

MPS814 - Tópicos Especiais em Epidemiologia

R para epidemiologia

Rodrigo Citton Padilha dos Reis
rodrigocpdosreis@gmail.com

UNIVERSIDADE FEDERAL DE MINAS GERAIS
FACULDADE DE MEDICINA
PROGRAMA DE PÓS-GRADUAÇÃO EM SAÚDE PÚBLICA

Belo Horizonte
Junho de 2017

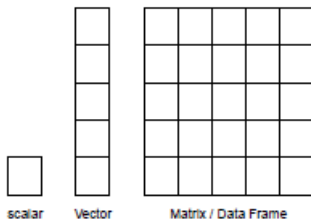
Data Frames

O que são matrizes e dataframes?

- ▶ Até agora, você deve estar confortável com escalares e vetores.
- ▶ No entanto, você notou que nenhum destes tipos de objetos são apropriados para o armazenamento de **grandes quantidades de dados**, tais como os resultados de **uma pesquisa** ou de **um experimento**.
- ▶ Felizmente, o R tem dois tipos de objetos que representam grandes estruturas de dados de forma muito melhor: **matrizes** e **dataframes**.
- ▶ Matrizes e dataframes são muito semelhantes a uma planilha em Excel ou um arquivo de dados em SPSS (**tabela de dados brutos**).

O que são matrizes e dataframes?

- ▶ Cada matriz ou dataframe contém **linhas** e **colunas**.
- ▶ Assim, enquanto um vetor tem uma dimensão (seu comprimento), as matrizes e dataframes **têm duas dimensões** - (largura e altura).
- ▶ Você pode pensar em uma matriz ou dataframe como **uma combinação** de **p** vetores, em que cada vetor tenha comprimento igual a **n**.



O que são matrizes e dataframes?

- ▶ A principal diferença entre matrizes e dataframes está no fato que matrizes podem conter colunas de apenas uma classe, enquanto que dataframes podem conter colunas de diferentes classes.
- ▶ Porque dataframes são mais flexíveis, a maioria dos conjuntos de dados do mundo real, tais como pesquisas contendo dados numéricos (por exemplo, idade, tempo de gestação) e caracteres (por exemplo, sexo, filme favorito), serão armazenados como dataframes no R.

Criando matrizes e dataframes

- ▶ Existem diversas maneiras de criarmos objetos matrizes e dataframes no R.
- ▶ Por serem uma combinação de vetores, cada função terá um ou mais vetores como inputs e retornará uma matriz ou um dataframe.

Função	Descrição
<code>cbind()</code>	Combina vetores em <i>colunas</i> em uma matriz/dataframe.
<code>rbind()</code>	Combina vetores em <i>linhas</i> em uma matriz/dataframe.
<code>matrix()</code>	Cria uma matriz com o número desejado de linhas e colunas a partir de um único vetor.
<code>data.frame()</code>	Combina vetores como colunas em um dataframe.

Criando matrizes e dataframes

- ▶ Vamos criar um dataframe simples utilizando a função `data.frame()`.

```
# Criar um dataframe com as colunas col.1,  
# col.2 e col.3  
data.frame("id" = c(1, 2, 3, 4, 5),  
           "sexo" = c("m", "m", "m", "f", "f"),  
           "idade" = c(99, 46, 23, 54, 23)  
           )
```

```
##      id sexo idade  
## 1    1    m    99  
## 2    2    m    46  
## 3    3    m    23  
## 4    4    f    54  
## 5    5    f    23
```

Criando matrizes e dataframes

- ▶ Note que a função `data.frame()` apenas combina diferentes vetores em um mesmo objeto.
- ▶ Se quisermos armazenar o resultado da função `data.frame()` devemos criar um novo objeto utilizando o operador atribui (`<-`).

Funções de matrizes e dataframes

- ▶ O R possui várias funções para visualizar matrizes e dataframes e retornar informações a respeito destes.

Função	Descrição
<code>head()</code> <code>tail()</code> <code>View()</code>	Apresenta as primeiras linhas no console. Apresenta as últimas linhas no console. Abre uma nova janela com todo o objeto.
<code>dim()</code> <code>ncol()</code> , <code>nrow()</code>	Conta o número de linhas e colunas. Conta o número de colunas (ou linhas).
<code>colnames()</code> , <code>rownames()</code> <code>names()</code>	Mostra os nomes das colunas (ou das linhas). Mostra os nomes das colunas (apenas para dataframes).
<code>str()</code> <code>summary()</code>	Mostra a estrutura do dataframe. Apresenta estatísticas resumo.

Sua vez!

1. Instale o pacote survey.
2. Carregue o pacote survey.
3. Carregue o dataframe nhanes com a função data() (**consulte o help da função 'data()' e do dataframe 'nhanes'**).
4. Utilize as funções apresentadas anteriormente no dataframe nhanes.
5. Qual a classe da variável agecat do dataframe nhanes?
6. O resumo da variável race está correto? Se não, o que podemos fazer? (**Consulte o help da função factor**)
7. Repita os exercícios de 1-4 considerando o pacote Epi e dataframe births

Acessando colunas de um dataframe com \$

- ▶ Para **acessar uma coluna específica de um dataframe pelo nome**, usaremos o operador \$ na forma

```
df$nomecoluna
```

em que df é o dataframe e nomecoluna é o nome da coluna que estamos interessados.

- ▶ Esta operação retornará a coluna que desejamos **como um vetor**.

```
# Retorna a coluna peso ao nascer bweight
births100$bweight
```

```
##      [1] 2974 3270 2620 3751 3200 3673 3628 3773 3960 3405 4020 2724 3001 3039
##     [15] 3662 3035 3351 3804 3573 3283 2894 1203 3306 3753 2844 3585 3798 3164
##     [29] 3739 1780 4022 3942 2887 2391 3911 3509 3566 3652 3279 3007 3053 3503
##     [43] 3120 3743 3592 3184 3234 2581 3305 3678 3542 2148 3774 3079 3465 2887
##     [57] 4501 3375 3886 2849 2002 2213 2797 3303 3296 2921 2929 3385 1946 3354
##     [71] 3189 3392 2696 2597 3409 3190 3571 4512 2215 3315 3341 3451 3338 2507
##     [85] 3316 4141 4071 2893 3064 3603 3554 4027 2418 3092 2671 3430 3244 3259
##     [99] 3942 3576
```

Acessando colunas de um dataframe com \$

- ▶ Veja que podemos utilizar todas as funções de vetores que aprendemos combinadas com o operador \$.

```
# Qual a média do peso ao nascer?
```

```
mean(births100$bweight)
```

```
## [1] 3256.47
```

```
# Tabela de frequências da variável sexo
```

```
table(births100$sex)
```

```
##
```

```
##  1  2
```

```
## 60 40
```

Acrescentando novas colunas a um dataframe

- ▶ Podemos acrescentar novas colunas a um dataframe utilizando os operadores \$ e <-.
- ▶ Para isto, basta usar a notação df\$nomecoluna e atribuir um novo vetor a este.
 - ▶ Vamos criar a variável **peso ao nascer em quilogramas** a partir da variável bweight do dataframe births100:

```
# Acrescentando uma nova coluna de peso em kg  
# bweight / 1000  
births100$bweight.kg <- births100$bweight / 1000  
births100$bweight.kg
```

Acrescentando novas colunas a um dataframe

```
##      [1] 2.974 3.270 2.620 3.751 3.200 3.673 3.628 3.773 3.960 3.405 4.020
##     [12] 2.724 3.001 3.039 3.662 3.035 3.351 3.804 3.573 3.283 2.894 1.203
##     [23] 3.306 3.753 2.844 3.585 3.798 3.164 3.739 1.780 4.022 3.942 2.887
##     [34] 2.391 3.911 3.509 3.566 3.652 3.279 3.007 3.053 3.503 3.120 3.743
##     [45] 3.592 3.184 3.234 2.581 3.305 3.678 3.542 2.148 3.774 3.079 3.465
##     [56] 2.887 4.501 3.375 3.886 2.849 2.002 2.213 2.797 3.303 3.296 2.921
##     [67] 2.929 3.385 1.946 3.354 3.189 3.392 2.696 2.597 3.409 3.190 3.571
##     [78] 4.512 2.215 3.315 3.341 3.451 3.338 2.507 3.316 4.141 4.071 2.893
##     [89] 3.064 3.603 3.554 4.027 2.418 3.092 2.671 3.430 3.244 3.259 3.942
##    [100] 3.576
```

Acrescentando novas colunas a um dataframe

- ▶ Podemos adicionar novas colunas a um dataframe com qualquer informação que desejarmos.

```
# Acrescentando uma nova coluna de baixo peso  
births100$baixo.peso <- births100$bweight.kg < 2.5  
table(births100$baixo.peso)
```

```
##  
## FALSE  TRUE  
##    91     9
```

Mudando os nomes das colunas de um dataframe

- ▶ Para mudarmos os nomes de uma ou mais colunas de um dataframe utilizaremos a **seguinte combinação**:
 - ▶ Função `names()`, indexação e reatribuição.

```
# Nomes das colunas do dataframe births100
names(births100)
```

```
## [1] "id"          "bweight"     "lowbw"       "gestwks"     "preterm"
## [6] "matage"      "hyp"         "sex"         "bweight.kg"  "baixo.peso"
```

```
# Alterando o nome da segunda coluna
# para peso.nascer
names(births100)[2] <- "peso.nascer"
# Novos nomes das colunas do dataframe births100
names(births100)
```

```
## [1] "id"          "peso.nascer" "lowbw"       "gestwks"     "preterm"
## [6] "matage"      "hyp"         "sex"         "bweight.kg"  "baixo.peso"
```


Indexando dataframes com colchetes

- ▶ Assim como foi visto para os vetores, podemos acessar dados específicos em um dataframe usando colchetes.
- ▶ Mas agora, ao invés de utilizar **um vetor indexador**, utilizaremos **dois vetores indexadores**: um para linhas e outro para colunas.
- ▶ Para isto, usaremos a notação

`dados[linha, coluna],`

em que `dados` é o nome do dataframe e `linha` e `coluna` são vetores de inteiros.

```
# Linhas de 1 a 5 e coluna 1 de births100  
births100[1:5, 1]
```

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

```
# Linhas de 1 a 3 e colunas 1 e 3 de births100  
births100[1:3, c(1,3)]
```

```
##   id lowbw  
## 1  1     0  
## 2  2     0  
## 3  3     0
```

Indexando dataframes com colchetes

Obtendo uma linha (ou coluna) inteira

```
# Linha 1 de births100
```

```
births100[1, ]
```

```
##   id peso.nascer lowbw gestwks preterm matage hyp sex bweight.kg
## 1   1         2974     0   38.52         0   34  0   2         2.974
##   baixo.peso
## 1         FALSE
```

```
# Coluna 2 de births100
```

```
births100[, 2]
```

```
##   [1] 2974 3270 2620 3751 3200 3673 3628 3773 3960 3405 4020 2724 3001 3039
##   [15] 3662 3035 3351 3804 3573 3283 2894 1203 3306 3753 2844 3585 3798 3164
##   [29] 3739 1780 4022 3942 2887 2391 3911 3509 3566 3652 3279 3007 3053 3503
##   [43] 3120 3743 3592 3184 3234 2581 3305 3678 3542 2148 3774 3079 3465 2887
##   [57] 4501 3375 3886 2849 2002 2213 2797 3303 3296 2921 2929 3385 1946 3354
##   [71] 3189 3392 2696 2597 3409 3190 3571 4512 2215 3315 3341 3451 3338 2507
##   [85] 3316 4141 4071 2893 3064 3603 3554 4027 2418 3092 2671 3430 3244 3259
##   [99] 3942 3576
```


subset()

- ▶ De forma alternativa, podemos obter subconjuntos de um dataframe utilizando a função `subset()`.

