UNIVERSIDADE FEDERAL DE SÃO CARLOS - UFSCar
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS - CCET
DEPARTAMENTO DE ESTATÍSTICA - DEs

PROGRAMA DE EDUCAÇÃO TUTORIAL - PET

Minicurso RStudio

São Carlos - SP
2019

Conteúdo

1 Introdução

O R é uma linguagem e ambiente de computação estatística. Considerado uma variante da linguagem S (laboratórios Bell, desenvolvida por John Chambers e seus colegas), surgiu pela criação da R Foundation for Statistical Computing, com o objetivo de criar uma ferramenta gratuita e de utilização livre para análise de dados e construção de gráficos.

2 Instalando e conhecendo o RStudio

Estamos interessados em familiarizar com o RStudio afim de obter maior conhecimento sobre o software, para que tenhamos uma melhor análise sobre os nossos dados.

2.1 Instalando o R

Para instalar o R no Windows, é necessário entrar no link a seguir para fazer download do instalador com a versão mais atualizada: *https://cran.rproject.org/bin/windows/base/* após baixar, salve o arquivo em qualquer pasta do seu computador.

Com o arquivo já salvo no computador, agora vamos instalar o nosso software, clique duas vezes para instalar o arquivo, em seguida em avançar até aparecer a seguinte tela:

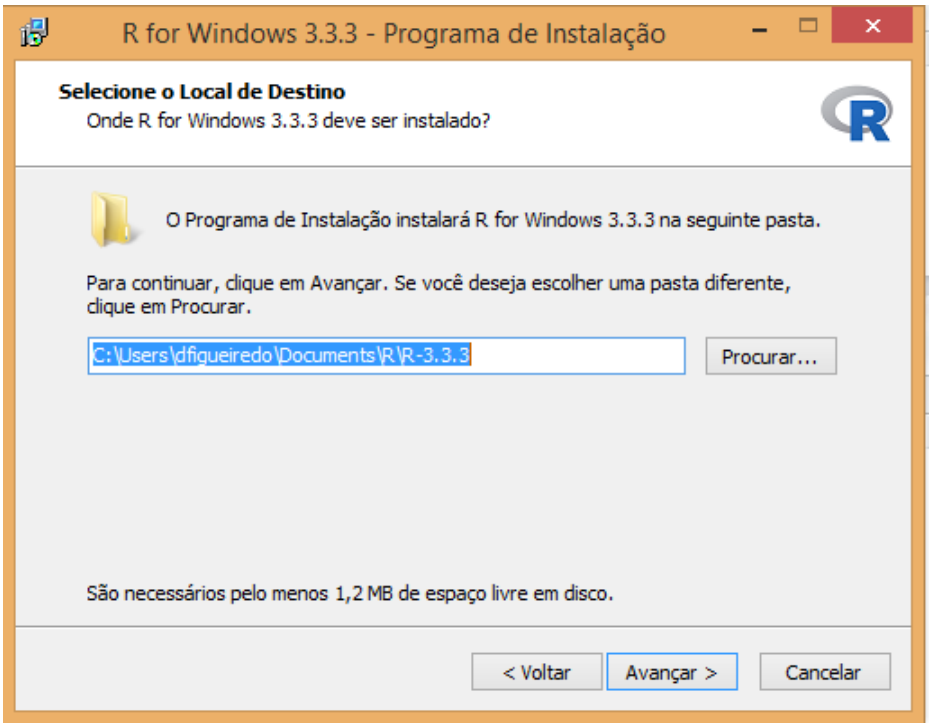

Figura 1: Programa de instalação do R

Continue clicando em “Avançar” até chegar no botão “Concluir”.

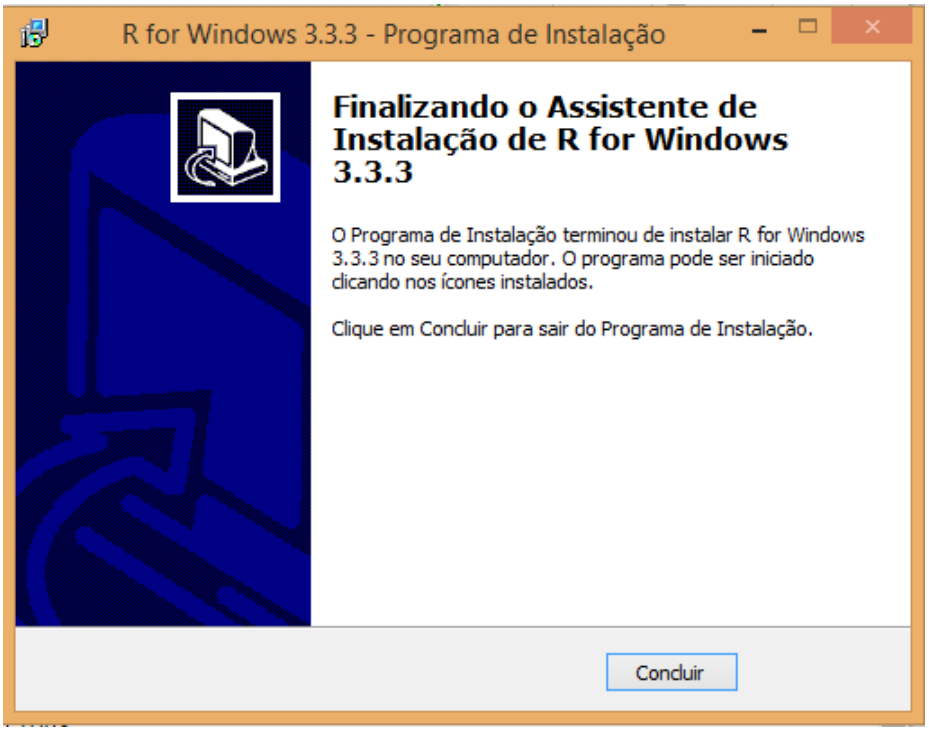

Figura 2: Programa de instalação do R

Assim, já temos o R instalado em nossa máquina.

2.2 Instalando do RStudio

Agora vamos instalar o RStudio, que é onde iremos trabalhar com toda a parte de programação. Para fazer o download é necessário acessar o site do RStudio pelo link: ***https://www.rstudio.com/products/rstudio/download/***.

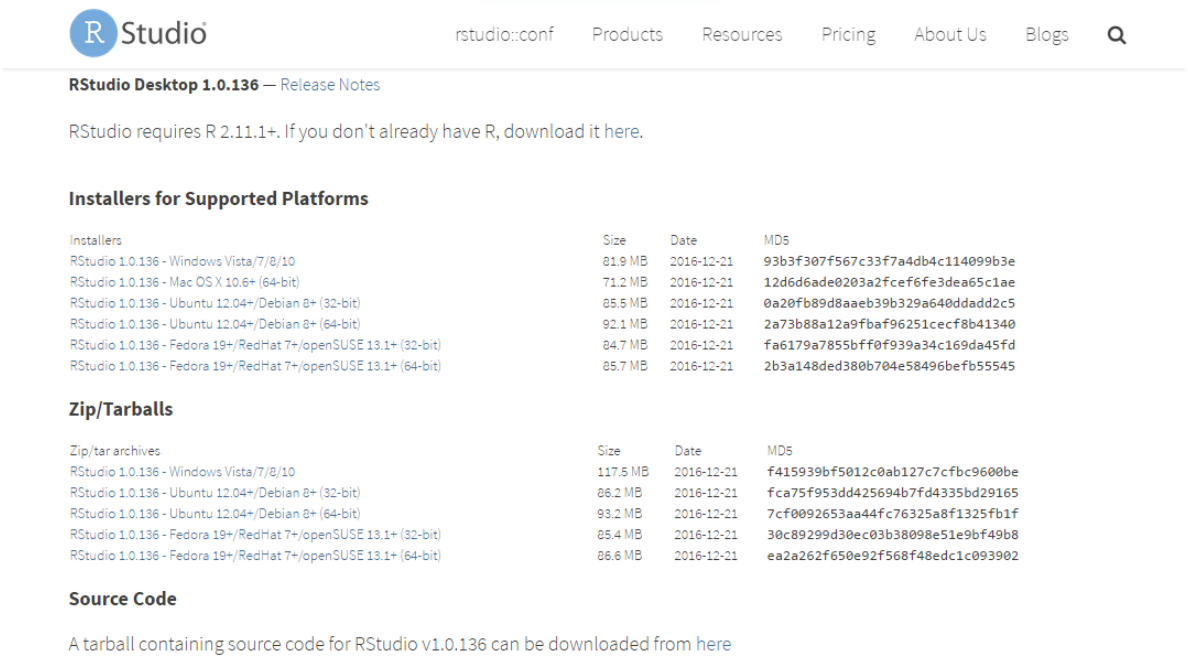

Figura 3: Site para download do RStudio

Nessa imagem temos dois possíveis casos:

1. Administrador: só escolher a versão compatível com o sistema operacional na guia *Installers for Supported Platforms*. Em seguida a instalação será bem simples, apenas clicando em "Avançar".

