

Governança e Qualidade de Dados

Laboratório Qualidade de Dados (utilizando R)

Arnaldo Vitaliano, MSc.

2017

Governança e Qualidade de Dados

Laboratório Qualidade de Dados (utilizando R)

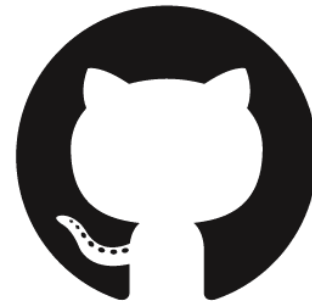
Arnaldo Vitaliano, MSc.

2017



- Laboratório Prático
- Objetivos:
 - Utilizar técnicas de qualidade de dados
 - Utilizar linguagem R para realizar as atividades
 - Utilizar controle de versão (git)
 - Utilizar repositório para código e documentação (github)

Quem estudou antes da aula?



GitHub



Não vamos nos aprofundar nestes assuntos



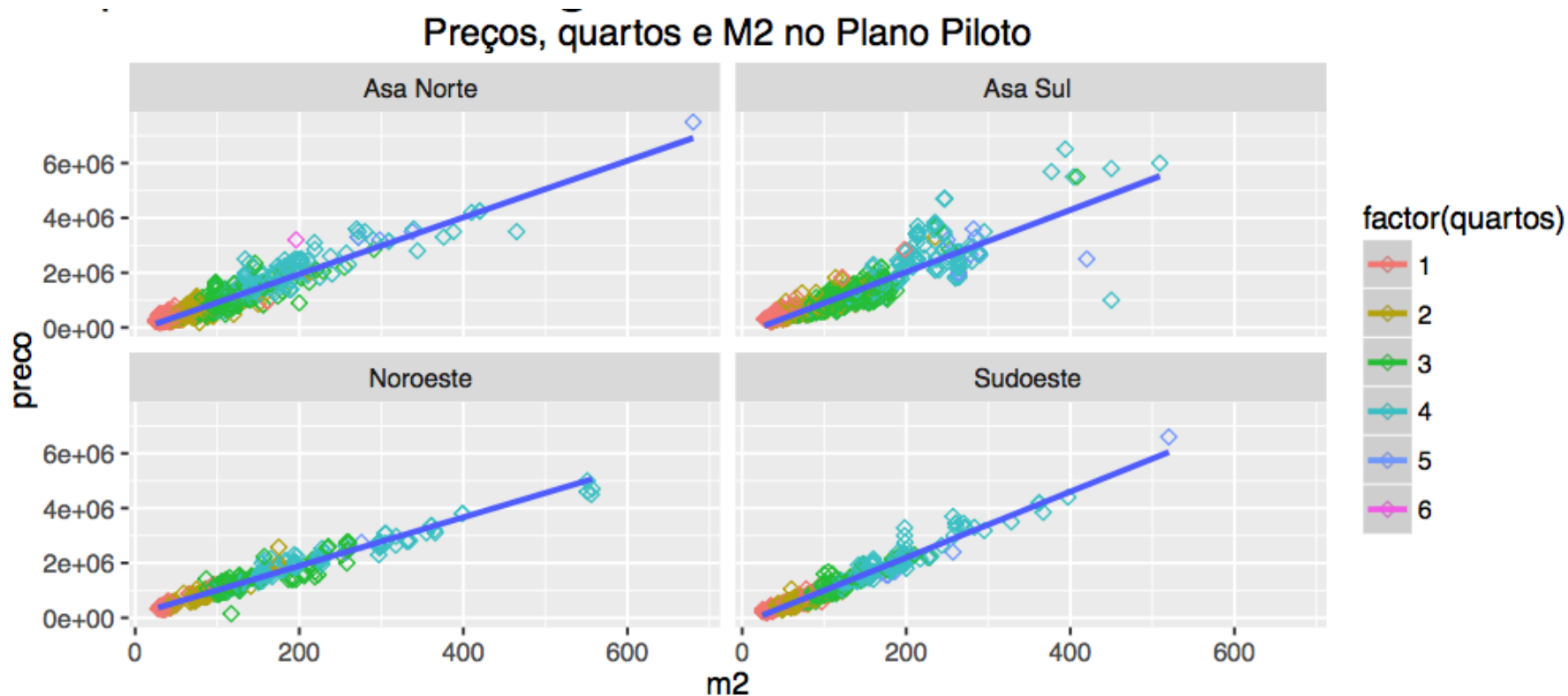
Acessar o repositório para este laboratório no github:

avitaliano/iesb-posbi-dataquality



- R é uma linguagem de programação com foco em análise de dados;
- Criada na Nova Zelândia, por 2 estatísticos: Ross Ihaka e Robert Gentleman;
- É uma linguagem de programação interpretada, voltada à interação dinâmica com os dados e modelos;
- É gratuito e compatível com todas as plataformas;

Gráficos e estatística: gráfico de dispersão com preço contra metro quadrado por bairro, cor dos pontos de acordo com número de quartos e linha de regressão.



Mapas e estatística: Webscraping de dados de roubos e download do mapa de Brasília no Google Maps; manipulação dos dados e construção de gráficos de calor geolocalizados.



Carlos Cinelli e Edson Bastos

1. Abra o R Gui

2. Escreva os seguintes comandos:

- `1 + 1`
- `plot(1:10)`
- `hist(rnorm(1000))`
 - (repita várias vezes e veja o que acontece)

Apesar de o R vir com uma interface gráfica bem interessante, existe um IDE (Integrated Development Environment - Ambiente de Desenvolvimento Integrado) chamado RStudio, com várias funcionalidades e gratuito.

O RStudio tem algumas vantagens em relação ao R Gui:

- Highlight do código;
- Autocomplete;
- Match automático de parenteses e chaves;
- Interface intuitiva para objetos, gráficos, script;
- Projetos (com controle de versão);
- Interação com HTML, entre outras.

Vamos criar um projeto do Rstudio para nosso laboratório. Esta é uma maneira simples de gerenciar scripts, dados e documentos relacionados ao projeto R.

