

# Análise dos Dados com R

---

## Arquivos e Programação em R

James R. Hunter, PhD

27 de outubro de 2024

# **Tipos/Classes de Dados em R**

# Tipos de Dados Atômicos

- <int> **integer**
  - Números inteiros (sem parte decimal)
  - Designados em R com um "L" (26L)
- <dbl> **double**
  - Números "Real"
  - Tem uma parte decimal (mesmo se for implicita)
  - Designados em R como "numeric"
- <chr> **character**
  - "Strings"
  - Cercados por aspas simples ou duplas (' ' or "")
- <lgl> **logical**
  - Só podem assumir os valores TRUE ou FALSE

# Dados Lógicos

- Podem ser interpretados como números (*integers*)
  - FALSE = 0
  - TRUE = 1
- Podem ser atribuidos como
  - `x <- TRUE`
- Podem ser o resultado de cálculos *lógicos*
  - Dois números são iguais, não iguais, ...

# Cálculos Lógicos

- Operadores lógicos
  - == lado esquerdo igual ao lado direto
  - != lado esquerdo **não** igual ao lado direto
  - >= lado esquerdo maior ou igual ao lado direto
  - < lado esquerdo menor do lado direto

# Exemplos

```
x <- 6 # atribuir um valor a x  
x == 6 # testar se x é igual a 6
```

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

```
2 > 4
```

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

# Quer Saber Qual Tipo de Variável É Ela?

- `typeof()` - tipo interno do objeto
- `class()` - tipo geral do objeto
  - Muito útil para tipos compostos (ie, não atômicos)

# Exemplos dos Tipos atômicos

```
int_var <- 25L  
typeof(int_var)
```

```
## [1] "integer"
```

```
num_var <- 25.879  
typeof(num_var)
```

```
## [1] "double"
```

```
log_var <- TRUE  
typeof(log_var)
```

```
## [1] "logical"
```

```
char_var <- "abcd"  
typeof(char_var)
```

# Hierarquia das Classes - Coerção

- R pensa que não pode misturar operações que envolvem operações com classes diferentes
  - Não pode multiplicar um string por um número
  - R retorna um erro

```
> char_var * 2  
Error in char_var * 2 : non-numeric argument to binary operator
```

# Hierarquia

- De mais restritivo ao mais geral
  - **Logical**
    - Só tem dois valores
    - Também são *integers* (com valores 0 e 1) e podem estar usados para os cálculos
  - **Integer**
    - Pode ser calculados como se fossem *numeric*
    - Armazenados numa forma mais compacta na memoria que *than numeric*
  - **Numeric**
    - Qualquer número pode ser um valor *numeric*
  - **Character**
    - Qualquer outra classes pode ser gravada em memoria como *character*
    - Não pode fazer cálculos com *character*

# Coerção

- Aplica-se principalmente aos *vectors*
- Vetores devem ter valores de um tipo só
- R forçará os valores a serem o tipo *mais geral* dos tipos que você fornecer

# Exemplos da Coerção

```
(vect1 <- c(7L, 27890L))
```

```
## [1] 7 27890
```

```
typeof(vect1)
```

```
## [1] "integer"
```

- 2 *integers*, tipo resultado = *integer*

## Exemplo 2

```
(vect2 <- c(7L, 27.333))
```

```
## [1] 7.000 27.333
```

```
typeof(vect2)
```

```
## [1] "double"
```

- *Integer e numeric* valores, tipo resultado = *double*

## Exemplo 3

```
(vect3 <- c(27.333, "cat"))
```

```
## [1] "27.333" "cat"
```

```
typeof(vect3)
```

```
## [1] "character"
```

- *Double and character values, tipo resultado = character*

## Exemplo 4

```
(vect4 <- c(TRUE, 7L, 27.333, "cat"))
```

```
## [1] "TRUE"     "7"        "27.333"   "cat"
```

```
typeof(vect4)
```

```
## [1] "character"
```

- Combinação de todos os tipos atômicos, tipo resultado = *character*

# Exemplo 5

```
(vect5 <- c(7L, TRUE))
```

```
## [1] 7 1
```

```
typeof(vect5)
```

```
## [1] "integer"
```

- *Integer* e *Logical* valores, tipo resultado = *integer*
- Somas desses valores - possível

```
sum(vect5)
```

```
## [1] 8
```

