

Introdução ao R para Análise de Dados

Minicurso

Paulo Alexandrino

Março 2023

Sumário

1	Conceitos Fundamentais	3
1.1	Operadores	3
1.1.1	Operadores matemáticos	3
1.1.2	Operadores lógicos	4
1.2	Objetos, Tipos de Dados e Variáveis	6
1.2.1	Vetores	6
1.2.2	Data frames	8
1.3	Funções	8
1.3.1	Explorando os dados	9
1.3.2	Estatísticas Descritivas	12
1.3.3	Criação de vetores	13
1.3.4	Amostragem	14
1.4	Baixando e executando pacotes externos	15
2	Importação	16
2.1	Arquivos separados por vírgula (.csv)	16
2.2	Arquivos Excel (.xls e .xlsx)	17
3	Visualização	19
3.1	Gráfico de linha	19
3.2	Gráfico de dispersão	20
4	Organização e Transformação	22

5	Modelagem	23
5.1	Regressão Linear	23
5.1.1	Regressão Linear Simples	23
5.1.2	Regressão Linear Múltipla	27
5.2	Logit	29
5.3	Probit	30
6	Dicas para Continuar os Estudos	32

Capítulo 1

Conceitos Fundamentais

1.1 Operadores

1.1.1 Operadores matemáticos

A forma mais elementar de utilizar o R é como uma calculadora que nos permite realizar todo tipo de operação matemática desejada.

```
# Soma (+)  
5 + 7
```

```
## [1] 12
```

```
# Subtração (-)  
10 - 16
```

```
## [1] -6
```

```
# Multiplicação (*)  
2 * 7
```

```
## [1] 14
```

```
# Exponenciação (^)  
2^4
```

```
## [1] 16
```

```
# Divisão (/)  
5 / 2
```

```
## [1] 2.5
```

```
# Divisão Inteira (%%)  
5 %% 2
```

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

```
# Módulo - Resto da divisão inteira (%%)  
5 %% 2
```

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

1.1.2 Operadores lógicos

Operadores lógicos nos permitem estabelecer relações entre objetos e julgar proposições (orações declarativas) como verdadeiras (TRUE) ou falsas (FALSE). Serão muito úteis a construção de filtros.

```
# Menor que (<)  
5 < 3
```

```
## [1] FALSE
```

```
# Menor ou igual que (<=)  
2.5 <= 7
```

```
## [1] TRUE
```

```
# Maior que (>)  
10 > 15
```

```
## [1] FALSE
```

```
# Maior ou igual que (>=)  
11 >= 11
```

```
## [1] TRUE
```

```
# Igual a (==)
2 == 3
```

```
## [1] FALSE
```

```
# Diferente de (!=)
5 != 4
```

```
## [1] TRUE
```

A negação de uma proposição é seu inverso lógico.

```
# Negação (!)
2 > 5
```

```
## [1] FALSE
```

```
!(2 > 5)
```

```
## [1] TRUE
```

Os conectivos de conjunção (\wedge , lê-se E) e disjunção (\vee , lê-se OU) são operadores lógicos que permitem criar proposições compostas. Sejam p e q proposições quaisquer:

- A conjunção $p \wedge q$ é verdadeira se p e q são ambas verdadeiras; se ao menos uma delas for falsa, $p \wedge q$ é falsa.

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

```
# E (&)
(2 > 1) & (3 != 0)
```

```
## [1] TRUE
```

```
(3 < 2) & (5 < 11)
```

```
## [1] FALSE
```

- A disjunção $p \vee q$ é verdadeira se p e q se ao menos uma das proposições p ou q é verdadeira; se p e q são ambas falsas, então $p \vee q$ é falsa.

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

```
# OU ()  
(3 < 2) | (5 < 11)
```

```
## [1] TRUE
```

```
(5 == 6) | (12 < 8)
```

```
## [1] FALSE
```

1.2 Objetos, Tipos de Dados e Variáveis

Em R praticamente tudo é um objeto: um número, uma palavra, um vetor contendo diversos números, uma tabela contendo diferentes vetores, uma lista contendo várias tabelas...as coisas podem ficar tão complicadas quanto se queira.

1.2.1 Vetores

Vetores são objetos formados por dados de um mesmo tipo. Um vetor pode conter nenhum, um ou mais valores. Alguns dos principais tipos de dados são:

- **logical**: TRUE ou FALSE
- **integer**: números inteiros
- **numeric** ou **double**: números reais
- **character**: texto

```
# Para atribuir um valor a uma variável utilizados "<-" ou então "=".  
# Não confundiro o = de atribuição com o == operador lógico.  
# Vetores podem ser criados, ou concatenados como veremos mais adiante,  
# utilizando a função c().
```

```
# Criando a variável vetor_vazio e deixando ela vazia  
vetor_vazio <- c()  
# Imprimindo a variável vetor_vazio  
vetor_vazio
```

```
## NULL
```

```
# Podemos atribuir apenas um valor a uma variável  
var_texto <- "Palavra"  
var_texto
```

```
## [1] "Palavra"
```

```
# Podemos atribuir qualquer valor a uma variável  
var_numerica <- 3*(5 + 2)  
var_numerica
```

```
## [1] 21
```

```
# Criando um vetores com três valores  
var_texto <- c("Palavra", "Word", "Mot")  
var_texto
```

```
## [1] "Palavra" "Word"      "Mot"
```

```
# Criando um vetor lógico  
var_logico = c(3 + 4 < 5, 2 == 1 + 1)  
var_logico
```

```
## [1] FALSE TRUE
```