2. Caso não seja administrador, será necessário baixar o arquivo compactado na guia *Zip/Tarballs*. Após extrair a pasta baixada, terá que ir na subpasta *bin*, e executar o arquivo rstudio.

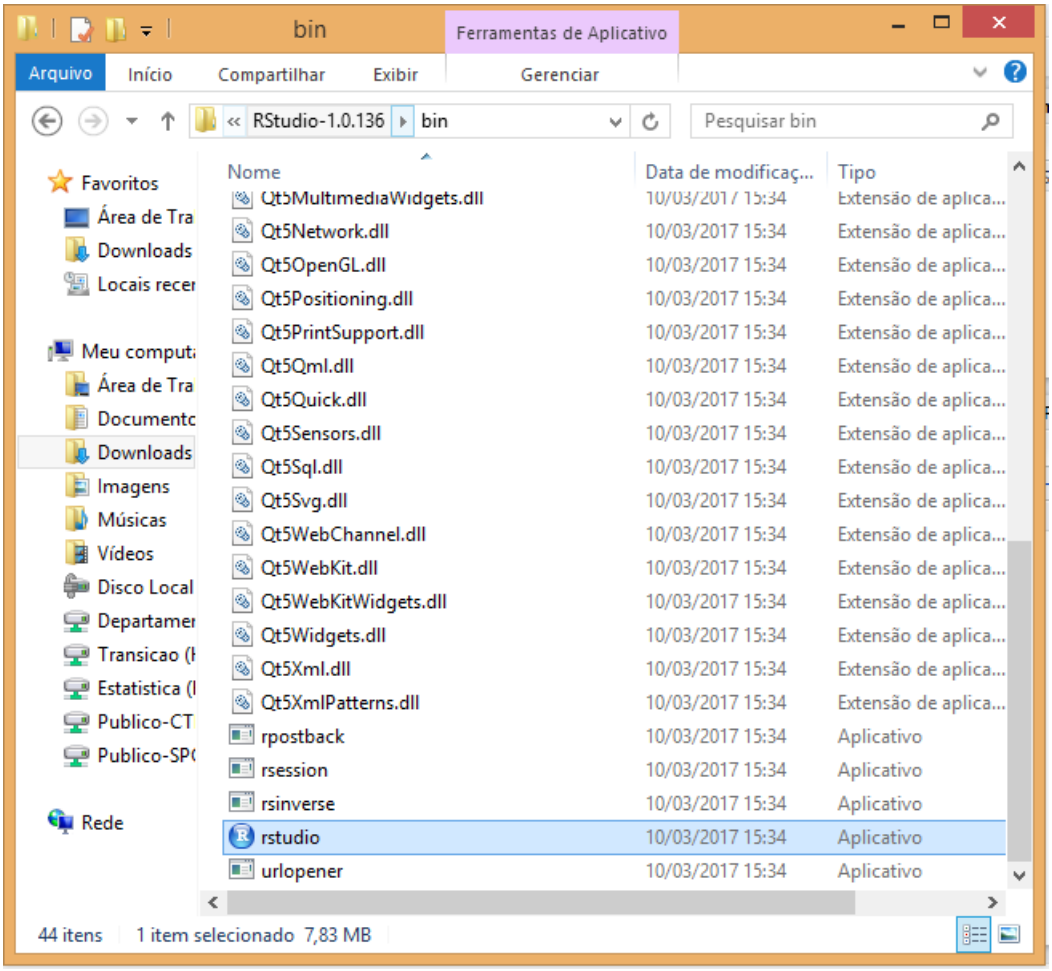

Figura 4: Arquivo de execução do RStudio

2.3 Conhecendo o RStudio

Ao abrir o RStudio temos a interface com 4 quadrantes, com as seguintes funções:

Editor/Script: É onde escrevemos os códigos.

Console: É onde rodamos os códigos e recebemos as saídas.

Environment: Painel com todos os objetos criados no R, ao lado tem a aba *History* que contém o histórico com todos os comandos utilizados anteriormente.

Output: Responsável por toda comunicação do R, onde temos a aba *Files* que mostra os diretórios dos trabalhos. Já a aba *Plots* é toda a saída gráfica, ou seja, qualquer gráfico que for criado na compilação é exibido nesta aba, já o *Help* exibe todas as documentações das funções.

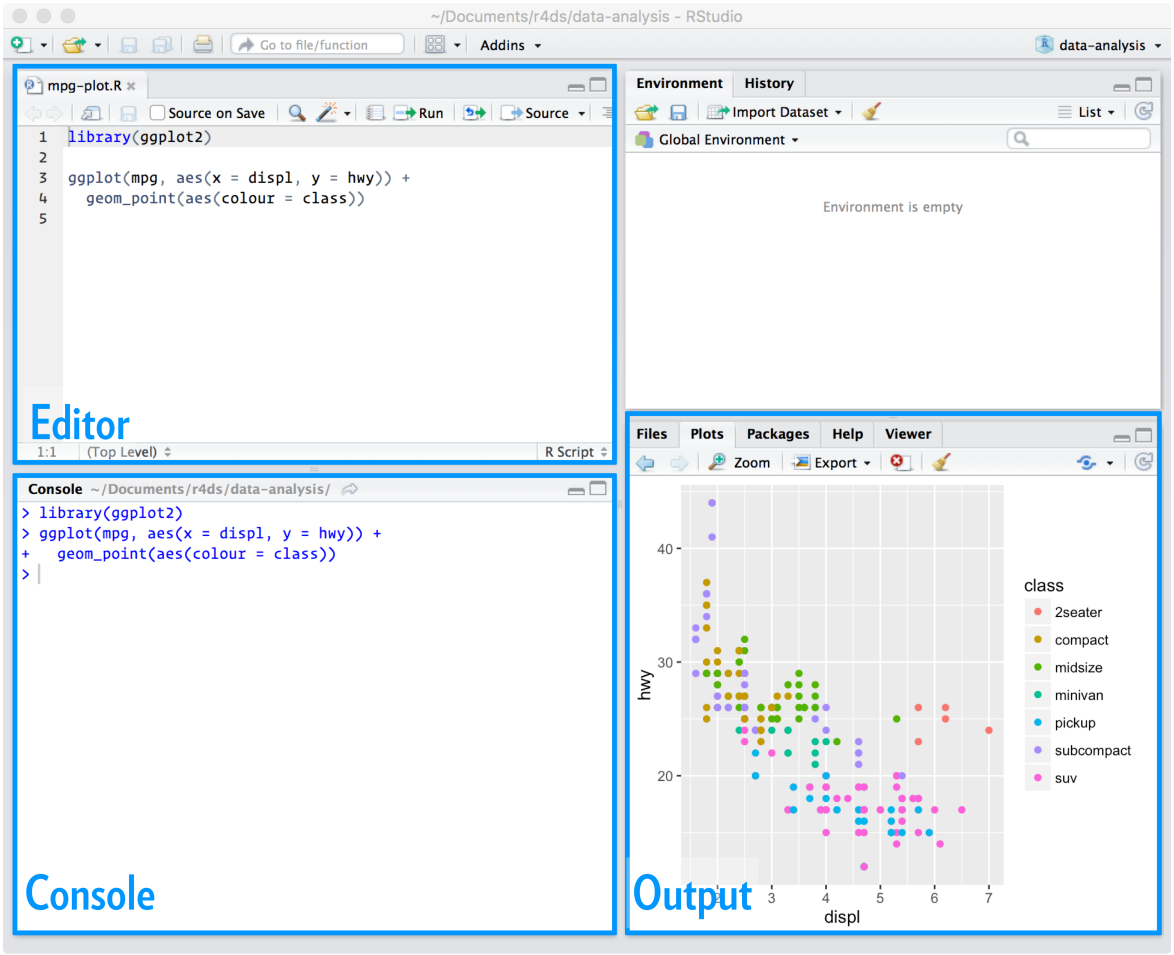

Figura 5: Ambiente de trabalho do RStudio

2.4 Instalação de pacotes

Para realizarmos algumas tarefas no R, são necessários alguns pacotes específicos. Vamos, por exemplo, instalar o pacote *ggplot2*, que é utilizado para a elaboração de gráficos mais refinados, para instalá-lo basta digitar: *install.packages("nome do pacote")* no console, depois clicar em *Enter*. Após a instalação é necessário ainda mais um comando: library(ggplot2), com isso todas as funções ficam disponíveis para serem utilizadas.