# Coerção com Comandos

- Pode forçar a coerção de uma variável para um tipo diferente
  - Se for um tipo apropriado para o valor
- Comandos `as.xxx()`
  - `as.logical()`
  - `as.integer()`
  - `as.numeric()`
  - `as.character()`

# Exemplos de Coerção

```
as.character(25.3)
```

```
## [1] "25.3"
```

```
as.integer(25.3) # perderá a parte decimal
```

```
## [1] 25
```

```
as.numeric("25.3") # vai tornar um número
```

```
## [1] 25.3
```

```
as.numeric("cat") # Não é permitido; resultado = NA
```

```
## Warning: NAs introduced by coercion
```

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

# Mais Classes de Dados

# Factors

- Maneira compacta para gravar informações categóricas
- Transforma *strings*, números e valores lógicos em um fatorConverts strings, numbers and even logical variables into factor
- Duas partes para um *factor*
  - Lista interna das possibilidades alternativas (*levels*)
  - *Integers* que atribuem cada valor para um *level*
- Úteis quando tem um número limitado de possibilidades
  - Gênero, tipo de tratamento, etc.
  - Menos de 10 níveis (*levels*) em geral

# Exemplo Factor - 1

- Variável para *gênero* em um estudo
- 4 níveis possíveis:
  - "masculino", "feminino", "não quer declarar", "outro"
- *gênero* atualmente uma variável *character*

```
genero <- c("feminino", "não quer declarar", "outro", "masculino", "feminino",
          "masculino")
```

```
genero
```

```
## [1] "feminino"           "não quer declarar" "outro"
## [4] "masculino"           "feminino"           "masculino"
```

```
typeof(genero)
```

```
## [1] "character"
```

# Exemplo Factor - 2

- Converter `genero` para um *factor* com `factor()`

```
generof <- factor(genero)  
typeof(generof)
```

```
## [1] "integer"
```

```
class(generof)
```

```
## [1] "factor"
```

## Exemplo Factor - 3

```
str(generof)
```

```
##  Factor w/ 4 levels "feminino", "masculino", ... : 1 3 4 2 1 2
```

```
levels(generof)
```

```
## [1] "feminino"           "masculino"          "não quer declarar"  
## [4] "outro"
```

# Representação das Datas e Horas

- Várias classes tratam de datas e horas
- Vamos focar na classe *Date*
- Pacotes que facilitam trabalho com datas e horas
  - `lubridate`
  - `hms`
- Em *databases*, datas normalmente têm representação como *strings* ("04-09-2021")

# Converter *String* de Data para Classe de Data

- Função `as.Date()`
- Se data não está no formato de data ISO ("YYYY-mm-dd")
  - Precisa avisar `as.Date()` qual formato que tem
    - "%d" = dia
    - "%m" = mês
    - "%y" ou "%Y" = ano (2 dígitos or 4)
    - O *string* inteiro em aspas ("")

# Exemplo Converter Data

```
d <- "28/05/2021"  
d_date <- as.Date(d, format = "%d/%m/%Y")  
d_date
```

```
## [1] "2021-05-28"
```

```
class(d_date)
```

```
## [1] "Date"
```

# Matemática com Datas

- Pode calcular intervalos de tempo utilizando o tipo de dado **Date**
- Operações aritméticas normais
- VSS: função `Sys.Date()` retorna a data atual
- Aritmética de datas útil para calcular idade, tempo desde infecção

```
hoje <- Sys.Date()  
hoje
```

```
## [1] "2024-10-16"
```

```
hoje - d_date
```

```
## Time difference of 1237 days
```

# Trabalho com Vetores

# Indexação dos Vetores

- Acessar qualquer elemento de um vetor com um número cercado por "[]" (colchetes)
- Vetor `x` tem 10 números reais
  - Primeiro número: `x[1]`
  - Sexto número: `x[6]`

```
x <- c(177.89, 194.47, 32.24, 99.56, 205.34,  
      -0.95, 171.96, 112.65, 32.93, 60.53)
```

```
# 1o numero  
x[1]
```

```
## [1] 177.89
```

```
# 6o numero  
x[6]
```

```
## [1] -0.95
```