`subset(x, subset, select)`

x

Os dados (usualmente um dataframe).

subset

Um vetor de lógicos indicando quais linhas devem ser selecionadas.

select

Um vetor opcional com nomes das colunas que devem ser selecionadas.

subset()

```
# Subconjunto contendo apenas
# meninas com baixo peso e
# tempo de gestação menor
# que 40 semanas
subset(births100,
  subset = sex == 2 &
    peso.nascer < 2500 &
    gestwks < 40
)
```

```
##      id peso.nascer lowbw gestwks preterm matage hyp sex bweight.kg
## 52 52      2148      1  38.44      0     29  0  2      2.148
## 61 61      2002      1  36.48      1     37  1  2      2.002
## 79 79      2215      1  38.37      0     35  0  2      2.215
##      baixo.peso
## 52      TRUE
## 61      TRUE
## 79      TRUE
```

Sua vez!

1. Instale o pacote Epi.
2. Carregue o pacote Epi.
3. Carregue o dataframe diet.
4. Conheça o dataframe diet.
5. Apresente os dados da linha 35.
6. Apresente os dados das linhas 35, 53 e 82 e colunas y, job e chd.
7. Crie uma nova variável imc no dataframe diet a partir das variáveis height e weight.
8. Crie um objeto com um subconjunto do dataframe diet com indivíduos que atendem aos critérios `energy > 28`, `imc >= 26`.
9. Apresente as principais estatísticas resumo por grupo `chd = 0` e `chd = 1`.

Importando e exportando dados no R

Conjuntos de dados

- ▶ A maioria das tarefas de análise de dados utiliza como fonte **conjuntos de dados** armazenados em um **formato tabular**.
- ▶ Uma **tabela de dados** consiste em uma estrutura bidimensional:
 - ▶ **As linhas** representam **observações** (unidades/indivíduos) de um fenômeno.
 - ▶ **As colunas** contém informação obtida para cada observação, ou seja, representam **as variáveis**.
- ▶ No R, o objeto utilizado para armazenar estas estruturas de dados é o **dataframe**.
- ▶ Veremos exemplos de como **importar dados em diferentes formatos** no R em um objeto dataframe.

Conjuntos de dados internos do R

- ▶ Qualquer instalação do R inclui muitos conjuntos de dados.
- ▶ A lista destes conjuntos de dados pode ser acessada com a função `data()`.
- ▶ Cada novo pacote que instalarmos também deverá conter novos conjuntos de dados para propósitos de ilustração (exemplos de como as funções são utilizadas).

```
data() # lista dos diversos conjuntos de dados do pacote base R
data(package = "survival") # lista os conj. de dados de um pacote específico
data(package = .packages(all.available = TRUE)) # lista todos os conj. de dados disponíveis
```

- ▶ Estes conjuntos de dados podem ser usados/carregados com a função `data()`:

```
data(esoph)
head(esoph, n = 4)
```

```
##   agegp   alcgp   tobgp ncases ncontrols
## 1 25-34 0-39g/day 0-9g/day     0         40
## 2 25-34 0-39g/day 10-19      0         10
## 3 25-34 0-39g/day 20-29      0          6
## 4 25-34 0-39g/day 30+        0          5
```

O formato RData

- ▶ Qualquer dataframe (qualquer objeto do R) pode ser salvo em um arquivo **RData**.

```
dados <- data.frame("sexo" = c("m", "m", "m", "f", "f"),
                    "idade" = c(99, 46, 23, 54, 23))
save(dados, file = "Exemplo1.RData")
```

- ▶ Posteriormente estes dados podem ser carregados no R utilizando a função `load()`:

```
rm(dados) # removendo o objeto dados
dados # apenas checando que o objeto dados não mais existe
```

```
## Error in eval(expr, envir, enclos): objeto 'dados' não encontrado
```

```
load("Exemplo1.RData") # carregando os dados do arquivo
head(dados, n = 5)
```

```
##   sexo idade
## 1    m    99
## 2    m    46
## 3    m    23
## 4    f    54
## 5    f    23
```

Importando dados de um arquivo texto

- ▶ Arquivos de texto são uma forma frequente de armazenar e compartilhar conjuntos de dados.
- ▶ Linhas do arquivo normalmente correspondem a linhas (indivíduos) do conjunto de dados.
- ▶ Os valores nas colunas (variáveis) são armazenados numa linha única separados por algum caractere especial.
- ▶ Os arquivos de texto são muitas vezes a maneira mais fácil de importação de dados para o R (**PORÉM, ESTA NÃO É A ÚNICA FORMA!**).
- ▶ A maneira mais comum de importarmos um conjunto de dados no formato texto para o R é através da função `read.table()`.

Importando dados de um arquivo texto

- ▶ Por **default** a função `read.table()` assume que o arquivo é delimitado por espaços:

Arquivo "Ex2.txt"

1	"f"	221.7	68.1	143
2	"m"	195	30.5	122
3	"f"	235.8	NA	113
4	"m"	234.4	31.2	126

Carregando os dados em um dataframe:

```
dadosEx2 <- read.table("Ex2.txt")  
dadosEx2
```

Importando dados de um arquivo texto

```
##      V1 V2      V3      V4      V5
## 1    1  1  f 221.7 68.1 143
## 2    2  2  m 195.0 30.5 122
## 3    3  3  f 235.8   NA 113
## 4    4  4  m 234.4 31.2 126
```

Importando dados de um arquivo texto

- ▶ Dados com nomes de colunas (variáveis):

Arquivo "Ex3.txt"

```
"id" "sexo" "col" "hdl" "pas"  
1 "f" 221.7 68.1 143  
2 "m" 195 30.5 122  
3 "f" 235.8 NA 113  
4 "m" 234.4 31.2 126
```

Carregando os dados em um dataframe:

```
dadosEx3 <- read.table("Ex3.txt", header = TRUE)  
dadosEx3
```

```
##   id sexo   col  hdl pas  
## 1  1    f 221.7 68.1 143  
## 2  2    m 195.0 30.5 122  
## 3  3    f 235.8   NA 113  
## 4  4    m 234.4 31.2 126
```

Importando dados de um arquivo texto

- Dados com símbolo de decimal diferente de .:

Arquivo "Ex4.txt"

```
"id" "sexo" "col" "hdl" "pas"
1 "f" 221,7 68,1 143
2 "m" 195 30,5 122
3 "f" 235,8 NA 113
4 "m" 234,4 31,2 126
```

Carregando os dados em um dataframe:

```
dadosEx4 <- read.table("Ex4.txt", header = TRUE, dec = ",")
dadosEx4
```

```
##   id sexo   col  hdl pas
## 1  1    f 221.7 68.1 143
## 2  2    m 195.0 30.5 122
## 3  3    f 235.8   NA 113
## 4  4    m 234.4 31.2 126
```

Importando dados de um arquivo texto

- Dados com caractere especial de separação diferente de espaços:

Arquivo "Ex5.txt"

```
"id","sexo","col","hdl","pas"  
1,"f",221.7,68.1,143  
2,"m",195,30.5,122  
3,"f",235.8,NA,113  
4,"m",234.4,31.2,126
```

Carregando os dados em um dataframe:

```
dadosEx5 <- read.table("Ex5.txt", header = TRUE, sep = ",")  
dadosEx5
```

```
##   id sexo   col  hdl pas  
## 1  1    f 221.7 68.1 143  
## 2  2    m 195.0 30.5 122  
## 3  3    f 235.8  NA  113  
## 4  4    m 234.4 31.2 126
```


Importando dados de um arquivo texto

- ▶ Dados com símbolo de decimal diferente de `.` e com caractere especial de separação diferente de espaços:

Arquivo "Ex6.txt"

```
"id";"sexo";"col";"hdl";"pas"  
1;"f";221,7;68,1;143  
2;"m";195;30,5;122  
3;"f";235,8;NA;113  
4;"m";234,4;31,2;126
```

Carregando os dados em um dataframe:

```
dadosEx6 <- read.table("Ex6.txt", header = TRUE, sep = ";", dec = ",")  
dadosEx6
```

Importando dados de um arquivo texto

```
##   id sexo   col  hdl pas
## 1  1    f 221.7 68.1 143
## 2  2    m 195.0 30.5 122
## 3  3    f 235.8   NA 113
## 4  4    m 234.4 31.2 126
```

Importando dados de um arquivo texto

- ▶ Dados com código diferente de NA para valores ausentes (**missing data**):

Arquivo "Ex7.txt"

	"id"	"sexo"	"col"	"hdl"	"pas"
1	"f"	221.7	68.1	143	
2	"m"	195	30.5	122	
3	"f"	235.8	9999	113	
4	"m"	234.4	31.2	126	

Carregando os dados em um dataframe:

```
dadosEx7 <- read.table("Ex7.txt", header = TRUE, na.strings = "9999")
dadosEx7
```

Importando dados de um arquivo texto

```
##   id sexo   col  hdl pas
## 1  1    f 221.7 68.1 143
## 2  2    m 195.0 30.5 122
## 3  3    f 235.8   NA 113
## 4  4    m 234.4 31.2 126
```

Arquivos CSV

- ▶ Arquivos de dados no **formato CSV** (*comma-separted values*) utilizam os caracteres especiais **vírgula (,)** ou **ponto e vírgula (;)** para delimitar as colunas (variáveis) do conjunto de dados.
- ▶ Para importar dados neste formato para o R, podemos utilizar a função `read.table()` especificando de modo adequado o argumento `sep`, ou utilizar as seguintes funções:
 - ▶ `read.csv()` para arquivos com valores separados por **vírgula**.
 - ▶ `read.csv2()` para arquivos com valores separados por **ponto e vírgula**.
- ▶ Os argumentos destas funções são idênticos aos argumentos da função `read.table()`.

Importando arquivos CSV

- ▶ Valores separados por **vírgula**:

Arquivo "Ex8.csv"

```
"id","sexo","col","hdl","pas"  
1,"f",221.7,68.1,143  
2,"m",195,30.5,122  
3,"f",235.8,NA,113  
4,"m",234.4,31.2,126
```

Carregando os dados em um dataframe:

```
dadosEx8 <- read.csv("Ex8.csv", header = TRUE)  
dadosEx8
```

```
##   id sexo   col  hdl pas  
## 1  1    f 221.7 68.1 143  
## 2  2    m 195.0 30.5 122  
## 3  3    f 235.8   NA 113  
## 4  4    m 234.4 31.2 126
```

Importando arquivos CSV

- ▶ Valores separados por **ponto e vírgula**:

Arquivo "Ex9.csv"

```
"id";"sexo";"col";"hdl";"pas"  
1;"f";221.7;68.1;143  
2;"m";195;30.5;122  
3;"f";235.8;NA;113  
4;"m";234.4;31.2;126
```

Carregando os dados em um dataframe:

```
dadosEx9 <- read.csv2("Ex9.csv", header = TRUE)  
dadosEx9
```

```
##   id sexo   col  hdl pas  
## 1  1    f 221.7 68.1 143  
## 2  2    m 195.0 30.5 122  
## 3  3    f 235.8   NA 113  
## 4  4    m 234.4 31.2 126
```

Arquivos de campos com largura fixa

- ▶ Alguns arquivos armazenam dados em campos (variáveis) com largura fixa (*fixed width file*) pré-estabelecida.
 - ▶ Um exemplo desse tipo acontece com os dados da **Pesquisa Nacional por Amstras de Domicílios (PNAD)**.
- ▶ Para importar arquivos neste formato, o R possui a função `read.fwf()`.
 - ▶ Um dos principais argumentos desta função é o argumento `width`, um vetor numérico especificando para a função `read.fwf()` os tamanhos dos campos.
- ▶ A melhor maneira de trabalharmos com um arquivo deste tipo é em conjunto com um dicionário de dados.

