

# Aula 1

Frederico Bertholini  
Álvaro João Pereira Filho

Curso de R  
10/06/2019



# O que é R?

- R é um ambiente de desenvolvimento para cálculos estatísticos e gráficos
- Por enquanto, divide com o Python o posto de software mais popular entre cientistas sociais
- Vantagens
  - Software Livre;
  - Documentação completa e acessível;
  - Diversidade de arquivos;
  - Replicabilidade de rotinas

# Lei de ouro

- Existe uma Lei de ouro em relação ao R:
- Se é possível fazer no Excel, é possível fazer no R
- Entretanto, se é possível fazer no R, não necessariamente é possível fazer no Excel
- Vários cálculos estatísticos mais sofisticados estão disponíveis no R através de pacotes desenvolvidos pela comunidade

# Habilidades necessárias

## ■ Escrita:

- Elementos (ex: numeric, character, factor...);
- Funções básicas (ex: sum(), table(), sd());
- Composição do script (ex: c(), for loop);
- Sensibilidade

## ■ Leitura:

- Identificação de funções;
- Diferentes soluções;
- Pacotes intuitivos;
- Alertas de erros;

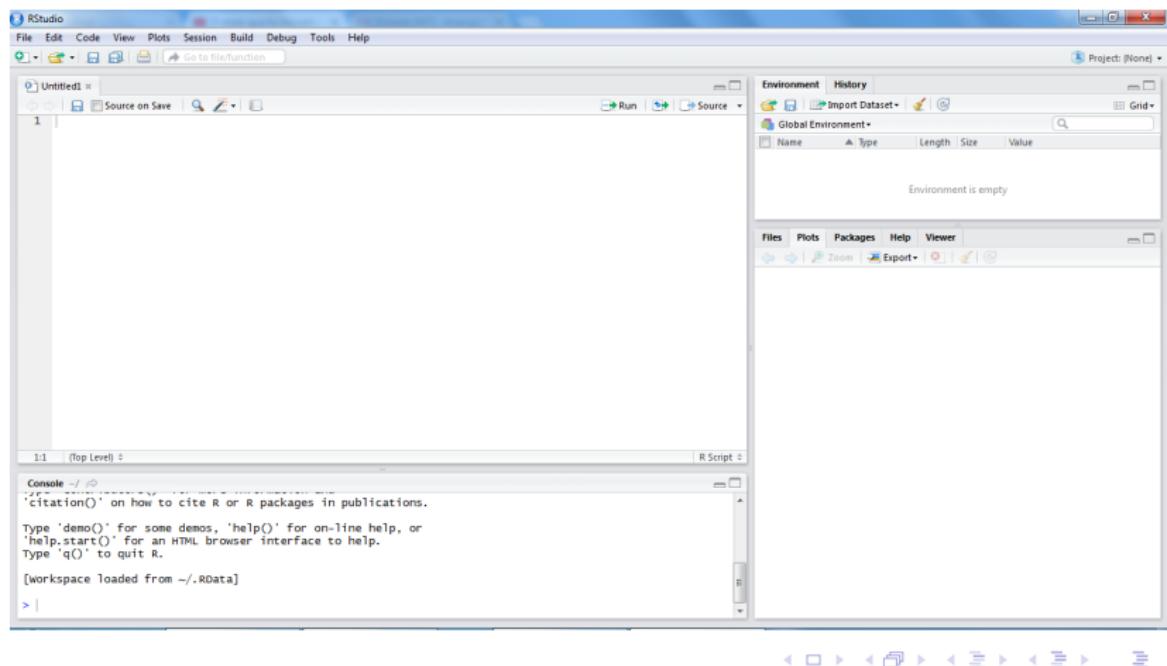
# Habilidade mais importante

- Devido à multiplicidade de soluções e de alertas de erros
- Também, ao universo de pacotes com diversas funções
- A habilidade mais importante é saber pesquisar corretamente no Google
  - Melhor em inglês pela quantidade de fóruns;
  - Melhor fórum: stackoverflow.com
- É importante adquirir autonomia com o software

# RStudio

- RStudio é um ambiente para desenvolvimento do R
- Não é o melhor, porém é um dos mais amigáveis
- Ao longo do curso utilizaremos o Rstudio
  - Utilizando nada dos botões ou possibilidades específicas do RStudio
  - Portanto, utilizaremos a programação somente

# Aparência do RStudio



# Básico do básico

- Para começar é importante apresentar um operador e um comando
- Hastags
  - Insere comentários sem gerar outputs
  - É importante para organizar e registrar dentro dos scripts
- Ctrl + enter
  - Roda as funções e programações escritas nas linhas selecionadas
  - As linhas escritas no ambiente acima do console, onde é possível deixar registrado o script

# Como sai no R?

```
> #Insere comentarios sem rodar o comando  
  
> Importante para organizar e  
Error: unexpected symbol in "Importante para"  
  
> ##Registrar o que se vai fazer ou foi feito
```

- Quando o comando Ctrl + enter é acionado, o campo **Console** registra > e o comando escrito no script, em seguida

# R como calculadora

- Um função primordial e básica do R, como software estatístico, é de calculadora
- O R possui os operadores básicos da matemática como '+', '-', '\*' e '/'

5 + 5

5 - 3

4 \* 9

16 / 2

# R como calculadora

- Um função primordial e básica do R, como software estatístico, é de calculadora
- O R possui os operadores básicos da matemática como '+', '-', '\*' e '/'

```
> 5 + 5  
[1] 10  
> 5 - 3  
[1] 2  
> 4 * 9  
[1] 36  
> 16 / 2  
[1] 8
```

# R como calculadora

- Assim como na matemática, atenção em relação aos ()

`(5 + 6) * 3`

`5 + 6 * 3`

- Além das funções de exponencial e raiz quadrada
- Respectivamente, é `sqrt()`

`2 ^ 2`

`sqrt(36)`

# R como calculadora

- Assim como na matemática, atenção em relação aos ()

```
> (5 + 6) * 3  
[1] 33  
> 5 + 6 * 3  
[1] 23
```

- Além das funções de exponencial e raiz quadrada
- Respectivamente, é sqrt()

```
> 2 ^ 2  
[1] 4  
> sqrt(36)  
[1] 6
```

# Lógica

- O R permite também avaliações lógicas
- Ou seja, o software possui operadores lógicos afim de fazer testes lógicos com resultados de Verdadeiro ou Falso de acordo com a proposição
- Os principais operadores são ==, < , >, <= , >= e !=

5 == 5

5 <= 5 / 5

5 \* 4 > 5

3 != 6

# Lógica

- O R permite também avaliações lógicas
- Ou seja, o software possui operadores lógicos afim de fazer testes lógicos com resultados de Verdadeiro ou Falso de acordo com a proposição
- Os principais operadores são ==, < , >, <= , >= e !=

```
> 5 == 5  
[1] TRUE  
5 <= 5 / 5  
[1] FALSE  
5 * 4 > 5  
[1] TRUE  
3 != 6  
[1] TRUE
```

# Lógica

- Testamos também Verdade e Falsidade

```
TRUE == TRUE
```

```
TRUE <= FALSE
```

- Assim como estamos caracteres

```
"Python" == "python"
```

```
"Stata" != "Sasta"
```

# Lógica

- Testamos também a relação entre Verdade e Falsidade

```
> TRUE == TRUE  
[1] TRUE  
> TRUE <= FALSE  
[1] FALSE
```

- Assim como testamos a relação entre caracteres, primeiro com letras maiúsculas e com a ordem diferente

```
> "Python" == "python"  
[1] FALSE  
> "Stata" != "Sasta"  
[1] TRUE
```

# Operadores lógicos especiais

- Atenção para os operadores e e ou
- O primeiro, para ser verdade, precisa que todos os pressupostos sejam verdadeiros

$(3 == 3) \& (4 != 5)$

- O ou, por sua vez, para ser verdade precisa que apenas 1 pressuposto seja verdadeiro

$(3 != 3) | (4 != 5)$

# Operadores lógicos especiais

- Atenção para os operadores e e ou
- O primeiro, para ser verdade, precisa que todos os pressupostos sejam verdadeiros

```
> (3 == 3) & (4 != 5)  
[1] TRUE
```

- O ou, por sua vez, para ser verdade precisa que apenas 1 pressuposto seja verdadeiro

```
> (3 != 3) | (4 != 5)  
[1] TRUE
```

# A famosa setinha

- Trata-se da famosa setinha que indica objetos (valores, vetores, dataframes) para alguma etiqueta
- Dessa maneira, podemos 'salvar' os objetos nas etiquetas para utilizarmos através dessas em qualquer momento ao longo do script
- Quando utilizadas em operações, as etiquetas representam aquilo que fora atribuído a elas
- Quando criamos a etiqueta, não geramos *outputs*, apenas quando rodamos diretamente a etiqueta

```
sorte <- 5
```

# A famosa setinha

- Trata-se da famosa setinha que indica objetos (valores, vetores, dataframes) para alguma etiqueta
- Dessa maneira, podemos 'salvar' os objetos nas etiquetas para utilizarmos através dessas em qualquer momento ao longo do script
- Quando utilizadas em operações, as etiquetas representam aquilo que fora atribuído a elas
- Quando criamos a etiqueta, não geramos *outputs*, apenas quando rodamos diretamente a etiqueta

```
> sorte <- 5
```

# A famosa setinha

- Trata-se da famosa setinha que indica objetos (valores, vetores, dataframes) para alguma etiqueta
- Dessa maneira, podemos 'salvar' os objetos nas etiquetas para utilizarmos através dessas em qualquer momento ao longo do script
- Quando utilizadas em operações, as etiquetas representam aquilo que fora atribuído a elas
- Quando criamos a etiqueta, não geramos *outputs*, apenas quando rodamos diretamente a etiqueta

```
> sorte <- 5
```

```
> sorte  
[1] 5
```

# Regras do uso da setinha

- Atenção, letras maiúsculas e minúsculas importam

```
> sorte <- 5
```

```
> Sorte
```

```
Error: object 'Sorte' not found
```

- Também não podemos criar etiquetas que começam com números

```
> 15luck <- 15
```

```
Error: unexpected symbol in "15luck"
```

- Cuidado com a utilização de etiquetas com o mesmo nome de funções, pode gerar confusão no script

# Classes

- Em basicamente tudo que iremos fazer no R, a classe da informação importa
- Em termos elementares, ou ao nível dos valores, existem três grandes classes:
  - Numeric (númerico);
  - Logical (lógico);
  - Character ou factor (caracteres);
- Para obter a informação sobre a classe, iremos aprender nossa primeira função no R: `class()`

# Númerico

- Numeric é a classe composta por valores néricos

```
> class(sorte)  
[1] "numeric"
```

- Objetos do tipo "numeric" permite funções matemáticas como média, mediana e etc...
- No caso de valores decimais, utilizamos ponto ao invés de vírgula

```
> decimal <- 3,5  
Error: unexpected ',', in "decimal<-3,"  
  
> decimal <- 3.5  
[1] 3.5
```

# Lógico

- Logical é a classe composta por TRUE, FALSE e NA

```
> vdd <- TRUE
```

```
> class(vdd)  
[1] "logical"
```

- Não é preciso escrever sempre TRUE e FALSE, isto é, podemos resumir para T e F, respectivamente
- Por trás dos valores T e F, há valores numéricos correspondente a 1 e 0

```
> T + F  
[1] 1
```

# Caracteres

- Characters é a classe composta por nomes
- Importante característica é que os nomes devem estar dentro de aspas, caso contrário, o R não reconhecerá como caracter

```
> nome <- "Alvaro"
```

```
> nome  
[1] "Alvaro"
```

```
> class(nome)  
[1] "character"
```

# Caracteres vs Fatores

- Em termos estatísticos, não há diferença entre caracteres e fatores
- Fatores apresentam as categorias por dentro de vetor (conceito que será apresentado na próxima aula)
- Em termos de manipulação de dataframes, porém, veremos que esses dois objetos terão tratamento distintos, a começar pela importação da base e a definição de string tratada como factor

# Escrita

- 1 Atribua valor da sua altura à etiqueta chamada 'altura'
- 2 Qual é a classe do seu objeto 'altura'?
- 3 Faça a média da sua altura com a minha altura (1,72)
- 4 Um triângulo reto com catetos igual a 6 e 8, qual é o comprimento da hipotenusa?
- 5 A hipotenusa encontrada é diferente de 11? Menor ou igual?

# Leitura

- Leia o script abaixo e comente com as hashtags o que cada código quer dizer:

```
#  
((T == F) | (F != T)) != ((T == F) & (F != T))  
#  
senador <- 334  
#  
class(senador)  
#  
name <- "Flavio"  
#  
class(senador) != class(name)
```

# O que são vetores?

- Vetores são combinações de valores em uma estrutura unidimensional
- Ou seja, podem ser combinações de números, valores lógicos, nomes e várias outras possibilidades
- Por exemplo, combinação de valores pares: 2,4,6,8
- Ou a combinações de nomes começando por P: Pedro, Paula, Pietro, Paloma
- Combinação de valores lógico: TRUE, FALSE, TRUE

# Criação de vetores

- Para criar vetores é importante combinar valores
- A função necessária para a criação de vetores, portanto, é a função `c()`, c de *combine* ou *concatenate*

`c(2, 4, 6, 8)`

`c("Pedro", "Paula", "Pietro", "Paloma")`

`c(TRUE, FALSE, TRUE, FALSE)`

# Criação de vetores

- Para criar vetores é importante combinar valores
- A função necessária para a criação de vetores, portanto, é a função `c()`, c de *combine* ou *concatenate*

```
> c(2,4,6,8)  
[1] 2 4 6 8
```

```
> c("Pedro","Paula","Pietro","Paloma")  
[1] "Pedro"  "Paula"  "Pietro" "Paloma"
```

```
> c(TRUE ,FALSE ,TRUE ,FALSE)  
[1] TRUE FALSE TRUE FALSE
```

# Etiquetas para vetores

- Assim como fizemos com os valores, atribuímos também etiquetas aos vetores
- Dessa maneira, podemos trabalhar com as sequências ao longo do script, assim como dar razão ou justificativa aos valores combinados

```
> n.pares <- c(2,4,6,8)
```

```
> nomes.com.p <-  
c("Pedro","Paula","Pietro","Paloma")
```

```
> valores.log <- c(TRUE,FALSE,TRUE,FALSE)
```

# Classes dos vetores

- Os vetores também possuem classes
- Essas categorias determinam as operações possíveis dentro de um vetor
- Para obter a informação sobre a classe de um vetor, utilizamos a função `class()`

```
class(n.pares)
```

```
class(nomes.com.p)
```

```
class(valores.log)
```

# Classes dos vetores

- Os vetores também possuem classes
- Essas categorias determinam as operações possíveis dentro de um vetor
- Para obter a informação sobre a classe de um vetor, utilizamos a função `class()`

```
> class(n.pares)
[1] "numeric"

> class(nomes.com.p)
[1] "character"

> class(valores.log)
[1] "logical"
```

# Comprimento de vetores

- Os vetores podem ser medidos em relação ao seu comprimento
- O seu comprimento define a extensão do vetor, assim como quantos elementos estão presentes dentro da combinação de valores
- O comprimento pode ser medido pela função `length()`

```
length(n.pares)
```

```
length(nomes.com.p)
```

```
length(valores.log)
```

# Comprimento de vetores

- Os vetores podem ser medidos em relação ao seu comprimento
- O seu comprimento define a extensão do vetor, assim como quantos elementos estão presentes dentro da combinação de valores
- O comprimento pode ser medido pela função `length()`

```
> length(n.pares)
[1] 4
> length(nomes.com.p)
[1] 4
> length(valores.log)
[1] 4
```

# Somatório de vetores

- Para vetores numéricos podemos somar os valores de um vetor
- A função se chama `sum()` que corresponde ao  $\sum$

```
> sum(n.pares)  
[1] 20
```

- Em vetores com valores lógicos, com TRUE e FALSE, o `sum` soma o número de T que temos dentro de um vetor

```
> sum(valores.log)  
[1] 2
```

# Somatório de vetores

- A função `sum()` pode se tornar ainda um contador de um teste
- Por exemplo, queremos saber quantos nomes são iguais ao de Pedro no vetor `'nomes.com.p'`

```
> teste1 <- nomes.com.p == "Pedro"

> sum(teste1)
[1] 1

> teste1
[1] TRUE FALSE FALSE FALSE
```

# Seleção de elementos

- Queremos selecionar no vetor nomes.com.p o segundo elemento que é Paula
- Vejamos, primeiro, como se compõe o vetor de caracteres nomes.com.p e a lógica do posicionamento dentro do vetor

```
> nomes.com.p  
[1] "Pedro", "Paula", "Pietro", "Paloma"
```

# Seleção de elementos

- Vamos testar se Paula se inclui dentro do vetor sem termos que ver no console
- As vezes, vetores são maiores do que temos como exemplo
- O teste lógico, portanto, utilizamos o operado `%in%`, que retorna TRUE, caso o valor esteja incluso no vetor

```
> Paula %in% nomes.com.p  
[1] TRUE
```

# Seleção de elementos

- Algumas operações se restrigem à determinados elementos ou à um conjunto deles incluso dentro de um vetor
- Para realizar tais procedimentos precisamos saber escolher elementos
- Para isso, utilizaremos o operador '[' ]' depois de apontar o vetor
- Mais especificamente:
  - Vetor;
  - [];
  - Posição ou regra;

# Seleção de elementos

- Para escolher Paula, portanto, vamos definir como a segunda posição dentro do vetor

```
> nomes.com.p[2]  
[1] "Paula"
```

- O posicionamento dos nomes ao longo do vetor determina como seleciona-lo individualmente
- Porém, podemos selecionar através do valor

```
> nomes.com.p[nomes.com.p == "Paula"]  
[1] "Paula"
```

# Seleção de elementos

- Selecionamos também o último valor do vetor nomes.com.p
- Sabemos que o vetor possui 4 valores utilizando a função length()

```
> length(nomes.com.p)  
[1] 4
```

- Assim o último valor é igual a 4
- Para selecionarmos o último valor, ou definimos como a posição 4 ou apenas o length() do vetor

```
> nomes.com.p[4]  
[1] "Paloma"
```

```
> nomes.com.p[length(nomes.com.p)]  
[1] "Paloma"
```

# Seleção de elementos

- Para seleção de elementos através de regras, podemos utilizar o vetor numérico de n.pares

```
> n.pares <- c(2,4,6,8)
```

```
> n.pares  
[1] 2 4 6 8
```

- O objetivo aqui é selecionar os elementos maiores que 5, que é a mediana do vetor

```
> median(n.pares)  
[1] 5
```

# Seleção de elementos

- Selecionamos, assim, os elementos acima da mediana do vetor n.pares, que são os valores 6 e 8

```
> n.pares[n.pares > 5]  
[1] 6 8
```

- Ou ainda podemos definir o valor 5 como o valor da mediana
- O resultado é o mesmo, porém, deixa claro ao leitor do seu script que se trata dos valores acima da mediana do vetor n.pares

```
> n.pares[n.pares > median(n.pares)]  
[1] 6 8
```

# Aula 1

Frederico Bertholini  
Álvaro João Pereira Filho

Curso de R  
10/06/2019

# Aula 2

Frederico Bertholini  
Álvaro João Pereira Filho

Curso de R  
12/06/2019



LAPCIPP

Laboratório de Pesquisa em  
Comportamento Político,  
Instituições e Políticas Públicas

# Operações com vetores

- Em vetores numéricos, podemos fazer operações matemáticas a partir do seu posicionamento
- Vamos calcular o alcance do vetor "n.pares"

```
#ultimo elemento  
> n.pares[length(n.pares)]  
[1] 8
```

```
#primeiro elemento  
> n.pares[1]  
[1] 2
```

- Subtraindo esses dois elementos, temos o alcance do vetor

```
> n.pares[length(n.pares)] - n.pares[1]  
[1] 6
```

# Operações com vetores

- Ainda em vetores numéricos, vamos calcular vetores com regras
- Selecionamos elementos maiores que 5

```
> n.pares[n.pares >= 5]  
[1] 6 8
```

- Queremos duplicar os valores inclusos dentro da condição

```
> n.pares[n.pares >= 5] * 2  
[1] 12 16
```

- Por fim, operacionar um conjunto com um elemento

```
> n.pares[n.pares >= 5] * n.pares[1]  
[1] 12 16
```

# Conjuntos

- Utilizamos a teoria de conjuntos no R com a função de manipular e identificar elementos comuns ou diferente entre vetores
- Já temos o vetor "n.pares" e vamos criar um vetor com números naturais

```
> n.naturais <- c(0,1,2,3,4,5,6,7,8,9)
```

```
> n.naturais  
[1] 0 1 2 3 4 5 6 7 8 9
```

