Dicas e bons costumes no R

Luben Miguel

Universidade Federal de São Carlos

15/07/2022

Objetos e suas classes

- O R te permite salvar valores dentro de um *objeto*
- *Objeto*: nome que guarda um valor.
- Memória reservada para guardar valores

```
# objeto a que armazena o valor 1
a <- 1

# avaliando o objeto
a
```

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

- Algumas regrinhas para nomes:
    - Começar com uma letra
    - Pode conter números mas não começar
    - Pode usar `.` e `_` para separar palavras

```r
# Permitido
x <- 1
x1 <- 2
obj <- 3
meu_obj <- 4
meu.obj <- 5

# Não permitido
1x <- 1
_objeto <- 2
meu-objeto <- 3
```

- O R também diferencia letras maiusculas de minúsculas

```r
b <- 2
B <- 3
b
```

```
## [1] 2
```

```r
B
```

```
## [1] 3
```

Classes e estruturas

- Classes: dita qual o tipo de informação guardada no objeto
- Principais classes mais básicas:
    - `numeric`
    - `character`
    - `logical`

```
x <- 1
class(x)
```

```
## [1] "numeric"
```

```
y <- "a"
class(y)
```

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

```
z <- TRUE
class(z)
```

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

Classes e estruturas

- Outras classes também usadas:
  - `factor`
  - `integer`
  - `complex`

```
x <- factor(c("tipo1", "tipo2"))
x
```

```
## [1] tipo1 tipo2
## Levels: tipo1 tipo2
```

```
class(x)
```

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

```
y <- 12L
class(y)
```

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

Classes e estruturas

- Outros tipos de objetos/classes mais complexas:
    - `vetores`
    - `data frames`
    - `listas`
    - `matrizes`

```
a <- c(1, 2, 3)
class(a)
```

```
## [1] "numeric"
```

```
class(mtcars)
```

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

```
b <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
class(b)
```

```
## [1] "matrix" "array"
```

Vetores

Vetores

- Conjunto indexados de valores
- Cada coluna de um data frame é um vetor
- Operador `c()`

```
numeros <- c(1, 3, 7, -2)

numeros
```

```
## [1]  1  3  7 -2
```

```
letras <- c("a", "b", "c")

letras
```

```
## [1] "a" "b" "c"
```

Vetores

- Diversas maneiras de criar vetores.
- Usando o operador `:`

```
# sequencia de 1 a 10
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

- Armazenando espaço para um vetor de certo tamanho

```
# criando espaco para um vetor numerico de tamanho 10
a <- numeric(10)
a
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

- Criar uma "grade" de valores

```
grid <- seq(1, 3, 0.5)
grid
```

Vetores

- Cada valor dentro de um vetor tem uma posição
- Indexação: acessar as posições de interesse

```
vetor <- c("a", "b", "c", "d")

# primeiro elemento
vetor[1]
```

```
## [1] "a"
```

```
# segundo elemento
vetor[2]
```

```
## [1] "b"
```

```
# fora de posicao
vetor[5]
```

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

Vetores

- Conjunto de indices entre `[]` (subsetting)

```
# primeiro, segundo e quarto elemento
vetor[c(1, 2, 4)]
```

```
## [1] "a" "b" "d"
```

- Vetores apenas armazenam um tipo de classe:

```
vetor1 <- c(1, 5, 3, -10)
class(vetor1)
```

```
## [1] "numeric"
```

```
vetor2 <- c("a", "b", "c")
class(vetor2)
```

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

Vetores

- Misturar duas classes: *coerção*

```
vetor <- c(1, 2, "a")
vetor
```

```
## [1] "1" "2" "a"
```

```
class(vetor)
```

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

- Operações aritméticas com vetor:

```
vetor <- c(0, 5, 20, -3)
vetor + 1
```

```
## [1]  1  6 21 -2
```

```
vetor - 1
```

```
## [1] -1  4 19 -4
```