Arquivos de campos com largura fixa

```
# Carregando arquivo dicionário de dados
dic <- read.csv2("dicPNAD2011.csv", header = TRUE)
head(dic)
```

```
##      inicio   cod tamanho      desc
## 1         1 V0101        4    ANO DA PESQUISA
## 2         5    UF         2           UF
## 3         5 V0102        8 NUMERO DE CONTROLE
## 4        13 V0103        3    NUMERO DE SERIE
## 5        16 V0104        2 TIPO DE ENTREVISTA
## 6        18 V0105        2 TOTAL DE MORADORES
```

```
# Carregando arquivo de dados da PNAD
# utilizando o vetor tamanho do dataframe
# dic para especificar o argumento
# width
dadosPNAD <- read.fwf("pnad2011.txt", width = dic$tamanho)
names(dadosPNAD) <- dic$cod
dadosPNAD[1:4,1:10]
```

Arquivos de campos com largura fixa

##	V0101	UF	V0102	V0103	V0104	V0105	V0106	V0201	V0202	V0203
## 1	2011	11	1500	101	4	3	12	2	1	0
## 2	2011	11	1500	201	4	4	12	1	1	0
## 3	2011	11	1500	303	NA	NA	NA	NA	NA	NA
## 4	2011	11	1500	401	2	2	12	2	1	0

Arquivos armazenados na web

- ▶ O principal argumento das funções que acabamos de ver é o argumento `file`.
- ▶ Este argumento especifica o caminho do arquivo de dados.
- ▶ Para carregarmos dados que estejam armazenados em algum endereço da web, basta especificar o argumento `file` com este endereço.

```
# Dados em
# http://web1.sph.emory.edu/dkleinb/allDatasets/datasets/evans.dat
dadosWEB <- read.table(file =
  "http://web1.sph.emory.edu/dkleinb/allDatasets/datasets/evans.dat")
head(dadosWEB)
```

```
##   V1 V2 V3 V4  V5 V6 V7  V8  V9 V10 V11 V12
## 1 21  0  0 56 270  0  0  80 138  0  0  0
## 2 31  0  0 43 159  1  0  74 128  0  0  0
## 3 51  1  1 56 201  1  1 112 164  1  1 201
## 4 71  0  1 64 179  1  0 100 200  1  1 179
## 5 74  0  0 49 243  1  0  82 145  0  0  0
## 6 91  0  0 46 252  1  0  88 142  0  0  0
```

Resumo dos formatos de texto

`read.table(file, header, sep, dec)`

`file`

O caminho do arquivo do documento (certifique-se de entrar com uma string com aspas!) ou um link HTML para um arquivo.

`header`

O valor lógico (TRUE ou FALSE) indicando se o arquivo possui uma linha de cabeçalho (nomes de variáveis na primeira linha do arquivo).

`sep`

Um caractere indicando como as colunas estão separadas.

Para arquivos separados por vírgula, use “,”, para arquivos delimitados por tab, use “\t”.

`dec`

Um caractere indicando o símbolo de decimal.

Resumo dos formatos de texto

► Algumas funções auxiliares:

- `getwd()`
- `setwd()`
- `file.path()`
- `merge()`

► Veja também:

- O pacote IBGEPesq do **IBGE** para leitura de dados em R de pesquisas do IBGE:

<http://www.ibge.gov.br/home/estatistica/populacao/trabalhoerendimento/pnad2014/microdados.shtm>

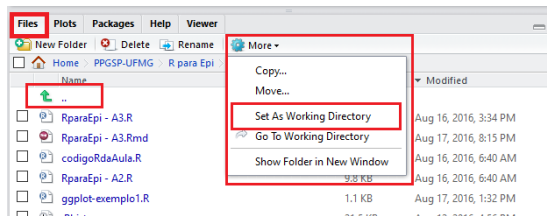
- As funcionalidades do RStudio para importar arquivos de dados



Environment is empty

Resumo dos formatos de texto

e para especificação do diretório de trabalho



Importando planilhas (Excel)

- ▶ Em alguns casos os dados estarão armazenados em uma planilha do Excel (**arquivos .xls ou .xlsx**).
- ▶ O pacote `readxl` possui a função `read_excel()` para importar arquivos neste formato.

```
# Instale o pacote readxl  
install.packages("readxl")
```

```
# Carregue o pacote readxl  
library(readxl)
```

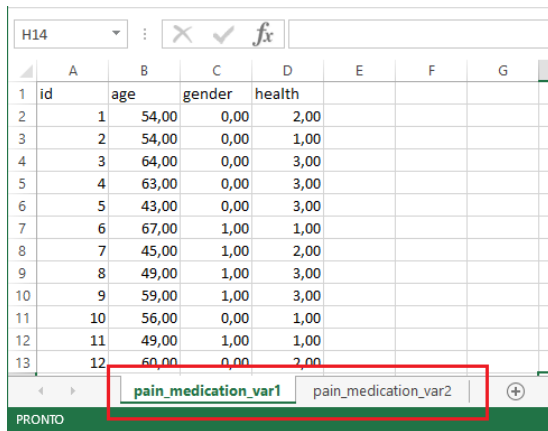
```
dadosEXCEL <- read_excel("pain_medication.xlsx", # caminho do arquivo  
                        sheet = 1) # número da planilha do arquivo  
head(dadosEXCEL, n = 4)
```

```
##   id age gender health  
## 1  1  54      0      2  
## 2  2  54      0      1  
## 3  3  64      0      3  
## 4  4  63      0      3
```

Importando planilhas (Excel)

- ▶ O argumento `sheet` é importante para especificar a planilha do arquivo que se deseja importar.
 - ▶ Pode ser especificado utilizando o **número** ou o **nome** da planilha.

Importando planilhas (Excel)



The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E	F	G
1	id	age	gender	health			
2	1	54,00	0,00	2,00			
3	2	54,00	0,00	1,00			
4	3	64,00	0,00	3,00			
5	4	63,00	0,00	3,00			
6	5	43,00	0,00	3,00			
7	6	67,00	1,00	1,00			
8	7	45,00	1,00	2,00			
9	8	49,00	1,00	3,00			
10	9	59,00	1,00	3,00			
11	10	56,00	0,00	1,00			
12	11	49,00	1,00	1,00			
13	12	60,00	0,00	2,00			

The bottom tab bar shows two sheets: 'pain_medication_var1' and 'pain_medication_var2'. The 'pain_medication_var2' tab is selected and highlighted with a red box.

```
dadosEXCEL2 <- read_excel("pain_medication.xlsx", # caminho do arquivo
                           sheet = "pain_medication_var2") # nome da planilha
head(dadosEXCEL2)
```

Importanto planilhas (Excel)

```
##      id treatment dosage status time
## 1  1      0      1      1  1.2
## 2  2      0      1      1  4.0
## 3  3      0      1      1  7.4
## 4  4      1      1      1  7.3
## 5  5      1      1      0  7.4
## 6  6      0      1      1  0.6
```

Importando arquivos “forasteiros”

- ▶ Ainda podemos encontrar dados em formatos bastante específicos, como SPSS (**arquivos .sav**) ou Stata (**arquivos .dta**).
- ▶ Para estes casos temos o pacote `foreign` que possui diversas funções para importar arquivos em diferentes formatos.

```
# Instale o pacote foreign  
install.packages("foreign")
```

```
# Carregue o pacote foreign  
library(foreign)
```

Importando arquivos “forasteiros”

SPSS:

```
dadosSPSS <- read.spss("dietstudy.sav", # caminho do arquivo
  to.data.frame = TRUE, # armazenar em objeto dataframe
  use.value.labels = TRUE, # transformar labels em factor
  use.missings = TRUE) # usar definições de missing
```

```
## re-encoding from UTF-8
```

```
head(dadosSPSS)
```

```
##   patid age gender tg0 tg1 tg2 tg3 tg4 wgt0 wgt1 wgt2 wgt3 wgt4
## 1     1  45   Male 180 148 106 113 100  198  196  193  188  192
## 2     2  56   Male 139  94 119  75  92  237  233  232  228  225
## 3     3  50   Male 152 185  86 149 118  233  231  229  228  226
## 4     4  46 Female 112 145 136 149  82  179  181  177  174  172
## 5     5  64   Male 156 104 157  79  97  219  217  215  213  214
## 6     6  49 Female 167 138  88 107 171  169  166  165  162  161
```

Importando arquivos “forasteiros”

Stata:

```
dadosStata <- read.dta("evans.dta")  
head(dadosStata)
```

##	id	chd	cat	age	chl	smk	ecg	dbp	sbp	hpt	cc	ch
## 1	21	0	0	56	270	0	0	80	138	0	0	0
## 2	31	0	0	43	159	1	0	74	128	0	0	0
## 3	51	1	1	56	201	1	1	112	164	1	1	201
## 4	71	0	1	64	179	1	0	100	200	1	1	179
## 5	74	0	0	49	243	1	0	82	145	0	0	0
## 6	91	0	0	46	252	1	0	88	142	0	0	0

Importando arquivos “forasteiros”

- ▶ **Stata**: a função `read.dta()` importa dados somente de arquivos oriundos de versões entre 5 e 12 do Stata.
- ▶ Para versões mais atuais, utilize o pacote `readstata13`.
- ▶ **Veja também**: pacote `haven`
(<https://github.com/tidyverse/haven>).

Exportando dados do R

- ▶ Para exportar dados (dataframes) do R em formatos específicos basta utilizar as funções:
 - ▶ `write.table()` para exportar arquivos no formato **.txt**.
 - ▶ `write.csv()` para exportar arquivos no formato **.csv** (valores separados por vírgula).
 - ▶ `write.csv2()` para exportar arquivos no formato **.csv** (valores separados por ponto e vírgula).
 - ▶ `write.dta()` do pacote `foreign` para exportar arquivos no formato **.dta**.

Banco de dados: usando MySQL do R

- ▶ Um **banco de dados** é uma coleção organizada de dados.
 - ▶ Os dados são organizados em modelos que tentam representar a realidade de uma forma que o processamento (adição, remoção e consulta de dados) se torna mais eficiente.
- ▶ O MySQL é um programa gerenciador de banco de dados que implementa a linguagem SQL, e podemos montar um servidor com vários bancos de dados, cada banco de dados com seu próprio *schema*.
 - ▶ SQL é uma linguagem estruturada para “falar” com o banco de dados, que serve para criar e acessar e alterar os dados que você quer.
- ▶ O pacote RMySQL permite que você acesse o MySQL do R.

```
# Instale o pacote RMySQL  
install.packages("RMySQL")
```

```
# Carregue o pacote RMySQL  
library(RMySQL)
```


RMySQL: um exemplo

```
# connecting to genome mysql database
ucsDb <- dbConnect(MySQL(),
                    user = "genome",
                    host = "genome-mysql.cse.ucsc.edu")

result <- dbGetQuery(ucsDb, "show databases;")
dbDisconnect(ucsDb)
head(result)
nrow(result)
```

RMySQL: um exemplo

```
# connect to db hg19 and list its tables
hg19 <- dbConnect(MySQL(),
                  user = "genome",
                  db = "hg19",
                  host = "genome-mysql.cse.ucsc.edu")

hg19Tbls <- dbListTables(hg19)
length(hg19Tbls)
hg19Tbls[1:15]

# get dim for specific tbl
hg19Flds <- dbListFields(hg19, "affyU133Plus2")
hg19Flds
```

RMySQL: um exemplo

```
# "nrow" of hg19
dbGetQuery(hg19,"select count(*) from affyU133Plus2")

# read from tbl
affyData <- dbReadTable(hg19, "affyU133Plus2")
head(affyData)

# select a subset
query <- dbSendQuery(hg19,"select * from affyU133Plus2 where misMatches between 1 and 10")
affyMis <- fetch(query)
quantile(affyMis$misMatches)

# fetch only 10 rows (limit query to top n rows)
affyMisSmall <- fetch(query,n=10)
dbClearResult(query)
dim(affyMisSmall)

# remember to disconnect when done!
dbDisconnect(hg19)
```

Sua vez!

1. Importe o conjunto de dados do arquivo **maternal-smoking.txt**.
 - 1.1 Tome conhecimento dos dados.
 - 1.2 Apresente as principais estatísticas resumo para as variáveis dos dados.
 - 1.3 Apresente as principais estatísticas resumo para a variável **Birthweight** estratificado por **Smoke** (consulte o help da função `by()`).
 - 1.4 Calcule a mediana do peso da mãe (**MotherWeight**) por estado civil (**Marital**).
 - 1.5 Exporte o conjunto de dados para o formato `.dta`.