- Se perguntarmos por valores que não estão no vetor "n.naturais", o resultado será conjunto vazio

```
> n.naturais[n.naturais == 10]  
numeric(0)
```

# Conjuntos

- Outro ponto importante é saber se os elementos estão contidos dentro de outro vetor
- A pergunta a se fazer é: os elementos do vetor "n.partidos" está contido no vetor "n.naturais"
- No R, contidos é igual a %in%

```
> n.pares %in% n.naturais  
[1] TRUE TRUE TRUE TRUE
```

- Isto é, o vetor "n.pares" está contido no vetor "n.naturais" e é um subconjunto
- Outra maneira, porém é utilizando a função is.element()

```
> is.element(n.pares,n.naturais)  
[1] TRUE TRUE TRUE TRUE
```

# Conjuntos

- Para encontrar os valores exclusivos de um conjunto, vamos utilizar a escrita da seleção de elementos
- Podemos incluir a notação de "!", que quer dizer diferente
- Portanto, selecionamos os valores que estão contido no conjunto dos naturais, porém não no conjunto dos valores pares

```
> n.naturais[!is.element(n.naturais,n.pares)]  
[1] 0 1 3 5 7 9
```

# Conjuntos

- Por fim, vamos testar se um valor está presente em um desses vetores
- Testaremos os valores 1, 11 e 21

```
> 1 %in% c(n.naturais,n.pares)
[1] TRUE
```

```
> 11 %in% c(n.naturais,n.pares)
[1] FALSE
```

```
> 21 %in% c(n.naturais,n.pares)
[1] FALSE
```

- Assim, não temos os valores 11 e 21 em nenhuma dos vetores, entendidos como conjuntos

# Escrita

- 1 Crie um vetor com o nome dos países da América Latina e chame de AL
- 2 Utilizando a função nchar(), que calcula o número de caracteres de um valor nominal, crie um vetor com o número de caracteres de cada país do vetor AL. Qual é o maior valor e o menor valor
- 3 Quantos países possuem mais de 6 caracteres no nome?
- 4 Crie um novo vetor a partir do vetor AL com países que não estão incluídos dentro do MERCOSUL
- 5 Qual é o somatório dos nchar() desse novo vetor

# Leitura

- Leia o script abaixo e comente com as hashtags o que cada código quer dizer:

```
#  
n.idade <- c(15,19,12,22,32,15)  
#  
n.idade [which.max(n.idade)]  
#  
class(n.idade)  
#  
n.idade[n.idade >= 18]  
#  
(n.idade [1] + n.idade [2])/2 == median(n.idade)
```

# Matrizes

- Nas duas primeiras aulas, construímos objetos unidimensionais, ou vetores
- Agora vamos trabalhar com matrizes que possuem duas dimensões: as colunas e as linhas
- Para criar, porém, uma matriz, podemos utilizar a função `matrix()`

```
#Definimos dois vetores:  
> n.idade <- c(25,28,27,30)  
> n.peso <- c(76,81,67,95)  
  
#Criamos a matrix  
> matrix(c(n.idade,n.peso),  
         ncol = 2, nrow = 4, byrow = F)
```

# Matrizes

```
> matrix(c(n.idade,n.peso),  
+ ncol = 2, nrow = 4, byrow = F)  
[,1] [,2]  
[1,] 25 76  
[2,] 28 81  
[3,] 27 67  
[4,] 30 95
```

- O primeiro argumento, que é primeiro valor inserido dentro da função, refere-se aos dados a serem convertidos em matriz, no caso os dois vetores
- Em seguida, nos argumentos 2 e 3 escolhemos o número de colunas e de linhas, respectivamente
- Por fim, no argumento 'byrow', decidimos que a matriz fosse construída pelas colunas. Por isso inserimos o F

# Operações com matrizes

- Após a construção da matriz, podemos salvá-la dentro de uma etiqueta
- Vamos chamar a matriz de a

```
> minha_matriz <- matrix(c(n.idade,n.peso),  
+ ncol = 2, nrow = 4, byrow = F)
```

```
> class(minha_matriz)  
[1] "matrix"
```

- Temos assim a matriz minha\_matriz e de classe "matrix"
- Isso significa que podemos fazer operações com esse objeto

# Operações com matrizes

- Podemos, por exemplo, transpor a matriz
- Isto é, uma matriz A é transposta quando as linhas se tornam colunas
- No R, a função `t()` é responsável pela transposição da matriz

```
> t(minha_matriz)
      [,1]  [,2]  [,3]  [,4]
[1,]    25    28    27    30
[2,]    76    81    67    95
```

- Uma das propriedades mais conhecidas da matriz transposta é que  $(A^t)^t$  é igual a matriz original A

```
> t(t(minha_matriz)) == minha_matriz
      [,1]  [,2]
[1,] TRUE TRUE
[2,] TRUE TRUE
[3,] TRUE TRUE
```

# Operações com matrizes

- Porém, a função mais importante é a de combinar matrizes
- Isso será também importante quando falarmos de base de dados
- Imagine uma outra matriz com 1 coluna e 4 linhas

```
> n.altura <- matrix(c(173,172,180,156),  
+ ncol = 1, nrow = 4)
```

- Com o objetivo de combinarmos duas matrizes pelas colunas, utilizaremos o comando cbind()

```
> cbind(minha_matriz,n.altura)  
      [,1] [,2] [,3]  
[1,]    25   76  173  
[2,]    28   81  172  
[3,]    27   67  180  
[4,]    30   95  156
```

# Operações com matrizes

- Combinaremos também linhas com a função rbind()

```
minha_obs <- matrix(c(29,75),  
ncol = 2, nrow = 1)
```

- Vamos combinar então a minha\_matriz com a nova matriz chamada minha\_obs

```
> rbind(minha_matriz,minha_obs)  
      [,1] [,2]  
[1,]    25   76  
[2,]    28   81  
[3,]    27   67  
[4,]    30   95  
[5,]    29   75
```

- rbind() e cbind() serão duas funções importantes na manipulação de dados

# Seleção de elementos

- Dessa vez, para selecionar elementos dentro da matriz, seguiremos os passos a seguir
  - Matriz;
  - [ , ];
  - Posição ou regra;
- A vírgula dentro do colchete separa duas informações essenciais para esse tipo de objeto, linha e coluna nessa ordem

```
> minha_matriz  
      [,1] [,2]  
[1,]   25   76  
[2,]   28   81  
[3,]   27   67  
[4,]   30   95
```

```
> minha_matriz[2,1]
```

# Seleção de elementos

- Assim como elementos de vetores, podemos fazer operações entre os elementos das matrizes
- podemos multiplicar o elemento da 1<sup>a</sup> coluna com um elemento da 2<sup>a</sup> coluna

```
> minha_matriz[1,1] * minha_matriz[3,2]  
[1] 1675
```

- Podemos verificar também se algum valor está incluído na matriz

```
> 21 %in% minha_matriz  
[1] FALSE
```

# Seleção de elementos

- O R entende o espaço vazio nessa notação como a seleção da linha ou coluna inteira
- Por exemplo, quero selecionar a linha 1 e 3 completas com as duas colunas

```
> minha_matriz[c(1,3),]  
      [,1] [,2]
```

```
[1,]    25    76  
[2,]    27    67
```

```
> minha_matriz[c(1,3),c(1,2)] ==  
minha_matriz[c(1,3),]  
      [,1] [,2]  
[1,] TRUE TRUE  
[2,] TRUE TRUE
```

# Seleção de elementos

- No caso para selecionar toda uma coluna, deixamos o espaço da linha vazio

```
> minha_matriz[,1]  
[1] 25 28 27 30
```

```
> minha_matriz[c(1:4),1] == minha_matriz[,1]  
[1] TRUE TRUE TRUE TRUE
```

- Essas seleções serão importantes para situações como o loop for e recodificação de variáveis
- Agora, podemos selecionar esses elementos dando nome às linhas e/ou às colunas

## Aula 2

Frederico Bertholini  
Álvaro João Pereira Filho

Curso de R  
12/06/2019

# Aula 3

Frederico Bertholini  
Álvaro João Pereira Filho

Curso de R  
14/06/2019



# Matrizes

- Nas duas primeiras aulas, construímos objetos unidimensionais, ou vetores
- Agora vamos trabalhar com matrizes que possuem duas dimensões: as colunas e as linhas
- Para criar, porém, uma matriz, podemos utilizar a função `matrix()`

```
#Definimos dois vetores:  
> n.idade <- c(25,28,27,30)  
> n.peso <- c(76,81,67,95)  
  
#Criamos a matrix  
> matrix(c(n.idade,n.peso),  
        ncol = 2, nrow = 4, byrow = F)
```

# Matrizes

```
> matrix(c(n.idade,n.peso),  
+ ncol = 2, nrow = 4, byrow = F)  
[,1] [,2]  
[1,] 25 76  
[2,] 28 81  
[3,] 27 67  
[4,] 30 95
```

- O primeiro argumento, que é primeiro valor inserido dentro da função, refere-se aos dados a serem convertidos em matriz, no caso os dois vetores
- Em seguida, nos argumentos 2 e 3 escolhemos o número de colunas e de linhas, respectivamente
- Por fim, no argumento 'byrow', decidimos que a matriz fosse construída pelas colunas. Por isso inserimos o F

# Operações com matrizes

- Após a construção da matriz, podemos salvá-la dentro de uma etiqueta
- Vamos chamar a matriz de a

```
> minha_matriz <- matrix(c(n.idade,n.peso),  
+ ncol = 2, nrow = 4, byrow = F)  
  
> class(minha_matriz)  
[1] "matrix"
```

- Temos assim a matriz minha\_matriz e de classe "matrix"
- Isso significa que podemos fazer operações com esse objeto

# Operações com matrizes

- Podemos, por exemplo, transpor a matriz
- Isto é, uma matriz A é transposta quando as linhas se tornam colunas
- No R, a função `t()` é responsável pela transposição da matriz

```
> t(minha_matriz)
      [,1]  [,2]  [,3]  [,4]
[1,]    25    28    27    30
[2,]    76    81    67    95
```

- Uma das propriedades mais conhecidas da matriz transposta é que  $(A^t)^t$  é igual a matriz original A

```
> t(t(minha_matriz)) == minha_matriz
      [,1]  [,2]
[1,] TRUE TRUE
[2,] TRUE TRUE
[3,] TRUE TRUE
```

# Operações com matrizes

- Porém, a função mais importante é a de combinar matrizes
- Isso será também importante quando falarmos de base de dados
- Imagine uma outra matriz com 1 coluna e 4 linhas

```
> n.altura <- matrix(c(173,172,180,156),  
+ ncol = 1, nrow = 4)
```

- Com o objetivo de combinarmos duas matrizes pelas colunas, utilizaremos o comando cbind()

```
> cbind(minha_matriz,n.altura)  
      [,1] [,2] [,3]  
[1,]    25   76  173  
[2,]    28   81  172  
[3,]    27   67  180  
[4,]    30   95  156
```

# Operações com matrizes

- Combinaremos também linhas com a função rbind()

```
minha_obs <- matrix(c(29,75),  
                      ncol = 2, nrow = 1)
```

- Vamos combinar então a minha\_matriz com a nova matriz chamada minha\_obs

```
> rbind(minha_matriz, minha_obs)  
      [,1] [,2]  
[1,]    25   76  
[2,]    28   81  
[3,]    27   67  
[4,]    30   95  
[5,]    29   75
```

- rbind() e cbind() serão duas funções importantes na manipulação de dados

# Seleção de elementos

- Dessa vez, para selecionar elementos dentro da matriz, seguiremos os passos a seguir
  - Matriz;
  - [,];
  - Posição ou regra;
- A vírgula dentro do colchete separa duas informações essenciais para esse tipo de objeto, linha e coluna nessa ordem

```
> minha_matriz  
      [,1] [,2]  
[1,]    25   76  
[2,]    28   81  
[3,]    27   67  
[4,]    30   95
```

```
> minha_matriz[2,1]
```

# Seleção de elementos

- Assim como elementos de vetores, podemos fazer operações entre os elementos das matrizes
- podemos multiplicar o elemento da 1<sup>a</sup> coluna com um elemento da 2<sup>a</sup> coluna

```
> minha_matriz[1,1] * minha_matriz[3,2]  
[1] 1675
```

- Podemos verificar também se algum valor está incluído na matriz

```
> 21 %in% minha_matriz  
[1] FALSE
```

# Seleção de elementos

- O R entende o espaço vazio nessa notação como a seleção da linha ou coluna inteira
- Por exemplo, quero selecionar a linha 1 e 3 completas com as duas colunas

```
> minha_matriz[c(1,3),]  
      [,1] [,2]  
[1,]    25    76  
[2,]    27    67  
  
> minha_matriz[c(1,3),c(1,2)] ==  
minha_matriz[c(1,3),]  
      [,1] [,2]  
[1,] TRUE TRUE  
[2,] TRUE TRUE
```

# Seleção de elementos

- No caso para selecionar toda uma coluna, deixamos o espaço da linha vazio

```
> minha_matriz[,1]  
[1] 25 28 27 30
```

```
> minha_matriz[c(1:4),1] == minha_matriz[,1]  
[1] TRUE TRUE TRUE TRUE
```

- Essas seleções serão importantes para situações como o loop for e recodificação de variáveis
- Agora, podemos selecionar esses elementos dando nome às linhas e/ou às colunas

# Nomeando linhas e colunas

- Para atribuímos nomes para colunas e linhas, temos que usar as funções `colnames()` e `rownames()`
- Intuitivamente, `colnames()` serve para listar como também para atribuir nome para as colunas

```
> colnames(minha_matriz)  
NULL
```

- Portanto, a matriz não possui nome na suas colunas
- Para isso, temos que definir um vetor com nomes e atribuir para o `colnames`

```
> colnames(minha_matriz) <- c("idade", "peso")
```

# Nomeando linhas e colunas

- As colunas, por outro lado, definimos como `rownames()` a função para listar e atribuir nome às linhas
- Faremos o mesmo procedimento de quando definimos o nome das colunas

```
> rownames(minha_matriz) <-  
  c("aluno1", "aluno2", "aluno3", "aluno4")
```

- Temos, portanto

```
> minha_matriz  
      idade peso  
aluno1    25   76  
aluno2    28   81  
aluno3    27   67  
aluno4    30   95
```

# Nomeando linhas e colunas

- Com nomes nas linhas e colunas, criamos mais uma opção de seleção de nomes e linhas
- Ao invés do número da coluna, podemos trocar pelo nome
- Assim como a linha, podemos trocar pelo nome da linha

```
> minha_matriz["aluno2", "idade"]  
[1] 28
```

```
> minha_matriz[c("aluno1", "aluno3"),]  
      idade peso  
aluno1     25    76  
aluno3     27    67
```

# Listas

- No R, listas são vetores genéricos contendo diversos objetos
- Podem obter objetos de diferentes classes

```
> idade <- c(45, 48, 39, 56, 23)
```

```
> mencoes <- c("MM", "MS", "SS", "MM", "MS")
```

```
> genero <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
```

- Unimos todos esses vetores dentro da função list()

```
> minha_lista <- list(idade, mencoes, genero)
```

# Listas

- Para acessarmos um pedaço da lista, basta usarmos um simples colchete '[ ]'
- Assim, listas são tratadas como simples vetores

```
> minha_lista[2]  
[[1]]  
[1] "MM" "MS" "SS" "MM" "MS"
```

- Para selecionar, porém, um elemento de dentro da lista, utilizamos colchete duplo '[[ ]]' e em seguida colchete simples '[ ]' com posição do elemento dentro do vetor

```
> minha_lista[[2]][2]  
[1] "MS"
```

# Nome nas listas

- Para dar nome aos componentes da lista, vamos refazer o comando `list()`

```
> minha_lista2 <-  
list(idade=c(45, 48, 39, 56, 23),  
mencoes=c("MM", "MS", "SS", "MM", "MS"),  
genero=c(TRUE, FALSE, TRUE, TRUE, FALSE))  
$ 'idade'  
[1] 45 48 39 56 23  
  
$mencoes  
[1] "MM" "MS" "SS" "MM" "MS"  
  
$genero  
[1] TRUE FALSE TRUE TRUE FALSE
```

# Nome nas listas

- Podemos nas listas com nomes selecionar de outra maneira
- Utilizando o nome dentro do colchete duplo, selecionamos o vetor específico

```
> minha_lista2[["idade"]]  
[1] 45 48 39 56 23
```

- Ou ainda podemos utilizar o operador \$, selecionando assim o vetor específico
- Veremos em data.frame que essa é uma boa maneira de selecionar variáveis, fique de olho!

```
> minha_lista2$idade  
[1] 45 48 39 56 23
```

# Nome nas listas

- Podemos trabalhar assim de diversas maneiras
- Por exemplo, definir idades maiores que 40 anos

```
> minha_lista2$idade[minha_lista2$idade > 40]  
[1] 45 48 56
```

- Definir notas igual ou maior do que MS

```
> minha_lista2$mencoes[minha_lista2$mencoes ==  
"MS" | minha_lista2$mencoes == "SS"]  
[1] "MS" "SS" "MS"
```

- E definir quantos homens tem na amostra

```
sum(minha_lista2$genero)  
[1] 3
```

## Textos em listas

- Funções simples na mineração de textos passam pelo conhecimento de listas
  - Primeira função básica é o número de caracteres, ou a função nchar()
  - Vamos imaginar primeiro o meu nome completo

```
> nome <- "Alvaro Joao Pereira Filho"
```

> nome

```
[1] "Alvaro Joao Pereira Filho"
```

- aplicamos a função e temos a contagem de caracteres utilizados para formação do meu nome

```
> nchar(nome)
```

[1] 25

# Textos em listas

- Contando rapidamente vemos que, na verdade, são apenas 22 caracteres
- Porém, nossa contagem apresenta 25 caracteres, logo a função nchar conta os espaços entre os nomes
- Vamos usar nosso conhecimento de listas e resolver esse problema
- Primeiro, utilizaremos a função strsplit() que reparte os nomes segundo algum critério localizado no segundo argumento, que é obrigatório

```
> nome2 <- strsplit(nome, split = " ")  
  
> nome2  
[[1]]  
[1] "Alvaro"    "Joao"       "Pereira"   "Filho"
```

# Textos em listas

- O objeto nome2 é uma lista

```
> class(nome2)  
[1] "list"
```

- Porém, queremos retirar dessa classe para unir os caracteres novamente, sem os espaços
- Utilizaremos a função unlist()

```
> nome3 <- unlist(nome2)
```

```
> class(nome3)  
[1] "character"
```

# Textos em listas

- Finalmente, para unir essas palavras do vetor em apenas 1, utilizamos a função `paste()`
- Ainda definimos no segundo argumento como uniremos todos os caracteres

```
> paste(nome3, collapse = "")  
[1] "AlvaroJoaopereiraFilho"
```

- Podemos contar quantos caracteres meu nome possui

```
> nchar(paste(nome3, collapse = ""))  
[1] 22
```

# **data.frame**

- Um **data.frame** é o mesmo que uma tabela do SQL ou uma planilha Excel
- seus dados provavelmente serão importados para um objeto **data.frame**
- **data.frame's** são listas especiais em que todos os elementos possuem o mesmo comprimento.
- Cada elemento dessa lista pode ser pensado como uma coluna da tabela - ou como uma variável.
- Seu comprimento representa o número de linhas - ou seja, de observações

# **data.frame**

- Como `data.frames`'s são listas, suas colunas podem ser de classes diferentes. Essa é a grande diferença entre `data.frame`'s e matrizes.

