Hands On

Linguagem R

Hands On - Agenda

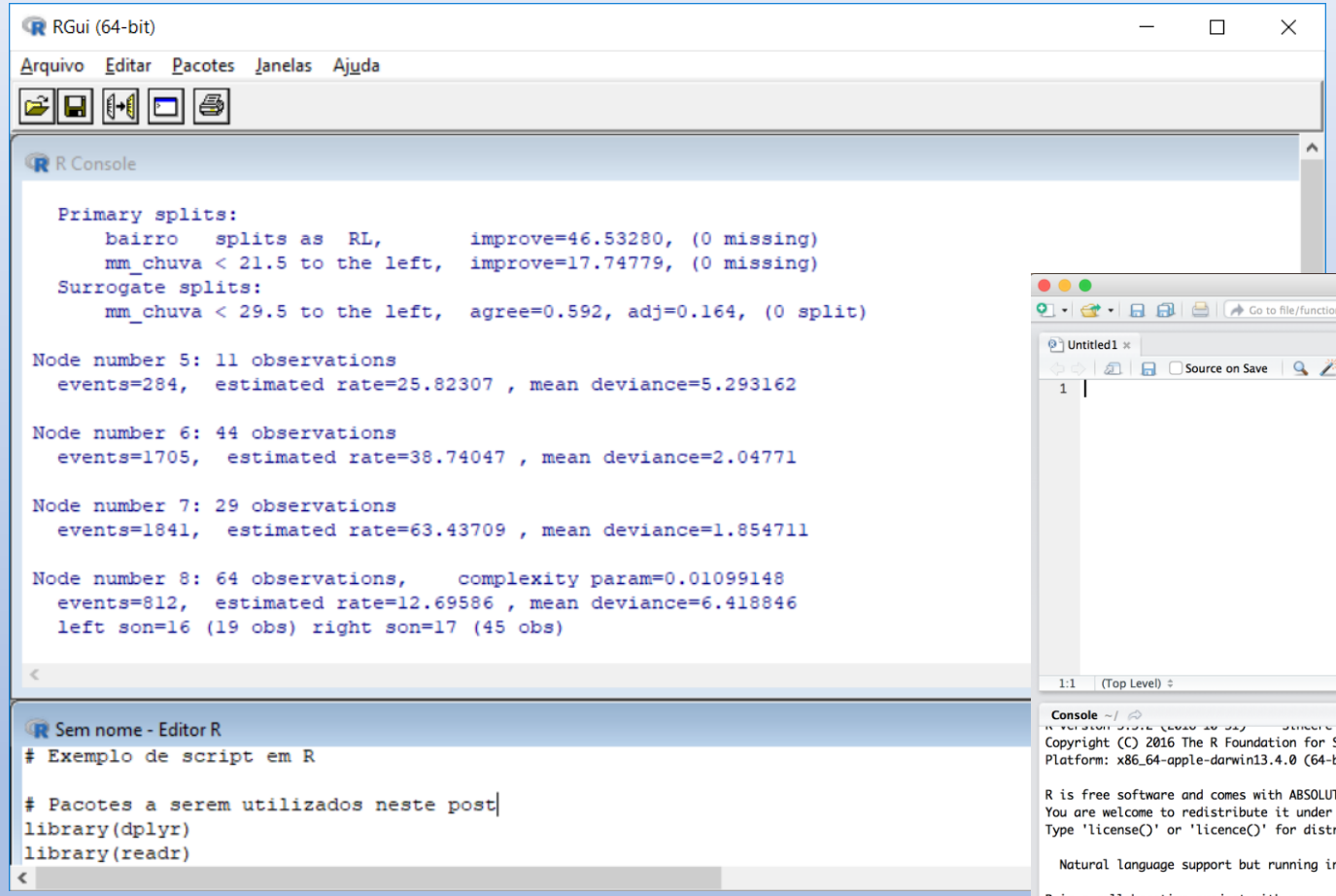
Hands-on Linguagem R básica para Análise de Dados