Vetores

- Operações aritméticas com vetor:

```
vetor / 2
```

```
## [1]  0.0  2.5 10.0 -1.5
```

```
vetor * 10
```

```
## [1]   0  50 200 -30
```

```
# soma de dois vetores
vetor1 <- c(1, 2, 3)
vetor2 <- c(10, 20, 30)

vetor1  + vetor2
```

```
## [1] 11 22 33
```

Vetores

- Operações aritméticas com vetor

```
# soma de dois vetores de tamanho diferente (reciclagem)
vetor1 <- c(1, 2)
vetor2 <- c(10, 20, 30, 40)

vetor1  + vetor2
```

```
## [1] 11 22 31 42
```

```
# multiplicacao indice por indice e soma
vetor1 <- c(1, 2, 3)
vetor2 <- c(10, 20, 30)
sum(vetor1*vetor2)
```

```
## [1] 140
```

```
# multiplicação matricial
vetor1 %*% vetor2
```

```
##      [,1]
## [1,]  140
```

Vetores

- Testes lógicos em vetores

```
valores <- c(1, 2, 5, 8, 10, 12, -1)

# vetor de booleana
valores > 3
```

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

```
# selecionando valores maiores que 3 por subsetting
valores[valores > 3]
```

```
## [1]  5  8 10 12
```

Vetores

- Grande parte dos operadores e funções do R são vetorizados
- Utilização da reciclagem

```
set.seed(125)
# gerando uniformes com diferentes maximos e minimos aproveitando da reciclagem
runif(6, min = c(-1 ,1), max = c(1, 2))
```

```
## [1]  0.6493488  1.1168510 -0.4004389  1.3565607  0.9303900  1.9675605
```

- Podemos também aumentar vetores usando o operador `c()`, ainda que subótimo (não é boa prática)

```
a <- c(10, 20, 30)
a <- c(a, 40)
a
```

```
## [1] 10 20 30 40
```

Valores especiais

Valores especiais

- `NA`: Ausência de informação, *"missing"*
- `NaN`: Indefinições matematicas
- `Inf`: Número muito grande
- `NULL`: ausência de objeto
- Para checar se um objeto é algum dos valores especiais podemos usar funções do tipo `is.na()`, `is.nan()`, `is.null()` e assim por diante

```
nao_sou_um_numero <- NaN
objeto_nulo <- NULL

is.nan(nao_sou_um_numero)
```

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

```
is.null(objeto_nulo)
```

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

Listas

Listas

- Parecida com vetores
- Diferença: pode misturar diferentes classes de objetos dentro dela.

```
list(1,
     "a",
     TRUE,
     c(1, 2),
     c("1", "2"))
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1 2
##
## [[5]]
## [1] "1" "2"
```

Listas

- Subsetting diferente de vetores
- Cada elemento de uma lista também é uma lista

```
lista <- list(1, "a", TRUE)
lista[1]
```

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

```
class(lista[1])
```

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

Listas

- Usar `[[]]` para acessar o elemento de cada posição

```
lista[[1]]
```

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

```
class(lista[[1]])
```

```
## [1] "numeric"
```

Listas

- Podemos dar nome para cada posição

```
cliente <- list(cliente = "Pedro Silva", idade = 24, estado_civil = NA)
cliente
```

```
## $cliente
## [1] "Pedro Silva"
##
## $idade
## [1] 24
##
## $estado_civil
## [1] NA
```

- Quando com nomes, podemos acessar cada posição usando o operador $

```
cliente$idade
```

```
## [1] 24
```

- Todo data.frame é uma lista

Listas

- Estrutura de dados muito flexivel
- Podemos aumentar o tamanho de listas e adicionar novos elementos, desde que indexado corretamente

```
cliente$salario = 4000
cliente
```

```
## $cliente
## [1] "Pedro Silva"
##
## $idade
## [1] 24
##
## $estado_civil
## [1] NA
##
## $salario
## [1] 4000
```