Gráficos

Gráficos no R

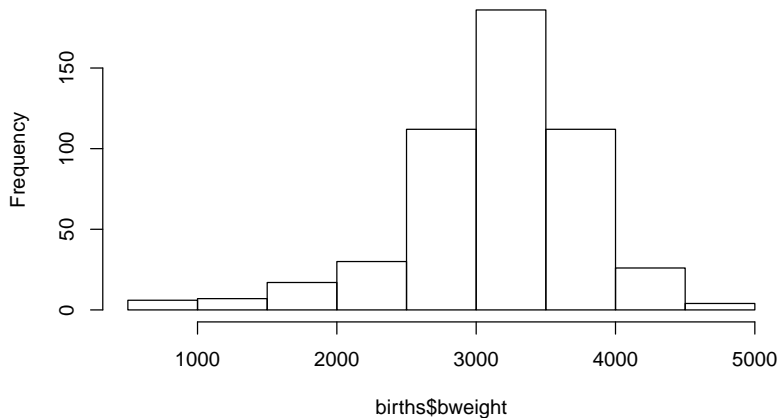
- ▶ O R possui diversas funções e pacotes específicos para a geração de variados gráficos.
- ▶ Inicialmente iremos conhecer os gráficos básicos de duas formas: (1) utilizando funções do pacote base do R e (2) utilizando funções do pacote ggplot2.
- ▶ Materiais de referência do pacote ggplot2:
 - ▶ <http://docs.ggplot2.org/current/>
 - ▶ Os arquivos [ggplot2-book](#) e [handout ggplot2](#).

Histograma

```
hist(births$bweight)
```

Histograma

Histogram of births\$bweight

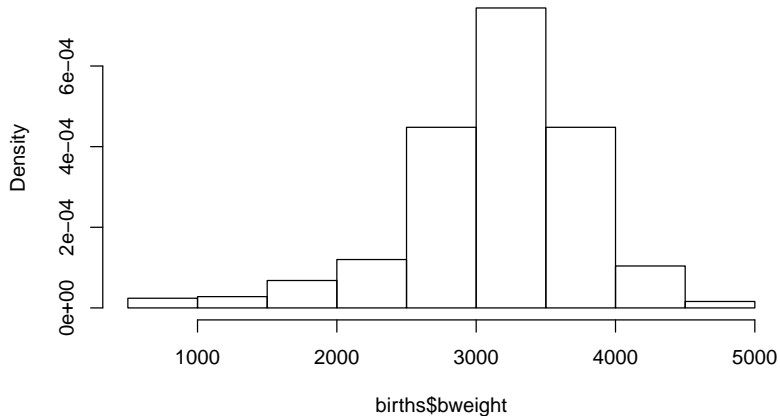


Histograma

```
hist(births$bweight, probability = TRUE)
```

Histograma

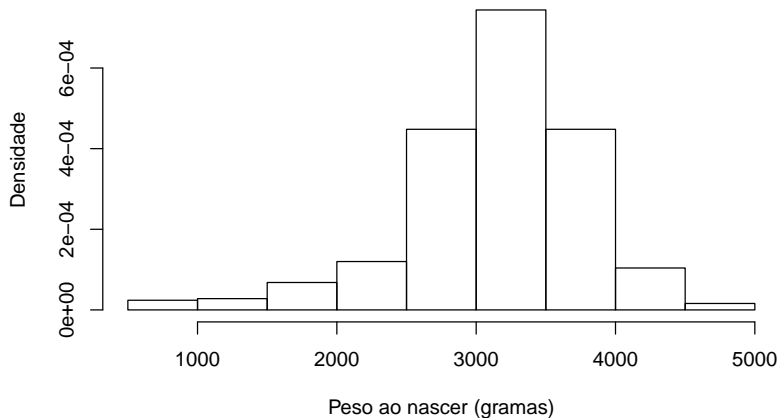
Histogram of births\$bweight



Histograma

```
hist(births$bweight, probability = TRUE,  
     main = "", # Título do gráfico  
     xlab = "Peso ao nascer (gramas)", # Título do eixo x  
     ylab = "Densidade") # Título do eixo y
```

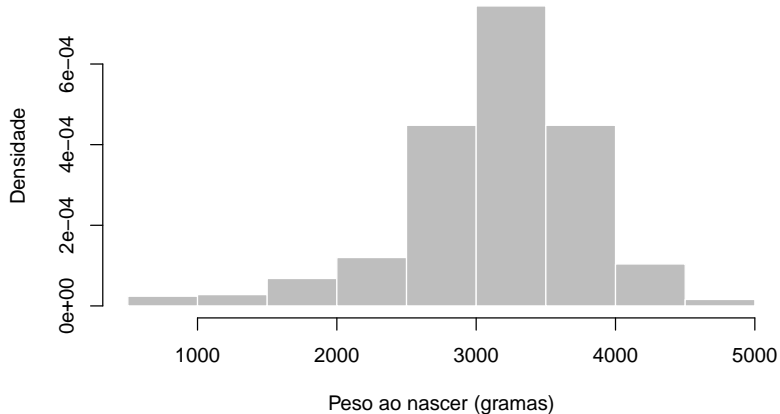
Histograma



Histograma

```
hist(births$bweight, probability = TRUE,  
     main = "",  
     xlab = "Peso ao nascer (gramas)",  
     ylab = "Densidade",  
     border = "white", # Cor da borda  
     col = "gray") # Cor dos retângulos
```

Histograma



Histograma

Cores: o R possui uma vasta paleta de cores que podem ser especificadas pelo nome ou um número.

```
hist(births$bweight, probability = TRUE,  
     main = "",  
     xlab = "Peso ao nascer (gramas)",  
     ylab = "Densidade",  
     border = 3, # Cor da borda  
     col = 2) # Cor dos retângulos
```

Histograma

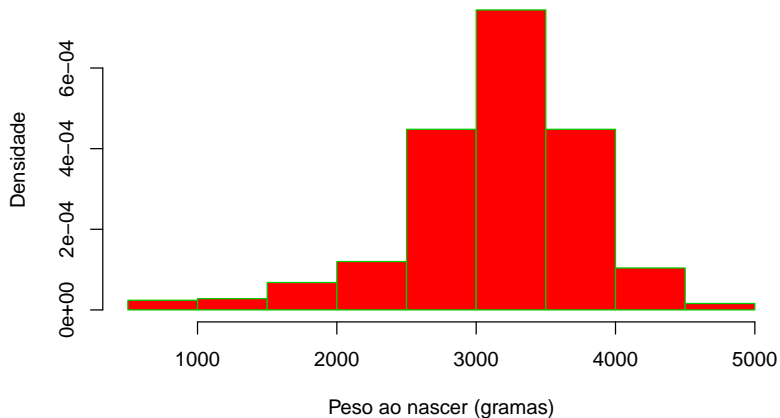


Gráfico de linha

- ▶ Uma das mais frequentes funções utilizadas para fazer gráficos no R é a função `plot()`.
 - ▶ **IMPORTANTE:** esta é uma função genérica, ou seja, o tipo de gráfico produzido é dependente da classe do primeiro argumento.

```
plot(x = testisDK$P[testisDK$A == 0], # var eixo x  
     y = testisDK$D[testisDK$A == 0]) # var eixo y
```

Gráfico de linha

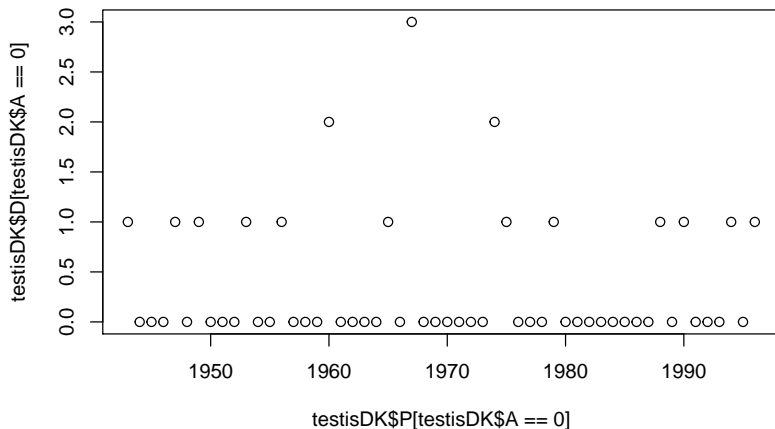


Gráfico de linha

```
plot(x = testisDK$P[testisDK$A == 0], # var eixo x  
     y = testisDK$D[testisDK$A == 0], # var eixo y  
     type = "l", # tipo de grafico  
     xlab = "Anos", # Título do eixo x  
     ylab = "Número de eventos") # Título do eixo y
```

Gráfico de linha

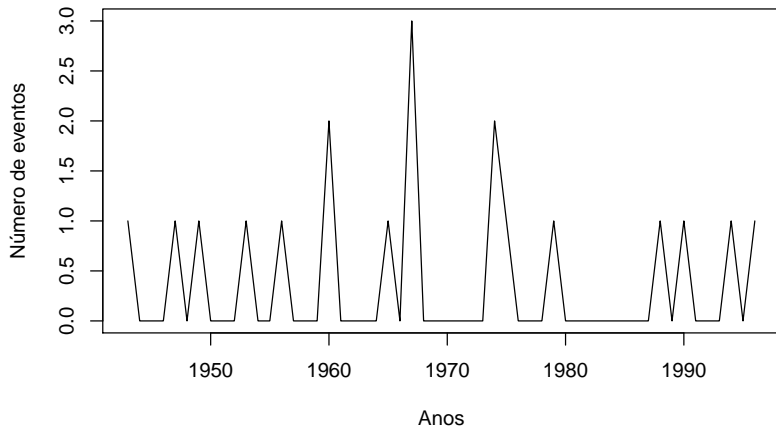


Gráfico de linha

```
plot(x = testisDK$P[testisDK$A == 0], # var eixo x
     y = testisDK$D[testisDK$A == 0], # var eixo y
     type = "l", # tipo de grafico
     lty = 2, # tipo de linha
     lwd = 2, # espessura da linha
     col = "steelblue", # cor da linha
     xlab = "Anos", # Título do eixo x
     ylab = "Número de eventos") # Título do eixo y
```

Gráfico de linha

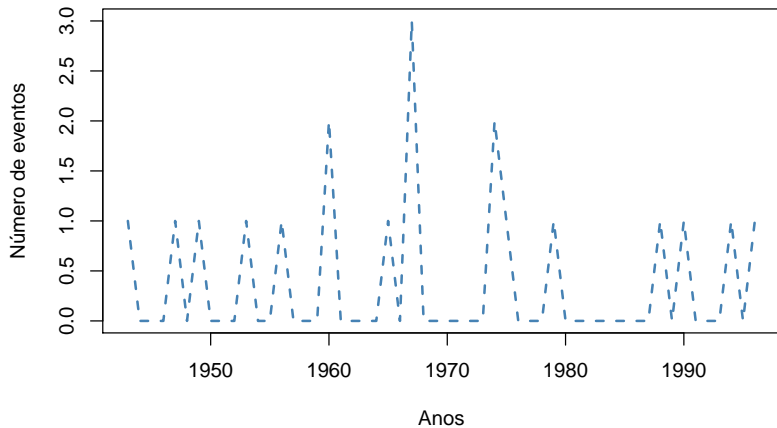


Gráfico de barras

```
barplot(table(births$lowbw))
```

Gráfico de barras

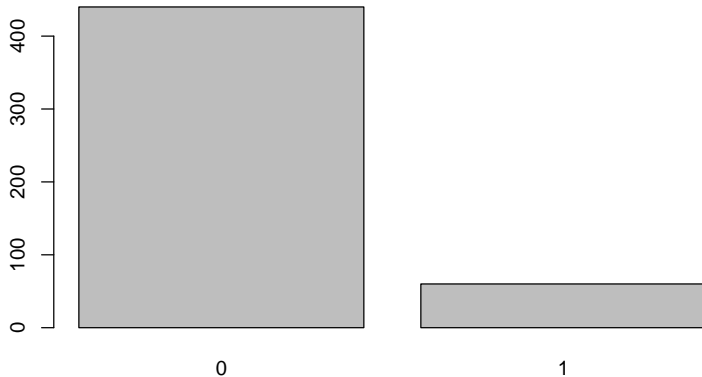


Gráfico de barras

```
# Transformando var numérica em factor
births$lowbw <- factor(births$lowbw,
                       labels = c("Normal",
                                   "Baixo peso"))

barplot(table(births$lowbw),
        xlab = "Baixo peso",
        ylab = "Frequência",
        col = "lightsalmon",
        border = "white")
```

