Prática no R! Roteiro 3 - Análise Exploratória II

Elaborado por Luis Felipe Bortolatto da Cunha

21 de setembro de 2020

Contents

1. Introdução

Você pode baixar este roteiro em formato PDF neste endereço.

Este roteiro tem como objetivo apresentar novas funções para a execução **análise exploratória**, mas dessa vez baseadas no pacote `tidyverse` que estão organizadas em torno de um **conjunto de princípios básicos** que tornam a análise de dados mais acessível para quem está tendo o primeiro contato com a programação.

A Figura 1 ilustra 6 etapas fundamentais da análise de dados que o software R auxilia a executar.

Nesse roteiro, utilizaremos duas bases de dados demográficos e de consumo de água de 2010, extraídas do Censo Demográfico (IBGE) e Sistema Nacional de Informações sobre Saneamento (SNIS), para uma amostra de 4.417 municípios, organizada por Carmo et al., 2013.

As bases de dados estão disponíveis para download no endereço abaixo:

https://1drv.ms/u/s!AjettDH-3Gbni9kLp9z7BUHOtvlrng?e=DKVRmS

1.1. Importando os dados

As duas bases de dados podem ser importadas conforme as instruções do Roteiro 2.

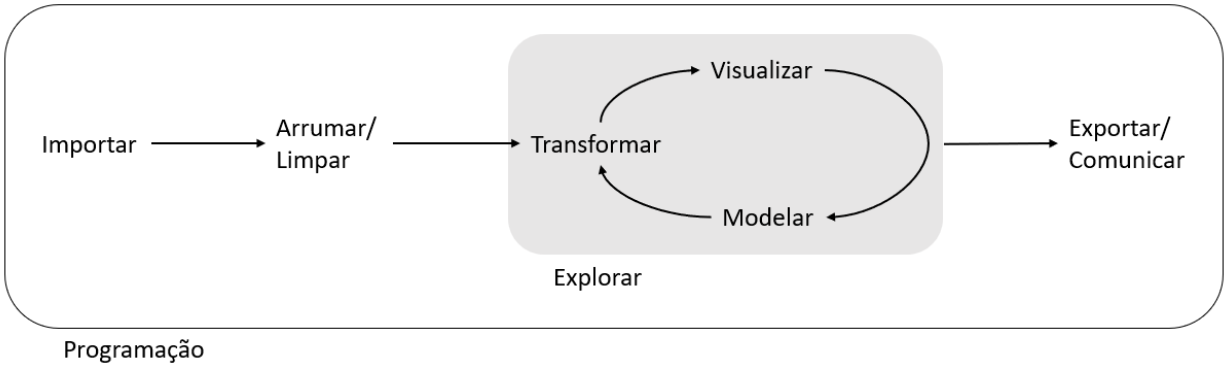


Figure 1: Etapas da análise de dados. Fonte: adaptado de Grolemund & Wickham, 2017.

Como elas estão hospedadas na nuvem, também podem ser importadas com o endereço web como argumento, ao invés do endereço local, conforme o exemplo abaixo. É importante lembrar que a importação pelo endereço web exige conexão com a internet!

```
agua1 <- read.csv2("https://raw.githubusercontent.com/luisfelipebr/mti/master/dados/agua1.csv",
                   encoding="UTF-8")
```

```
rede1 <- read.csv2("https://raw.githubusercontent.com/luisfelipebr/mti/master/dados/rede1.csv",
                   encoding="UTF-8")
```

1.2. Explorando os dados

A função `names()` permite ver o nome das variáveis das bases de dados. Como os nomes estão codificados, as tabelas abaixo fornecem uma correspondência do código com a descrição da variável.

```
names(agua1)
```

```
##  [1] "ID_IBGE"   "ID_SNIS"   "NOME_MUN"  "UF"        "REGIAO"    "PIB"
##  [7] "RENDAPITA" "GINI"      "IDH"       "IDH_CLASS" "GE012"     "AG001"
## [13] "AG020"     "AG022"
```

Código	Descrição
ID_IBGE	Código IBGE (7 dígitos)
ID_SNIS	Código IBGE (6 dígitos)
NOME_MUN	Nome do Município
UF	Unidade da Federação
REGIAO	Região do País
PIB	PIB 2010
RENDAPITA	Renda per Capita 2010
GINI	Índice GINI 2010
IDH	Índice de Desenvolvimento Humano 2010
IDH_CLASS	Classificação do Índice de Desenvolvimento Humano 2010: Muito Alto >= 0,9; Alto >= 0,8; Médio >= 0,5; Baixo < 0,5.
GE012	População Total Residente no Município
AG001	População Total Atendida com Abastecimento de Água
AG020	Volume Micromedido nas Economias Residenciais Ativas de Agua - 1.000 m3/ano
AG022	Quantidade de Economias Residenciais Ativas Micromedidas

```r
names(rede1)
```

```
## [1] "ID_IBGE" "UF"      "REGIAO"  "DOMICIL" "REDE"
```

Código	Descrição
ID_IBGE	Código IBGE (7 dígitos)
UF	Unidade da Federação
REGIAO	Região do País
DOMICIL	Quantidade de Domicílios
REDE	Quantidade de Domicílios com Acesso à Rede Geral de Água

2. Instalando e abrindo o pacote "tidyverse"

Para instalar o pacote `tidyverse`, usaremos a função `install.packages()`, com o nome do pacote entre aspas como argumento.

