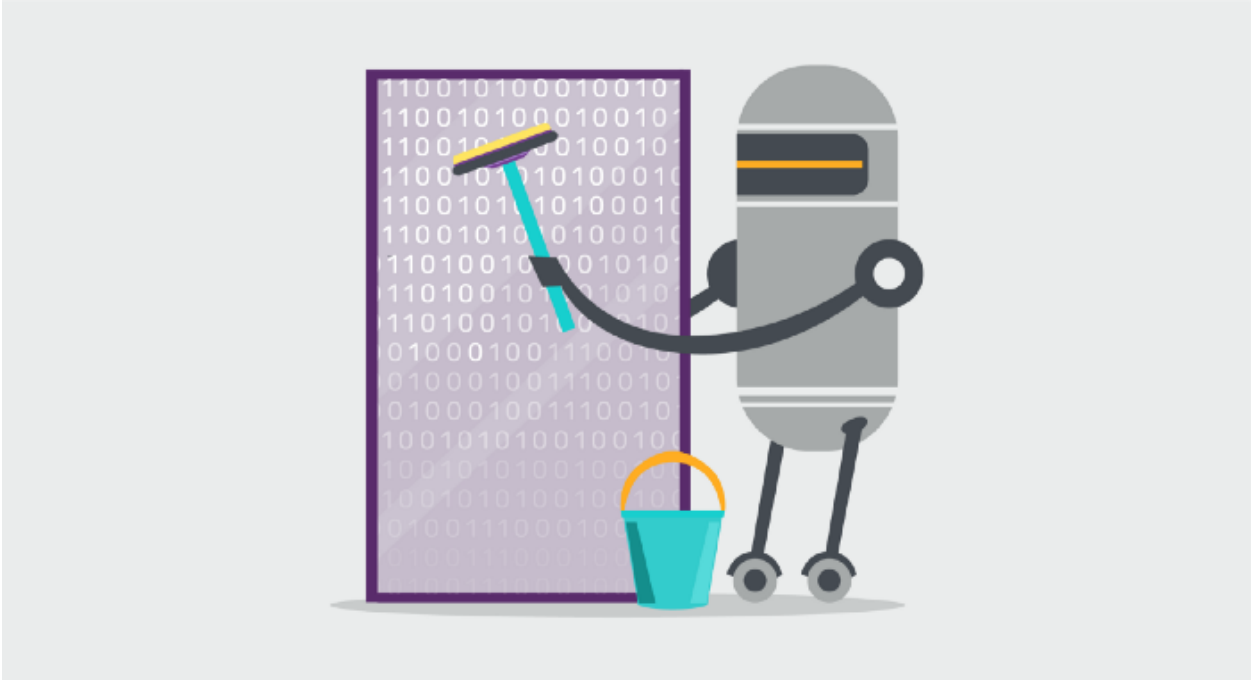
Limpeza e pré-processamento dos dados



Como sabemos, os dados podem vir de várias fontes e em vários formatos diferentes. Devido a erros humanos, imprecisão dos aparelhos de medição ou defeitos no processo de coleta dos dados, na maioria dos casos, eles não vão estar disponíveis de maneira conveniente para a análise ou para a criação de modelos de machine learning, sendo necessário fazer ajustes prévios no conjunto de dados.

Naturalmente, o número de erros que podem estar presentes num banco de dados é infinito. Nesse material, iremos estudar alguns dos mais comuns e a solução para eles.

Limpeza de dados

Nesa primeira etapa, estamos interessados em corrigir ou remover dados incorretos ou formatadas de maneira inconsistente com a linguagem R.

Problema 1 : Dados no formato errado

1.1 : Valores numéricos como caracteres

Vamos importar um conjunto de dados sobre o consumo de cerveja numa região de São Paulo. Esse arquivo está disponível para download no Kaggle.

Esse banco de dados contém dados de precipitação, temperatura e consumo de cerveja durante o ano de 2015.

```
library(readr)
dados_cerveja <- read_csv("Consumo_cerveja.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Data = col_date(format = ""),
##   `Temperatura Media (C)` = col_number(),
##   `Temperatura Minima (C)` = col_number(),
##   `Temperatura Maxima (C)` = col_number(),
##   `Precipitacao (mm)` = col_character(),
##   `Final de Semana` = col_double(),
##   `Consumo de cerveja (litros)` = col_double()
## )
```

Como podemos ver no código retornado ao importar o banco, a coluna "precipitação" foi considerada pelo R como uma coluna de caracteres, não uma coluna numérica, como era de se esperar.