- Criar projeto de Rstudio em pasta existente;
- Selecionar o nosso repositório clonado do GitHub;
- Abrir o script exemplos.R

- $1 + 1$ (CTRL+ENTER): roda o comando
- $3 * 4$
- $2 - (4 * 2) / 5$
- 4^2 #exponencial
- $21 \% / \% 4$ #divisão inteira
- $21 \% \% 4$ #resto de divisão inteira

- `1:10`
- `plot(1:10)`
- `rnorm(10)` #números na distribuição normal
- `hist(rnorm(1000))` #histograma

- Atribuição:
- $x \leftarrow 100$ #atribui o valor 100 à variável x
- $y \leftarrow 23$ #atribui o valor 23 à variável y

- $x + y$
- $x * y$
- x / y

- Concatenação
- `vetor <- c(1, 4, 5, 10, 2, x, y)` #concatena os valores e atribui à variável “vetor”
- Vetores são sempre de um único tipo de dados
 - Números
 - Inteiros
 - Texto
 - Números complexos
 - Lógicos (booleanos)

- `numero <- c(546.90, 10, 789)`
- `inteiro <- c(1L, 98L)` `# note o L`
- `complexo <- c(20i, 2+9i)` `# note o i`
- `texto <- c("meu", "vetor", "com", "varias", "palavras")`
 `# note as aspas`
- `logico <- c(TRUE, FALSE, TRUE)` `# sempre maiúsculo`

- Classe dos vetores
 - numero
 - inteiro
 - complexo
 - texto
 - logico

- `as.character(numero)`
- `# TRUE -> 1, FALSE -> 0`
- `as.numeric(logico)`
- `# Não faz sentido`
- `as.numeric(texto)`
- `# Faz sentido`
- `as.numeric("129873")`

- `str(x)`
- `str(y)`
- `str(vetor)`
- `str(numero)`
- `str(inteiro)`
- `str(complexo)`
- `str(texto)`
- `str(logico)`

- `length(x)`
- `length(y)`
- `length(vetor)`
- `length(texto)`

- `x <- c(1, 3, -5, 10)`
- `abs(x)` # valor absoluto
- `exp(x)` # exponencial em e
- `sqrt(x)` # raiz quadrada
- `factorial(3)` # fatorial
-
- `choose(20,2)` # análise combinatória

- `mean(x)` `#media`
- `sum(x)` `#somatoria`
- `prod(x)` `#produtória`
- `cumsum(x)` `#somatório cumulado`
- `cumprod(x)` `# produtório cumulado`

- `var(x)` # variância
- `sd(x)` # desvio padrão
- `median(x)` # mediana
- `min(x)` # mínimo
- `max(x)` # máximo

- $1 == 0$ # igual
- $1 != 0$ # diferente
- $1 > 0$ # maior que
- $1 < 0$ # menor que
- $! 1 < 0$ # negação
- $\text{TRUE} \& \text{FALSE}$ # operador AND
- $\text{TRUE} | \text{FALSE}$ # operador OR

- Rode o seguinte código:
- `set.seed(1)`
- `horas_trabalhadas <- rnorm(1000, 8, 0.5)`
- `valor_por_hora <- rnorm(1000, 30, 2)`

1. Veja a estrutura dos vetores. Qual sua classe?
2. Quantas observações cada vetor tem?
3. Quais os valores máximos e mínimos?
4. Crie um vetor com o valor recebido por dia. Encontre o mínimo e o máximo.
5. Qual o valor total recebido no período?

- `str(horas_trabalhadas); str(valor_por_hora)`
- `class(horas_trabalhadas); class(valor_por_hora)`
- `length(horas_trabalhadas); length(valor_por_hora)`
- `min(horas_trabalhadas); min(valor_por_hora)`
- `max(horas_trabalhadas); max(valor_por_hora)`
- `valor_recebido_dia <- horas_trabalhadas *
valor_por_hora`
- `sum(valor_recebido_dia)`

- Toda vez que você abre uma sessão do R, ele realizará operações de leitura e gravação de dados no diretório de trabalho.
- *# qual é o diretório de trabalho atual?*
*# no seu computador o resultado vai ser diferente **getwd()***
get working directory
[1] "C:/"
- A qualquer momento você pode alterar o diretório de trabalho com `setwd()`.
- **`setwd("D:/Curso de R") ; getwd()`**

- Note que o separador é o contrário do que se usa no windows, pois a barra \ é um comando especial. Uma opção, caso você queira usar o padrão do windows, é usar duas barras.
- **setwd("C:\\diretorio1\\diretorio2")**
Existe, ainda, uma função de conveniência que cria o caminho do arquivo para você a `file.path()`
- **file.path("C:", "diretorio1", "diretorio2", "arquivo.csv")**
- **## [1] "C:/diretorio1/diretorio2/arquivo.csv"**

Uma das formas mais convenientes de importar e exportar dados pelo R e por meio de arquivos .csv. O csv é um formato entendido por virtualmente quase todo software. Existe uma série de funções (derivadas da `read.table`) que fazem este trabalho, veja mais em `?read.table`. Argumentos que necessitam atenção especial:

- `dec`: determina o símbolo utilizado para decimal. No Brasil, utilizamos “,” mas em muitos outros lugares o padrão é “.”
- `sep`: como os dados estão separados? Por tab (`\t`), por vírgula (`,`), por ponto e vírgula (;), por espaço (`" "`)?
- `fileEncoding`: o padrão do R, em geral, é trabalhar com caracteres ASCII. No Brasil, são comuns os padrões latin1 ou UTF-8. Ler o arquivo com o padrão errado pode resultar em caracteres “estranhos”.

Para ilustrar como salvar um csv, vamos utilizar a função `write.csv2()`. Ela é igual à `write.csv()` mas já tem como padrão `sep = ";"` e `dec = ","`, que são comuns no Brasil.

- **mtcars**
- **`write.csv2(mtcars, "mtcars.csv")`**

Para ilustrar como ler um csv, vamos utilizar a função `read.csv()`, colocando os parâmetros corretos para leitura:

- `carros <- read.csv("mtcars.csv", dec = ",", sep = ";")`
- Neste caso poderíamos ter lido também com `read.csv2()` que já teria os parâmetros desejados.

Data Frames são estruturas da linguagem R muito parecidas com tabelas relacionais.

- Data frames possuem colunas com nomes
- Cada coluna tem um único tipo (classe)
- Podemos aplicar várias funções para manipular dados de data frames, como agrupar, filtrar, selecionar, somar, etc.