Para apagar uma variável, utilizamos a função `rm()`. Aprenderemos sobre funções mais adiante. A princípio basta saber que dentro da função `rm()` colocaremos o nome das variáveis que queremos apagar.


```
# Apagando as variáveis var_logico, var_numerica e var_texto
# criadas anteriormente
rm(var_logico, var_numerica, var_texto)

# Para apagar todas as variáveis
# Use com cuidado!
rm(list = ls())
```

1.2.2 Data frames

Podemos agrupar vetores em um data frame. Em R, data frames são tabelas bidimensionais, semelhantes às planilhas do Excel. Este tipo de objeto será o mais utilizado em nosso dia-a-dia.

Vamos construir um data frame dos estados da região sudeste do Brasil, com as informações de nome, sigla, nome da capital, código DDD da capital e se a capital possui acesso para o mar.

```
# Criando um data frame:
estados_sudeste <- data.frame(
  estado = c("Rio de Janeiro", "Sao Paulo", "Minas Gerais", "Espírito Santo"),
  sigla = c("RJ", "SP", "MG", "ES"),
  capital = c("Rio de Janeiro", "Sao Paulo", "Belo Horizonte", "Vitoria"),
  DDD_capital = c(21, 11, 31, 27),
  mar_na_capital = c(TRUE, FALSE, FALSE, TRUE)
)

estados_sudeste
```

	estado	sigla	capital	DDD_capital	mar_na_capital
## 1	Rio de Janeiro	RJ	Rio de Janeiro	21	TRUE
## 2	Sao Paulo	SP	Sao Paulo	11	FALSE
## 3	Minas Gerais	MG	Belo Horizonte	31	FALSE
## 4	Espírito Santo	ES	Vitoria	27	TRUE

Esse procedimento funciona para data frames pequenos, mas não é difícil notar que se torna uma tarefa trabalhosa conforme aumenta-se o número de observações e variáveis. Felizmente, o R permite importar dados de outras fontes, o que aprenderemos a fazer em breve.

1.3 Funções

Funções são ferramentas, porções de código reutilizáveis. Elas nos permitem realizar os mais variados tipos de operações, seja criar ou modificar objetos,

realizar comandos no computador, e muito mais.

Funções quase sempre recebem argumentos, de forma a ajustar seu comportamento de acordo com algum objetivo específico. Alguns argumentos são obrigatórios, outros opcionais, ou seja, possuem valores pré-definidos.

Caso queira obter mais informações sobre uma função, você pode acessar sua documentação. A documentação contém tudo, ou a maior parte, do que se precisa saber sobre uma função: quais seus argumentos, como eles se comportam, quais os valores pré-definidos. Para acessar a documentação basta digitar um ponto de interrogação seguido do nome da função.

Para aprendermos a usar algumas funções essenciais, usaremos o dataset `USArrests`, que contém dados sobre criminalidade nos 50 estados dos EUA no ano de 1973, e já vem pré-carregado no R. Assim, não precisaremos importar nada.

1.3.1 Explorando os dados

```
# Vendo a estrutura dos dados:  
str(USArrests)
```

```
## 'data.frame': 50 obs. of 4 variables:  
## $ Murder : num 13.2 10 8.1 8.8 9 7.9 3.3 5.9 15.4 17.4 ...  
## $ Assault : int 236 263 294 190 276 204 110 238 335 211 ...  
## $ UrbanPop: int 58 48 80 50 91 78 77 72 80 60 ...  
## $ Rape : num 21.2 44.5 31 19.5 40.6 38.7 11.1 15.8 31.9 25.8 ...
```

```
# Vendo um resumo dos dados:  
summary(USArrests)
```

```
##      Murder      Assault      UrbanPop      Rape  
## Min.   : 0.800   Min.   : 45.0   Min.   :32.00   Min.   : 7.30  
## 1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07  
## Median : 7.250   Median :159.0   Median :66.00   Median :20.10  
## Mean   : 7.788   Mean   :170.8   Mean   :65.54   Mean   :21.23  
## 3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18  
## Max.   :17.400   Max.   :337.0   Max.   :91.00   Max.   :46.00
```

```
# Vendo as primeiras observações:  
head(USArrests)
```

```
##      Murder Assault UrbanPop Rape
```

```
## Alabama      13.2      236      58 21.2
## Alaska       10.0      263      48 44.5
## Arizona       8.1      294      80 31.0
## Arkansas      8.8      190      50 19.5
## California    9.0      276      91 40.6
## Colorado      7.9      204      78 38.7
```

```
# Vendo as últimas observações:
tail(USArrests)
```

```
##           Murder Assault UrbanPop Rape
## Vermont        2.2      48      32 11.2
## Virginia        8.5     156      63 20.7
## Washington      4.0     145      73 26.2
## West Virginia   5.7      81      39  9.3
## Wisconsin       2.6      53      66 10.8
## Wyoming         6.8     161      60 15.6
```

```
# Número de linhas:
nrow(USArrests)
```

```
## [1] 50
```

```
# Número de colunas:
ncol(USArrests)
```

```
## [1] 4
```