Isso ocorre porque no Brasil, as casas decimais são separadas por vírgulas, enquanto na maioria dos outros países, se utiliza o ponto (.). Logo, ao deparar-se com uma vírgula, o R interpreta toda a coluna como um texto.

Essa formatação errada dos dados causa erros. Por exemplo: caso queirammos calcular a média da precipitação, vamos obter o seguinte erro:

```
mean(dados_cerveja$`Precipitacao (mm)`, na.rm = TRUE)
```

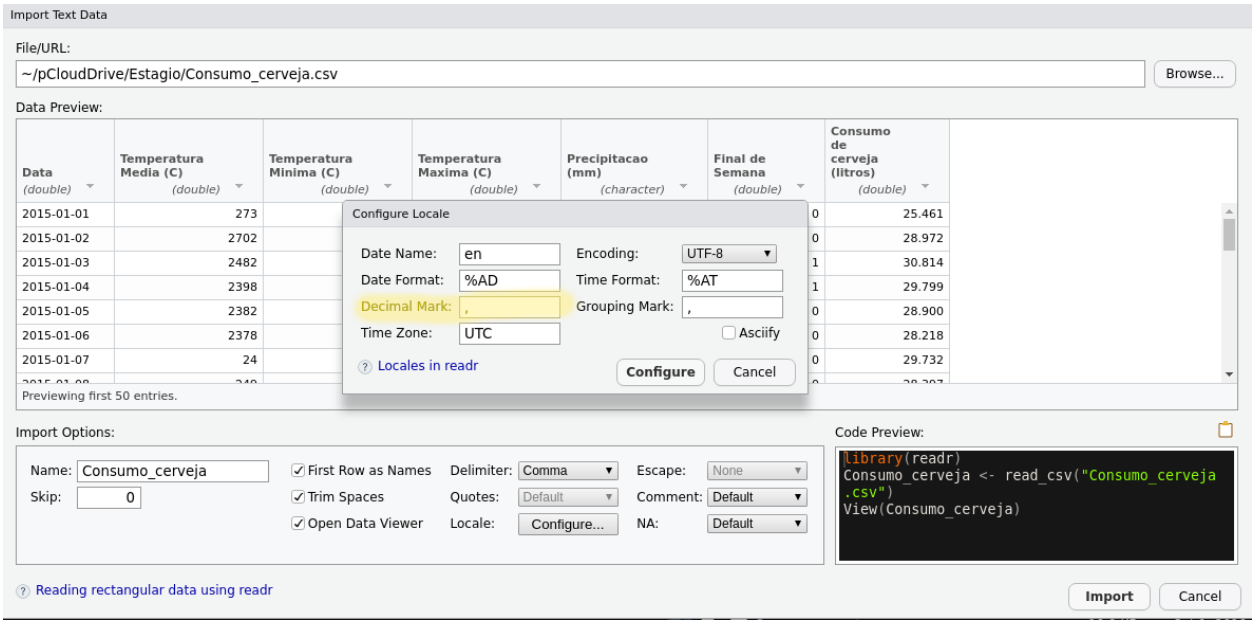
```
## Warning in mean.default(dados_cerveja$`Precipitacao (mm)`, na.rm = TRUE):
## argument is not numeric or logical: returning NA
```

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

Isso porque função "mean" só aceita dados numéricos ou lógicos, o tipo "character" é, então, rejeitado.

Podemos resolver esse problema de duas maneiras. A primeira maneira é configurar os parâmetros da função de importação para considerar a vírgula como indicador de decimais. Isso pode ser feito através da interface gráfica do RStudio.

Supondo que o conjunto de dados está em formato csv na memória local do seu computador : vá em "Import Dataset" > "From Text (readr)", em sequida, selecione o arquivo desejado e clique em "Configure" (ao lado da opção "locale"). Mude a marca decimal conforme a imagem abaixo:




```
dados_cerveja_2 <- read_csv("Consumo_cerveja.csv",
    locale = locale(decimal_mark = ","))
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Data = col_date(format = ""),
##   `Temperatura Media (C)` = col_double(),
##   `Temperatura Minima (C)` = col_double(),
##   `Temperatura Maxima (C)` = col_double(),
##   `Precipitacao (mm)` = col_double(),
##   `Final de Semana` = col_double(),
##   `Consumo de cerveja (litros)` = col_number()
## )
```

Como vemos, a coluna "precipitação" está agora sendo interpretada como um "double" (uma variável numérica pertencente aos números reais).