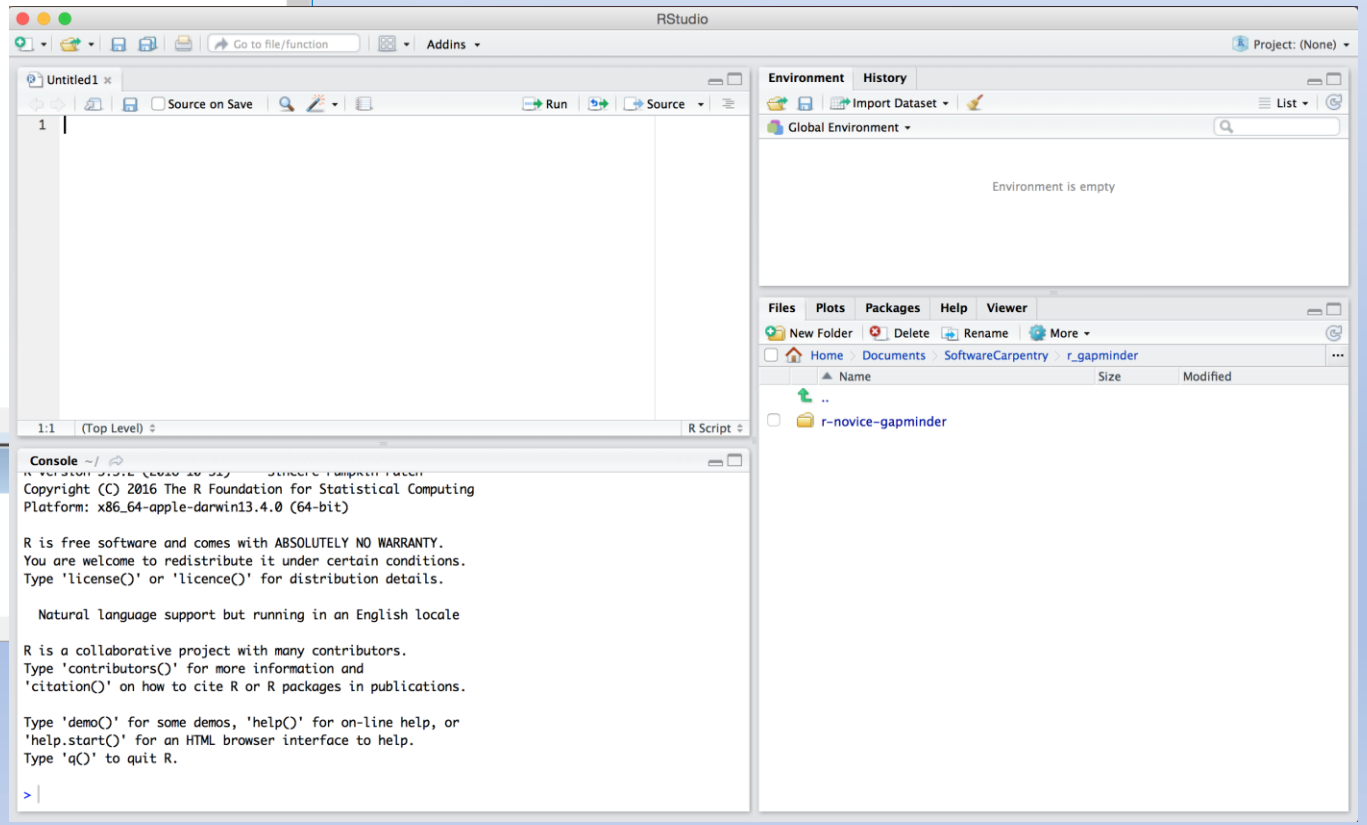
1. R-GUI e R-Studio (1)
2. Biblioteca Cran-R (2)
3. Estrutura da Linguagem (3)

3.1 Objetos R (3.1)
3.2 Operações simples com objetos (3.2)
3.3 Criando funções (3.3)
3.4 Concatenação e Arrays (3.4)
3.5 Listando e removendo objetos (3.5)
3.6 Geração de Sequências (3.6)
3.7 Indexação de Objetos (3.7)
3.8 Data frames (3.8)
3.9 Data Querying e estatística base (3.9)
3.10 Lidando com dados não disponíveis (3.10)
3.11 Lendo dados de um dataset em arquivo (3.11)
3.12 Salvando um dataset em arquivo (3.12)
3.13 Funções estatísticas básicas (3.13)
3.14 Estruturas de programação (3.14)

