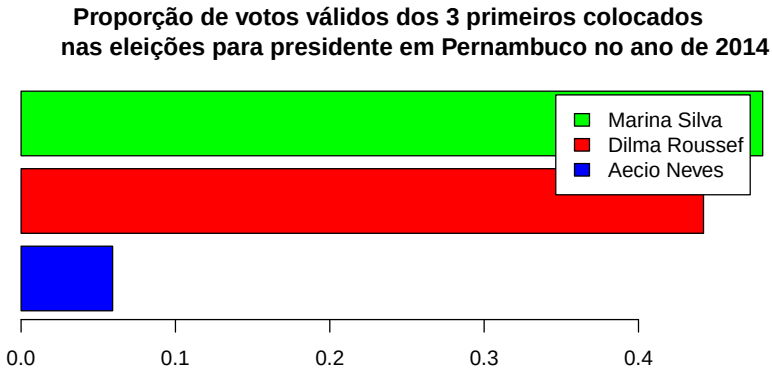


Figura 3.5: Gráfico com as barras coloridas, na horizontal, com título e legenda

O argumento legend.text cria uma caixa de legendas utilizando um vetor de nomes, onde cada elemento representa o nome de um elemento do vetor utilizado para criar as barras.

```
#Grafico na horizontal, com titulo e cores para cada candidato e legenda
#para cada cor e sem cobrir as barras.
barplot(c(prop.votos.total.aecio,prop.votos.total.dilma,prop.votos.total.marina),
        horiz = TRUE, main = "Proporção de votos válidos dos 3 primeiros colocados
        nas eleições para presidente em Pernambuco no ano de 2014",
        col = c("blue","red","green"),args.legend= list(x =0.5, y =1.55, bty="n"),
        legend.text = c("Aecio Neves","Dilma Roussef","Marina Silva"))
```

O argumento args.legend é utilizado para marcar as opções de modificação da caixa onde ficam as legendas. x indica a coordenada do eixo x onde se encontra o fim da caixa(parte horizontal), y é a coordenada do eixo y onde se encontra o topo da caixa(parte vertical) e bty é referente ao tipo de caixa. Como utilizamos bty = "n", omitimos a caixa de legenda. **Exemplo 2:**

Vamos calcular a colocação dos 3 candidatos anteriores nas cidades que pertencem a região metropolitana de recife e as que não pertencem e fazer um gráfico de barras para visualizar esta informação. Para isso, precisamos criar uma variável que indique as cidades que pertencem a RMR. Utilizaremos o valor 1 caso a cidade pertença a RMR e 0, caso contrário.

```
rm = idhm$municipio
```

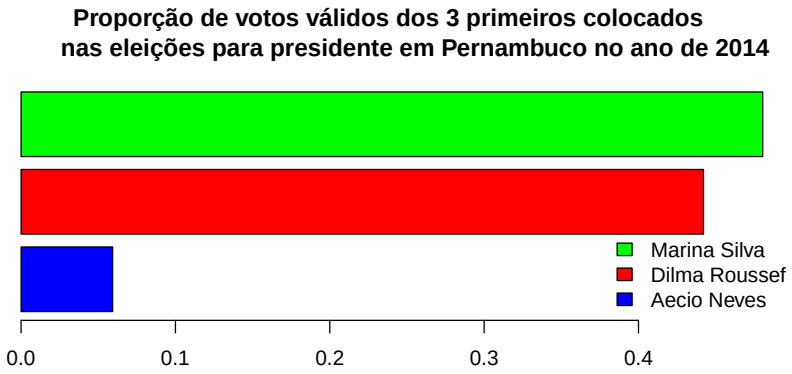


Figura 3.6: Gráfico com as barras coloridas, na horizontal, com título e legenda sem cobrir as barras

```
rm[rm=="OLINDA"|rm=="PAULISTA"|rm=="RECIFE"|rm=="JABOATÃO DOS GUARARAPES"|
   rm=="IGARASSU"|rm=="ABREU E LIMA"|rm=="CAMARAGIBE"|
   rm=="CABO DE SANTO AGOSTINHO"|rm=="IPOJUCA"|
   rm=="SÃO LOURENÇO DA MATA"|rm=="ARAÇOIABA"|rm=="ILHA DE ITAMARACÁ"|
   rm=="ITAPISSUMA"|rm=="MORENO"] = "1"

rm[rm!="1"] = "0"
```

Agora que já temos a variável que indica se uma cidade pertence ou não a RMR podemos calcular a proporção de votos válidos para os candidatos em cada parte do estado.

```
cand.rm = c(sum(idhm$Aecio.Neves[rm=="1"])/sum(idhm$Total[rm=="1"]),
            sum(idhm$Dilma.Rousseff[rm=="1"])/sum(idhm$Total[rm=="1"]),
            sum(idhm$Marina.Silva[rm=="1"])/sum(idhm$Total[rm=="1"]))

cand.int = c(sum(idhm$Aecio.Neves[rm=="0"])/sum(idhm$Total[rm=="0"]),
            sum(idhm$Dilma.Rousseff[rm=="0"])/sum(idhm$Total[rm=="0"]),
            sum(idhm$Marina.Silva[rm=="0"])/sum(idhm$Total[rm=="0"]))

nomes = c("Aécio Neves","Dilma Rousseff", "Marina Silva")

names(cand.rm) = nomes

names(cand.int) = nomes
```

Criamos as variáveis com a proporção de votos válidos para os 3 canditados em cada região. O vetor nomes é utilizado para dar nome a cada elemento dos vetores. Para dar os nomes utilizamos a função names().

```
barplot(cbind(cand.rm,cand.int), beside = TRUE,
        names.arg = c("Região Metropolitana","Demais cidades"),
        col =c("blue","red","green"),
        args.legend = list(x = 7, y = -0.1, bty = "n", ncol = 3),
        legend.text = nomes,
        main = "Proporção de votos válidos para os 3 primeiros colocados
        na região metropolitana e demais cidades")
```

Note que agora não estamos mais utilizando concatenar (c) para criar o objeto dos dados e sim cbind. Esse procedimento é necessário pois estamos interessados em criar um gráfico com dois subconjuntos (RMR e demais cidades). Outros argumentos que não utilizamos no gráfico anterior são:
beside: Indica se as categorias serão postas lado a lado. Só funciona caso o conjunto de dados utilizado como argumento seja uma matriz.
names.arg: Fornece o nome dos subconjuntos ou categorias que ficarão por baixo das barras.
ncol: Informa o número de colunas da caixa de legendas. Só pode ser utilizado dentro do argumento args.legend.

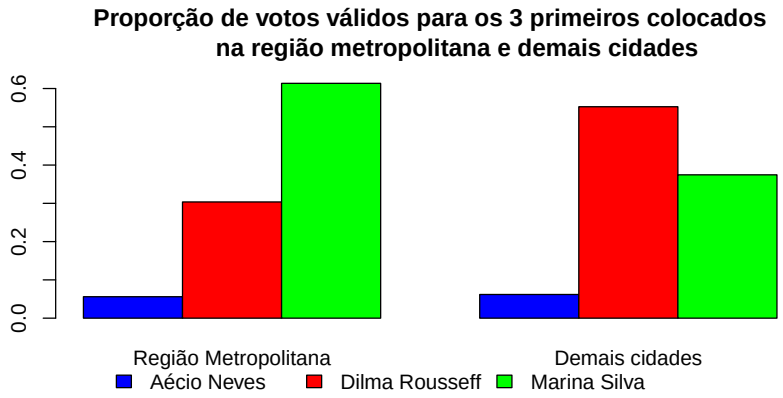


Figura 3.7: Gráfico de barras com vários subconjuntos.