# E Tem Mais ...

- Acessar o 2º e 5º elementos de x
  - Utilize a função `c()`
- Acessar o 2º **ate** o 6º elementos
  - Utiliza o operador `:`

```
x[c(2,5)]
```

```
## [1] 194.47 205.34
```

```
x[2:5]
```

```
## [1] 194.47 32.24 99.56 205.34
```

# head() & tail()

- Acessar os primeiros  $n$  valores do vetor com `head()`
  - $n$  como argumento dentro das parênteses (default = 6)
- Acessar os últimos  $n$  valores do vetor com `tail()`

```
head(x)
```

```
## [1] 177.89 194.47 32.24 99.56 205.34 -0.95
```

```
tail(x)
```

```
## [1] 205.34 -0.95 171.96 112.65 32.93 60.53
```

```
head(x, n = 3)
```

```
## [1] 177.89 194.47 32.24
```

# Classes de Conjuntos de Dados

# Conjuntos de Dados em R

- Coleção de variáveis
  - Cada uma tem vários casos
- Variável
  - Identificador
  - Classificador categórico
  - Valor numérico ou lógico
- Variável é um vetor
  - Todos os valores devem ser do mesmo tipo
  - Todas as variáveis num conjunto (exceto *list*) devem ter o mesmo número de casos
- 3 classes de conjuntos de dados
  - List
  - Data.frame
  - Tibble

# List

- Tipo mais geral
- Qualquer tipo de dado
- R não aplica coerção
- Em análises de dados cotidianas, não é muito comum
- Mas, `data.frame` e `tibble` são subtipos de `list`

```
vect4 <- list(TRUE, 7L, 27.333, "cat")
typeof(vect4)
```

```
## [1] "list"
```

```
typeof(vect4[[2]])
```

```
## [1] "integer"
```

# Data.frame

- Tipo mais comum para conjuntos de dados
- Uma lista com estrutura adicional (bem nas entranhas de R)
  - Variáveis - Colunas
  - Casos - Fileiras
- Todas as variáveis devem ter o mesmo número dos casos
- Variáveis têm nomes
- Casos *podem* ter nomes (não obrigatório)
- Visualizar a estrutura com `str()`

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:  
## $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...  
## $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...  
## $ disp: num  160 160 108 258 360 ...  
## $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...  
## $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...  
## $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...  
## $ qsec: num  16.5 17 18.6 19.4 17 ...  
## $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...  
## $ am  : num  1 1 1 0 0 0 0 0 0 0 ...  
## $ gear: num  4 4 4 3 3 3 3 4 4 4 ...  
## $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
head(mtcars, n = 3)
```

```
##          mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6 160 110 3.90 2.620 16.46  0  1     4     4
## Mazda RX4 Wag 21.0   6 160 110 3.90 2.875 17.02  0  1     4     4
## Datsun 710    22.8   4 108  93 3.85 2.320 18.61  1  1     4     1
```

```
class(mtcars)
```

```
## [1] "data.frame"
```

```
typeof(mtcars)
```

```
## [1] "list"
```

# Construir um Data.frame

- Utilizar a função `data.frame()`
  - Fazer uma lista das variáveis e os valores que você quer que elas assumem
  - Separar as variáveis com vírgulas
  - R atribui tipos de dados para as variáveis que criamos

```
df <- data.frame(name = c("Jim", "Fernanda", "Ana"),
                  gender = c("male", "female", "female"),
                  score = c(89, 91, 93),
                  passed = c(TRUE, TRUE, TRUE))
str(df)
```