Ao tentar calcular a média da precipitação, agora obtemos uma resposta correta:

```
mean(dados_cerveja_2$`Precipitacao (mm)`,na.rm=TRUE)
```

```
## [1] 5.196712
```

A segunda maneira de resolver esse problema é alterar manualmente o banco de dados.

```
dados_cerveja$`Precipitacao (mm)` <- sub(',','.',dados_cerveja$`Precipitacao (mm)`)

dados_cerveja$`Precipitacao (mm)` <- as.numeric(dados_cerveja$`Precipitacao (mm)`)
```

No primeiro comando, utilizamos a função **sub** para substituir todas as vírgulas da variável "precipitação" por pontos.

No segundo comando, utilizamos a função **as.numeric** para coagir a variável a ser interpretada como uma variável numérica. Existem várias funções de coação : **as.character**, **as.data.frame**, **as.factor**, **as.matrix**, entre muitas outras.

1.2 Fatores como caracteres

Outro erro de formatação comum é que variáveis categóricas (fatores, ou *factors*, como chamadas pelo R) sejam consideradas como caracteres.

Vamos importar um conjunto de dados com informações de alguns passageiros do Titanic. Esse conjunto de dados está disponível no Kaggle.

```
titanic <- read_csv("train.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   PassengerId = col_double(),
##   Survived = col_double(),
##   Pclass = col_double(),
##   Name = col_character(),
##   Sex = col_character(),
##   Age = col_double(),
##   SibSp = col_double(),
##   Parch = col_double(),
##   Ticket = col_character(),
##   Fare = col_double(),
##   Cabin = col_character(),
##   Embarked = col_character()
## )
```

Vemos que as colunas "Sex" e "Embarked" foram interpretadas como caracteres, mas deveriam ser interpretadas como variáveis categóricas.

A variável sexo é autoexplicativa. A variável "Embarked" toma três valores : "C", caso o passageiro tenha embarcado no porto de Cherbourg, no noroeste da França, "Q", caso tenha embarcado em Queenstown (hoje chamada Cobh), na Irlanda ou "S", caso tenha embarcado em Southampton, na Inglaterra.

Os valores que um fator pode assumir são chamados pelo R de *levels* (níveis).

Vamos coagir a variável a ser interpretada como um fator utilizando a função **as.factor**, mencionada anteriormente:

```
titanic$Sex <- as.factor(titanic$Sex)
titanic$Embarked <- as.factor(titanic$Embarked)
```

Podemos checar os níveis dos fatores com a função **levels**

```
levels(titanic$Sex)
```

```
## [1] "female" "male"
```

```
levels(titanic$Embarked)
```

```
## [1] "C" "Q" "S"
```

Agora, essas variáveis estão sendo interpretadas de maneira correta.

Problema 2: variáveis com nomes inconvenientes

Quando se programa em qualquer linguagem, é considerada uma boa prática que os nomes das variáveis sejam curtos, sem caracteres especiais ou espaços e em inglês (caso os dados venham a ser diponibilizados para um público internacional). No entanto, essas boas práticas muitas vezes não são seguidas. Quando vamos analisar um conjunto de dados criados por terceiros, pode ser útil trocar os nomes das variáveis por outros mais adequados.

Vamos usar como exemplo o banco de dados de consumo de cerveja citado acima. Podemos verificar os nomes das variáveis do conjunto de dados com a função **names**.

```
names(dados_cerveja)
```

```
## [1] "Data"                          "Temperatura Media (C)"
## [3] "Temperatura Minima (C)"        "Temperatura Maxima (C)"
## [5] "Precipitacao (mm)"             "Final de Semana"
## [7] "Consumo de cerveja (litros)"
```

Como vemos, os nomes das variáveis são longos, possuem espaços e estão em Português. Vamos mudar os nomes das variáveis para outros mais curtos e em inglês.

```
names(dados_cerveja) <- c('date','med_temp','min_temp','max_temp',
                          'precip','weekend','beer_cons')
```

Utilizamos a função **c** para criar um vetor de caracteres com novos nomes e atribuímos esse vetor aos nomes das variáveis do conjunto de dados.

PS: É importante que os nomes do vetor criado estejam na ordem equivalente aos nomes anteriores.

Essa solução, apesar de prática, pode ser inviável caso se deseje renomear poucas variáveis de um conjunto maior.