4. Avaliando uma análise de dados em R com um modelo de árvore de decisão (4)

1. R-GUI e R-Studio





2. Biblioteca Cran-R

A CRAN-R (Comprehensive R Archive Network) é um repositório de pacotes/bibliotecas para a linguagem R.

```
>install.packages("dplyr")
>install.packages(“rpart”)
>install.packages(“rpart.plot”)
```

```
> data(package="dplyr")
```

```
> library(dplyr)
```

2. Biblioteca Cran-R

As principais fontes de informação sobre a linguagem R estão nos seguintes endereços:

- https://www.rdocumentation.org
- https://www.r-project.org/other-docs.html

3. Estrutura da Linguagem

3.1 Objetos R

Podemos dizer que quase tudo em R é um objeto (variáveis, instruções, funções, etc). Entretanto, os objetos desta linguagem possuem uma característica própria: eles são não-transientes na seção. Antes de explicar o que isso significa, vamos criar um objeto:

```
> d <- 30
```

3. Estrutura da Linguagem

3.2 Operações simples com objetos

A linguagem R é dotada de todas as funções matemáticas desde as mais básicas até as mais complexas. Para ilustrar, podemos começar com os seguintes objetos:

```
> dist1 <- 15
> dist2 <- 20
> dist1 + dist2
[1] 35
```

3. Estrutura da Linguagem

3.2 Operações simples com objetos

A seguir temos uma lista básica das principais funções para a realização de outras operações matemáticas na linguagem R:

Função Matemática	Função R
Raiz quadrada de x	sqrt(x)
X elevado a y	X^y
Exponencial de x	exp(x)
Logaritmo natural de x na base y	log(x,y)
Fatorial de x	factorial(x)
Seno de x em radianos	sin(x)
Cosseno de x em radianos	cos(x)
Tangente de x em radianos	tan(x)
Valor absoluto/módulo de x	abs(x)
Média das linhas do data frame x	rowMeans(x)
Soma das colunas do data frame x	colMeans(x)
Soma das linhas do data frame x	rowSums(x)

3. Estrutura da Linguagem

3.3 Criando funções

Anteriormente utilizamos algumas funções matemáticas para fazermos alguns cálculos. Em R, funções são objetos criados com pedaços de códigos organizados para realizar uma operação desejada com base em parâmetros e que retornam valores resultantes para quem acionou a função. Tomemos os seguintes exemplos a seguir:

```
> somador <- function(a,b,c) {
+ result <- a+b+c
+ print(result)
+ }
> diminuidor <- function(a,b,c) {
+ a-b-c
+ }
```

3. Estrutura da Linguagem

3.4 Concatenação e Arrays

Em R a forma que utilizamos para criar arrays (estruturas compostas de dados) é através da concatenação. Nesta linguagem, concatenação é o processo de combinar, ligar ou juntar valores numéricos ou caracteres. Esta é uma das operações mais utilizadas para a realização de análise de dados e isto é alcançado através da função concatenar, que é conhecida pela forma: c().

```
> postos <- c(1,2,3,4,5,6)
> postos
[1] 1 2 3 4 5 6
```

3. Estrutura da Linguagem

3.5 Listando e removendo objetos

Lembra quando foi dito que os objetos em R são não-transientes na seção? Pronto, agora vamos ver o que isto quer dizer. Experimente executar a função a seguinte:

```
> ls()
```

```
> rm(a..b)
```

```
> rm(list=ls())
```

3. Estrutura da Linguagem

3.6 Geração de Sequências

Na linguagem R é possível gerar sequências numéricas de várias formas e a mais comum é através do símbolo ":" (dois pontos). Através dele é possível gerar sequencias lineares regulares, como por exemplo:

```
> 1:15
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> 21:28
 [1] 21 22 23 24 25 26 27 28
```

3. Estrutura da Linguagem

3.6 Geração de Sequências (cont.)

Caso haja a necessidade de construir uma sequência mais elaborada, podemos utilizar a função seq(). Através dela podemos utilizar uma razão para estabelecer o crescimento da sequência. No exemplo a seguir temos uma sequência numérica de 1 a 10 com a razão 2:

```
> seq(from=1, to=10, by=2)
[1] 1 3 5 7 9
```