ATENÇÃO: só é necessário instalar o pacote uma única vez em cada computador.

```r
install.packages("tidyverse")
```

Após instalar o pacote, utilizamos a função `library()` para abri-lo, com o nome do pacote como argumento. Não é necessário colocar o nome do pacote entre aspas.

ATENÇÃO: é necessário abrir o pacote todas as vezes que o programa R/RStudio é aberto ou a sessão é reiniciada.

```r
library(tidyverse)
```

3. Conhecendo o operador pipe (%>%)

O **operador pipe (%>%)** é a principal contribuição do `tidyverse` à análise de dados. Ele é uma ferramenta poderosa para **expressar** de forma clara **uma sequencia de operações**.

Para entender como ele funciona, podemos traçar o paralelo com uma receita de bolo (curso-r, 2018). Usando o R básico, a receita poderia ser representada da seguinte forma:

```r
esfrie(
  asse(
    coloque(
      bata(
        acrescente(
          recipiente(
            rep("farinha", 2),
            "água",
            "fermento",
            "leite",
            "óleo"
          ),
          "farinha",
          ate = "macio"
        ),
        duracao = "3min"
      ),
      lugar = "forma",
```

```
      tipo = "grande",
      untada = TRUE
    ),
    duracao = "50min"
  ),
  lugar = "geladeira",
  duracao = "20min"
)
```

O R básico processa o que está em parênteses primeiro (assim como na matemática), tornando a leitura do código pouco intuitiva. O pipe funciona como um tubo que **conecta as etapas da análise de dados em ordem cronológica**, facilitando a leitura. O mesmo código escrito com o R básico, poderia ser representado com o pipe da seguinte forma:

```
recipiente(rep("farinha", 2), "água", "fermento", "leite", "óleo") %>%
  acrescente("farinha", até = "macio") %>%
  bata(duração = "3min") %>%
  coloque(lugar = "forma", tipo = "grande", untada = TRUE) %>%
  asse(duração = "50min") %>%
  esfrie("geladeira", "20min")
```

Esperamos que esse exercício de abstração tenha convencido-o a adotar o operador pipe na análise de dados. A partir desse roteiro, o pipe será usado em todos os exemplos, apesar do seu uso ser opcional.

4. Filtrar e Selecionar (filter, select)

Enquanto no R básico a criação de subconjuntos exige o uso de colchetes, o tidyverse adiciona as funções: `filter()` para filtrar observações e `select()` para selecionar variáveis.

Tomemos o mesmo exemplo do Roteiro 2 - para visualizar o nome e IDH dos municípios que satisfazem as condições: Unidade da Federação é igual a São Paulo E IDH maior que 0,85.

No R básico, isso pode ser feito da seguinte forma:

```
agua1[which(agua1$UF == "SP" & agua1$IDH > 0.85),c("NOME_MUN", "IDH")]
```

```
##                 NOME_MUN   IDH
## 3273  Águas de São Pedro 0.908
## 3375            Campinas 0.852
## 3560             Jundiaí 0.857
## 3754      Ribeirão Preto 0.855
## 3773            Saltinho 0.851
## 3795 Santana de Parnaíba 0.853
## 3809              Santos 0.871
## 3812  São Caetano do Sul 0.919
## 3903             Vinhedo 0.857
```

No tidyverse, ficaria assim:

```
agua1 %>%
  filter(UF == "SP" & IDH > 0.85) %>%
  select(NOME_MUN, IDH)
```

```
##              NOME_MUN   IDH
## 1  Águas de São Pedro 0.908
## 2            Campinas 0.852
## 3             Jundiaí 0.857
```

```
## 4      Ribeirão Preto 0.855
## 5            Saltinho 0.851
## 6 Santana de Parnaíba 0.853
## 7              Santos 0.871
## 8  São Caetano do Sul 0.919
## 9             Vinhedo 0.857
```

5. Transformações (mutate)

Definir novas variáveis no R básico é bem simples. Mas para incluir essa etapa no fluxo do `tidyverse`, podemos usar a função `mutate()`.

Seguindo o exemplo do Roteiro 2, podemos definir as variáveis CONSUMO1 e CONSUMO2 da seguinte forma com o R básico:

- `CONSUMO1`: Consumo de Água per capita - População Total - m3/ano (AG020/GE012)
- `CONSUMO2`: Consumo de Água per capita - População Atendida - m3/ano (AG020/AG001)

```
agua1$CONSUMO1 <- agua1$AG020 * 1000 / agua1$GE012
agua1$CONSUMO2 <- agua1$AG020 * 1000 / agua1$AG001
```

A mesma operação ficaria da seguinte forma no tidyverse:

```
agua2 <- agua1 %>%
  mutate(CONSUMO1 = AG020 * 1000 / GE012,
         CONSUMO2 = AG020 * 1000 / AG001)
```

Vamos chamar o novo objeto de agua2, pois ele é igual à tabela que foi exportada com esse nome no Roteiro 2.

6. União (join)

Como supõe o nome, a união serve para unir duas bases de dados a partir de um identificador.