Outra solução pode ser utilizar a função **rename**, do pacote dplyr.

Vamos supor que queremos mudar apenas o nome da variável de “date” para “day”.

```
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
```

```
dados_cerveja <- dados_cerveja %>%
  rename(
    day = date
  )
```

Novamente, podemos verificar a mudança de nome da variável com a função names:

```
names(dados_cerveja)
```

```
## [1] "day"       "med_temp"  "min_temp"  "max_temp"  "precip"    "weekend"
## [7] "beer_cons"
```

Problema 3 : Valores duplicados

Alguns bancos de dados podem conter valores duplicados. Isso é prejudicial à análise dos dados porque algumas observações serão tidas como mais comuns do que realmente são. Por isso, é necessário remover as duplicatas.

No entanto, é preciso haver cuidado neste passo. Alguns bancos de dados podem conter falsas duplicatas. Estas são observações que, apesar de iguais, não são erros do banco de dados. Por exemplo, um conjunto de dados pode registrar um mesmo exame feito pelo mesmo paciente em dois momentos diferentes, com o mesmo resultado. Nesse caso, não se trata de um erro, mas uma mesma observação em dois momentos diferentes.

Por outro lado, podem haver casos em que uma duplicata verdadeira não aparece com valores exatamente iguais no banco de dados Por exemplo, caso um crime ocorra numa área de divisa entre duas cidades, a mesma ocorrência pode ser registrada por órgãos dos dois municípios. Ao se analisar os dados de crime da região, a ocorrência estará duplicada, mas com a variável "cidade" diferente. Esse tipo de erro é muito comum quando o banco de dados utilizado é uma junção de dois ou mais bancos.

Vamos demonstrar como remover observações duplicadas com o conjunto de dados airAccs, que contém dados de acidentes aéreos desde 1908. Esse conjunto de dados é muito grande, logo, para facilitar a demonstração, vamos utlizar apenas os acidentes que ocorreram até 1930.

```
library(dplyr)

airAccs <- read_csv("datasets/airAccs.csv",col_types = cols(X1 = col_skip(), Date = col_date(format = "
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
airAccs <- airAccs %>%
  filter(Date < as.Date('1930-12-31'))
```

Primeiramente, vamos verificar se há observações duplicadas exatas (nos quais todas as colunas têm o mesmo valor). Fazemos isso com a função *duplicated*, disponível na base do R. Essa função retorna TRUE caso a linha em questão seja duplicada e FALSE, caso não seja.

```
sum(duplicated(airAccs))
```

```
## [1] 0
```

a função **sum** considera valores TRUE como 1 e FALSE como 0. Como a soma de valores duplicados é 0, podemos concluir que não há valores duplicados exatos no conjunto de dados.

Caso o conjunto de dados possua valores duplicados exatos, é possível removê-los com a função **distinct**, do pacote dplyr.

Vamos verificar se há valores duplicados não-exatos. Vamos filtrar apenas os dias nos quais houveram mais de um acidente e verificar se os acidentes foram da mesma linha aérea, com o mesmo modelo de avião e ocorreram em locais próximos. Caso sim, iremos deletar uma das observações.

```
datas_dup <- duplicated(pull(airAccs,Date)) | duplicated(pull(airAccs,Date),fromLast = TRUE)

acidentes_data_dup <- airAccs[datas_dup,c('Date','location','operator','planeType')]

knitr::kable(acidentes_data_dup)
```

