Manual básico de R para EAD0752

Geovana Lopes Batista

Manual básico de R para EAD0752

Geovana Lopes Batista

Sumário

Introdução

Este manual nasceu com o objetivo de ser um material de apoio para os alunos da disciplina EAD0752(Técnicas Estatísticas de Discriminação) da Faculdade de Economia, Administração e Contabilidade da Universidade de São Paulo. Ele é resultado do Projeto de Ensino intitulado “Modelos Estatísticos na Área de Negócios” para o Programa Unificado de Bolsas (2018) da USP; o qual ocorreu sob orientação da Prof Daielly Nassif Mantovani.
O uso deste documento é livre e gratuito.

R e RStudio

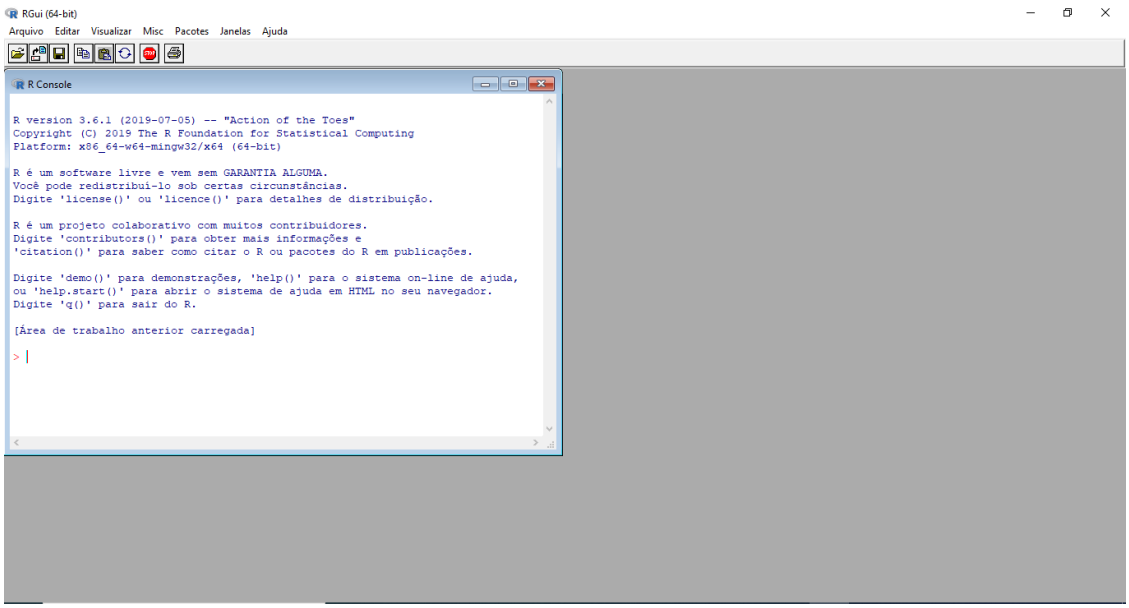
O R e o RStudio são softwares gratuitos, eles oferecem um vasto campo de aplicações e vêm sendo cada vez mais utilizados pela comunidade de Estatística e Ciência de Dados. Ambos já têm distribuições compiladas para Linux, Mac OS e Windows.

Lembrando que, para utilizar o RStudio, é necessário que o R já esteja instalado no seu computador.

Instalando o R e o RStudio

Para instalar o R, baixe a versão compatível com o seu computador em: https://cloud.r-project.org/

O R, por padrão, vem com uma interface gráfica para o usuário (Graphical User Interface – GUI).



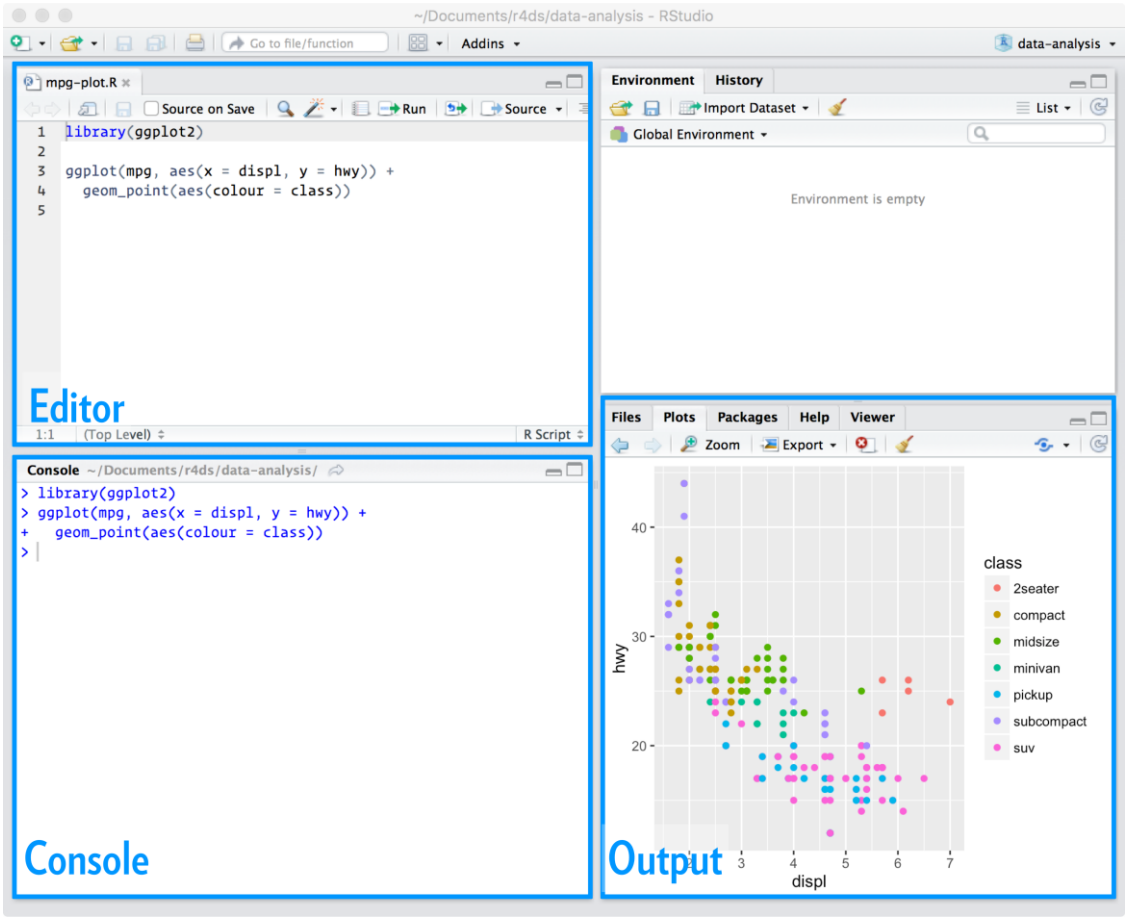
RGUI

O Rstudio tem algumas vantagens em relação ao RGUI, como por exemplo:

- Autocomplete;
- Match automático de parênteses e chaves;

- Interface intuitiva para objetos, gráficos e script;
- Criação de “projetos” com interface para controle de versão;
- Facilidade na criação de pacotes;
- Interação com HTML, entre outras.

Para instalar o RStudio, baixe a versão adequada para seu computador em: https://www.rstudio.com/products/rstudio/download/



RStudio

- Editor/Scripts: é onde você escreverá os seus códigos;
- Console: é onde você verá a maior parte dos resultados dos seus códigos;
- Environment: painel com todos os objetos criados na sessão.
- Output:
  - Files: mostra os arquivos no diretório de trabalho.
  - Plots: painel onde você verá os seus gráficos.
  - Help: janela onde a documentação das funções será apresentada.
  - Packages: onde você verá os seus pacotes instalados.

Atualizado o R e o RStudio

Para atualizar o R, você pode rodar o seguinte comando:

```
install.packages("installr")

library (installr)

updateR() #Atualiza a versão do R
```

Para atualizar o RStudio, é necessário acessar a página de download e baixar a versão mais recente. https://www.rstudio.com/products/rstudio/download/

Você pode verificar se há atualizações disponíveis em *Help > Check for Updates* na barra superior da sua tela do RStudio.

Como obter ajuda

A probabilidade de que alguém não precise de ajuda criando um script é baixíssima, até mesmo os mais experientes em algum momento precisam de ajuda com algo. O lado bom é que a comunidade R é muito colaborativa e o erro que está aparecendo na sua tela possivelmente já apareceu na tela de outra pessoa também.

Mas como posso pedir ajuda?

1. No próprio R:
- Utilizando a função *help(pacote)*
- Utilizando o equivalente *'?pacote'*

Ele trará a documentação do pacote e possivelmente algum exemplo de utilização.

2. No Stackoverflow

O https://stackoverflow.com/ e o https://pt.stackoverflow.com/ (português) são comunidades de perguntas e respostas sobre programação, incluindo o R, é claro.

Fique atento à algumas dicas:

- Confira se a sua dúvida já não foi respondida em outro momento;
- Seja claro na sua pergunta;
- Dê um exemplo do código que está gerando o erro;
- O seu exemplo precisa ser reprodutível para que outras pessoas o consigam testar e te ajudar.

3. Outras pessoas. Não tenha vergonha, tire sua dúvida com outras pessoas e ajude sempre que puder.

4. Github e a comunidade do RStudio (https://community.rstudio.com/).

5. Cheatsheets são folhas de dicas muito úteis, tanto para aprender como usar recursos, como para se ter como folha de cola. Descubra as Cheatsheets em *Help>Cheatsheets*.

Acessando bases de dados públicas

Nessa caminhada, as bases de dados serão suas aliadas e, há muitas que são públicas e de fácil acesso.

Você pode consegui-las no próprio R, com o comando *data()* ou *library(help = "datasets")*. Com ele você consegue ver uma lista de todas as bases de dados disponíveis no R e a descrição de cada uma delas.

No Kaggle (https://www.kaggle.com/), que é uma plataforma de aprendizado e competição de ciência de dados; onde também é possível conseguir bases de dados para treino e aprendizado.

Além disso, há portais governamentais como, por exemplo, o *dados.gov.br* que disponibiliza dados para uso público.

Atalhos úteis

Atalhos de teclado são muitos úteis quando se trata do R, você pode encontrar todos esses atalhos em *Tools > Keyboards Shortcuts Help* ou 'Alt Shift K'. Os principais atalhos de teclado são:

- CTRL+ENTER: roda a linha selecionada no script.
- ALT+-: (<-) sinal de atribuição.
- CTRL+SHIFT+M: (%>%) operador pipe.
- CTRL+1: muda o cursor para o script.
- CTRL+2: muda o cursor para o console.