Uma outra maneira de instalar pacotes no R é manualmente. Para isso, é só clicar em *Tools* na parte superior do programa e em seguida em *Install Packages*. Com isso, irá aparecer uma janela com um espaço disponível para digitar o nome do pacote que se deseja baixar. Após instalado é necessário o carregamento do pacote. Isso é feito por meio da função *library()*, como já visto acima.

3 Criando conjunto de dados

Existe basicamente duas maneiras diferentes de criar um conjunto de dados no R. Os dados podem ser digitados em uma planilha dentro do próprio R ou ainda podemos importar um arquivo contendo o banco de dados que já esteja salvo no computador.

3.1 Digitando os Dados

Quando é apenas uma única variável que iremos criar, o comando *scan()* é suficiente para criar o conjunto. Exemplo:

```
dados = scan()
    1: 12
    2: 13
    3: 14
    4: 2
    5: 10
    6: 6
    7: 9
    8:
Read 7 items
```

Caso o objetivo é criar um conjunto com várias variáveis faz-se necessário a utilização do comando *edit(data.frame())*. Esse comando faz com que abra uma planilha para que os dados sejam digitados, inclusive nomeando as variáveis. Exemplo:

```
dados = edit(data.frame())
```

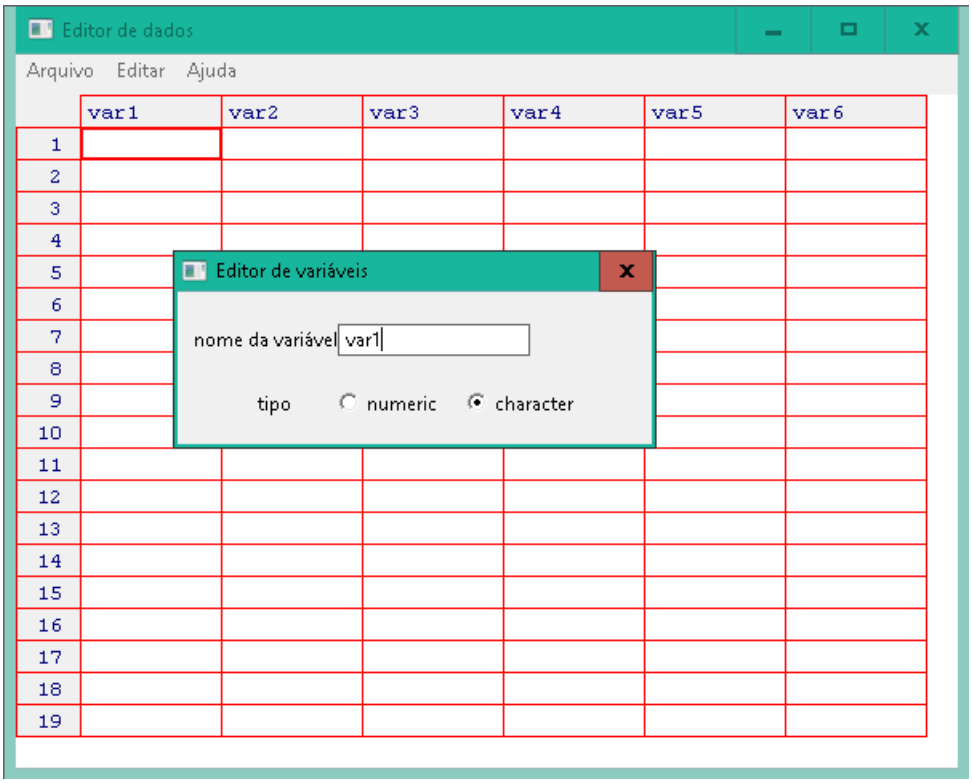

Figura 6: Editor de texto

Nesta planilha, o nome da variável pode ser alterado clicando duas vezes no nome var1, e assim sucessivamente. Depois de digitar os dados basta fechar a planilha para ter todos os valores salvos dentro do R. Caso seja necessário editar os dados, ou acrescentar variáveis basta utilizar o comando *fix(dados)* que abrirá a tela vista anteriormente para edição dos valores.

3.2 Importando os dados

Nos casos em que temos um conjunto de dados salvo no computador, e deseja utilizar para manipulação dentro do R é possível importar este arquivo. O comando utilizado para fazer essa importação é o *read.table()*. O R permite importar arquivos em diferentes formatos entre eles, xlss, csv, txt, entre outros.

Considere um arquivo de dados salvo como **exemplo** em formato txt e que está na pasta Minicurso R da área de trabalho do usuário, o código é dado por:

```
exemplo = read.table("C:Documents and Settings\\Administrador\\
                     Meus documentos\\exemplo.txt",header=T)
```

Entendendo um pouco mais como importar um arquivo, dentre os parâmetros que utilizamos na função *read.table()*, o primeiro é o diretório do arquivo entre aspas. Outro parâmetro é o *header=TRUE* se o conjunto de dados contém o nome da variável na primeira linha do banco, caso não tenha basta colocar igual a *FALSE* ou simplesmente não declarar nada.

Outros parâmetros que podem ser declarados são: conjunto de dados que contém dados com casa decimais, dados separados por tabulação, ponto e vírgula, espaço, entre outros.

Considere agora um arquivo com casas decimais separados por vírgula, e que a separação dos valores esteja por tabulação. O código ficará da seguinte maneira:

```
exemplo = read.table("C:Documents and Settings\\Administrador\\
                     Meus documentos\\exemplo.txt",header=T,
                     dec=",", sep="\t")
```

Caso o diretório do arquivo seja muito grande, não é interessante escrevê-lo, neste caso podemos utilizar o comando *file.choose()*. Usando este comando aparecerá uma janela, na qual você irá escolher o caminho onde se encontra o arquivo. Exemplo:

```
dados = read.table(file.choose(),header=T, dec=",",
                   sep="\t")
```

Alguns outros casos são:

```
dados2 = read.csv(file.choose(), header=T, dec=",")

dados3 = read.csv2(file.choose(), header=T, dec=".", sep="\t")
```

4 R como Calculadora

Pelo console, é possível fazer inúmeros comandos no R. Temos as operações básicas de matemática como: soma, subtração, divisão, potência, entre outras funções. A seguir temos alguns exemplos:

Operação	Código
Soma	+
Subtração	-
Multiplicação	*
Divisão	/
Parte inteira da divisão	%/%
Resto da divisão	%%
Potência	^
Raiz quadrada	sqrt(n)

Tabela 1: Operações matemáticas

Além do mais, as operações e suas precedências são mantidas como na matemática, ou seja, divisão e multiplicação são calculadas primeiro e depois a subtração e adição.

Observação: Parênteses nunca é demais.

4.1 Objeto

Um objeto na linguagem de programação, consiste em qualquer elemento que pode receber um determinado valor, ou característica, como funções, entre outros. O R permite salvar dados dentro de um objeto, para isso é utilizado o operador "=" ou "<-" na qual simboliza atribuição.

```
valor = 2
valor <- 2
```

Deste modo o objeto **valor** passou a possuir o valor 2, ou seja, toda vez que ele executar um código e deparar com o símbolo **valor**, automaticamente substituíra por 2.

4.1.1 Vetores

Vetores no R são objetos simples que armazenam mais de um objeto/valor. Para criarmos um vetor basta utilizar a função *c()*. No R os valores são separados por vírgula, enquanto que as casa decimais são separadas por ".". Podemos observar no seguinte exemplo:

```
vetor1 = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
vetor2 = c(1.1, 2.1, 3, 4, 5, 6, 7.5, 9, 10.1)
```

Com vetores, também é possível realizar as operações matemáticas já citadas anteriormente. Mas para realizar operações é necessário que os vetores tenham o mesmo tamanho. Para verificarmos essa suposição, temos a função *length()*. Podemos ver o seguinte exemplo:

```
x = c(1,2,3,4,5,6)
y = c(3,4,5,6,7,8)

length(x)
length(y)

x - y
x + y
x ^ y
```

4.1.2 Matrizes

Matizes são vetores de duas dimensões. Para criarmos matrizes o comando utilizado é *matrix()*. No comando dessa função os argumentos mais importantes são: *nrow* e *ncol*, que especificam o número de linhas e colunas, respectivamente. Outro argumento não menos importante é o *byrow*, quando utilizarmos o *byrow=TRUE* o R preencherá a matriz por linha.

```
m = matrix(c(1,2,3,4,5,6,7,8,9), nrow=3, ncol=3, byrow = T)
n = matrix(c(1,2,3,4,5,6,7,8,9), nrow=3, ncol=3, byrow = F)

o = matrix(c(1,2,3,4,5,6,7,8,9), 3, 3, byrow = T)
```

Vale ressaltar que um banco de dados pode ser representado como uma matriz, e se houver a necessidade de nomear as colunas e linhas temos duas maneiras com os seguintes comandos:

```
#Primeira maneira
dimnames(m) = list(vertical=c("obs1","obs2","obs3"),
                   horizontal=c("var1", "var2","var3"))

#Segunda maneira
rownames(m) = c("obs1","obs2","obs3")
colnames(m) = c("col1","col2","col3")
```

Vale notar que uma matriz não é, de fato, um banco de dados. Alguns comandos só são aplicados convertendo essas matrizes em matriz para estrutura de dados. Para tal é utilizado a seguinte função: *matriz1 = as.data.frame(m)*. Assim **matriz1** recebe os elementos da matriz **m** no formato de dados.