Gráfico de barras

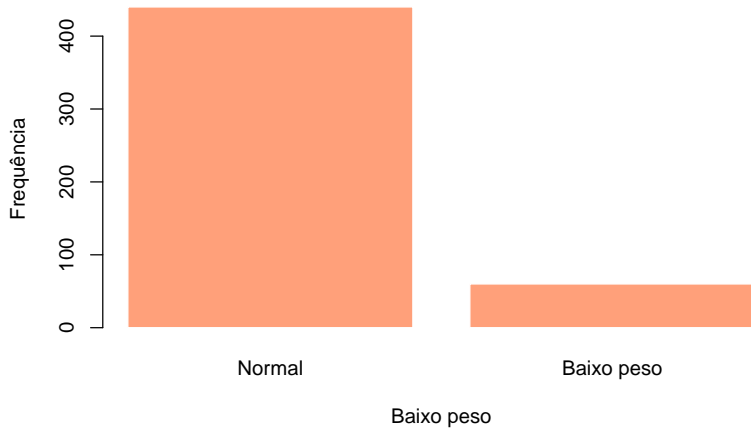


Gráfico de barras

```
xx <- barplot(table(births$lowbw),  
              xlab = "Baixo peso",  
              ylab = "Frequência",  
              col = "lightsalmon",  
              border = "white",  
              ylim = c(0, 1.3 * max(table(births$lowbw))))  
## Adicionando texto no topo das barras  
text(x = xx, y = table(births$lowbw),  
     label = table(births$lowbw),  
     pos = 3, cex = 0.8, col = "red")
```

Gráfico de barras

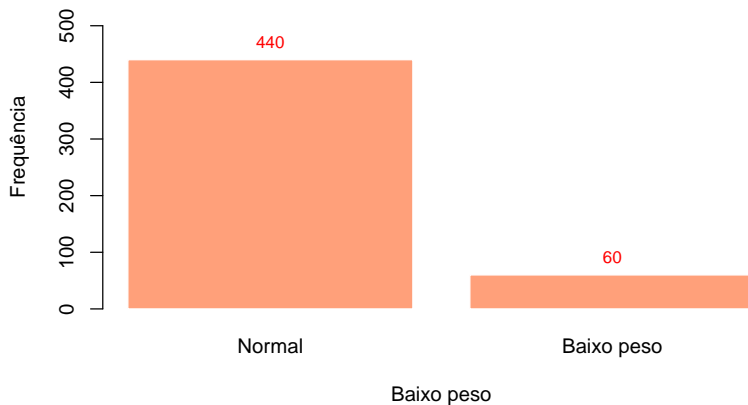
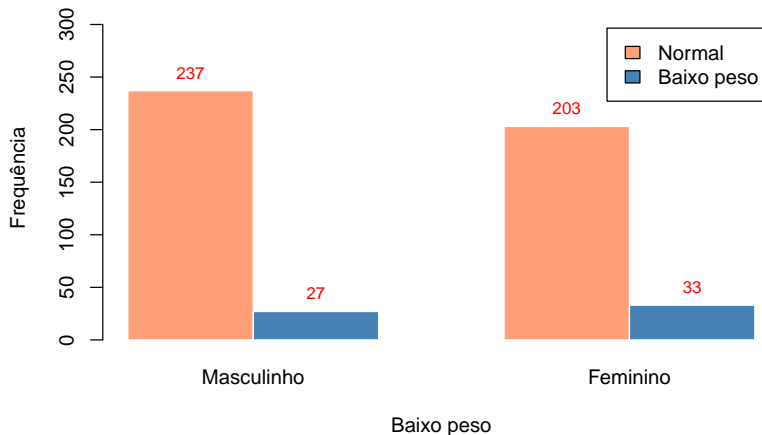


Gráfico de barras

```
# Transformando var numérica em factor
births$sex <- factor(births$sex,
                     labels = c("Masculinho",
                                "Feminino"))

freq <- table(births$lowbw, births$sex)
xx <- barplot(freq,
              xlab = "Baixo peso",
              ylab = "Frequência",
              col = c("lightsalmon", "steelblue"),
              border = "white",
              legend = rownames(freq),
              beside = TRUE,
              ylim = c(0, 1.3 * max(freq)))
text(x = xx, y = freq,
     label = freq,
     pos = 3, cex = 0.8, col = "red")
```

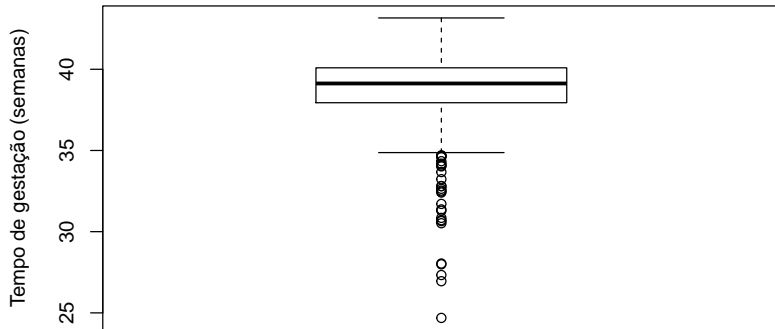
Gráfico de barras



Boxplot

```
boxplot(births$gestwks,  
        ylab = "Tempo de gestação (semanas)")
```

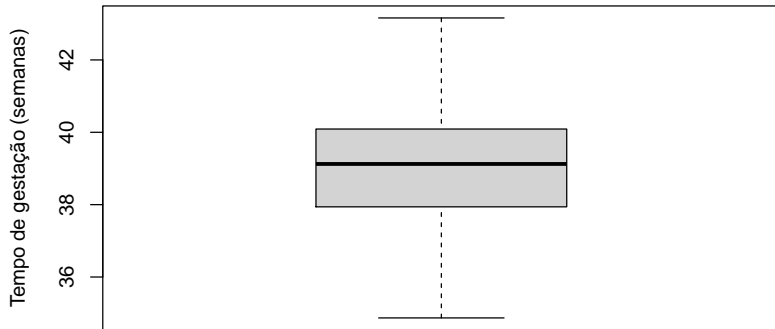
Boxplot



Boxplot

```
# sem outliers e mudando a cor  
boxplot(births$gestwks,  
        ylab = "Tempo de gestação (semanas)",  
        outline = FALSE,  
        col = "lightgray")
```

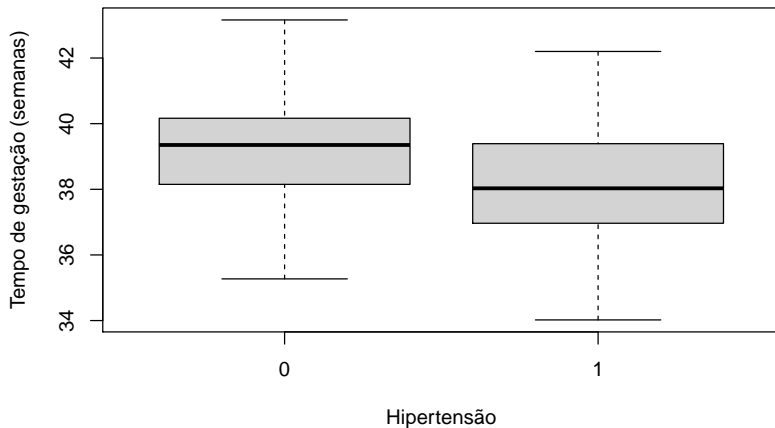
Boxplot



Boxplot

```
# Comparando dist de dois grupos  
boxplot(gestwks ~ factor(hyp), # FORMULA!  
        data = births, # objeto dataframe  
        xlab = "Hipertensão",  
        ylab = "Tempo de gestação (semanas)",  
        outline = FALSE,  
        col = "lightgray")
```

Boxplot



Boxplot

```
births$hyp <- factor(births$hyp,
                     labels = c("Não",
                                "Sim"))

# Utilizando a função plot
plot(gestwks ~ hyp, # FORMULA!
     data = births, # objeto dataframe
     xlab = "Hipertensão",
     ylab = "Tempo de gestação (semanas)",
     outline = FALSE,
     col = "lightgray")
```

Boxplot

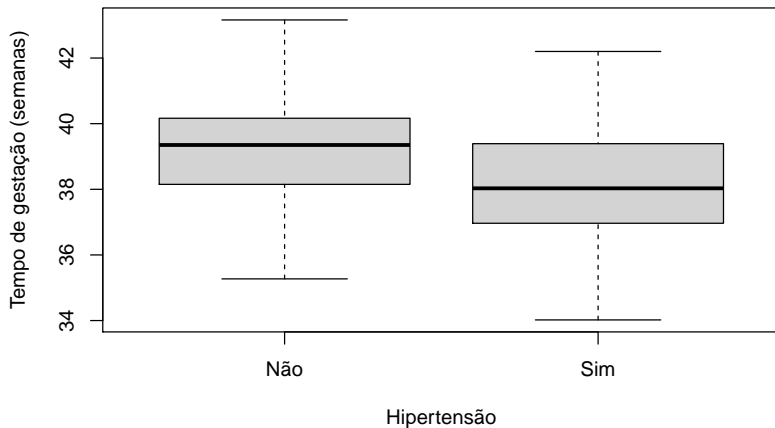


Gráfico de dispersão

```
plot(bweight ~ gestwks,  
      data = births)
```

Gráfico de dispersão

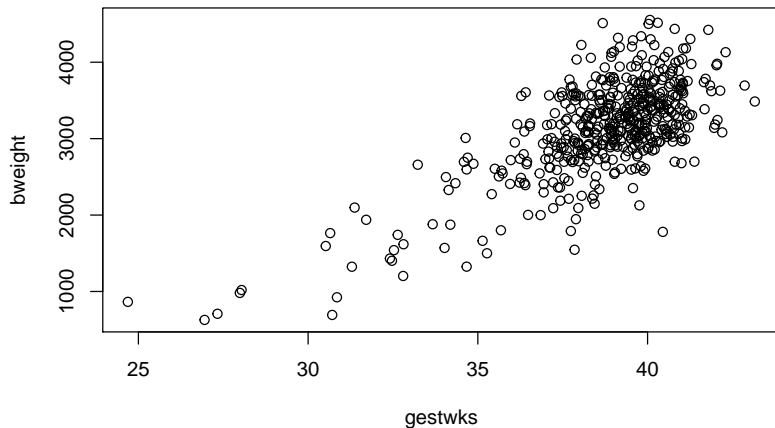
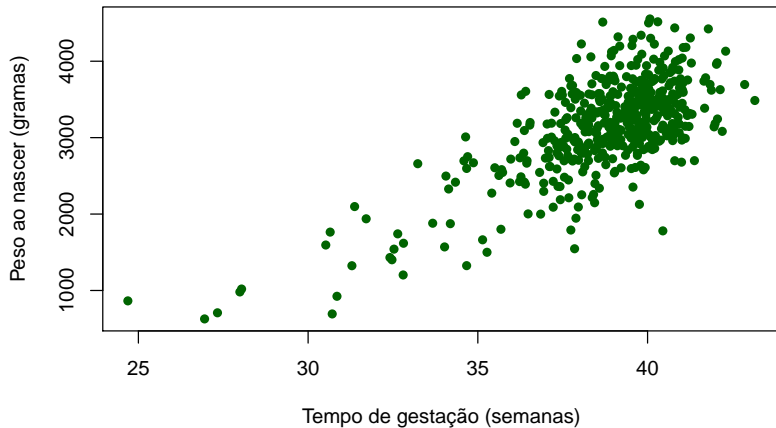


Gráfico de dispersão

```
plot(bweight ~ gestwks,  
     data = births,  
     xlab = "Tempo de gestação (semanas)",  
     ylab = "Peso ao nascer (gramas)",  
     pch = 16,  
     col = "darkgreen")
```

Gráfico de dispersão



ggplot2

- ▶ Podemos gerar os mesmos gráficos **com apenas uma função** (mais fácil?):
 - ▶ A função `qplot()` (*quick plot*) do pacote `ggplot2`.
- ▶ O pacote `ggplot2` é uma implementação da **gramática dos gráficos** para o R (veja o artigo sobre a gramática dos gráficos <http://oestatistico.com.br/2016/08/09/a-gramatica-dos-graficos/>).
- ▶ O pacote `ggplot2` possui outras funções que permitem a criação de gráficos bastante elegantes por camadas (`ggplot()`, `aes()`, `geoms`, etc.).
 - ▶ Para maiores detalhes, veja o material de referência do pacote `ggplot2`.
- ▶ Para utilização das funções do pacote `ggplot2` instale e carregue-o.