Conceitos Básicos

Boas Práticas

- Tenha sempre um backup dos seus códigos e bases de dados originais.
- Comente os seus códigos, é só usar o "#". Isso é bom para você entender o seu código no futuro e para que outras pessoas consigam interpretá-los.
- Crie variáveis com nomes coerentes.
- Tenha um código limpo, organizado e legível, não use linhas de comando muito grandes.
- Instale e carregue os pacotes no começo do seu script.

Pacotes

Os pacotes configuram uma importante parte do R. Apesar dessa linguagem ter bastante funções nativas e pacotes pré-instalados, você dificilmente não usará um pacote.

Alguns dos pacotes mais utilizados e populares são o ggplot2 (gráficos), dplyr (manipulação de data.frames), tidyr (transformação de data.frames), lubridate (manipulação de datas), stringr (manipulação de strings), knitr/rmarrkdown (relatórios), caret (modelagem estatística).

Para usar um pacote, você precisa instalar e carregar esse pacote. A instalação é feita uma única vez, mas você precisará carregá-lo sempre que for utilizar.

Você pode instalar um *package* das seguintes formas:

Pelo CRAN (Comprehensive R Archive Network), a forma mais usual;

```
install.packages("nome-do-pacote")
```

Pelo Github;

```
devtools::install_github("nome-do-repositorio/nome-do-pacote")
```

E via arquivo.

```
zip/.tar.gz: install.packages("C:/caminho/nome-do-pacote.zip", repos =
NULL).
```

Para carregar, basta rodar o seguinte comando:

```
library(nome-do-pacote)
```

Operações Matemáticas/Estatísticas Básicas

Muito provavelmente, em algum momento, você precisará fazer algumas operações com o seu conjunto de dados. Para fazer isso, no R, é muito simples.

Veremos abaixo alguns operadores matemáticos e estatísticos básicos:

Operador	Descrição
+	Operador de soma
-	Operador de subtração
*	Operador de multiplicação
/	Operador de divisão
^	Operador de potenciação
%%	Resto da divisão
%/%	Parte inteira da divisão
log(x)	Log
log10(x)	Log base 10
sqrt(x)	Raíz quadrada
abs(x)	Módulo ou valor absoluto
summary(x)	Resumo dos dados
mean(x)	Média
sd(x)	Desvio padrão
max(x)	Máximo
min(x)	Mínimo
quantile(x)	Quartis
cor(x)	Correlação linear
var(x)	Variância
cov(x)	Covariância

O pacote “stats”, nativo do R, traz inúmeras funções estatísticas. Para consultar essa lista de funções, rode a linha *library(help = “stats”)* e conheça todas elas.

Funções

Ao utilizar uma linguagem de programação, você tem a vantagem de automatizar processos e isso será possível, além de outras formas, pela criação de funções.
Veremos alguns exemplos simples de como criar uma função. A partir disso, você será capaz de criar a sua própria. A criatividade é livre!

A estrutura básica de uma função é a seguinte:

```
nomeDaFuncao <- function(argumento1, argumento2, argumento3 =
default3,  ...){

  # corpo da função: comandos a serem realizados.

  return(resultado) # opcional
}
```

- O comando function() indica a definição de uma função;
- os valores dentro dos parênteses de function() são os argumentos (ou parâmetros) da função. Argumentos podem ter valores default (padrão), que são definidos com o sinal de igualdade (no caso arg3 tem como default o valor default3);
- dentro das chaves está a estrutura da função, comandos a serem realizados;
- o comando return() encerra a função e retorna seu argumento. O return() é opcional, a função retorna o último objeto calculado caso ele não seja indicado.

Exemplo 1

```
# retorna o quadrado de um número
quadrado <- function(x){
  x^2
}

quadrado(5)

## [1] 25
```

Exemplo 2

```
#Função diz se número é ímpar ou par

parimpar <- function(x){
  if (x%%2 ==0){
    print("Par")
  }

  else {
    print("Ímpar")
  }

}

parimpar(7)

## [1] "Ímpar"
```

Leitura e Manipulação de Dados

A etapa de manipulação e limpeza dos dados é uma das mais importantes quando se trata de análise e ciência de dados; é também a mais trabalhosa e chata. Sim, pode ser meio chato mesmo, mas alguns pacotes e técnicas podem ajudar nesse processo.

Leitura de Dados

Com o R você consegue ler e manipular diversos tipos de arquivos, como por exemplo, .csv(Comma separated values), tabelas, .xls, html, json, xml, pdf, SPSS entre outros.

! Lembre-se sempre de manipular o seu arquivo em modo de leitura para ter o original como backup!

Arquivos de texto

```
texto <- readLines("Nome-do-seu-arquivo.txt") #Lê o seu arquivo de texto

text02 <- readLines("Nome-do-seu-arquivoo.txt",2) #Lê apenas 2 linhas do seu arquivo de texto

text03  <- readLines(file.choose()) # O 'file.choose' abre a janela de arquivos do seu computador para que você escolha o documento
```

Tabela

Lendo arquivos em formato de tabela.

```
tabela1 <- read.table("Caminho-do-seu-arquivo.txt", header=TRUE, sep="\t") #Arquivo separado por TAB

tabela2 <- read.table("Caminho-do-seu-arquivo.csv", header=TRUE, sep=",") #Aruivo separado por vírgula

tabela3 <- read.table("Caminho-do-seu-arquivo.txt", skip=4, header=TRUE, sep="\t") #O argumento 'skip' indica quantas linhas pular, caso necessário
```

CSV e separadores

O CSV é um dos formatos de arquivos mais utilizados. Neste formato, o separador normalmente é uma vírgula ‘,’ (csv) ou ponto e vírgula ‘;’ (csv2). Ele pode ter um cabeçalho com os nomes das colunas ou não.

```
arq <- read.csv("Caminho-do-seu-arquivo.csv", header=TRUE) #Separador
,

arq2 <- read.csv2("Caminho-do-seu-arquivo.csv", header=TRUE)
#Separador ;

arq3 <- read.delim("Caminho-do-seu-arquivo.csv", header=TRUE, sep =
"Insira o separadador") #Delim lê qualquer separador, basta indicar
qual é o separador

head(arq) #Mostra as 6 primeiras linhas do seu arquivo, incluindo o
cabeçalho
```

XLS

```
install.packages("xlsx") #Instalando o pacote que faz leitura de xls

library(xlsx)

data <- read.xlsx("Caminho-do-seu-arquivo.xlsx", "Nome-da-aba")
```

Excel

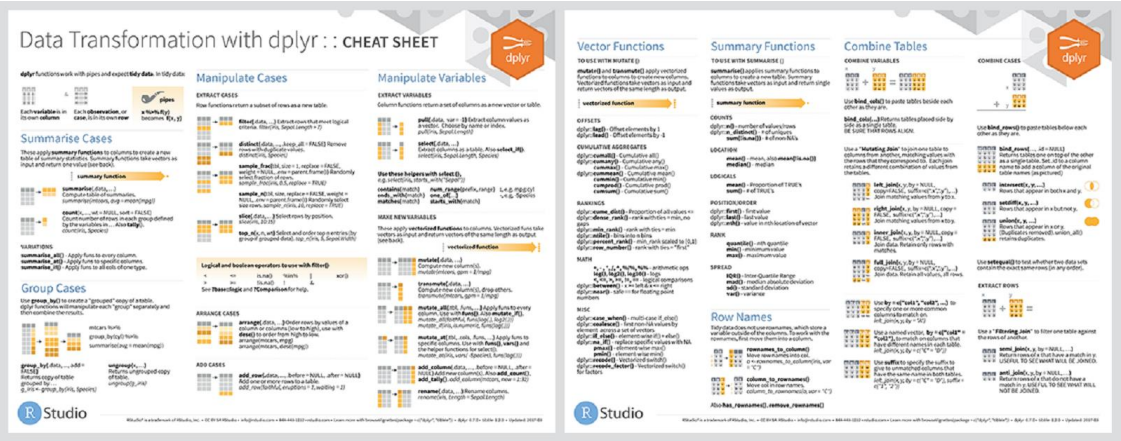
```
exl <- read_excel("Caminho-do-seu-arquivo.xlsx",
                  sheet = "nome-da-aba",
                  col_types = c("date", "text", "text", "numeric",
"numeric", "text"))

#Indique a aba apenas se necessário
#'col_types' discrimina o tipo de variável em cada coluna.
```

Manipulação de Dados

Para a parte de manipulação de dados, iremos usar o pacote **dplyr**.

O dplyr faz parte do tidyverse, um ecossistema de pacotes projetados com APIs comuns e uma filosofia compartilhada.

Lembrando que também é possível manipular dados com o RBase, mas os pacotes deixam esse tipo de tarefa mais fácil.



Data Transformation with dplyr :: Cheat Sheet

Vamos falar um pouco sobre as principais funções desse pacote:

- filter( ) - filtro, funciona como o filtro do Excel e de outras ferramentas;
- select( ) - seleciona colunas;
- mutate( ) - cria/modifica colunas;
- arrange( ) - ordena a base de acordo com o critério determinado;
- summarise( ) - sumariza a base;
- count( ) - também pode ser usada para sumarizar em relação à frequência;
- group_by( ) - agrupamento.

Importante!!!

Vocês notarão, nos exemplos a seguir, o comando “%>%”. Este é o pipe.

O atalho de teclado para inserir o pipe é “CTRL SHIFT m”.

Esse comando cria uma ordenação, é como se o argumento da esquerda puxasse o da direita através do pipe. Com os códigos, o entendimento do que o pipe faz acaba sendo mais claro e intuitivo.