Como vimos anteriormente, um **data frame** é uma representação tabular, onde cada linha corresponde a uma observação e cada coluna uma variável, ou atributo. Para obtermos uma coluna específica de um **data frame**, podemos utilizar o símbolo \$.

```
# Selecionando apenas a coluna Murder:
USArrests$Murder
```

```
## [1] 13.2 10.0  8.1  8.8  9.0  7.9  3.3  5.9 15.4 17.4  5.3  2.6 10.4  7.2  2.2
## [16]  6.0  9.7 15.4  2.1 11.3  4.4 12.1  2.7 16.1  9.0  6.0  4.3 12.2  2.1  7.4
## [31] 11.4 11.1 13.0  0.8  7.3  6.6  4.9  6.3  3.4 14.4  3.8 13.2 12.7  3.2  2.2
## [46]  8.5  4.0  5.7  2.6  6.8
```

```
# Classe de um objeto:  
class(USArrests$Murder)
```

```
## [1] "numeric"
```

```
# Mínimo:  
min(USArrests$Murder)
```

```
## [1] 0.8
```

```
# Máximo:  
max(USArrests$Murder)
```

```
## [1] 17.4
```

```
# Somatório:  
sum(USArrests$Murder)
```

```
## [1] 389.4
```

```
# Comprimento de um vetor:  
length(USArrests$Murder)
```

```
## [1] 50
```

```
# Ordenar um vetor em ordem crescente:  
sort(USArrests$Murder)
```

```
## [1] 0.8 2.1 2.1 2.2 2.2 2.6 2.6 2.7 3.2 3.3 3.4 3.8 4.0 4.3 4.4  
## [16] 4.9 5.3 5.7 5.9 6.0 6.0 6.3 6.6 6.8 7.2 7.3 7.4 7.9 8.1 8.5  
## [31] 8.8 9.0 9.0 9.7 10.0 10.4 11.1 11.3 11.4 12.1 12.2 12.7 13.0 13.2 13.2  
## [46] 14.4 15.4 15.4 16.1 17.4
```

```
# A função sort possui o argumento opcional decreasing = FALSE.  
# Para ordenar a lista de forma decrescente, podemos alterá-lo.  
sort(USArrests$Murder, decreasing = TRUE)
```

```
## [1] 17.4 16.1 15.4 15.4 14.4 13.2 13.2 13.0 12.7 12.2 12.1 11.4 11.3 11.1 10.4  
## [16] 10.0 9.7 9.0 9.0 8.8 8.5 8.1 7.9 7.4 7.3 7.2 6.8 6.6 6.3 6.0  
## [31] 6.0 5.9 5.7 5.3 4.9 4.4 4.3 4.0 3.8 3.4 3.3 3.2 2.7 2.6 2.6  
## [46] 2.2 2.2 2.1 2.1 0.8
```

1.3.2 Estatísticas Descritivas

Vamos usar agora outras colunas para calcularmos estatísticas descritivas utilizando funções do R.

```
# Média:  
mean(USArrests$Assault)
```

```
## [1] 170.76
```

```
# Mediana:  
median(USArrests$Assault)
```

```
## [1] 159
```

```
# Quantis:  
quantile(USArrests$Assault)
```

```
##    0%   25%   50%   75%  100%  
##    45   109   159   249   337
```

```
# Variância:  
var(USArrests$Assault)
```

```
## [1] 6945.166
```

```
# Desvio padrão:  
sd(USArrests$Assault)
```

```
## [1] 83.33766
```

```
# Covariância:  
cov(USArrests$Assault, USArrests$Murder)
```

```
## [1] 291.0624
```

```
# Correlação:  
cor(USArrests$Assault, USArrests$Murder)
```

```
## [1] 0.8018733
```

Ao resultado de uma função pode ser aplicada outra função, o que é chamado de aninhamento. Vamos supor, por exemplo que queiremos arredondar, com três casas decimais, o valor calculado para a correlação entre as variáveis `Assault` e `Murder`.

```
# A função utilizada para arredondar um número é round().  
# Para que tenha 2 casa decimais, precisaremos ajustar o parâmetro digits.  
round(cor(USArrests$Assault, USArrests$Murder), digits = 3)
```

```
## [1] 0.802
```

1.3.3 Criação de vetores

Vamos conhecer algumas funções que podem ser usadas para criar vetores numéricos de forma fácil.

```
# Criar um vetor de com o valor x repetido n vezes:  
rep(2, 3)
```

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

```
# Criar uma sequencia de números: seq()  
seq(from = 1, to = 50, by = 3)
```

```
## [1] 1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49
```

```
# Podemos substituir, por exemplo seq(from = x, to = y, by = 1)  
3:15
```

```
## [1] 3 4 5 6 7 8 9 10 11 12 13 14 15
```

```
# Vamos atribuir os valores anteriormente encontrados a variáveis:  
var1 <- rep(2, 3)  
var2 <- seq(from = 1, to = 50, by = 3)  
  
# Para concatenar vetores, utilizamos a função c()  
meu_vetor <- c(var1, var2)  
meu_vetor
```

```
## [1] 2 2 2 1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49
```

