

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
Campus CERRO LARGO

PROJETO DE EXTENSÃO
Software R:
Capacitação em análise estatística
de dados utilizando um software livre.



Fonte: <https://www.r-project.org/>

Aula 1

Blog do projeto: <https://softwarelivrer.wordpress.com/equipe/>

Equipe:

Coordenadora:

Profe. Iara Endruweit Battisti (iara.battisti@uffs.edu.br)

Colaboradores:

Profa. Denize Reis

Prof. Erikson Kaszubowski

Prof. Reneo Prediger

Profa. Tatiane Chassot

Mestrando Felipe Smolski

Bolsista:

Djaina Rieger - aluna de Engenharia Ambiental (djaina.rieger@outlook.com)

Voluntárias:

Jaíne Frank

Jaqueline Caye

Sumário

1	Introdução	3
1.1	Download e instalação do R e Rstudio	3
1.2	Link para o curso de extensão do Software R – curso básico e avançado	3
1.3	Painéis	3
1.4	Help	4
1.5	Abrir arquivo de dados	4
1.6	Salvar arquivo de dados	5
1.7	Diretórios de trabalho	6
1.8	Instalação de pacotes	6
1.9	Operações	7
1.9.1	Operações Aritméticas	7
1.9.2	Operações Lógicas	8
2	Manipulação básica	9
2.1	Variáveis	9
2.1.1	Criação de variáveis	9
2.1.2	Conversão de uma variável	10
2.2	Primeiros passos	10
2.2.1	Comando <code>tapply</code>	12
2.2.2	Comando <code>'subset'</code>	13
2.3	Estrutura de dados	13
2.3.1	Vetores	13
2.3.2	Matrizes	16
2.3.3	Listas	16
2.3.4	Dataframes	18
3	Manipulação de banco de dados	18
3.1	Função <code>edit()</code>	18
3.2	Comando <code>head</code> , <code>tail</code>	19
3.3	Funções	20
3.4	Funções Matemáticas	22
3.5	Subconjuntos de Matrizes	24
3.6	Conversão de datas	24
4	REFERÊNCIAS BIBLIOGRÁFICAS	25

1 Introdução

O R é um ambiente voltado para análise de dados com o uso de uma linguagem de programação, frente a isso um conhecimento prévio dos princípios de programação facilita a compreensão da condução das análises aplicadas no software.

1.1 Download e instalação do R e Rstudio

versão 3.4.0 — www.r-project.org

Clique em Download (CRAN) - escolha o link de um repositório - clique no link do sistema operacional (Linux, Mac ou Windows) - clique em “install R for de first time” - Download.

versão 1.0.136 — www.rstudio.com/products/rstudio/download

Lembrando que:

- R é o software
- RStudio é uma ferramenta amigável para o R

1.2 Link para o curso de extensão do Software R – curso básico e avançado

<https://softwarelivrer.wordpress.com/links-para-download/>

1.3 Painéis

O RStudio é a interface que faz com que seja mais fácil a utilização da programação em R.

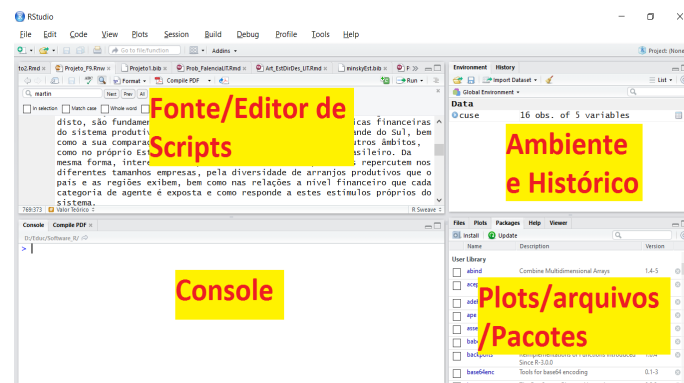


Figura 1: Painéis do Rstudio.

- **Fonte/Editor de Scripts:** se constitui do ambiente onde serão abertos os scripts previamente salvos nos mais diversos formatos ou mesmo sendo o local de visualização das bases de dados.
- **Console:** local onde será efetuada a digitação das linhas de código que serão interpretadas pelo R.
- **Ambiente e Histórico:** o ambiente será visualizado os objetos criados ou carregados durante a seção e; a aba History retoma os scripts digitados no console.
- **Plots/arquivos/Pacotes:** local onde podem ser acessados os arquivos salvos no computador pela aba *files*; a aba *Plots* carrega os gráficos e plotagens; a aba *Packages* contém os pacotes instalados em seu computador, onde são ativados ou instalados novos; em *Help* constam as ajudas e explicações dos pacotes e; *Viewer* visualiza documentos do tipo html.

1.4 Help

Acessamos o help por meio do comando "help()", através da aba "Help" ou ao clicar no nome do pacote.

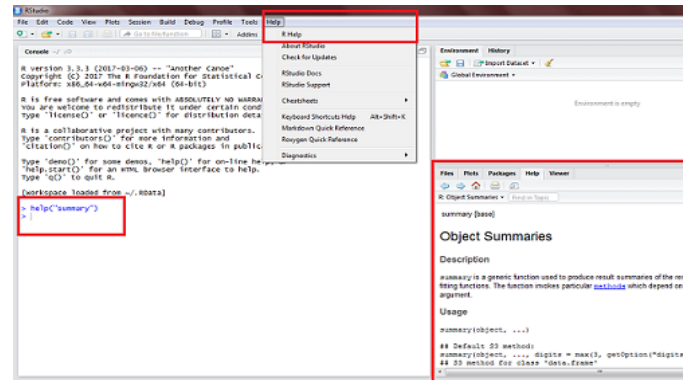


Figura 2: Comando Help.

Ou diretamente no console digitamos ? e a função desejada, exemplo: ?mean.