Instalando e carregando o pacote

```
install.packages("tidyverse") # A forma mais fácil, pois instalada todos os recursos do tidyverse

install.packages("dplyr") #Instala apenas o dplyr

devtools::install_github("tidyverse/dplyr") ## Você pode instalar puxando do github

library(dplyr) #Carregando o pacote
```

Para testar algumas funções, vamos usar o dataset “iris”, que faz parte dos datasets pré-carregados do R.

```
dados <- iris #Carregando o dataset
```

Filter ( )

```
virginic <- dados %>% filter(Species=="virginica") #Filtrando apenas
virginica na coluna de espécies

head(virginic) #Visualizando as 6 primeiras linhas
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          6.3         3.3          6.0         2.5 virginica
## 2          5.8         2.7          5.1         1.9 virginica
## 3          7.1         3.0          5.9         2.1 virginica
## 4          6.3         2.9          5.6         1.8 virginica
## 5          6.5         3.0          5.8         2.2 virginica
## 6          7.6         3.0          6.6         2.1 virginica
```

Select ( )

```
selecionando <- dados %>% select(Species, Petal.Width) #Selecionando
apenas 2 colunas

head(selecionando) #Visualizando as 6 primeiras linhas
```

```
##   Species Petal.Width
## 1  setosa         0.2
## 2  setosa         0.2
## 3  setosa         0.2
## 4  setosa         0.2
## 5  setosa         0.2
## 6  setosa         0.4
```

Mutate ( )

```
diferenca <- dados %>% mutate(dif = Sepal.Width - Petal.Width)
#Criando uma coluna a partir de um cálculo, nova coluna "dif"

head(diferenca) #Visualizando as 6 primeiras linhas
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species dif
## 1          5.1         3.5          1.4         0.2  setosa 3.3
## 2          4.9         3.0          1.4         0.2  setosa 2.8
## 3          4.7         3.2          1.3         0.2  setosa 3.0
## 4          4.6         3.1          1.5         0.2  setosa 2.9
## 5          5.0         3.6          1.4         0.2  setosa 3.4
## 6          5.4         3.9          1.7         0.4  setosa 3.5
```

Arrange ( )

```
crescente <- dados %>% arrange(Petal.Width) #Ordenando de forma
crescente a partir da coluna selecionada

head(crescente) #Visualizando as 6 primeiras linhas
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          4.9         3.1          1.5         0.1  setosa
## 2          4.8         3.0          1.4         0.1  setosa
## 3          4.3         3.0          1.1         0.1  setosa
## 4          5.2         4.1          1.5         0.1  setosa
## 5          4.9         3.6          1.4         0.1  setosa
## 6          5.1         3.5          1.4         0.2  setosa
```

```
decrescente <- dados %>% arrange(desc(Petal.Width)) #Ordenando de
forma decrescente

head(decrescente) #Visualizando as 6 primeiras linhas
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          6.3         3.3          6.0         2.5 virginica
## 2          7.2         3.6          6.1         2.5 virginica
## 3          6.7         3.3          5.7         2.5 virginica
## 4          5.8         2.8          5.1         2.4 virginica
## 5          6.3         3.4          5.6         2.4 virginica
## 6          6.7         3.1          5.6         2.4 virginica
```

Summarise ( )

```
suma <- dados %>% summarise(sumarizando=mean(Petal.Width))

head(suma)
```

```
##   sumarizando
## 1    1.199333
```

Count ( )

```
contador <- dados %>% count(Species) #Contagem de espécies

contador
```

```
## # A tibble: 3 x 2
##   Species        n
##   <fct>      <int>
## 1 setosa        50
## 2 versicolor    50
## 3 virginica     50
```

Group_by ( )

```
grupo <- dados %>% group_by(Species)

head(grupo)
```

```
## # A tibble: 6 x 5
## # Groups:   Species [1]
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          5.1         3.5          1.4         0.2 setosa
## 2          4.9         3            1.4         0.2 setosa
## 3          4.7         3.2          1.3         0.2 setosa
## 4          4.6         3.1          1.5         0.2 setosa
## 5          5           3.6          1.4         0.2 setosa
## 6          5.4         3.9          1.7         0.4 setosa
```

Group_by e summarise

```
agrupando <- dados %>% group_by(Species) %>%
summarise(media=mean(Petal.Width)) #Agrupando espécies e calculando a
média da variável "Petal,Width"

agrupando #Visualizando o agrupamento
```

```
## # A tibble: 3 x 2
##   Species    media
##   <fct>      <dbl>
## 1 setosa     0.246
## 2 versicolor 1.33
## 3 virginica  2.03
```

O **dplyr** tem várias outras funções que são muito úteis no dia a dia e facilitam muito a manipulação de dados. Vale a pena pesquisar e explorar as inúmeras possibilidades que o pacote traz.

NA

Basta qualquer valor NA (Not Available), NaN (Not a Number) ou NULL (sem valor, diferente de NA ou NaN) para que o R não faça nenhuma operação e retorne um erro.

Sobre os valores faltantes:

- NA é *Not Available* (value). NaN é *Not a Number* (value).
- NaN é um NA, mas um NA não é um NaN.
- NA e NaN têm comprimento 1.
- NULL é um objeto especial com comprimento 0.
- NA não é um número 0.

Não existe uma única solução para tratar os NA's, isso vai depender do seu conjunto de dados, do seu conhecimento sobre eles e do que você está

disposto a perder em relação a eles.
NA’s podem ser oriundos de erro de digitação, perguntas não respondidas em pesquisas ou uma série de outros fatores.

Dito isso, vou listar algumas opções para tratamentos de NA’s e, cabe a você decidir o que vai fazer com eles.

Inicialmente, vamos criar um data.frame pequenininho com alguns NA’s, só para testar as funções.

```
dat <-data.frame(satisfacao = c("bom","muito bom","ruim","muito
bom","bom","ruim","bom"),nota =
c(4,5,1,NA,4,1,NA),nota2=c(5,9,2,8,3,1,7))

dat #Visualizando o dataframe

##   satisfacao nota nota2
## 1        bom    4     5
## 2  muito bom    5     9
## 3       ruim    1     2
## 4  muito bom   NA     8
## 5        bom    4     3
## 6       ruim    1     1
## 7        bom   NA     7
```

Agora, precisamos investigar se há algum NA ou não.

```
which(is.na(dat)) # Retorna a posição das variáveis que contêm NA's.

## [1] 11 14

is.na(dat) #retorna TRUE ou FALSE em cada uma das variáveis do seu
data frame. Não é muito prático quando se tem um conjunto de dados
muito grande.

##      satisfacao  nota nota2
## [1,]      FALSE FALSE FALSE
## [2,]      FALSE FALSE FALSE
## [3,]      FALSE FALSE FALSE
## [4,]      FALSE  TRUE FALSE
## [5,]      FALSE FALSE FALSE
## [6,]      FALSE FALSE FALSE
## [7,]      FALSE  TRUE FALSE

is.null(dat) #Verifica se o valor é nulo

## [1] FALSE
```

Agora que já investigamos os nossos dados e percebemos que há observações com NA, vamos fazer algumas transformações no nosso data.frame.

```r
na.omit(dat) #Omite os NA's do data frame

na.exclude(dat$nota) #Exclui os NA da coluna NOTA

dat[complete.cases(dat$nota),] #Retorna apenas os casos completos, é o mesmo que na.omit()
```

Podemos, também, substituir os valores faltantes pela média da coluna.

Note que o comando na.rm=TRUE remove o NA para fazer a operação desejada

```r
dat[is.na(dat$nota)] = mean(dat$nota,na.rm=TRUE)


dat
```

Substituindo por um número escolhido, nesse caso o 10.

```r
dat$nota[is.na(dat$nota)] = 10 #As observações com NA agora serão o número 10.
```

Com uma abordagem um pouco mais avançada, poderíamos também, prever o valor dos NA, Nan ou Null e substituí-los por essa estimativa.

```r
library(dplyr)

dat <-data.frame(satisfacao = c("bom","muito bom","ruim","muito bom","bom","ruim","bom"),nota = c(4,9,1,NA,4,1,NA),nota2=c(5,9,2,8,5,1,7))

dat2 <- na.omit(dat) #omitindo os nas


linear <- lm(nota~nota2+satisfacao, data = dat2)# criando uma regressão

df <- dat %>%
  mutate(nota=ifelse(is.na(nota), predict(linear,.), nota))  # Substituindo NA pelo valor previsto na regressão

df

##   satisfacao nota nota2
## 1        bom    4     5
## 2  muito bom    9     9
## 3       ruim    1     2
## 4  muito bom    9     8
## 5        bom    4     5
## 6       ruim    1     1
## 7        bom    4     7
```

Criando arquivos

Agora que a leitura e manipulação dos dados já foi feita, você já pode criar e exportar o seu arquivo com eles.
Serão apresentados alguns exemplos básicos de como exportar dados no R, mas você pode explorar outras opções com pacotes e funções adicionais.

Tabela

```
write.table(seus_dados, "nome_do_seu_arquivo.extensão", sep='|',
na="ND", eol="\r\n")
```

Neste caso, criamos um arquivo on o separador é "|", onde houver NA será inserido "ND" e teremos uma quebra de linha *eol* no formato do windows.

CSV e Separadores

```
write.csv(seus_dados, "nome_do_arquivo.csv") # Cria um arquivo csv

write.csv(seus_dados, "nome_do_arquivo.csv", na = "NA") # Insere "NA"
nos valores faltantes

write_delim(seus_dados, "nome_do_arquivo.csv", delim = "|") #Cria um
arquivo delimitado por "|"
```

XLS

```
install.packages("WriteXLS")

library(WriteXLS)

WriteXLS(seus_dados, "nome_do_arquivo.xls")

WriteXLS(c(seus_dados1, seus_dados2), "nome_do_seu_arquivo.xls",
SheetNames = c("nome_aba1", "nome_aba2")) #Cria um XLS com duas abas
```

Visualização de Dados

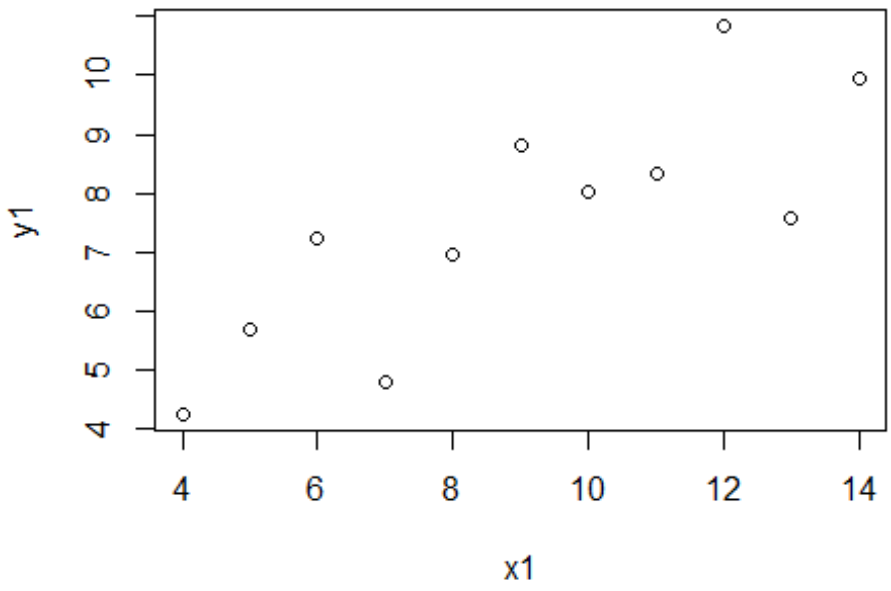
Gráficos usando RBase

Para criarmos alguns gráficos com o RBase, vamos usar o dataset "anscombe", já existente no R.