1.3.4 Amostragem

O R foi uma linguagem criada por estatísticos e até hoje possui como um de seus pontos fortes sua capacidade de lidar com amostragem e simulações.

```
# Permutação aleatórias dos números naturais menores que 7:  
sample(7)
```

```
## [1] 1 5 7 6 2 3 4
```

```
# Permutação aleatória dos números entre 2 e 5:  
sample(2:5)
```

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

```
# Sorteio, sem reposição, de 5 números aleatórios  
# da variável var2 que criamos anteriormente:  
sample(var2, 5)
```

```
## [1] 19 10 25 13 46
```

```
# Sorteio, com reposição, de 6 números aleatórios entre 1 e 4:  
sample(1:4, size = 6, replace = TRUE)
```

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

```
# Sorteio, sem reposição, de uma palavra em uma lista:  
sample(c("pera", "uva", "banana", "caqui"), 1)
```

```
## [1] "banana"
```

Para replicar um experimento utilizando funções que produzem resultados aleatórios, precisamos definir uma “semente”. Ela faz com que o computador gere sempre os mesmos números. Para tanto utilizamos a função `set.seed(n)`, onde `n` é um número inteiro qualquer (um dado número sempre gerará o mesmo resultado). Preferencialmente deve estar no começo do seu script, sempre antes de rodar as funções.

```
# Definindo 42 como a semente, poderia ser qualquer  
# outro número.  
set.seed(42)
```

1.4 Baixando e executando pacotes externos

Até agora utilizamos apenas funções pré-instaladas no R. Para ter acesso a novos pacotes, será necessário instalá-los primeiro. Esse procedimento só é necessário ser feito uma vez, a não ser que o R precise ser reinstalado. Sempre que iniciar uma sessão será necessário carregar os pacotes que serão utilizados.

```
# Instalando o tidyverse  
install.packages("tidyverse")  
  
# Carregando o tidyverse  
library("tidyverse")
```

O **tidyverse** é um conjunto de pacotes que juntos fornecem uma ampla gama de ferramentas para análise de dados. Algumas dessas ferramentas apresentam versões otimizadas de funções já existentes no R.

O operador pipe (`%>%`) permite substituir o aninhamento de funções por uma cadeia de operação, o que torna os códigos mais legíveis. Veremos aplicações do pipe e de algumas dos principais pacotes e funções do **tidyverse** nos capítulos posteriores.

Capítulo 2

Importação

O R nos permite importar e utilizar em nossas análises uma série de diferentes tipos de arquivos. Isso pode ser feito por meio de funções nativas ou através de ferramentas disponíveis em pacotes criados por outros usuários. Por exemplo, o **tidyverse** possui os pacotes **readr**, que permite a leitura de arquivos como **.csv**, **.fwf** e **.txt**, e o pacote **readxl**, para leitura de arquivos Excel.

2.1 Arquivos separados por vírgula (.csv)

Comma-separated values é um formato de arquivo de texto onde as observações de uma tabela são representadas por linhas e as colunas são separadas por vírgula (no padrão americano) ou ponto-e-vírgula (no padrão europeu e brasileiro). No padrão separado por vírgula, o indicador decimal é o ponto; no padrões separado por ponto-e-vírgula, a vírgula. É importante estar atento ao tipo de arquivo que estamos lidando.

O dataset **gapminder** contém dados econômicos e demográficos para diversos países a cada 5 anos de 1952 a 2007. O arquivo que usaremos como exemplo apresenta uma amostra contendo as informações disponíveis para o Brasil, Botsuana e Coreia do Sul. Ele foi salvo em **.csv** com padrão separado por ponto-e-vírgula.

```
# Primeira forma: ajustando os parâmetros da função read.csv
gapminderCountries <- read.csv("Aula1/gapminderCountries.csv",
                              sep = ";",
                              dec = ",")

# Segunda forma: utilizando a função read.csv2
gapminderCountries <- read.csv2("Aula1/gapminderCountries.csv")
```

```
# Mostrando um pedaço do dataset
head(gapminderCountries)
```

```
##   country continent year lifeExp    pop gdpPercap
## 1 Botswana      Africa 1952  47.622 442308   851.2411
## 2 Botswana      Africa 1957  49.618 474639   918.2325
## 3 Botswana      Africa 1962  51.520 512764   983.6540
## 4 Botswana      Africa 1967  53.298 553541  1214.7093
## 5 Botswana      Africa 1972  56.024 619351  2263.6111
## 6 Botswana      Africa 1977  59.319 781472  3214.8578
```

Para maiores informações, leia a documentação das funções `read.csv` e `read.csv2`.

2.2 Arquivos Excel (.xls e .xlsx)

O Microsoft Excel é um dos *softwares* de planilha mais populares do mundo. São muito utilizados não só no meio corporativo, mas também no meio acadêmico. Apesar de limitações principalmente relacionadas a lidar com datas e tabelas muito grandes, é uma ferramenta poderosa e torna-se ainda melhor se aliada ao R.