1.5 Abrir arquivo de dados

Dispondo de um banco de dados em uma planilha eletrônica (LibreOffice Calc. ou EXCEL), neste caso utilizaremos o arquivo 'arvores' como exemplo o banco de dados. Os dados provem de uma pesquisa com espécies de árvores registrando as variáveis diâmetro altura do peito (DAP) e altura. Dados cedidos pela professora Tatiane Chassot.

Neste caso:

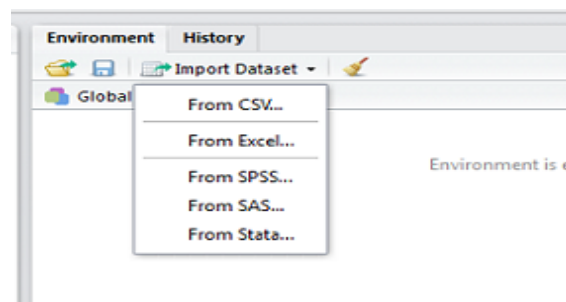


Figura 3: Aba Import Dataset.

Ainda pode-se utilizar a linha de comando da seguinte forma:

```
nome.do.arquivo = read_csv ("endereço")
```

Na caixa correspondente a File/Url colocamos o endereço virtual ou o local onde se encontra o arquivo.

Ao importar os dados, teremos tabela com as informações contidas no arquivo.

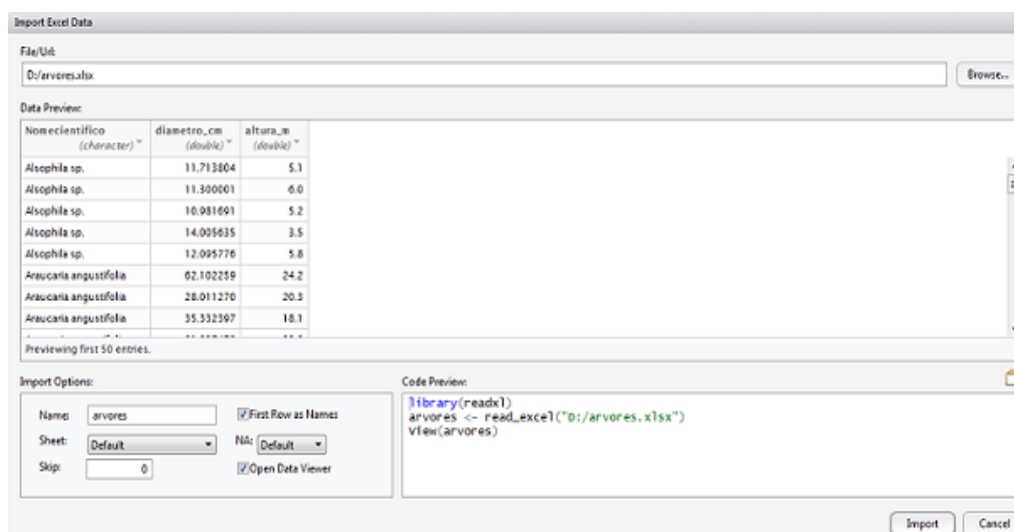


Figura 4: Caixa de informações do Import Data.

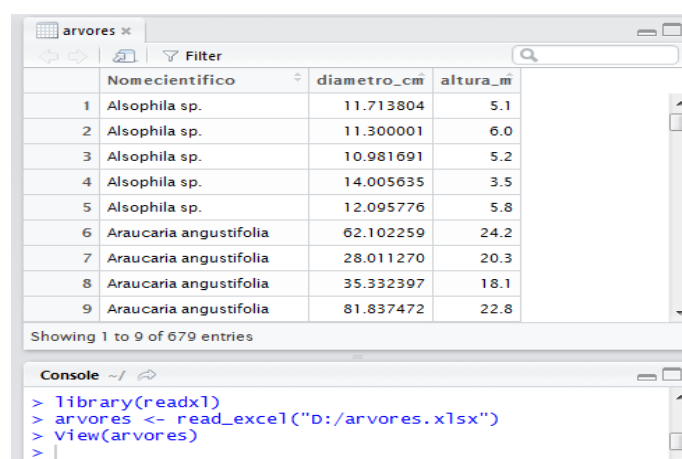


Figura 5:

1.6 Salvar arquivo de dados

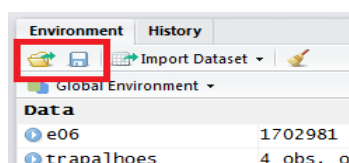


Figura 6: Atalho para salvar arquivo de dados.

O banco de dados que o R armazena na memória pode ser salvo, junto com todo o ambiente, usando o ícone de disquete na aba 'Environment' (salva como arquivo .RData), e depois carregado pelo ícone de pasta (Abrir dados...) na mesma aba.

Ou

Outra opção com mesmo efeito é utilizar o comando `save('nomeDoObjeto', file='nomeDoArquivo.RData')`

(o nome do objeto pode ser uma lista de objetos para salvar mais de um objeto do ambiente, `'list=('objeto1', 'objeto2')`). Para carregar um arquivo RData no ambiente, o comando é `load('arquivo.RData')`, desde que o arquivo esteja no diretório de trabalho do R.

Ou

Ainda podemos salvar uma matriz ou data frame, ou um banco de dados convertível para esse tipo de objeto através da função `write.csv('nomeDoObjeto', file='nomeDoArquivo.RData')`, estruturando. Porém é necessário lembrar que o nome do arquivo deve ser distinto do original, caso contrário o R salvará o novo arquivo sobre a versão antiga. A fim de salvar em formato Excel, seguimos o seguinte formato: `write.csv('nomeDoObjeto', file='nomeDoArquivo.csv')`

Portanto:

- `write.table(x,'file.txt')` (Salvando em arquivo .txt)
- `write.csv(x,'file.csv')` (Salvando em arquivo .csv)
- `save(x,'file.Rdata')` (Salvando em arquivo de dados em R)

1.7 Diretórios de trabalho

Os trabalhos efetuados via Rstudio, incluindo as bases de dados, os objetos, os resultados das fórmulas, os cálculos aplicados sobre os vetores e demais arquivos resultantes da utilização do programa podem ser salvos em seu diretório de arquivos. Após instalado o Rstudio destina um diretório padrão salvar estes arquivos, o qual pode ser verificado com o comando `getwd()`. Este caminho padrão, por sua vez, pode ser alterado via comando `setwd('C://file/path')`, onde o usuário escolhe a pasta desejada que ficará como padrão.