3. Estrutura da Linguagem

3.6 Geração de Sequências (cont.)

Mantendo sua característica estatística e matemática, R também consegue calcular a razão da sequência entre dois valores com base no seu comprimento. Se pegarmos o exemplo anterior e fizermos o seguinte:

```
> seq(from=1, to=10, length=5)
[1] 1.00 3.25 5.50 7.75 10.00
```

3. Estrutura da Linguagem

3.6 Geração de Sequências (cont.)

Vamos tentar agora o seguinte exemplo:

```
> seq(from=-10, to=10, length=8)

[1] -10.000000  -7.142857  -4.285714  -1.428571
1.428571   4.285714   7.142857

[8]  10.000000
```

3. Estrutura da Linguagem

3.6 Geração de Sequências (cont.)

No próximo exemplo, temos a versatilidade da função seq(), desta vez sem utilizar o parâmetro “to”.

```
> seq(from=10, length=5, by=1.5)
[1] 10.0 11.5 13.0 14.5 16.0
```

E agora utilizando o parâmetro “along”:

```
> salas=1:5
> salas
[1] 1 2 3 4 5
> seq(from=10, to=12, along=salas)
[1] 10.0 10.5 11.0 11.5 12.0
```

3. Estrutura da Linguagem

3.7 Indexação de Objetos

Indexação de objetos e subscripting representam o ato de acessar ou extrair valores de um objeto multivalorado. Acessar dados através de indexação é a forma básica de realizar consultas e realizar análises em R.

Para esta atividade utilizamos o colchetes “[]” quando formos acessar um elemento específico de um array. Vamos criar o seguinte array:

```
> litros<-seq(from=2, to=10, length=6)
> litros
[1]  2.0  3.6  5.2  6.8  8.4 10.0
```

Agora vamos consultar o terceiro elemento deste array:

```
> litros[3]
[1] 5.2
```

3. Estrutura da Linguagem

3.7 Indexação de Objetos (cont.)

Caso você queira extrair todos os valores de litros com exceção do segundo valor, utilize:

```
> litros[-2]
```

3. Estrutura da Linguagem

3.7 Indexação de Objetos (cont.)

Caso você queira extrair dois ou mais valores, você pode utilizar a função de concatenação c(). Por exemplo, para extrair o segundo e o quinto, basta fazer o seguinte:

```
> litros[c(2,5)]
```

Caso você não queira nem o segundo nem o quinto, use:

```
> litros[c(-2,-5)]
```

3. Estrutura da Linguagem

3.7 Indexação de Objetos (cont.)

Outra forma de consulta é utilizando sequências:

```
> litros[2:4]
[1] 3.6 5.2 6.8
```

3. Estrutura da Linguagem

3.7 Indexação de Objetos (cont.)

Subscripting é uma técnica de indexação baseada em valores lógicos. Em R podemos fazer uma consulta para listar os valores acima de 6 litros da seguinte forma:

```
> litros[ litros>6 ]
[1]  6.8  8.4 10.0
```

3. Estrutura da Linguagem

3.8 Data Frames

Data frames são as estruturas de dados mais comuns utilizadas na linguagem R. Outras estruturas como vetores, arrays, listas, matrizes e fatores também estão disponíveis, entretanto, devido a característica de ser um objeto composto por múltiplos valores estruturados como uma tabela (com linhas e colunas), os Data Frames circundam o mundo das análises dados.

3. Estrutura da Linguagem

3.8 Data Frames (cont.)

Data frames podem ser criados a partir de arquivos ou através de outros objetos, vamos ver um exemplo:

```
> df_reservatorio <- data.frame(Tanque=11:20,
Litros=seq(from=20, by=1.25, length=10))

> df_reservatorio
   Tanque Litros
1      11  20.00
2      12  21.25
3      13  22.50
4      14  23.75
5      15  25.00
6      16  26.25
7      17  27.50
8      18  28.75
9      19  30.00
10     20  31.25
```

3. Estrutura da Linguagem

3.8 Data Frames (cont.)

Os data frames são organizados em linhas e colunas e podem ser consultados de forma parecida com os arrays utilizando os colchetes da seguinte forma:

```
> df_reservatorio[3,2]
[1] 22.5
```