Plot

Por *default* o comando plot faz um gráfico de pontos, como o abaixo.

```
attach(anscombe) #Carregando os dados

plot(x1,y1)
```



```r
plot(x1,y1, xlab = "Meu eixo x", ylab = "Meu eixo y", main = "Meu gráfico de dispersão", sub = "Subtítulo do meu gráfico")
```

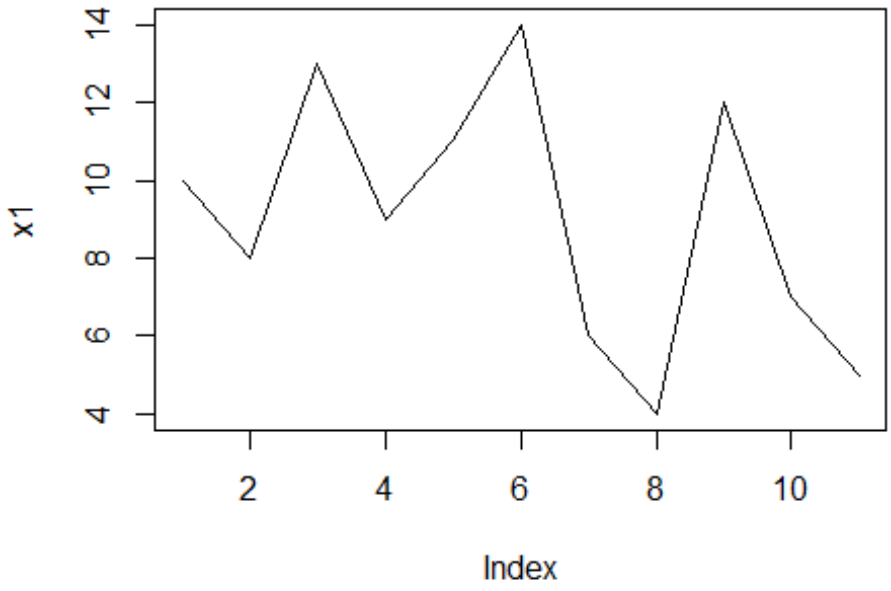

```r
#ylab = Nome do eixo y
#xlab = Nome do eixo x
#main = Título do gráfico
#sub = Subtítulo do gráfico
```

Podemos adicionar outros parâmetros ao gráfico com o comando “type”:

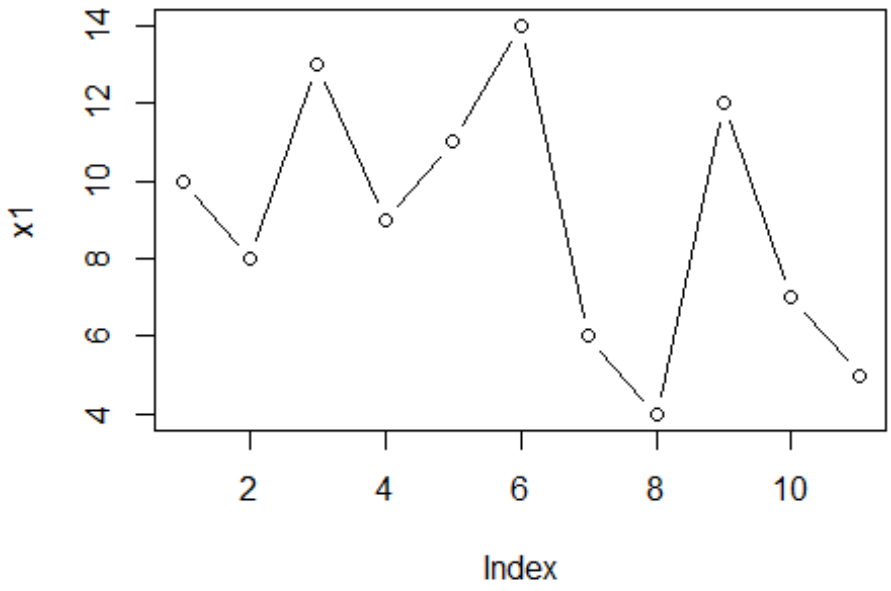
type = “p” - pontos, type = “l” - linhas, type = “b” - (both) ambos linhas e pontos, type = “h” - ‘histograma’ (ou ‘high-density’) de linhas verticais, type = “n” - não plota nada.

Além desses, existem outros “type’s”. Digite help(plot) e você verá mais opções.

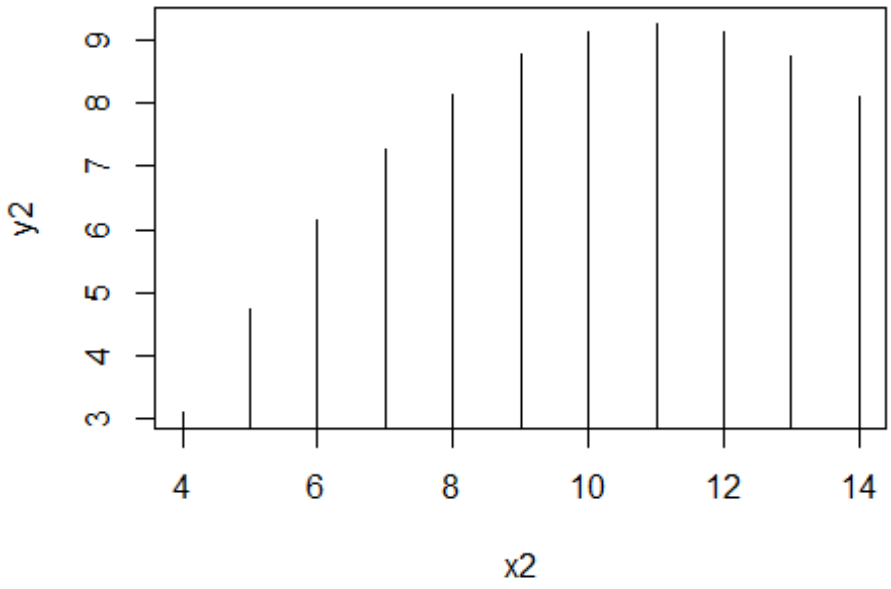
```r
attach(anscombe)

## The following objects are masked from anscombe (pos = 3):
##
##     x1, x2, x3, x4, y1, y2, y3, y4

plot(x1, type="l") #linhas
```



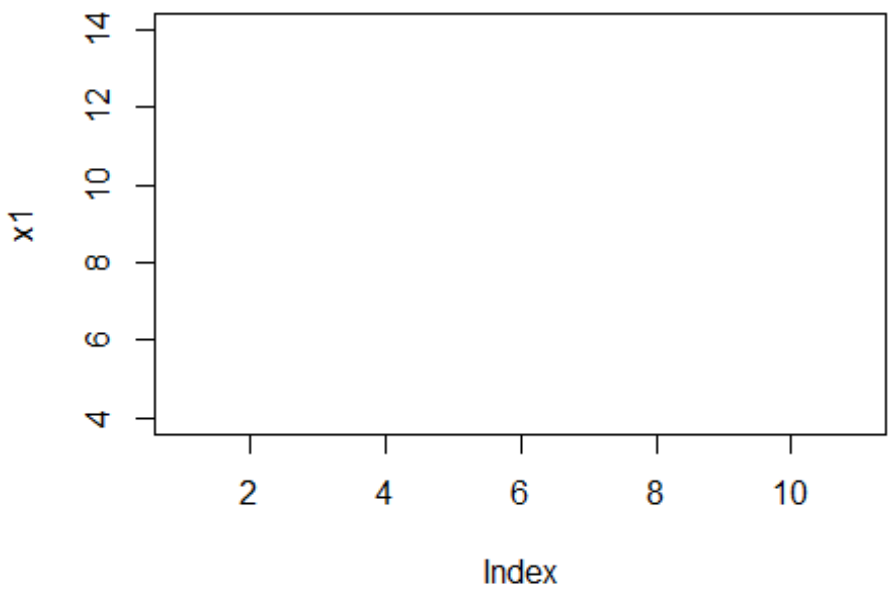
```r
plot(x1, type = "b") #Linhas e pontos
```




```r
plot(x2,y2,type="h") #Linhas verticais
```



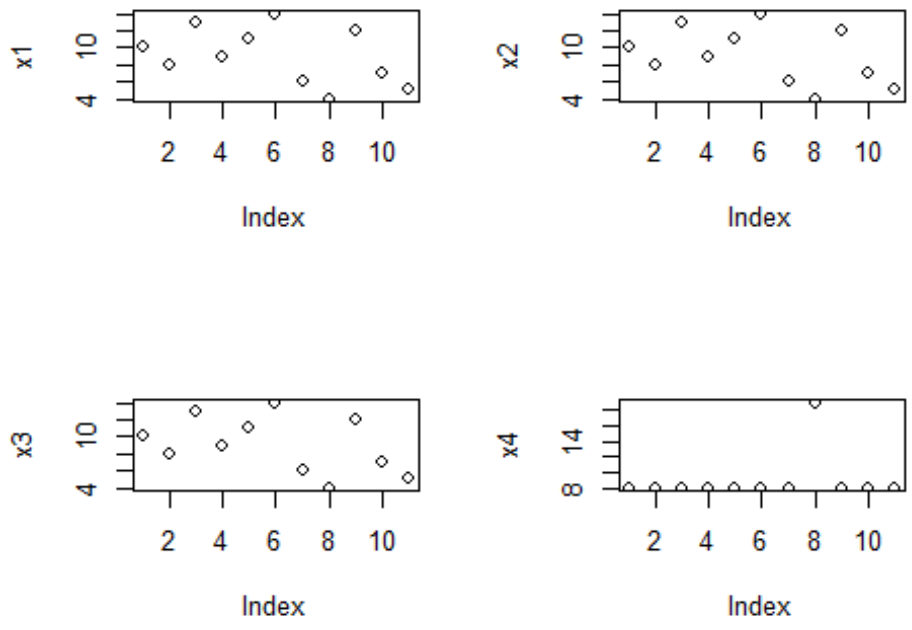
```r
plot(x1, type="n") #não plota nada
```



Criando figuras com mais de um gráfico.