Funções úteis:

```
>head() # Mostra as primeiras 6 linhas.  
>tail() # Mostra as últimas 6 linhas.  
>dim() # Número de linhas e de colunas.  
>names() # Os nomes das colunas (variáveis).  
>str() # Estrutura do data.frame. Mostra, entre outras coisas:  
>cbind() # Acopla duas tabelas lado a lado.  
>rbind() # Empilha duas tabelas.
```

- Uma lista pode virar `data.frame` se todos os elementos tiverem o mesmo comprimento.

```
> as.data.frame(minha_lista)}
```

# Escrita

- 1 Crie uma matriz quadrada com os seguinte valores: 12, 8, 15; 1, 6, 20; 9, 0, 4. Defina sua montagem a fim de que  $a_{11}$  seja 12,  $a_{21}$  seja 8 e assim por diante
- 2 Faça a transposição dessa matriz
- 3 Sabendo que os valores apresentados são correspondente ao número de faltas em disciplinas, de o nome das linhas para cada aluno (invente nomes) e para as colunas o nome de disciplinas
- 4 Encontre o somatório de todas as faltas em cada disciplina da sua matriz

# Leitura

- Leia o script abaixo e comente com as hashtags o que cada código quer dizer:

```
#  
dados <-  
list(altura = c(1.72,1.90,1.56,1.72,1.80,1.65),  
peso=c(79,90,67,82,76,63),  
imc=peso/(altura)^2)  
#  
which.max(dados[["altura"]])  
#  
length(dados$imc[dados$imc <= 25])  
#  
unlist(dados)
```

# O que são pacotes

- O R possui diversas funções já instaladas dentro da sua programação
- Exemplos são `sum()`, `length()`, `class()`, `c()`
- Outras, porém, devem ser instaladas para que possam ser utilizadas pelos usuários
- A maneira com a qual instalamos novas funções, não definidas anteriormente no software, é através de pacotes
- Pacotes concentram diversas funções para diversas demandas
  - Importação de dados;
  - Organização de banco de dados;
  - Análises estatísticas específicas;
  - Gráficos diferenciados;

# O que são pacotes

- A instalação de qualquer pacote pode ser feita por dentro do R
- Para isso, porém, é preciso primeiro conexão com a internet, já que o R busca o novo pacote no repositório de pacotes
- A função para instalar pacote, portanto, é `install.packages()`
- O nome da nova função deve vir, primeiramente entre parenteses
- Podemos começar instalando o pacote para importação de bases de dados: `foreign`  

```
> install.packages("foreign")
```
- Após alguns segundos, e algumas mensagens no console, a instalação será efetivada

# Ativar pacotes

- Cada pacote, inclusive o foreign, tem uma documentação disponível na internet
- Nessa documentação estão disponíveis as funções que o pacote possui, além do nome do seu criador
- As função não ficam disponíveis assim que o pacote termina a instalação
- Para ativar as funções do pacote, é preciso utilizar a função `library()`

```
library(foreign)
```

- Repare, que uma vez instalado, o nome do pacote não precisa mais estar entre aspas

# Ativar pacotes

- Portanto, para começarmos o procedimento de instalação do pacote, seguimos os passos a seguir:
  - Caso não esteja instalado, instalar o pacote através da função `install.packages()`;
  - Para ativar o pacote, utilizar a função `library()` sem as aspas no nome do pacote instalado

```
install.packages("foreign")
```

```
library(foreign)
```

- Uma vez instalado o pacote, não é preciso instalar mais a não ser que você reinstale o R

# Importação de dados - Passo a passo

- A importação é uma das tarefas que demandam mais atenção no R
  - É preciso ter um conhecimento prévio de como sua base externa está constituída
  - Outra informação importante é a extensão do arquivo da base
  - Primeiramente, a informação que deve ser dada ao software é onde está a base - diretório de trabalho
  - A função necessária é setwd() que define o diretório da sua seção no R
  - Dentro da função, iremos inserir o local do arquivo
    - Em caso de Windows, inverta as barras ou duplique;
    - Não se esqueça das aspas;
- ```
> setwd("C:/Users/alvar/Documents/  
Database_jobs/Denilson")
```

# Passo a passo

- Esse diretório definido significa que as bases e os gráficos produzidos serão enviados para essa pasta
- Finalmente, vamos importar as bases de dados
- Primeiro, vamos importar a base de extensão txt com o nome baserm
- Não é preciso de pacote para esse procedimento

```
lines <- readLines("baserm.txt")
```

```
baserm <- read.table(text = lines, sep = '\t')
```

- Repare que definimos a base dentro das aspas e com a extensão
- Na segunda linha, o primeiro argumento é o texto, o segundo argumento trata de como os dados estão separados, geralmente txt vem separado assim

# Passo a passo

- Vamos importar uma base CSV, separado por ponto e vírgula
- Seleccionaremos a base de educacao.csv
- Primeiro procedimento é definir qual é a separação e, também, se a base possui na primeira linha o nome das variáveis

```
> educacao <- read.csv("educacao.csv", header = T,  
sep = ",")
```

- No primeiro argumento, a base de dados presente na pasta
- O segundo argumento, header=, define se tem ou não cabeçalho de variáveis na base. No caso tem
- Por fim, o separador da base está por vírgulas, por isso usamos o argumento sep= e entre aspas o separador

# Passo a passo

- No pacote foreign, a forma mais genérica de importação da base é o `read.table()`
- Entretanto, o pacote apresenta uma série de especialidades, a depender da extensão em questão
- Para CSV, vimos que tem a `read.csv()`. Já para dta, base de origem do stata, temos a função `read.dta()`
- O pacote foreign não possui a extensão `xlsx` e `xls`, extensão muito encontrada e comum entre as bases de dados disponíveis
- Para isso, vamos instalar um novo pacote `readxl`  
`> install.packages("readxl")`
- Esse pacote disponibiliza as funções `read_xls()` e `read_xlsx()`

# Passo a passo

- Vamos ativar as funções disponíveis no pacote `readxl` com a função `library()`

```
> library(readxl)
```

- Vamos importar a base `controle_cgumunicípios.xlsx`

```
> cgu <-  
read_xlsx("controle_cgumunicípios.xlsx")
```

- Repare que acessamos apenas a primeira página da base

- Para acessarmos a segunda páginas, utilizamos o argumento `sheet=`

```
> cgu <-  
read_xlsx("controle_cgumunicípios.xlsx",  
sheet = 2)
```

- Alguns sinais de alerta surgem, porém não se trata de erro

# Importação por pacote

- Por fim, podemos importar dados através de pacotes
- Após ativar o pacote "PNADcIBGE", a função `get_pnadc()` fica disponível para a importação
- Como toda a função, ou quase todas, `get_pnadc()` possui argumentos importantes:
- 'year =' se refere ao ano de extração do PNAD
- 'quarter =' se refere ao trimestre de extração do PNAD
- 'design =' é um arg lógico para retornar o objeto com a configuração do pacote 'survey', porém, aqui colocaremos FALSE nesse arg para utilizarmos maior número de funções já existente no próprio R

# Importação do PNAD

- Vamos começar importando o PNAD do 1º trimestre de 2018, nos retornando um objeto em '*tbl\_df*' e data frame

```
> pnad2018 <- get_pnadc(year = 2018,  
                           quarter = 1, design = F)
```

- Provavelmente, demorará alguns segundos, até minutos, para a importação
- Além disso, essa técnica demanda acesso à internet

# Visualizando a base

- Primeira coisa importante de se informar é a classe desses objetos
- Temos 4 objetos: baserm, cgu, educacao e pnad2018

```
> class(baserm)
[1] "data.frame"

> class(cgu)
[1] "tbl_df"      "tbl"          "data.frame"

> class(educacao)
[1] "data.frame"

> class(pnad2018)
[1] "tbl_df"      "tbl"          "data.frame"
```

# Visualizando a base

- Uma visão completa da base é o comando `View()`
- Entretanto, cuidado, dependendo do tamanho da base, podemos travar o software
  - > `View(baserm)`
- Repare no V maiúsculo, lembre-se que o R é bastante sensível na sua linguagem
- O `View()` abre uma nova aba com a base no formato de grade
- Podemos, assim, visualizar a base de dados na forma mais intuitiva

# Visualizando a base

- Porém, para bases como a pnad2018, por exemplo, sabemos que é grande demais para sua visualização ser feita através do `View()`
- Algumas funções podem nos ajudar nessa tarefa
- A primeira é o `dim`, que as dimensões da base

```
> dim(pnad2018)
[1] 560741      216
```
- O primeiro valor sempre retrata o número de linhas, ou observações, enquanto o segundo valor apresenta o número de colunas, ou variáveis
- Portanto, a base do pnad2018 que extraímos possui mais de 560000 observações e pouco mais de 200 variáveis
- A função `ncol()` e `length()` também indicam quantas colunas, ou variáveis estão presentes na base

# Visualizando a base

- Outra função importante na visualização de bases de dados é a lista de nomes
- A função `names()` descreve as variáveis presentes na base
- Isso facilita no momento de selecionar as variáveis que entrarão na análise de vocês

```
> names(educacao)
[1] "X"   "COD.IBGE7"  "REGI_O"  "UF"    "PORTE"
[6] "MUNICPIO" "falha.em.educacao" "ANO_FALHA"
[11] "serv_per_cap_med" "Conselho.edu.cria_o"
[16] "NEP" "Ideologia" "Arrec_Prop" "PIBpercapita"
[21] "tm" "PERCENT_ganhador" "TaxadeUrbanizacao"
[26] "Sudeste" "Nordeste" "Norte" "POP" "LN_POP"
[31] "RM"
```

# Visualizando a base

- Outra função possível é o str()
- Essa função apresenta o nome das variáveis, a classe de cada uma delas e os primeiros valores

```
Classes      tbl _ d f ,      tbl      and 'data.frame':
56 variables:
 $ COD.IBGE7           : num  1100015 1100023 1
 $ REGI 0                : chr  "1\u2014Nort" "1\u2014\u2014"
 $ UF                   : chr  "11" "11" "11" "11"
 $ PORTE                : chr  "4\u2014\u201420001\u2014at \u20145"
 $ MUNIC PIO             : chr  "Alta\u2014Floresta\u2014D"
 $ falha                : num  1 1 0 1 1 1 1 1 0
 $ tempo_falha_02        : num  0 1 13 9 0 2 4 5
 $ tempo_falha_01        : num  1 2 14 10 1 3 5 6
 $ reicid ncia_falha    : num  3 3 0 1 4 3 1 2
 $ ano_eleitoral          : num  0 0 0 0 0 1 0 0 0
 $ PERCENT ganhador 2000 : chr  "0.4094630000000000"
```

# Visualizando a base

- Finalmente, a função `head()` e `tail()`
- A primeira função apresenta os primeiros valores de uma base de dados

```
> head(baserm, 2)
```

- O segundo argumento serve para indicar quantas linhas serão apresentadas
- `Tail()`, por outro lado, apresenta os últimos valores de uma base de dados

```
> tail(baserm, 4)
```

# Escrita

- 1 Acesse o site <http://dados.gov.br/> e selecione 5 bases
- 2 Defina um diretório comum a todas essas bases
- 3 Importe essas bases corretamente
- 4 Quais são as classes de cada base? E quais são as dimensões de cada uma?
- 5 Insira todas as 5 bases dentro de um objeto list

# Leitura

- Leia o script abaixo e comente com as hashtags o que cada código quer dizer:

```
#  
setwd("C:/Users/alvaro/Documentos/Curso_de_R")  
#  
detalhes <- readLines("base_secreta.TXT")  
#  
dados <- read.table(text = detalhes, sep = "\t")  
#  
dados <- subset(dados, dados$paises == "Argentina")
```

# Passo a passo

- No pacote foreign, a forma mais genérica de importação da base é o `read.table()`
- Entretanto, o pacote apresenta uma série de especialidades, a depender da extensão em questão
- Para CSV, vimos que tem a `read.csv()`. Já para dta, base de origem do stata, temos a função `read.dta()`
- O pacote foreign não possui a extensão `xlsx` e `xls`, extensão muito encontrada e comum entre as bases de dados disponíveis
- Para isso, vamos instalar um novo pacote `readxl`  
`> install.packages("readxl")`
- Esse pacote disponibiliza as funções `read_xls()` e `read_xlsx()`

# Passo a passo

- Vamos ativar as funções disponíveis no pacote `readxl` com a função `library()`

```
> library(readxl)
```

- Vamos importar a base `controle_cgumunicípios.xlsx`

```
> cgu <-  
read_xlsx("controle_cgumunicípios.xlsx")
```

- Repare que acessamos apenas a primeira página da base

- Para acessarmos a segunda páginas, utilizamos o argumento `sheet=`

```
> cgu <-  
read_xlsx("controle_cgumunicípios.xlsx",  
sheet = 2)
```

- Alguns sinais de alerta surgem, porém não se trata de erro

# Importação por pacote

- Por fim, podemos importar dados através de pacotes
- Após ativar o pacote "PNADcIBGE", a função `get_pnadc()` fica disponível para a importação
- Como toda a função, ou quase todas, `get_pnadc()` possui argumentos importantes:
- 'year =' se refere ao ano de extração do PNAD
- 'quarter =' se refere ao trimestre de extração do PNAD
- 'design =' é um arg lógico para retornar o objeto com a configuração do pacote 'survey', porém, aqui colocaremos FALSE nesse arg para utilizarmos maior número de funções já existente no próprio R

# Importação do PNAD

- Vamos começar importando o PNAD do 1º trimestre de 2018, nos retornando um objeto em '*tbl\_df*' e data frame

```
> pnad2018 <- get_pnadc(year = 2018,  
                           quarter = 1, design = F)
```

- Provavelmente, demorará alguns segundos, até minutos, para a importação
- Além disso, essa técnica demanda acesso à internet

# Visualizando a base

- Primeira coisa importante de se informar é a classe desses objetos
- Temos 4 objetos: baserm, cgu, educacao e pnad2018

```
> class(baserm)
[1] "data.frame"

> class(cgu)
[1] "tbl_df"      "tbl"          "data.frame"

> class(educacao)
[1] "data.frame"

> class(pnad2018)
[1] "tbl_df"      "tbl"          "data.frame"
```

# Visualizando a base

- Uma visão completa da base é o comando `View()`
- Entretanto, cuidado, dependendo do tamanho da base, podemos travar o software
  - > `View(baserm)`
- Repare no V maiúsculo, lembre-se que o R é bastante sensível na sua linguagem
- O `View()` abre uma nova aba com a base no formato de grade
- Podemos, assim, visualizar a base de dados na forma mais intuitiva

# Visualizando a base

- Porém, para bases como a pnad2018, por exemplo, sabemos que é grande demais para sua visualização ser feita através do `View()`
- Algumas funções podem nos ajudar nessa tarefa
- A primeira é o `dim`, que as dimensões da base

```
> dim(pnad2018)
[1] 560741      216
```
- O primeiro valor sempre retrata o número de linhas, ou observações, enquanto o segundo valor apresenta o número de colunas, ou variáveis
- Portanto, a base do pnad2018 que extraímos possui mais de 560000 observações e pouco mais de 200 variáveis
- A função `ncol()` e `length()` também indicam quantas colunas, ou variáveis estão presentes na base

# Visualizando a base

- Outra função importante na visualização de bases de dados é a lista de nomes
- A função `names()` descreve as variáveis presentes na base
- Isso facilita no momento de selecionar as variáveis que entrarão na análise de vocês

```
> names(educacao)
[1] "X"   "COD.IBGE7"  "REGI_O"  "UF"    "PORTE"
[6] "MUNICPIO" "falha.em.educacao" "ANO_FALHA"
[11] "serv_per_cap_med" "Conselho.edu.cria_o"
[16] "NEP" "Ideologia" "Arrec_Prop" "PIBpercapita"
[21] "tm" "PERCENT_ganhador" "TaxadeUrbanizacao"
[26] "Sudeste" "Nordeste" "Norte" "POP" "LN_POP"
[31] "RM"
```

# Visualizando a base

- Outra função possível é o str()
- Essa função apresenta o nome das variáveis, a classe de cada uma delas e os primeiros valores

```
Classes      tbl _ d f ,      tbl      and 'data.frame':
56 variables:
 $ COD.IBGE7           : num  1100015 1100023 1
 $ REGI 0                : chr  "1\u2014Nort" "1\u2014\u2014"
 $ UF                   : chr  "11" "11" "11" "11"
 $ PORTE                : chr  "4\u2014\u201420001\u2014at \u20145"
 $ MUNIC PIO             : chr  "Alta\u2014Floresta\u2014D"
 $ falha                : num  1 1 0 1 1 1 1 1 0
 $ tempo_falha_02        : num  0 1 13 9 0 2 4 5
 $ tempo_falha_01        : num  1 2 14 10 1 3 5 6
 $ reicid ncia_falha    : num  3 3 0 1 4 3 1 2
 $ ano_eleitoral         : num  0 0 0 0 0 1 0 0 0
 $ PERCENT ganhador 2000 : chr  "0.4094630000000000"
```

# Visualizando a base

- Finalmente, a função `head()` e `tail()`
- A primeira função apresenta os primeiros valores de uma base de dados

```
> head(baserm, 2)
```

- O segundo argumento serve para indicar quantas linhas serão apresentadas
- `Tail()`, por outro lado, apresenta os últimos valores de uma base de dados

```
> tail(baserm, 4)
```

# Escrita

- 1 Acesse o site <http://dados.gov.br/> e selecione 3 bases
- 2 Defina um diretório comum a todas essas bases
- 3 Importe essas bases corretamente
- 4 Quais são as classes de cada base? E quais são as dimensões de cada uma?
- 5 Insira todas as 3 bases dentro de um objeto list

# Leitura

- Leia o script abaixo e comente com as hashtags o que cada código quer dizer:

```
#  
setwd("C:/Users/alvaro/Documentos/Curso_de_R")  
#  
detalhes <- readLines("base_secreta.TXT")  
#  
dados <- read.table(text = detalhes, sep = "\t")  
#  
dados <-  
subset(dados, dados$paises == "Argentina")
```

# Aula 3

Frederico Bertholini  
Álvaro João Pereira Filho

Curso de R  
14/06/2019

# Aula 4

Frederico Bertholini  
Álvaro João Pereira Filho

Curso de R  
17/06/2019



LAPCIPP

Laboratório de Pesquisa em  
Comportamento Político,  
Instituições e Políticas Públicas

# Visualizando a base

- Primeira coisa importante de se informar é a classe desses objetos
- Temos 4 objetos: baserm, cgu, educacao e pnad2018

```
> class(baserm)
[1] "data.frame"

> class(cgu)
[1] "tbl_df"     "tbl"        "data.frame"

> class(educacao)
[1] "data.frame"

> class(pnad2018)
[1] "tbl_df"     "tbl"        "data.frame"
```

# Visualizando a base

- Uma visão completa da base é o comando `View()`
- Entretanto, cuidado, dependendo do tamanho da base, podemos travar o software
  - > `View(baserm)`
- Repare no V maiúsculo, lembre-se que o R é bastante sensível na sua linguagem
- O `View()` abre uma nova aba com a base no formato de grade
- Podemos, assim, visualizar a base de dados na forma mais intuitiva

# Visualizando a base

- Porém, para bases como a pnad2018, por exemplo, sabemos que é grande demais para sua visualização ser feita através do `View()`
- Algumas funções podem nos ajudar nessa tarefa
- A primeira é o `dim`, que as dimensões da base

```
> dim(pnad2018)
[1] 560741      216
```
- O primeiro valor sempre retrata o número de linhas, ou observações, enquanto o segundo valor apresenta o número de colunas, ou variáveis
- Portanto, a base do pnad2018 que extraímos possui mais de 560000 observações e pouco mais de 200 variáveis
- A função `ncol()` e `length()` também indicam quantas colunas, ou variáveis estão presentes na base

# Visualizando a base

- Outra função importante na visualização de bases de dados é a lista de nomes
- A função `names()` descreve as variáveis presentes na base
- Isso facilita no momento de selecionar as variáveis que entrarão na análise de vocês

```
> names(educacao)
[1] "X"   "COD.IBGE7"  "REGI_O"  "UF"    "PORTE"
[6] "MUNICPIO" "falha.em.educacao" "ANO_FALHA"
[11] "serv_per_cap_med" "Conselho.edu.cria_o"
[16] "NEP" "Ideologia" "Arrec_Prop" "PIBpercapita"
[21] "tm" "PERCENT_ganhador" "TaxadeUrbanizacao"
[26] "Sudeste" "Nordeste" "Norte" "POP" "LN_POP"
[31] "RM"
```

# Visualizando a base

- Outra função possível é o str()
- Essa função apresenta o nome das variáveis, a classe de cada uma delas e os primeiros valores

```
Classes      tbl _ d f ,      tbl      and 'data.frame':  
56 variables:  
 $ COD.IBGE7          : num  1100015 1100023 1  
 $ REGI 0              : chr  "1\u2014Nort" "1\u2014\u2014"  
 $ UF                 : chr  "11" "11" "11" "11"  
 $ PORTE              : chr  "4\u201420001\u2014at \u20145"  
 $ MUNIC PIO           : chr  "Alta\u2014Floresta\u2014D"  
 $ falha               : num  1 1 0 1 1 1 1 1 0  
 $ tempo_falha_02       : num  0 1 13 9 0 2 4 5  
 $ tempo_falha_01       : num  1 2 14 10 1 3 5 6  
 $ reicid ncia_falha    : num  3 3 0 1 4 3 1 2  
 $ ano_eleitoral        : num  0 0 0 0 0 1 0 0 0  
 $ PERCENT ganhador 2000 : chr  "0.4094630000000000"
```

# Visualizando a base

- Finalmente, a função `head()` e `tail()`
- A primeira função apresenta os primeiros valores de uma base de dados
  - > `head(baserm, 2)`
- O segundo argumento serve para indicar quantas linhas serão apresentadas
- `Tail()`, por outro lado, apresenta os últimos valores de uma base de dados
  - > `tail(baserm, 4)`