No tidyverse, a união pode ser feita usando uma de quatro funções, a depender do seu objetivo:

- `left_join()`: Adiciona à primeira base as variáveis da segunda base que possuem correspondência ao identificador.
- `right_join()`: Adiciona à segunda base as variáveis da primeira base possuem correspondência ao identificador.
- `inner_join()`: O resultado é uma base de dados que exclui as observações sem correspondência ao identificador.
- `full_join()`: O resultado é uma base de dados com todas as observações, da primeira e segunda base, adicionando valores faltantes (NA) quando não há correspondência ao identificador.

Neste roteiro vamos explorar a proporção de domicílios com acesso à rede geral de água (PROPREDE). A proporção de domicílios com acesso à rede geral pode ser calculada com a função `mutate()` para dividir a quantidade de domicílios com acesso à rede geral pela quantidade total de domicílios (REDE/DOMICIL).

- `PROPREDE`: Proporção de Domicílios com com Acesso à Rede Geral de Água (REDE/DOMICIL)

Como estes dados estão fragmentados em duas bases de dados diferentes, vamos uni-las com a função `full_join()`.

Além disso, como as duas bases de dados possuem variáveis repetidas usaremos a opção `select()` para selecionar apenas as variáveis de interesse.

Todas essas etapas da análise de dados podem ser escritas em um único fluxo de análise de dados com o tidyverse, da seguinte forma:

Figure 2: Tipos de união. Fonte: Surles, 2017.

```
agua_rede1 <- rede1 %>%
  mutate(PROPREDE = REDE/DOMICIL) %>%
  select(-c(UF, REGIAO)) %>%
  full_join(agua2, by = "ID_IBGE")
```

ATENÇÃO: o sinal menos (-) significa que vamos selecionar TODAS as variáveis EXCETO as especificadas.

7. Agrupar e resumir (group_by, summarize)

Agrupar e resumir são duas etapas da análise de dados, geralmente aplicadas juntas, para calcular estatísticas básicas em subconjuntos.

Na aula teórica, aprendemos a calcular o intervalo de confiança.

Para explorar a desigualdade de PROPREDE1, vamos primeiro filtrar os valores faltantes (`drop_na()`), agrupar por regiões (`group_by()`), resumir estatísticas básicas (`summarize()`) e calcular o intervalo de confiança para cada região (`mutate()`) com um intervalo de confiança de 95% (escore-z da curva normal é igual a 1,96). O resultado será salvo como o objeto de nome `tabela_PROPREDE`.

```
tabela_PROPREDE <- agua_rede1 %>%
  drop_na(PROPREDE) %>%
  group_by(REGIAO) %>%
  summarize(n_obs = n(),
            media = mean(PROPREDE),
            desvio_padrao = sd(PROPREDE)) %>%
  mutate(erro = 1.96*desvio_padrao/sqrt(n_obs),
         limite_superior = media + erro,
         limite_inferior = media - erro)
```

É possível visualizar essa tabela chamando o seu nome.

```
tabela_PROPREDE
```

```
## # A tibble: 5 x 7
##   REGIAO       n_obs media desvio_padrao    erro limite_superior limite_inferior
##   <chr>        <int> <dbl>         <dbl>   <dbl>           <dbl>           <dbl>
## 1 Centro-Oeste   466 0.694         0.160 0.0146            0.709           0.680
## 2 Nordeste      1789 0.654         0.189 0.00875           0.662           0.645
## 3 Norte          449 0.498         0.232 0.0214            0.520           0.477
## 4 Sudeste       1668 0.766         0.164 0.00786           0.774           0.758
## 5 Sul           1188 0.716         0.194 0.0110            0.727           0.705
```

8. Gráficos (ggplot)

Dentre os componentes do tidyverse, está o pacote `ggplot2`, que permite a criação de gráficos a partir de uma linguagem universal entre os programadores e designers, chamada de a gramática dos gráficos. Não é o objetivo deste roteiro explorar a visualização de dados em detalhes, mas os comandos abaixo mostram como construir alguns tipos de gráficos.

Com o pacote `ggplot2`, qualquer gráfico bidimensional pode ser construído partindo da função `ggplot()`, que segue a seguinte lógica:

ggplot(data = *base_de_dados*, aes(x = *codigo_variavel*, y = *codigo_variavel*)) + geom_*tipo_de_geometria*()

Em que os argumentos obrigatórios são:

- `data` = base de dados (tabela)
- `aes()` = estética. É aqui que você vai especificar os eixos x e y (se houver), assim como atributos adicionais como cor, tamanho, espessura e formato dos pontos/linhas/polígonos

Alguns dos tipos de geometria mais comuns são:

- `+ geom_histogram()` = histograma
- `+ geom_boxplot()` = box-plot
- `+ geom_point()` = gráfico de dispersão
- `+ geom_line()` = gráfico de linhas
- `+ geom_abline()` = reta de tendência (correlação)
- `+ geom_smooth()` = reta de tendência (regressão)
- `+ geom_bar()` = gráfico de barras (pré-tabulação/contagem)
- `+ geom_col()` = gráfico de barras (pós-tabulação/contagem)
- `+ geom_errorbar()` = barras de erro

Alguns argumentos adicionais que podem ser explorados são:

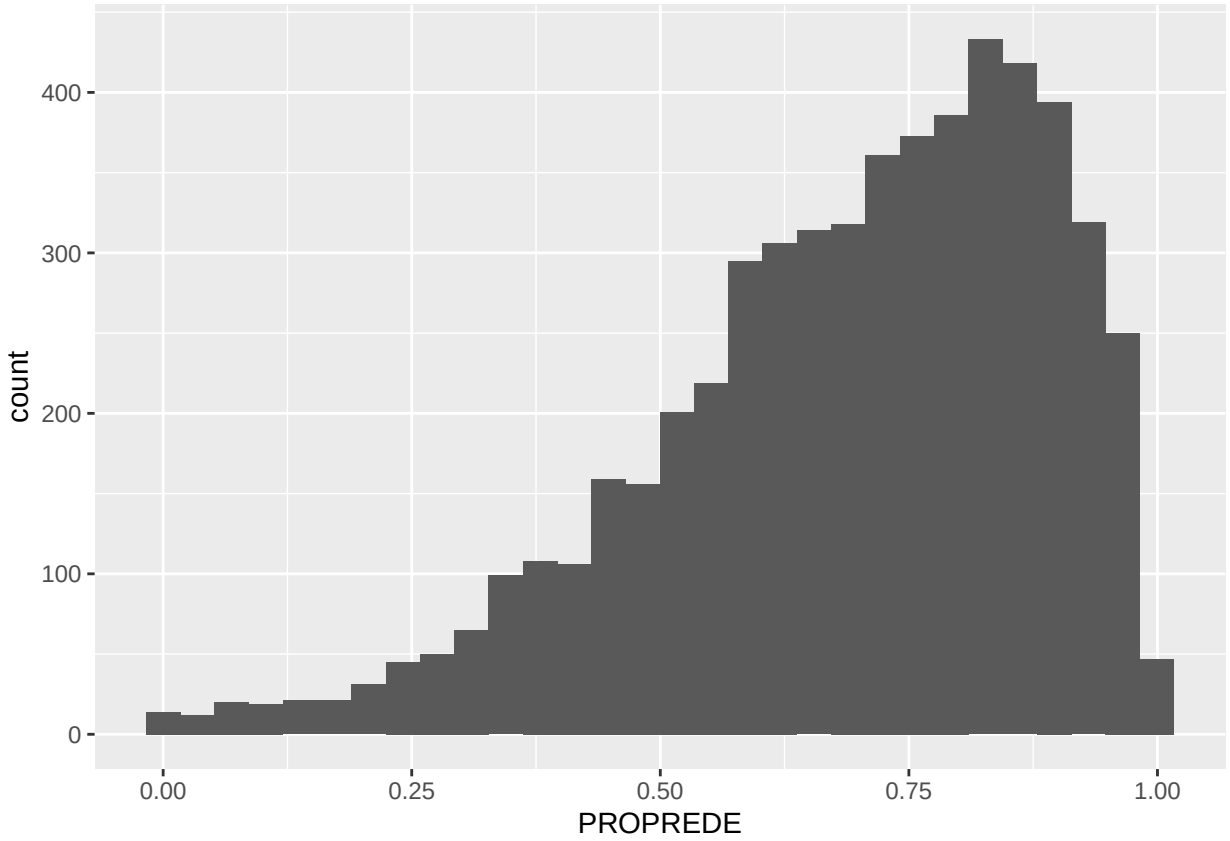
- `+ facet_wrap(~*codigo_variavel*)` = replica um gráfico para diferentes classes
- `+ coord_flip()` = inverte as coordenadas x e y
- `+ ggtitle("Título em parênteses")` = adiciona um título
- `+ theme()` = para customizar os componentes não relacionados aos dados

Argumentos adicionais podem ser consultados acessando a bibliografia complementar.

Vamos usar a função `ggplot()` para explorar com maior profundidade as desigualdades regionais relacionadas à PROPREDE.