Gráfico de setores

pie(objeto, labels)

Exemplo 1:

Iremos fazer um gráfico de setores para a população total do estado de Pernambuco no ano de 2010 categorizado entre RMR e as demais cidades do estado.

```
pie(c(sum(idhm$Populacao_2010[rm=="1"]),
      sum(idhm$Populacao_2010[rm=="0"])),
    labels = c("Região metropolitana","Demais cidades"),
    main = "População total do estado de Pernambuco em 2010.",
    col = c("darkblue","lightblue"))
```

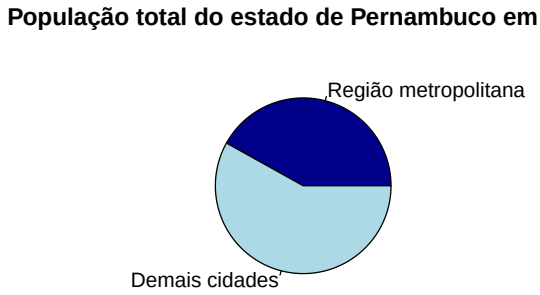


Figura 3.8: Gráfico de setores simples

Exemplo 2:

Faremos o mesmo gráfico anterior, mas agora mostraremos a % e criaremos uma legenda para cada setor. Para isso temos que criar um objeto com as respectivas porcentagens.

```
pop.per = c(sum(idhm$Populacao_2010[rm=="1"])/sum(idhm$Populacao_2010),
               sum(idhm$Populacao_2010[rm=="0"])/sum(idhm$Populacao_2010))

pop.per = round(pop.per*100,2)

pie(c(sum(idhm$Populacao_2010[rm=="1"]),
      sum(idhm$Populacao_2010[rm=="0"])),labels = pop.per,
    main = "População total do estado de Pernambuco em 2010.",
    col = c("darkblue","lightblue"))

legend("topright",c("Região metropolitana", "Demais cidades"),
       fill = c("darkblue","lightblue"), cex = 1)
```

Note que agora tivemos que fazer a legenda fora da função que faz o gráfico. Isso se deve pelo fato de que na função pie não temos a opção de legenda. A sintaxe da função para criar legendas é:

legend(x, y, legenda, fill, bty, cex)

x: Indica a coordenada do eixo x.
y: Indica a coordenada do eixo y.
legenda: É o vetor com o nome dos setores.
fill: Objeto que irá preencher as caixas das legendas com as cores indicadas.
bty: Tipo de borda da caixa da legenda. Se bty = ”n”a caixa é formada sem as bordas.
cex: Tamanho da caixa da legenda dada em proporção. **Observação: Os argumentos x e y podem ser omitidos utilizando como argumento posições pré-definidas tais como: ”topleft”, ”topright”, ”center”, ”left”etc.**

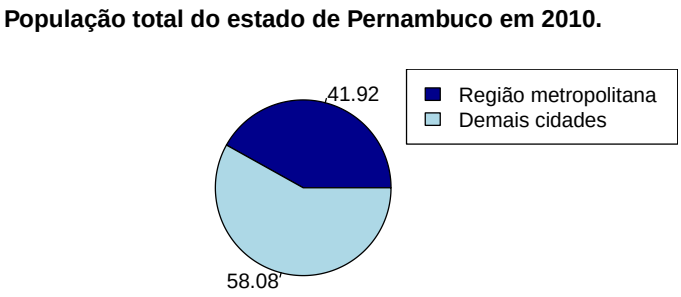


Figura 3.9: Gráfico de setores com porcentagens e legenda

Exemplo 3:
Faremos o mesmo gráfico do exemplo 1 em 3D. Para tal teremos que utilizar a função pie3D do pacote plotrix. A sintaxe da função pie3D é muito parecida com a pie, tendo como novo somente o argumento explode que indica o quanto as "fatias"estarão separadas.

```
library("plotrix")

pie3D(c(sum(idhm$Populacao_2010[rm=="1"]),
      sum(idhm$Populacao_2010[rm=="0"])),
      labels = c("Região metropolitana","Demais cidades"),
    main = "População total do estado de Pernambuco em 2010.",
    col = c("darkblue","lightblue"), explode = 0.1)
```

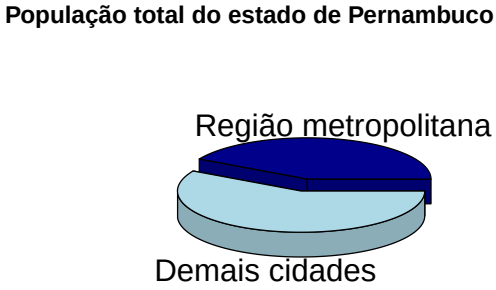


Figura 3.10: Gráfico de setores em 3D

Diagrama de caixas (Boxplot)

boxplot(objeto1, objeto2 ,..., names, horizontal, col, main)

objeto: Cria um boxplot para cada objeto fornecido. Os objetos devem ser vetores.
names: Será o nome de cada vetor.

horizontal: Indica se as caixas estarão na horizontal. Por padrão horizontal = TRUE.
col: Vetor de cores para cada objeto.
main: Título do gráfico.
Exemplo 1:
Vamos análisar o índice de desenvolvimento humano municipal das cidades do estado de Pernambuco dos três anos em que foram calculados.

```
boxplot(idhm$IDHM_1991,idhm$IDHM_2000,idhm$IDHM_2010,
        names=c("1991","2000","2010"),
        main = "IDHM do estado de PE nos anos de 1991, 2000 e 2010")
```

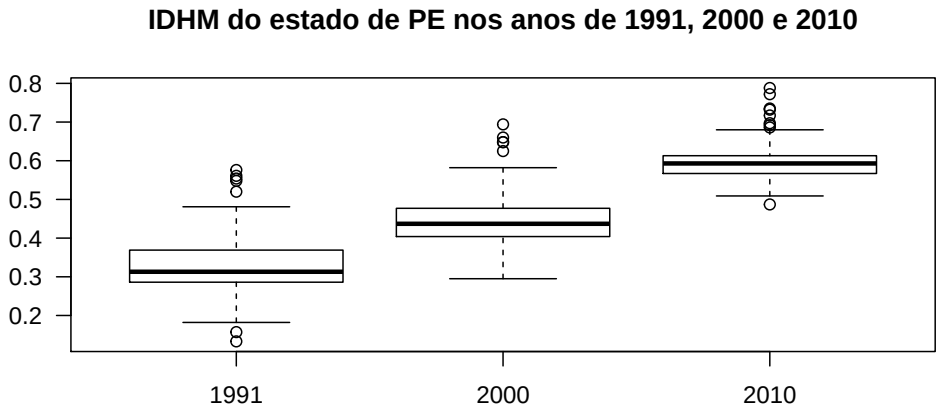


Figura 3.11: Boxplot do IDHM de PE ao longo dos anos

Exemplo 2:
Vamos análisar o índice de desenvolvimento humano municipal das cidades do estado de Pernambuco categorizando pelos que pertencem ou não a região metropolitana de Recife.

```
boxplot(idhm$IDHM_2010[rm=="1"],idhm$IDHM_2010[rm=="0"],
        names = c("Região Metropolitana", "Demais cidades"),
        main = "IDHM do estado de PE em 2010")
```

