Universidade Federal de Uberlândia

Lista do Módulo 1

Levy Gabriel da Silva Galvão

Uberlândia
2021

Módulo 1

Parte 1

Exercício 1

> Crie uma função em R para fazer a conversão entre unidades de temperatura (De Kelvin para Celcius). Forneça um exemplo de uso.

```
kelvin2celsius <- function(temp_kelvin)
{
  temp_celsius <- temp_kelvin - 273.15
  return(temp_celsius)
}
```

Um exemplo para este caso é encontrar a temperatura padrão das condições normais de temperatura e pressão (CNTP) de 273.15 K em Celsius, resultando em:

```
cntp_kelvin <- 273.15
cntp_celsius <- kelvin2celsius(cntp_kelvin)
print(cntp_celsius)
```

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

Exercício 2

> Crie um loop que imprima os valores ímpares no intervalo de 1 a 100.

```
num <- 0
for(num in 1:100)
```

```
{
  if(num %% 2) # if number is odd
  {
    print(num)
  }
}
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
## [1] 11
## [1] 13
## [1] 15
## [1] 17
## [1] 19
## [1] 21
## [1] 23
## [1] 25
## [1] 27
## [1] 29
## [1] 31
## [1] 33
## [1] 35
## [1] 37
## [1] 39
## [1] 41
## [1] 43
## [1] 45
## [1] 47
## [1] 49
## [1] 51
## [1] 53
## [1] 55
## [1] 57
## [1] 59
## [1] 61
## [1] 63
## [1] 65
## [1] 67
## [1] 69
## [1] 71
## [1] 73
## [1] 75
## [1] 77
## [1] 79
## [1] 81
## [1] 83
## [1] 85
## [1] 87
## [1] 89
## [1] 91
## [1] 93
## [1] 95
## [1] 97
## [1] 99
```

Exercício 3

> Qual a diferença entre os operadores “&”, “&&”, “|” e “||”. Ilustre o uso destes operadores por meio de um programa exemplo. resposta

Os operadores simples “&” e “|” atuam realizando a operação elemento a elemento (*element wise*) entre seus operandos, enquanto que os operadores “&&” e “||” executam a operação em um único elemento e, se utilizado entre *arrays*, a operação será feita somente no primeiro elemento. Vide exemplo abaixo.

```
# test arrays
x <- c(FALSE, FALSE, TRUE, TRUE)
y <- c(FALSE, TRUE, FALSE, TRUE)
# element wise operation
print(x&y)
```

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

```
print(x|y)
```

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

```
# single element operation
print(x&&y)
```

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

```
print(x||y)
```

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

Pela distribuição dos *arrays*, permite-se traçar as tabelas verdades da operação AND por meio do operador “&” e da operação OR por meio do operador “|”, enquanto que como o esperado os operadores duplos atuam somente nos primeiros elementos.

Exercício 4

> Faça um programa que exemplifique o uso do operador “%in%”

O operador “%in%” verifica se o conteúdo do *array* como primeiro operando está contido no *array* como segundo operando. O retorno é um *array* binário que pode ser usado para indexar todo o conteúdo do primeiro operando que está contido no segundo operando. Vide os exemplos abaixo.

```
x <- c(0, 2, 4, 6)
y <- 1:10
```

```
# boolean array that indicates whether content of first operand is in the second operand
print(x%in%y)
```

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

```
# this result can be used to index all the content from the first operand that meets the requ
irements
print(x[x%in%y])
```

```
## [1] 2 4 6
```

Exercício 5

> Dado a string “O Sol é muito maior do que a Terra”, faça um programa que:

- Substitua a palavra “Terra” por “Lua”
- Conte o número de caracteres da string original e da string alterada;
- Ordene a string alterada
- Faça uma comparação usando o operador & entre a string alterada e original.
- Explique o resultado obtido.
- Utilize o operador == para identificar os caracteres iguais.

O primeiro passo é substituir o trecho “Terra” da *string* por “Lua”, resultando em (com o auxílio da biblioteca **stringr**):

```
library(stringr) # library to handle strings
str_ <- "O Sol é muito maior do que a Terra"
str_new <- str_replace(str_, "Terra", "Lua")
print(str_new)
```

```
## [1] "O Sol é muito maior do que a Lua"
```

Assim, o número de caracteres da *string* original será:

```
print(str_length(str_))
```

```
## [1] 34
```

E da *string* modificada:

```
print(str_length(str_new))
```

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

Assim, mostrando que, uma vez que a palavra inserida “Lua” np lugar de “Terra” por ter uma quantidade de caracteres menor, permitiu que a nova *string* modificada possua, também, menos caracteres.