```
## 'data.frame':    3 obs. of  4 variables:
## $ name : chr  "Jim" "Fernanda" "Ana"
## $ gender: chr  "male" "female" "female"
## $ score : num  89 91 93
## $ passed: logi  TRUE TRUE TRUE
```

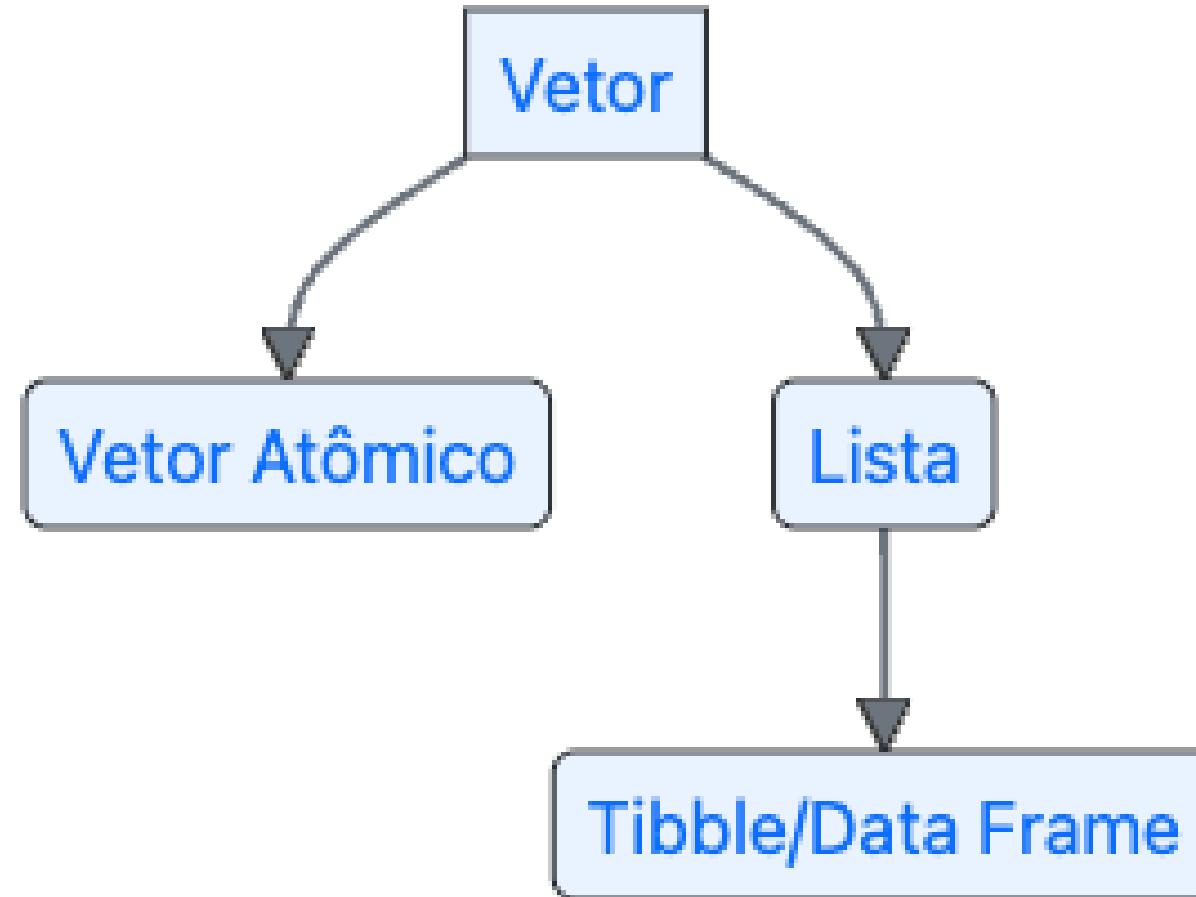
# Notação \$

- O que é o \$ na função str()?
  - `$ name : chr "Jim" "Fernanda" "Ana"`
- \$ separa o nome do `data.frame` do nome de variável
- `df$name` terá resultado dos 3 nomes
  - Jim, Fernanda, Ana
- Notação de colchetes ao final pode selecionar entre os nomes
  - `df$name[2]` -> Fernanda

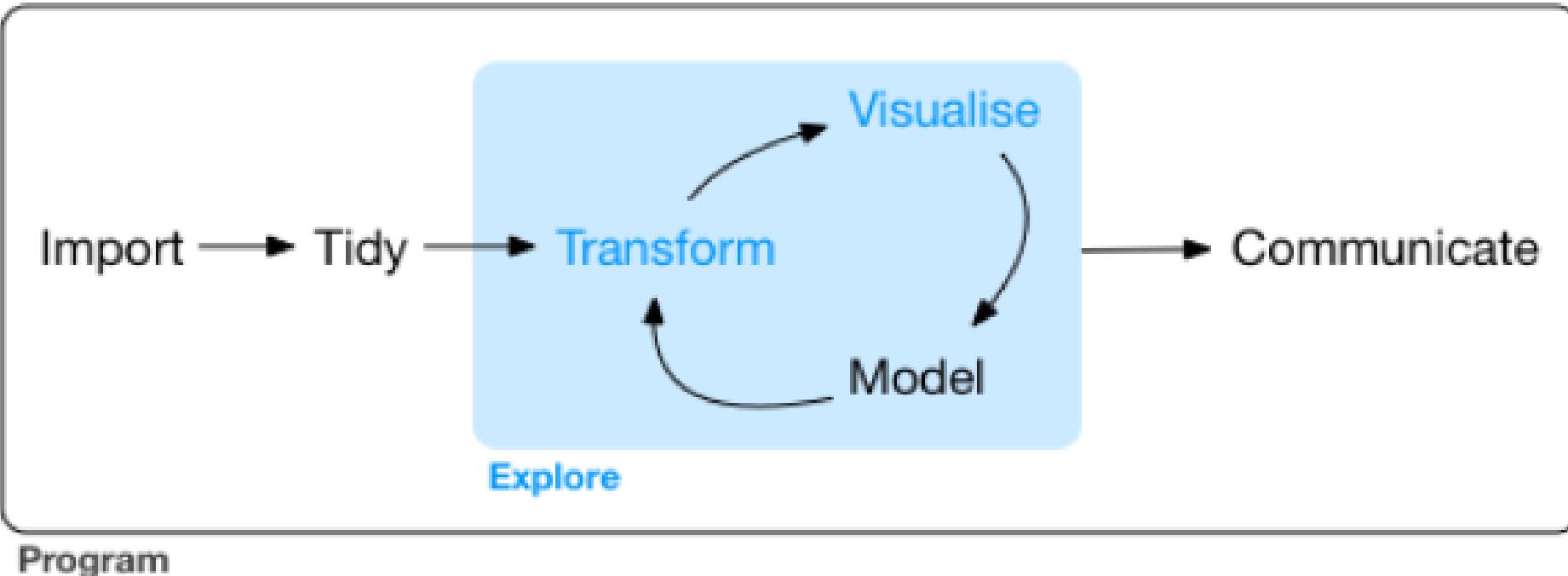
# Tibble

- Forma avançada de um `data.frame`
- Parte do sistema **Tidyverse**
- Utiliza as mesmas regras que os `data.frame`
- Estrutura interna um pouco diferente
- Faster processing (most of the time)

# Vetores até Tibbles



# Fluxo de Trabalho de Análise de Dados - "The Big Picture"



# Importar Dados

- Primeiro passo em qualquer projeto em R
- Fontes dos dados:
  - Excel (.csv or .xlsx) **minha escolha normal**
  - Arquivos de texto (.txt or .csv)
  - Fasta
  - Arquivos dos outros processos bioinformáticos
- R não é muito eficiente para preparação dos dados

# "Tidy Data"

- Garantir que os dados aparecem numa forma útil para a análise
- Maioria dos conjuntos de dados são recolhidos e armazenados ao acaso
  - Muitos com muito pontos de dados faltando ("*missing data*")