Note que no segundo boxplot existem alguns outliers. Para identifica-los vamos utilizar a seguinte função: **boxplot.stats(objeto)** onde $out nos indica os outliers.

```
boxplot.stats(idhm$IDHM_2010[rm=="0"])
idhm$municipio[idhm$IDHM_2010==0.680]
idhm$municipio[idhm$IDHM_2010==0.677]
idhm$municipio[idhm$IDHM_2010==0.788]
idhm$municipio[idhm$IDHM_2010==0.487]
idhm$municipio[idhm$IDHM_2010==0.697]
```

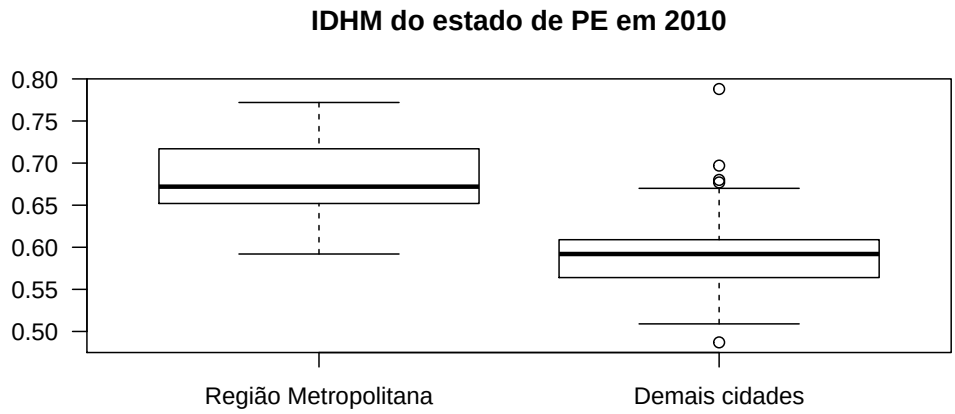


Figura 3.12: Boxplot do IDHM de PE em 2010

Diagrama de dispersão e gráfico de linhas

plot(x,y,type,pch,col,xlim,ylim,xlab,ylab)

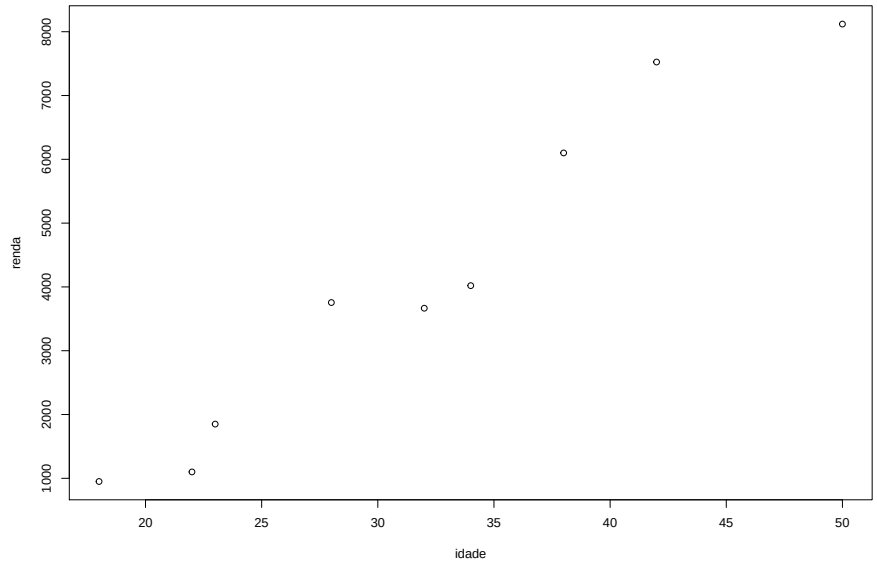
x: Variável do eixo x.
y: Variável do eixo y.
type: Tipo de ligação dos pontos.
pch: Ícone dos pontos.
col: Cor dos pontos.
xlim: Vetor que indica os limites do eixo x.
ylim: Vetor que indica os limites do eixo y.
xlab: Nome para a variável do eixo x.
ylab: Nome para a variável do eixo y.

Para os exemplos utilizaremos os seguintes dados:

```
escolaridade = c(8,5,6,2,4,3,8,6,7)
renda = c(8120,3666,4020,950,1100,1850,7525,3755,6100)
idade = c(50,32,34,18,22,23,42,28,38)
genero = c("F","M","M","F","F","M","M","M","F","F")
setor = c("privado","privado","publico","privado",
"privado","privado","publico","publico","privado")
```

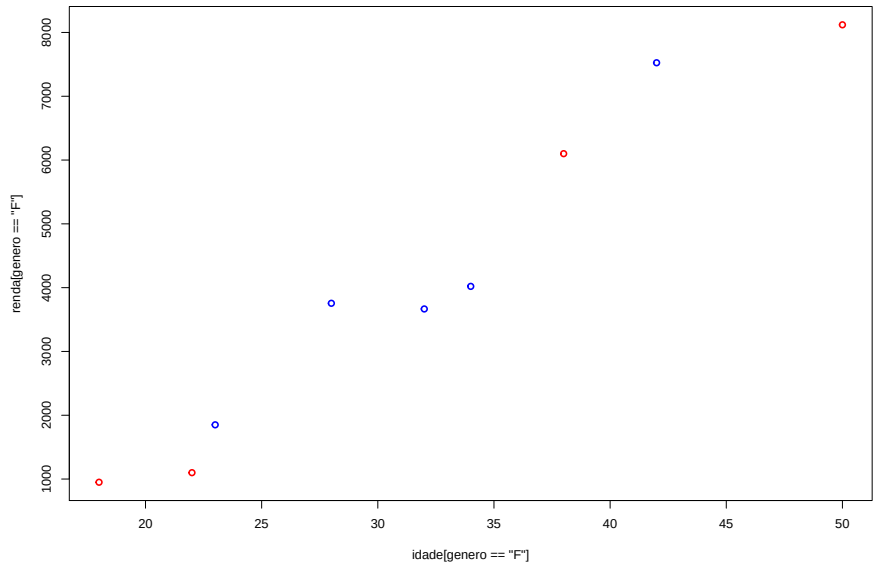
Exemplo 1:
Vamos observar como se comporta a renda em função da idade.

```
plot(idade,renda)
```