Date	location	operator	planeType
1920-08-16	College Park, Maryland	US Aerial Mail Service	De Havilland DH-4
1920-08-16	Bedford, England	By Air	Armstrong-Whitworth F-K-8
1927-04-22	Floh, Germany	Deutsche Lufthansa	Fokker FG III
1927-04-22	Goshen, Indiana	US Aerial Mail Service	Douglas M-4
1927-09-17	Hadley, New Jersey	Reynolds Airways	Fokker F-VII
1927-09-17	Near Dunellen, New Jersey	Reynolds Airways	Fokker F-V!!b-3m
1927-09-23	Schleiz, Germany	Deutsche Lufthansa	Dornier Merkur
1927-09-23	Near schleiz, Thuringia„ Germany	Deutsche Lufthansa	Dornier Merkur D-585
1927-11-16	Strasburg, France	CIDNA	Farman F-121 Jabiru
1927-11-16	Over the Gulf of Finland	Aero O-Y	Junkers F-13
1928-01-22	France	Aeropostale	Breguet 14
1928-01-22	Tarragona, Spain	Aeropostale	Breguet 14
1928-09-04	Pocatello, Idaho	National Parks Airways	Fokker Super Universal
1928-09-04	Adelaide Hills, Australia	Qantas	de Havilland DH.50J
1928-11-25	Edgerton, Ohio	Continental Air Lines	Travel Air 4000
1928-11-25	Bristolville, Ohio	National Air Transport	Douglas M-4
1929-01-31	Morgantown, West Virginia	Skyline Transportation Company	Travel Air 4000
1929-01-31	Off Morocco	Aeropostale	Latecoere 26
1929-06-29	Columbus, Ohio	Continental Air Lines	Travel Air 4000
1929-06-29	Near Lindau, Bavaria, Germany	Bodensee Aerolloyd	Domier Delphin III (flying boat)
1929-08-24	Sochi, Russia	Ukvozduchput	Kalinin K-4
1929-08-24	Elm, Germany	Deutsche Lufthansa	Fokker FG II
1929-09-17	Off Larache, Morocco	Aeropostale	Latecoere 25-3-R

Date	location	operator	planeType
1929-09-17	Mt Lamentation, Connecticut	Colonial Air Transport	Pitcairn PA-6 Mailwing
1929-09-17	Jacumba, California	Pickwick Airways	Fairchild 71
1930-10-30	Near Neufchatel, France	Imperial Airways	Handley Page W-8
1930-10-30	Brookston, Indiana	Embry Riddle Company	Pitcairn PA-6 Mailwing

Para entender esse bloco de código é importante entender como funciona a função duplicated.

duplicated percorre uma determinada tabela na ordem em que está organizada e identifica quais linhas são repetições de linhas anteriores. A primeira linha do conjunto de duplicatas não é incluída. Para ver todas as linhas duplicadas, utilizamos a função **duplicated** duas vezes, sendo a segunda vez na ordem reversa.

Usamos a função **pull**, equivalente ao operador "$", que extrai um vetor de um data frame. Definimos o parâmetro fromLast = TRUE para extrair o vetor na ordem reversa.

Finalmente, interpretamos a tabela para verificar se há valores duplicados não-exatos:

Os dois acidentes registrados em 16/08/1920 ocorreram em países diferentes e distantes, logo não se trata de uma duplicata.

Já os dois registros em 17/09/1927 ocorreram com uma aeronave da mesma companhia, ambos no estado de Nova Jérsei. Ao observar um mapa, vemos que Hadley e Dunellen são vizinhas. É extremamente improvável que dois acidentes da mesma comapnhia aérea tenham ocorrido no mesmo local no mesmo dia. Logo, consideramos que se trata de uma duplicata e removemos uma das observações.

Ao analisar os outros dados vemos que os seguintes acidentes também estão duplicados:

- 23/09/1927 - Alemanha
- 22/01/1928 - França/Espanha
- 25/11/1928 - Ohio

Identificamos quais são as linhas com os valores em questão com a função **which**:

```
which(airAccs$Date==as.Date('1927-09-17'))
```

```
## [1] 126 127
```

```
which(airAccs$Date==as.Date('1927-09-23'))
```

```
## [1] 128 129
```

```
which(airAccs$Date==as.Date('1928-01-22'))
```

```
## [1] 140 141
```

```
which(airAccs$Date==as.Date('1928-11-25'))
```

```
## [1] 166 167
```

E as removemos do data frame:

```r
airAccs <- airAccs[-c(126,128,140,166),]
```

Pré-processamento dos dados

Dados existem em diferentes formatos : imagens, texto, arquivos de áudio, tabelas, entre outros. No entanto, apesar desses formatos serem compreensíveis para humanos, eles precisam ser processados para que possam ser entendidos por máquinas.

Nessa seção, falaremos dos passos necessários para que possamos aplicar um algoritmo de machine learning a um conjunto de dados tabulares. Como na seção anterior, lidaremos com os problemas mais comuns de pré-processamento em Data Science, mas nem todos os ajustes abaixo precisam ser aplicados a todos os desafios e alguns deles precisarão de ajustes mais específicos, que não trataremos abaixo (como o processamento de áudio ou imagens).