Carregando o data frame que utilizaremos no laboratório (censo):

- `censo <- read.csv2("data/Censo.csv", stringsAsFactors = FALSE)`
- `str(censo)` `#estrutura do data frame`
- `names(censo)` `#nomes das colunas do data frame`
- `View(censo)` `#visualizar o data frame`

Acessar colunas do data frame

- `censo[3]` #pelo índice da coluna
- `censo$Nome` #pelo nome da coluna
- `censo["Nome"]` #outra maneira de acesso pelo nome
- `censo[c(2,3)]` #várias colunas ao mesmo tempo
- `censo[c("Nome","CPF")]`

Selecionando coluna para realizar análise:

- `altura <- censo$Altura.cm` `#joga a coluna Altura para um vetor`
- `summary(altura)` `#verifica estatísticas básicas`
- `media.altura <- mean(altura)` `#média`
- `desvio.altura <- sd(altura)` `#desvio padrão`
- `altura > media.altura + 4*desvio.altura` `# valores que são maiores que média + 4*desvios`
- `gigantes <- altura[altura > media.altura + 4*desvio.altura]`
- `gigantes`
- `# [1] 270 268 267 262 266 267 251 263 266 263 256 253 259 254 256 270 269 257`

Adicionando colunas ao data frame

- `Dataframe$novacoluna <- valores`
- Exercício:
- Incluir a coluna IMC no data frame do censo
- Fórmula:
 - $IMC = PESO \text{ (em KG)} / ALTURA^2 \text{ (em M)}$

Adicionando colunas ao data frame

- `Dataframe$novacoluna <- valores`
 - Exercício:
 - Incluir a coluna IMC no data frame do censo
 - Fórmula:
 - $IMC = PESO \text{ (em KG)} / ALTURA^2 \text{ (em M)}$
 - Solução: `censo$IMC<- censo$Peso.kg / (censo$Altura.cm/100)^2`
 - `summary(censo$IMC)`
- # Min. 1st Qu. Median Mean 3rd Qu. Max.
- # 5.421 23.650 26.260 26.820 29.320 475.000

Removendo colunas do data frame

- `Dataframe$coluna <- NULL`
- Exemplo:
- `censo$IMC <- NULL`
- `censo$IMC`

Funções para manipulação de data frames

- `head(censo)` `#mostra as primeiras linhas`
- `tail(censo)` `#mostra as últimas linhas`
- `Subset`
 - `censo[1,]` `#primeira linha, todas colunas`
 - `censo[1, 1]` `#primeira linha, primeira coluna`
 - `censo[, 3]` `#todas linhas, terceira coluna`
 - `censo[1:10, c(3,4,5)]` `#primeiras 10 linhas, e colunas 3,4,5`
 - `censo[, c("Nome", "CPF", "Sexo")]` `#colunas nomeadas`
- `unique(censo$Sexo)`

Fornece meios de utilizar funções de data frames de maneira similar à linguagem SQL

- **filter**: filtra um data.frame com vetores lógicos. Por exemplo, filtrar valores de Salario menores ou maiores do que determinado nível.
select: seleciona uma ou mais colunas de um data.frame. Por exemplo, selecionar a coluna de idade.
- **mutate**: cria uma nova coluna. Por exemplo, criar a coluna IMC.
arrange: ordena o data.frame com base em uma coluna. Por exemplo, ordenar do maior ao menor Salario.
- **group_by**: agrupa um data.frame por índices. Por exemplo, agrupar os dados do censo por Categoria e número de AnosEstudo.
- **summarise**: geralmente utilizado após o group_by. Calcula valores por grupo. Por exemplo, tirar a média ou mediana do Salario, depois de agrupar por categoria e anos de estudo.

- O dplyr vem também com o *pipe operator* `%>%` do pacote magrittr. Basicamente, o operador faz com que você possa escrever `x %>% f()` ao invés de `f(x)`.
- Na prática, isso tem uma grande utilidade: você vai poder escrever o código de manipulação dos dados da mesma forma que você pensa nas atividades.

- Exemplo:
 - Quantos fumantes por sexo há na secretaria?

```
fumantes.por.sexo <- censo %>%
```

```
  filter(Fuma == 1) %>%
```

```
  group_by(Sexo) %>%
```

```
  summarise(quantidade=n())
```

```
fumantes.por.sexo
```

```
# A tibble: 4 x 2
```

```
  Sexo quantidade
```

```
<chr>    <int>
```

```
1  F      3554
```

```
2 Fem.    186
```

```
3  M     5031
```

```
4 Masc.   878
```

A tibble: 4 x 2

Sexo quantidade

<chr> <int>

1	F	3554
2	Fem.	186
3	M	5031
4	Masc.	878

- Como padronizar a coluna Sexo?
- Substituindo valores errados por valores corretos

```
# substituir Fem. por F, e Masc. por M
```

```
# seleciona o indice das linhas que contem "Fem." e "Masc."
```

```
fem.errados <- censo$Sexo == "Fem."
```

```
masc.errados <- censo$Sexo == "Masc."
```

```
# substitui os valores pelos valores corretos, aplicando o filtro
```

```
censo[fem.errados,]$Sexo <- "F"
```

```
censo[masc.errados,]$Sexo <- "M"
```

- Agora sim:

```
fumantes.por.sexo <- censo %>%  
  filter(Fuma == 1) %>%  
  group_by(Sexo) %>%  
  summarise(quantidade=n())
```

```
fumantes.por.sexo
```

```
# A tibble: 2 x 2
```

```
  Sexo quantidade
```

```
  <chr>    <int>
```

```
1    F      3740
```

```
2    M      5909
```


- Agora agrupando por sexo,fuma:

```
quantidade.por.sexo.fumante <- censo %>%
```

```
  group_by(Sexo, Fuma) %>%
```

```
  summarise(quantidade=n())
```

```
quantidade.por.sexo.fumante
```

```
Sexo Fuma quantidade
```

```
<chr> <int>    <int>
```

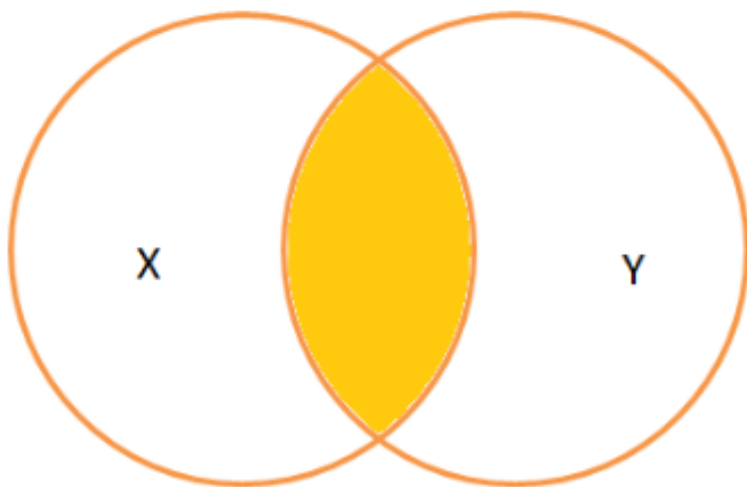
```
1    F    0    5740
```

```
2    F    1    3740
```