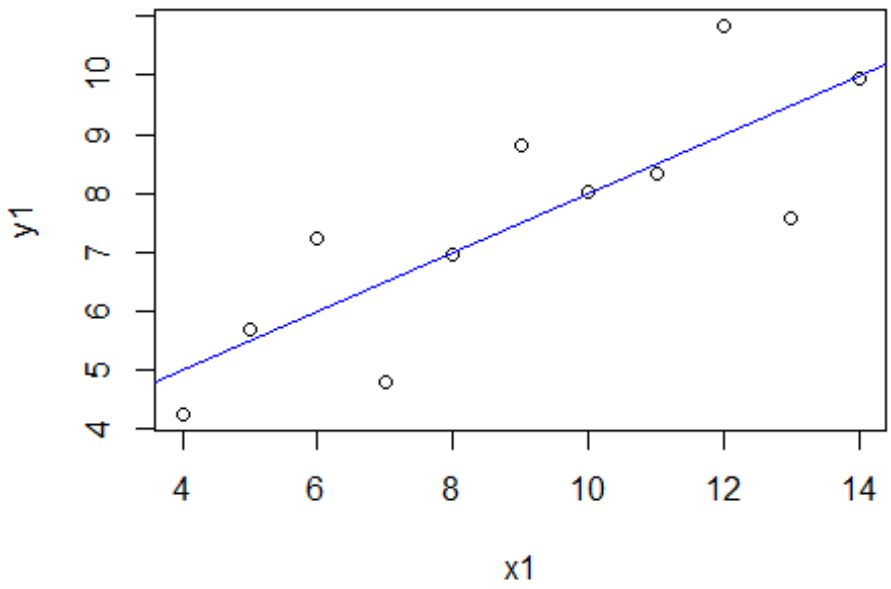
par(mfrow=c(número de linhas, número de colunas))

```
attach(anscombe)

## The following objects are masked from anscombe (pos = 3):
##
##     x1, x2, x3, x4, y1, y2, y3, y4

## The following objects are masked from anscombe (pos = 4):
##
##     x1, x2, x3, x4, y1, y2, y3, y4

par(mfrow=c(2,2)) #2 linhas e 2 colunas

plot(x1)

plot(x2)

plot(x3)

plot(x4)
```



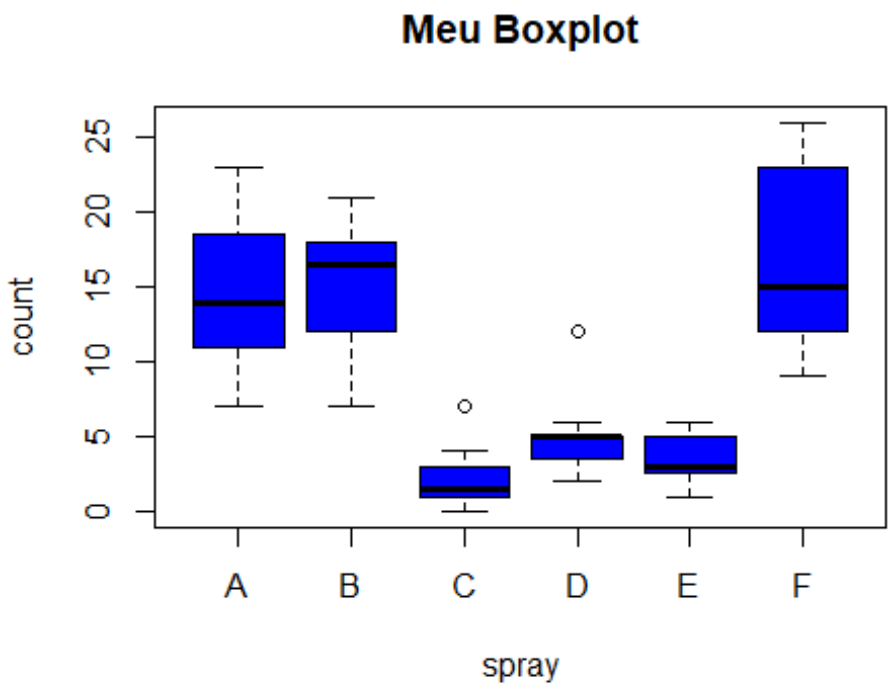
Abline ( )

```
attach(anscombe)

## The following objects are masked from anscombe (pos = 3):
##
##     x1, x2, x3, x4, y1, y2, y3, y4

## The following objects are masked from anscombe (pos = 4):
##
##     x1, x2, x3, x4, y1, y2, y3, y4

## The following objects are masked from anscombe (pos = 5):
##
##     x1, x2, x3, x4, y1, y2, y3, y4

plot(x1,y1)

abline(lm(y1~x1), col="blue") #Insere uma linha a partir de uma
regressão linear
```



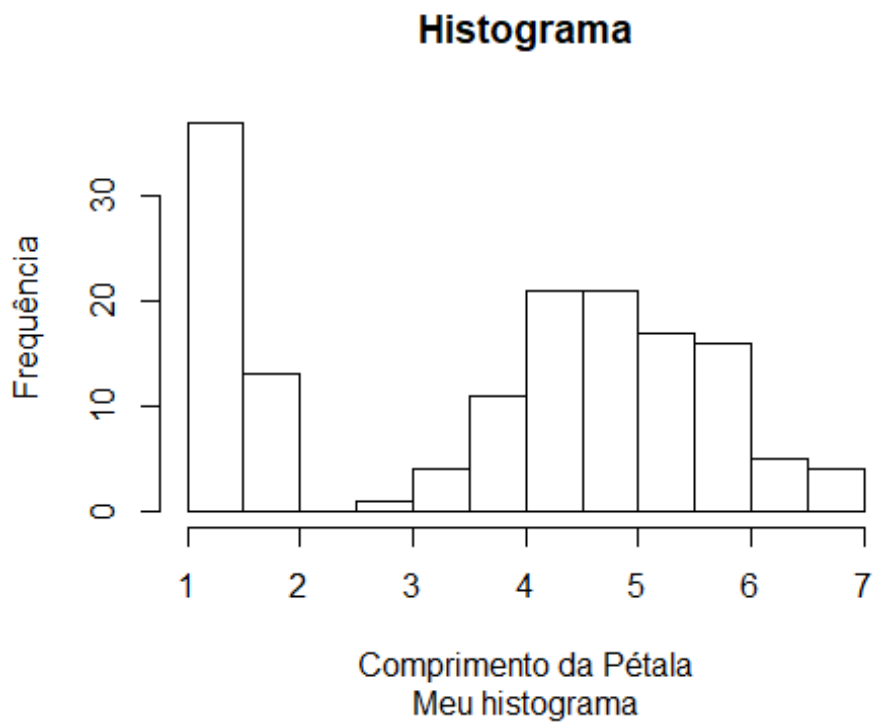
Boxplot

```
boxplot(count ~ spray, data = InsectSprays, col = "blue", main= "Meu
Boxplot")
```



Histograma

```
hist(iris$Petal.Length, ylab = "Frequência", xlab = "Comprimento da
Pétala", main = "Histograma", sub="Meu histograma")
```



Gráficos usando o pacote ggplot2

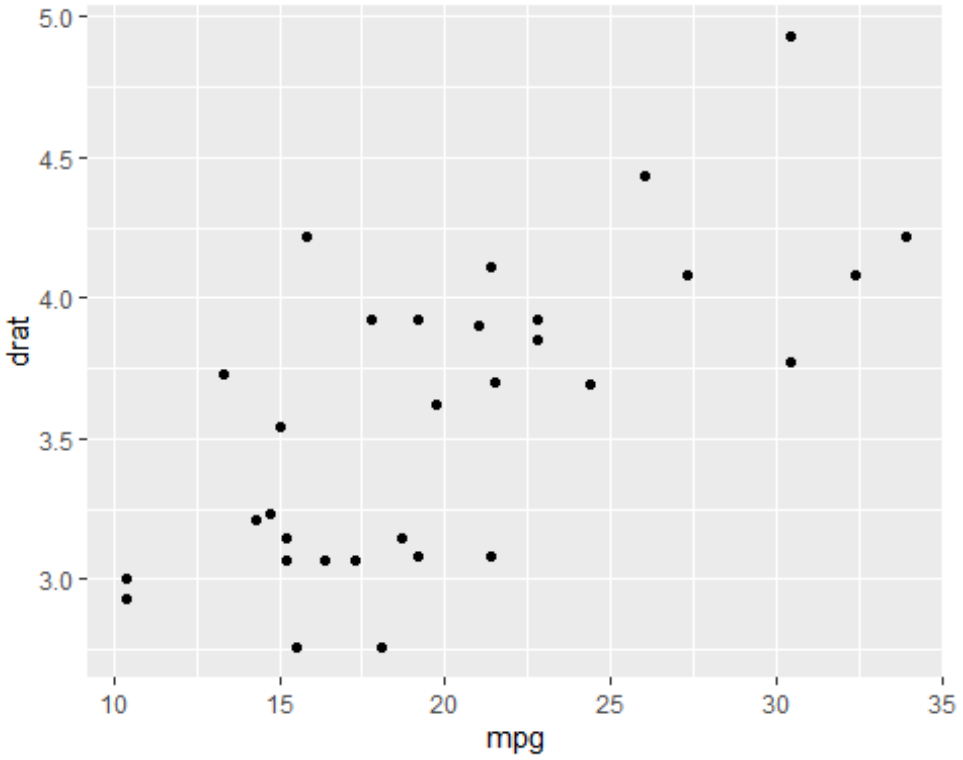
Ilustração Allison Horst https://twitter.com/allison_horst

O ggplot2 é fruto da tese de doutorado de Hadley Wickham. Ele foi criado em 2005 e é uma implementação do *The Grammar of graphics* deLeland Wilkinson. Esse pacote segue uma lógica de camadas, similar ao encadeamento do Pipe (%>%), mas utiliza o operador + para isso.