Listas

- Podemos também usar a função `append()` do R

```
append(lista, "d")
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] "d"
```

Data Frames

Data Frames

- Tudo que vale para uma lista vale para um Data Frame
- Podemos usar o operador $ para selecionar uma coluna dos dados (vetor)

```
mtcars$mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

Data Frames

- Se selecionar por subsetting, o elemento também será um data.frame

```
mtcars[1]
```

```
##                    mpg
## Mazda RX4         21.0
## Mazda RX4 Wag     21.0
## Datsun 710        22.8
## Hornet 4 Drive    21.4
## Hornet Sportabout 18.7
## Valiant           18.1
## Duster 360        14.3
## Merc 240D         24.4
## Merc 230          22.8
## Merc 280          19.2
## Merc 280C         17.8
## Merc 450SE        16.4
## Merc 450SL        17.3
## Merc 450SLC       15.2
## Cadillac Fleetwood  10.4
## Lincoln Continental 10.4
## Chrysler Imperial   14.7
## Fiat 128          32.4
## Honda Civic       30.4
```

Data Frames

- Tipo específico de lista:
  - Todos os seus elementos (colunas) precisam ter o mesmo comprimento (número de linhas).
  - Todos os seus elementos (colunas) precisam ser nomeados
  - Data frames têm 2 dimensões (linhas e colunas)

```
dados_cliente <- list(
  cliente = c("Pedro Silva", "Vitor Pereira", "Carla Souza"),
  idade = c(24, 28, 21),
  estado_civil = c(NA, "Solteiro", "Casada")
)

as.data.frame(dados_cliente)
```

```
##         cliente idade estado_civil
## 1   Pedro Silva    24         <NA>
## 2 Vitor Pereira    28     Solteiro
## 3   Carla Souza    21       Casada
```

Data Frames

- Terceira propriedade: Data frames retangulares

```
dim(mtcars)
```

```
## [1] 32 11
```

- Usar dois indices para realizar subsetting

```
# observacao 2 e variavel disp
mtcars[2, 3]
```

```
## [1] 160
```

Data Frames

- Usar dois indices para realizar subsetting

```
# todas as linhas da coluna 1
mtcars[, 1]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

```
# todas as colunas da linha 1
mtcars[1, ]
```

```
##           mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4  21   6  160 110  3.9 2.62 16.46  0  1    4    4
```

Data Frames

- Subsetting e operadores lógicos semelhante ao de vetores

```
# selecionando as linhas 1,2 e 4
mtcars[c(1, 2, 4), ]
```

```
##                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
## Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
```

```
# vetor logico indicando as linhas que o numero de cilindros eh 4
mtcars$cyl == 4
```

```
##  [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
## [25] FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE
```

Data Frames

- Subsetting e operadores lógicos semelhante ao de vetores

```
mtcars[mtcars$cyl == 4, ]
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Bons costumes e algumas correções

Importação dos dados

- Maus costumes:
  - Usar setwd para setar diretório da lista/dos dados
  - Usar path específico dos dados

```
# maus costumes
setwd("/home/kuben/estatistica_UFSCAR/Doutorado/monitoria_mineracao/slides/slides_bom_R")

dados <- fread("/home/kuben/estatistica_UFSCAR/Doutorado/monitoria_mineracao/slides/slides_bom_R/w
```

- Bons costumes:
  - Ler dados diretamente do site
  - Criar rproj, nao precisando de path muito pessoal

```
# bons costumes
dados <- read.delim("http://www.rizbicki.ufscar.br/dados/worldDevelopmentIndicator
s.csv",header = T, sep=",")

dados <- readr::read_csv("worldDevelopmentIndicators.csv")
# pode usar fread com esse path tambem
```

Transformação dos dados

- Maus costumes:
  - Criar diversos objetos para manipulação dos dados
  - Mais de 1 data.frame