3. Estrutura da Linguagem

3.8 Data Frames (cont.)

Podemos também visualizar todos os valores de uma coluna como um array. Por exemplo:

```
> df_reservatorio[,2]
 [1] 20.00 21.25 22.50 23.75 25.00 26.25 27.50
28.75 30.00 31.25
```

3. Estrutura da Linguagem

3.8 Data Frames (cont.)

Podemos forçar o array da coluna se comportar como um data frame, usando um cast. Por exemplo:

```
> as.data.frame(df_reservatorio[,2])
   df_reservatorio[, 2]
1                 20.00
2                 21.25
3                 22.50
4                 23.75
5                 25.00
6                 26.25
7                 27.50
8                 28.75
9                 30.00
10                31.25
```

3. Estrutura da Linguagem

3.8 Data Frames (cont.)

Podemos também mostrar os valores de uma linha específica:

```
> df_reservatorio[2,]
  Tanque Litros
2     12  21.25
```

Assim como fizemos com o array, podemos indexar os valores utilizando sequências:

```
> df_reservatorio[2:5,]
  Tanque Litros
2     12  21.25
3     13  22.50
4     14  23.75
5     15  25.00
```

3. Estrutura da Linguagem

3.8 Data Frames (cont.)

Os data frames permitem a extração de valores via subscripting pelo nome da coluna, utilizando o cifrão ($).

```
> df_reservatorio$Litros
 [1] 20.00 21.25 22.50 23.75 25.00 26.25 27.50 28.75
30.00 31.25
```

É possível também utilizar o subscripting via nome e especificar a posição do elemento:

```
> df_reservatorio$Litros[5]
[1] 25
```

3. Estrutura da Linguagem

3.8 Data Frames (cont.)

Assim como podemos utilizar sequência e concatenação:

```
> df_reservatorio$Litros[1:3]
[1] 20.00 21.25 22.50

> df_reservatorio$Litros[c(2,6)]
[1] 21.25 26.25
```

3. Estrutura da Linguagem

3.8 Data Frames (cont.)

Data frames são úteis para manipular Data Sets e por isso muitas vezes é importante conhecermos o número de dimensões que eles possuem. Por este motivo, existem funções específicas para obtermos informações essenciais sobre como estes objetos estão dimensionados.

Se quisermos verificar a quantidade de colunas ou de linhas de um Data Frame, podemos fazer o seguinte:

```
> ncol(df_reservatorio)
[1] 2
> nrow(df_reservatorio)
[1] 10
```

3. Estrutura da Linguagem

3.8 Data Frames (cont.)

Data frames são úteis para manipular Data Sets e por isso muitas vezes é importante conhecermos o número de dimensões que eles possuem. Por este motivo, existem funções específicas para obtermos informações essenciais sobre como estes objetos estão dimensionados.

Se quisermos verificar a quantidade de colunas ou de linhas de um Data Frame, podemos fazer o seguinte:

```
> ncol(df_reservatorio)
[1] 2
> nrow(df_reservatorio)
[1] 10
```

3. Estrutura da Linguagem

3.8 Data Frames (cont.)

Outra forma de fazer isso é através da função dim(), que retorna o número de linhas e colunas:

```
> dim(df_reservatorio)
[1] 10  2
```

No caso de arrays, existe a função length() que retorna a quantidade de elementos existentes no mesmo:

```
> length(df_reservatorio[,2])
[1] 10
```

3. Estrutura da Linguagem

3.9 Data Querying e estatística base

Quando começamos atrabalhar com um dataset desconhecido devemos analisá-lo de forma a termos ideia de suas dimensões e das características dos dados contidos nele. Por isso, é comum realizarmos questionamentos que nos levarão a obtermos a devidas respostas sobre o perfil do dataset e seus dados.