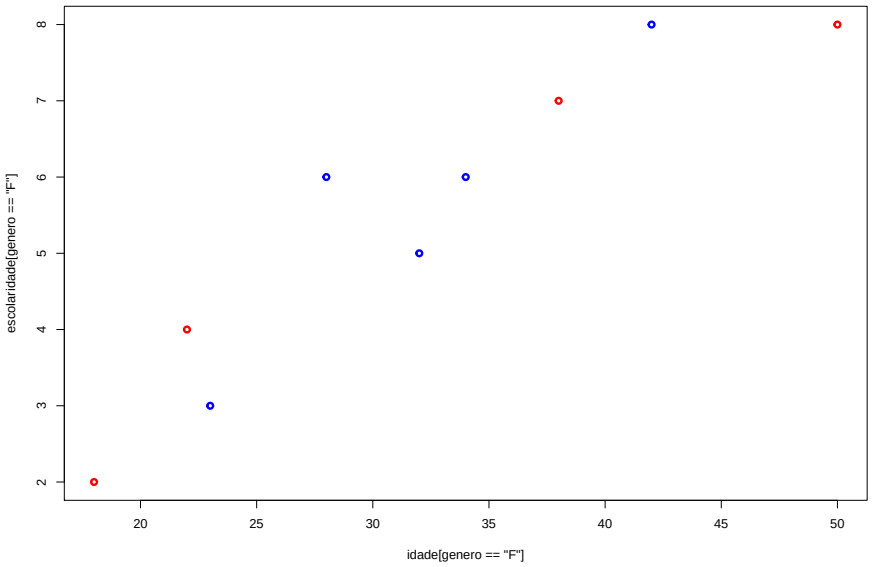
Exemplo 2:
Vamos fazer o mesmo gráfico anterior mas agora diferenciado entre os gêneros. Para isso precisaremos plotar dois gráficos, sendo que para adicionar pontos em um gráfico já existente utilizaremos a função points que possui os mesmos argumentos da função plot.

```
plot(idade[genero=="F"],renda[genero=="F"], col = "red", lwd = 2)
points(idade[genero=="M"],renda[genero=="M"], col = "blue", lwd = 2)
```



Exemplo 3:
Vamos averiguar se nesta amostra a escolaridade aumenta de acordo com a idade de cada individuo. Iremos utilizar o mesmo raciocínio do exemplo anterior.

```
plot(idade[genero=="F"],escolaridade[genero=="F"], col = "red", lwd = 3)
points(idade[genero=="M"],escolaridade[genero=="M"], col = "blue", lwd = 3)
```

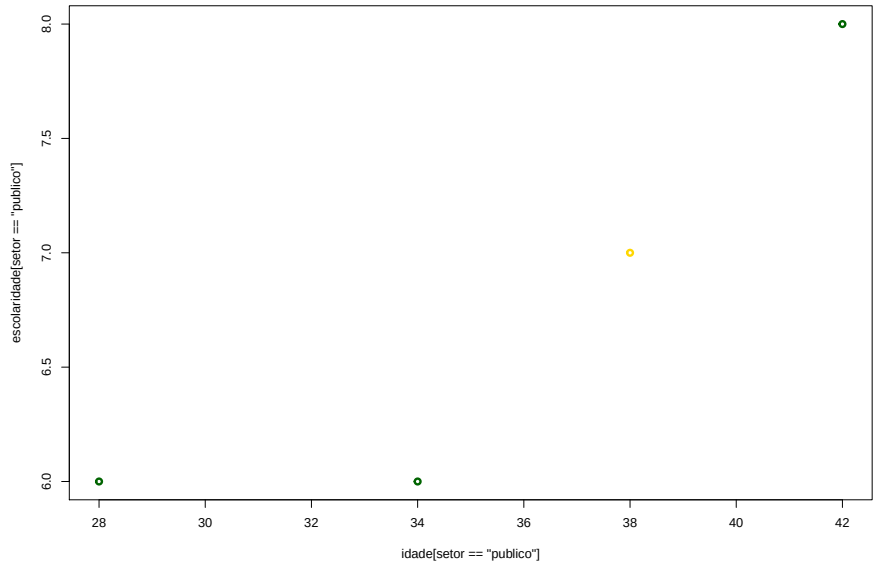


Exemplo 4:
Agora, averiguaremos como se comporta a escoralidade em função da idade sendo que agora obsevaremos o setor em que cada individuo trabalha.

```
plot(idade[setor=="publico"],escolaridade[setor=="publico"],
col = "darkgreen", lwd = 3)
points(idade[setor=="privado"],escolaridade[setor=="privado"],
col = "gold", lwd = 3)
```

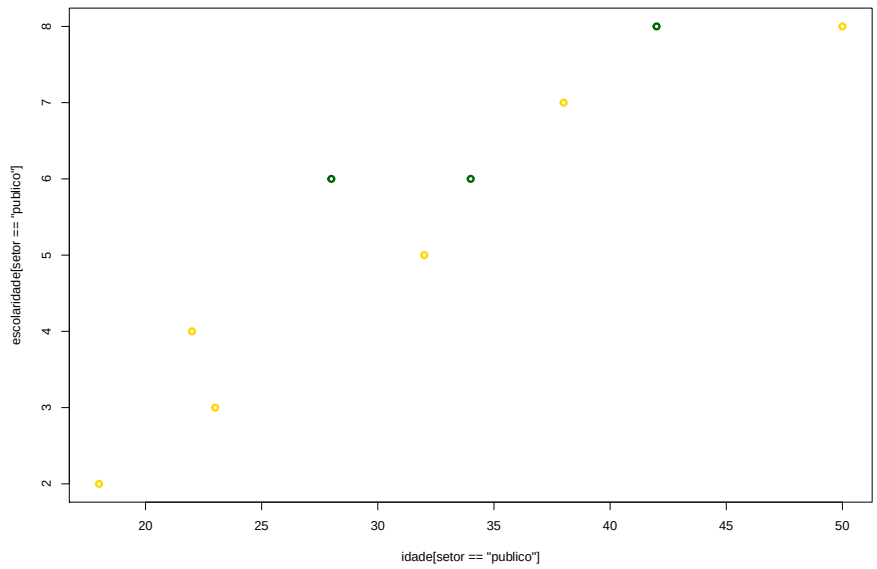
Note que só foram inseridos 4 pontos no gráfico, isso ocorreu devido o intervalo utilizado para fazer o **plot** ser muito pequeno, fazendo com que os pontos da função **points** não aperececem no gráfico inicial. Para resolver este problema temos que estipular limites para o eixo x e y. Para isso, faremos uma simples análise do sumário das variáveis que compoem o eixo x e y afim de identificar os valores de máximo e mínimo, viabilizando a criação de um intervalo para que todas as informações sejam visíveis.

```
summary(idade)
summary(escolaridade)
```



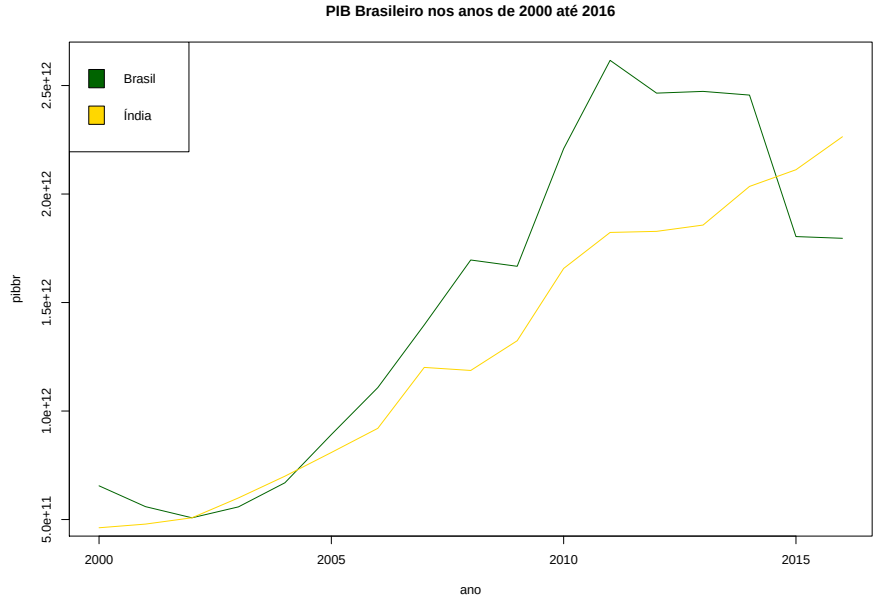
Agora que já sabemos o intervalo onde cada variável se encontra podemos fazer o gráfico proposto corretamente.

```
plot(idade[setor=="publico"],escolaridade[setor=="publico"],
col = "darkgreen", lwd = 3,xlim = range(18:50),
 ylim = range(2:8))
points(idade[setor=="privado"],escolaridade[setor=="privado"],
col = "gold", lwd = 3)
```