# Escrita

- 1 Acesse o site <http://dados.gov.br/> e selecione 5 bases
- 2 Defina um diretório comum a todas essas bases
- 3 Importe essas bases corretamente
- 4 Quais são as classes de cada base? E quais são as dimensões de cada uma?
- 5 Insira todas as 5 bases dentro de um objeto list

# Leitura

- Leia o script abaixo e comente com as hashtags o que cada código quer dizer:

```
#  
setwd("C:/Users/alvaro/Documentos/Curso_de_R")  
#  
detalhes <- readLines("base_secreta.TXT")  
#  
dados <- read.table(text = detalhes, sep = "\t")  
#  
dados <- subset(dados, dados$paises == "Argentina")
```

# Passo a passo

- No pacote foreign, a forma mais genérica de importação da base é o `read.table()`
- Entretanto, o pacote apresenta uma série de especialidades, a depender da extensão em questão
- Para CSV, vimos que tem a `read.csv()`. Já para dta, base de origem do stata, temos a função `read.dta()`
- O pacote foreign não possui a extensão `xlsx` e `xls`, extensão muito encontrada e comum entre as bases de dados disponíveis
- Para isso, vamos instalar um novo pacote `readxl`  
`> install.packages("readxl")`
- Esse pacote disponibiliza as funções `read_xls()` e `read_xlsx()`

# Passo a passo

- Vamos ativar as funções disponíveis no pacote `readxl` com a função `library()`

```
> library(readxl)
```

- Vamos importar a base `controle_cgumunicípios.xlsx`

```
> cgu <-  
read_xlsx("controle_cgumunicípios.xlsx")
```

- Repare que acessamos apenas a primeira página da base

- Para acessarmos a segunda páginas, utilizamos o argumento `sheet=`

```
> cgu <-  
read_xlsx("controle_cgumunicípios.xlsx",  
sheet = 2)
```

- Alguns sinais de alerta surgem, porém não se trata de erro

# Importação por pacote

- Por fim, podemos importar dados através de pacotes
- Após ativar o pacote "PNADclIBGE", a função *get\_pnadc()* fica disponível para a importação
- Como toda a função, ou quase todas, *get\_pnadc()* possui argumentos importantes:
- 'year =' se refere ao ano de extração do PNAD
- 'quarter =' se refere ao trimestre de extração do PNAD
- 'design =' é um arg lógico para retornar o objeto com a configuração do pacote 'survey', porém, aqui colocaremos FALSE nesse arg para utilizarmos maior número de funções já existente no próprio R

# Importação do PNAD

- Vamos começar importando o PNAD do 1º trimestre de 2018, nos retornando um objeto em '*tbl\_df*' e data frame

```
> pnad2018 <- get_pnadc(year = 2018,  
                           quarter = 1, design = F)
```

- Provavelmente, demorará alguns segundos, até minutos, para a importação
- Além disso, essa técnica demanda acesso à internet

# Visualizando a base

- Primeira coisa importante de se informar é a classe desses objetos
- Temos 4 objetos: baserm, cgu, educacao e pnad2018

```
> class(baserm)
[1] "data.frame"

> class(cgu)
[1] "tbl_df"     "tbl"        "data.frame"

> class(educacao)
[1] "data.frame"

> class(pnad2018)
[1] "tbl_df"     "tbl"        "data.frame"
```

# Visualizando a base

- Uma visão completa da base é o comando `View()`
- Entretanto, cuidado, dependendo do tamanho da base, podemos travar o software
  - > `View(baserm)`
- Repare no V maiúsculo, lembre-se que o R é bastante sensível na sua linguagem
- O `View()` abre uma nova aba com a base no formato de grade
- Podemos, assim, visualizar a base de dados na forma mais intuitiva

# Visualizando a base

- Porém, para bases como a pnad2018, por exemplo, sabemos que é grande demais para sua visualização ser feita através do `View()`
- Algumas funções podem nos ajudar nessa tarefa
- A primeira é o `dim`, que as dimensões da base

```
> dim(pnad2018)
[1] 560741      216
```
- O primeiro valor sempre retrata o número de linhas, ou observações, enquanto o segundo valor apresenta o número de colunas, ou variáveis
- Portanto, a base do pnad2018 que extraímos possui mais de 560000 observações e pouco mais de 200 variáveis
- A função `ncol()` e `length()` também indicam quantas colunas, ou variáveis estão presentes na base

# Visualizando a base

- Outra função importante na visualização de bases de dados é a lista de nomes
- A função `names()` descreve as variáveis presentes na base
- Isso facilita no momento de selecionar as variáveis que entrarão na análise de vocês

```
> names(educacao)
[1] "X"   "COD.IBGE7"  "REGI_O"  "UF"    "PORTE"
[6] "MUNICPIO" "falha.em.educacao" "ANO_FALHA"
[11] "serv_per_cap_med" "Conselho.edu.cria_o"
[16] "NEP" "Ideologia" "Arrec_Prop" "PIBpercapita"
[21] "tm" "PERCENT_ganhador" "TaxadeUrbanizacao"
[26] "Sudeste" "Nordeste" "Norte" "POP" "LN_POP"
[31] "RM"
```

# Visualizando a base

- Outra função possível é o str()
- Essa função apresenta o nome das variáveis, a classe de cada uma delas e os primeiros valores

```
Classes      tbl _ d f ,      tbl      and 'data.frame':  
56 variables:  
 $ COD.IBGE7          : num  1100015 1100023 1  
 $ REGI 0              : chr  "1\u2014Nort" "1\u2014\u2014"  
 $ UF                 : chr  "11" "11" "11" "11"  
 $ PORTE              : chr  "4\u201420001\u2014at \u20145"  
 $ MUNIC PIO           : chr  "Alta\u2014Floresta\u2014D"  
 $ falha               : num  1 1 0 1 1 1 1 1 0  
 $ tempo_falha_02       : num  0 1 13 9 0 2 4 5  
 $ tempo_falha_01       : num  1 2 14 10 1 3 5 6  
 $ reicid ncia_falha    : num  3 3 0 1 4 3 1 2  
 $ ano_eleitoral        : num  0 0 0 0 0 1 0 0 0  
 $ PERCENT ganhador 2000 : chr  "0.4094630000000000"
```

# Visualizando a base

- Finalmente, a função `head()` e `tail()`
- A primeira função apresenta os primeiros valores de uma base de dados
  - > `head(baserm, 2)`
- O segundo argumento serve para indicar quantas linhas serão apresentadas
- `Tail()`, por outro lado, apresenta os últimos valores de uma base de dados
  - > `tail(baserm, 4)`

# Escrita

- 1 Acesse o site <http://dados.gov.br/> e selecione 5 bases
- 2 Defina um diretório comum a todas essas bases
- 3 Importe essas bases corretamente
- 4 Quais são as classes de cada base? E quais são as dimensões de cada uma?
- 5 Insira todas as 5 bases dentro de um objeto list

# Leitura

- Leia o script abaixo e comente com as hashtags o que cada código quer dizer:

```
#  
setwd("C:/Users/alvaro/Documentos/Curso_de_R")  
#  
detalhes <- readLines("base_secreta.TXT")  
#  
dados <- read.table(text = detalhes, sep = "\t")  
#  
dados <-  
subset(dados, dados$paises == "Argentina")
```

# Aula 4

Frederico Bertholini  
Álvaro João Pereira Filho

Curso de R  
17/06/2019

# Introdução à Manipulação

- Vamos manipular as base de dados e também revisar tudo que aprendemos até aqui
  - Melhor ainda, vamos usar uma base comum para o CNPq
  - Primeiro, vamos limpar no enviroment
  - Usaremos a função rm() de remove
  - Definimos, também, os objetos do enviroment que queremos excluir no argumento list =
  - Para excluir totalmente, usamos ls(), que é o código do enviroment
- ```
> rm(list = ls())
```

# Introdução à Manipulação

- Agora vamos importar a planilha universal de indicadores
- O que temos que saber, primeiramente, é a extensão dessa base
- Trata-se de uma base em CSV, separado em ponto e vírgula
- Portanto, primeiro vamos ativar o pacote do foreign, que permite a importação da base em CSV
  - > `library(foreign)`
- Agora sim, temos a função `read.csv` disponível

# Introdução à Manipulação

- Definimos assim o diretório de trabalho através do setwd()
- Lá deve estar a base que iremos trabalhar
- Lembre-se de inverter ou duplicar a barra, caso seja usuário do Windows
  - > `setwd("C:\\\\Users\\\\alvar\\\\Documents\\\\R\\\\cnpq")`
- Não esqueça das aspas entre o local do arquivo
- Uma possibilidade de encontrar o local é acessando a propriedade do arquivo, com o botão direito do mouse, e copiar o local que aparece na janela da propriedade

# Introdução à Manipulação

- Finalmente, vamos importar a base
- Sabendo que é um CSV, vamos definir um separador pelo argumento `sep =`
- Nesse caso, a separação está em ponto e vírgula

```
cnpq  
-> read.csv  
("Planilha_Universal_2018_indicadores.csv",  
header = T, sep = ";", stringsAsFactors = F)
```

- O argumento `header =` significa que a primeira linha é o cabeçalho de variáveis
- E o argumento `stringAsFactors = F` significa que as variáveis nominais serão importadas em `character`, não é `factor`

# Introdução à Manipulação

- Pronto, a base está no importada na forma que queremos
- Algumas informações são importantes

```
> class(cnpq)
[1] "data.frame"
```

```
> dim(cnpq)
[1] 46288    117
```

- As funções names(), str(), head() e tail() também vão nos ajudar a pré-visualizar a base de dados

# Frequências

- A função mais importante no momento em que se busca explorar cada variável é a função `table()`
- A função `table()` é responsável por criar uma tabela de frequência de uma variável
- Por exemplo, queremos saber quantas observações estão distribuídas na variável modalidade

```
> table(cnpq$MODALIDADE)
APQ
46288
```

- A função `table()` talvez funcione bem para variáveis discretas, ou `character` e `lógica`, mas pouco faz sentido para variáveis contínuas

# Frequências

- Variáveis dicotômicas também funcionam bem com `table()`
- Por exemplo, vamos calcular a frequência da variável Líder do grupo de pesquisa

```
> table(cnpq$LIDER_GRUPO_PESQUISA)
      Sim
 27554 18734
```

- No caso, 18734 pesquisadores são líderes de grupo de pesquisa
- Apesar de estar em branco, 27554 não são líderes

# Frequências

- Com `table()` ainda podemos fazer frequência cruzada entre variáveis
- Ou seja, a frequência entre duas variáveis de uma base de dados

```
> table(cnpq$NIVEL_ATUAL_PQ ,  
cnpq$LIDER_GRUPO_PESQUISA)
```

	Sim	
23132	11448	
1A	136	562
1B	178	562
1C	272	764
1D	558	1236
2	3268	4132
SR	10	30

# Frequências

- Para variáveis contínuas, a forma de captar as frequências é através da função `summary`
- Essa função apresenta valores como:
  - Valores mínimo e máximo;
  - Distribuição dos quartis;
  - Média e mediana;
- Melhor exemplo é utilizar a variável `Artigos_total`

```
> summary(cnpq$ARTIGOS_TOTAL)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.00	8.00	16.00	23.91	30.00	814.00

# Novas colunas

- A seleção de variáveis pode ser feita pelo operador \$
- Podemos usar esse operador, também, nos casos de criar novas variáveis com os valores de outras variáveis do banco
- Para isso, criamos um novo nome para a coluna, colocando no formato de seleção de variável

```
> cnpq$y <- rep(c(0,1), nrow(cnpq)/2)
```

```
> table(cnpq$y)
```

0	1
23144	23144

- A função rep() funciona para repetir os valores 0 e 1
- O segundo argumento o número de vezes que cada um desses valores deveria repetir

# Novas colunas

- Quando olhamos para a distribuição do total de artigos, vemos um pesquisador com 814 artigos publicados
- A pergunta que surge é: será que esse tem publicações em A1?
- Podemos olhar diretamente quantas publicações em A1 esse indivíduo possui
- Selecionando a variável Art\_qualis\_A1, queremos ver selecionar com a função which() o indivíduo com 814 artigos publicados

```
> cnpq$ART_QUALIS_A1  
[which(cnpq$ARTIGOS_TOTAL == 814)]  
[1] 0
```

- A função which() funciona para detectar em qual posição o pesquisador com 814 publicações está na variável

# Novas colunas

- Se quisermos ver todas as suas características, já sabemos como fazer

```
cnpq [which(cnpq$ARTIGOS_TOTAL == 814),]
```

- Assim, disponibilizamos a linha completa dessa observação
- Para apenas 1 informação, ou variável, basta mudarmos o nome que precede o colchete, pela seleção de uma variável

```
> cnpq$AREA_CONHECIMENTO  
[which(cnpq$ARTIGOS_TOTAL == 814)]  
[1] "Sistemas\ude\Computa\ u008do"
```

# Novas colunas

- Vamos aumentar a complexidade
- Imagina que queremos tirar uma média ponderada entre a qualis de cada publicação que cada pesquisador teve
- Daremos peso maior para A1 e menor para Nulo
- Como o cálculo será feito para cada valor, usaremos looping

```
for(i in 1:nrow(cnpq)){
  wt <- c(8:1)
  x <- c(cnpq$ART_QUALIS_A1[i],
  cnpq$ART_QUALIS_A2[i],cnpq$ART_QUALIS_B1[i],
  cnpq$ART_QUALIS_B2[i],cnpq$ART_QUALIS_B3[i],
  cnpq$ART_QUALIS_B4[i],cnpq$ART_QUALIS_B5[i],
  cnpq$ART_QUALIS_C[i])
  cnpq$media_ponderada[i] <- weighted.mean(x,wt)
}
```

# Novas colunas

- Vendo a distribuição dessa nova variável, temos:

```
> summary(cnpq$media_ponderada2)
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.0000 0.7778 1.8889 3.0533 3.8611 115.6389
```

- Mais importante é que nosso colega com 814 publicações, adquiriu uma média ponderada de apenas 33

```
> cnpq$media_ponderada
[which(cnpq$ARTIGOS_TOTAL == 814)]
[1] 32.77778
```

# Subset

- Uma importante função em manipulação de base de dados é `subset()`
- Com essa função, pode selecionar ou definir partes de uma base maior
- Por exemplo, retornando à variável `y`, que distribuiu valores 0 e 1
- Podemos ver que ela pode separar os valores duplicados da variável `Cod_rh`

```
> length(unique(cnpq$COD_RH)) ==  
length(cnpq$COD_RH)/2  
[1] TRUE
```

# Subset

- Para selecionar apenas as observações com o registro de 5 anos, que estão codificados como 0, iremos realizar um `subset()`
- Primeiro argumento é a base que iremos recortar, o segundo argumento o critério de recorte

```
> cnpq.mod <- subset(cnpq, cnpq$y == 0)
```

```
> dim(cnpq.mod)
[1] 23144    120
```

```
> dim(cnpq)
[1] 46288    120
```

# Subset

- Podemos selecionar a partir da função subset algumas variáveis de interesse
- Essa programação é importante para a construção de gráficos
- Vamos selecionar apenas três variáveis: Cod\_rh, Artigos\_totais e Art\_Primeiro\_Autor
- Dentro da função não é preciso informar de qual base as variáveis estão saindo, vocês já informaram no primeiro argumento

```
> cnpq.mod2 <- subset(cnpq.mod, select =  
+ c(COD_RH, ARTIGOS_TOTAL, ART_PRIMEIRO_AUTOR))  
  
> names(cnpq.mod2)  
[1] "COD_RH" "ARTIGOS_TOTAL"  
[3] "ART_PRIMEIRO_AUTOR"
```

# Subset

- Se fizermos outros subset(), selecionando outras duas variáveis mais o Cod\_rh, que é único no cnpq.mod
- Vamos selecionar: Cod\_rh, Area\_conhecimento e data\_primeiro\_pq

```
> cnpq.mod3 <- subset(cnpq.mod, select =  
+ c(COD_RH, AREA_CONHECIMENTO, DATA_PRIMEIRO_PQ))
```

- Podemos realizar o que chamamos de merge, ou a fundição de dois data.frame a partir de valores de uma ou mais variáveis

```
cnpq.final <- merge(cnpq.mod2, cnpq.mod3,  
by = "COD_RH")
```

- Onde o argumento by= define qual é a chave-fechadura entre as duas bases

# Aula 4

Frederico Bertholini  
Álvaro João Pereira Filho

Curso de R  
17/06/2019

# Aula 5

Frederico Bertholini  
Álvaro João Pereira Filho

Curso de R  
19/06/2019



# Recodificação

- Na recodificar variáveis, usaremos os operadores de seleção de elementos juntamente com operadores lógicos
- Vamos começar adicionando o valor "Não" na variável LIDER\_GRUPO\_PESQUISA

```
> table(cnpq$LIDER_GRUPO_PESQUISA)
      Sim
 27554 18734
```

- O objetivo é inserir o character "Nao" para o espaço sem valor

```
cnpq$LIDER_GRUPO_PESQUISA
[cnpq$LIDER_GRUPO_PESQUISA == ""] <- "Nao"
```

# Recodificação

- A recodificação de variáveis, portanto, funciona com a seleção das variáveis
- Com o detalhe da atribuição do novo valor
- Conferindo o novo valor

```
> table(cnpq$LIDER_GRUPO_PESQUISA)
  Nao    Sim
  27554 18734
```

- Porém, diversas vezes teremos que atribuir valores às variáveis nominais
- O objetivo aqui é atribuir o valor 1 a "Sim" e 0 ao "Nao"

# Recodificação

- O primeiro passo é substituir a classe da informação

```
> class(cnpq$LIDER_GRUPO_PESQUISA)  
[1] "character"
```

- Em seguida, vamos atribuir os novos valores que queremos

```
> cnpq$LIDER_GRUPO_PESQUISA2 <- cnpq$LIDER_GRUPO_P
```

```
> cnpq$LIDER_GRUPO_PESQUISA2  
[cnpq$LIDER_GRUPO_PESQUISA2 == "Nao"] <- 0  
> cnpq$LIDER_GRUPO_PESQUISA2  
[cnpq$LIDER_GRUPO_PESQUISA2 == "Sim"] <- 1
```

- Se consultarmos a tabela de frequência, temos

```
> table(cnpq$LIDER_GRUPO_PESQUISA2)  
0 1  
27554 18734
```

# Recodificação

- Todavia, a variável continua como caracter
- Isto é, a variável permanece com aspas e não é possível fazer cálculos matemáticos com ela

```
> class(cnpq$LIDER_GRUPO_PESQUISA2)
[1] "character"
```

- Para mudar para numérico, que é a classe que desejamos, iremos usar a função `as.numeric()`
- sim, existem `as.character()` e `as.logical()`, cada um para transformar para a sua especificidade

```
cnpq$LIDER_GRUPO_PESQUISA
<- as.numeric(cnpq$LIDER_GRUPO_PESQUISA2)
```

# Recodificação

- Vamos recodificar agrupando valores para a variável de artigos totais
- Até o 1º quartil, vamos atribuir o valor 1, do primeiro quartil até a mediana 2, da mediana até o 3º quartil 3 e, por fim, 3º quartil até o máximo atribuiremos o valor 4

```
> summary(cnpq$ARTIGOS_TOTAL)
Min. 1st Qu. Median      Mean 3rd Qu.      Max.
0.00     8.00    16.00    23.91   30.00   814.00
```

- Para isso, precisaremos utilizar regras

# Recodificação

```
> cnpq$ARTIGOS_TOTAL2 <- cnpq$ARTIGOS_TOTAL  
  
> cnpq$ARTIGOS_TOTAL2  
[cnpq$ARTIGOS_TOTAL2 <= 8] <- 1  
> cnpq$ARTIGOS_TOTAL2[cnpq$ARTIGOS_TOTAL2 > 8  
& cnpq$ARTIGOS_TOTAL2 <= 16 ] <- 2  
> cnpq$ARTIGOS_TOTAL2[cnpq$ARTIGOS_TOTAL2 > 16  
& cnpq$ARTIGOS_TOTAL2 <= 30 ] <- 3  
> cnpq$ARTIGOS_TOTAL2  
[cnpq$ARTIGOS_TOTAL2 > 30] <- 4
```

- Temos que

```
> table(cnpq$ARTIGOS_TOTAL2)  
1 2 3 4  
12572 11231 11127 11358
```

# Outras maneiras

- Porém, como tudo no R, há outras maneiras de se recodificar as variáveis
- Algumas dessas de forma muito mais eficiente
- A função `ifelse` é muito mais eficiente para recodificar variáveis dicotômicas

```
> m <- median(cnpq$ART_JCR_MAIOR_IGUAL_6)