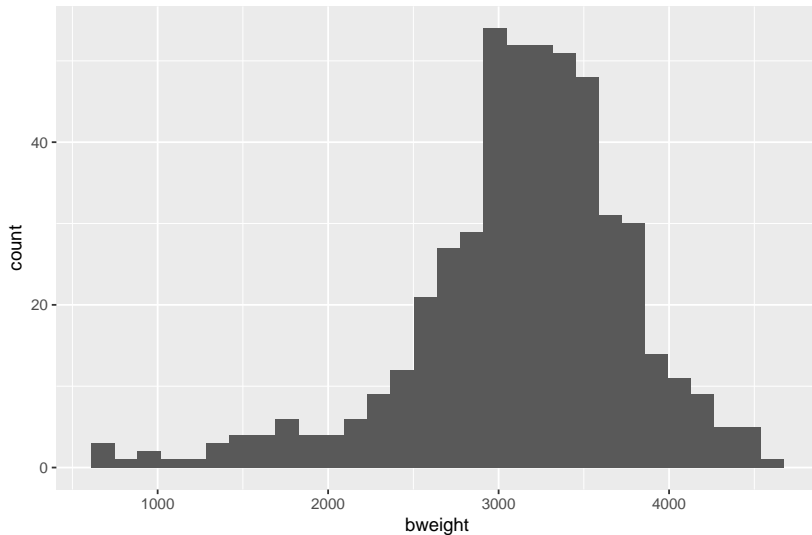
```
install.packages("ggplot2")
```

```
library(ggplot2)
```

Histograma (versão `qplot()`)

```
qplot(bweight, # Nome da variável  
      data = births, # Nome do dataframe  
      geom = "histogram") # Tipo do gráfico
```

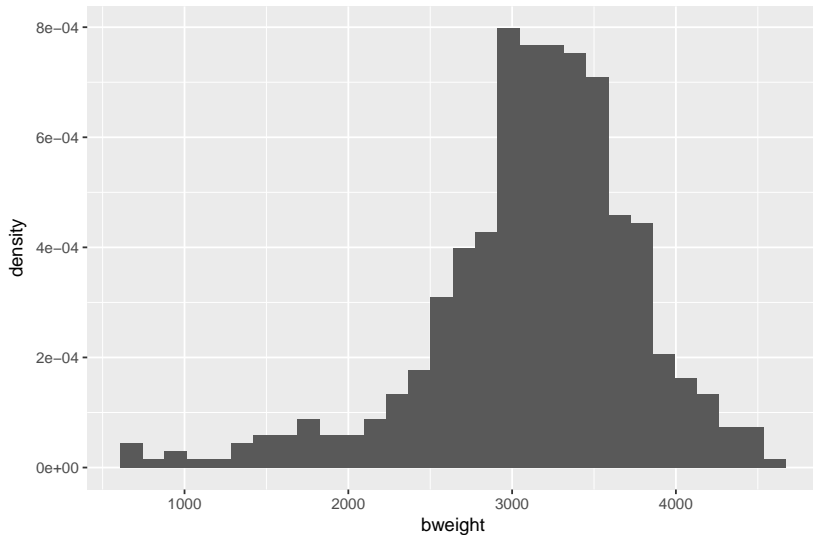
Histograma (versão `qplot()`)



Histograma (versão `qplot()`)

```
qplot(bweight, # Nome da variável  
      data = births, # Nome do dataframe  
      geom = "histogram", # Tipo do gráfico  
      y = ..density..)
```

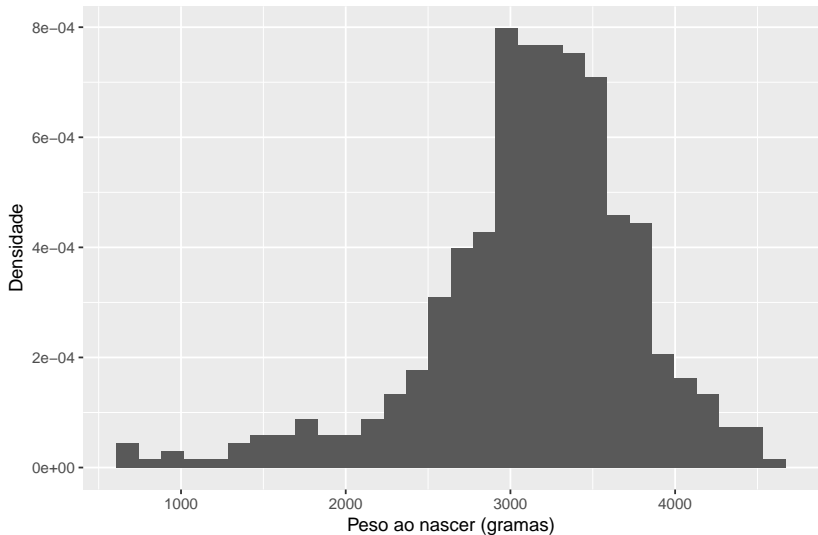
Histograma (versão `qplot()`)



Histograma (versão `qplot()`)

```
qplot(bweight, # Nome da variável
      data = births, # Nome do dataframe
      geom = "histogram", # Tipo do gráfico
      y = ..density..,
      xlab = "Peso ao nascer (gramas)", # Título do eixo x
      ylab = "Densidade") # Título do eixo y
```

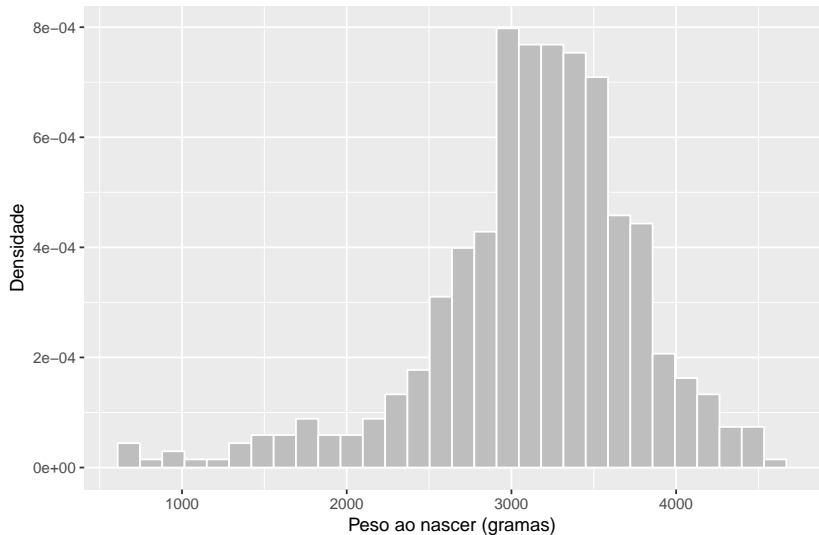

Histograma (versão `qplot()`)



Histograma (versão `qplot()`)

```
qplot(bweight, # Nome da variável
      data = births, # Nome do dataframe
      geom = "histogram", # Tipo do gráfico
      y = ..density..,
      xlab = "Peso ao nascer (gramas)", # Título do eixo x
      ylab = "Densidade", # Título do eixo y
      colour = I("white"), # Cor da borda
      fill = I("gray")) # Cor dos retângulos
```

Histograma (versão `qplot()`)



Histograma (versão `qplot()`)

```
qplot(bweight, # Nome da variável
      data = births, # Nome do dataframe
      geom = "histogram", # Tipo do gráfico
      y = ..density..,
      xlab = "Peso ao nascer (gramas)", # Título do eixo x
      ylab = "Densidade", # Título do eixo y
      colour = I(3), # Cor da borda
      fill = I(2)) # Cor dos retângulos
```

Histograma (versão `qplot()`)

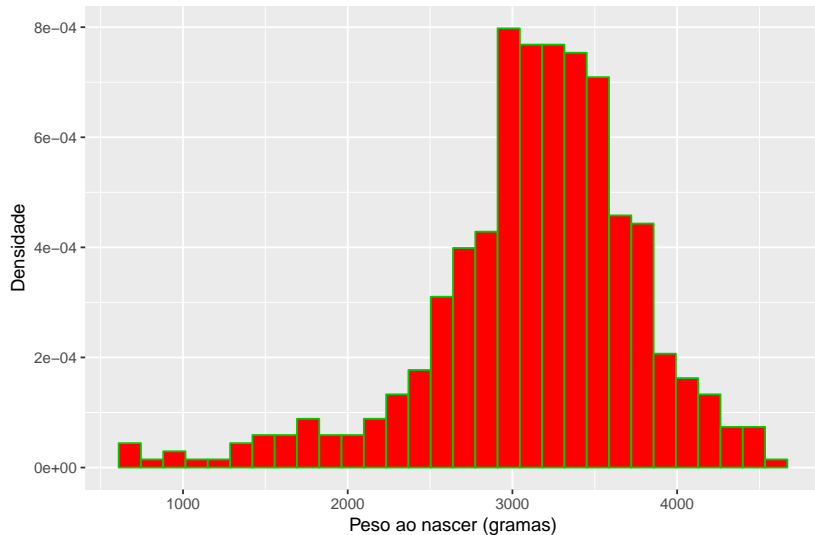


Gráfico de linha (versão `qplot()`)

```
qplot(x = P, # var eixo x
      y = D, # var eixo y
      data = subset(testisDK, subset = A == 0), # dataframe
      geom = "line", # tipo do gráfico
      xlab = "Anos", # Título do eixo x
      ylab = "Número de eventos") # Título do eixo y
```

Gráfico de linha (versão `qplot()`)

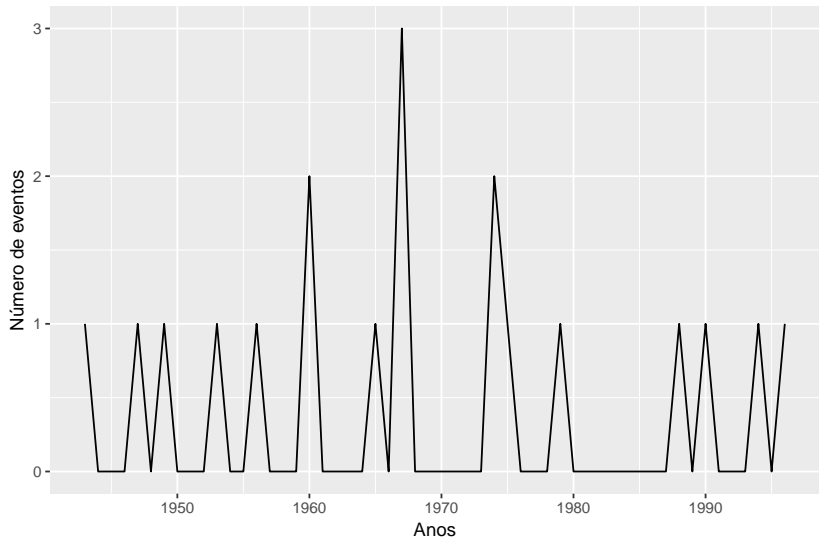


Gráfico de linha (versão `qplot()`)

```
qplot(x = P, # var eixo x
      y = D, # var eixo y
      data = subset(testisDK, subset = A == 0), # dataframe
      geom = "line", # tipo do gráfico
      xlab = "Anos", # Título do eixo x
      ylab = "Número de eventos", # Título do eixo y
      linetype = I("dashed"), # tipo de linha
      size = I(1), # espessura da linha
      colour = I("steelblue")) # cor da linha
```