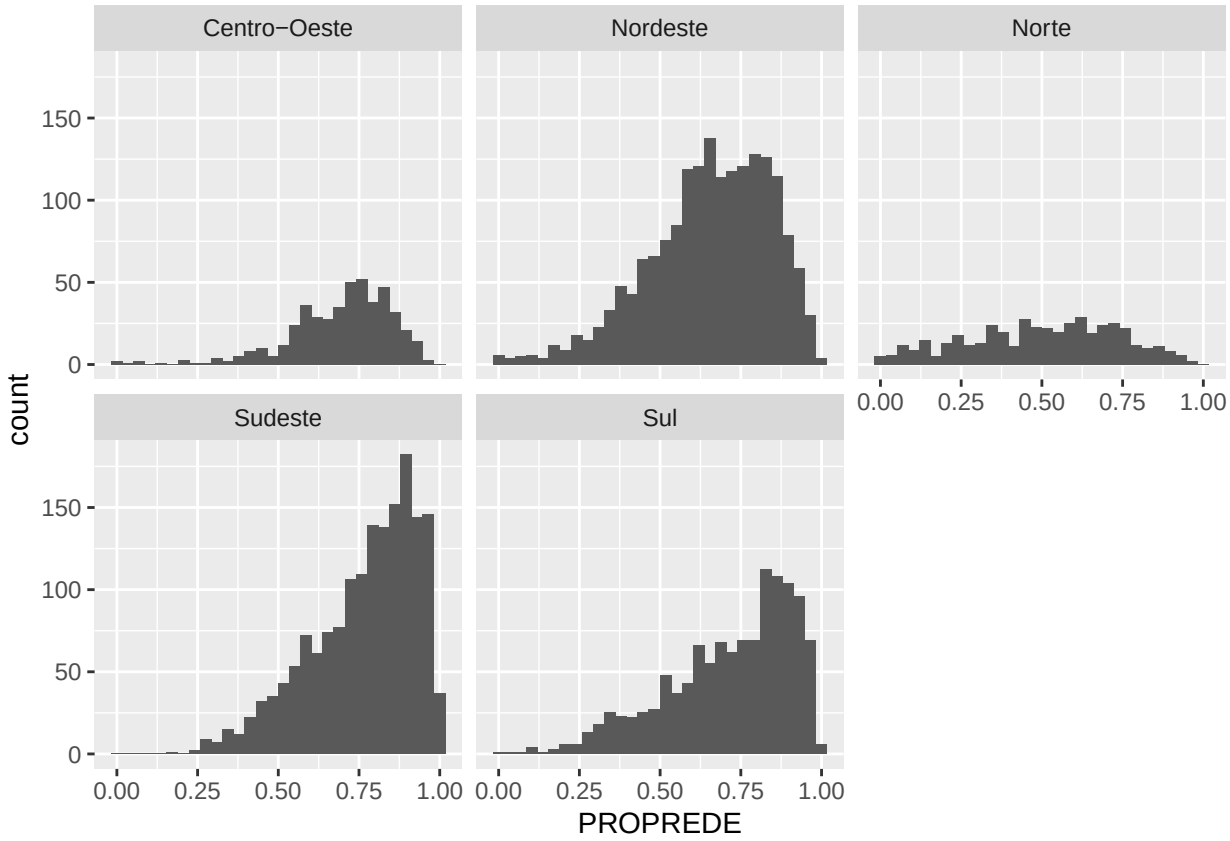
O código abaixo desenha um histograma de PROPREDE:

```
ggplot(data = agua_rede1, aes(x = PROPREDE)) +
  geom_histogram()
```



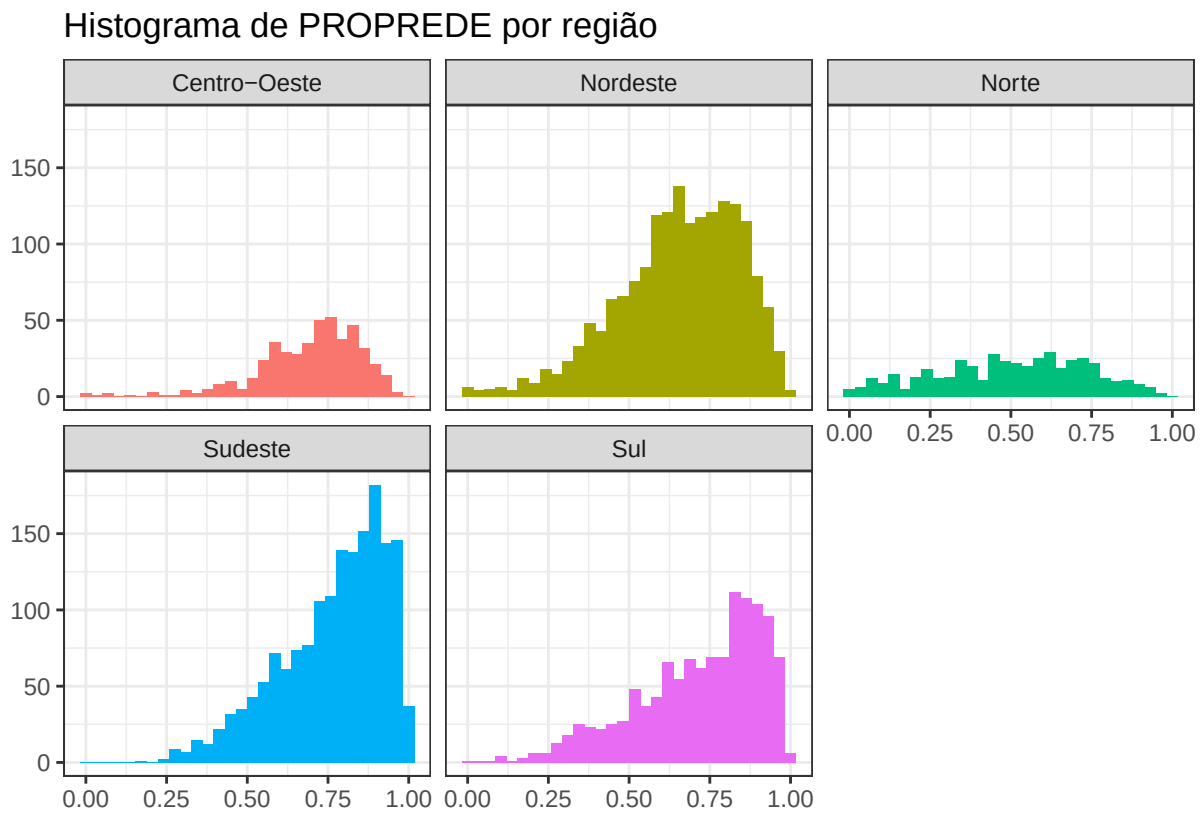
Adicionando o argumento `facet_wrap()` é possível replicar esse gráfico para cada região:

```
ggplot(data = agua_rede1, aes(x = PROPREDE)) +
  geom_histogram() +
  facet_wrap(~REGIAO)
```