Operações com matrizes

Quando trabalhamos com matrizes, podemos nos deparar com a necessidade de realizar alguma operação. A seguir, apresentaremos algumas já utilizadas além daquelas já vistas anteriormente.

Para trabalhar com matriz temos alguns comandos como:

- *dim(m):* retorna o número de linhas e colunas da matriz **m**.

- *nrow(m):* retorna o número de linhas da matriz **m**.
- *ncol(m):* retorna o número de colunas da matriz **m**.
- *m[3,]:* Seleciona a terceira linha.
- *m[,4]:* Seleciona a quarta coluna.
- *m[1,2]:* Seleciona o primeiro elemento da segunda coluna.
- *t(m):* Calcula a matriz transposta da matriz **m**.
- *m %*% n:* Calcula o produto matricial entre as matrizes **m** e **n**.
- *solve(m):* Calcula a inversa da matriz **m**.

4.1.3 Valores especiais

Existem ainda alguns valores especiais que representam valores faltantes, infinitos, e indefinições matemáticas.

NA (Not Available) representa dados faltantes no conjunto de observação. Pode ser tanto caractere ou numérico.

NaN (Not a Number) representa indefinições matemáticas, como divisão por zero, logaritmo de número negativo, podemos notar que um NaN é um NA, mas a volta não é verdadeira.

Inf (Infinito) representa um número muito grande ou um limite matemático. O oposto também existe. Ex: (-Inf).

NULL utilizamos para representar a ausência de observação, temos que a diferença entre um NULL e NA ou NaN é bem leve, enquanto os NA estão relacionados a análise estatística, o que seria a ausência de uma observação no conjunto de dados, o NULL está associado a lógica de programação.

Para testar se o elemento possui algum valor especial, utilizamos as seguintes funções: is.na(), is.nan(), is.infinite() e is.null().

```
x = c(1,2,3,NA,NULL,NaN,4,Inf)
b=NULL

is.na(x)
is.nan(x)
is.infinite(x)
is.null(b)
```

4.1.4 Lista

A lista é um vetor especial que pode armazenar valores de classes diferentes. É um dos objetos mais importantes para armazenar dados, então é interessante saber manusear bem.

Para criar uma lista utilizamos o comando *list()* o qual aceita qualquer tipo de objeto dentro da lista. É importante destacar que é possível incluir uma lista dentro de uma lista, a seguir temos um exemplo de lista onde seus objetos são numéricos, caracteres, lógicos e vetor.

```
x = list(1:5, "Z", TRUE, c("a", "b"))
x[[5]] = list(c(1:10), c("PET", "EJE", "SEst"))
```

4.1.5 Data Frame

O objeto *data frame* é o mesmo que uma tabela do *SQL* ou uma tabela do *Excel*, sendo assim um dos objetos mais utilizados no curso para análise, onde por sua maioria será importado para o R. Para utilizar a função, basta utilizar o comando *data.frame()*.

Podemos considerar um *data frame* como uma lista, mas em que todas as variáveis possuem a mesma quantidade de itens, e já que são listas cada coluna possui uma classe específica, sendo essa a principal diferença entre *data frame* e matrizes. A seguir temos algumas funções úteis do *data frame*:

head() – Mostra as seis primeiras observações (default).
tail() – Mostra as seis últimas observações (default).
dim() – Retorna as dimensões do data frame (número de linhas e colunas).
names() – Exibe o nome das colunas do data frame (nome das variáveis).
cbind() – Acopla dois datas frame por coluna (lado a lado).
rbind() – Acopla dois data frames por linha (embaixo do outro).

4.1.6 Criando funções

Em algumas tarefas, faz-se necessário criarmos funções mais específicas, e temos a vantagem de personalizá-las da melhor maneira para a realização da tarefa. Vejamos um exemplo de uma função que o usuário utiliza como *input* um número e a saída dela é a soma desse número com 10:

```
SomaUm = function(x) {
    y = x+1
    return(y)
}
```

Considere o caso em que deseja calcular a média de um vetor, temos o seguinte caso:

```
media = function(x) {
  soma = sum(x)
  n_obs = length(x)
  media = soma/n_obs

  return(media)
}
```

Desta forma não a necessidade de escrever os mesmos comandos repetidamente. E ainda, podemos usar essa função em diferentes códigos utilizando a função *source()*. Para utilizar a função *SomaUm* em um outro código qualquer, basta chamar o código que contém essa função digitando o endereço e nome da função entre aspas.

No exemplo, vamos salvar a função *SomaUm* com o nome *funcoes.r* em uma pasta no desktop. Feito isso, basta abrir um novo código e digitar o seguinte comando:

```
source("C:\\Users\\PET_01\\Documents\\Membros do PET
        \\Clézio Lopes\\funcoes.r")
```

Pronto, basta utilizar a função *SomaUm* dentro deste novo arquivo. Esta é uma forma muito útil para ganhar tempo, basta escrever as funções uma única vez e acessá-las quando necessário.

4.2 Controle de fluxo

Como qualquer outra linguagem de programação bem estruturada, o R também possui controles de fluxo como *if*, *else*, *for*, *while* entre outros.

4.2.1 if e else

O *if()* serve para executarmos alguma tarefa se atender tal característica. Antes de vermos a aplicação do controle de fluxo precisamos atentar para algumas expressões:

== Igualdade.
!= Diferença.

Temos um código que só será executado se o objeto x for igual a 1.

```
x = 2
if(x == 1) {
  Sys.time()       # Devolve a data/hora no momento da execução.
}
```

Caso o objeto x seja diferente de 1, ele não retornaria nenhum valor.

Como o R só executará o que está dentro, se o que está dentro do () for verdadeiro (TRUE), então podemos ter expressões como if else, else if para atender a necessidade desejada.

```
if(x < 0) {
   sinal = "negativo"
} else if(x == 0) {
   sinal = "neutro"
} else if(x > 0) {
   sinal = "positivo"
}
sinal
```

Caso a estrutura do código seja apenas um else, podemos simplificar usando o comando *ifelse()*.

```
ifelse(x<0, "negativo", "positivo")
```

4.2.2 for

Vamos utilizar a função *for()* para somar os elementos de um vetor, por exemplo:

```
x = 1:10   # Cria um vetor com a sequência 1, 2, ..., 10.
soma = 0

for(i in 1:10) {
  soma = soma + x[i]
}
soma
```

De maneira equivalente podemos utilizar a função *sum()*.

Agora, se estivermos interessados em imprimir o resultado da divisão de um determinado vetor por 2. Utilizaremos a função *print()*.

```
vetor = 30:35
indices = seq_along(vetor) # cria o vetor de índices segundo o tamanho
                           # do objeto vetor.
for(i in indices) {
  print(vetor[1:i] / 2)
}
```

4.2.3 while

A função *while()* segue o mesmo raciocínio que a *for()*, vamos ao exemplo onde será mostrado o valor enquanto ele é menor que 6:

```
i = 1
while (i < 6) {
    print(i)
    i = i + 1
}
```

Como resumo, temos os principais operadores lógicos na tabela a seguir:

Operador	Descrição
x < y	x menor que y
x <= y	x menor ou igual a y
x > y	x maior que y
x >= y	x maior ou igual a y
x != y	x diferente de y
x == y	x igual a y
x | y	x ou y são verdadeiros
x & y	x e y são verdadeiros

Tabela 2: Operadores lógicos

5 Probabilidade e Estatística

A seguir, podemos destacar alguns pontos que são mais utilizados no curso e tem uma grande aplicação.

5.1 Probabilidade

5.1.1 Função Densidade (ou Probabilidade)

Quando falamos de função densidade estamos interessados em calcular o valor da densidade, no caso das distribuições contínuas, ou a probabilidade P(X = x) no caso das distribuições discretas, para cada valor do vetor no R. No R o nome dessa função é iniciado pela letra d mais o nome da distribuição. Exemplo: (dbim, dpois, etc.).