Exemplo 5:
Faremos um gráfico de série temporal do PIB do Brasil e da Índia. Para fazer um gráfico típico de série temporal basta adicionar o argumento **type** e faze-lo receber "l", veja abaixo:

```
pibbr = c(655400000000,559400000000,508000000000,558300000000,669300000000,
891600000000,1108000000000,1397000000000,1696000000000,
1667000000000,2209000000000,2616000000000,2465000000000,
2473000000000,2456000000000,1804000000000,1796000000000)
pibin = c(462100000000,479000000000,508100000000,599600000000,699700000000,
808900000000,920300000000,1201000000000,1187000000000,
1324000000000,1657000000000,1823000000000,1828000000000,
1857000000000,2035000000000,2112000000000,2264000000000)
ano = c(2000:2016)

plot(ano,pibbr,type = "l",col = "darkgreen",
main = "PIB Brasileiro nos anos de 2000 até 2016")
points(ano,pibin, type = "l", col = "gold")
legend("topleft",c("Brasil","Índia"), fill = c("darkgreen","gold"))
```



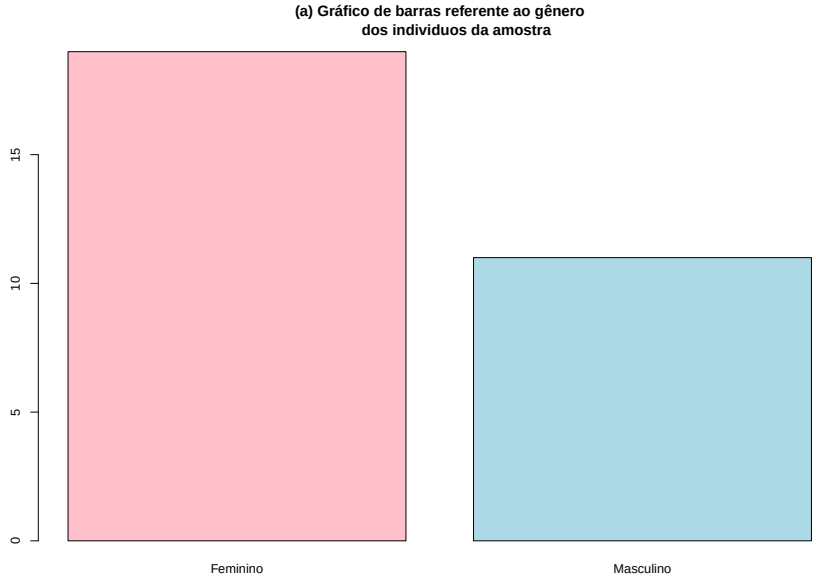
Exercícios

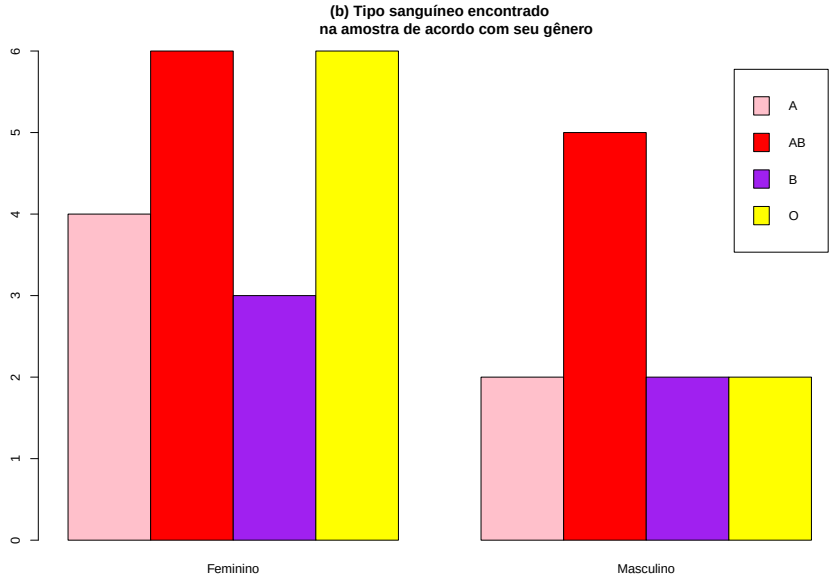
Exercício 1: Faça os seguintes gráficos utilizando os dados a seguir:

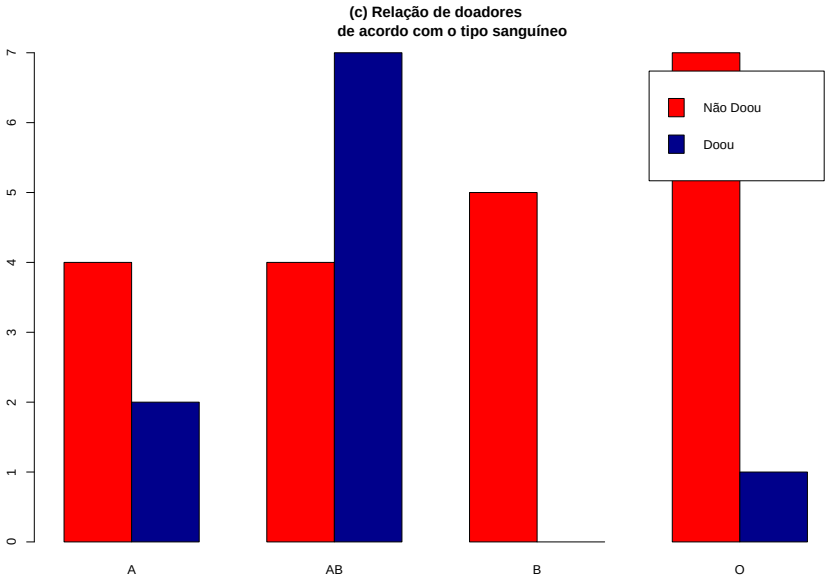
```
genero = c("F", "M", "F", "F", "F", "F", "F", "M", "F",
           "F", "M", "F", "M", "M", "M", "F", "F", "F",
           "M", "M","F", "F", "F", "F","M", "M", "M",
           "F", "F", "F")

sangue = c("AB", "AB", "A",  "A",  "A",  "AB", "O",
           "AB", "AB", "O",  "B",  "B",  "AB", "B",
           "A",  "AB", "O",  "O",  "O",  "O",  "O",
           "AB", "O",  "B",  "AB", "AB", "A","B",
           "AB", "A")
```

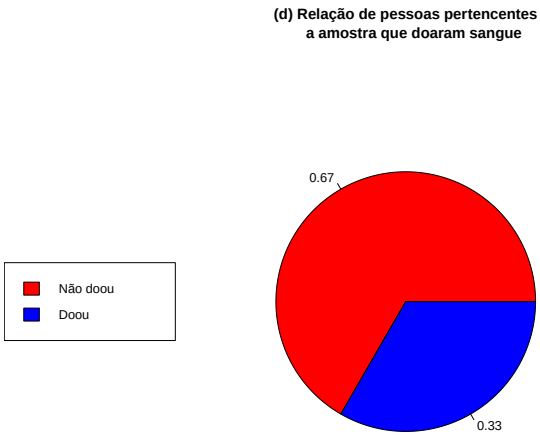
```
doou = c("Sim", "Sim", "Sim", "Nao", "Sim", "Sim",
         "Sim", "Sim", "Nao", "Nao", "Nao", "Nao",
         "Nao", "Nao", "Nao", "Nao", "Nao", "Nao",
         "Nao", "Nao", "Nao", "Sim", "Nao", "Nao",
         "Sim", "Nao", "Nao", "Nao", "Sim", "Nao")
```

Dica: Utilize a função **table**.