Gráfico de linha (versão `qplot()`)

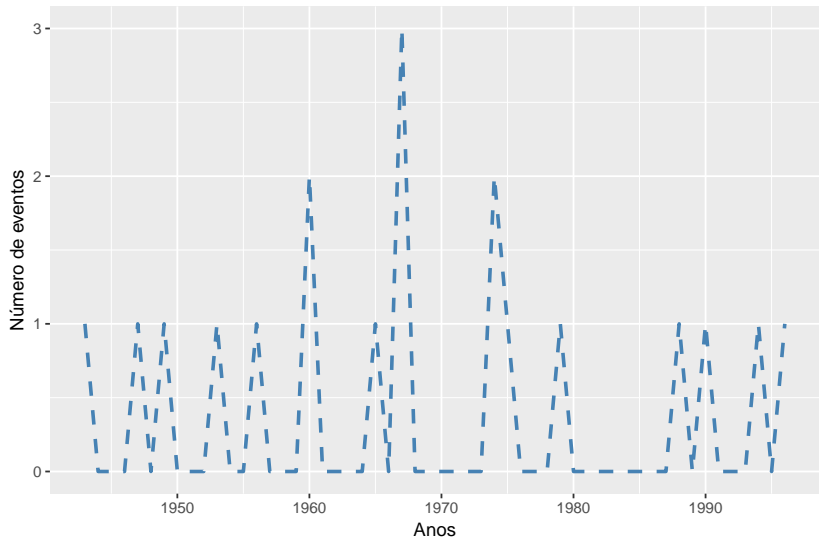


Gráfico de barras (versão `qplot()`)

```
qplot(lowbw,  
      data = births,  
      geom = "bar")
```

Gráfico de barras (versão `qplot()`)

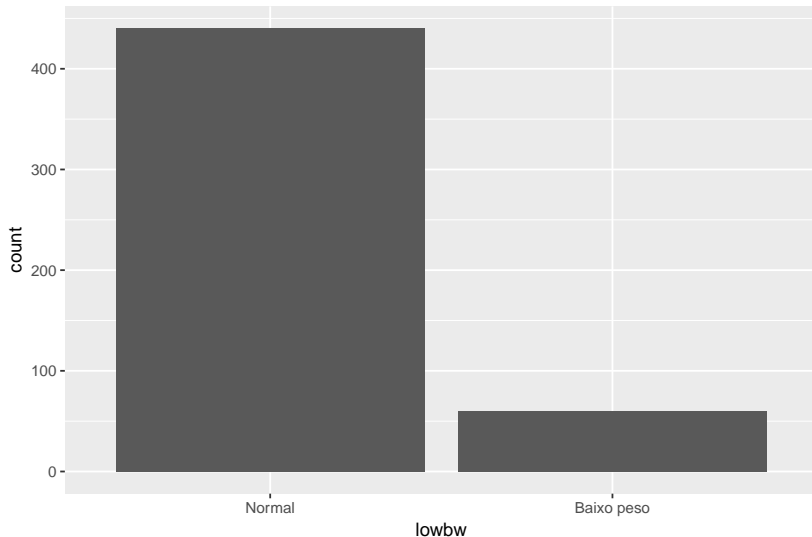


Gráfico de barras (versão `qplot()`)

```
# Transformando var numérica em factor
births$lowbw <- factor(births$lowbw,
                       labels = c("Normal",
                                   "Baixo peso"))

qplot(lowbw,
      data = births,
      geom = "bar",
      xlab = "Baixo peso",
      ylab = "Frequência",
      fill = I("lightsalmon"),
      colour = I("white"))
```

Gráfico de barras (versão `qplot()`)

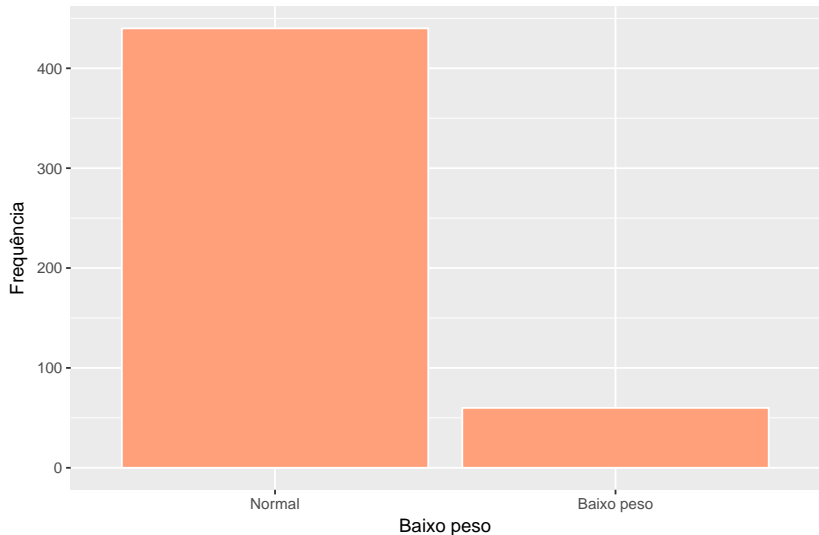
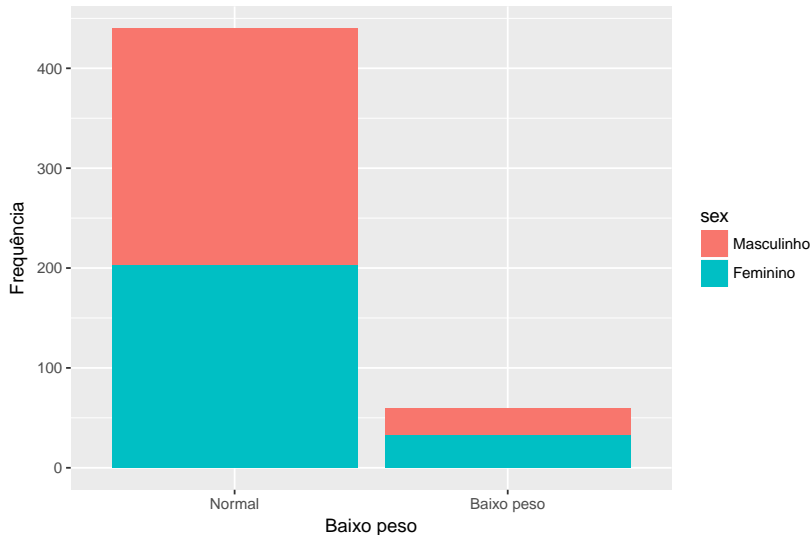


Gráfico de barras (versão `qplot()`)

```
# Transformando var numérica em factor
births$sex <- factor(births$sex,
                     labels = c("Masculinho",
                                "Feminino"))

qplot(lowbw,
      data = births,
      geom = "bar",
      xlab = "Baixo peso",
      ylab = "Frequência",
      fill = sex)
```

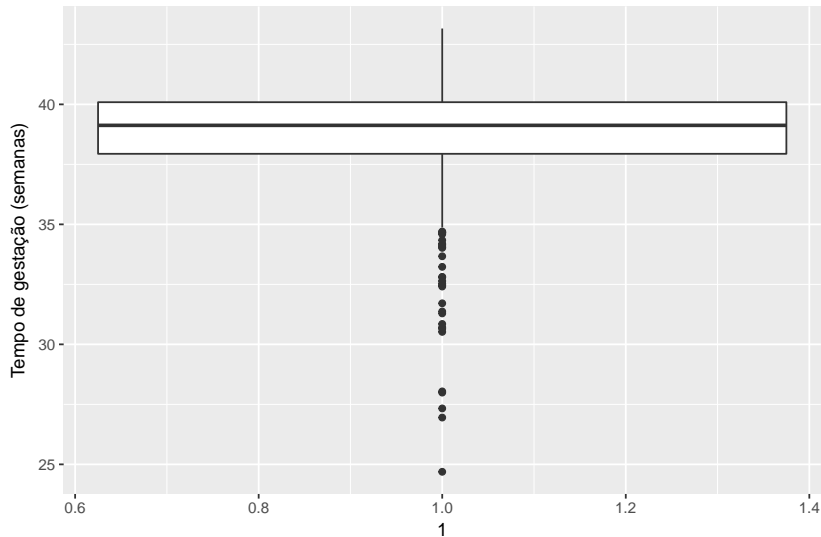
Gráfico de barras (versão `qplot()`)



Boxplot (versão `qplot()`)

```
qplot(1,  
      gestwks,  
      data = births,  
      geom = "boxplot",  
      ylab = "Tempo de gestação (semanas)")
```


Boxplot (versão `qqplot()`)

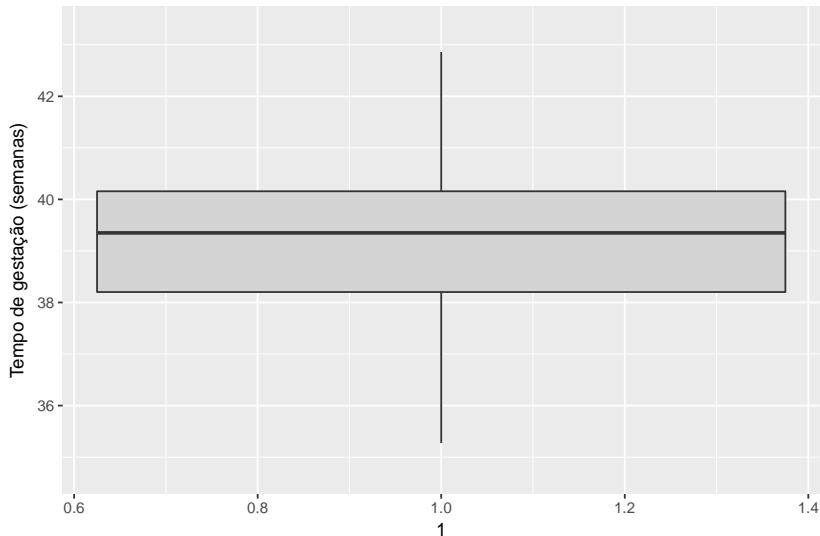


Boxplot (versão `qplot()`)

sem outliers e mudando a cor

```
iqr <- IQR(births$gestwks, na.rm = T)
quartis <- quantile(births$gestwks, probs = c(0.25, 0.75), na.rm = T)
limites <- c(quartis[1] - 1.5 * iqr, quartis[2] + 1.5 * iqr)
qplot(1,
      gestwks,
      data = births,
      geom = "boxplot",
      ylab = "Tempo de gestação (semanas)",
      outlier.shape = NA,
      ylim = limites,
      fill = I("lightgray"))
```

Boxplot (versão `qqplot()`)



Boxplot (versão `qplot()`)

```
# Comparando dist de dois grupos
births$hyp <- factor(births$hyp,
                     labels = c("Não",
                                "Sim"))

qplot(x = hyp,
      y = gestwks,
      data = births,
      geom = "boxplot",
      ylab = "Tempo de gestação (semanas)",
      xlab = "Hipertensão",
      outlier.shape = NA,
      ylim = limites,
      fill = I("lightgray"))
```

Boxplot (versão `qplot()`)