> cnpq$ART_JCR_MAIOR_IGUAL_6.MOD
<- ifelse(cnpq$ART_JCR_MAIOR_IGUAL_6 > m, 1, 0)

> table(cnpq$ART_JCR_MAIOR_IGUAL_6.MOD)
  0      1
37274  9014
```

# Outras maneiras

- Para variáveis contínuas, podemos fazer de moda tradicional
- Ou utilizando a função `cut()`

```
> cnpq$ARTIGOS_TOTAL3 <-  
  cut(cnpq$ARTIGOS_TOTAL,  
       breaks = c(-Inf, 8, 16, 30, Inf),  
       labels = c(1, 2, 3, 4))
```

- Podemos definir os parametros através do argumento `breaks=` e os novos valores nos `labels=`

```
> table(cnpq$ARTIGOS_TOTAL3)  
 1      2      3      4  
12572 11231 11127 11358
```

# Escrita

- 1 Faça um subsetting da Área de conhecimento de Políticas Públicas
- 2 Selecione algumas variáveis, entre elas Livros, Artigos qualis A1 e Termo da primeira graduação
- 3 Quantos livros tem o pesquisador que graduou a mais tempo?
- 4 Quantos artigos A1 possui o graduado a menos tempo?
- 5 Recodifique a variável tempo da graduação em que quem se graduou depois de 2010 tenha o valor de "Recente" e quem tem o ano igual ou antes de 2010 de "Antigo"

# Gráficos

- Os Gráficos em R podem ser os melhores e são os objetos mais complexos produzidos no software
- Existem duas possibilidade de produção de gráficos no R
  - Utilizando os comandos básicos ou a função `qplot()`
  - Utilizando um pacote `ggplot2` e sua função `ggplot()`
- Aqui! Utilizaremos o pacote `ggplot2`

# Estrutura

- Na aula de gráficos de hoje, vamos apresentar os principais tipos de gráficos que temos aqui no R
- Sim! Temos vários, mas esses são os tipos mais encontrados em revistas e artigos
- Vamos começar com a estrutura básica das funções de construção de gráfico

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

```
> library(ggplot2)
```

# Estrutura

- Primeira parte na função do `ggplot()` é a definição do arg da base de dados ou vetor e dos eixos x e y que irão compor o gráfico
- Vamos definir a fonte e as variáveis que serão representadas no nosso gráfico

```
ggplot(dados, aes(x = x, y = y))
```

- A função `aes()` é a responsável por definir os eixos do gráfico, primeiro o eixo x e depois o y

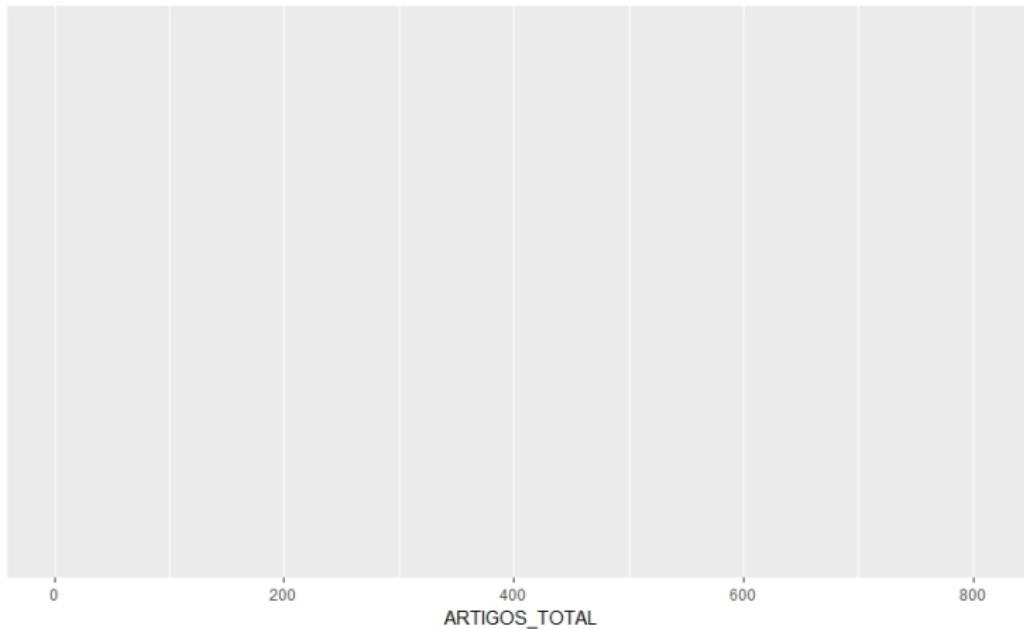
# Estrutura

- Primeira parte na função do ggplot() é a definição do dados e dos eixos x e y do gráfico
- Vamos definir a fonte e as variáveis que serão representadas no gráfico
  - > `ggplot(cnpq, aes())`
- A função aes() é a responsável por definir os eixos do gráfico, primeiro o eixo x e depois o y

# Estrutura

- Primeira parte na função do ggplot() é a definição do data e dos eixos x e y do gráfico
- Vamos definir a fonte e as variáveis que serão representadas no gráfico
- A função aes() é a responsável por definir os eixos do gráfico,  
> `ggplot(cnpq ,aes(x = ARTIGOS_TOTAL))`

# Estrutura

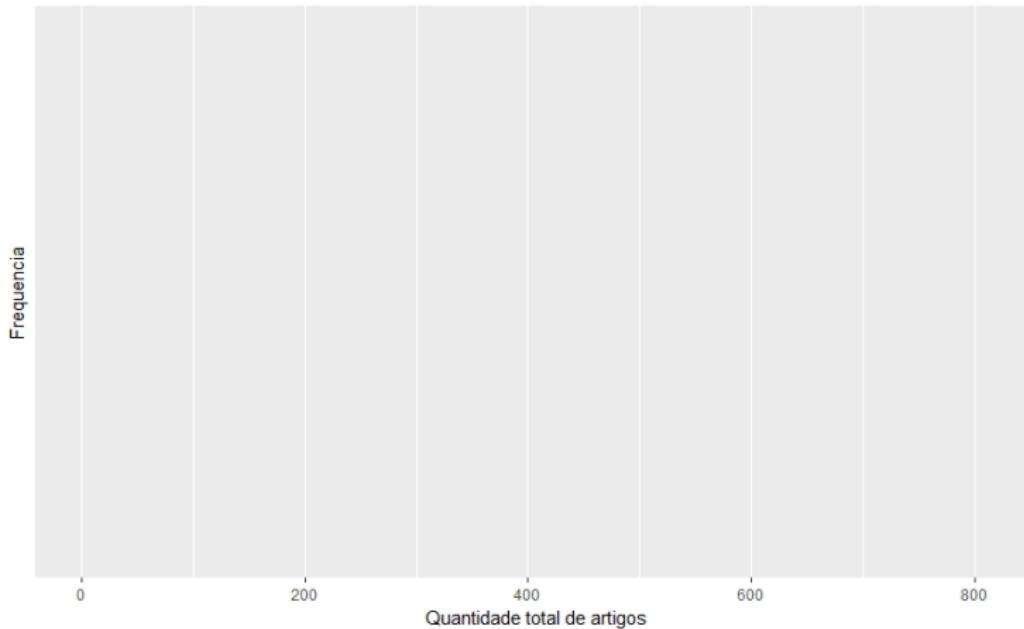


# Estrutura

- Vamos definir os nomes dos eixos através das funções `xlab()` e `ylab()`;
- A escrita do pacote segue a ordem de função aglutinadas pelo operador `'+'`

```
ggplot(cnpq,aes(x = ARTIGOS_TOTAL)) +  
  xlab("Total de artigos") +  
  ylab("Frequencia")
```

# Estrutura

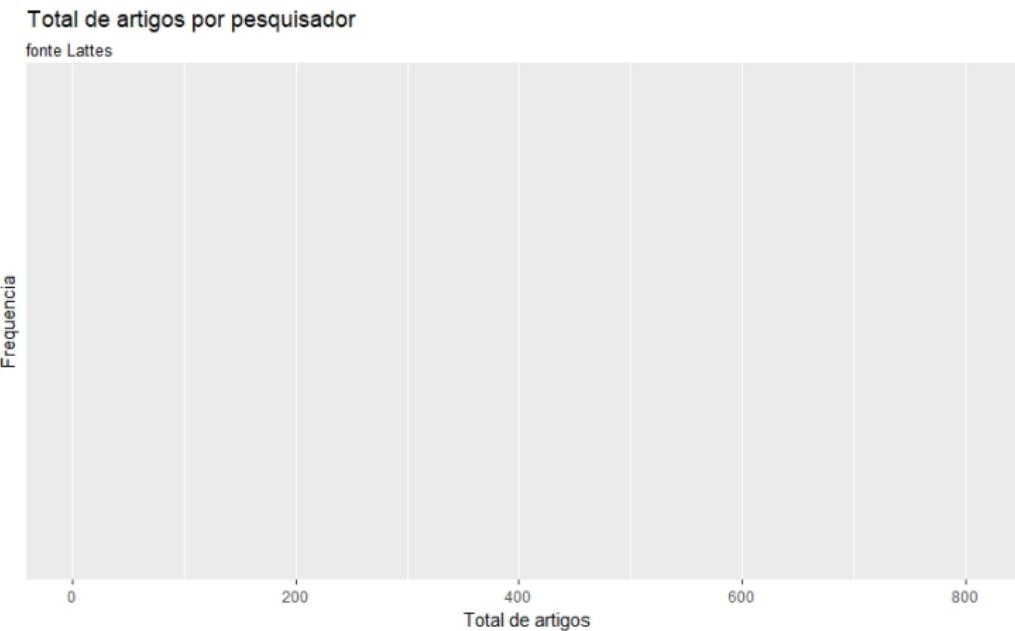


# Estrutura

- Ainda definimos o título do gráfico, além do subtítulo, através da função `ggtitle()`
- Para escrever o subtítulo, ou legenda do título, basta ajustar o argumento `subtitle =`

```
ggplot(cnpq,aes(x = ARTIGOS_TOTAL)) +  
  xlab("Total de artigos") +  
  ylab("Frequencia") +  
  ggtitle("Total de artigos por pesquisador",  
          subtitle = "fonte Lattes")
```

# Estrutura



# Tipos de gráficos

- Alguns tipos de gráficos são os mais comuns e básicos dentro da produção científica
- O `ggplot2` possui uma lista gigantesca de tipos de gráficos
- Aqui vamos apresentar os principais
- Começando pelo histograma, que apresenta a distribuição de uma variável contínua

# Histograma

- Para esse tipo de gráfico selecionaremos a já trabalhada variável running.time
- Primeiro, construímos a estrutura básica

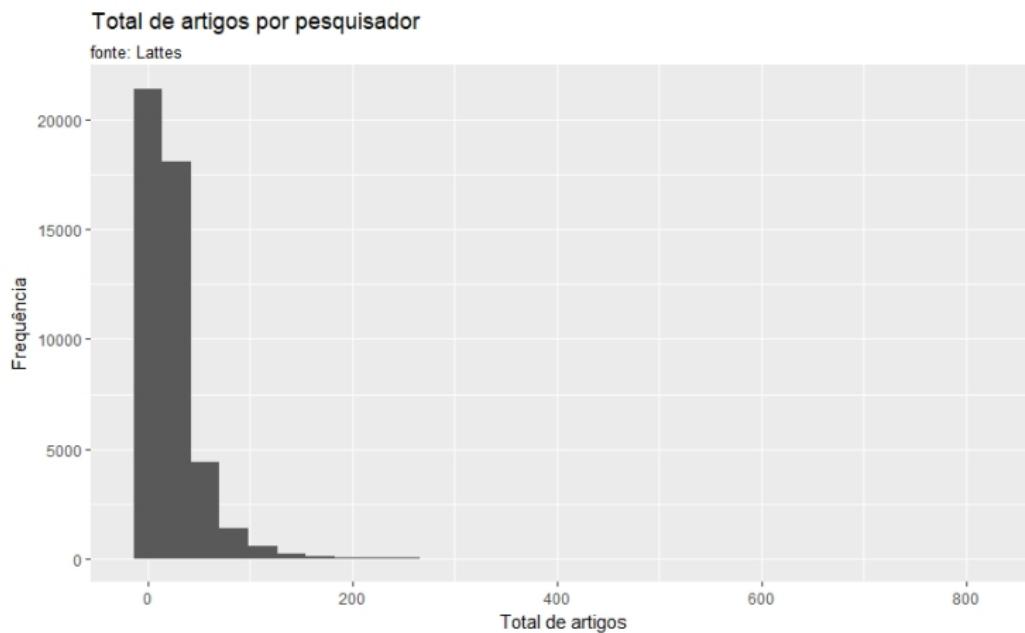
```
> ggplot(cnpq, aes(x = ARTIGOS_TOTAL))
```

# Histograma

- Em seguida, adicionamos o tipo de gráfico com a função `geom_histogram()`
- O argumento `binwidth` ajusta o tamanho do silos, que já vem com o tamanho 30

```
> ggplot(cnpq, aes(x = ARTIGO_TOTAL)) +  
  geom_histogram() +  
  xlab("Total de artigos") +  
  ylab("Frequência") +  
  ggtitle("Total de artigos por pesquisador",  
          subtitle = "fonte: Lattes")
```

# Histograma

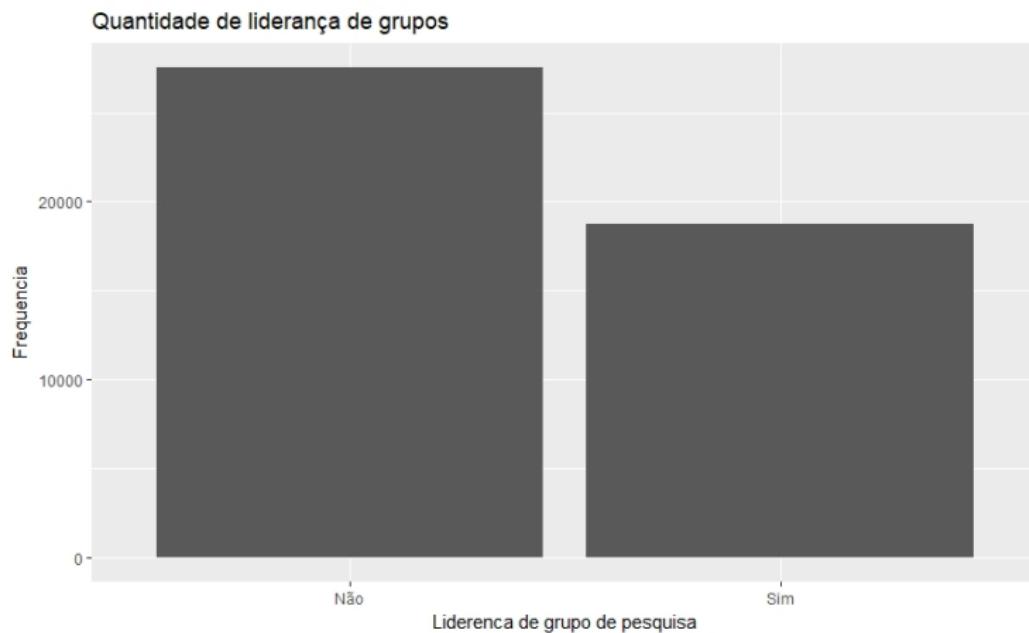


# Barras

- Para o gráfico de barras, com o x categórica e o y contínua, vamos alterar a função `geom_histograma()` por `geom_bar()`
- No `geom_bar()` é preciso estar atento à demanda pelo argumento `stat = 'identity'`. Quando são duas variáveis, é necessário a inclusão desse argumento, porém, no caso de representar frequência, não

```
> ggplot(cnpq, aes(x = LIDER_GRUPO_PESQUISA)) +  
  geom_bar() +  
  xlab("Liderança de grupo de pesquisa") +  
  ylab("Frequência") +  
  ggtitle("Quantidade de líderes a de grupos")
```

# Barras

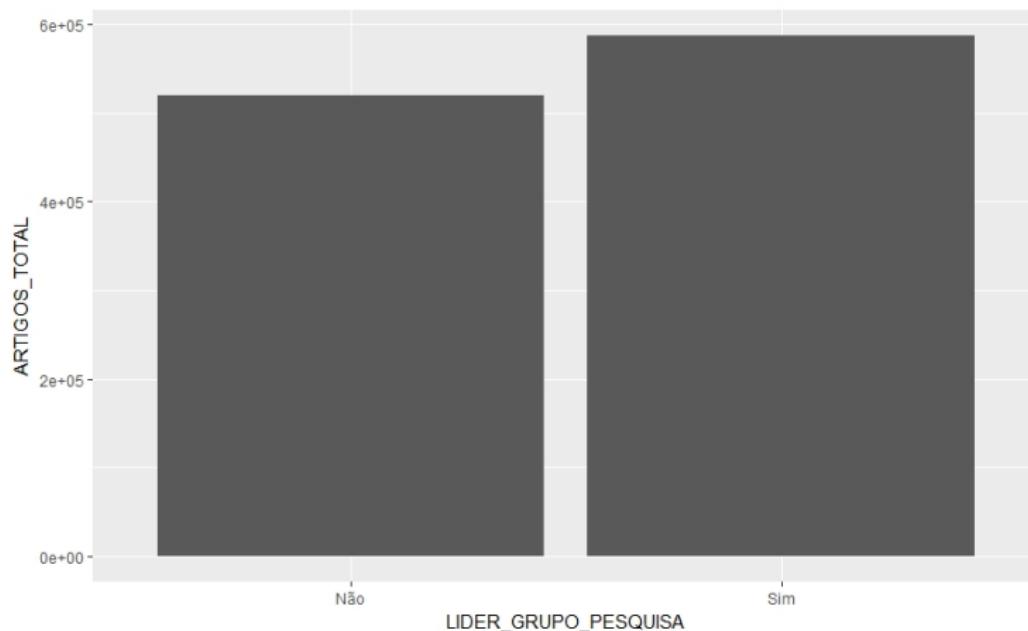


# Barras

- No caso de duas variáveis, novamente, temos que inserir na função `geom_bar()`, o arg `stat = "identity"`

```
> ggplot(cnpq, aes(x = LIDER_GRUPO_PESQUISA,  
y=ARTIGOS_TOTAL))  
+ geom_bar(stat = "identity")
```

# Barras

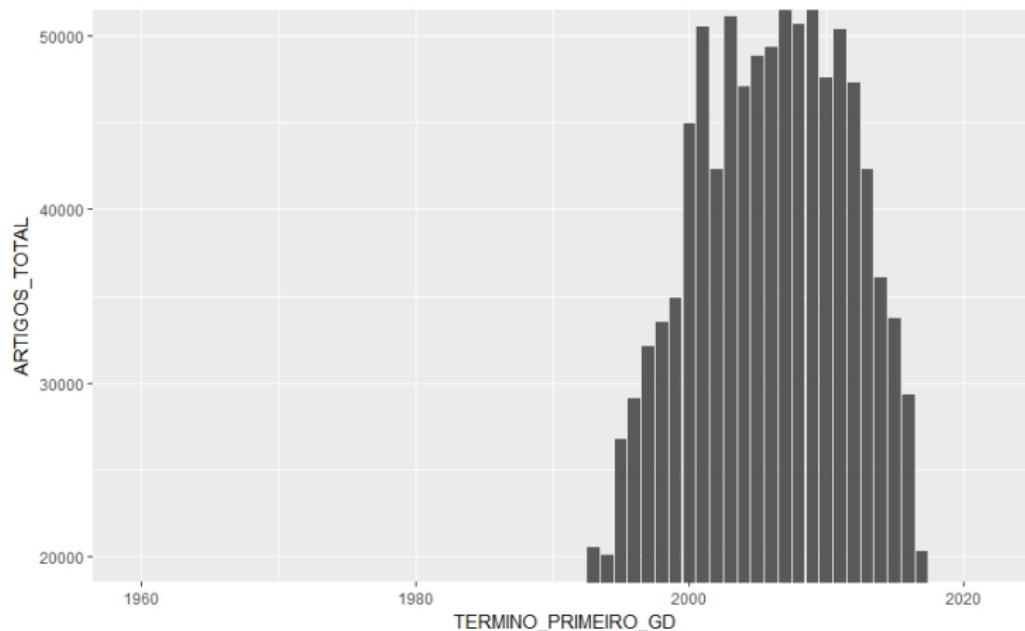


# Barras

- Vamos redefinir a escala de x e y
- No caso do y, vamos limitar os valores a partir de 55 até 80
- Existem algumas funções que desempenham esse papel de recortar a escala, usaremos o *coord\_cartesian()* definindo o *ylim = c(min,max)*