Exercício 2: Fazer os seguintes gráficos para os dados a seguir:

```
renda.anual = c(21950,22500,25000,24200,42000,6000,19750,
                110000,23000,22711,35000,81000,72000,62000)

idade = c(30,25,42,35,28,18,25,55,24,35,29,49,45,42)

area.trabalho = c("rural","rural","rural","rural","urbano",
                  "urbano","urbano","urbano","urbano",
                  "rural","urbano","urbano","urbano","urbano")
```

3.2 Inferência Estatística

Inferência é a área da Estatística que busca, a partir de valores observados (amostra) representar o todo (população). A inferência estatística se difere da análise descritiva pois, na inferência busca-se encontrar um "padrão", diferente de só descrever os fatos, como ocorre na análise descritiva.

3.2.1 Testes de hipotéses

Nesta sessão serão abordados os testes mais encontrados na literatura e como utiliza-los no R e também sua interpretação.

Teste T - Comparação de médias

Testa se as médias de duas variáveis são iguais. Muito utilizado para comparação de desempenho etc.

$$t.test(x, y, alternative, conf.level, paired)$$

$$H_0 : \mu_x = \mu_Y$$

Onde:
x: Variável 1. Deve ser da classe "numeric"
y: Variável 2. Deve ser da classe "numeric"
alternative: Indica o tipo da hipotése alternativa.
"two.sided"indica $\mu_x \neq \mu_y$; "greater"indica $\mu_x > \mu_y$; "less"indica $\mu_x < \mu_y$.
Por padrão alternative = "two.sided".
conf.level: Nível de confiança para o intervalo de confiança do teste. Por padrão conf.level = 0.95.
paired: Indica se o teste é pareado ou não.
Por padrão paired = FALSE.

Pré-suposições: Para aplicação do teste T, algumas pré-suposições devem ser cumpridas, que são:

- As variâncias devem ser iguais.
- AS distribuições devem de probababilidade das variáveis devem ser normais.

Ou seja, para aplicarmos o teste T precisamos testar duas hipotéses antes, se as variáveis possuem a mesma variância e se são gaussianas.

Teste F - Igualdade de variâncias

$$var.test(x, y, alternative, conf.level)$$

$$H_0 : \sigma_x^2 = \sigma_y^2$$

Onde:
x: Variável 1. Deve ser da classe "numeric"
y: Variável 2. Deve ser da classe "numeric"

alternative: Indica o tipo da hipotése alternativa.
conf.level: Nível de confiança para o intervalo de confiança do teste. Por padrão conf.level = 0.95.

Teste Shapiro Wilk para normalidade

$$shapiro.test(x)$$

$H_0 : x$ têm distribuição normal vs $H_1 : x$ não têm distribuição normal

Onde:
x: Variável a ser testada. Deve ser da classe "numeric".

Agora que sabemos como testar as pré-suposições para a aplicação do teste T, vamos fazer um exemplo.
Exemplo 1: Vamos testar se as variáveis x e y possuem a mesma média.

```
x = rnorm(50)
y = rnomr(50,mean - 5, sd = 1)
```

O comando "rnorm"gera aleatóriamente uma amostra a partir de uma distribuição normal. Por padrão mean = 0 e sd = 1(Normal padrão).

```
var.test(x,y) #Testando se x e y possuem mesma variância

F test to compare two variances

data:  x and y
F = 1.1804, num df = 49, denom df = 49, p-value = 0.5637
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.6698742 2.0801661
sample estimates:
ratio of variances
          1.180445
```

Como o p-valor é maior que 0.05(nível de significância utilizado por padrão), não rejeitamos H_0, ou seja, x e y possuem mesma variância. Vamos seguir para o próximo passo.

```
shapiro.test(x) #Testando se x é normal

Shapiro-Wilk normality test

data:  x
W = 0.99008, p-value = 0.948

shapiro.test(y) #Testando se y é normal

Shapiro-Wilk normality test

data:  y
W = 0.97851, p-value = 0.4909
```

Como o p-valor de ambos os testes são menores que 0.05, então não rejeitamos H_0, ou seja, x e y possuem distribuição normal. Ótimo ! Podemos utilizar o teste T.

```
t.test(x,y) # Comparando as médias de x e y

Welch Two Sample t-test

data:  x and y
t = -26.681, df = 97.333, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -5.319016 -4.582498
sample estimates:
mean of x mean of y
0.1530503 5.1038073
```

O p-valor neste caso foi menor que 2.2e-16(0.00000000000000022), sendo assim, rejeitamos H_0 ao nível de significância de 0.05.

E se quisermos testar se a média de y é maior que a de x? É muito simples, basta trocar a hipótise alternativa. Como a hipótise nula é estipulada em função da variável x, devemos utilizar $H_1 : \mu_x < \mu_y$.

```
t.test(x,y, alternative = "less")

Welch Two Sample t-test

data:  x and y
t = -26.681, df = 97.333, p-value < 2.2e-16
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
      -Inf -4.642613
sample estimates:
mean of x mean of y
0.1530503 5.1038073
```

Novamente o p-valor foi muito pequeno, rejeitando H_0 ao nível de significância de 0.05, ou seja, a média de x é muito inferior a média de y. Para termos uma noção gráfica do problema faça o boxplot das duas variáveis.

Como pudemos notar, a aplicação do teste T requer algumas pré-suposições "fortes"oque pode torna-lo inviável em muitos casos. Mas calma, existem testes similares tão poderosos quanto o teste T.

Teste de Wilcoxon-Maan-Whitney

Quando temos uma amostra muito pequena (n ¡ 50) os testes de normalidade tendem a errar mais, sou seja, possa ser que a variável de estudo não seja normal mas o teste diga que é, gerando implicações erroneas. Uma saída para este problema é o teste de Wilcoxox-Maan-Whitney. Oque este teste faz é basicamente o mesmo do teste T, porém ele não usa parâmetros da distribuição de probabilidade, oque o torna menos poderoso, o que pode ser visto como o preço a se pagar por não podermos utilizar o teste T.

$$wilcox.test(x, y, alternative, paired)$$

Onde os argumentos possuem os mesmos significados do teste T.

Atenção: Se paired = TRUE, será utilizado o teste de Wilcoxon para amostras pareadas, caso contrário, será feito o teste de Maan-Whitney para amostras independentes.