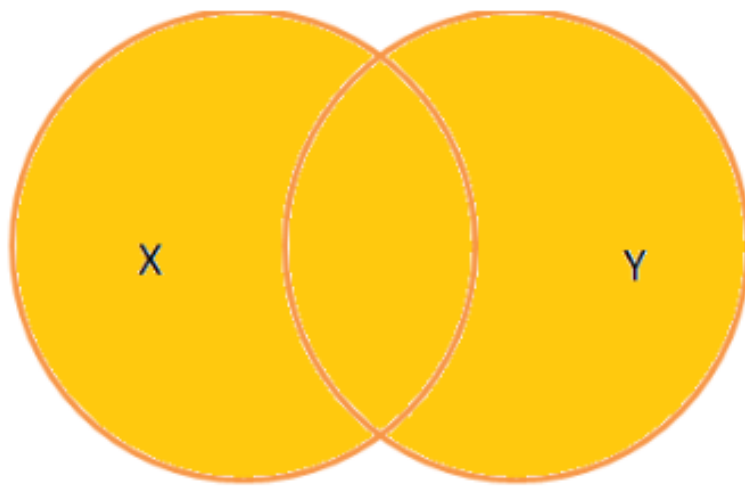
```
3    M    0    4611
```

```
4    M    1    5909
```

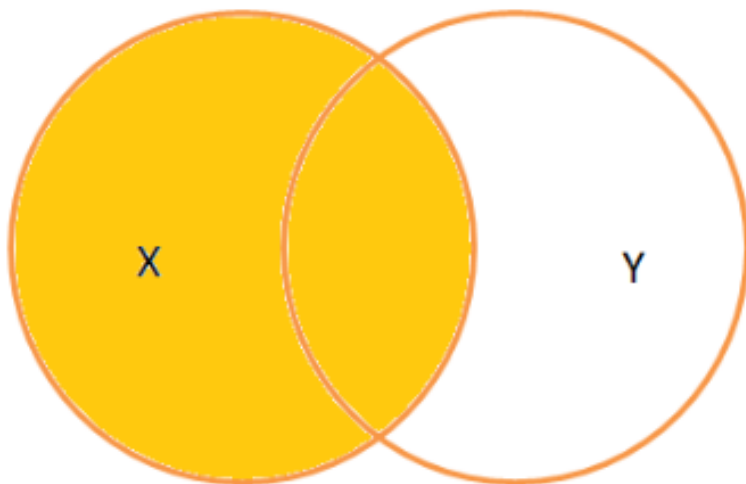
- A função `merge()` serve para combinar `data.frames`. A função tenta identificar quais são as colunas identificadores em comum entre dois `data.frames` para realizar a combinação. Para quem conhece SQL, a função `merge()` é equivalente ao `join`.
- Alguns argumentos da função são:
- **x**: um `data.frame`
- **y**: um `data.frame`
- **by**: a coluna em comum nos `data.frames` pela qual será feito o `merge`.
- **all**, **all.x**, **all.y**: especifica o tipo do `merge`. O default é `FALSE` e é equivalente ao “natural join” do SQL; “all” é equivalente ao “outer join”; “all.x” é equivalente ao “left outer join” e “all.y” é equivalente ao “right outer join”.



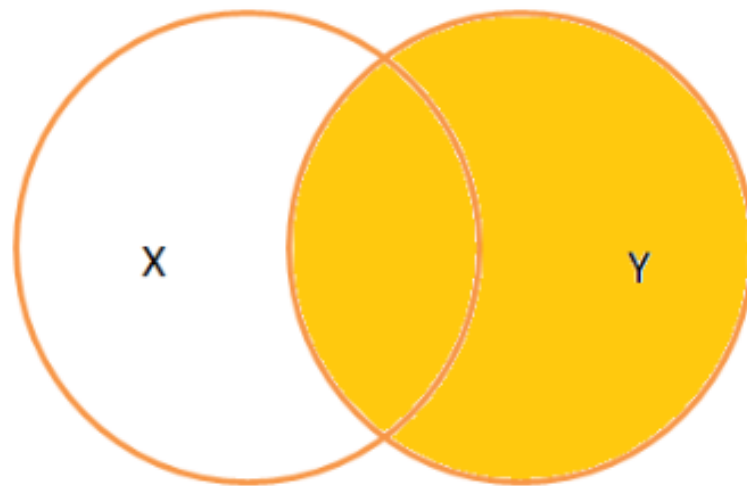
all = FALSE



all = TRUE



all.x = TRUE



all.y = TRUE

- Juntar o data frame de censo com o data frame de cadastro de CPF e procurar por divergências.
- # Função merge
- `censo.nome.cpf <- censo[, c("CPF", "Nome")]`
- `head(censo.nome.cpf)`
- `cadastro <- read.csv2("data/CadastroCPF.csv")`
- `head(cadastro)`
- `m <- merge(cadastro, censo.nome.cpf, by = "CPF", all.y=TRUE)`
- `registros.problema.identificacao <- m %>% filter(is.na(Nome.x))`
- `View(registros.problema.identificacao)`