Para fins de estudo, R disponibiliza por padrão os chamados in-built datasets. Na listagem a seguir temos 104 deles listados via comando ls("package:datasets"):

```
> ls("package:datasets")
```

3. Estrutura da Linguagem

3.9 Data Querying e estatística base (cont.)

Vamos escolher o dataset sunspot.year que contém o número de manchas solares observadas e registradas entre os anos de 1700 e 1988.

```
> sunspot.year
Time Series:
Start = 1700
End = 1988
Frequency = 1
  [1]   5.0  11.0  16.0  23.0  36.0  58.0  29.0  20.0  10.0   8.0   3.0
0.0
 [13]   0.0   2.0  11.0  27.0  47.0  63.0  60.0  39.0  28.0  26.0  22.0
11.0
 [25]  21.0  40.0  78.0 122.0 103.0  73.0  47.0  35.0  11.0   5.0  16.0
34.0
 [37]  70.0  81.0 111.0 101.0  73.0  40.0  20.0  16.0   5.0  11.0  22.0
40.0
 [49]  60.0  80.9  83.4  47.7  47.8  30.7  12.2   9.6  10.2  32.4  47.6
54.0
```

3. Estrutura da Linguagem

3.9 Data Querying e estatística base (cont.)

Como foi dito anteriormente, a melhor forma de trabalharmos com um dataset e colocando ele num data frame. Faremos isto da seguinte forma:

```
> manchas_solares <- data.frame(Ano=1700:1988,
NumManchas=sunspot.year)
```

3. Estrutura da Linguagem

3.9 Data Querying e estatística base (cont.)

Agora que temos nosso data frame criado, podemos utilizar algumas funções de Data Querying para explorá-lo. Muitas vezes é útil visualizarmos o início de um dataset para sabermos como ele está organizado:

```
> head(manchas_solares)
   Ano NumManchas
1 1700          5
2 1701         11
3 1702         16
4 1703         23
5 1704         36
6 1705         58
```

3. Estrutura da Linguagem

3.9 Data Querying e estatística base (cont.)

Quando estamos estudando o perfil dos dados, muitas vezes precisamos saber se eles se enquadram em algum critério de qualidade. Por exemplo, a existência de valores negativos é indesejável num dataset de manchas solares. Podemos testar isso da seguinte forma:

```
> any(manchas_solares[,2]<0)
[1] FALSE
```

A função any() verificou se existia algum valor negativo na coluna NumManchas e retornou FALSE, indicando que não há. Da mesma forma, poderíamos verificar se existe algum ano que esteja fora do período anunciado dos dados que vai de 1700 a 1988.

```
> any(manchas_solares[,1]<1700 | manchas_solares[,1] >
1988)
[1] FALSE
```

3. Estrutura da Linguagem

3.9 Data Querying e estatística base (cont.)

Após verificarmos que o dataset está em condições de uso, podemos iniciar um estudo dos dados com alguma estatística descritiva sobre ele. Por exemplo, podemos verificar a média do manchas solares do período:

```
> mean(manchas_solares[,2])
[1] 48.61349

> round(mean(manchas_solares[,2]),2)
[1] 48.61
```

3. Estrutura da Linguagem

3.9 Data Querying e estatística base (cont.)

Agora quero saber quais os anos onde não foi registrado nenhuma mancha solar.

```
> which(manchas_solares[,2]==0)
[1]  12  13 111

> manchas_solares[which(manchas_solares[,2]==0),]
     Ano NumManchas
12  1711          0
13  1712          0
111 1810          0
```

A função which() permite realizar Data Querying em datasets grandes com excelente performance. No segundo exemplo anterior, listamos o conteúdo das linhas onde a mancha solar é zero, enquanto no primeiro apenas obtivemos as linhas referentes a zero na contagem de manchas.

3. Estrutura da Linguagem

3.9 Data Querying e estatística base (cont.)

Se você quisesse saber a quantidade de anos onde a leitura foi zero, bastaria fazer o seguinte:

```
> length(which(manchas_solares[,2]==0))
[1] 3
```

A variedade de funções de Data Querying em R é bastante variada. Podemos de uma única vez realizar um sumário estatístico dos dados fazendo:

```
> summary(manchas_solares[,2])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00   15.60   39.00   48.61   68.90  190.20
```

3. Estrutura da Linguagem

3.10 Lidando com dados não disponíveis

Uma coisa muito comum para quem lida com Data Sets é a situação onde em determinadas colunas existem lacunas de dados ou dados ausentes. Isto faz parte da nossa rotina.