Exemplo: Vamos testar o desempenho de uma droga utilizada para aumentar o efeito de certa substância no corpo de equinos. Suponha que x2 siga uma distribuição T-Student com 2 g.l. e y2 siga uma normal padrão com o mesmo tamanho amostral. Queremos saber se elas possuem o mesmo efeito.

Primeiro de tudo vamos testar se elas são normais, pois se forem talvez possamos utilizar o teste T.

```
x2 = c(-1.86207107, -3.60011179,  0.28272644,  0.16534428,
    -1.68032082,  1.90796732,  2.33243093, -0.25985349,
      0.72470610,  0.02867185,  1.53790545,  1.29315464,
      -2.46519014, -0.12045813, -0.74972548)
y2 = c(1.6109836, -0.4901811,  1.6301452,  0.5101216,
       -0.1337461, -0.8469878, -0.6615919,  0.1340608,
        0.6062589, -0.7680674, -1.1309730, -0.4350395,
        -0.3230224 -1.4337886 -0.9710675)

shapiro.test(x2)
shapiro.test(y2)
```

Note que o resultado para o teste de x2 nos diz que que ela possui distribuição normal, porém, este vetor foi gerado a partir de uma distribuição T-Studente com 2 g.l. e neste caso o teste errou.

Já que uma das suposições para aplicação do teste T foi quebrada, vamos utilizar a alternativa que temos, que é o Teste de Maan-Whitney já que nossas variaveis são independentes.

```
wilcox.test(x2,y2)

Wilcoxon rank sum test

data:  x2 and y2
W = 125, p-value = 0.6236
alternative hypothesis: true location shift is not equal to 0
```

Teste Qui-quadrado de Pearson

O teste qui-quadrado é amplamente utilizado em todas as áreas do conhecimento graças a sua flexibilidade, já que possui 3 hipóteses distintas (homogenêidade, aderência e independência) utilizando a mesma estatística de teste.

Teste de Aderência

O teste qui-quadrado de Pearson para aderência testa se determinado conjunto de dados se adequa a uma específica distribuição de probabilidades dada pelo usuário. Muito utíl após fazer uma amostragem por exemplo.

$$H_0 : F_x(discreta) = F_y(discreta)$$

Sintaxe:

$$chisq.test(amostra, p)$$

Onde amostra é o vetor de frequências da amostra que coletamos e p é o vetor de probabilidades que esperamos que a nossa variável aleátoria possua.

Exemplo: Suponha que uma pessoa faça o lançamento de uma moeda 120 vezes e ele queira saber se a mesma é honesta. Utilizando a tabela a seguir faremos o teste para saber se a moeda é honesta.

Cara	Coroa
198	202

```
chisq.test(c(198/400,202/400), p = c(0.5,0.5))

Chi-squared test for given probabilities

data:  c(198/400, 202/400)
X-squared = 1e-04, df = 1, p-value = 0.992
```

Teste de independência

H_0: x é independente de y vs H_1: x é dependente de y

Em muitas situações queremos saber se duas variáveis possuem dependência entre si. Para sabermos essa informação em variáveis categoricas usualmente usamos o teste qui-quadrado de independência. Para aplicarmos este teste precisamos atender algumas pré-suposições antes que sâo:

- O teste exige uma tabela de contingência de tamanho no mínimo 2x2.
- Cada casela da tabela deve possuir valor maior ou igual a 5.
- A amostra deve ser de tamanho maior que 40.

Exemplo:

Suponha que temos a seguinte tabela representando a escolaridade e a faixa salarial de trabalhadores de determinada cidade e queremos saber se existe alguma associação entre estas duas variáveis.

	17 anos de estudo ou menos	mais de 17 anos de estudo
Renda abaixo de R$ 2500,00	45	5
Renda a partir de R$ 2500,00	35	42

Utilizando o teste qui-quadrado temos:

```
tabela = cbind(c(41,35),c(5,42))
chisq.test(tabela)

Pearson's Chi-squared test with Yates' continuity correction

data:  tabela
X-squared = 21.453, df = 1, p-value = 3.626e-06
```

Sendo assim, rejeitamos H_0 a 5% de significância. Portando, as variáveis renda e anos de estudo não são independentes.

Teste de Homogêneidade

Agora estamos interessados em testar se duas proporções advindas de populações diferentes são iguais. Para isso também podemos utilizar o teste qui-quadrado de Pearson.

$$H_0 : \text{p1} = \text{p2}$$

Síntaxe:

$$prop.test(numerador, denominador, alternative)$$

Onde:
numerador: É o vetor de numeradores.
denominador: É o vetor de denominadores.
alternative: É o tipo da hipótise alternativa.

Exemplo:
Suponha que em determinada Cidade de aproximadamente 150 mil habitantes o número de índividuos com problemas cardíacos era de 12 mil. Já em uma outra cidade, de aproximadamente 230 mil habitantes este número era de 18 mil. Um pesquisador está interessado em saber se a proporção de habitantes com problemas cardíacos é igual entre estas duas cidades, para isso ele utilizou o teste qui-quadrado de Pearson no software R fazendo o seguinte código:

```
 prop.test(c(12000,18000),c(150000,230000))

2-sample test for equality of proportions with continuity correction

data:  c(12000, 18000) out of c(150000, 230000)
X-squared = 3.7525, df = 1, p-value = 0.05273
alternative hypothesis: two.sided
95 percent confidence interval:
 -2.412897e-05  3.502390e-03
sample estimates:
    prop 1     prop 2
0.08000000 0.07826087
```

Logo, o pesquisador pôde concluir que a fatia da população com problemas cardíacos é igual em ambas as cidades.

3.3 Modelos de Regressão linear

Caso estejamos interessados em verificar se existe alguma relação entre duas ou mais variáveis, é oportuno estabelecer um modelo matemático fixando uma variável resposta(Y) e buscando explica-la por uma ou mais variáveis(X_i). Contudo, se a variável Y tiver teor aleátorio um modelo matemático determinístico não é o mais apropriado, oque nos leva a utilizar outros métodos para explicar a relação entre Y e as variáveis X_i que são os modelos estatísticos.

Nesta sessão, serão abordados os modelos de regressão linear e métodos para conseguir suas estimativas. A função para estimar os parâmetros do modelo linear é dada a por:

$$lm(Y \sim X_1 + X_2 + X_3 + ... + X_n, data)$$

onde:
Y: Variável resposta
$X_1, X_2, ..., X_n$**:** Variáveis explicativas
data: É o data.frame onde os dados se encontram. Por padrão este argumento é omitido.
Exemplo 1:

Em uma rede de 26 lojas de materiais de construção é vendido um tipo de telhado. Com o intuito de tentar prever o número de vendas desse produto foram coletados o número de clientes cadastrados nas 26 lojas (em milhares) e a quantidade de telhados vendidos (em milhares).

```
telhas <- c(79.3, 200.1, 163.2, 200.1, 146.0, 177.7,  30.9, 291.9, 160.0,
            339.4, 159.6,  86.3, 237.5, 107.2, 155.0, 291.4, 100.2, 135.8,
            223.3, 195.0,  73.4,  47.7, 140.7,  93.5, 259.0, 331.2)

clientes <- c(31, 55, 67, 50, 38, 71, 30, 56, 42, 73, 60, 44, 50, 39,
              55, 70, 40, 50, 62, 59, 53, 38, 43, 26, 75, 71)
```