```
# depois de importar os dados
y <- dados$LifeExpectancy

x <- (dados$GDPercapita - min(dados$GDPercapita))/(max(dados$GDPercapita)-
min(dados$GDPercapita))

df <- data.frame(y, x)
```

- Bons costumes:
  - Transformar e utilizar os dados já importados
  - Fazer todas as transformações no próprio data.frame

Transformação dos dados

```
# usando tidyverse
dados <- readr::read_csv("worldDevelopmentIndicators.csv") |> select(-CountryName) |>
rename(y = LifeExpectancy, x = GDPercapita) |>
mutate(x = (x-min(x))/(max(x)-min(x)))

dados |>
glimpse()
```

```
## Rows: 211
## Columns: 2
## $ y <dbl> 60.50912, 51.46400, 77.35046, 69.94970, 76.95788, 76.01268, 74.43722…
## $ x <dbl> 0.004210438, 0.050492852, 0.036183948, 0.071037163, 0.399979495, 0.1…
```

```
# sem tidyverse
dados <- data.table::fread("worldDevelopmentIndicators.csv", header = TRUE)
dados <- dados[, -1]
colnames(dados) <- c("y", "x")
dados$x <- (dados$x-min(dados$x))/(max(dados$x)-min(dados$x))
head(dados, 2)
```

```
##           y           x
## 1: 60.50912 0.004210438
## 2: 51.46400 0.050492852
```

Criar objetos para armazenamento e laço

- Maus costumes:
  - Usar `c()`
  - Usar `NULL`
  - Usar `rep(NA, n)`
  - Hardcoding nos laços e variaveis

```
for(j in 1:211){
  ...
}

preditos <- matrix(0, nrow=211, ncol=30)
formulas <-  NULL
risco <- rep(NA, 30)
```

- Bons costumes:
  - Usar `numeric(n)`
  - Usar `list()`
  - Utilizar indices genéricos

Laço do exercicio 2

```
p <- 30
str <- "y ~ "
formulas <- list()
risco <- numeric(p)

for(j in 1:p){
  erro_temp <- numeric(nrow(dados))
  str <- paste0(str, "sin(2*", j, "*pi*x) + cos(2*", j, "*pi*x)")
  formulas[[j]] <- str
    for(i in 1:nrow(dados)){
    mod <- lm(formulas[[j]], data = dados[-i, ])
    erro_temp[i] <- abs(dados$y[i] - predict(mod, dados[i, ]))
  }
  risco[j] <- mean(erro_temp)
  str <- paste0(str, " + ")
}

results <- data.frame(p = 1:p, MAE = risco)
glimpse(results)
```

```
## Rows: 30
## Columns: 2
## $ p   <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,…
## $ MAE <dbl> 5.449510e+00, 5.096368e+00, 4.734910e+00, 4.705431e+00, 4.568074e+…
```

Gerar gráficos de curvas

- Maus costumes:
  - Usar varios data.frames para gerar um gráfico
  - Não ajustar limites
  - Usar `geom_line()` nas bases diferentes

```
...
ggplot() +
geom_point(mapping= aes(x=xnorm,y=Y),data=dados_norm)+
geom_line(mapping = aes(x=grade,
                        y=pred1, colour = "p = 1"), data= dados_predito)
+ ...
```

- Bons costumes:
  - Utilizar uma única base para gerar o gráfico
  - Usar `coord_cartesian()`
  - Funções como `geom_smooth()` e `stat_function()` na mesma base

Gerar gráfico de curvas