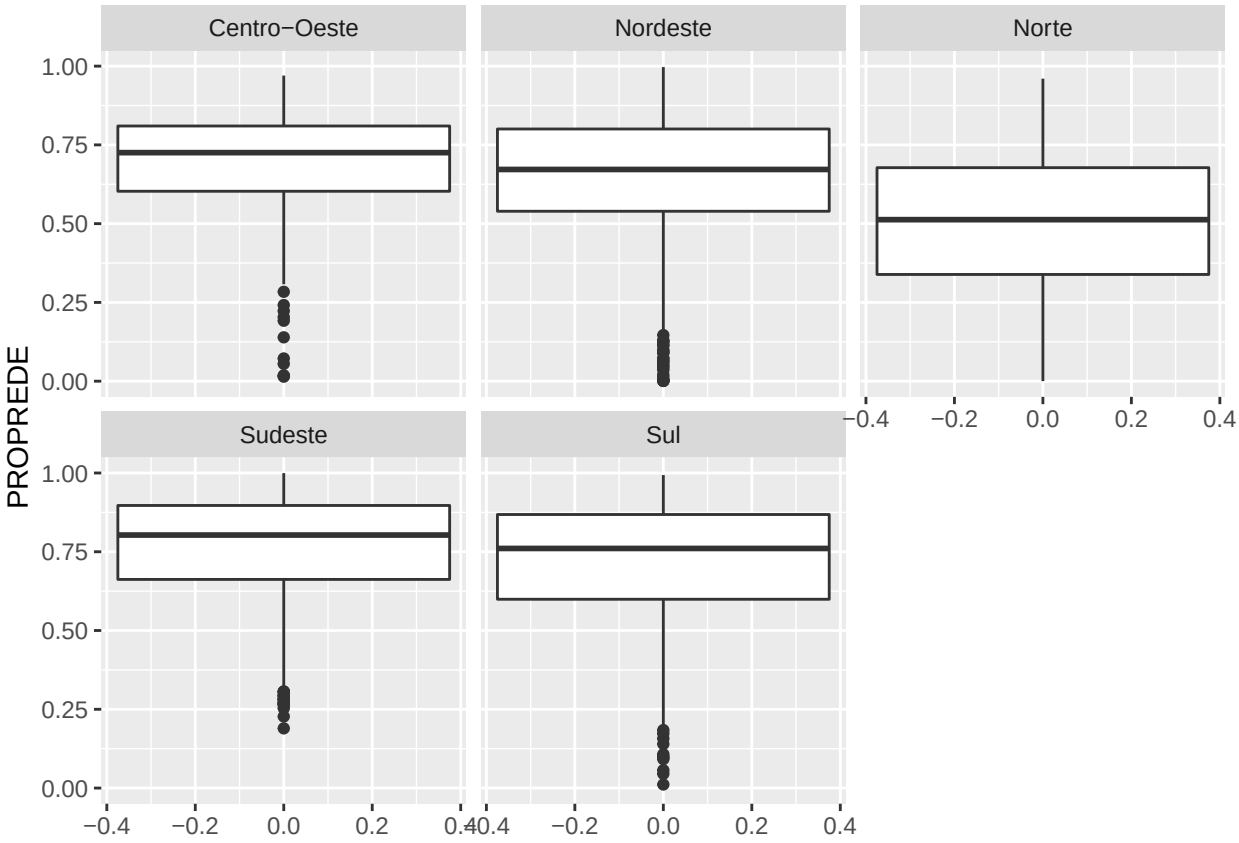
Adicionando mais alguns argumentos, a visualização do gráfico pode ser aprimorada ainda mais, embora essa etapa é opcional.

```
ggplot(data = agua_rede1, aes(x = PROPREDE, fill = REGIAO)) +
  geom_histogram() +
  facet_wrap(~REGIAO) +
  ggtitle("Histograma de PROPREDE por região") +
  xlab("") +
  ylab("") +
  theme_bw() +
  theme(legend.position = "none")
```