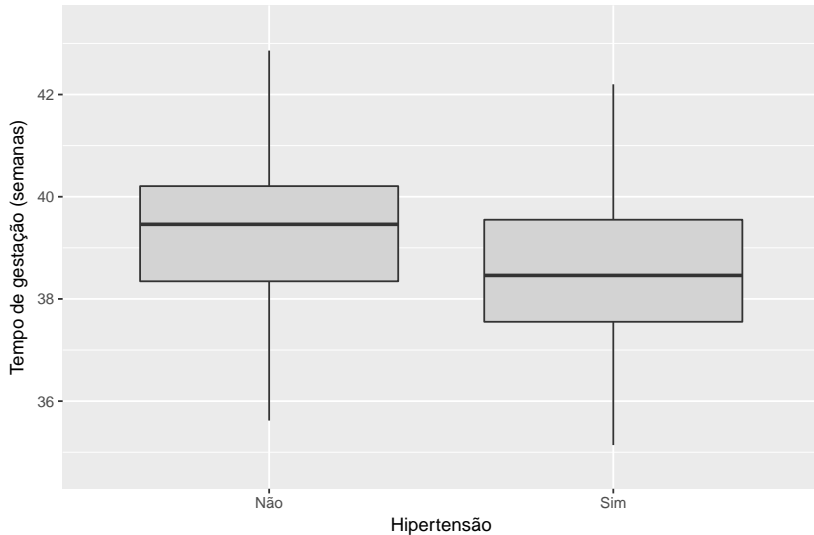


Gráfico de dispersão (versão `qplot()`)

```
qplot(x = gestwks,  
      y = bweight,  
      data = births)
```

Gráfico de dispersão (versão `qplot()`)

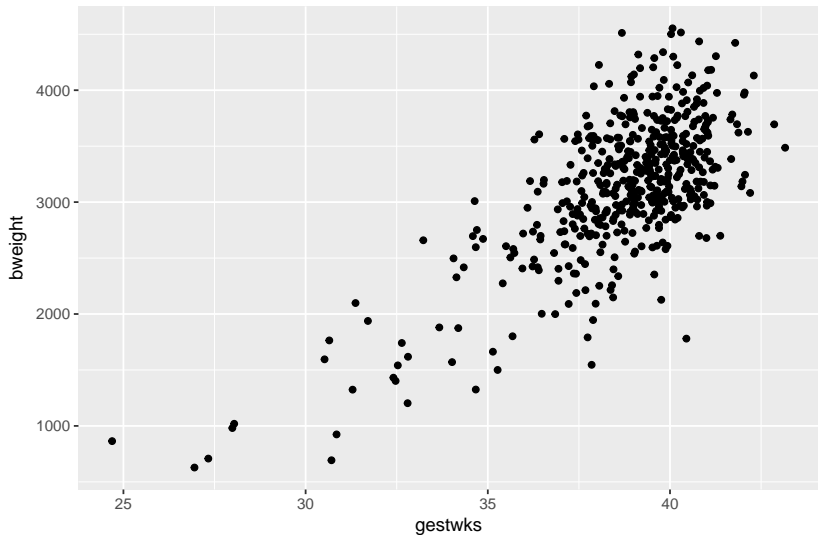
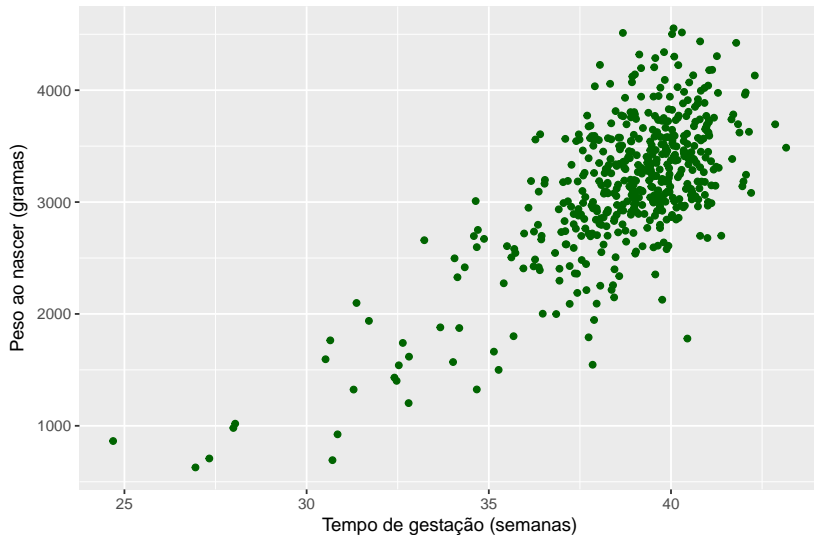


Gráfico de dispersão (versão `qplot()`)

```
qplot(x = gestwks,  
      y = bweight,  
      data = births,  
      xlab = "Tempo de gestação (semanas)",  
      ylab = "Peso ao nascer (gramas)",  
      colour = I("darkgreen"))
```


Gráfico de dispersão (versão `qplot()`)



Veja também

https://rstudio-pubs-static.s3.amazonaws.com/228019_f0c39e05758a4a51b435b19dbd321c23.html

<http://minimaxir.com/2015/02/ggplot-tutorial/>

<http://www.r-graph-gallery.com/portfolio/ggplot2-package/>

<https://timogrossenbacher.ch/2016/12/>

[beautiful-thematic-maps-with-ggplot2-only/](#)

<http://r-statistics.co/>

[Top50-Ggplot2-Visualizations-MasterList-R-Code.html#Scatterplot](#)

<http://faculty.washington.edu/kenrice/heartgraphs/>

Sua vez!

1. Importe os dados do arquivo **evans.dta**.
2. Veja a descrição das variáveis a seguir:

<i>Nome da variável</i>	<i>Descrição da variável</i>	<i>Código da variável</i>
chd	Ocorrência de doença coronariana	0 = não caso 1 = novo caso
cat	Nível sérico de catecolaminas	0 = baixo 1 = alto
age	Idade	Anos
chl	Colesterol	mg/100 mL
smk	Tabagismo	0 = nunca fumou 1 = fumante
ecg	Alterações do eletrocardiograma	0 = ECG normal 1 = qualquer alteração
dbp	Pressão arterial diastólica	mmHg
sbp	Pressão arterial sistólica	mmHg
hpt	Presença de pressão alta	0 = normal 1 = pressão alta ¹
cc	Termo produto de $cat \times hpt$	
ch	Termo produto de $cat \times chl$	

Sua vez!

3. Faça as transformações nas variáveis que julgar necessárias.
4. Apresente gráficos para descrever as variáveis do conjunto de dados.

Inferência estatística no R

Testes de hipóteses e intervalos de confiança

- ▶ Agora que já vimos como descrever o nosso conjunto de dados, vamos ver como realizar alguns testes de hipóteses no R, tais como:
 - ▶ Testes para média e comparação de médias.
 - ▶ Teste para comparação de variâncias.
 - ▶ Testes para proporção e comparação de duas proporções.
 - ▶ Testes para normalidade.

Teste t

- ▶ A função `t.test()` realiza o teste t de Student para uma ou duas amostras.
 - ▶ Testando $H_0 : \mu_{idade} = 30 \times H_1 : \mu_{idade} \neq 30$:

```
t.test(f.dados$idade, mu = 30)
```

```
##  
## One Sample t-test  
##  
## data: f.dados$idade  
## t = 4.5197, df = 35, p-value = 6.78e-05  
## alternative hypothesis: true mean is not equal to 30  
## 95 percent confidence interval:  
## 32.78222 37.31963  
## sample estimates:  
## mean of x  
## 35.05093
```

Teste t

- Testando $H_0 : \mu_{idade} = 35 \times H_1 : \mu_{idade} \geq 35$ (teste unilateral):

```
t.test(f.dados$idade,  
       alternative = "greater",  
       mu = 35)
```

```
##  
## One Sample t-test  
##  
## data: f.dados$idade  
## t = 0.04557, df = 35, p-value = 0.482  
## alternative hypothesis: true mean is greater than 35  
## 95 percent confidence interval:  
## 33.16278 Inf  
## sample estimates:  
## mean of x  
## 35.05093
```


Teste t

- ▶ Testando igualdade de médias (dois grupos)

$$H_0 : \mu_{idade|solteiro} = \mu_{idade|casado} \times H_1 : \mu_{idade|solteiro} \neq \mu_{idade|casado}$$

```
t.test(idade ~ civil,
       data = f.dados)
```

```
##
## Welch Two Sample t-test
##
## data:  idade by civil
## t = -0.55667, df = 27.772, p-value = 0.5822
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -6.090324  3.488240
## sample estimates:
## mean in group solteiro    mean in group casado
##           34.32812           35.62917
```

Teste t

- Note que por *default* a função `t.test()` realiza um teste de comparação de médias considerando que as variâncias das duas amostras **não são iguais**. Podemos alterar esta opção com o argumento `var.equal`:

```
t.test(idade ~ civil,  
       data = f.dados,  
       var.equal = TRUE)
```

```
##  
## Two Sample t-test  
##  
## data: idade by civil  
## t = -0.57292, df = 34, p-value = 0.5705  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -5.916023 3.313939
```

Teste t

```
## sample estimates:  
## mean in group solteiro    mean in group casado  
##           34.32812           35.62917
```

ANOVA

- ▶ Quando o objetivo for comparar mais de dois grupos, utilizaremos a **análise de variância (ANOVA)** para testar a seguinte hipótese:

$$H_0 : \mu_1 = \mu_2 = \mu_3$$

```
anovaIdade <- aov(idade ~ instrucao,  
                  data = f.dados)  
summary(anovaIdade)
```

```
##           Df Sum Sq Mean Sq F value Pr(>F)  
## instrucao    2   91.7   45.87    1.022  0.371  
## Residuals   33 1481.8   44.90
```

Qui-quadrado

- ▶ Testando associação entre variáveis categóricas com o **teste qui-quadrado de Pearson**:

```
chisq.test(table(f.dados$instrucao, f.dados$regiao))
```

```
## Warning in chisq.test(table(f.dados$instrucao, f.dados$regiao)): Chi-  
## squared approximation may be incorrect
```

```
##  
## Pearson's Chi-squared test  
##  
## data:  table(f.dados$instrucao, f.dados$regiao)  
## X-squared = 0.66142, df = 4, p-value = 0.956
```

Wilcoxon

- ▶ Quando a suposição de normalidade do teste t não é atendida, podemos utilizar o **teste de Wilcoxon** para comparar duas amostras:

```
wilcox.test(idade ~ civil,  
            data = f.dados)
```

```
## Warning in wilcox.test.default(x = c(26.25, 20.8333333333333,  
## 40.5833333333333, : cannot compute exact p-value with ties
```

```
##  
## Wilcoxon rank sum test with continuity correction  
##  
## data: idade by civil  
## W = 151, p-value = 0.7867  
## alternative hypothesis: true location shift is not equal to 0
```

Kruskal-Wallis

- ▶ De maneira semelhante, temos o teste de **Kruskal-Wallis** como alternativa a ANOVA:

```
kruskal.test(idade ~ instrucao,  
             data = f.dados)
```

```
##  
##  Kruskal-Wallis rank sum test  
##  
## data:  idade by instrucao  
## Kruskal-Wallis chi-squared = 2.3877, df = 2, p-value = 0.3031
```

Testando normalidade

- ▶ Existem diversos testes para avaliar a normalidade de uma variável H_0 : “os dados vêm de uma distribuição normal”
- ▶ Um dos mais conhecidos é o **teste de Shapiro-Wilks**:

```
shapiro.test(f.dados$idade)
```

```
##  
##  Shapiro-Wilk normality test  
##  
## data:  f.dados$idade  
## W = 0.99179, p-value = 0.9942
```


Sua vez!

1. Utilize os testes que acabamos de ver nas variáveis do conjunto de dados do arquivo **evans.dta** (formule suas hipóteses).
2. **Para casa:** instale o pacote do R `compareGroups`.
 - 2.1 Após carregar o pacote, consulte o seu help (`help(package = "compareGroups")`) e acesse a vinheta **User guides, package vignettes and other documentation**.
 - 2.2 Estude esta vinheta.