Para analisar o modelo posteriormente devemos salva-lo em um objeto. Vamos salvar nosso modelo no objeto ex1. Para acessar as informações sobre o modelo estimado, utilizamos a função summary no objeto em que salvamos o modelo como segue:

```
ex1 <- lm(telhas~clientes)
summary(ex1)

Call:
lm(formula = telhas ~ clientes)

Residuals:
     Min       1Q   Median       3Q      Max
-102.180  -35.661    1.311   38.032  102.350

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -71.2084    40.5584  -1.756   0.0919 .
clientes      4.6564     0.7555   6.164 2.28e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 53.69 on 24 degrees of freedom
Multiple R-squared:  0.6128,Adjusted R-squared:  0.5967
F-statistic: 37.99 on 1 and 24 DF,  p-value: 2.282e-06
```

Veja que ao utilizar summary no objeto ao qual o modelo pertence temos várias informações que são:
Residuals: É o sumário dos resíduos do modelo. Note que a média foi omitida, isso se deve pois por definição o somátorio dos resíduos é zero.
Coefficients: Aqui se encontram as estimativas dos parâmetros, seu desvio padrão, a estatística T e o p-valor do teste de adequacidade dos parâmetros individuais dado pela estatística T.
Signif. codes: É o significado dos asteriscos ao lado do resultado do teste T.
Residual standard error: Indica o desvio padrão dos resíduos. Note que neste caso é bem alto, oque explica o baixo R^2.
Multiple R-squared: É o R^2.
Adjusted R-squared: É o R^2 ajustado.
F-statistics: É o valor da estatística F para aceitação do modelo que vem acompanhado de seu p-valor.

Veja que o intercpeto oteve um baixo valor na significância do teste, oque a um nível de significância de 5% nos levaria a retira-lo do modelo. Então vamos estimar o novo modelo sem o intercepto, para isso basta adicionar o termo -1 após a inserção das variáveis:

```
ex11 <- lm(telhas~clientes-1)
summary(ex11)
Call:
lm(formula = telhas ~ clientes - 1)

Residuals:
     Min       1Q   Median       3Q      Max
-105.502  -40.903   -4.301   18.104  102.871

Coefficients:
         Estimate Std. Error t value Pr(>|t|)
clientes   3.3755     0.2041   16.54 5.69e-15 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 55.88 on 25 degrees of freedom
Multiple R-squared:  0.9162,Adjusted R-squared:  0.9129
F-statistic: 273.4 on 1 and 25 DF,  p-value: 5.692e-15
```

Veja que agora o R^2 aumentou bastante, porém, isso não quer dizer que o nosso modelo seja bom. Observe que o desvio padrão dos resíduos aumentou. O R^2 aumentou pelo simples fato de estarmos retirando o efeito da média do modelo. Portanto, muito cuidado.

Bom, já que o modelo não é bom uma saída é tentar coletar dados de uma nova variável ou então desistir. Suponha que a empresa optou pela primeira opção e que agora tenhamos mais uma variável a ser estudada que é a quantidade de lojas concorrentes na região.

```
conc <- c(10, 8, 12,  7,  8, 12, 12,  5,  8,  5, 11, 12,  6, 10,
          10,  6, 11, 11,  9,  9, 13, 13,  9,  8,  8,  4)
```

Fazendo o novo modelo obtemos:

```
ex111 <- lm(telhas~clientes+conc)
summary(ex111)

Call:
lm(formula = telhas ~ clientes + conc)

Residuals:
     Min       1Q   Median       3Q      Max
-18.4136  -6.1499  -0.5683   6.2472  20.3185

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 186.6940    12.2587   15.23 1.66e-13 ***
clientes      3.4081     0.1458   23.37  < 2e-16 ***
conc        -21.1930     0.8028  -26.40  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 9.803 on 23 degrees of freedom
Multiple R-squared:  0.9876,Adjusted R-squared:  0.9866
F-statistic: 918.3 on 2 and 23 DF,  p-value: < 2.2e-16
```

Veja que agora os resíduos variam muito menos do que antes e que todos os testes obtveram um ótimo resultado.

Exemplo 2:

Para este exemplo vamos utilizar os dados da sessão de estatística descritiva na subsessão de gráficos de dispersão para observar a relação entre renda (Y) e as demais variáveis.

```
escolaridade = c(8,5,6,2,4,3,8,6,7)
renda = c(8120,3666,4020,950,1100,1850,7525,3755,6100)
idade = c(50,32,34,18,22,23,42,28,38)
genero = c("F","M","M","F","F","M","M","M","F","F")
setor = c("privado","privado","publico","privado",
"privado","privado","publico","publico","privado")
```

Estimando os parametros do modelo temos:

```
ex2 <- lm(renda~idade+escolaridade)
summary(ex2)
Call:
lm(formula = renda ~ idade + escolaridade)

Residuals:
   Min     1Q Median     3Q    Max
-711.0 -315.6  140.2  314.6  656.5

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)  -3608.89     689.35  -5.235  0.00195 **
idade          181.24      55.55   3.263  0.01719 *
escolaridade   358.15     270.20   1.326  0.23324
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 583.5 on 6 degrees of freedom
Multiple R-squared:  0.9636,Adjusted R-squared:  0.9515
F-statistic: 79.51 on 2 and 6 DF,  p-value: 4.807e-05
```

Veja que a escolaridade não explica bem a variável renda quando em conjunto com a variável idade. Isso ocorre pois estas variáveis são correlacionadas. Para calcular a correlação linear temos a seguinte função:

$$cor(X_1, X_2)$$

Calculando a correlação temos que as duas variáveis são fortemente correlacionadas.

```
> cor(escolaridade,idade)
[1] 0.9334035
```

Estimando o novo modelo sem a escolaridade:

```
 ex22 <- lm(renda~idade)
summary(ex22)
Call:
lm(formula = renda ~ idade)

Residuals:
    Min      1Q  Median      3Q     Max
-628.39 -527.99  -48.67  451.71  876.81

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -3850.75     699.78  -5.503 0.000904 ***
idade         249.97      20.98  11.913 6.68e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 614.3 on 7 degrees of freedom
Multiple R-squared:  0.953,Adjusted R-squared:  0.9463
F-statistic: 141.9 on 1 and 7 DF,  p-value: 6.678e-06
```

E se quisermos averiguar se há difereça entre o aumento da idade e da renda observando o genêro? Basta utilizar variáveis dummy em nosso modelo. Para utilizar este tipo de variável no comando do R basta que a mesma seja um fator.

```
ex222 <- lm(renda~idade*genero)
summary(ex222)

Call:
lm(formula = renda ~ idade * genero)
```

```
Residuals:
     1      2      3      4      5      6      7      8      9
-295.5 -552.8 -754.4  264.3 -552.0  131.5  528.2  647.5  583.2

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   -3662.26     930.79  -3.935 0.011020 *
idade           241.55      27.00   8.945 0.000291 ***
generoM       -1008.74    1836.10  -0.549 0.606374
idade:generoM    36.25      55.78   0.650 0.544465
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 691.7 on 5 degrees of freedom
Multiple R-squared:  0.9574,Adjusted R-squared:  0.9319
F-statistic: 37.48 on 3 and 5 DF,  p-value: 0.0007501
```

Observe que só foi mostrada a estimativa para o genêro M, isto ocorre dada a definicação da variável dummy. Sendo assim, podemos ver que não a diferenças já que os coeficientes extras não foram aprovados nos testes.