Vamos utilizar outro dataset in-built do R, desta vez sacaremos o airquality que possui informações sobre a qualidade dor ar medida na cidade de Nova Iorque entre os meses de maio e setembro de 1973.

```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

3. Estrutura da Linguagem

3.10 Lidando com dados não disponíveis (cont.)

A presença de valores NA nos impedem de realizar alguns cálculos como por exemplo:

```
> mean(airquality[,2])
[1] NA
```

Este resultado foi exibido porque a função mean() não sabia o que fazer ao encontrar NA. Entretanto, é possível orientar a função a desconsiderar o valor NA através do parâmetro na.rm (NA remove):

```
> mean(airquality[,2],na.rm=TRUE)
[1] 185.9315
```

Você pode se antecipar ao utilizar um dataset e verificar se o mesmo contém valores NA através da função any() em conjunção com uma função de teste lógico: is.na():

```
> any(is.na(airquality))
[1] TRUE
```

3. Estrutura da Linguagem

3.10 Lidando com dados não disponíveis (cont.)

Caso haja uma decisão de projeto onde as linhas que contém NA deverão ser desconsideradas, podemos utilizar a função na.omit() para fazer isso gerando um novo dataset.

```
> airquality_ok <- na.omit(airquality)
> head(airquality_ok)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
7    23     299  8.6   65     5   7
8    19      99 13.8   59     5   8
```

3. Estrutura da Linguagem

3.11 Lendo dados de um dataset em arquivo

Como em qualquer outra linguagem, R também possui recursos para trabalhar com fontes de dados externas. Uma das formas mais comuns é a leitura de datasets contidos em arquivos texto. Para isso a linguagem fornece a função read.table() que cria automaticamente um data frame a partir dos dados lidos.

Vamos começar com um dataset in-built que contém informações mensais sobre o quantitativo de passageiros em voos internacionais entre 1949 e 1960, chamado AirPassengers. Vamos pegar seu conteúdo, colar num bloco de notas e salvá-lo no Desktop com o nome “Passageiros.txt”.

3. Estrutura da Linguagem

3.11 Lendo dados de um dataset em arquivo (cont.)

Agora vamos definir o Desktop como nossa pasta de trabalho:

```
> setwd("c:/Documents and Settings/seu_usuario/Desktop/")
> getwd() # use para verificar se funcionou
```

Agora vamos criar nosso data frame lendo o arquivo que acabamos de criar:

```
> passageiros <- read.table("Passageiros.txt", header=TRUE, sep="")
> head(passageiros)
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949 112 118 132 129 121 135 148 148 136 119 104 118
1950 115 126 141 135 125 149 170 170 158 133 114 140
1951 145 150 178 163 172 178 199 199 184 162 146 166
1952 171 180 193 181 183 218 230 242 209 191 172 194
1953 196 196 236 235 229 243 264 272 237 211 180 201
1954 204 188 235 227 234 264 302 293 259 229 203 229
```

3. Estrutura da Linguagem

3.12 Salvando um dataset em arquivo

Seguindo uma forma similar à leitura, temos a operação de salvamento de um dataset em arquivo e para isto, temos a função write.table(). Antes de utilizá-la, vamos incrementar um pouco o dataset do tópico anterior.

Vamos adicionar a ele uma coluna com o total anual de passageiros.

```
> passageiros$TotAnual <- cbind(rowSums(passageiros))
```

3. Estrutura da Linguagem

3.12 Salvando um dataset em arquivo (cont.)

Utilizamos a função cbind() para gerar uma coluna de dados anônima e em seguida criamos uma nova coluna (TotAnual) em nosso data frame com a utilização do cifrão ($).

Feito isso, podemos gravar nosso novo dataset:

```
> write.table(passageiros,"tot_passageiros.txt",col.names=NA,
row.names=TRUE, quote=FALSE, sep="\t")
```

3. Estrutura da Linguagem

3.13 Funções estatísticas básicas

A seguir temos um pequeno resumo de algumas funções estatísticas básicas disponíveis em R.

Função em R	Valor retornado
mean(x)	Média aritmética dos valores do objeto x
median(x)	Mediana dos valores do objeto x
max(x)	Máximo valor no objeto x
min(x)	Mínimo valor no objeto x
range(x)	Vetor do min(x) e max(x)
sum(x)	Total de todos os valores em x
var(x)	Variância de x
quantile(x)	Vetor contendo o mínimo, o menor quantil, mediana, quantil superior e o máximo de x
cumsum(x)	Soma acumulativa de x
cumprod(x)	Produto acumulativo de x
colMeans(x)	Média das colunas do data frame x
rowMeans(x)	Média das linhas do data frame x
colSums(x)	Soma das colunas do data frame x
rowSums(x)	Soma das linhas do data frame x

3. Estrutura da Linguagem

3.14 Estruturas de programação

Escrever scripts ou programas em R é similar a outras linguagens, por exemplo:

```
# Exemplo de script em R
fatura1 <- 110
fatura2 <- 350

fat_medio <- (fatura1+fatura2)/2

if (fat_medio < 50) {
   print("Faturamento abaixo da meta")
} else {
   print("Meta cumprida!")
}
fat_medio
```

3. Estrutura da Linguagem

3.14 Estruturas de programação (cont.)

Podemos também utilizar um conceito de iteração restrita para a criação de laços:

```
# Exemplo de script em R

for( i in 1:5 ) {
   if ( i < 3 ) {
      cat("Setor Sul",i,"\n")
   } else {
      cat("Setor Norte",i,"\n")
   }
}
```

3. Estrutura da Linguagem

3.14 Estruturas de programação (cont.)

Outro conceito importante é o de iteração irrestrita para a criação de loops:

```
# Exemplo de script em R
i <- 1
while( i <= 5 ) {
   if ( i < 3 ) {
      cat("Setor Sul",i,"\n")
   } else {
      cat("Setor Norte",i,"\n")
   }
   i <- i + 1
}
```

Iterações irrestrita é algo que deve ser evitado ao se trabalhar com analytics devido à possibilidade de estabelecer um loop infinito. Quando pensamos em grande volume de dados, uma farm de servidores pode ser sacrificada por um deslize de um programador ao utilizar este recurso, portanto tenha cautela ao utilizá-las.

4. Avaliando uma análise de dados em R com um modelo de árvore de decisão

```
####
# Exemplo de Arvore de Decisao em R
# com base em regressao linear
# Utilizando uma base de teste de transito em dias de chuva
# Arquivo utilizado: chuva_transito.csv
#

# importa as bibliotecas
library(readr)  # lib especializada em data frames
library(rpart)  # lib especializada em arvores de regressao e
particionamento de ramos
library(rpart.plot) # lib especializada em plotar arvores de
regressao e particionamento de ramos

# ler o arquivo com as informacoes sobre chuva e o transito
chuvaTransito = read.csv("chuva_transito.csv",header=TRUE, sep=";")

# calcula a regressao linear para a construcao da arvore
arv_transito = rpart(velocidade ~ mm_chuva + bairro,
data=chuvaTransito, method="poisson")

# realiza a plotagem da arvore de decisao
prp(arv_transito)
```

4. Avaliando uma análise de dados em R com um modelo de árvore de decisão

Criamos um dataframe chuvaTransito para armazenar os dados do arquivo CSV. Aqui utilizamos uma variação da função read.table() que também pode ser escrita como read.csv() (isso é outra coisa comum em R).

```
arv_transito = rpart(velocidade ~ mm_chuva + bairro,
data=chuvaTransito, method="poisson")
```

Na linha acima, estabelecemos a fórmula do modelo analítico que queremos explorar. A função rpart funciona com base em uma fórmula matemática para estabelecer uma relação causal to tipo y em função de x (ou y ~ x), sendo que x é uma operação direta entre valores.

Desta forma, podemos traduzir a fórmula da seguinte maneira:

velocidade ~ mm_chuva + bairro

significa que queremos a correlação linear da velocidade do trânsito em função da chuva levando em consideração o bairro. Para isso, informamos à função que ela deverá utilizar como fonte de dados o data frame chuvaTransito e que a árvore deve utilizar o método de Poisson. Os métodos aceitos pela rpart são: class, Poisson, anova e exp (somente para objetos do tipo survive, que são orientados a eventos ao longo do tempo).

4. Avaliando uma análise de dados em R com um modelo de árvore de decisão