  - Valores impossíveis
    - O homem de "135" anos
- Precisa captar esses erros
  - Corrigir o que for possível
  - Ter uma estratégia clara para lidar com o resto

# "Tidy Data"

- Conjunto de princípios sobre a organização dos dados para a análise
- "Tidyverse"
  - Conjunto de pacotes "opinativos" que promovem e utilizam os princípios de formatação "*tidy*"
- A marca registrada dos "tidy data" é a consistência



Horst

# Definição de Hadley Wickham sobre "Tidy Data"

Like families, tidy datasets are all alike, but every messy dataset is messy in its own way....

A dataset is a collection of *values*, usually either numbers (if quantitative) or strings (if qualitative). Values are organized in two ways.

Every value belongs to a *variable* and an *observation*. A variable contains all values that measure the same underlying attribute (like height, temperature, duration) across units. An observation contains all values measured on the same unit (like a person, or a day, or a race) across attributes....

Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types.\*

\*Wickham, Hadley. "Tidy Data". Journal of Statistical Software 59, 10 (2014). <https://doi.org/10.18637/jss.v059.i10>.

# As Três Regras de Dados "Tidy"

- Cada variável deve ter sua própria coluna
- Cada observação (caso) deve ter sua própria fileira
- Cada valor deve ter sua própria célula\*

\*Wickham and Grolemund, Ch. 12.1.

country	year	cases	population
Afghanistan	1999	745	1847071
Afghanistan	2000	1666	20395360
Brazil	1999	3737	172006362
Brazil	2000	80488	174604898
China	1999	21258	1272015272
China	2000	21666	128042583

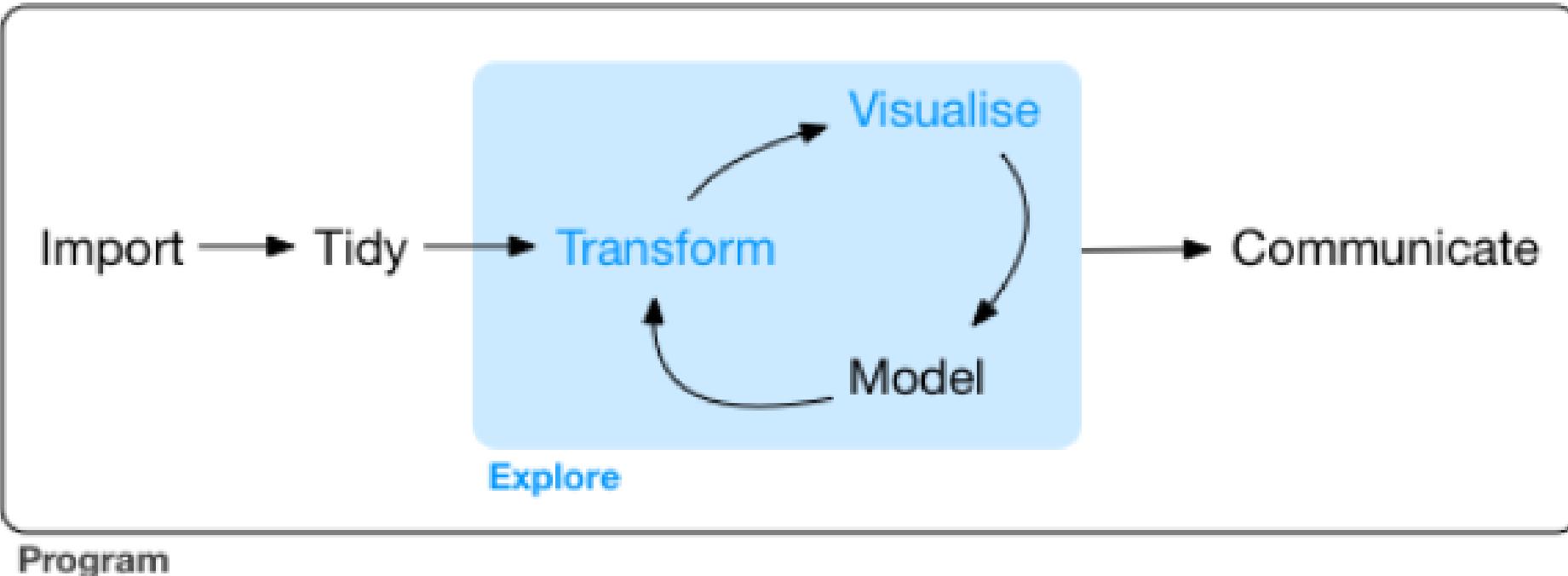
variables

country	year	cases	population
Afghanistan	1999	745	1847071
Afghanistan	2000	1666	20395360
Brazil	1999	3737	172006362
Brazil	2000	80488	174604898
China	1999	21258	1272015272
China	2000	21666	128042583

observations

country	year	cases	population
Afghanistan	1999	745	1847071
Afghanistan	2000	1666	20395360
Brazil	1999	3737	172006362
Brazil	2000	80488	174604898
China	1999	21258	1272015272
China	2000	21666	128042583

values



# Exploração dos Dados

- Nas aulas de estatística iniciais, este é a fase que você pensou era **tudo** de análise de dados
- Construir modelos
- Avaliar modelos
- Testar hipóteses
- Construir visualizações
  - Exploração (para você)
  - Apresentação (para o público)

# Comunicar Resultados

- Escrever relatórios, fazer apresentações, etc.
- *R Markdown* ajuda inestimável em tudo isso
  - Todos os slides, lições de casa, etc. preparados em *R Markdown* e *Quarto*

# Divisão de Trabalho Verdadeiro

- Importar e Limpar Dados - **60 - 70 %**
- Explorar Dados - 20 - 30 %
- Communicar Resultados - 10 - 20 %