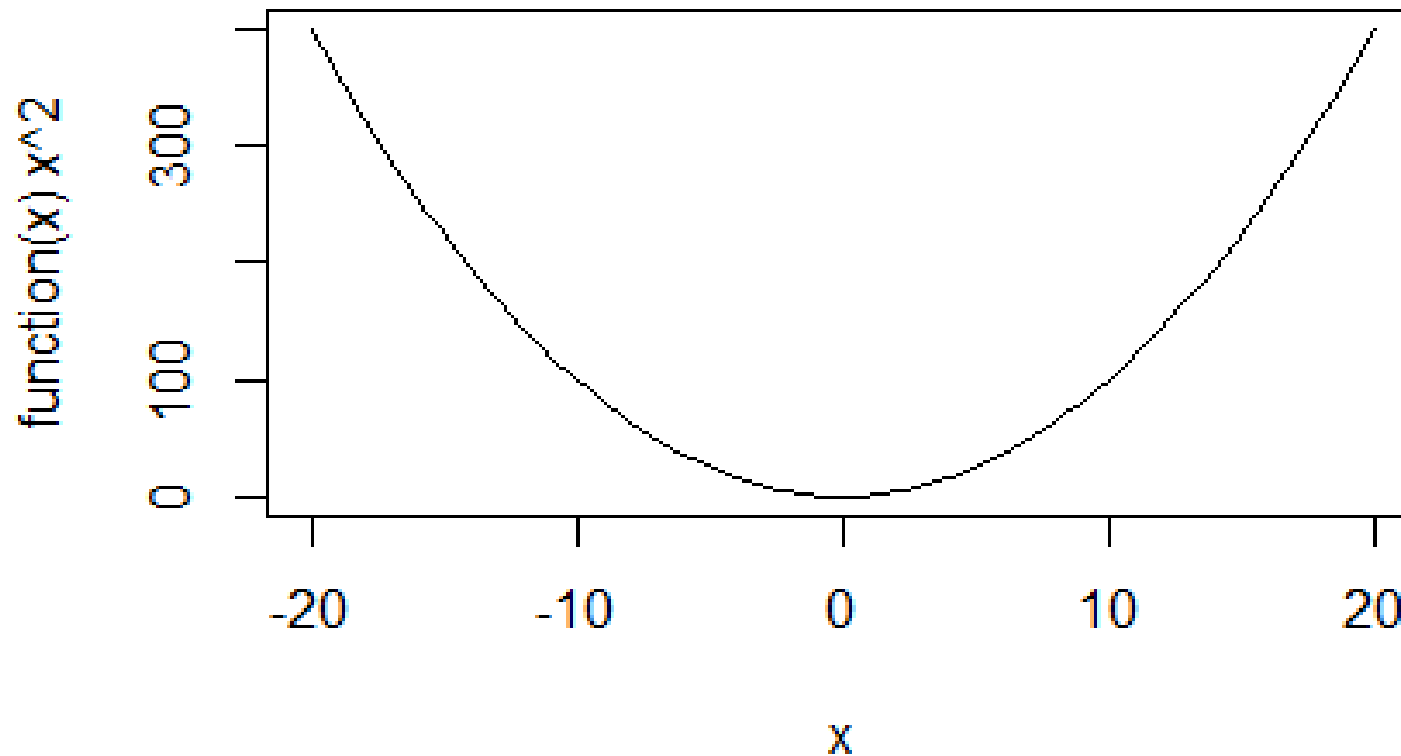
- Base R
 - Pacote “nativo” do R
- Ggplot2
 - Pacote externo, mais flexível e elegante

A função `plot` é uma função genérica, e possui vários métodos diferentes dependendo do tipo de objeto. Veja `methods(plot)`.
Exemplo: `lm`

- `x <- rnorm(100)`
- `y <- 10 + 2*x + rnorm(100)`
- `m <- lm(y~x)`
- `plot(m)`

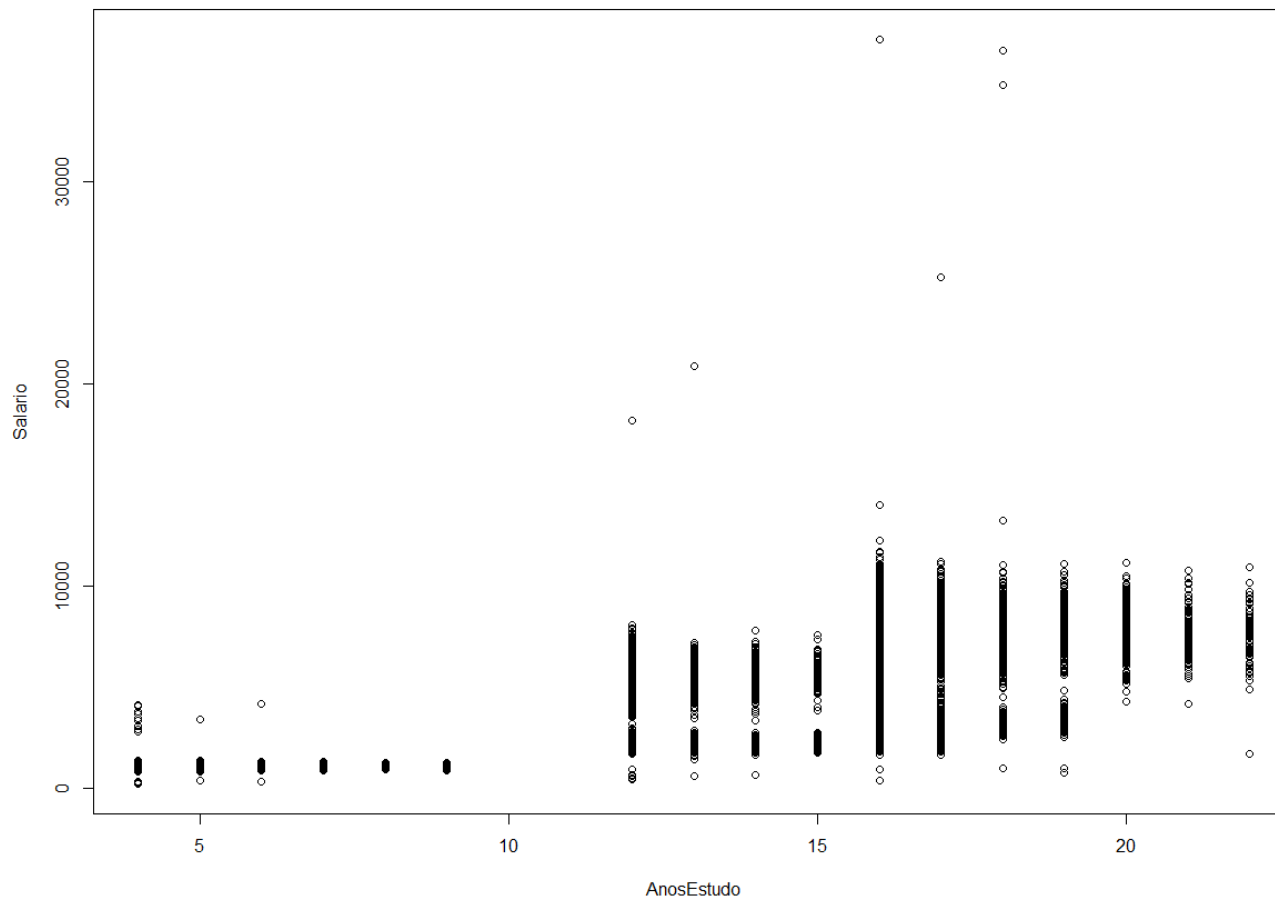
Plot de Função

plot(function(x) x^2 , from = -20, to = 10)