```
ggplot(cnpq,aes(x = TERMINO_PRIMEIRO_GD,  
y=ARTIGOS_TOTAL)) +  
  geom_bar(stat = "identity") +  
  coord_cartesian(ylim = c(20000,50000))
```

# Barras

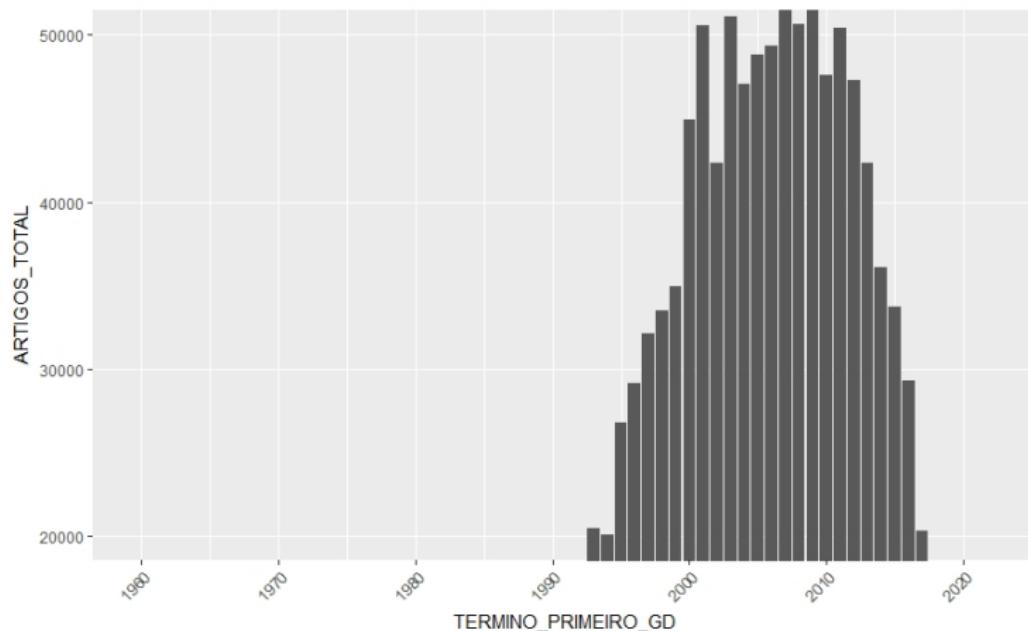


# Barras

- Para inserirmos todos os valores nos 'sticks', usaremos a função `scale_x_continuous()`
- O argumento que usaremos é de 'breaks =', portanto

```
> ggplot(cnpq, aes(x=TERMINO_PRIMEIRO_GD,  
y=ARTIGOS_TOTAL)) +  
  geom_bar(stat = "identity") +  
  coord_cartesian(ylim = c(20000,50000)) +  
  scale_x_continuous(breaks =  
    c(1960,1970,1980,1990,2000,2010,2020)) +  
  theme(axis.text.x =  
    element_text(angle=45,hjust = 1,vjust = 1),  
    panel.grid.minor.y = element_blank())
```

# Barras



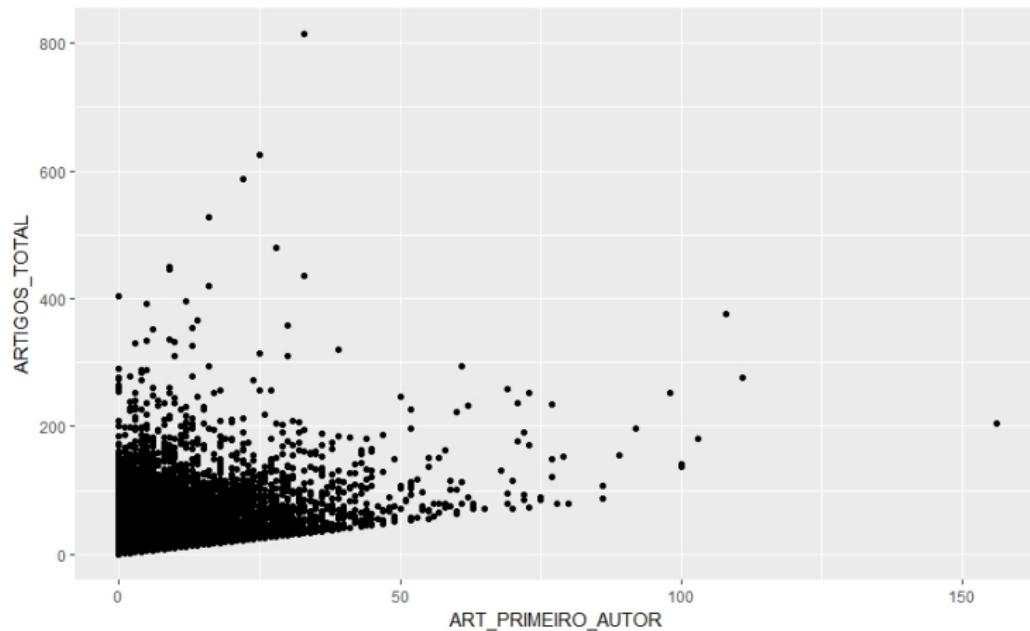
# Gráficos de pontos

- Para representar a correlação entre duas variáveis, usamos o gráficos de pontos
- Podemos ver se há correlação entre o número de artigos e o número de artigos como primeiro autor

```
> cor(cnpq$ARTIGOS_TOTAL,  
cnpq$ART_PRIMEIRO_AUTOR)  
[1] 0.3879819
```

```
ggplot(cnpq,aes(x = ART_PRIMEIRO_AUTOR,  
y = ARTIGOS_TOTAL))  
+ geom_point()
```

# Gráficos de pontos

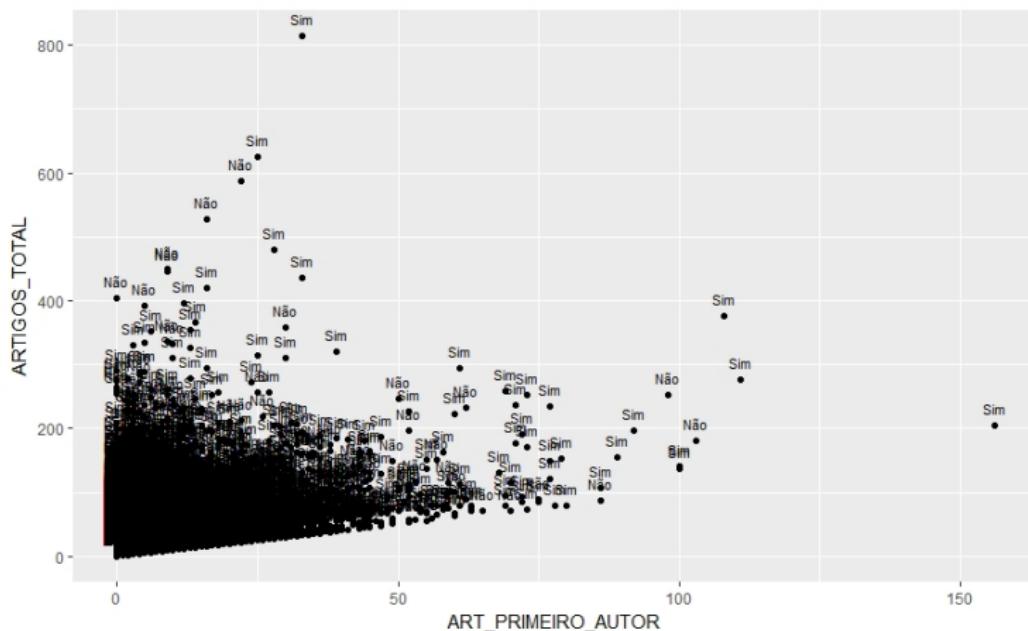


# Gráficos de pontos

- Vamos dividir essas observações por liderança de grupo de pesquisa:
- Podemos, nesse caso, plotar a legenda dentro do gráfico com a função *geom\_text*, em seguida ajustando a posição dessa legenda

```
ggplot(cnpq, aes(x = ART_PRIMEIRO_AUTOR,  
y = ARTIGOS_TOTAL))  
+ geom_point() +  
geom_text(aes(label = LIDER_GRUPO_PESQUISA),  
size = 3, vjust = -1)
```

# Gráficos de pontos

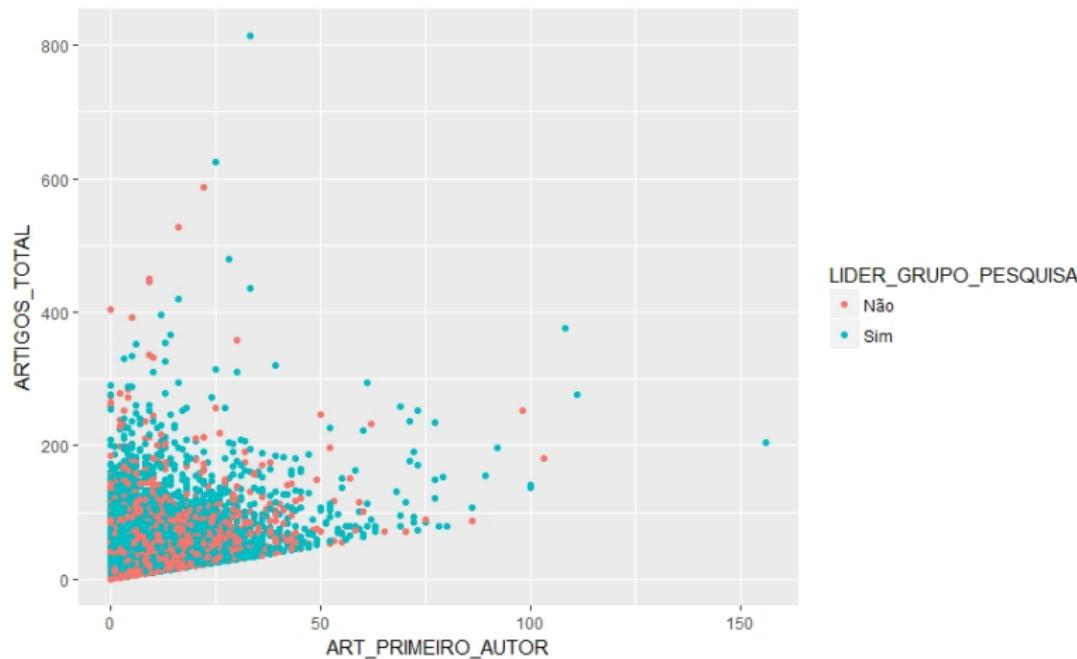


# Gráficos de pontos

- Ou então, podemos dividir os pontos em cores diferentes, um para cada categoria
- Utilizaremos dentro do arg 'aes()' um sub-argumento 'colour ='

```
ggplot(cnpq,aes(x = ART_PRIMEIRO_AUTOR, y = ARTIGO  
colour = LIDER_GRUPO_PESQUISA)) +  
geom_point()
```

# Gráficos de pontos



# Aula 5

Frederico Bertholini  
Álvaro João Pereira Filho

Curso de R  
19/06/2019

# Aula 6

Álvaro João Pereira Filho

Curso de R  
21/06/2019



# Gráfico de linhas

- Gráficos de linhas podem ser construídos a partir de variáveis discretas no x e contínua no y, ou entre contínuas.

```
cnpq.mod <- subset(cnpq, cnpq$REF_ANOS == 5 &  
cnpq$TERMINO_PRIMEIRO_GD >= 2015)
```

- Quero, porém, gerar uma pequena tabela com o somatório dos artigos por ano do término do primeiro doutorado

```
x <- list(a = cnpq.mod$ARTIGOS_TOTAL  
[which(cnpq.mod$TERMINO_PRIMEIRO_GD == 2015)],  
           b = cnpq.mod$ARTIGOS_TOTAL  
[which(cnpq.mod$TERMINO_PRIMEIRO_GD == 2016)],  
           c = cnpq.mod$ARTIGOS_TOTAL  
[which(cnpq.mod$TERMINO_PRIMEIRO_GD == 2017)],  
           d = cnpq.mod$ARTIGOS_TOTAL  
[which(cnpq.mod$TERMINO_PRIMEIRO_GD == 2018)])
```

# Gráfico de linhas

- Gerada a lista com os vetores, usaremos a função `sapply()`, que opera funções entre as listas

```
> somatorio <- sapply(x, sum)
```

```
> ano <- c(2015:2018)
```

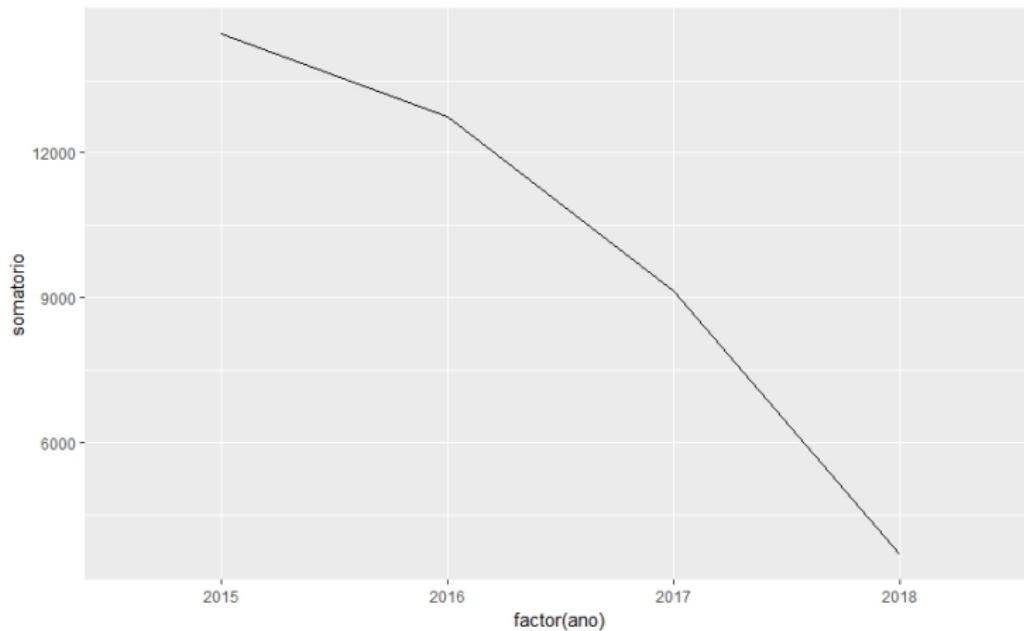
```
> base <- data.frame(ano, somatorio)
```

- Ao final, temos um `data.frame` com duas variáveis: somatório e ano

```
ggplot(base, aes(x = factor(ano), y = somatorio,  
group = 1)) + geom_line()
```

- Utilizamos o argumento `group = 1` para unir os pontos que serão plotados

# Gráficos de linhas



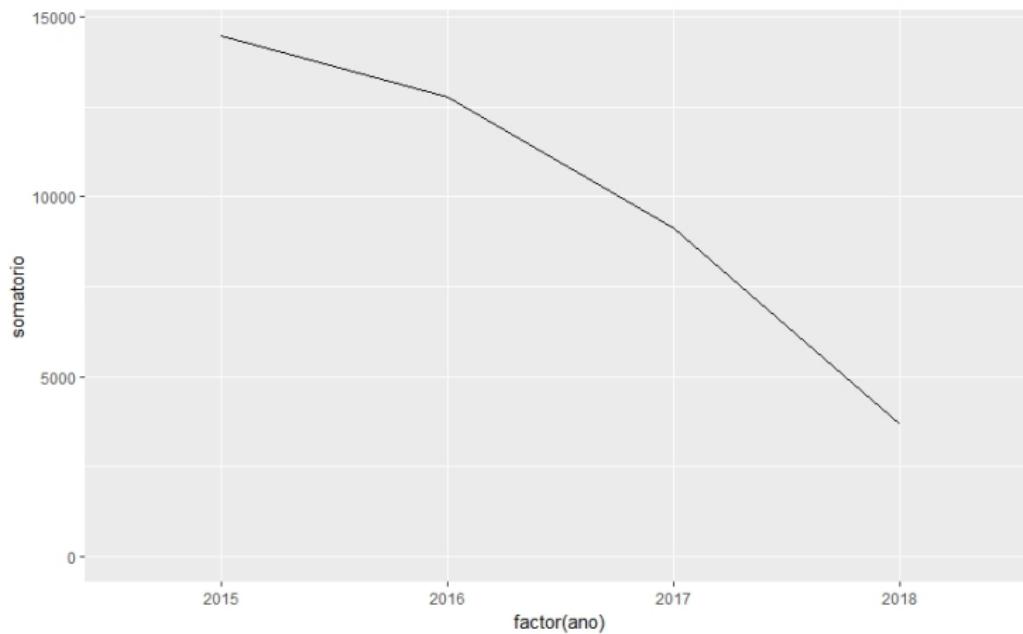
# Gráfico de linhas

- A inclusão do valor 0 pode ser importante em certas ocasiões;
- Podemos incluir expandindo o eixo y através da função `expand_limits()`, ou `ylim()`

```
ggplot(base, aes(x = factor(ano), y = somatorio,  
group = 1)) + geom_line() + expand_limits(y=0)
```

```
ggplot(base, aes(x = factor(ano), y = somatorio,  
group = 1)) + geom_line() +  
ylim(0, max(base$somatorio))
```

# Gráficos de linhas

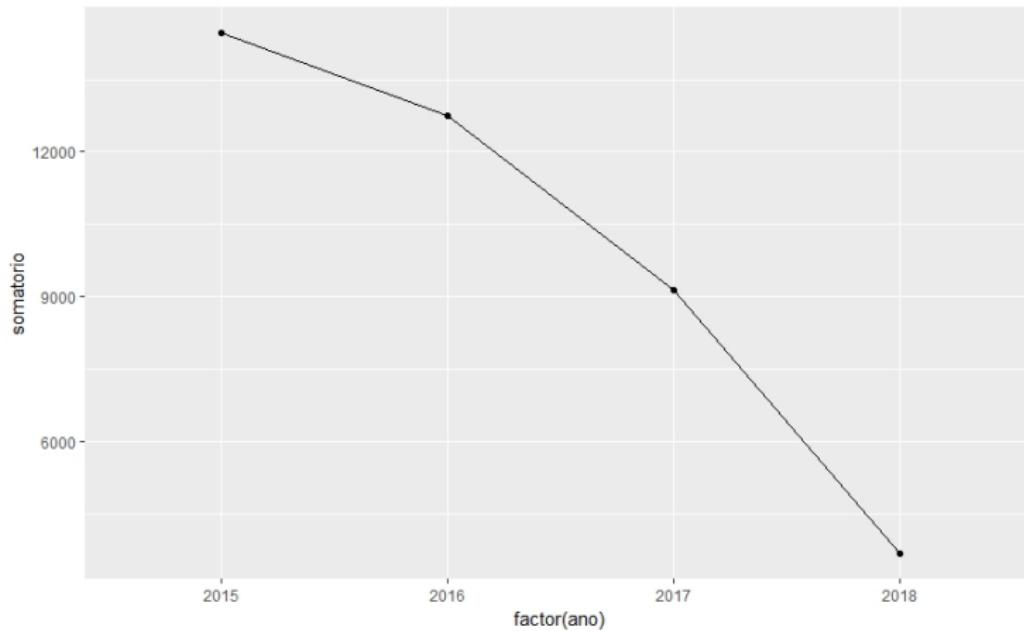


# Gráfico de linhas

- Adicionaremos pontos à linha, o que pode ser manipulado com size, colour e fill
- Para isso adicionamos a função geom\_point() à estrutura do gráfico de linhas

```
ggplot(base, aes(x = factor(ano), y = somatorio,  
group = 1)) + geom_line() + geom_point()
```

# Gráficos de linhas



# Gráfico de linhas

- Por fim, vamos adicionar duas linhas com o somatório dos artigos como primeiro autor e como último autor
- Primeiro, vamos realizar o mesmo que fizemos para os artigos totais, em seguida juntar todos e criar uma nova variável