O *tidyverse* não carrega automaticamente o pacote necessário para ler arquivos Excel, portanto, precisaremos carregar o `readxl`. Utilizaremos como exemplo uma amostra do dataset `gapminder`, mas desta vez contendo informações para todos os países no ano de 2007.

```
# Carregando o pacote necessário
library(readxl)

# Importando o arquivo de interesse
gapminder2007 <- read_excel("Aula1/gapminder2007.xlsx")

# Mostrando um pedaço do dataset
head(gapminder2007)
```

```
## # A tibble: 6 x 6
##   country    continent year lifeExp    pop gdpPercap
##   <chr>      <chr>    <dbl> <dbl>    <dbl>    <dbl>
## 1 Afghanistan Asia      2007   43.8 31889923    975.
## 2 Albania    Europe    2007   76.4  3600523   5937.
## 3 Algeria    Africa    2007   72.3 33333216   6223.
```

## 4	Angola	Africa	2007	42.7	12420476	4797.
## 5	Argentina	Americas	2007	75.3	40301927	12779.
## 6	Australia	Oceania	2007	81.2	20434176	34435.

Capítulo 3

Visualização

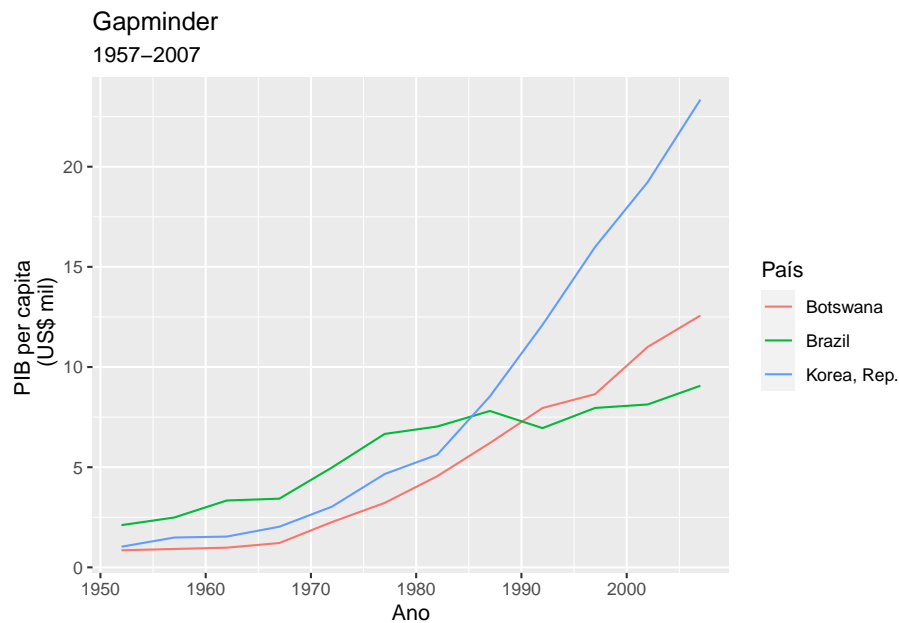
A visualização pode ser utilizada tanto para conhecer os dados com que iremos trabalhar, quanto para comunicar com ao público aquilo que descobrimos. Para tanto, o `tidyverse` fornece o pacote `ggplot2`, uma ferramenta poderossíma!

Ele permite construir desde gráficos simples até visualizações extremamente elaboradas, onde diversas variáveis são mapeadas em diferentes atributos de um gráfico. Somado a outras ferramentas, o `ggplot2` também pode ser utilizado para a construção de mapas.

3.1 Gráfico de linha

Gráficos de linhas são muito utilizados para mostrar tendências ao longo do tempo. Utilizando o dataset `gapminderCountries` que contruímos anteriormente, poderemos ver de que forma o PIB per capita de Brasil, Botsuana e Coreia do Sul se comportaram ao longo do tempo.

```
gapminderCountries %>%  
  ggplot() +  
  geom_line(aes(x = year,  
                y = gdpPercap/103,  
                color = country)) +  
  labs(title = "Gapminder",  
        subtitle = "1957-2007",  
        color = "País") +  
  xlab("Ano") +  
  ylab("PIB per capita \n (US$ mil)")
```

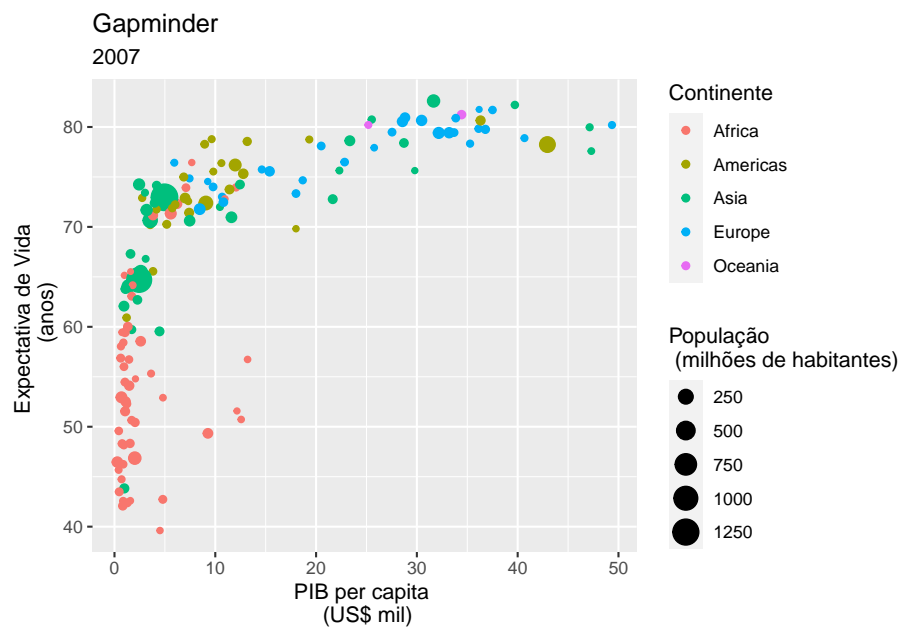
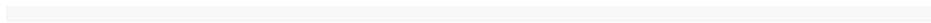