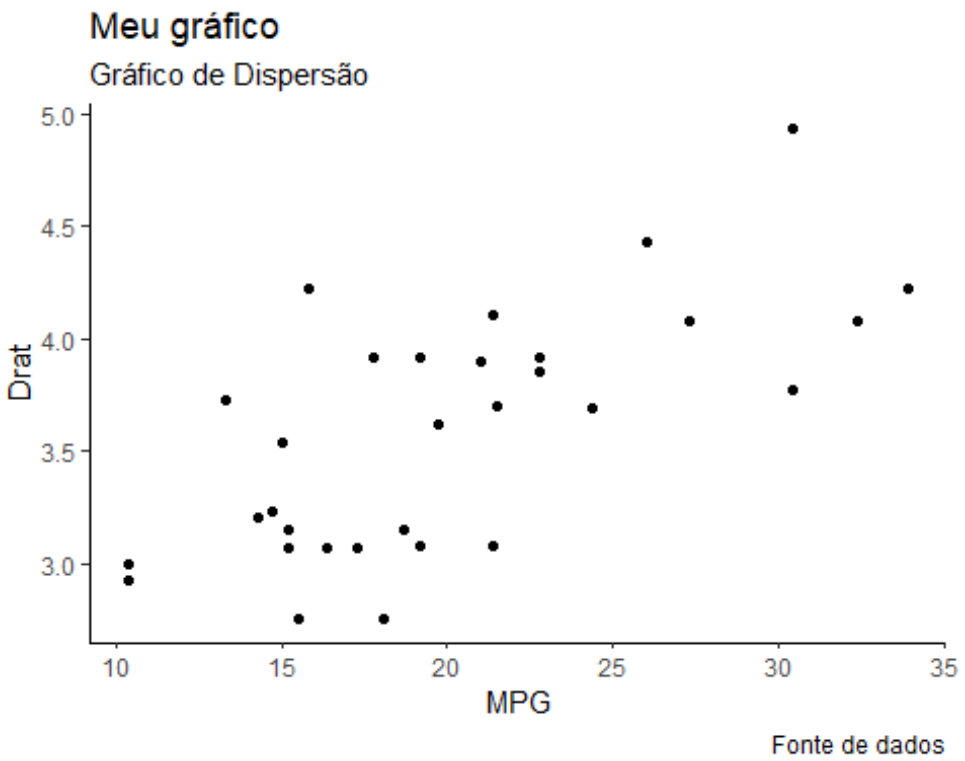
```
#Instalando o pacote

install.packages("ggplot2")

#Carregando o pacote

library(ggplot2)
```

Gráfico de Pontos

```
ggplot(mtcars, aes(mpg, drat)) +
  geom_point()
```

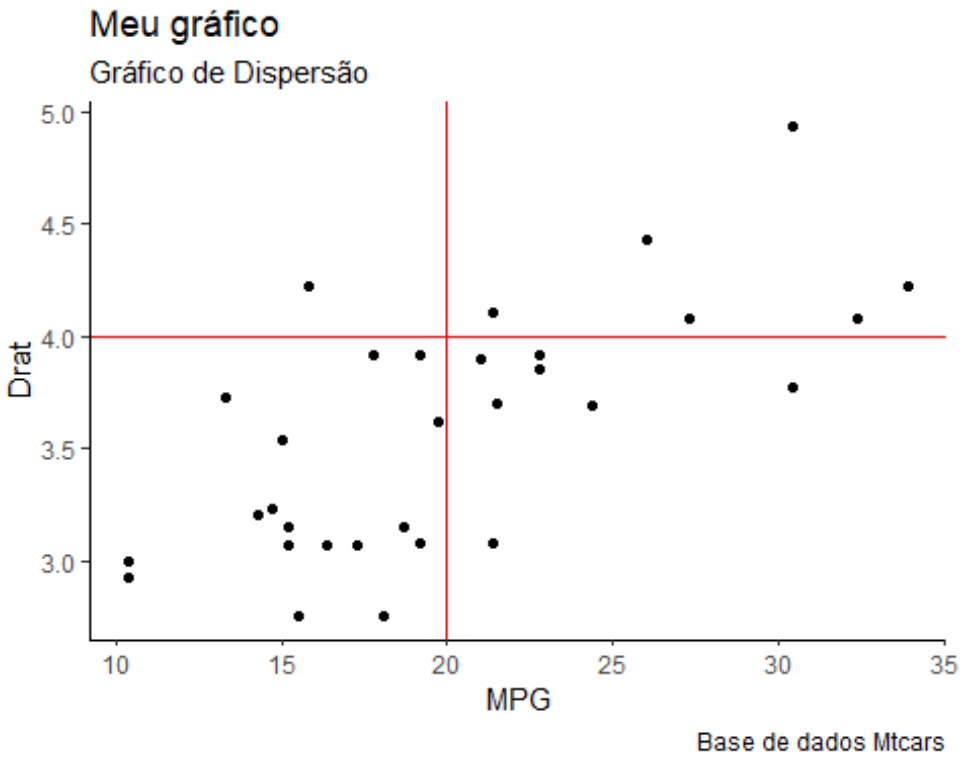


```r
ggplot(mtcars, aes(mpg, drat)) +
  geom_point() +
  theme_classic() +
  labs(
    title = "Meu gráfico",
    subtitle = "Gráfico de Dispersão",
    x = "MPG",
    y = "Drat",
    caption = "Fonte de dados")
```



title = “Título do gráfico”, subtitle = “Subtítulo do gráfico”, x = “Nome do eixo x”, y = “Nome eixo y”, caption = “Fonte de dados”

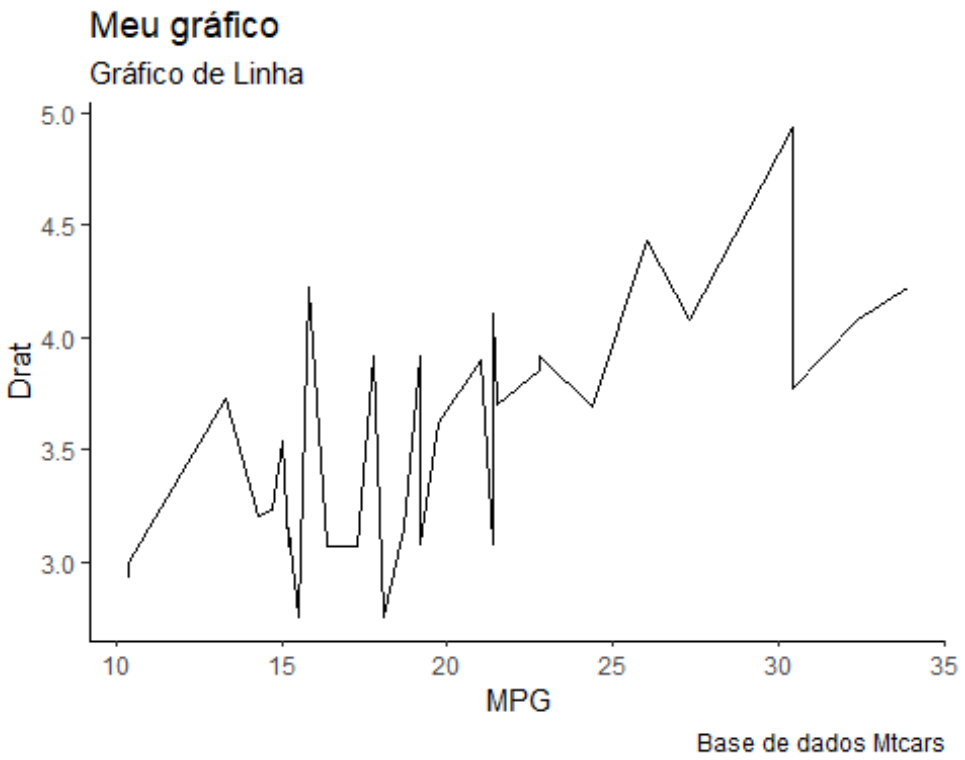
```r
ggplot(mtcars, aes(mpg, drat)) +
  geom_point() +
  theme_classic() +
  labs(
    title = "Meu gráfico",
    subtitle = "Gráfico de Dispersão",
    x = "MPG",
    y = "Drat",
    caption = "Base de dados Mtcars") +
  geom_hline(yintercept = 4, col="red")+
  geom_vline(xintercept = 20, col="red")
```



geom_hline: insere uma linha horizontal;
geom_vline: insere uma linha vertical.

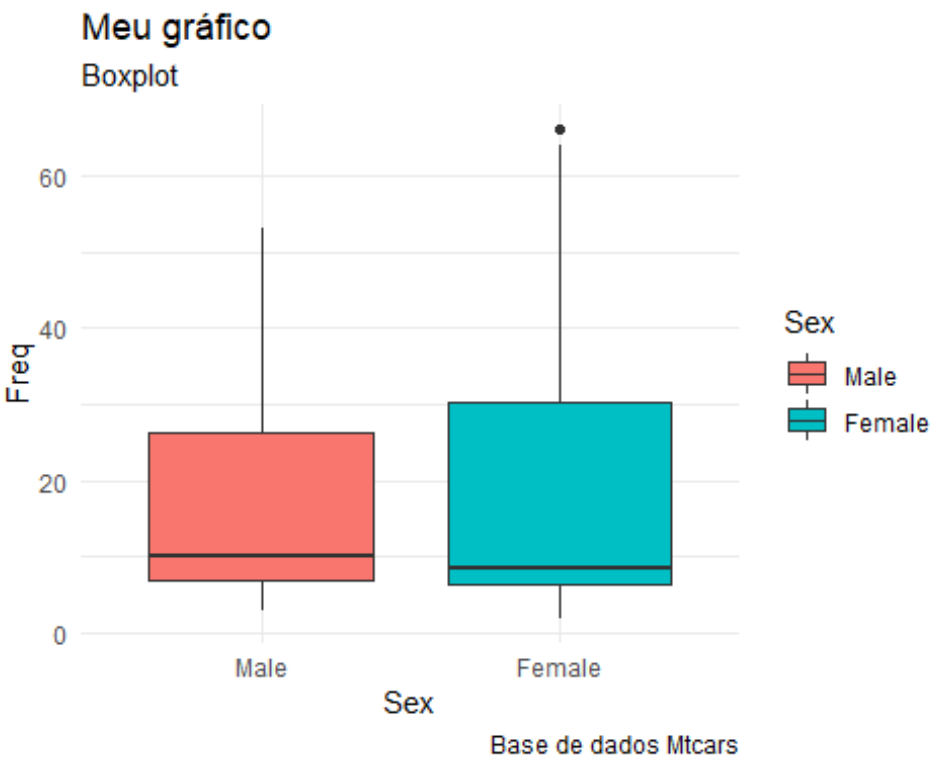
Gráfico de Linhas

```
ggplot(mtcars, aes(mpg, drat)) +
  geom_line() +
  theme_classic() +
  labs(
    title = "Meu gráfico",
    subtitle = "Gráfico de Linha",
    x = "MPG",
    y = "Drat",
    caption = "Base de dados Mtcars")
```