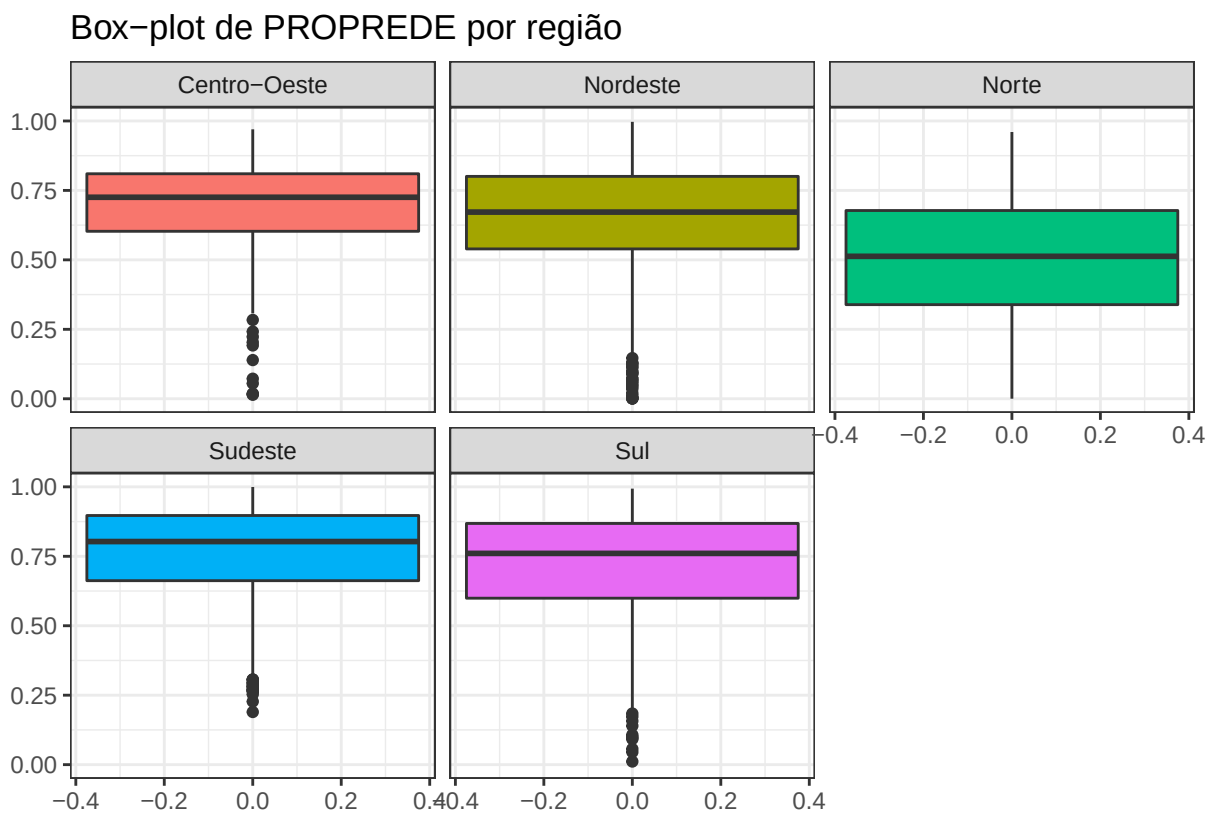
O código abaixo desenha um box-plot que compara `PROPREDE` entre as regiões.

```
ggplot(data = agua_rede1, aes(y = PROPREDE)) +
  geom_boxplot() +
  facet_wrap(~REGIAO)
```



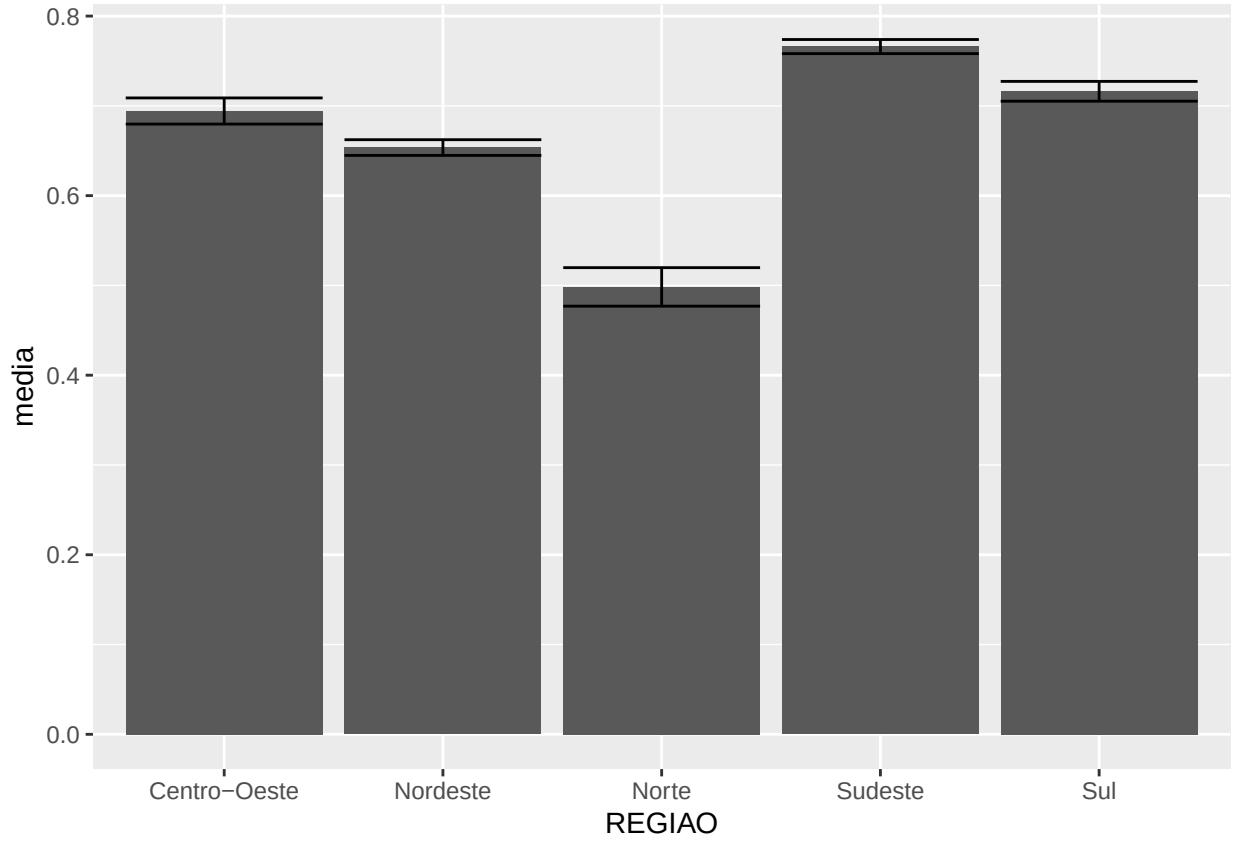
Também é possível adicionar alguns argumentos para aprimorar a visualização desse gráfico.

```
ggplot(data = agua_rede1, aes(y = PROPREDE, fill = REGIAO)) +
  geom_boxplot() +
  facet_wrap(~REGIAO) +
  ggtitle("Box-plot de PROPREDE por região") +
  xlab("") +
  ylab("") +
  theme_bw() +
  theme(legend.position = "none")
```