Ajuste 1 : Lidar com valores ausentes

É muito comum a presença de dados ausentes em conjuntos de dados. Isso pode acontecer porque a informação em questão não estava disponível no momento da coleta (por exemplo, em uma pesquisa de campo, um entrevistado pode se recusar a responder uma porgunta) ou por alguma regra de validação dos dados (por exemplo, num conjunto de dados de casas, a variável "qualidade da garagem" estará ausente caso a casa em questão não tenha garagem).

De qualquer maneira, os algoritmos de machine learning não aceitam dados com valores ausentes. Assim, um desses dois passo deve ser tomado:

1. Eliminar observações (linhas) com valores ausentes
2. Eliminar variáveis (colunas) com valores ausentes
3. Preencher valores ausentes com uma estimativa

Tanto a solução 1 como a 2 envolvem perda de dados. Isso pode não ser um grande problema caso hajam poucas observações com valores ausentes ou caso uma variável seja composta majoritariamente de valores ausentes.

Existem várias maneiras de preencher o valor de uma variável ausente. Aqui, vamos lidar com as alternativas mais comuns: preenchê-los com a média, mediana ou moda da variável em questão.

Observação

No R, assim como na maioria das linguagens, os valore ausentes são representados por NA (*not avaliable*, não disponível). Em alguns bancos de dados, os valores ausentes podem estar representados de maneira diferente, por exemplo, com um caractere "não informado" ou "?". Nesses casos, é necessário substituir esses valores por NA.

Por exemplo, no conjunto de dados de acidentes aéreos, mencionado anteirormente, alguns valores da variável "operator" (linha aérea) estão preenchidos com "?". Vamos substituir esses valores por NA com a função **ifelse**.

```r
airAccs$operator <- ifelse(airAccs$operator=='?',NA,airAccs$operator)
```

A função ifelse realiza um teste lógico e retorna um certo valor caso o teste seja TRUE e outro caso o teste seja FALSE.

No nosso caso, testamos se a variável "operator" tem valor "?" (primeiro parâmetro), caso isso seja verdadeiro, a variável será substituída por NA(segundo parâmetro). Caso não seja verdadeiro, a variável deverá manter o valor que já possui (terceiro parâmetro).

Vamos demonstrar o processo de preenchimento de NAs utilizando o conjunto de dados de passageiros do Titanic, importado anteriormente.

Primeiramente, vamos verificar se há dados ausentes no conjunto:

```
colSums(is.na(titanic))
```

```
## PassengerId    Survived      Pclass        Name         Sex         Age
##           0           0           0           0           0         177
##       SibSp       Parch      Ticket        Fare       Cabin    Embarked
##           0           0           0           0         687           2
```

Vamos analisar essa expressão de dentro para fora:

A função **is.na** retorna TRUE caso um valor seja ausente (NA) e FALSE, caso contrário.

A função colSums peforma uma soma em cada coluna de um data frame. Como sabemos, valores TRUE são considerados como 1 em uma soma e valores FALSE são considerados como 0.

Logo, ao usar a combinação dessas duas funções, temos a quantidade de valores NA em cada coluna do data frame.

Como podemos ver, existem três variáveis com valores ausentes, a variável "Age" (idade do passageiro), "Cabin" (cabine na qual se alojava) e "Embarked", o port no qual o passageiro embarcou.

A variável Cabin possui 687 valores ausentes. Como podemos verificar com a função nrows, o conjunto de dados possui 891 linhas. Essa variável e majoritariamente ausente, logo, podemos eliminá-la (ou simplesmente não incluí-la no modelo a ser construído).

```
nrow(titanic)
```

```
## [1] 891
```

Por questões de demonstração, vamos eliminá-la do data frame. Para eliminar uma coluna, simplesmente atribula o valor NULL a ela.

```
titanic$Cabin <- NULL
```

A variável "Age" contém uma quantidade considerável de valores ausentes, mas uma remoção da variável ou das observações com valores ausentes causaria uma grande perda de informação.

Como se trata de uma variável numérica, podemos optar por preenchê-la com a média ou mediana dos valores não-ausentes. Ao fazermos um boxplot da idade, vemos que há alguns outliers. Como a média é mais influenciada por valores extremos, vamos optar pela variância.

```
titanic$Age[is.na(titanic$Age)] <- median(titanic$Age,na.rm = TRUE)
```

Aqui, subselecionamos os valores ausentes combinando o operador [] com a função is.na e os substituímos pela mediana, calculada com a função **median**. A função median, por padrão, sempre retornará NA caso um dos elementos do vetor utilizado como input seja NA. Para que seja calculado somente a mediana dos valores não-ausentes, defina o parâmetro na.rm como TRUE.