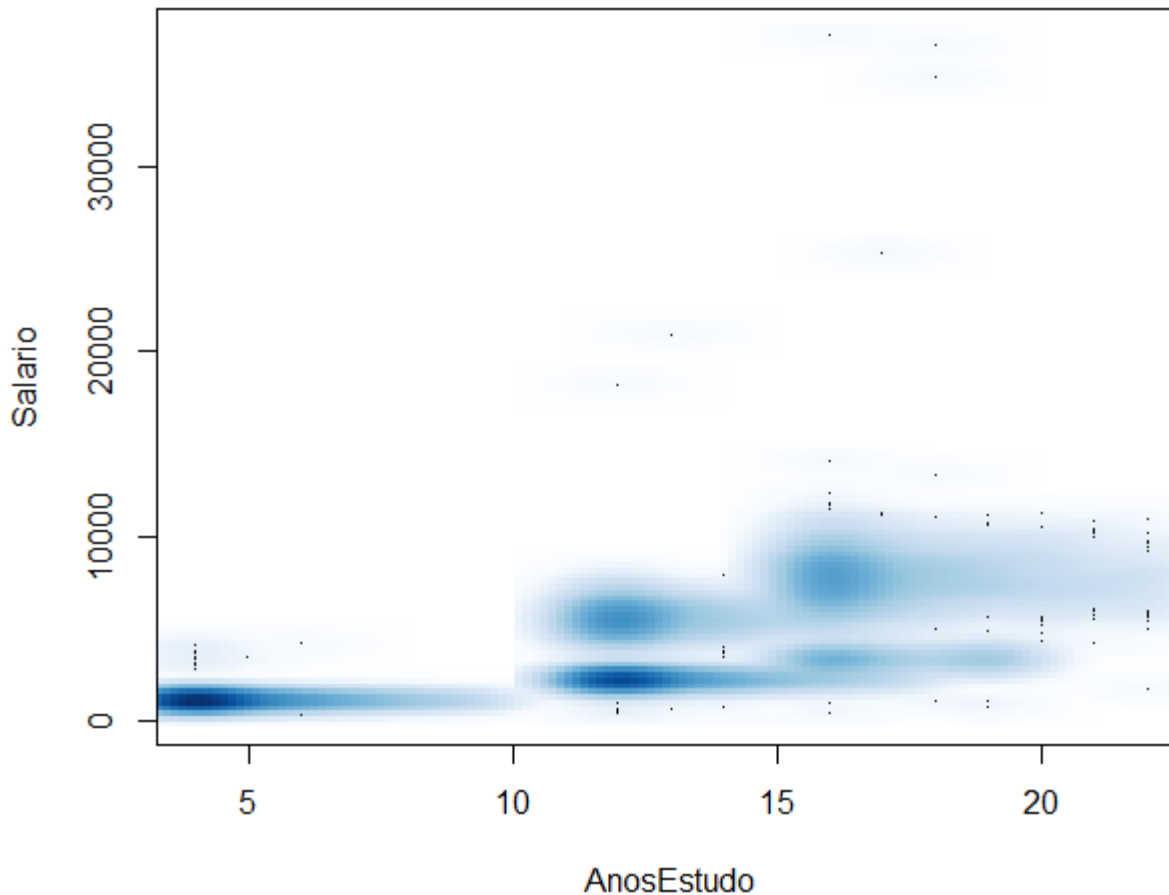
Scatter Plot

with(censo, plot(AnosEstudo, Salario))



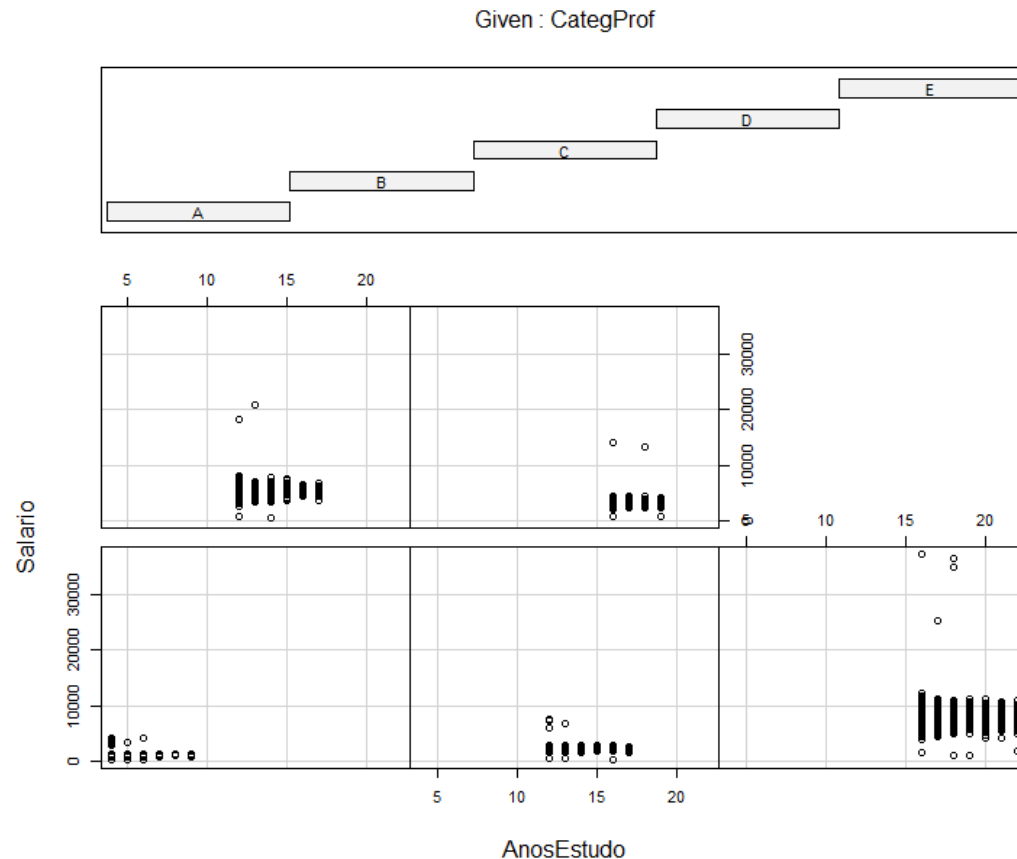
Smooth Scatter Plot

with(censo, smoothScatter(AnosEstudo, Salario))