5.1.2 Função distribuição

Esta função como já vimos em probabilidade no decorrer do curso, calcula a P(X $\leq$ x). O nome da função é iniciado pela letra p mais o nome da distribuição. Exemplo: (pnorm, pexp, etc.).

5.1.3 Função Probabilidade

Calcula o valor de x correspondente a probabilidade p acumulada. É o intervalo da função distribuição. Esta função tem seu nome iniciado pela letra "q"e mais o nome da distribuição igual nos casos anteriores. Exemplo: (qbeta, qcauchy, etc.).

5.1.4 Gerador Aleatório

Gera números aleatórios baseados na distribuição definida. O nome é iniciado pela letra r mais o nome da distribuição. Exemplo: (rnorm, rbinom, etc.).

5.1.5 Sorteio Aleatório

Para realizarmos um sorteio é bem simples com a ajuda do R, utilizaremos a função *sample()*. Exemplo: sample(1:30, 10, T)

Onde, o primeiro argumento corresponde a amostra a ser sorteada, o segundo elemento indica a quantidades de objetos que irá formar a nossa amostra, e o terceiro se o sorteio será com reposição.

5.2 Estatística

5.2.1 Tabela de Contingência

As tabelas de contingências são usadas para registrar observações independentes de duas ou mais variáveis aleatórias, normalmente qualitativas. Exemplo:

```
fumante<-c("Sim","Não","Sim","Sim","Sim")
sexo <- c("M","F","M","M","F")

table(fumante,sexo)

       sexo
fumante F M
    Não 1 0
    Sim 1 3
```

5.2.2 Média Aritmética

A média aritmética de um conjunto de valores é calculado pela função *mean()*. Exemplo:

```
x=c(22,32,46,57,10,12,2)
mean(x)

25.85714
```

5.2.3 Mediana

A mediana ou a observação central de um conjunto de dados é obtida através do comando *median()*. Exemplo:

```
x=c(22,32,46,57,10,12,2)
median(x)

x=sort(x)
median(x)
```

Obs.: O comando *sort()* ordena o vetor em ordem crescente. Não é necessário ordenar o vetor x para utilizar o comando *median()*.

5.2.4 Variância e Desvio Padrão

A variância é uma medida de dispersão dos dados em torno da média e é obtida pelo comando *var()*. Exemplo:

```
x=1:10
var(x)

y=rep(1,10)
var(y)
```

O desvio padrão é a raiz quadrada da variância e pode ser calculada tanto manualmente quanto pelo comando *sd()*. Exemplo:

```
x=1:10
var(x)

sqrt(var(x))

sd(x)
```

5.2.5 Resumo de dados

Para obter um resumo geral das medidas descritivas de um conjunto de dados, podemos utilizar o comando *summary()*. Exemplo:

```
x=c(4,10,3,0.1,19,55,6,4,21,12,23,39)
summary(x)
```

Para obtermos os valores dos quantis e os valores de máximo e mínimo, podemos utilizar os comandos *quantile()*, *max()*, e *min()*. Exemplo:

```
x=c(4,10,3,0.1,19,55,6,4,21,12,23,39)
quantile(x)

max(x)
min(x)
```

6 Gráficos

Nesta sessão será abordado como utilizar ferramentas para a construção de gráficos no R.

Os gráficos do R normalmente são mais interessantes do ponto de vista gráfico do que de outros pacotes estatísticos, devido sua grande versatilidade em recursos. O R possui uma enorme capacidade para gerar diversos tipos de gráficos de alta qualidade totalmente configuráveis, desde cores e tipos de linhas, até legendas e textos adicionais. Uma desvantagem é que para construir gráficos complexos demanda tempo e requer muitos detalhes de programação, porém, adquire-se isso com a prática.

Neste tópico será abordado os gráficos de barra, pizza, histograma, *boxplot*, e gráfico de pontos (ou gráfico de dispersão).

Argumentos gráficos

Para construção de gráficos no R ou o uso de qualquer função, é necessário conhecer os argumentos das mesmas. Estes permitem a construção de gráficos de forma mais completa, usando títulos, cores, etc.

Nome	Descrição
xlab	Nome do eixo x
ylab	Nome do eixo y
main	Título do gráfico
xlim	(início, fim) vetor contendo os limites do eixo x
ylim	(início, fim) vetor contendo os limites do eixo y
col	Cor de preenchimento do gráfico, pode ser um vetor

Tabela 3: Argumentos gráficos

6.1 Gráfico de barras

São gráficos em que visualizamos a frequência através de retângulos, sendo uma das suas dimensões proporcional à frequência.

Para fazer gráficos de barras no R a função é *barplot()*. Exemplo:

```
x = c(56,44)
names(x) = c("Casado","Solteiro")
barplot(x, col=c("blue","red"), ylim=c(0,60), space=1, width=c(.4,.4),
        main="Número de filhos por estado civil",
        xlab="Estado Civil", ylab="Número de Filhos")
text(locator(n=2),c("56%","44%"))

barplot(c(4500,3000,1200), col=c("blue","red","brown"),
        names.arg=c("Estatístico", "Engenheiro", "Matemático"),
        xlab= "Profissão", ylab="Salário", ylim=c(0,5000))
title(main="Salário da Empresa Xport")
```

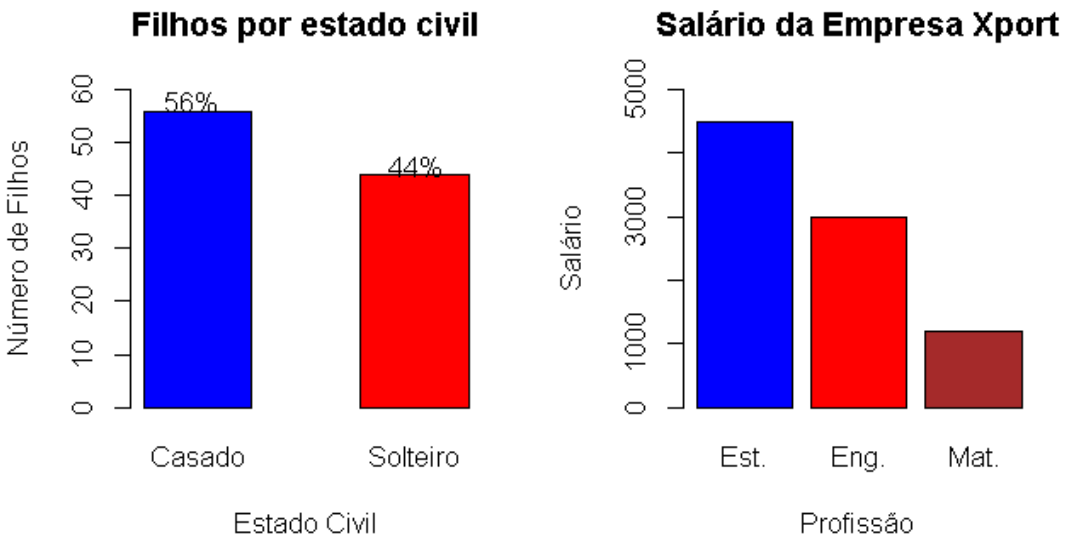

Figura 7: Gráfico de barras

Outra opção para cores é usar a função rainbow, *rainbow()*.

```
par(mfrow=c(1,1))
barplot(c(4500,3000,1200), col=rainbow(25),
        names.arg=c("Est.", "Eng.", "Mat."),
        xlab= "Profissão", ylab="Salário", ylim=c(0,5000))
title(main="Salário da Empresa Xport")
```

6.2 Gráfico de Pizza

Os gráficos de setores ou de pizza, como é mais comumente chamado, também mostram a frequência ou proporção através de fatias de uma circunferência.