O próximo passo é ordenar os caracteres da *string* modificada, resultando em:

```
str_splited = unlist(strsplit(str_new, "")) # split string to get characters
str_sorted <- str_sort(str_splited) # sorting in ascending order
str_merged <- str_c(str_sorted, collapse="")
print(str_merged)
```

```
## [1] "        aaadeéiilLmmooooOqrStuuu"
```

O passo seguinte é comparar a *string* alterada com a original com meio do operador “&” elemento a elemento.

```
print(str_merged & str_new)
```

```
## Error in str_merged & str_new: operações são possíveis somente para tipos numéricos, lógic
os ou complexos
```

A operação AND elemento a elemento não é definida para *strings* ou caracteres, por isso o erro persiste.

Porém a comparação de igualdade “==” existe e, por meio do vetor de caracteres, pode-se definir um vetor comparando quais caracteres são iguais.

```
print(unlist(strsplit(str_new, "")) == unlist(strsplit(str_, "")))
```

```
## Warning in unlist(strsplit(str_new, "")) == unlist(strsplit(str_, "")):
## comprimento do objeto maior não é múltiplo do comprimento do objeto menor
```

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

Apesar das *strings* diferirem no tamanho (a nova *string* é menor), a operação de igualdade ainda é feita, porém com um aviso. O vetor booleano resultante possuirá o tamanho da maior *string* e os últimos elementos que não existem na *string* menor serão preenchidos com FALSE.

Exercício 6

> Refaça todos os exemplos apresentados neste módulo Parte 1.

```
meuTexto <- "Bom dia!" # com comentarios para teste
print(meuTexto)
```

```
## [1] "Bom dia!"
```

```
class(meuTexto)
```

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

```r
v <- 100
class(v)
```

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

```r
v <- "100"
class(v)
```

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

```r
v <- F
class(v)
```

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

```r
v <- 100L
class(v)
```

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

```r
v <- 6 + 8i
class(v)
```

```
## [1] "complex"
```

```r
minhaVariavel <- 1
m_var <- 1
.mvar <- 1 # hiden object
```

```r
x <- 100 # 100 stored in x
print(x)
```

```
## [1] 100
```

```r
y <- 200 + 300 # 100 stored in y
print(y)
```

```
## [1] 500
```

```r
z <- x + y
print(z)
```

```
## [1] 600
```

```
100 -> x # works fine as in x <- 100
v_cor <- c('azul', 'amarelo', 'rosa')
print(v_cor)
```

```
## [1] "azul"    "amarelo" "rosa"
```

```
v01 <- c(3,4,5)
print(v01)
```

```
## [1] 3 4 5
```

```
cat(v01) # print on screen or file
```

```
## 3 4 5
```

```
ls() # show variables
```

```
##  [1] "cntp_celsius"   "cntp_kelvin"    "kelvin2celsius" "m_var"
##  [5] "meuTexto"       "minhaVariavel"  "num"            "str_"
##  [9] "str_merged"     "str_new"        "str_sorted"     "str_splited"
## [13] "v"              "v_cor"          "v01"            "x"
## [17] "y"              "z"
```

```
ls(pattern = "v") # variables with char v
```

```
## [1] "cntp_kelvin"    "kelvin2celsius" "m_var"          "minhaVariavel"
## [5] "v"              "v_cor"          "v01"
```

```
meuTexto <- "texto"
rm(meuTexto) # remove variable from workspace
print(meuTexto)
```

```
## Error in print(meuTexto): objeto 'meuTexto' não encontrado
```

```
a1 <- c(9, 0, -4)
a2 <- c(8, 2, -1)

print(a1 + a2) # sum
```

```
## [1] 17  2 -5
```

```r
s1 <<- c(1, 2, 3)
c(4, 5, 2) ->> s2

print(s1-s2) # subtraction
```

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

```r
m1 <- c(2, 5, 8)
m2 <- c(3, 6, 9)
print(m1*m2) # multiplication
```

```
## [1]  6 30 72
```

```r
d1 <- c(6,8,9)
d2 <- c(2,4,3)
print(d1/d2) # division
```

```
## [1] 3 2 3
```

```r
r1 <- c(10, 13, 19)
r2 <- c(3, 4, 5)
print(r1%%r2) # module operation
```

```
## [1] 1 1 4
```

```r
r1 <- c(10, 13, 19)
r2 <- c(3, 4, 5)
print(r1 %/% r2) # division quocient
```