```
dados |>
  ggplot(aes(x = x, y = y))+
  geom_point(alpha = 0.4) +
  geom_smooth(aes(color = "p=1"),method = 'lm', se = FALSE, formula = formulas[[1]]) +
  geom_smooth(aes(color = "p=5"),method = 'lm',se = FALSE, formula = formulas[[5]]) +
  geom_smooth(aes(color = "p=30"),method = 'lm', se = FALSE, formula = formulas[[30]]) +
  coord_cartesian(ylim = c(min(dados$y) - 5, max(dados$y) + 5))+
  scale_color_brewer(palette = "Set1", breaks = c("p=1", "p=5", "p=30")) +
  labs(color = 'Valor de P', x = 'PIB per capita normalizado',y = 'Expectativa de Vida',
       title = 'Curvas ajustadas para p = 1, 5 e 30')
```

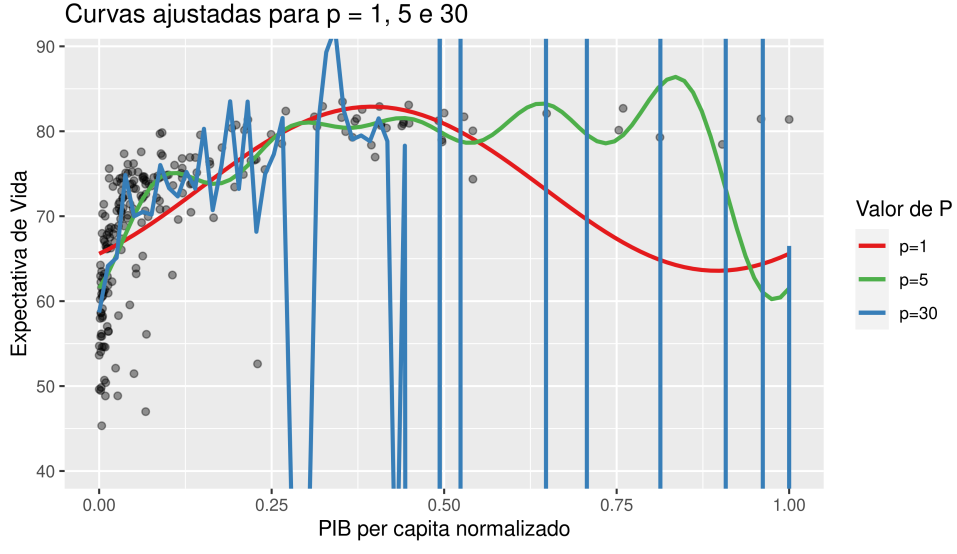


Gráfico de valores preditos versus observados

- Maus costumes:
  - Fazer gráficos separados
  - Usar a mesma escala no eixo de preditos
  - Hardcoding na criação da base para plotagem
- Bons costumes:
  - Fazer gráficos em facets
  - Usar diretamente a base de dados inicial
  - Organizar as informações em um mesmo data frame

Gráfico de valores preditos versus observados

- Criando a base que sera usada para gerar o grafico

```
# funcao para juntar tudo
idxs <- c(1, 5, 30)
for(id in idxs){
  pred <- numeric(nrow(dados))
  for(i in 1:nrow(dados)){
    mod <- lm(formulas[[id]], data = dados[-i, ])
    pred[i] <- predict(mod, dados[i, ])
  }
  var_name <- paste0("p = ", id)
  dados <- dados |>
    mutate({{var_name}} := pred)
  # usando rbase : dados[var_name] <- pred
}
dados |> glimpse()
```