3.2 Gráfico de dispersão

Gráficos de dispersão permitem visualizar a relação entre duas variáveis numéricas, o que nos permite visualizar possíveis correlações. Utilizaremos o dataset `gapminder2007` para ver de que forma o PIB per capita está relacionado à expectativa de vida nos diferentes países do mundo em 2007.

Desta vez poderemos mostrar de que forma o `ggplot2` pode mapear diferentes atributos das observações em um mesmo gráfico. Em nosso exemplo, cada ponto representa um país, cada cor um continente, e o tamanho dos pontos é dado por sua população.

```
gapminder2007 %>%
  ggplot() +
  geom_point(aes(x = gdpPercap/103,
                 y = lifeExp,
                 color = continent,
                 size = pop/106)) +
  labs(title = "Gapminder",
       subtitle = "2007",
       color = "Continente",
       size = "População \n (milhões de habitantes)") +
  xlab("PIB per capita \n (US$ mil)") +
  ylab("Expectativa de Vida \n (anos)")
```



Capítulo 4

Organização e Transformação

Capítulo 5

Modelagem

Após organizar nossos dados e obter as informações iniciais com as estatísticas descritivas, podemos dar mais um passo em nossa análise e buscar entender como as diferentes variáveis que temos a disposição se relacionam.

Para tanto, em economia e outras ciências sociais, partimos de modelos teóricos para a estimação de relações empíricas entre variáveis dependentes (variável resposta) e independentes (variáveis explicativas), através de técnicas de regressão.

O R possui ferramentas para estimar diferentes tipos de modelo. A seguir veremos como funções que nos permitem estimar modelos lineares simples e múltiplos, probit e logit.

5.1 Regressão Linear

A regressão linear assume que a relação entre a variável dependente e as variáveis independentes é uma função linear dos parâmetros.

5.1.1 Regressão Linear Simples

Em uma regressão linear simples, a variável dependente é função de apenas uma variável independente mais um termo de erro. Ou seja, o modelo de regressão linear simples pode ser representado como:

$$y = \beta_0 + \beta_1 x + u$$

Onde y é a variável dependente, x a variável independente, β_0 e β_1 são os parâmetros e u é o termo de erro. Para estimar os parâmetros de um modelo de

regressão linear, comumente utiliza-se o método dos mínimos quadrados ordinários. Em R, utilizaremos a função `lm` para estimar modelos lineares.

Para os exemplos, utilizaremos o pacote `wooldridge` que fornece os dataset usados como exemplo no livro “Introductory Econometrics: A Modern Approach, 7ed” do economista Prof^o Jeffrey Wooldridge.

```
# Carregando o pacote wooldridge  
library(wooldridge)
```

```
# Carregando o dataset wage1  
wage1 <- wooldridge::wage1
```

```
# Vendo a estrutura do dataset  
str(wage1)
```

```
## 'data.frame': 526 obs. of 24 variables:  
## $ wage : num 3.1 3.24 3 6 5.3 ...  
## $ educ : int 11 12 11 8 12 16 18 12 12 17 ...  
## $ exper : int 2 22 2 44 7 9 15 5 26 22 ...  
## $ tenure : int 0 2 0 28 2 8 7 3 4 21 ...  
## $ nonwhite: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ female : int 1 1 0 0 0 0 0 1 1 0 ...  
## $ married : int 0 1 0 1 1 1 0 0 0 1 ...  
## $ numdep : int 2 3 2 0 1 0 0 0 2 0 ...  
## $ smsa : int 1 1 0 1 0 1 1 1 1 1 ...  
## $ northcen: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ south : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ west : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ construc: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ ndurman : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ trcommpu: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ trade : int 0 0 1 0 0 0 1 0 1 0 ...  
## $ services: int 0 1 0 0 0 0 0 0 0 0 ...  
## $ profserv: int 0 0 0 0 0 1 0 0 0 0 ...  
## $ profocc : int 0 0 0 0 0 1 1 1 1 1 ...  
## $ clerocc : int 0 0 0 1 0 0 0 0 0 0 ...  
## $ servocc : int 0 1 0 0 0 0 0 0 0 0 ...  
## $ lwage : num 1.13 1.18 1.1 1.79 1.67 ...  
## $ expersq : int 4 484 4 1936 49 81 225 25 676 484 ...  
## $ tenursq : int 0 4 0 784 4 64 49 9 16 441 ...  
## - attr(*, "time.stamp")= chr "25 Jun 2011 23:03"
```

```
# Utilizando a função lm  
my_model <- lm(formula = wage ~ educ, data = wage1)
```

```
# Mostrando os resultados do modelo
```

```
summary(my_model)
```