```
> ano <- c(2015:2018, 2015:2018)
```

```
> somatorio <- c(base$primeiro, base$ultimo)
```

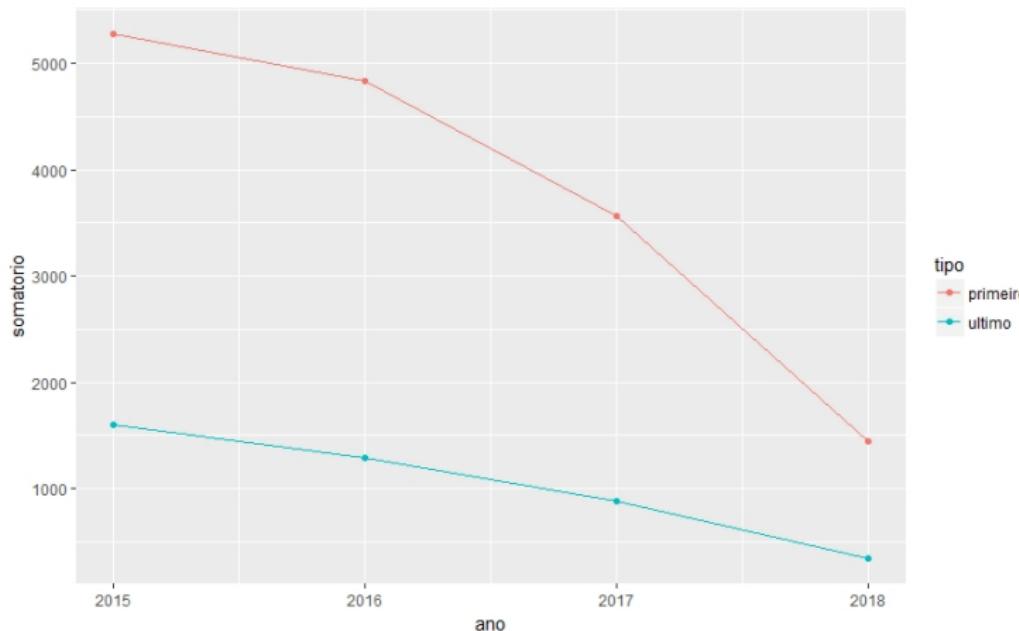
```
> tipo <- c("primeiro", "primeiro", "primeiro",
  "primeiro", "ultimo", "ultimo", "ultimo",
  "ultimo")
```

# Gráfico de linhas

- Em seguida, juntamos todos os vetores dentro de um data frame
- Faremos assim nosso gráfico com duas linhas

```
> base2 <- data.frame(ano,somatorio, tipo)  
  
> ggplot(base2, aes(x = ano, y= somatorio, colour = tipo)) + geom_line() + geom_point()
```

# Gráficos de linhas

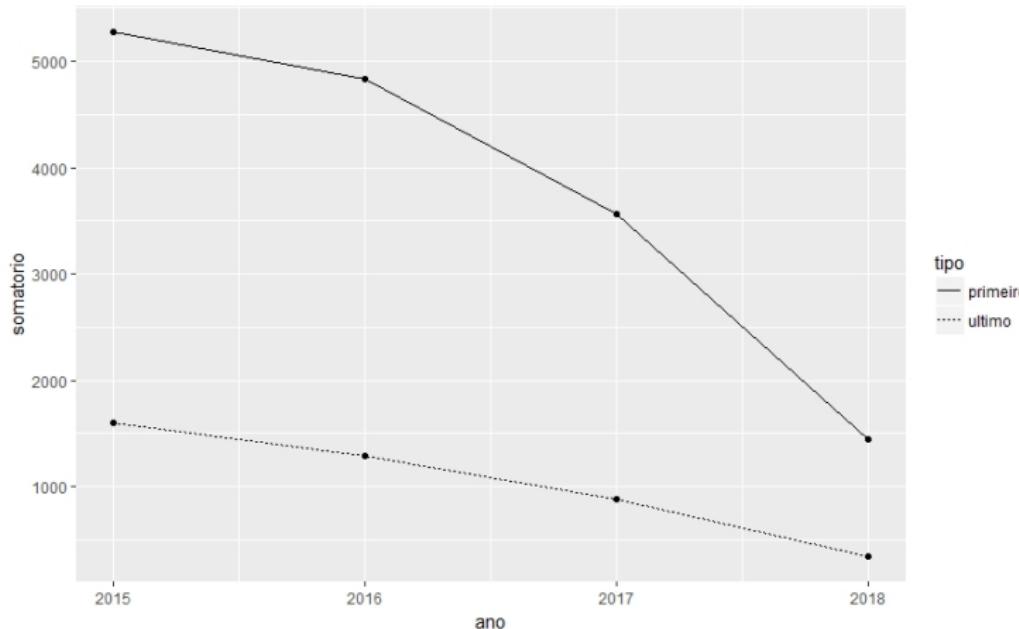


# Gráfico de linhas

- Ou podemos diferenciar as duas linhas por tipo:

```
ggplot(base2, aes(x = ano, y= somatorio,  
linetype = tipo)) + geom_line() + geom_point()
```

# Gráficos de linhas



# Básico box plot

- O box plot basicamente mapeia uma variável contínua no y e uma discreta no x
- Usaremos a variável de total de artigos

```
cnpq$NIVEL_ATUAL_PQ [cnpq$NIVEL_ATUAL_PQ == ""]  
<- "Sem"
```

```
table(cnpq$NIVEL_ATUAL_PQ)
```

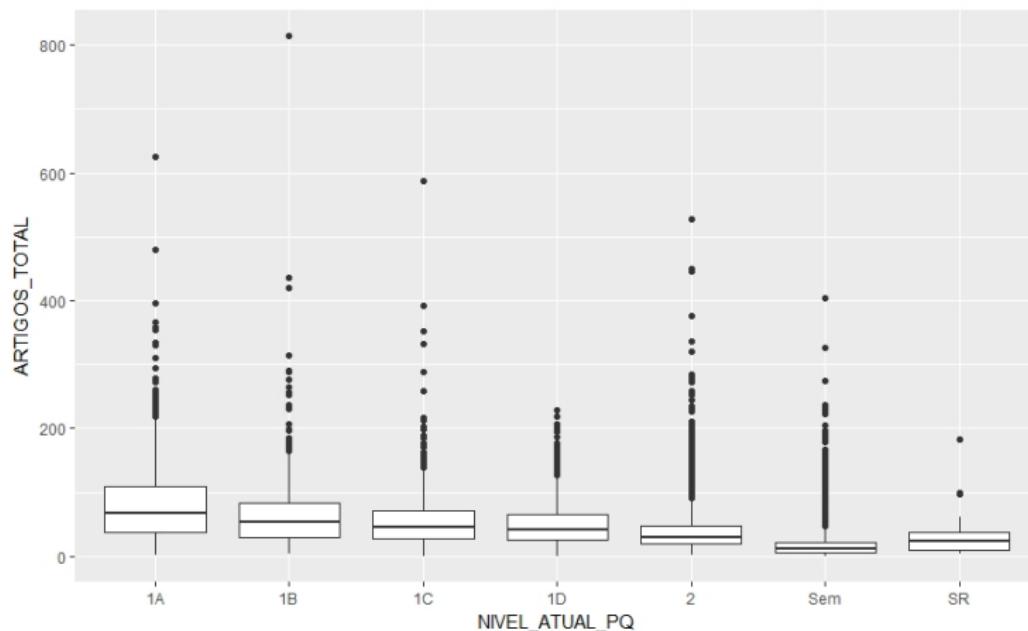
# Básico box plot

- Escolhemos a variável contínua do total de artigos como o y
- Para plotar o gráfico, portanto, utilizaremos a função geom\_boxplot()

```
summary(cnpq$ARTIGOS_TOTAL)
Min. 1st Qu. Median      Mean 3rd Qu.      Max.
0.00     8.00    16.00    23.91    30.00    814.
```

```
ggplot(cnpq, aes(x = NIVEL_ATUAL_PQ,
y = ARTIGOS_TOTAL)) + geom_boxplot()
```

# Gráficos de linhas

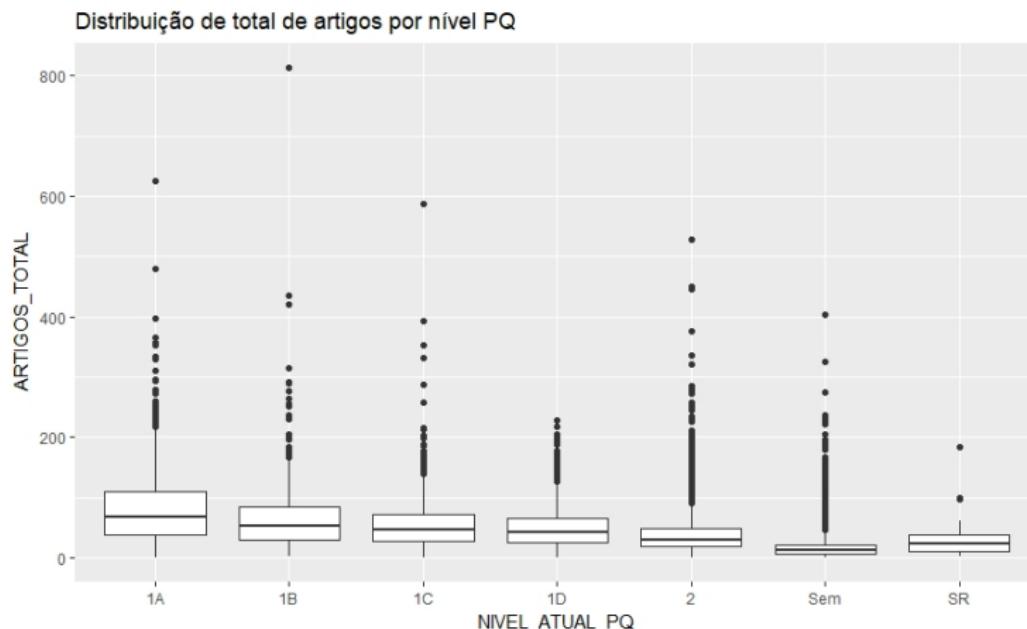


# Estética dos gráficos

- Todos os itens de estética dos gráficos podem ser controlados;
- O primeiro deles é o título do gráfico, que inserimos utilizando ou `ggtitle()` ou `labs(title = )`

```
ggplot(cnpq, aes(x = NIVEL_ATUAL_PQ,  
y = ARTIGOS_TOTAL)) + geom_boxplot() +  
  labs(title =  
    "Distribuição de total de artigos por nível")
```

# Gráficos de linhas

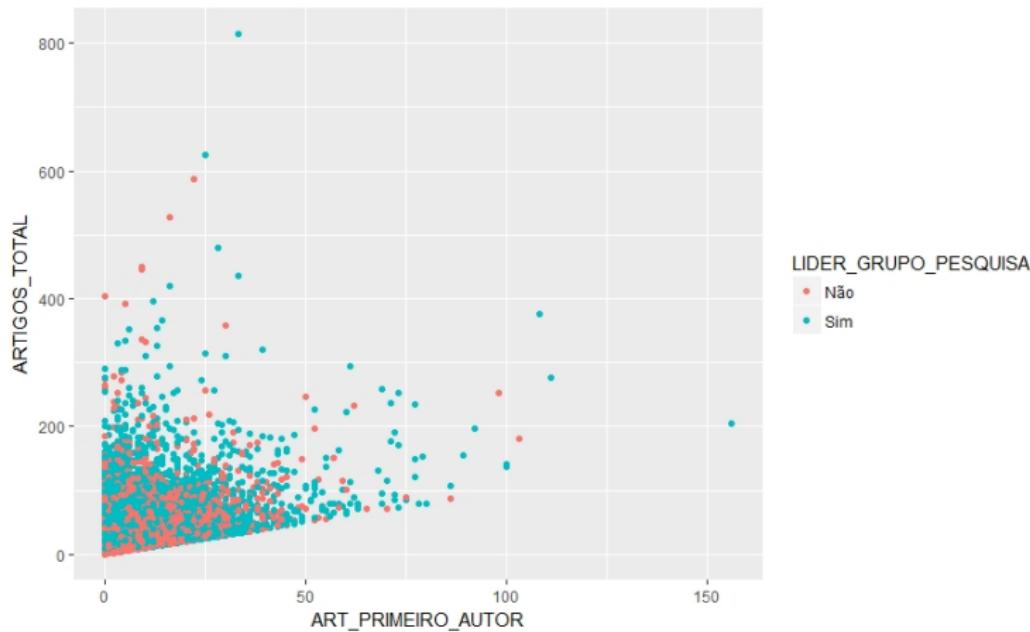


# Estética dos gráficos

- Uma das mais importantes funções no ggplot(), porém, é o theme()
- theme() controla aspectos de dentro do gráfico, como a grandeza de linhas e a coloração do fundo
- Vamos modificar o fundo de um gráfico de pontos

```
graph2 <- ggplot(cnpq, aes(x = ART_PRIMEIRO_AUTOR,  
y = ARTIGOS_TOTAL,  
colour = LIDER_GRUPO_PESQUISA)) +  
geom_point()
```

# Estética dos gráficos

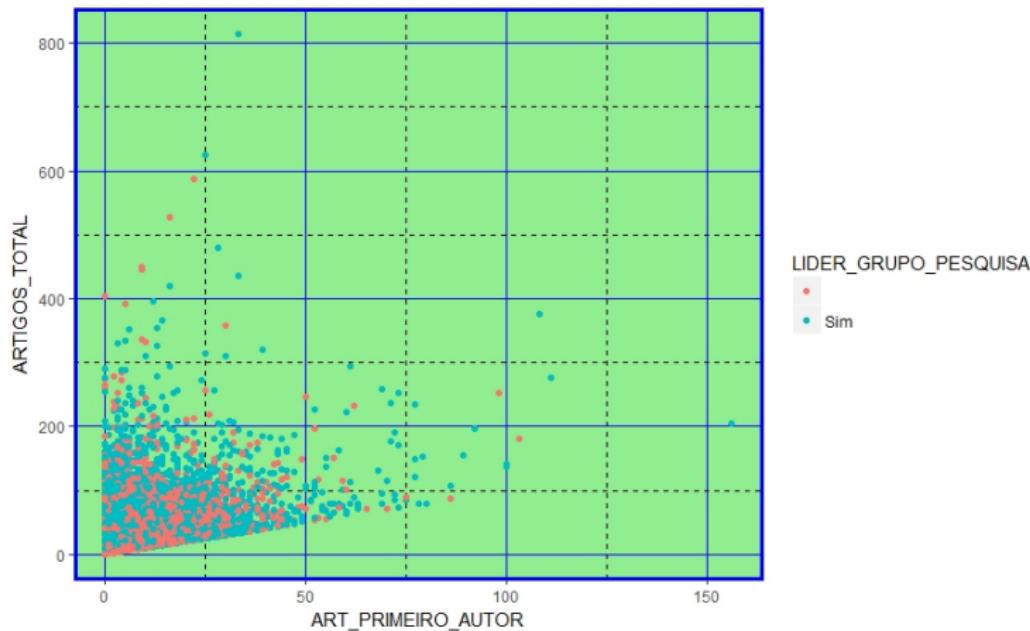


# Estética dos gráficos

- Vamos modificar o fundo do gráfico utilizando theme()

```
graph2 + theme(panel.grid.major =  
element_line(colour = "blue"),  
panel.grid.minor = element_line(colour = "black",  
linetype = "dashed", size = 0.5),  
panel.background =  
element_rect(fill = "lightgreen"),  
panel.border = element_rect(colour = "blue",  
fill = NA, size = 2))
```

# Estética dos gráficos

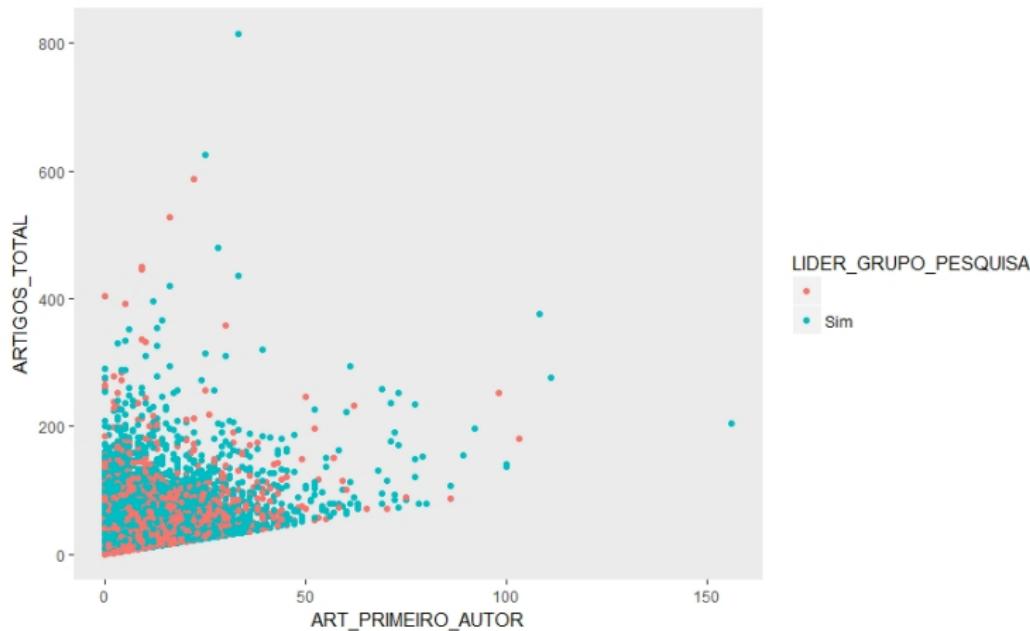


# Estética dos gráficos

- Mas se eu quiser sumir com as linhas do fundo do gráfico, eu utilizo como valor element\_blank()

```
graph2 +  
  theme(panel.grid.major = element_blank(),  
        panel.grid.minor = element_blank())
```

# Estética dos gráficos

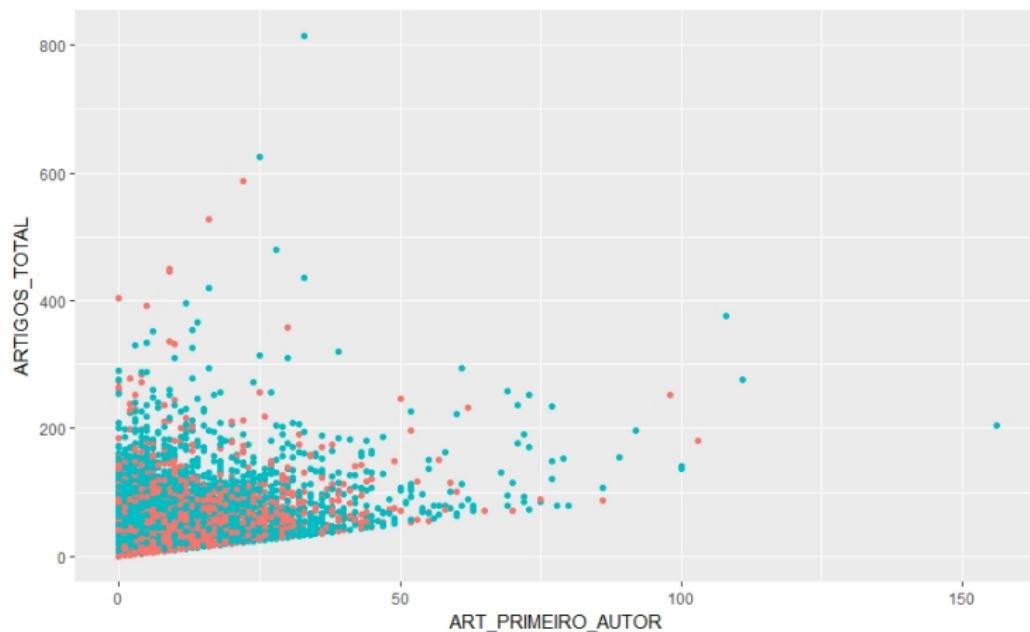


# Estética dos gráficos

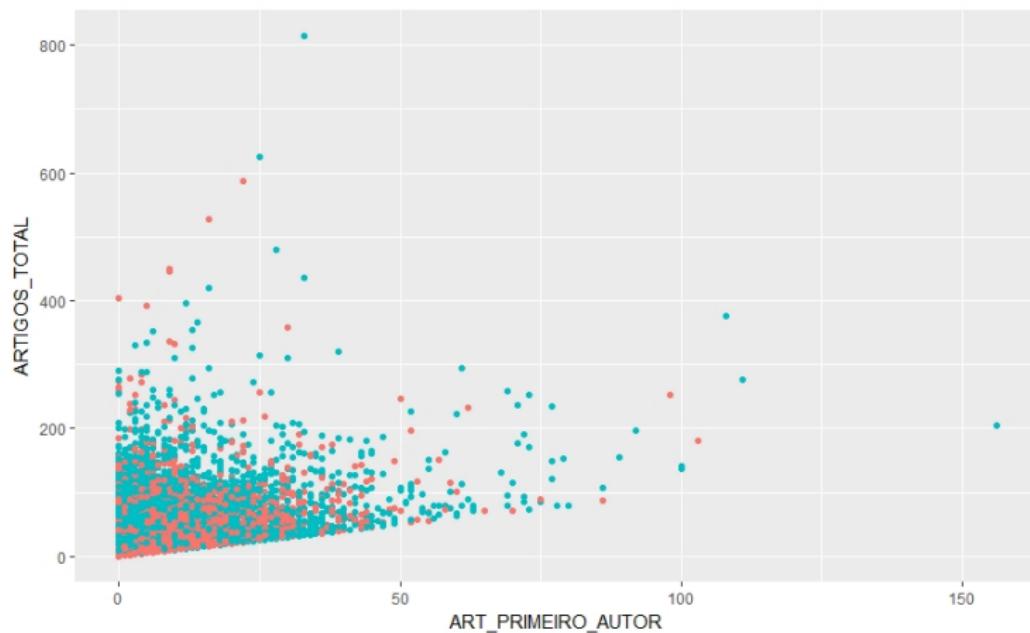
- Para ainda remover a legenda, utilizamos dentro de theme() o argumento legend.position = "none"
- Com o mesmo argumento podemos mover a legenda de lugar