1.8 Instalação de pacotes

Em alguns situações, o uso de pacotes pode dar ao trabalho mais praticidade, e para isso se faz necessário efetuar a sua instalação. Precisamos ir até a painél dos pacotes em packages, selecionar a opção instalar e inserir o nome do pacote desejado na janela indicada. Ao selecionar a opção instalar, no console receberemos informações do procedimento e do sucesso do mesmo.

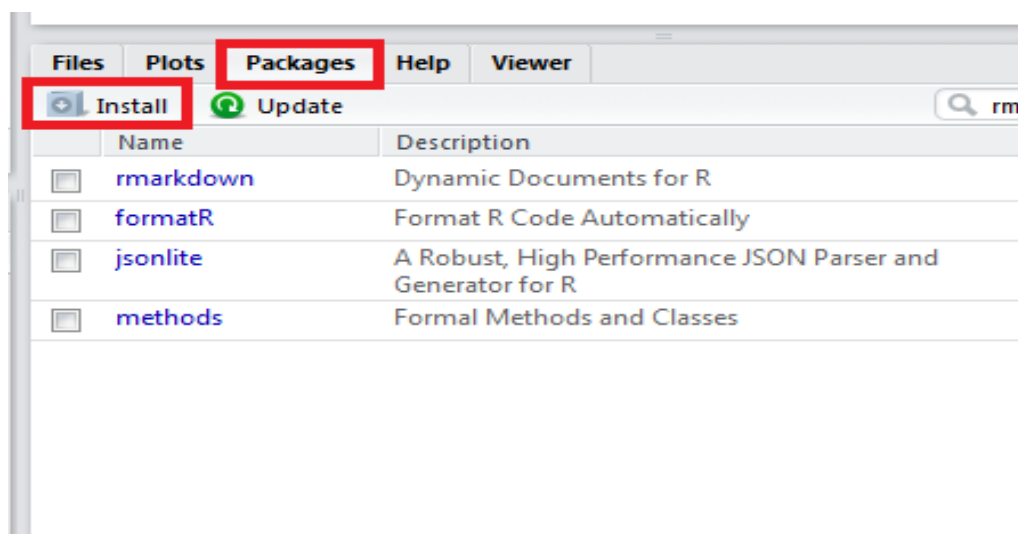


Figura 7: Instalação de pacotes.

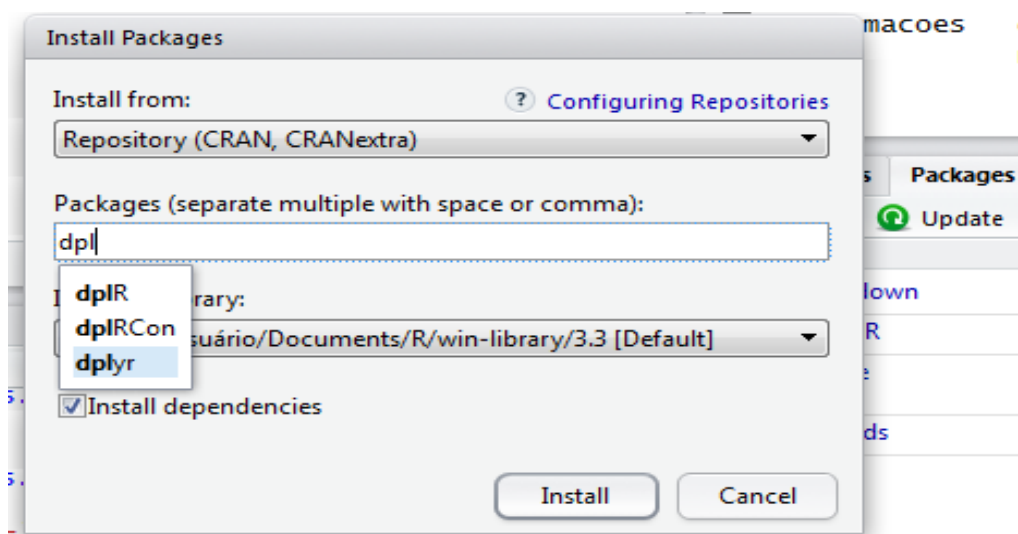


Figura 8: Caixa de informação de pacote a ser instalado.

A mesma função, para instalação de um pacote, pode ser efetuada via console: `install.packages('pacote')`. É importante ressaltar a função `library('nome do pacote')` que é utilizada no console para informar ao R e "carregar" o pacote que iremos utilizar. Podem ser instalados mais de um pacote ao mesmo tempo, como no exemplo `install.packages(c("tidyr", "devtools"))`.