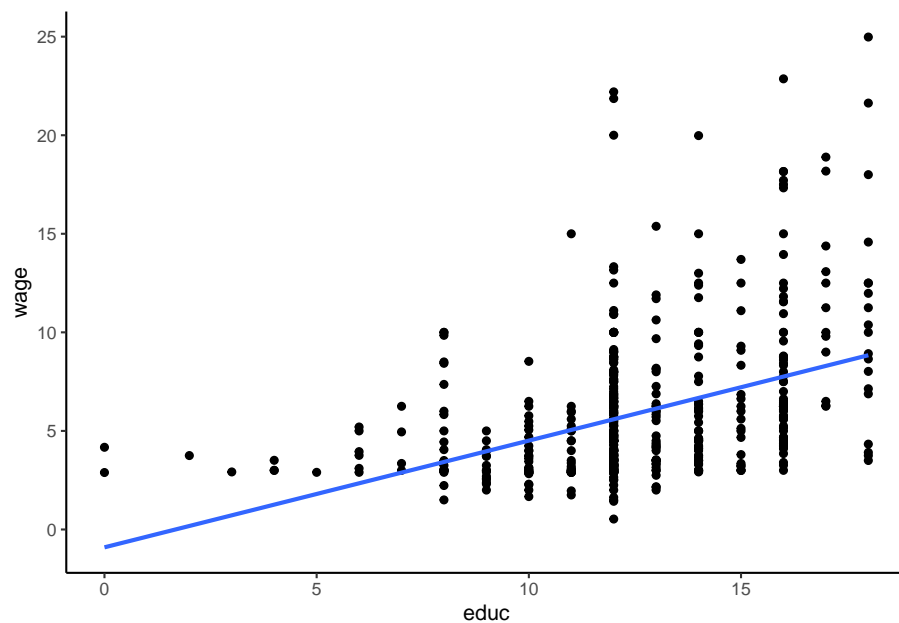
```
##
## Call:
## lm(formula = wage ~ educ, data = wage1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -5.3396 -2.1501 -0.9674  1.1921 16.6085
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.90485    0.68497  -1.321   0.187
## educ         0.54136    0.05325  10.167 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.378 on 524 degrees of freedom
## Multiple R-squared:  0.1648, Adjusted R-squared:  0.1632
## F-statistic: 103.4 on 1 and 524 DF,  p-value: < 2.2e-16
```

Utilizamos o dataset `wage1` para estimar o modelo econométrico

$$wage = \beta_0 + \beta_1 educ + u$$

Onde *wage* é medido em dólares por hora e *educ* são os anos de educação dos indivíduos da amostra. β_1 mede o efeito médio de um ano a mais de estudo no salário, tudo o mais constante. Podemos visualizar graficamente a relação.

```
wage1 %>%
  ggplot(aes(x = educ, y = wage)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  theme_classic()
```



A variável `my_model` contém as informações do nosso modelo. Podemos acessar informações de interesse como o valor do R^2 , os parâmetros e os resíduos por meio dos atributos de `my_model`.

```
# Coeficientes
```

```
my_model$coefficients
```

```
## (Intercept)      educ
## -0.9048516    0.5413593
```

```
# Informações sobre os coeficientes
```

```
summary(my_model)$coefficients
```

```
##              Estimate Std. Error  t value    Pr(>|t|)
## (Intercept) -0.9048516  0.68496782 -1.321013 1.870735e-01
## educ         0.5413593  0.05324804 10.166746 2.782599e-22
```

```
# R2
```

```
summary(my_model)$r.squared
```

```
## [1] 0.1647575
```

```
# R2 ajustado
summary(my_model)$adj.r.squared
```

```
## [1] 0.1631635
```

```
# Estatísticas descritivas dos resíduos
summary(my_model$residuals)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -5.3396 -2.1501 -0.9674  0.0000  1.1921 16.6085
```

5.1.2 Regressão Linear Múltipla

Em um modelo de regressão linear múltiplo, a variável dependente é função de múltiplas variáveis independentes. Ou seja,

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + u$$

Novamente utilizaremos o dataset `wage1`. Desta vez, para estimar o modelo

$$wage = \beta_0 + \beta_1 educ + \beta_2 exper + u$$

```
# Criando um novo modelo
my_model2 <- lm(wage ~ educ + exper, data = wage1)
```

```
# Mostrando resultados da regressão
summary(my_model2)
```

```
##
## Call:
## lm(formula = wage ~ educ + exper, data = wage1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -5.5532 -1.9801 -0.7071  1.2030 15.8370
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.39054    0.76657  -4.423 1.18e-05 ***
## educ           0.64427    0.05381  11.974 < 2e-16 ***
## exper          0.07010    0.01098   6.385 3.78e-10 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.257 on 523 degrees of freedom
## Multiple R-squared:  0.2252, Adjusted R-squared:  0.2222
## F-statistic: 75.99 on 2 and 523 DF,  p-value: < 2.2e-16
```