Para fazer gráficos de pizza utiliza-se a função *pie()*. Exemplo:

```
venda = c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
names(venda) = c("Morango","Holandesa","Mousse Maracujá",
                 "Paulista","outras","Chocolate")
pie(venda, col= c("red", "purple", "green3", "cornsilk", "cyan", "brown4"))
title(main="Tortas mais vendidas")

educa = c(0.55,0.26,0.19)
pie(educa,labels=c("Ensino Médio","Ensino Fundamental","Ensino Superior"),
    col=c("DeepSkyBlue","Maroon1","Plum1"), radius=1.1,
    main="Grau de instrução")
text(locator(n=3),c("55%","26%","19%"))
```

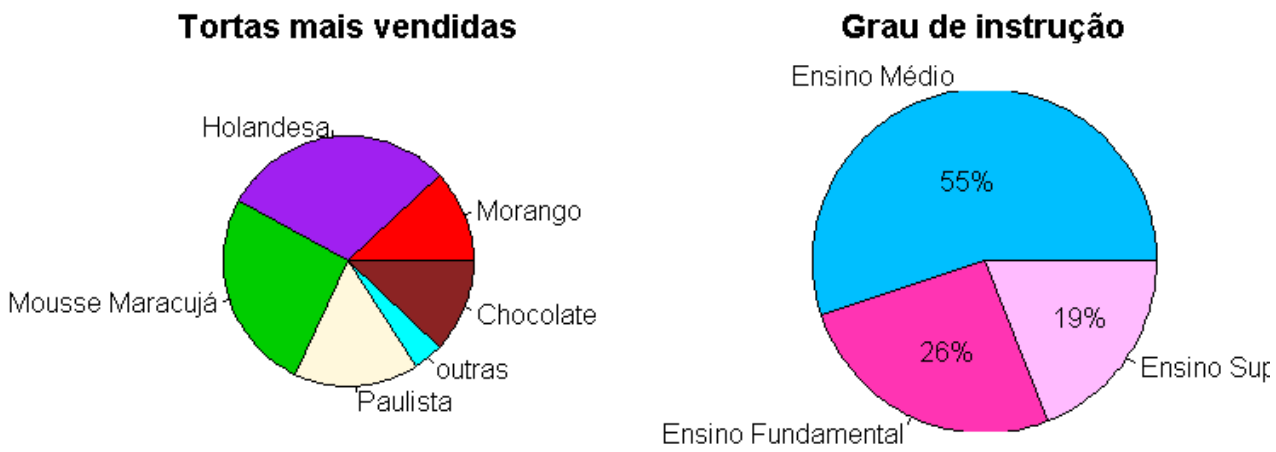

Figura 8: Gráfico de pizza

6.3 Gráfico Histograma

O histograma é uma representação gráfica da distribuição de frequência de uma determinada variável, normalmente um gráfico de barras verticais. O histograma é utilizado quando a variável em estudo é "contínua".

Suponha que temos um vetor de dados ou um *dataset* com n variáveis, e desejamos ter uma noção da distribuição de uma determinada variável. No R o comando utilizado para construir um histograma é a função *hist()*. Exemplo:

```
x=rnorm(1000,0,1)
hist(x, main="Histograma da variável x", ylab="Frequência")

y=rnorm(5000,20,1)
hist(y, main="Histograma da variavel Idade", prob=T, xlab = "Idade",
     ylab="Densidade", xlim=c(15,25), col="Darkred", breaks = 25)
```

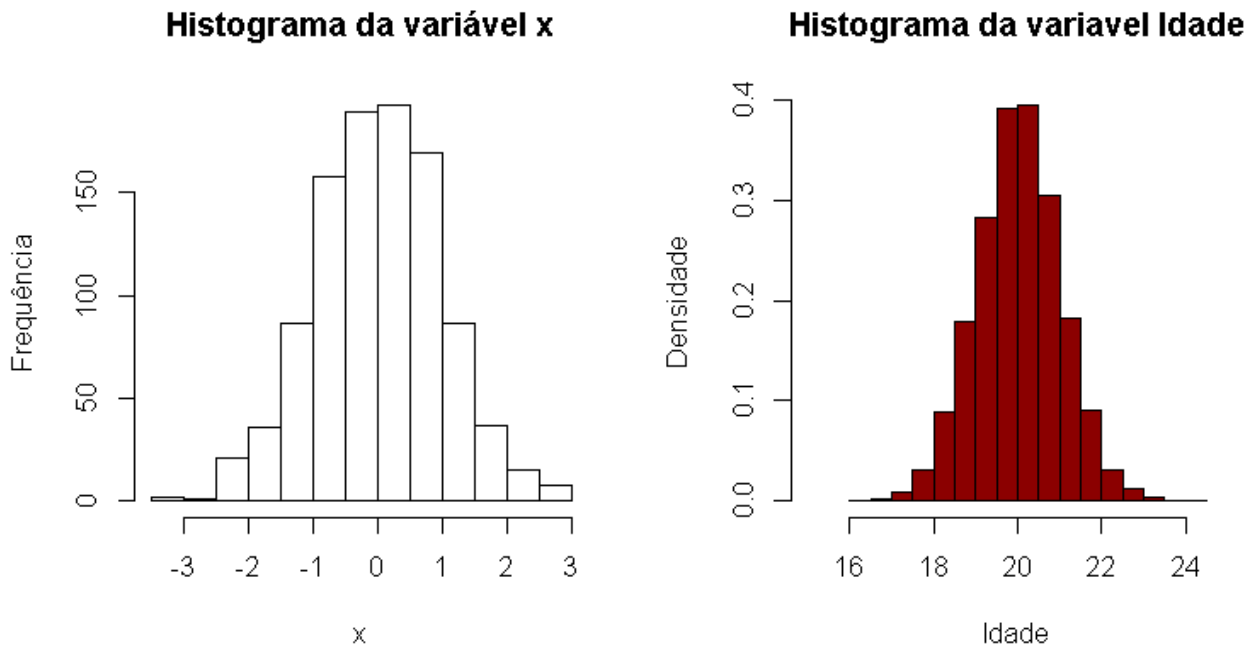

Figura 9: Gráfico histograma 1

```
r=rnorm(5000,20,1)
hist(r, main="Histograma da variavel Idade", prob=T, xlab = "Idade",
ylab="Densidade", xlim=c(15,25), col="DeepPink2", breaks = 65)

w=rnorm(500,4,1)
hist(w, main="Histograma da variavel W", prob=T, xlab = "Valores de w,
ylab="Densidade", xlim=c(0,8), col="Green3", breaks = 10)
```

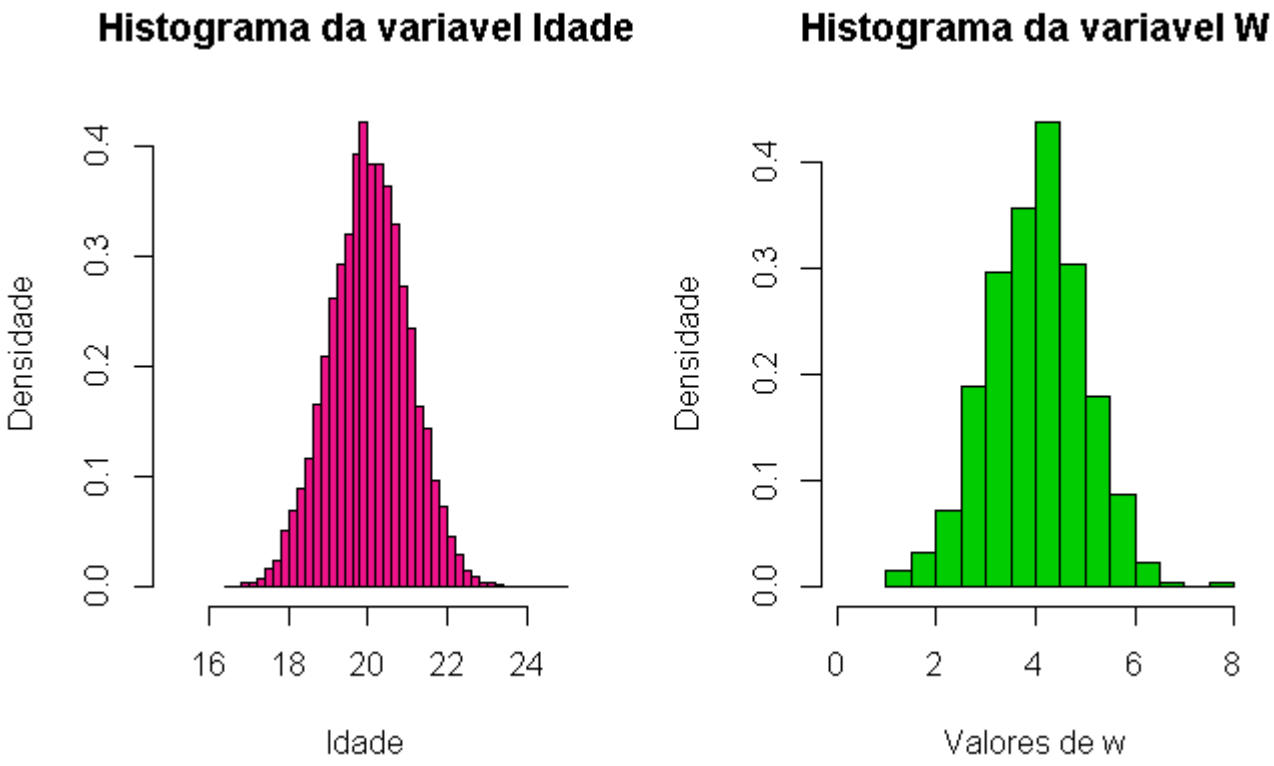

Figura 10: Gráfico histograma 2

6.4 Gráfico Boxplot

O boxplot é um gráfico que possibilita representar a distribuição de um conjunto de dados com base em alguns de seus parâmetros descritivos, conhecidos como quantis, que são a mediana(q2), o quartil inferior(q1) e o quartil superior(q3).