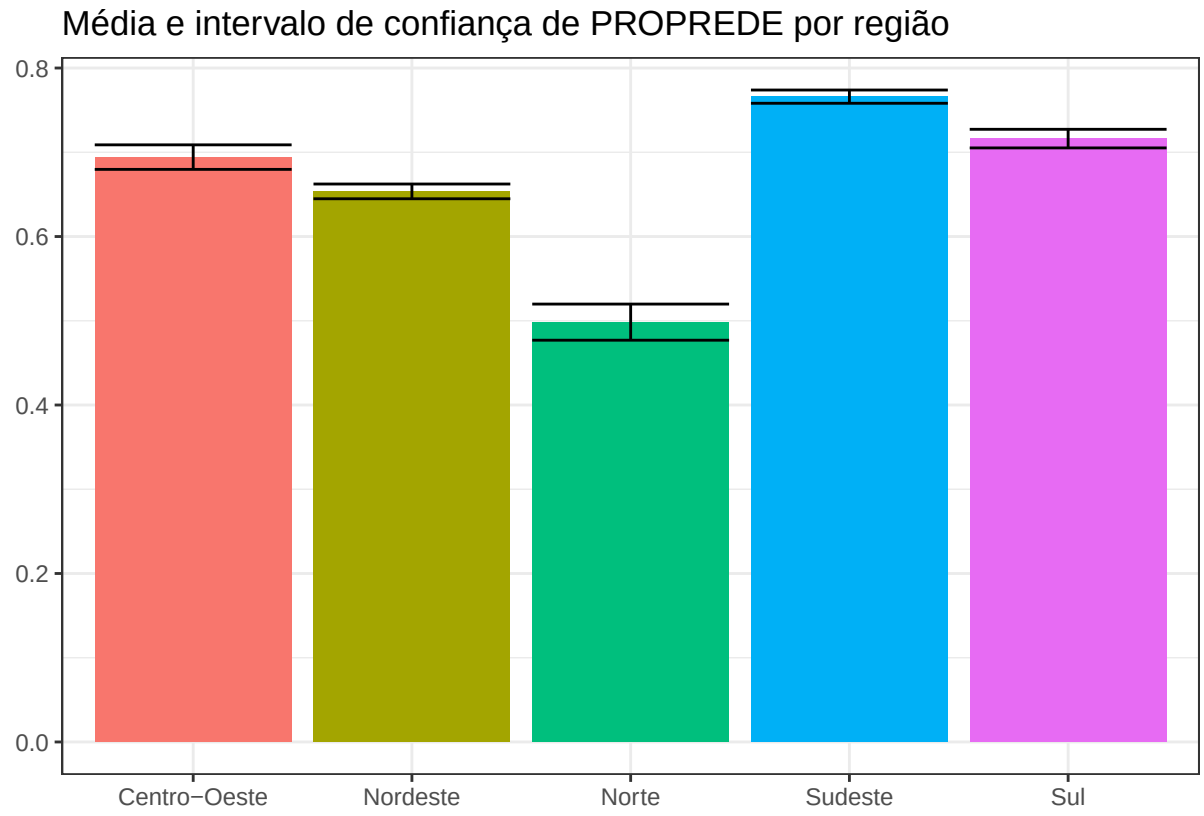


Por fim, o gráfico abaixo compara a média e intervalo de confiança de PROPREDE por região.

```
ggplot(data = tabela_PROPREDE, aes(x = REGIAO, y = media)) +
  geom_col() +
  geom_errorbar(aes(ymin = limite_inferior, ymax = limite_superior))
```



```r
ggplot(data = tabela_PROPREDE, aes(x = REGIAO, y = media, fill=REGIAO)) +
  geom_col() +
  geom_errorbar(aes(ymin = limite_inferior, ymax = limite_superior)) +
  ggtitle("Média e intervalo de confiança de PROPREDE por região") +
  xlab("") +
  ylab("") +
  theme_bw() +
  theme(legend.position = "none")
```



9. Exportando a base de dados

Não se esqueça de exportar a base de dados criada neste roteiro (`agua_rede1`), pois partiremos dela na próxima aula. Isso pode ser feito com a função `write.csv2`.

```r
write.csv2(agua_rede1, "dados/agua_rede1.csv")
```