A variável "Embarked" possui apenas 2 valores ausentes. Como se trata de uma variável categórica, vamos substuir os valores ausentes pela moda, isto é, o valor mais comum.

Vamos verificar qual é o valor mais comum com a função **table**.

```
table(titanic$Embarked)
```

```
##
##   C   Q   S
## 168  77 644
```

"S" é a moda da variável. Repetimos um processo semelhante ao anterior:

```
titanic$Embarked[is.na(titanic$Embarked)] <- as.factor("S")
```

Aqui, é importante que se utilize a função as.factor, para que o valor não seja considerado um caractere.

Verificamos novamente a quantidade de NAs no nosso conjunto de dados:

```
colSums(is.na(titanic))
```

```
## PassengerId    Survived      Pclass        Name         Sex         Age
##           0           0           0           0           0           0
##       SibSp       Parch      Ticket        Fare    Embarked
##           0           0           0           0           0
```

Vemos que agora não há valores ausentes no data frame.

Ajuste 2 : dividindo os dados entre um conjunto de treino e de teste

Como vamos ver nas aulas posteirores, muitos algoritmos de machine learning tendem a se viciar nos dados de treino e, caso não sejam ajustados, seu poder de previsão para novos dados fica aquém do desejado. Esse problema é conhecido como overfitting.

Para verificar a eficácia de um algoritmo para fazer previsões baseando-se em novos dados, repartimos os bancos de dados em dois : uma parte para treinar o algortimo e outra para testá-lo.

Quando estamos participando de uma competição de data science, muitas vezes, já recebemos o conjunto de dados repartido. No entanto, em aplicações do mundo real, é necessário fazer isso manualmente.

É possível fazer essa operação manualmente ou utilizando pacotes específicos.

Vamos utilizar o pacote caTools e demonstrar seu uso no conjunto de dados "iris", já presente na base do R.

Esse conjunto de dados possui dados da comprimento e largura das pétalas e sépalas de várias flores, assim como a espécie de cada uma. Esse banco de dados é um exemplo clássico para aplicação de algoritmos de classificação que preveem a espécie de uma flor baseando-se nas suas medidas.

```
knitr::kable(head(iris))
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

```
library(caTools)

set.seed(1212)
split <- sample.split(iris$Species,
                      SplitRatio = 8/10)

train_iris <- iris[split,]
test_iris <- iris[!split,]
```

Aqui, utilizamos a função **sample.split** para dividir o conjunto de dados. Essa função gera um vetor de valores lógicos, sendo TRUE os valores que irão para o conjunto de treino e FALSE, os que irão para o conjunto de teste. Salvamos esse vetor lógico na variável "split".

No primeiro parâmetro, indicamos a variável que se deseja prever, no caso, a espécie da flor. Em seguida, especificamos a razão de repartição. Não existe uma regra específica para repartir o conjunto de dados, mas é comum utilizar-se uma repartição de 70%, 75% ou 80% para o conjunto de treino e o restante para o teste.

No nosso caso, escolhemos uma repartição de 80% para o conjunto de treino e 20% para o conjunto de teste.

Em sequigda, criamos um subconjunto de treino e outro de teste utilizando subseleção. No segundo caso, utilizamos o operador "!", que indica negação. Isto é : os valores do vetor "split" que eram TRUE se transformam em FALSE e vice-versa. Desta forma, todas as observações que não foram inclusas no conjunto de treino são inclusas no conjunto de teste.

##Ajuste 3 : Padronização e normalização

Em alguns conjuntos de dados, a escala dos valores de certas variáveis é muito distinta devido a diferenças nas unidades de medida. Por exemplo, a idade de um conjunto de pessoas pode variar de 18 a 70 anos, enquanto sua altura, em centímetros, pode variar de 160 a 182.

Algus algoritmos de machine learning exigem que as variáveis estejam em escalas igualitárias, já outros algoritmos, apesar de funcionarem com dados de escalas discrepantes, têm uma performance melhor quando se utilizam escalas igualitŕias.

Existem duas forma de eliminar as discrepâncias de escala: a padronização e a normalização.

Padronização

A padronização consiste em fazer com que os dados tenham média 0 e variância 1. Dada uma variáxel X:

$$X_{padronizado} = \frac{X - E(X)}{\sigma_x}$$

Na qual σ_x é o desvio padrão da variável X, isto é,a raiz quadrada da variância.

Fica como desafio para o aluno utilizar as propiedades do valor esperado e da variância para demonstar que $X_{padronizado}$ tem valor esperado 0 e variância 1.

Vamos utiliar a função **preProcess**, do pacote caret, para padronizar as variáveis no conjunto de dados do Titanic. Note que a variável "Survived" deve ser interpretada como fator, logo, antes de padronizar os dados, iremos modificá-la.

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
titanic$Survived <- as.factor(titanic$Survived)

titanic_2 <- titanic

titanic.pre.padr <- preProcess(titanic_2,
                          method = c('center','scale'))

titanic <- predict(titanic.pre.padr,titanic_2)
```