Para construir um *boxplot* no R basta utilizar a função *boxplot()*. Exemplo:

```
a=rexp(100,1/2500)
boxplot(a, main="Boxplot da Variável A",xlab="Valores de a")

boxplot(a, main="Boxplot da Variável A", cex.main=2, cex.lab=1.5,
        xlab="Valores de a", pch=16, col="LightYellow4")
```

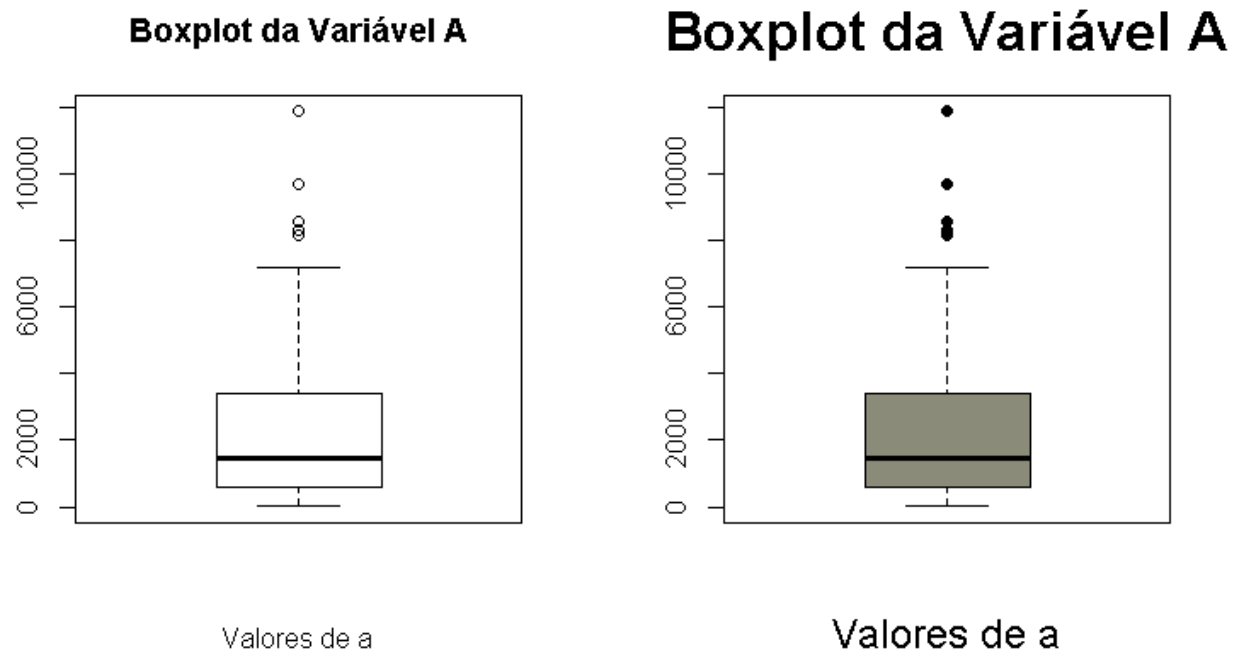

Figura 11: Gráfico boxplot 1

Em muitos casos não estamos interessados somente em observar o comportamento de uma variável em relação a apenas uma categoria, para esses casos não é nada mais complicado, utilizamos um "∼" entre as duas variáveis na hora de passar o argumento do boxplot.

```
notas = sample(0:10,30,T)
turma = c(rep("A",10),rep("B",10),rep("C",10))
boxplot(notas~turma,main="Boxplot de notas por turma de Cálculo I",
        xlab="Turmas", ylab="Notas", cex.main=1.5, cex.lab=1.1,
        col=c("SpringGreen1","IndianRed2","Purple2"))
```

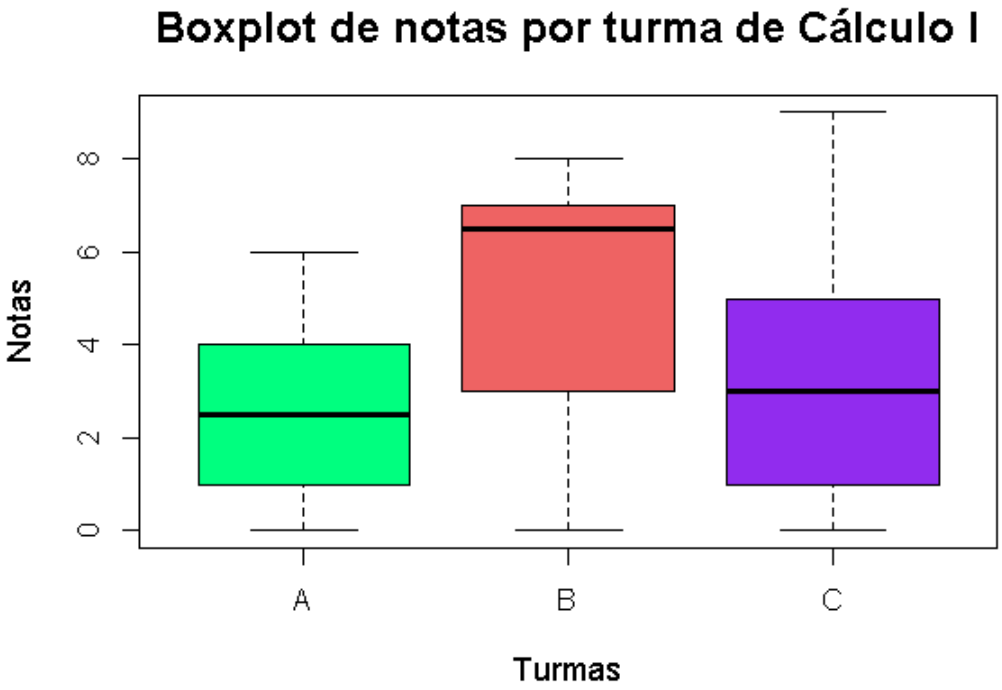

Figura 12: Gráfico boxplot 2

6.5 Gráfico de Dispersão

Como o nome já diz, são gráficos de pontos no eixo Y contra o eixo X. Na Estatística, Y geralmente é a letra usada em livros para indicar a variável resposta. Apesar de não ser uma norma, colocar variável resposta na vertical dos gráficos é um consenso entre a maioria dos estatísticos. Já X é chamada de variável independente e aparece no eixo horizontal.

É extremamente simples fazer um gráfico de pontos y entre x no R. A função utilizada para a elaboração deste gráfico é *plot()*, e necessita apenas de dois argumentos: o primeiro é o nome da variável do eixo X e o segundo da variável do eixo Y.

```
s = sample(35:100, 20, T)
r = 1:20
plot(r,s)
```

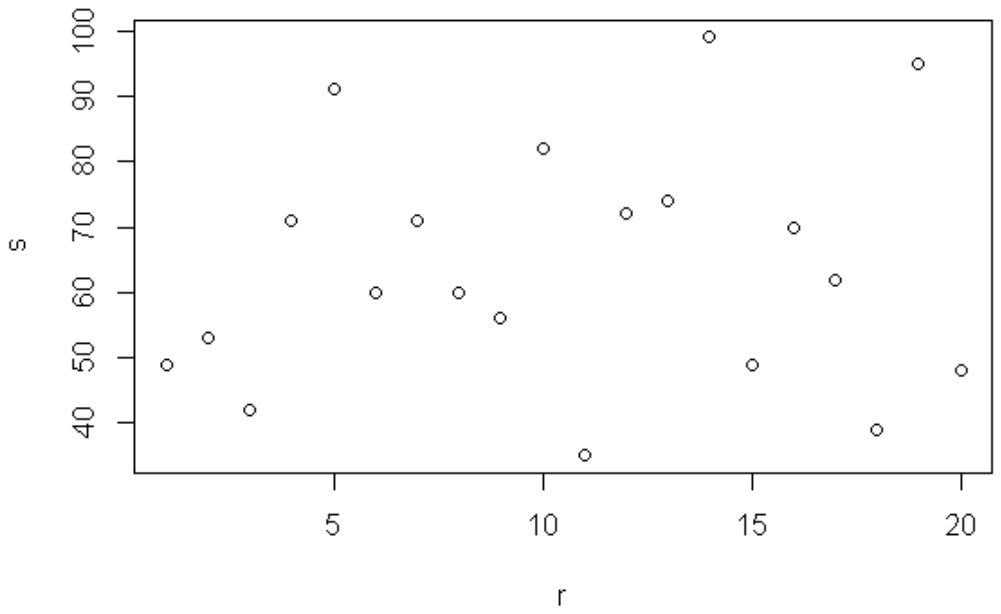

Figura 13: Gráfico de dispersão 1

Em muitos casos, precisamos detalhar melhor o que estamos fazendo, para isso produzimos gráficos cada vez mais elaborados e de fácil compreensão. Portanto a seguir podemos observar o que pode ser feito para elaborar melhores gráficos.

```
x = 1:100
y = 5 + 2 * x + rnorm(100, sd = 30)

plot(x, y, main = "Gráfico de pontos",
    sub = "Ordem dos pontos representados", type = "b",
    xlab = "Valores de r", ylab="Valores de s", pch=16, col="Blue")
```