1.9 Operações

1.9.1 Operações Aritméticas

A realização de uma operação aritmética no R acontece da seguinte forma: Onde a resolução das operações segue o padrão, ou seja, primeiro exponenciações, seguido de multiplicações e divisões, deixando por ultimo adições e subtrações, de acordo com a ordem que estão dispostas. Para alterar a prioridade da resolução de operações fazemos o uso do parenteses para destacar a operação que deve ser prioritaria na resolução.

```
# soma
19+26

## [1] 45

# subtração
19-26

## [1] -7

# divisão
4/2

## [1] 2

# multiplicação
4*2

## [1] 8
```

```
# exponenciação
4^2

## [1] 16

# prioridade de resolução
19 + 26 / 4 - 2 * 10

## [1] 5.5

((19 + 26) / (4 - 2)) * 10

## [1] 225

# raiz quadrada
sqrt(16)

## [1] 4

# Logaritmo
log(1)

## [1] 0
```

1.9.2 Operações Lógicas

O ambiente de programação Rstudio trabalha com algumas operações lógicas, que serão importantes na manipulação de bases de dados:

- **a==b** ('a' é igual a 'b')
- **a!=b** ('a' é diferente a 'b')
- **a b** ('a' é maior que 'b') **a b** ('a' é menor que 'b')
- **a = b** ('a' é maior ou igual a 'b') **a = b** ('a' é menor ou igual a 'b')
- **is.na** ('a' é missing - faltante)
- **is.null** ('a' é nulo)

Seguem alguns exemplos da aplicação das operações lógicas:

```
# maior que
2 > 1

## [1] TRUE

1 > 2

## [1] FALSE

# menor que
1 < 2

## [1] TRUE

# maior ou igual a
0 >= (2+(-2))
```



```
## [1] TRUE

# menor ou igual a
1 <= 3

## [1] TRUE

# conjunção
9 > 11 & 0 < 1

## [1] FALSE

# ou
6 < 5 | 0 > -1

## [1] TRUE

# igual a
1 == 2/2

## [1] TRUE

# diferente de
1 != 2

## [1] TRUE
```

2 Manipulação básica

2.1 Variáveis

2.1.1 Criação de variáveis

Primeiros Passos: a linguagem de programação R trata-se de uma linguagem orientada a objetos, ou seja, a todo tempo estamos criando diversos tipos de objetos e efetuando operações com os mesmos, como por exemplo criação de listas, bases de dados, união de bases de dados, data.frames e até mesmo mapas!

```
#Criando um objeto simples
objeto = "meu primeiro objeto" #enter
#Agora para retomar o objeto criado:
objeto #enter

## [1] "meu primeiro objeto"

#Pode ser efetuada uma operação:
a= 2+1
a

## [1] 3

#Remover um banco de dados
rm(a)
```

2.1.2 Conversão de uma variável

Para a aplicação de algumas funções é importante que cada variável esteja corretamente classificada, o que em alguns casos não ocorre durante o reconhecimento automático do R. Precisamos então reconhecê-la como variável texto, numérica ou fator. Além disso, a classe `ordered` se aplica a variáveis categóricas que podem ser consideradas ordenáveis.

```
idade=c('11', '12', '31')
nomes=c("Elisa", "Priscila", "Carol")
cep=c(98700000,98701000,98702000)
idade= as.numeric(idade)
idade

## [1] 11 12 31

cep = as.character(cep)
cep

## [1] "98700000" "98701000" "98702000"
```

2.2 Primeiros passos

Vamos realizar a importação de uma base de dados .csv, um arquivo onde as informações estão separadas por vírgulas, depositadas diretamente em um site. A função `head()` mostra as 6 primeiras colunas do arquivo para se ter uma noção do conteúdo. O comando 'summary' efetua o resumo dos dados, se for qualitativa mostra a frequência absoluta das categorias e se for quantitativa apresenta as categorias.

```
#Importando o arquivo de um site.
dados <- read.csv('http://www.stat.columbia.edu/~gelman/bda.course/nycData.csv',
                  sep=',', header = T)

#Visualizando as primeiras 6 colunas
head(dados)

##   X zipcode pop.total pop.white owner.occupied renter.occupied
## 1 1   10001   17310   10966         1970           6971
## 2 2   10002   84870   22140         4146          27366
## 3 3   10003   53673   41661         7830          21686
## 4 4   10004    1225     903          133           489
## 5 5   10005     884     595           35           489
## 6 6   10007    3522    2507          423           807
## median.household.income borough
## 1                40932 manhattan
## 2                24022 manhattan
## 3                60891 manhattan
## 4               101868 manhattan
## 5                79517 manhattan
## 6               112947 manhattan

#Para visualizar os nomes das colunas dos dados:
names(dados)

## [1] "X"                "zipcode"
## [3] "pop.total"        "pop.white"
```

```
## [5] "owner.occupied"      "renter.occupied"
## [7] "median.household.income" "borough"

#Resumo do objeto
summary(dados)

##           X           zipcode      pop.total      pop.white
## Min.      : 1.00      Min.      :10001      Min.      : 513      Min.      : 325
## 1st Qu.: 44.75      1st Qu.:10306      1st Qu.: 25784      1st Qu.: 8089
## Median : 88.50      Median :11106      Median : 40734      Median :15947
## Mean   : 88.50      Mean   :10834      Mean   : 45805      Mean   :20621
## 3rd Qu.:132.25      3rd Qu.:11361      3rd Qu.: 64098      3rd Qu.:28548
## Max.    :176.00      Max.    :11697      Max.    :106154      Max.    :91302
##
## owner.occupied renter.occupied median.household.income      borough
## Min.      : 35      Min.      : 43      Min.      :14271      bronx      :25
## 1st Qu.: 2055      1st Qu.: 4326      1st Qu.: 29489      brooklyn   :36
## Median : 4581      Median :11096      Median : 40998      manhattan  :39
## Mean   : 5367      Mean   :12003      Mean   : 43131      queens     :64
## 3rd Qu.: 7738      3rd Qu.:18897      3rd Qu.: 54689      staten     :12
## Max.    :21147      Max.    :40627      Max.    :112947
## NA's     :1        NA's     :1        NA's     :1

#vizualizar as ultimas seis linhas do objetos
tail(dados)

##           X zipcode pop.total pop.white owner.occupied renter.occupied
## 171 171    11436    18148      325          3871          1664
## 172 172    11691    56184    16841          4257         13938
## 173 173    11692    15893     2378          1473          3533
## 174 174    11693    11157     6606          1934          2246
## 175 175    11694    19278    16832          3566          4281
## 176 176    11697     4226     4194          1753           43
## median.household.income borough
## 171          40194 queens
## 172          27820 queens
## 173          29059 queens
## 174          37248 queens
## 175          48604 queens
## 176          58491 queens
```