Aqui, especificamos o parâmetro "method" para "center" (centrar, isto é, subtrair o valor esperado) e "scale" (escalar, isto é, dividir pelo desvio padrão).

A função preProcess não modifica o banco de dados diretamente. É necessário utilizar a função **predict** para modificá-lo.

Normalização

A normalização consiste em transformar as variáveis numéricas de modo que assumam valores de 0 a 1.

Vamos normalizar as variáveis numéricas do conjunto de dados do Titanic utilizando a função preProcess novamente.

```
titanic_3 <- titanic

titanic.pre.norm <- preProcess(titanic_3,
                               method = 'range')

titanic_3 <- predict(titanic.pre.norm,titanic_3)
```

Repetimos o processo anterior, apenas trocando o parâmetro "method" para "range".

A padronização pode ser útil quando os dados seguem uma distribuição normal, mas não exclusivamente nessa situação. Além disso, a padronização se comporta de maneira melhor que a normalização quando os dados possuem *outliers*.

Já a normalização é utilizada quando os dados não seguem distribuição normal. Esse método é bastante utilizado em conjunto com algoritmos de machine learning que não assumem qualquer distribuição dos dados, como as redes neurais ou o algoritmo KNN(K-nearest neighbours).

Ajuste 4 : Lidando com variáveis categóricas

Alguns algoritmos de machine learning não são capazes de interpretar variáveis categóricas da maneira como elas se encontram por padrão. É necessário criar uma matriz de modelagem, que contém apenas valores numéricos, para que o algoritmo funcione corretamente. Vamos explicar mais detalhadamente após a demonstração.

Utilizaremos novamente os dados do Titanic. Note que os NAs já foram preenchidos e as variáveis de texto que deveriam ser intepretadas como fatores já foram transformadas.

Vamos supor que queremos criar um modelo que prevê se o passageiro sobreviveu ou não baseado nas seguintes variáveis: Sexo, Idade, Preço da passagem (Fare) e local de embarcação.

Criamos a matriz de modelagem com a função **model.matrix**.

```
formula_titanic <- Survived ~ Sex + Age + Fare + Embarked

matriz_modelagem <- model.matrix(formula_titanic,
                                 titanic)
```

Vamos visualizar as primeiras linhas da matriz de modelagem:

```
head(matriz_modelagem)
```

```
##   (Intercept) Sexmale        Age       Fare EmbarkedQ EmbarkedS
## 1           1       1 -0.5654189 -0.5021631         0         1
## 2           1       0  0.6634884  0.7864036         0         0
## 3           1       0 -0.2581921 -0.4885799         0         1
## 4           1       0  0.4330683  0.4204941         0         1
## 5           1       1  0.4330683 -0.4860644         0         1
## 6           1       1 -0.1045787 -0.4778481         1         0
```

A matriz de modelagem transforma as variáveis categóricas (Sex e Embarked) em variáveis numéricas que assumem os valores 0 ou 1.

Caso a variável categórica só assuma 2 valores, como é o caso da variável "Sex", a matriz de modelagem irá transformá-la em uma variável "Sexmale", que assumirá valor 1, caso o passageiro seja do sexo masculino, e 0 caso contrário. Não existe regra específica para decidir qual nível será escolhido para essa nova variável. Além disso, para o modelo, não haveria diferença na predição caso o contrário ocorresse.

Caso a variável possa assumir mais de dois valores, como é o caso da variável "Embarked", que possui 3 valores, criam-se duas variáveis. No caso, foram criadas as variáveis "EmbarkedQ" e "EmbarkedS". Cada uma assume valor 1, caso o passageiro tenha embarcado em Queenstown ou Southampton, respoectivamete e 0, caso contrário. Caso o passageiro tenha embarcado em Cherbourg, as duas variáveis terão valor 0, não sendo necessário criar uma terceira variável.