```
graph2 + theme(legend.position = "none")
graph2 + theme(legend.position = "top")
graph2 + theme(legend.position = c(0.9,0.2))
```

# Estética dos gráficos



# Estética dos gráficos



# Estética dos gráficos

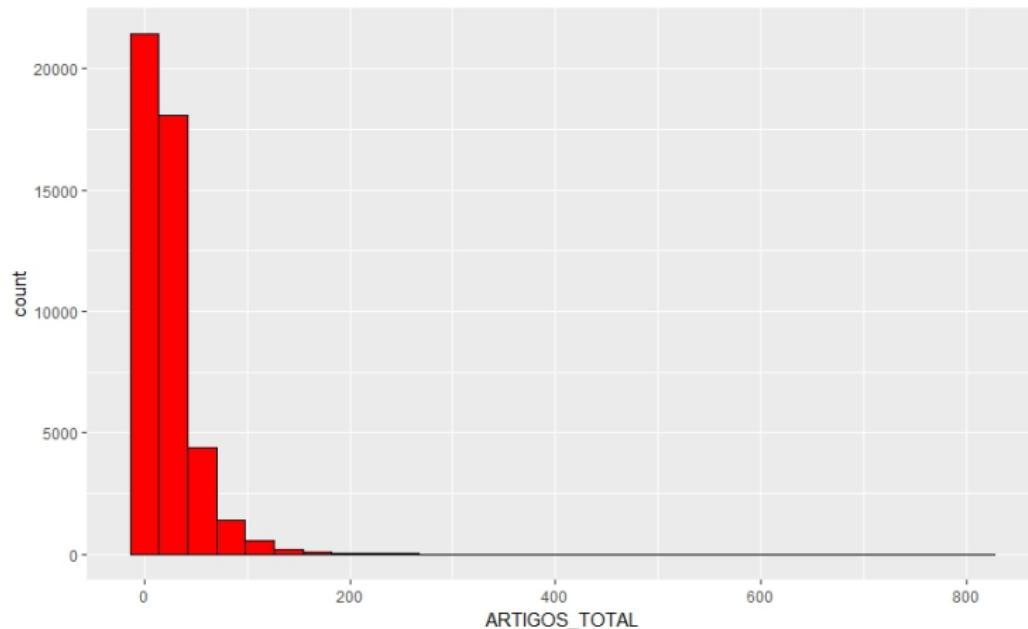


# Cores

- A utilização de cores é mais uma possibilidade de eixo no `ggplot()`, porém com maior grau de liberdade e escolhas possíveis
- Além de estética, portanto, cores são informações sobre a visualização dos dados
- Primeiramente, podemos definir cor através do `fill`, que preenche a área de polígono, e podemos utilizar `colour`, que preenche os contornos de um polígono

```
ggplot(cnpq, aes(x = ARTIGOS_TOTAL)) +  
  geom_histogram(fill = "red", colour = "black")
```

# Cores



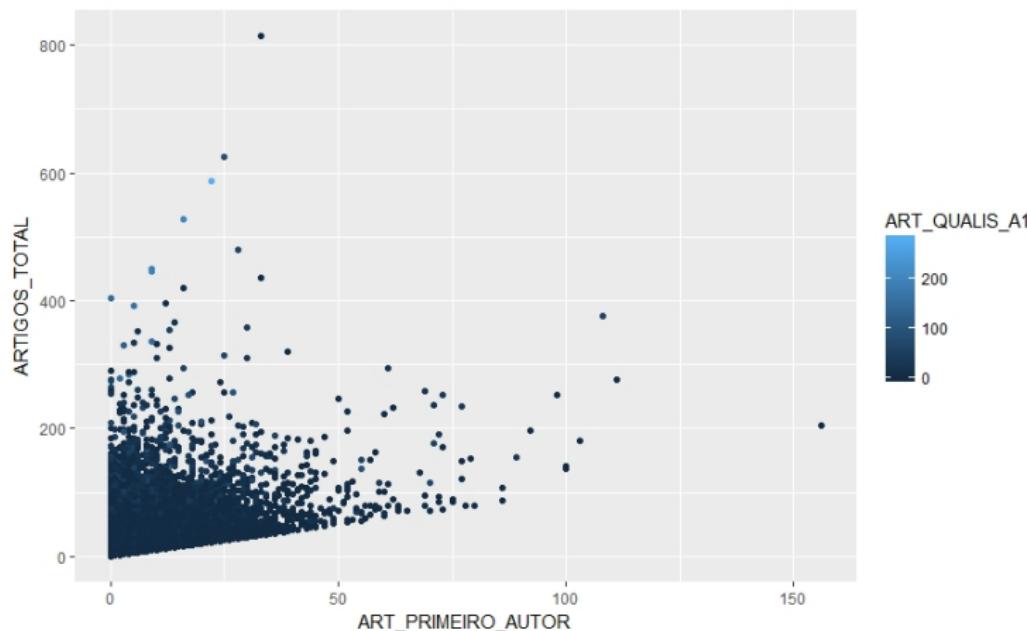
# Cores

- Cores também são utilizadas para destacar terceira variável do tipo contínua
- Da mesma forma, marcamos dentro do argumento aes() a opção colour= com a variável, obtendo um degradê de cores

```
ggplot(cnpq, aes(x = ART_PRIMEIRO_AUTOR,  
y = ARTIGOS_TOTAL,  
colour = ART_QUALIS_A1)) +  
geom_point()
```

- Podemos alterar a variável contínua para discreta com o factor(), porém o resultado pode não ser muito bom

# Cores



# Cores

- Vamos definir cores de forma manual aos nossos gráficos
- Primeiro já vamos redefinir as cores da variável contínua acima
- Atribuindo o label p ao nosso gráfico básico, em seguida, utilizaremos a função `scale_colour_gradient()`, definindo como primeiro argumento a cor do valor mais baixo (`low=`) e do mais alto (`high=`)

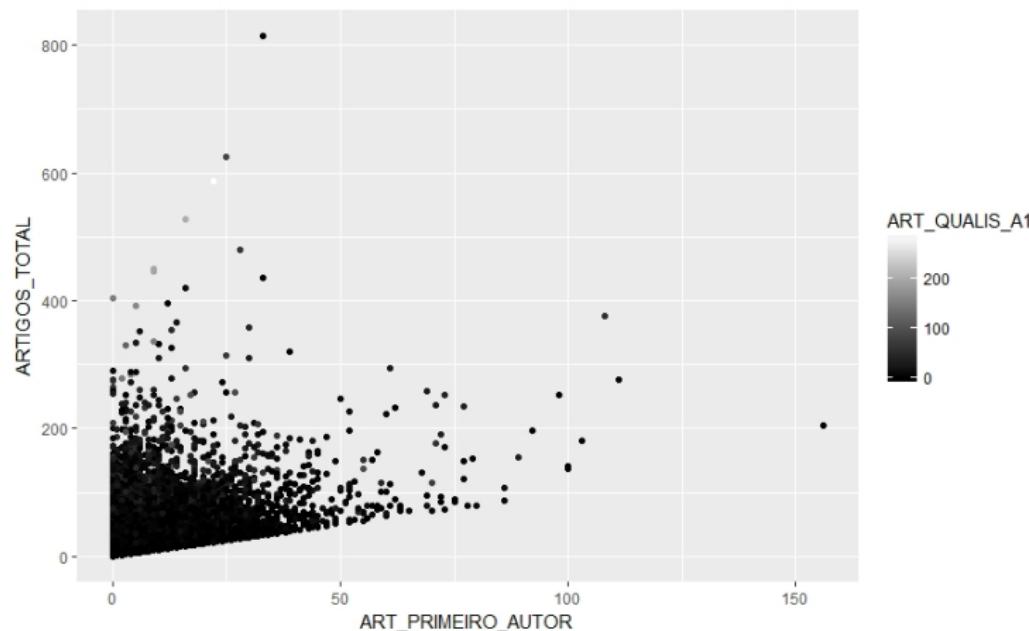
```
p <- ggplot(cnpq, aes(x = ART_PRIMEIRO_AUTOR,  
y = ARTIGOS_TOTAL,  
colour = ART_QUALIS_A1)) +  
  geom_point()  
  
p + scale_colour_gradient(low = "black",  
high = "white")
```

# Cores

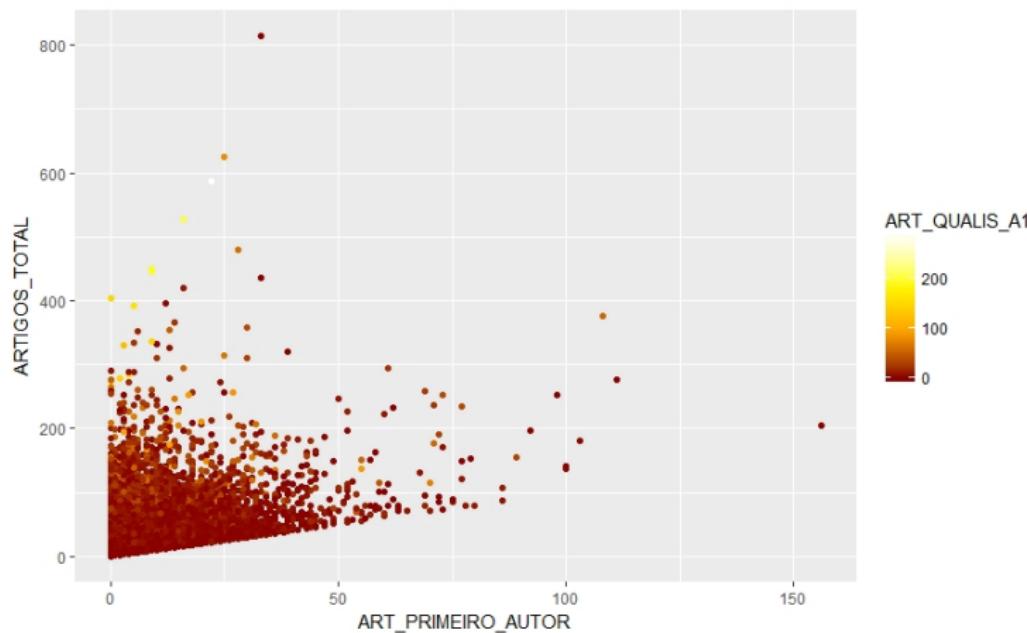
- Ou ainda definimos o gradiente com `colours=c()` na função `scale_colour_gradientn()`

```
p + scale_colour_gradientn(colours =  
c("darkred","orange","yellow","white"))
```

# Cores



# Cores

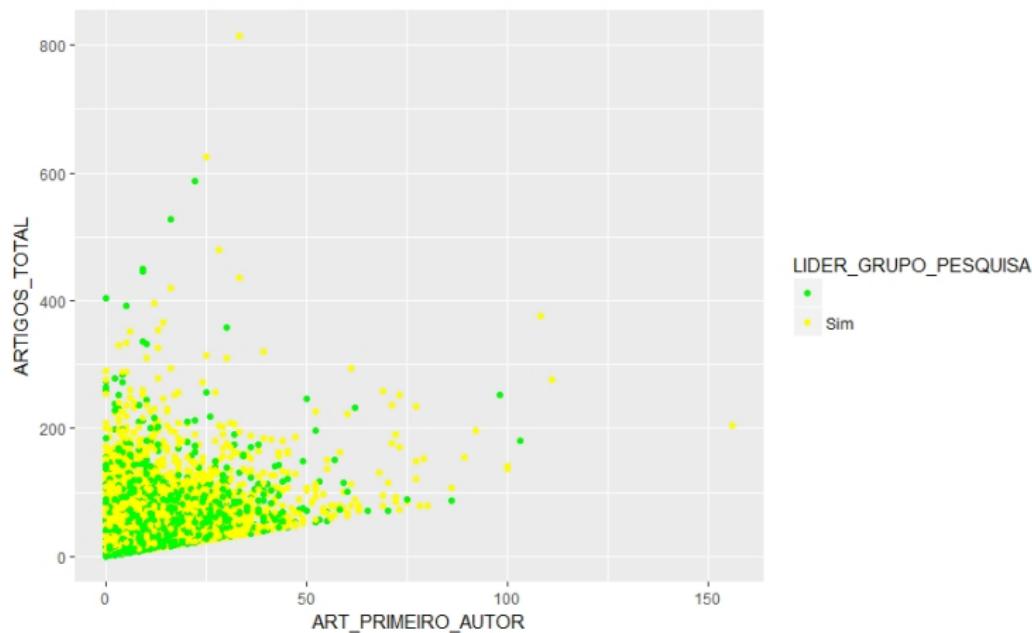


# Cores

- A definição de cores de uma terceira variável discreta é através da função `scale_colour_manual()`, com a utilização do argumento `values=c()`
- A ordem das cores será atribuída na ordem dos valores da legenda

```
y <- ggplot(cnpq, aes(x = ART_PRIMEIRO_AUTOR,  
y = ARTIGOS_TOTAL,  
colour = LIDER_GRUPO_PESQUISA)) +  
geom_point()  
  
y + scale_colour_manual(values = c("green",  
"yellow"))
```

# Cores

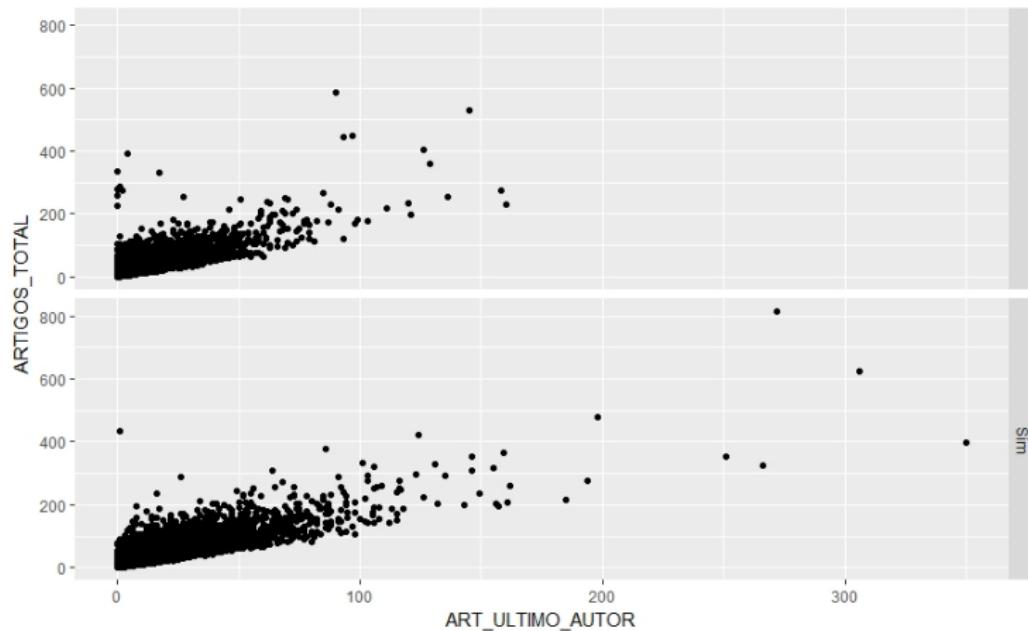


# Facet\_grid

- Outra forma bastante elegante de apresentação de variáveis discretas é o facet\_grid()
- Ou seja, separar por diferentes telas o gráfico de acordo com a variável discreta.
- Por exemplo, utilizar esse gráfico de dispersão com as variáveis primeiro autor

```
ggplot(cnpq, aes(x = ART_ULTIMO_AUTOR,  
y = ARTIGOS_TOTAL)) +  
geom_point() +  
facet_grid(LIDER_GRUPO_PESQUISA ~ .)
```

# Facet\_grid

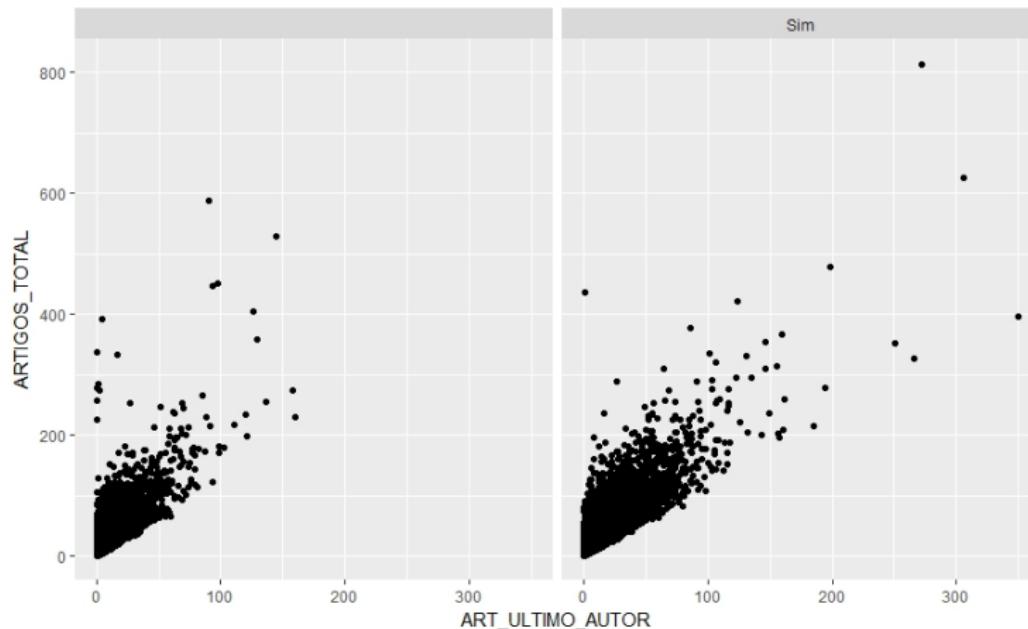


# Facet\_grid

- Para o corte ser feito na vertical, mudamos a ordem do ' ' e do '..'
- Ou seja, acima, colocamos primeiro a variável e em seguida ' ..', obtendo o corte horizontal
- No vertical, colocamos primeiro o '..', em seguida o ' ' e, por fim, a variável de interesse

```
ggplot(cnpq,aes(x = ART_ULTIMO_AUTOR ,  
y = ARTIGOS_TOTAL)) +  
geom_point() +  
facet_grid(LIDER_GRUPO_PESQUISA~.)
```

# Facet\_grid



# Edição de Funções

- Uma das habilidades possíveis no R e útil para índices é criar as próprias funções
- Por exemplo, podemos criar uma função de média

```
round(mean(cnpq$ARTIGOS_TOTAL),2)  
[1] 23.91
```

```
round(sum(cnpq$ARTIGOS_TOTAL)/  
length(cnpq$ARTIGOS_TOTAL),2)  
[1] 23.91
```

# Edição de Funções

- A notação básica de edição de função é:

```
funcao <- function(arg1, arg2, ...){  
  formula  
  return(objeto)  
}
```

- Assim, queremos gerar uma formula da média que produza um resultado

```
media <- function(x){  
  round(sum(x)/length(x),2)  
}  
  
class(media)  
[1] "function"
```

# Edição de Funções

- Em seguida, aplicamos a nova função com a etiqueta

```
media(cnpq$ARTIGOS_TOTAL)  
[1] 23.91
```

- Vamos criar uma função para a moda, que não possui no R

```
moda <- function(y){  
  uniqy <- unique(y)  
  uniqy[which.max(tabulate(match(y, uniqy)))]  
}
```

```
moda(cnpq$ARTIGOS_TOTAL)  
[1] 7
```

# Condições

- Para criar uma condição usamos if
- Vamos criar um protótipo do campo minado então

```
campo <- matrix(sample(c("b","w"),  
size = 16, replace = T), ncol = 4, nrow = 4,  
byrow = F)
```

```
x <- campo[2,4]
```

```
if(x == "b"){  
  print("booooooom")  
} else{  
  print("chuuuuuuu ")  
}
```

# Aula 6

Álvaro João Pereira Filho

Curso de R  
21/06/2019