Função view() e dim()

A função view permite visualizar os elementos no script do dataframe requisitado, enquanto a função dim (abreviatura de dimensões) fornece o número de linhas e de colunas, respectivamente.

```
View(dados)
dim(dados)

## [1] 176 8
```

Para alterar um nome de uma variável pode ser utilizado o comando colnames. No exemplo acima, vamos alterar o nome da coluna 'borough' para 'distrito'.

```
#Alterar o nome da coluna, sendo que o '[2]' indica que está na segunda coluna.
colnames(dados)[8]='distrito'
#Visualizando as primeiras 6 colunas.
head(dados)

##   X zipcode pop.total pop.white owner.occupied renter.occupied
## 1 1   10001   17310   10966         1970         6971
## 2 2   10002   84870   22140         4146        27366
## 3 3   10003   53673   41661         7830        21686
## 4 4   10004    1225    903          133         489
## 5 5   10005     884    595           35         489
## 6 6   10007    3522   2507          423         807
## median.household.income distrito
## 1                40932 manhattan
## 2                24022 manhattan
## 3                60891 manhattan
## 4               101868 manhattan
## 5                79517 manhattan
## 6               112947 manhattan
```

Para selecionarmos uma coluna do objeto 'dados', por exemplo a coluna 'pop.total', poderíamos digitar no console o comando **dados\$pop.total**. O padrão de carregamento da base de dados nos obriga a dizer ao R qual é a base que quer selecionar (dados), inserindo o símbolo '\$' e após o nome da coluna a qual deseja as informações. Para criar um novo objeto com esta informação, basta dizer ao R, como já visto acima, por exemplo: **novo.objeto=dados\$pop.total**.

No entanto, para acessar os dados sem o uso do símbolo '\$', podemos usar o seguinte comando: **attach(dados)**. Assim, podemos efetuar o sumário da coluna 'pop.total':

```
#Definindo a função attach para o objeto 'dados'.
attach(dados)
#Efetuando o sumário de 'pop.total'.
summary(pop.total)

##   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   513  25780   40730   45810   64100  106200

#Como a coluna 'distrito' é um fator, o sumário será
#a contagem da quantidade de cada fator na coluna.
summary(distrito)

##   bronx  brooklyn manhattan   queens   staten
##    25    36      39      64     12
```

2.2.1 Comando tapply

O comando 'tapply' agrega os dados pelos níveis das variáveis qualitativas. Note que a coluna 'distrito' possui dados em forma de fatores. Assim, para filtrarmos a população (coluna 'pop.total') média por distrito, podemos utilizar:

```
#Função 'tapply', número médio da população total por distrito.
tapply(pop.total, distrito, mean)

##   bronx  brooklyn manhattan   queens   staten
## 53107.60 67122.67 39055.38 36730.77 36977.33
```

No caso da coluna 'median.household.income', ela possui um registro NA (faltante), assim para que se efetue a média por distrito neste quesito, há que se adicionar o parâmetro 'na.rm=T', que ignora as células faltantes para calcular-se a média:

```
#Função 'tapply' considerando NAs:
tapply(median.household.income, distrito, mean)

##      bronx  brooklyn manhattan  queens  staten
## 29851.44 33296.39 51452.92      NA 53403.58

#Função 'tapply' sem considerar NAs:
tapply(median.household.income, distrito, mean, na.rm=T)

##      bronx  brooklyn manhattan  queens  staten
## 29851.44 33296.39 51452.92 46913.32 53403.58
```

2.2.2 Comando 'subset'

Utiliza-se o comando subset() para formar um subconjunto de dados o qual desejamos selecionar de um objeto. Por exemplo, se quisermos criar um novo objeto com somente os dados do 'distrito' Bronx:

```
dadosBronx=subset(dados, distrito=='bronx')
head(dadosBronx)

##      X zipcode pop.total pop.white owner.occupied renter.occupied
## 52 52 10451 40961 7816 1763 13050
## 53 53 10452 72138 11981 778 22096
## 54 54 10453 76775 11982 1232 22814
## 55 55 10454 34976 8584 532 10942
## 56 56 10455 37465 8605 815 11151
## 57 57 10456 76656 10864 1644 23525
##      median.household.income distrito
## 52 20307 bronx
## 53 20606 bronx
## 54 21109 bronx
## 55 14271 bronx
## 56 19389 bronx
## 57 16664 bronx
```

2.3 Estrutura de dados

2.3.1 Vetores

```
# Criação de um vetor
x= c(2, 4, 6)
x

## [1] 2 4 6

# Criação de um vetor a partir de uma sequencia numérica
x= c(2:6)
x
```

```
## [1] 2 3 4 5 6

# Criação de um vetor a partir do intervalo entre cada elemento e valores
#mínimo e máximo
x= seq(2, 3, by=0.5)
x

## [1] 2.0 2.5 3.0

# Criação de um vetor através de uma repetição
x= rep(1:2, times=4)
x

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

y= rep(1:2, each=3)
y

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

Vetores da Classe Fator e as Funções "table" e "tapply"

Os fatores são uma classe especial de vetores, que definem variáveis categóricas de classificação, como os tratamentos em um experimento fatorial, ou categorias em uma tabela de contingência.

A função `factor` cria um fator, a partir de um vetor :

```
sexo<-factor(rep(c("F", "M"),each=8))
sexo

## [1] F F F F F F F F M M M M M M M M
## Levels: F M

numeros=rep(1:3,each=3)
numeros

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

numeros.f<-factor(numeros)
numeros.f

## [1] 1 1 1 2 2 2 3 3 3
## Levels: 1 2 3
```

Fatores têm um atributo que especifica seus níveis ou categorias (levels), que seguem ordem alfanumérica crescente, por *default*. Em muitas análises essa ordem é de fundamental importância e dessa forma pode ser alterada através do argumento `levels`, por exemplo, para que possa ser colocado o controle antes dos tratamentos:

```
tratamentos=factor(rep(c("controle","adubo A","adubo B"), each=4))
tratamentos

## [1] controle controle controle controle adubo A adubo A adubo A
## [8] adubo A adubo B adubo B adubo B adubo B
## Levels: adubo A adubo B controle
```

```
tratamentos=factor(rep(c("controle","adubo A","adubo B"), each=4),
levels=c("controle", "adubo A", "adubo B"))
tratamentos

## [1] controle controle controle controle adubo A adubo A adubo A
## [8] adubo A adubo B adubo B adubo B adubo B
## Levels: controle adubo A adubo B
```

Fatores podem conter níveis não usados (vazios):

```
participantes=factor(rep("mulheres",10), levels=c("mulheres","homens"))
participantes

## [1] mulheres mulheres mulheres mulheres mulheres mulheres mulheres
## [8] mulheres mulheres mulheres
## Levels: mulheres homens
```

Função “tapply”

Para aplicar uma função aos subconjuntos de um vetor definidos por um fator use a função tapply:

```
sexo=factor(rep(c("F","M"),each=9))
dieta=factor(rep(rep(c("normal","light","diet"), each=3),2), levels=c("normal",
"light","diet"))
peso=c(90, 89, 78, 69, 85, 69, 77, 89, 80, 60, 75, 79, 65, 94, 69, 85, 69, 77)
sexo

## [1] F F F F F F F F F M M M M M M M M
## Levels: F M

dieta

## [1] normal normal normal light light light diet diet diet normal
## [11] normal normal light light light diet diet diet
## Levels: normal light diet

peso=as.numeric(peso)

# média de peso frente ao sexo e dieta
tapply(peso,list(sexo,dieta), mean)

##      normal      light diet
## F 85.66667 74.33333 82
## M 71.33333 76.00000 77
```

Função “table”

Para contar elementos em cada nível de um fator, usa-se a função table:

```
table(participantes)

## participantes
## mulheres  homens
##          10         0
```

A função pode fazer tabulações cruzadas, gerando uma tabela de contingência, esse tipo de tabela é usado para registrar observações independentes de duas ou mais variáveis aleatórias:

```
table(sexo,dieta)

##      dieta
## sexo normal light diet
##    F      3      3      3
##    M      3      3      3
```

2.3.2 Matrizes

A função `matrix` tem a finalidade de criar uma matriz com os valores do argumento `data`, argumento este que insere as variáveis desejadas na matriz. O número de linhas é definido pelo argumento `nrow` e o número de colunas é definido pelo argumento `ncol`:

```
nome.da.matriz= matrix(data=1:12,nrow = 3,ncol = 4)
nome.da.matriz

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

Por *default* (ação tomada pelo software), os valores são preenchidos por coluna. Para preencher por linha basta instruir o programa de outra forma, alterando o argumento `byrow` para `TRUE`:

```
nome.da.matriz= matrix(data=1:12,nrow = 3,ncol = 4, byrow=T)
nome.da.matriz

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

Se a matriz inserida tem menos elementos do que a ordem informada para a matriz, os são repetidos até preenchê-la:

```
lista= list(matrix=matrix(c(1,2,1), nrow=3, ncol=2))
lista

## $matrix
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    2
## [3,]    1    1
```

2.3.3 Listas

As listas podem ser criadas a partir do comando “`list`”.

`nrow`: corresponde ao número de linhas;
`ncol`: corresponde ao número de colunas.

Para ver quais elementos estão em suas listas é só chamar pelo nome que foi dado para ela, como no exemplo abaixo.

Representa uma coleção de objetos (PINTO JUNIOR, Jony Arrais).

```
lista= list(matriz=matrix(c(1,2,1,5,7,9), nrow=3, ncol=2),vetor=1:6)
lista

## $matriz
##      [,1] [,2]
## [1,]    1    5
## [2,]    2    7
## [3,]    1    9
##
## $vetor
## [1] 1 2 3 4 5 6
```

COMANDOS PARA MANIPULAÇÕES DE LISTAS (PINTO JUNIOR, Jony Arrais)

Para descobrirmos de maneira rápida o números de objetos que há na lista, utilizamos o comando `length(nome da lista)`.

```
lista

## $matriz
##      [,1] [,2]
## [1,]    1    5
## [2,]    2    7
## [3,]    1    9
##
## $vetor
## [1] 1 2 3 4 5 6

length(lista)

## [1] 2
```

O uso do comando “`names(nome da lista)`” retorna os nomes dos objetos que estão presentes na lista.

```
names(lista)

## [1] "matriz" "vetor"
```

Para chamar várias listas através usamos o comando “`c(nome 1, nome 2)`”, da seguinte forma:

```
lista.1= list(matriz=matrix(c(1,2,1,5,7,9), nrow=3, ncol=2),vetor=1:6)
lista.2= list(nomes=c("Marcelo", "Fábio", "Felipe"), idade=c(25, 34, 26))
c(lista.1,lista.2)

## $matriz
##      [,1] [,2]
## [1,]    1    5
## [2,]    2    7
```

```
## [3,] 1 9
##
## $vetor
## [1] 1 2 3 4 5 6
##
## $nomes
## [1] "Marcelo" "Fábio" "Felipe"
##
## $idade
## [1] 25 34 26
```

2.3.4 Dataframes

Com a função `data.frame` reunimos vetores de mesmo comprimento em um só objeto (Eco R). Neste caso são criadas tabelas de dados. Cada observação é descrita por um conjunto de propriedades. Abaixo podemos ver como inserir os dados para criar a “tabela”.

Similar como matrizes, porem diferentes colunas podem possuir elementos de natureza diferentes (PINTO JUNIOR, Jony Arrais).