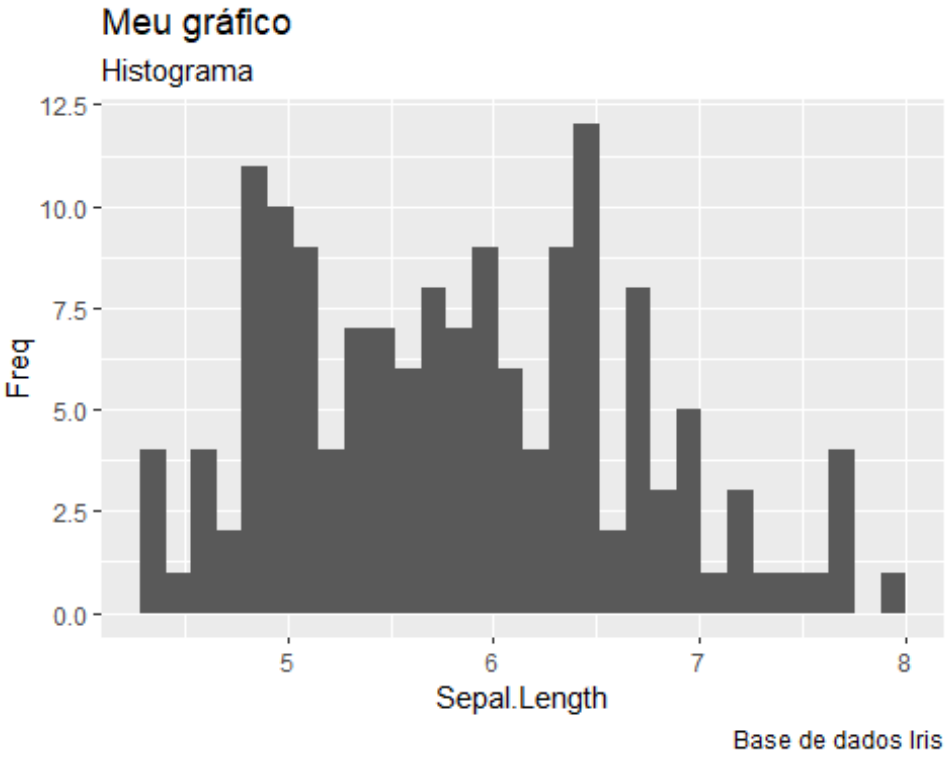
Boxplot

```
dfhair <- data.frame(HairEyeColor)

ggplot(dfhair, aes(x = as.factor(Sex), y = Freq,
                   fill = Sex)) +
  theme_minimal() +
  geom_boxplot()+
  labs(
    title = "Meu gráfico",
    subtitle = "Boxplot",
    x = "Sex",
    y = "Freq",
    caption = "Base de dados Mtcars")
```



Histograma

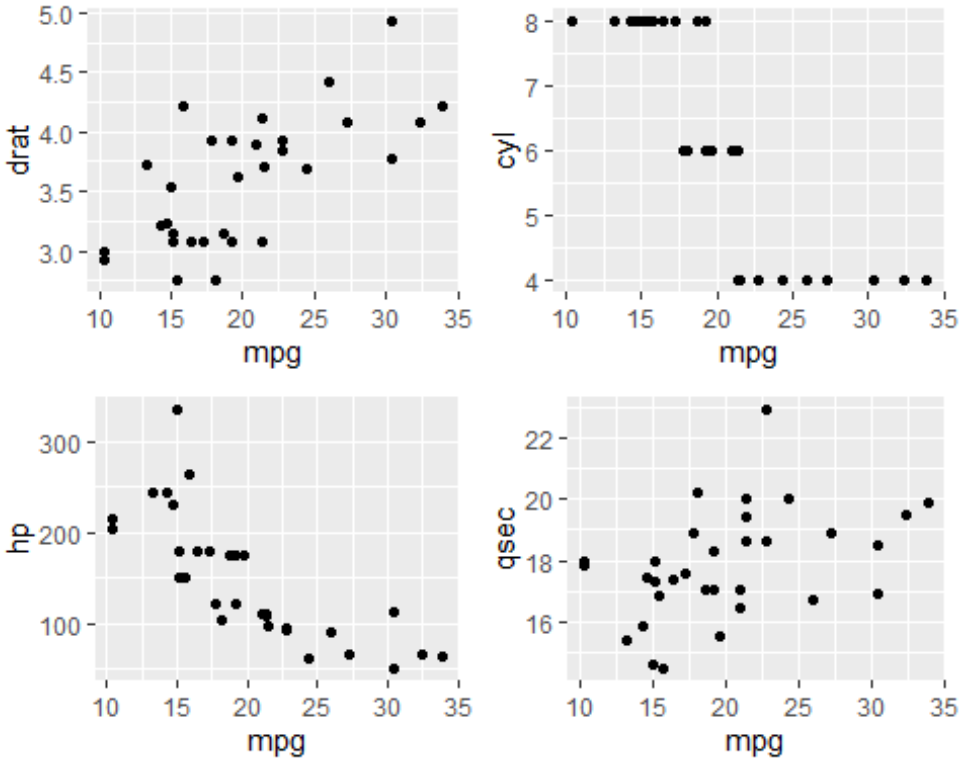
```
ggplot(iris, aes(x = Sepal.Length)) +
  geom_histogram()+
  labs(
    title = "Meu gráfico",
    subtitle = "Histograma",
    x = "Sepal.Length",
    y = "Freq",
    caption = "Base de dados Iris")
```



Plotando vários gráficos na mesma imagem

Para plotarmos vários gráficos em uma mesma imagem, iremos usar o pacote "gridExtra". Neste caso, salvaremos cada gráfico como um objeto.

```
#Instalando e carregando o pacote

install.packages("gridExtra")

library(gridExtra)

#Criando os objetos com os gráficos

grafico1 <- ggplot(mtcars, aes(mpg, drat)) +
  geom_point()

grafico2 <- ggplot(mtcars, aes(mpg, cyl)) +
  geom_point()

grafico3 <- ggplot(mtcars, aes(mpg, hp)) +
  geom_point()

grafico4 <- ggplot(mtcars, aes(mpg, qsec)) +
  geom_point()

#Plotando os gráficos
#ncol = número de colunas
#nrow = número de linhas

grid.arrange(grafico1,
             grafico2,
             grafico3,
             grafico4,
             ncol=2, nrow=2)
```



Tópicos para EAD0752 - Técnicas Estatísticas de Discriminação

Teste k-S

Os argumentos da função ks.test são:

x: um vetor numérico de valores de dados.
y: um vetor numérico de valores de dados ou uma cadeia de caracteres nomeando uma função de distribuição cumulativa ou uma função de distribuição cumulativa real, como “pnorm”.
exact: NULL ou uma lógica indicando se um valor p exato deve ser calculado.
… : Parâmetros da distribuição especificada (como uma cadeia de caracteres) por y.
alternative: Indica a hipótese alternativa e deve ser uma das “two.sided” (padrão), “less” ou “greater”. Você pode especificar apenas a letra inicial do valor, mas o nome do argumento deve ser dado integralmente.

```
ks.test(x, y, ...,alternative = c("two.sided", "less",
"greater"),exact =NULL)
```

Assimetria e curtose

Utilizaremos o pacote fBasics que fornece uma coleção de funções para explorar e investigar propriedades básicas de retornos financeiros e quantidades relacionadas. Os campos cobertos incluem técnicas de análise exploratória de dados e a investigação de propriedades distributivas, incluindo

estimação de parâmetros e testes de hipóteses. Ainda mais, existem várias funções de utilidade para manuseio e gerenciamento de dados.

```
install.packages("fBasics") #Instala o pacote

library(fBasics) #Carrega o Pacote

basicStats(seu_dado) #Essa função traz um "summary" mais completo que já calcula assimetria, curtose, variância etc

skewness(seu_dado) #assimetria

kurtosis(seu_dado) #curtose
```

Regressão Logística

Definição

- Técnica de análise multivariada utilizada para calcular a probabilidade de ocorrência de um evento de interesse;

- Estuda o efeito das variáveis explicativas sobre uma variável dependente binária, ou seja, identifica os efeitos das VI sobre a probabilidade de ocorrência do evento de interesse da VD (fraude, adimplência, intenção de compra, sinistro etc.);

- Variável dependente (VD): binária (0= ausência da característica de interesse; 1= presença da característica de interesse);

- Variáveis independentes ou explicativas (VI): métricas ou não-métricas (transformadas em variáveis dummy).

Regressão Logística X Regressão Linear

Situações de uso

- Credit Scoring: distinção entre bons e maus clientes com base em relações anteriores com o banco e dados cadastrais;

- Atuária: probabilidade de ocorrência de sinistro dadas características demográficas e de estilo de vida;

- Marketing: identificar a probabilidade de fidelidade do cliente dado um nível de satisfação e qualidade percebida.

Regressão Logística no R

A regressão logística é dada pela função *glm*, que compreende os seguintes parâmetros:

```
glm(formula, family = gaussian, data, weights, subset,
    na.action, start = NULL, etastart, mustart, offset,
    control = list(...), model = TRUE, method = "glm.fit",
    x = FALSE, y = TRUE, singular.ok = TRUE, contrasts = NULL, ...)
```

A forma mais simples de implementá-la é:

```
glm(Variavel de interesse~Variavel preditora, data = seu dataframe,
family = binomial())
```

No argumento *family* podemos usar os seguintes métodos:

- binomial(link = “logit”)
- gaussian(link = “identity”)
- Gamma(link = “inverse”)
- inverse.gaussian(link = “1/mu^2”)
- poisson(link = “log”)
- quasi(link = “identity”, variance = “constant”)
- quasibinomial(link = “logit”)
- quasipoisson(link = “log”)

Exemplo prático:

Temos um script que compreende a etapa de carregamento de dados, análise e tratamento dos dados, separação da base em treino e teste, o modelo e os resultados.

Nesta análise, vamos utilizar uma base do pacote “caret”, a “GermanCredit”, e tentar classificar os clientes como bons ou maus pagadores.

```
library(caret) #carregando o pacote "caret"

#  CARREGANDO OS DADOS

data("GermanCredit")

dados <- GermanCredit

which(is.na(dados)) #Verificando se há NA

## integer(0)

#  SEPARAÇÃO BASE TREINO / TESTE

vn <- sample(nrow(dados)) #embaralhando os dados

treino <- dados[vn[1:round(nrow(dados)*0.8)] ,] #Separando 80% da base
para treino do modelo

teste  <- dados[vn[ (nrow(treino)+1):(nrow(dados))] ,] #Separando o
restante da base para teste do modelo

#  MODELO

modelo <- glm(Class~Duration+Age, data=treino, family = binomial())
#Regressão logística

p <- predict(modelo, newdata=teste) #Prevendo o modelo no conjunto de
teste

prev <- ifelse(p<.5,0,1) #Se a probabilidade é menor que 50%
classificamos como 0 (não sucesso)
#Se maior que 50%, classificamos como 1 (sucesso)

#  Resultados

table(prev, teste$Class) # Matriz de Confusão

##
## prev Bad Good
##    0  21   24
##    1  37  118
```

Sobre o nosso modelo

- Pra essa base de dados vimos que não há NA, caso houvesse, teríamos que tratá-los de alguma forma.

- Separamos 80% da base para treinar o modelo e os 20% restantes para testá-lo.

- Nossa tentativa foi prever a variável “Class”, ou seja, a classificação do cliente em bom(Good) ou mau(Bad) pagador.

- Utilizamos as variáveis “Duration”(número de parcelas) e “Age”(idade) para fazer a classificação.