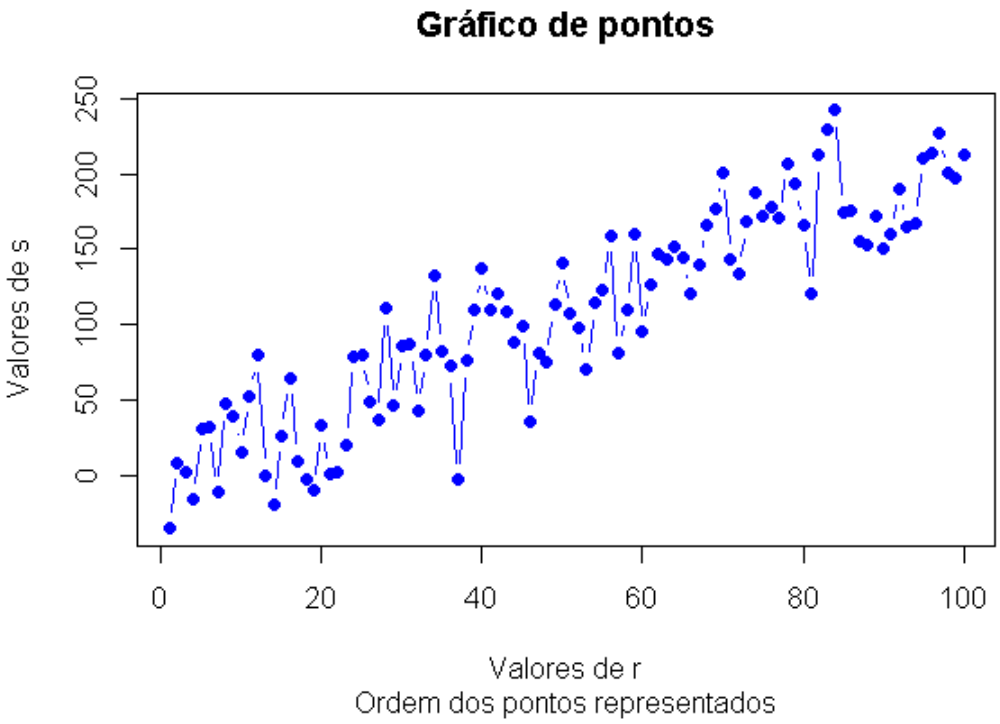

Figura 14: Gráfico de dispersão 2

```
plot(x, y, main = "Gráfico de pontos",
     sub = "Ordem dos pontos representados",type = "l",
     xlab="Valores de r", ylab="Valores de s", col="red", lwd=2)
```

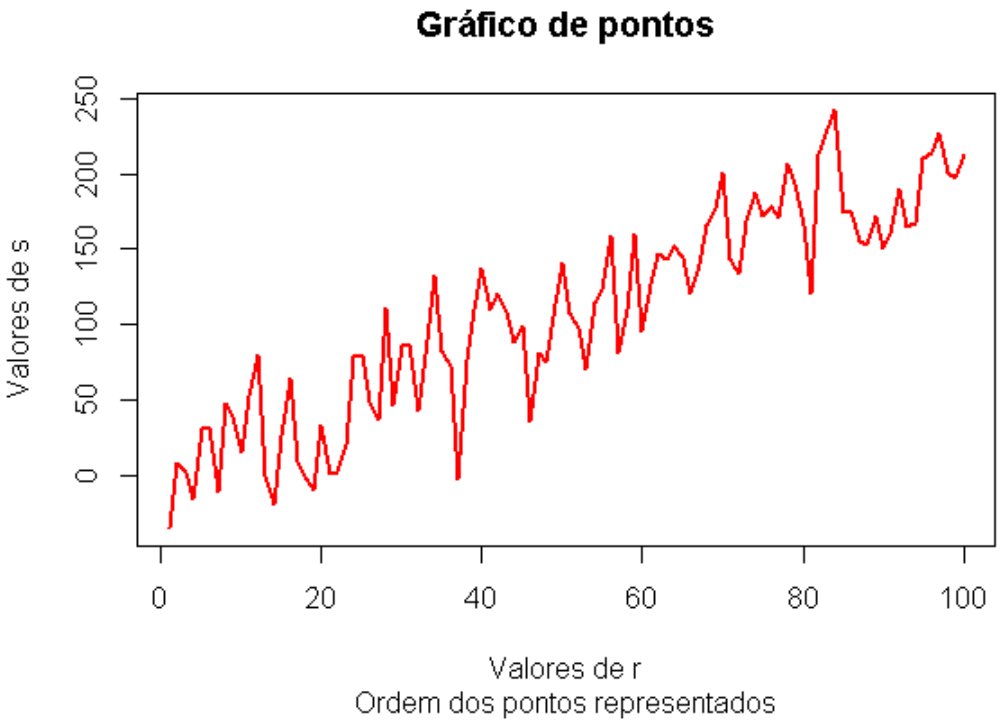

Figura 15: Gráfico de dispersão 3

```
plot(x, y, main = "Gráfico de pontos",
     sub = "Ordem dos pontos representados",type = "s",
     xlab = "Valores de r", ylab="Valores de s", col = "green", lwd=2)
```

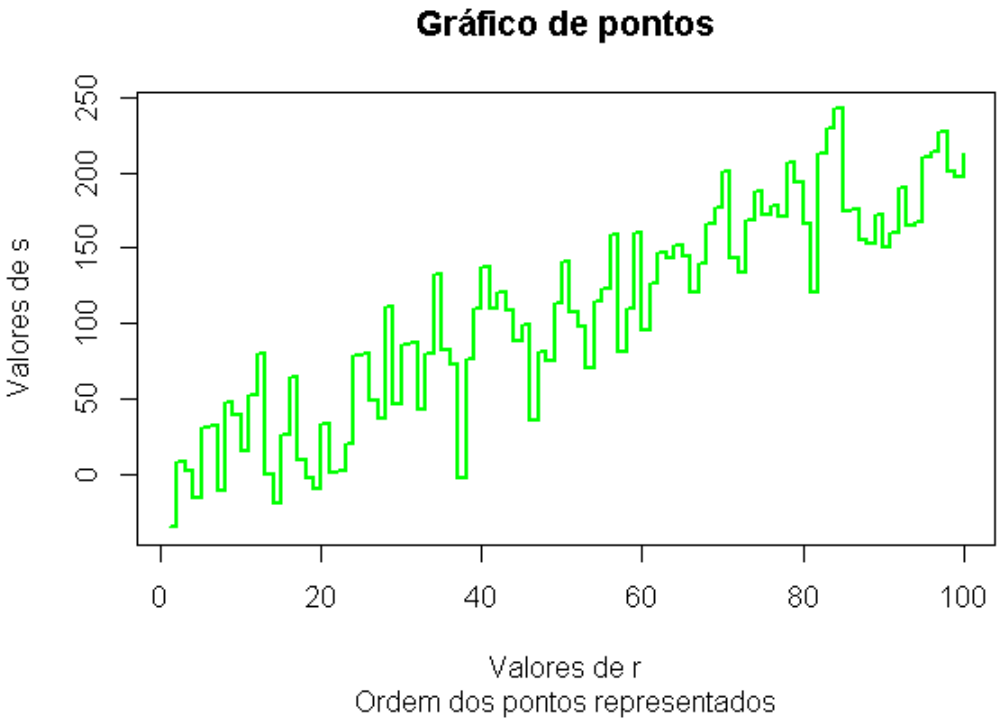

Figura 16: Gráfico de dispersão 4

Deixamos como dica, para a elaboração de gráficos mais elegantes, a utilização do pacote *ggplot2*. Não abordaremos esta ferramenta aqui, deixamos a cargo do leitor.