```
estudantes= c("Camila", "Pedro", "Marcelo","Guilherme")
idade=c(21,17,17,18)
peso=c(65,79,80,71)
informacoes=data.frame(estudantes,idade,peso)
informacoes

##   estudantes idade peso
## 1   Camila    21   65
## 2   Pedro    17   79
## 3  Marcelo    17   80
## 4  Guilherme    18   71
```

Adicionando colunas no `data.frame` através do comando “nome do `data.frame` \$ variável a ser adicionada”

```
informacoes$ciudades=c("Nova Hartz","Gramado","Soledade","Porto Alegre")
informacoes

##   estudantes idade peso   cidades
## 1   Camila    21   65  Nova Hartz
## 2   Pedro    17   79   Gramado
## 3  Marcelo    17   80   Soledade
## 4  Guilherme    18   71 Porto Alegre
```

3 Manipulação de banco de dados

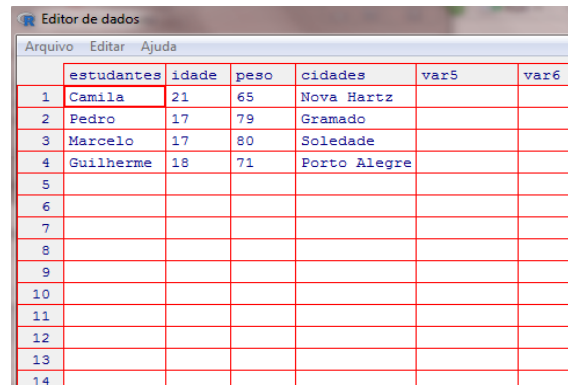
3.1 Função `edit()`

Esta função abre uma interface simples de edição de dados em formato planilha, e é útil para pequenas modificações. Mas para salvar as modificações atribua o resultado da função `edit` a um objeto (Eco R).

Utilizamos o comando da seguinte forma:

```
novo.nome.para.o.banco.de.dados = edit(nome.atual.do.banco.de.dados)
```

```
informacoes.2=edit(informacoes)
```



	estudantes	idade	peso	cidades	var5	var6
1	Camila	21	65	Nova Hartz		
2	Pedro	17	79	Gramado		
3	Marcelo	17	80	Soledade		
4	Guilherme	18	71	Porto Alegre		
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						

Figura 9: Editor de dados.

Basta clicar no retângulo correspondente a variável que deseja ser modificada, excluir ou adicionar novas colunas.



	estudantes	idade	peso	tecnico	var5
1	Camila	21	65	sim	
2	Pedro	17	79	não	
3	Marcelo	17	80		
4	Guilherme	18	71		
5					
6					
7					
8					
9					
10					
11					

Figura 10: Acréscimo de uma nova coluna através do editor de dados

Logo, chamando o novo banco de dados, teremos:

```
informacoes.2
## estudantes idade peso cidades
## 1 Camila 21 65 Nova Hartz
## 2 Pedro 17 79 Gramado
## 3 Marcelo 17 80 Soledade
## 4 Guilherme 18 71 Porto Alegre
```

3.2 Comando head, tail

Por meio da manipulação do comando head() podemos reproduzir os primeiros seis valores do objeto, no caso do mesmo ser um data.frame, podemos solicitar o número de valores ou linhas a serem mostrados no console através do parâmetro n ou na ausência deste, todas as linhas serão impressas (AQUINO, 2014).

3.3 Funções

As funções a seguir são aplicáveis a vetores, data.frames e listas, e em muitos casos trazem praticidade a uma análise estatística:

```
# União de um banco de dados (existencia de uma variavel em comum)

estudantes=c("Guilherme", "Marcelo", "Pedro", "Camila")
altura= c(1.60, 1.9, 1.74, 1.80)
informacoes.3=data.frame(estudantes, altura)
informacoes=merge(informacoes.2,informacoes.3, by="estudantes")
informacoes$Imc=c(peso/(altura^2))
informacoes

##      estudantes idade peso      cidades altura      Imc
## 1      Camila    21   65   Nova Hartz   1.80 25.39062
## 2  Guilherme    18   71 Porto Alegre   1.60 21.88366
## 3      Marcelo    17   80    Soledade   1.90 26.42357
## 4       Pedro    17   79     Gramado   1.74 21.91358

# Retirar as linhas que tenham pelo menos um NA:

informacoes<- na.omit(informacoes)
informacoes

##      estudantes idade peso      cidades altura      Imc
## 1      Camila    21   65   Nova Hartz   1.80 25.39062
## 2  Guilherme    18   71 Porto Alegre   1.60 21.88366
## 3      Marcelo    17   80    Soledade   1.90 26.42357
## 4       Pedro    17   79     Gramado   1.74 21.91358

# Substituir NA's por zero no data.frame

informacoes[is.na(informacoes)] = 0
informacoes

##      estudantes idade peso      cidades altura      Imc
## 1      Camila    21   65   Nova Hartz   1.80 25.39062
## 2  Guilherme    18   71 Porto Alegre   1.60 21.88366
## 3      Marcelo    17   80    Soledade   1.90 26.42357
## 4       Pedro    17   79     Gramado   1.74 21.91358

# Substituir números na coluna
informacoes$idade[informacoes$idade == 17] <- 19

# Classificar qualitativamente informações em um determinado intervalo
classificacao=ifelse(informacoes$Imc<25, "peso normal", "excesso de peso")
informacoes=cbind(informacoes, classificacao)

# Classificar informações usando o código binário em um determinado intervalo
classificacao=ifelse(informacoes$Imc<25, "peso normal", "excesso de peso")
binario= ifelse(informacoes$classificacao == 'peso normal', 1, 0)
cbind(informacoes, binario)

##      estudantes idade peso      cidades altura      Imc      classificacao
## 1      Camila    21   65   Nova Hartz   1.80 25.39062 excesso de peso
```

```
## 2  Guilherme      18   71 Porto Alegre   1.60 21.88366      peso normal
## 3   Marcelo      19   80   Soledade    1.90 26.42357  excesso de peso
## 4    Pedro      19   79    Gramado    1.74 21.91358      peso normal
##   binario
## 1         0
## 2         1
## 3         0
## 4         1