```
## Rows: 211
## Columns: 5
## $ y        <dbl> 60.50912, 51.46400, 77.35046, 69.94970, 76.95788, 76.01268, 7…
## $ x        <dbl> 0.004210438, 0.050492852, 0.036183948, 0.071037163, 0.3999794…
## $ `p = 1`  <dbl> 65.78438, 67.90199, 67.04082, 68.86715, 83.07883, 71.00311, 6…
## $ `p = 5`  <dbl> 62.05453, 70.37214, 67.36707, 73.12910, 81.28311, 75.08048, 6…
## $ `p = 30` <dbl> 5.913882e+01, 7.054834e+01, 7.340779e+01, 6.908822e+01, 8.387…
```

Gráfico de valores preditos versus observados

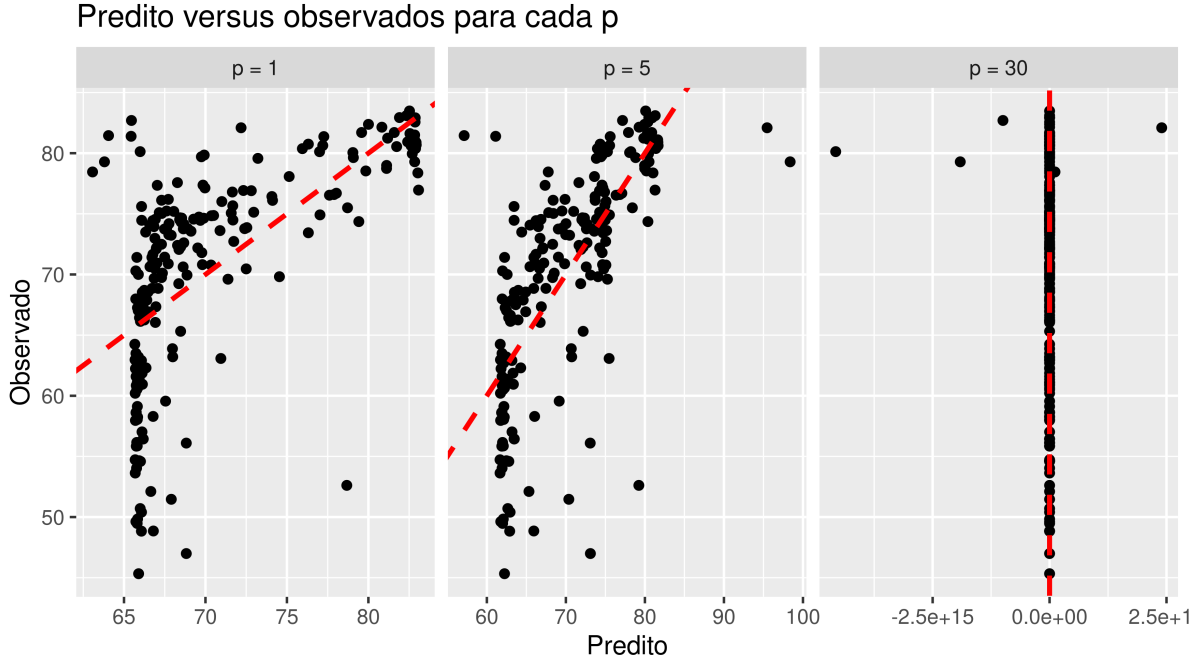
- "Derretendo" os dados e gerando a tabela para plotar em facets

```
dados_plot <- dados |> pivot_longer(`p = 1`:`p = 30`, names_to = "p",
                                    values_to = "pred")
dados_plot |> glimpse()
```

```
## Rows: 633
## Columns: 4
## $ y    <dbl> 60.50912, 60.50912, 60.50912, 51.46400, 51.46400, 51.46400, 77.35…
## $ x    <dbl> 0.004210438, 0.004210438, 0.004210438, 0.050492852, 0.050492852, …
## $ p    <chr> "p = 1", "p = 5", "p = 30", "p = 1", "p = 5", "p = 30", "p = 1", …
## $ pred <dbl> 65.78438, 62.05453, 59.13882, 67.90199, 70.37214, 70.54834, 67.04…
```

Gráfico de valores preditos versus observados

```
dados_plot |>
  mutate(p = factor(p, levels = c("p = 1", "p = 5", "p = 30"))) |>
  ggplot(aes(x = pred, y = y))+
  geom_point()+
  geom_abline(intercept =0 , slope = 1, size=1, linetype = "dashed", colour = "red")+
  facet_wrap(vars(p), ncol = 3, scales = "free_x")+
  labs(title = "Predito versus observados para cada p", x = "Predito", y = "Observado")
```



Muito obrigado!

Referências

- Livro da Curso-R, capítulo 3
- tutorialspoint - Learn R Programming