CoPlot

with(censo, coplot(Salario~AnosEstudo | CategProf))



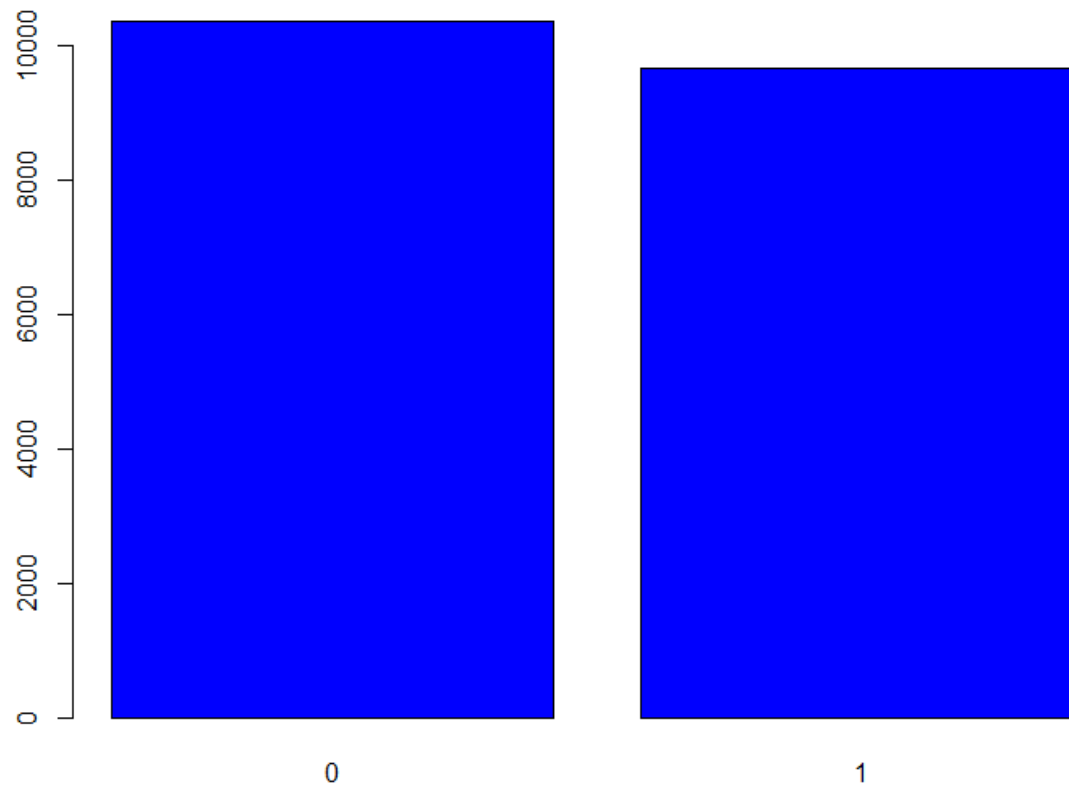
Personalizando parâmetros

(digite **?par** para ver os parâmetros)

```
with(censo, plot(AnosEstudo, Salario, col="darkblue",  
                main = "Salario por Anos de Estudo",  
                xlab = "Anos de Estudo",  
                ylab = "Salario",  
                pch = 3,  
                cex = 2))
```

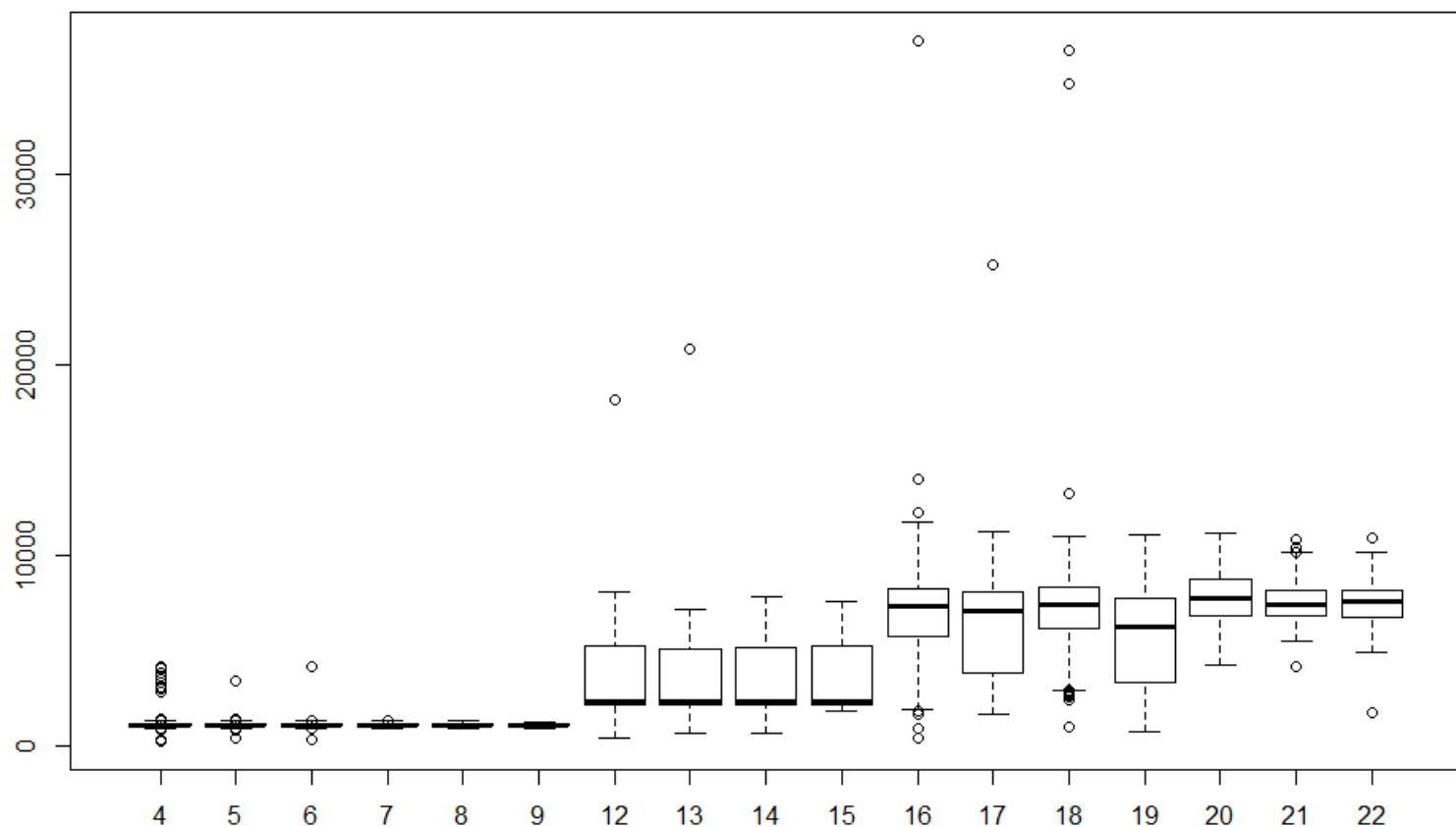
Barplot

```
barplot(table(censo$Fuma), col="blue")
```