- Em *p <- predict(modelo, newdata=teste)*, fizemos a previsão de teste com o modelo treinado na base com 80% das observações.

- Em *prev <- ifelse(p<.5,0,1)* nós determinamos que, se a probabilidade é menor do que 50%, o cliente é classificado como 0(mau pagador), do contrário, ele é classificado com 1(bom pagador).

- A matriz de confusão nos traz os resultados. Com ela podemos ver que, 21 maus pagadores e 118 bons pagadores foram classificados corretamente. Temos uma acurácia de 69,5%, ou seja, conseguimos prever corretamente 69,5% das observações.

Essa implementação de regressão logística é uma aplicação bem simples do modelo. Um ponto importante seria uma análise exploratória mais aprofundada. Poderíamos ter tratado melhor os dados, procurado outras variáveis preditoras com maior correlação com a variável de interesse, entre outras coisas.

Quando vemos um resumo da nossa variável de interesse, vemos que há um número maior de bons pagadores do que o de maus pagadores. É uma base desbalanceada e isso, possivelmente, fez com o nosso modelo conseguisse classificar melhor os bons pagadores e tivesse uma taxa de acerto muito baixa para os maus pagadores.
Balancear a base de treino seria uma das soluções para melhoria da nossa regressão. Poderíamos, na base de treino, ter selecionado a mesma porcentagem de observações de bons e maus pagadores, ou seja, 50% bons e 50% maus. Esse balanceamento faz com que o nosso modelo aprenda melhor sobre os dois casos e consiga classificar corretamente as duas situações.

```
summary(dados$Class)

##  Bad Good
##  300  700
```

KNN

Definição

- O algoritmo k-NN (k - Nearest Neighbor) é um dos mais simples que existem e é muito utilizado em Machine Learning. Ele é chamado de ‘Algoritmo Preguiçoso’.

- Ele é usado para classificar objetos através da distância entre eles, um grupo é definido a partir dos vizinhos mais próximos.

- O kNN pode ser aplicado em diversos segmentos de negócio, problemas relacionados a finanças, saúde, ciência política, reconhecimento de imagem, reconhecimento de vídeos, etc.

- Ele funciona como classificador e como regressor.

Procedimentos

- Todos os dados do dataset serão utilizados na fase de teste, o que gera um treino muito rápido e em um teste e validação lentos.

- É necessário definir uma métrica para calcular a distância entre os objetos da nossa base de treino;

- É necessário definir o valor de k (número de vizinhos mais próximos).

Como o algoritmo funciona

1. Calcula a distância entre os pontos/objetos;

Para calcularmos as distâncias entre os pontos podemos utilizar as seguintes medidas:

Distância Euclidiana (mais simples)
Distância de Hamming
Distância Manhattan
Distância de Markowski
Distância de Mahalanobis
De acordo com o seu dataset você pode usar a que julgar mais adequada.

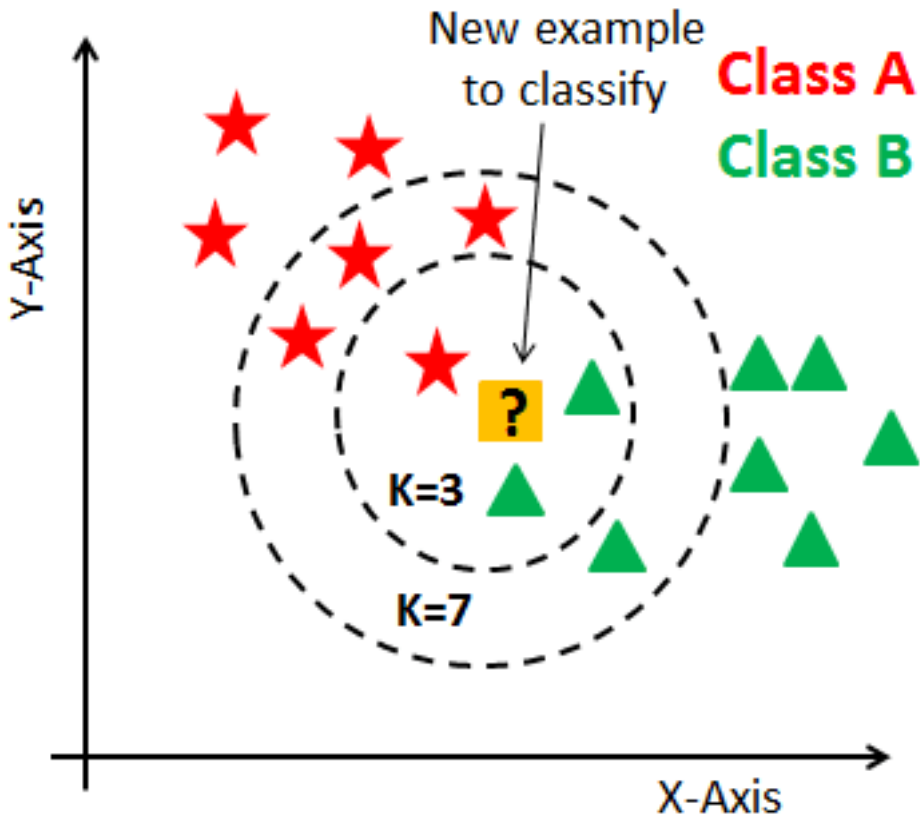
2. Encontra os objetos/pontos/vizinhos mais próximos;

3. Faz a predição do(s) ponto(s) desejado(s).

Na prática…

O algoritmo usa votação majoritária para determinar a variável que queremos predizer, ou seja, o maior número de pontos vizinhos considerando o ‘k’ determinado é o que vai dizer se o ponto é ‘quadrado’ ou ‘estrela’, ‘classe a’ ou ‘classe b’.

Para k = 3, o nosso algoritmo diria que o ponto é um triângulo. Para k = 7, o nosso ponto a ser predito é uma estrela.



Importante!

Se k for muito pequeno, o modelo fica sensível a ruídos;
Se k for muito grande, o modelo pode incluir pontos de outras classes, não fica muito preciso;
O ideal é escolher um valor ímpar para k, assim evitamos empate na votação;
Os dados precisam ser normalizados, dessa forma evitamos que um atributo domine as medidas:

Exemplo:
A altura de uma pessoa pode variar de 1,20m a 2,10m enquanto o peso pode variar de 40kg a 200kg.

KNN no R

A forma mais simples de implementar o KNN é:

```
knn(train, test, cl, k = 1)
```

Onde:

- ‘train’ é o seu conjunto de dados de treino;
- ‘test’ é o seu conjunto de dados de teste;
- ‘cl’ é a variável que você quer classificar;
- ‘k’ é o número de k vizinhos a serem considerados.

Exemplo prático:

Neste exemplo, vamos utilizar a mesma base de dados que utilizamos para o modelo de regressão logística.

```
library(class)


data("GermanCredit")

dados <- GermanCredit

which(is.na(dados)) #Verificando se há NA

## integer(0)

#  SEPARAÇÃO BASE TREINO / TESTE

#Vamos selecionar as mesma variáveis preditoras do exemplo anterior
dados <- dados %>% select(duracao=Duration, idade=Age, Class = Class)

head(dados) #6 primeras linhas do data frame

##   duracao idade Class
## 1       6    67  Good
## 2      48    22   Bad
## 3      12    49  Good
## 4      42    45  Good
## 5      24    53   Bad
## 6      36    35  Good

vn <- sample(nrow(dados)) #embaralhando os dados

treino <- dados[vn[1:round(nrow(dados)*0.8)] ,] #Separando 80% da base
para treino do modelo

teste  <- dados[vn[ (nrow(treino)+1):(nrow(dados))] ,] #Separando o
restante da base para teste do modelo


# Modelo KNN
modelok <- knn(treino[,-3], teste[,-3], cl= treino$Class,  k =3)

#Note que removemos a coluna "Class" do conjunto de treino e teste


#  Resultados

table(modelok, teste$Class) # Matriz de Confusão

##
## modelok Bad Good
##    Bad   12   30
##    Good  44  114
```

```
mean(teste[,3] != modelok) #Taxa de erro
```

```
## [1] 0.37
```

```
mean(teste[,3] == modelok) #Taxa de acerto
```

```
## [1] 0.63
```

Sobre o nosso modelo

O nosso modelo KNN teve uma performance um pouco inferior ao modelo de regressão.
Note que, em *k <- knn(treino[,-3], teste[,-3], cl= treino$Class, k =3)*, removemos a variável de interesse no treino e teste. Esse procedimento é necessário, pois não podemos usar a variável de interesse para predizer ela mesma.
As mesmas considerações feitas para o modelo de regressão são válidas também para este modelo KNN. Além disso, poderíamos ter normalizado e colocado as nossas variáveis preditoras na mesma escala.
A escolha do nosso k foi aleatória, mas há algumas técnicas para escolher o número ideal de k-vizinhos. Vale a pena pesquisar sobre isso.

Exemplo de função para normalização dos dados

```
#Normalização
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x))) }

dataset[,colunas] <- as.data.frame(lapply(dataset[,colunas],
normalize))
```

Referências e Links úteis

Funções: Definição, argumentos e operadores binários, Análise Real. Disponível em: https://analisereal.com/2015/04/04/funcoes-definicao-argumentos-e-operadores-binarios/ Acesso em: 20 nov. 2019.

GARRETT, Garrett; HADLEY, Wickham. R for Data Science: Visualize. Model. Transform. Tidy and Import Data.. [S. l.]: O'REILLY, 2017. E-book. Disponível em: https://r4ds.had.co.nz/index.html

HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second Edition. ed. [S. l.]: Springer Science & Business Media, 2013. 745 p. E-book.

INTRODUÇÃO ao ggplot. Blog Dados Aleatórios, 26 set. 2018. Disponível em: https://www.dadosaleatorios.com.br/post/introducao-ao-ggplot/. Acesso em: 13 fev. 2019.

MANIPULAÇÃO de dados. Curso R, 2018. Disponível em: http://material.curso-r.com/manip/. Acesso em: 11 abr. 2019.

OLIVEIRA, Paulo Felipe de; GUERRA, Saulo; MCDONNEL, Robert. Ciência de Dados com R – Introdução. Brasília: Editora IBPAD, 2018.

R e RStudio, Análise Real. Disponível em: https://analisereal.com/2015/01/20/r-e-rstudio-2/ Acesso em: 10 jan. 2019.

https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html

https://dplyr.tidyverse.org/

Valores ausentes em R. [S. l.], 2018. Disponível em: http://rstudio-pubs-static.s3.amazonaws.com/261838_71b13475011340ab94e9c51d8e462080.html. Acesso em: 20 mar. 2019.