Podemos realizar utilizar a notação da fórmula de forma a alterar a especificação do modelo, por exemplo, incluir um termo quadrático, ou estimar um modelo sem intercepto.

```
# Estimando o modelo anterior sem intercepto
lm(wage ~ educ + exper - 1, data = wage1)
```

```
##
## Call:
## lm(formula = wage ~ educ + exper - 1, data = wage1)
##
## Coefficients:
##      educ      exper
## 0.41705  0.04544
```

Vamos utilizar esses dados para estimar uma equação minceriana do tipo

$$\ln wage = \beta_0 + \beta_1 educ + \beta_2 exper + \beta_3 exper^2 + \gamma'x + u$$

Onde *wage*, é o salário em dólares por hora, *educ* são os anos de estudo, *exper* os anos de experiência profissional e *x* um vetor de características observáveis, como raça, gênero e status marital. Podemos estimar os parâmetros utilizando R da seguinte forma.

```
# Equação Minceriana
wage1 %>%
  lm(log(wage) ~ educ + exper + I(exper^2) + nonwhite + female + married, data = .) %>%
  summary()
```

```
##
## Call:
## lm(formula = log(wage) ~ educ + exper + I(exper^2) + nonwhite +
##      female + married, data = .)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.79493 -0.27927 -0.01887  0.26057  1.27372
```

```
##
## Coefficients:
##           Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.3954849  0.1030918   3.836  0.00014 ***
## educ         0.0826774  0.0070297  11.761 < 2e-16 ***
## exper        0.0358161  0.0052508   6.821 2.52e-11 ***
## I(exper^2)   -0.0006334  0.0001131  -5.600 3.48e-08 ***
## nonwhite     -0.0147385  0.0597847  -0.247  0.80537
## female      -0.3293054  0.0367083  -8.971 < 2e-16 ***
## married      0.0638034  0.0422190   1.511  0.13133
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4133 on 519 degrees of freedom
## Multiple R-squared:  0.4024, Adjusted R-squared:  0.3955
## F-statistic: 58.24 on 6 and 519 DF, p-value: < 2.2e-16
```

5.2 Logit

Usaremos o dataset `cps91` do pacote `wooldridge` para exemplificar o uso da função `glm` que permite estimar modelos logit e probit. O exemplo consiste de um modelo para explicar a participação das mulheres casadas na força de trabalho.

```
# Lendo o dataset
cps91 <- wooldridge::cps91
```

```
# Estimando modelo Logit
cps91 %>%
  glm(inlf ~ nwifeinc + educ + exper + I(exper^2) +
      age + kidlt6 + kidge6,
      family = binomial(link = "logit"),
      data = .) %>%
  summary()
```

```
##
## Call:
## glm(formula = inlf ~ nwifeinc + educ + exper + I(exper^2) + age +
##      kidlt6 + kidge6, family = binomial(link = "logit"), data = .)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.1504  -1.1642   0.7221   0.9986   2.2435
##
```

```
## Coefficients:
##           Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.4284265  8.5368907  -0.284  0.77606
## nwifeinc    -0.0150524  0.0011200 -13.440 < 2e-16 ***
## educ        -0.0899159  1.4241121  -0.063  0.94966
## exper       -0.2466021  1.4250709  -0.173  0.86262
## I(exper^2)   -0.0009311  0.0002833  -3.286  0.00102 **
## age          0.2574374  1.4240344   0.181  0.85654
## kidlt6       -0.7874204  0.0852829  -9.233 < 2e-16 ***
## kidge6        0.0608807  0.0779442   0.781  0.43476
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 7653.5  on 5633  degrees of freedom
## Residual deviance: 7075.2  on 5626  degrees of freedom
## AIC: 7091.2
##
## Number of Fisher Scoring iterations: 4
```

5.3 Probit

```
# Estimando modelo Probit
cps91 %>%
  glm(inlf ~ nwifeinc + educ + exper + I(exper^2) +
      age + kidlt6 + kidge6,
      family = binomial(link = "probit"),
      data = .) %>%
  summary()

##
## Call:
## glm(formula = inlf ~ nwifeinc + educ + exper + I(exper^2) + age +
##      kidlt6 + kidge6, family = binomial(link = "probit"), data = .)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.1969  -1.1654   0.7229   1.0024   2.2976
##
## Coefficients:
##           Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.4892097  5.3413422  -0.279  0.78039
```

```

## nwifeinc      -0.0091475  0.0006762 -13.528 < 2e-16 ***
## educ          -0.0626141  0.8909676  -0.070  0.94397
## exper         -0.1571615  0.8915253  -0.176  0.86007
## I(exper^2)    -0.0005574  0.0001719  -3.243  0.00118 **
## age           0.1631290  0.8909229   0.183  0.85472
## kidlt6        -0.4810831  0.0516287  -9.318 < 2e-16 ***
## kidge6         0.0409159  0.0472457   0.866  0.38648
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 7653.5  on 5633  degrees of freedom
## Residual deviance: 7076.8  on 5626  degrees of freedom
## AIC: 7092.8
##
## Number of Fisher Scoring iterations: 4

```


Capítulo 6

Dicas para Continuar os Estudos