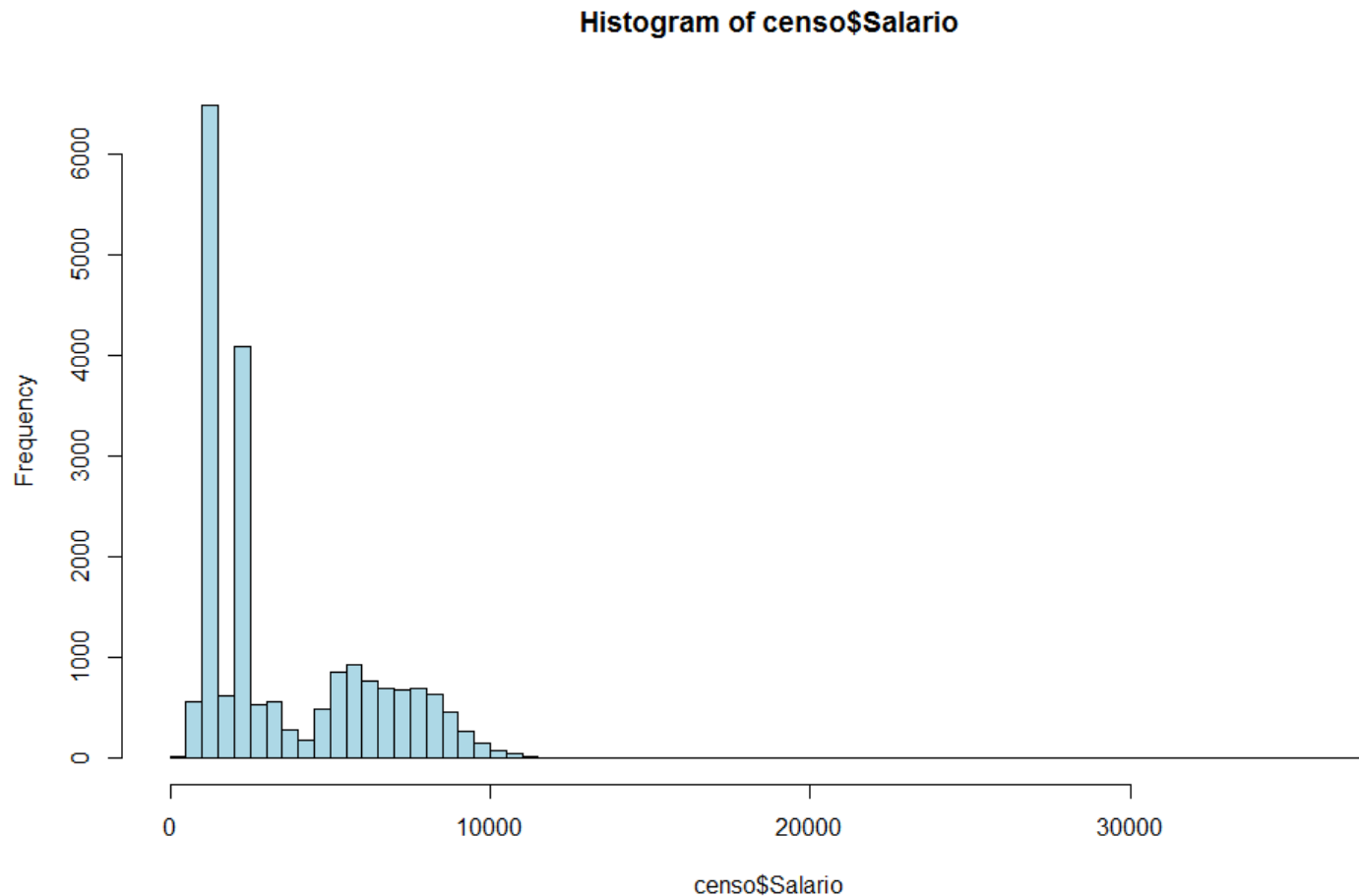
Boxplot

```
boxplot(Salario~AnosEstudo, data=censo)
```



Histograma

```
hist(censo$Salario, breaks = 100, col="lightblue")
```



Qualidade de Dados com R

- Estatísticas básicas
 - Quantidade de valores únicos: função **unique()**
 - Quantidade de valores nulos: função **is.na()**
 - Valores máximos e mínimos: **summary()**, **max()**, **min()**
 - Formatos: **is.numeric()**, **is.character()**, **is.logical()**, etc.
 - Frequências de valores:


```
dataframe %>%
  group_by(coluna) %>%
  summarise(frequencia=n()) %>%
  arrange(desc(frequencia))
```
- Processos de melhoria
 - Função **merge()** para identificar CPFs e Nomes com problemas
- Não se esqueça de sempre salvar suas alterações no seu repositório

- Não se esqueça de sempre salvar suas alterações no seu repositório
 - git status
 - git add **meuarquivo.R**
 - git commit -m “mensagem”
 - git push

Mãos à obra!