# Usando o Pacote DPLYR
novo1=data.frame(estudantes="Francisco", idade= 30, peso= 59, Imc= 21.3387,
                 classificacao= "peso normal")
informacoes= rbind(informacoes, novo1)

## Error in rbind(deparse.level, ...): numbers of columns of arguments do not match

library(dplyr)

##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##   filter, lag
## The following objects are masked from 'package:base':
##
##   intersect, setdiff, setequal, union

informacoes= mutate(informacoes, "faixa etaria"= ifelse(informacoes$idade<21,
                                                         "adolescente", "adulto"))

# Reordenar colunas
informacoes[c(2,3,4,1, 6, 5)]

##   idade peso      cidades estudantes      Imc altura
## 1    21   65   Nova Hartz      Camila 25.39062   1.80
## 2    18   71 Porto Alegre   Guilherme 21.88366   1.60
## 3    19   80   Soledade     Marcelo 26.42357   1.90
## 4    19   79    Gramado     Pedro 21.91358   1.74

# Inversão do posicionamento dos elementos
rev(informacoes)

##   faixa etaria  classificacao      Imc altura      cidades peso idade
## 1      adulto  excesso de peso 25.39062   1.80   Nova Hartz   65    21
## 2  adolescente  peso normal 21.88366   1.60 Porto Alegre   71    18
## 3  adolescente  excesso de peso 26.42357   1.90   Soledade    80    19
## 4  adolescente  peso normal 21.91358   1.74    Gramado    79    19
##   estudantes
## 1      Camila
## 2   Guilherme
## 3     Marcelo
## 4      Pedro

# contagem de objetos
table(idade)
```

```
## idade
## 17 18 21
## 2 1 1

# Ordenar os objetos em ordem crescente
sort(idade)

## [1] 17 17 18 21

# Ordenar os objetos a partir de uma variável

# Ordem decrescente
informacoes[order(informacoes$idade, decreasing = TRUE),]

##      estudantes idade peso      cidades altura      Imc      classificacao
## 1      Camila     21  65   Nova Hartz   1.80 25.39062 excesso de peso
## 3      Marcelo     19  80   Soledade    1.90 26.42357 excesso de peso
## 4      Pedro      19  79   Gramado     1.74 21.91358 peso normal
## 2  Guilherme     18  71 Porto Alegre  1.60 21.88366 peso normal
##      faixa etaria
## 1      adulto
## 3  adolescente
## 4  adolescente
## 2  adolescente

# ordem crescente
informacoes[order(informacoes$idade, decreasing = FALSE),]

##      estudantes idade peso      cidades altura      Imc      classificacao
## 2  Guilherme     18  71 Porto Alegre  1.60 21.88366 peso normal
## 3      Marcelo     19  80   Soledade    1.90 26.42357 excesso de peso
## 4      Pedro      19  79   Gramado     1.74 21.91358 peso normal
## 1      Camila     21  65   Nova Hartz   1.80 25.39062 excesso de peso
##      faixa etaria
## 2  adolescente
## 3  adolescente
## 4  adolescente
## 1      adulto
```

3.4 Funções Matemáticas

```
log(1)

## [1] 0

exp(1)

## [1] 2.718282

# Informar o valor máximo contido no conjunto
max(idade)

## [1] 21

max(peso)
```

```
## [1] 80

# Informar o valor mínimo contido no conjunto
min(idade)

## [1] 17

min(peso)

## [1] 65

# Para descobrir em qual posição se encontra o peso mínimo:
which.min(peso)

## [1] 1

# Agora, para saber qual é o nome a que se refere este peso:
informacoes$estudantes[which.min(peso)]

## [1] Camila
## Levels: Camila Guilherme Marcelo Pedro

# Arredondar para n casas decimais
round(pi, 2)

## [1] 3.14

# Determinar o número de algarismos significativos
signif(pi, 2)

## [1] 3.1

# Realiza a somatória dos valores
sum(idade)

## [1] 73

sum(peso)

## [1] 295

# Desvio padrão
sd(idade)

## [1] 1.892969

# Variância
var(idade)

## [1] 3.583333

# Calcula a média aritmética dos valores
mean(idade)

## [1] 18.25

# Informa o valor mediano do conjunto
median(idade)

## [1] 17.5

quantile(y, probs = c(0.5, 1, 2, 5, 10, 50)/100)

## 0.5%  1%   2%   5%  10%  50%
##  1.0  1.0  1.0  1.0  1.0  1.5

x= c(1, 1.5, 1.4, 1, 2, 1.8)
```

3.5 Subconjuntos de Matrizes

```
x= matrix(data=1:12,nrow = 3, ncol = 4)

x[, 1]
## [1] 1 2 3

x[1, 2]
## [1] 4

nrow(x)
## [1] 3

ncol(x)
## [1] 4

dim(x)
## [1] 3 4
```

3.6 Conversão de datas

```
abertura <- c("03/02/69", "17/08/67")
fechamento <- c("2000-20-01", "1999-14-08")
abertura <- as.Date(abertura, format = "%d/%m/%y")
fechamento <- as.Date(fechamento, format = "%Y-%d-%m")

# Diferença de dias dos intervalos informados
abertura-fechamento

## Time differences in days
## [1] -11308 24840
```


4 REFERÊNCIAS BIBLIOGRÁFICAS

PINTO JUNIOR, Jony Arrais. **Projeto: Métodos computacionais para estatística II.**

Curso Software R. Disponível em:

http://www.professores.uff.br/jony/lib/exe/fetch.php?media=disciplinas:curso_r.pdf

ECOR. **Leitura e Manipulação de Dados.** Disponível em:

http://ecologia.ib.usp.br/bie5782/doku.php?id=bie5782:03_apostila:04-dados Acesso em: 28 de nov.2016