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

```r
e1 <- c(2, 3, 4)
e2 <- c(3, 2, 2)
print(e1^e2) # exponenciation
```

```
## [1]  8  9 16
```

```r
g1 <- c(6,8,9)
g2 <- c(9,2,6)
print(g1>g2) # greather then
```

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

```r
g1 <- c(6, 8, 9)
g2 <- c(9,2, 6)
print(g1<g2) # less then
```

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

```r
e1 <- c(3, 6, 9)
e2 <- c(2, 7, 9)
print(e1==e2) # equal
```

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

```r
e1 <- c(3, 6, 9)
e2 <- c(2, 7, 9)
print(e1 >= e2) # greather then and equal
```

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

```r
e1 <- c(3, 6, 9)
e2 <- c(2, 7, 9)
print(e1 <= e2) # less then and equal
```

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

```r
e1 <- c(3, 6, 9)
e2 <- c(2, 7, 9)

print(e1!=e2) # different
```

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

```r
x <- c(TRUE,FALSE,0,6)
y <- c(FALSE,TRUE,FALSE,TRUE)

print(!x) # element wise NOT
```

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

```r
print(x&y) # element wise AND
```

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

```r
print(x&&y) # AND in first element
```

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

```
print(x|y) # element wise OR
```

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

```
print(x||y) # OR in first element
```

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

```
v <- 1:8
print(v) # range
```

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

```
1:3==1:3 && 1:3==1:3
```

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

```
x <- 1:5
x[x<5]
```

```
## [1] 1 2 3 4
```

```
x[x<5 & x>2]
```

```
## [1] 3 4
```

```
x[x<5 && x>2]
```

```
## integer(0)
```

```
v1 <- 6
v2 <- 15
m <- 1:8

print(v1 %in% m) # is in
```

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

```
print(v2 %in% m) # is in
```

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

```
x <- 1:10
print(x)
```

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

```
y <- c(1,3:11)
print(y)
```

```
##  [1]  1  3  4  5  6  7  8  9 10 11
```

```
x[x%in%y]
```

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

```
y[y%in%x]
```

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

```
y[x%in%y] # wrong indexing
```

```
## [1]  1  4  5  6  7  8  9 10 11
```

```
m <- 200
n <- 100

if(m>n){ # if
  print("m é maior do que n")
}
```

```
## [1] "m é maior do que n"
```

```
m <- 200
n <- 100

if(m<n){ # if then else
  print("m é menor do que n")
} else {
  print("m é maior do que n")
}
```

```
## [1] "m é maior do que n"
```

```r
#switch

var <- "vegetal"

switch(var,
       fruit = "maçã",
       vegetal = "espinafre",
       carne = "frango")
```

```
## [1] "espinafre"
```

```r
# repeat
var <- c("Very Good!")
num <- 0

repeat{
  print(var)
  num <- num + 1
  if(num > 5){

    break # para o loop
  }
}
```

```
## [1] "Very Good!"
## [1] "Very Good!"
## [1] "Very Good!"
## [1] "Very Good!"
## [1] "Very Good!"
## [1] "Very Good!"
```

```r
#while

var <- c("OLA")
num <- 0

while(num<5){
  print(var)
  num <- num + 1
}
```

```
## [1] "OLA"
## [1] "OLA"
## [1] "OLA"
## [1] "OLA"
## [1] "OLA"
```

```r
#for

var <- c(1:5)
```

```
for(num in var)
{
  print("ola")

}
```

```
## [1] "ola"
## [1] "ola"
## [1] "ola"
## [1] "ola"
## [1] "ola"
```

```
#for

var <- c(1:5)

for(num in var)
{
  print("ola")

}
```

```
## [1] "ola"
## [1] "ola"
## [1] "ola"
## [1] "ola"
## [1] "ola"
```

```
# function

myfunction <- function(x,y)
{
  num <- x + y
  return(num)

}

a <- myfunction(100, 200)
print(a)
```

```
## [1] 300
```

```
myfunction <- function(x,y)
{
  num <- 300 + 400
  return(num)

}

a <- myfunction()
print(a)
```

```
## [1] 700
```

```
myfunction <- function(x=3,y=5)
{
  num <- x + y
  return(num)
}

a <- myfunction()
print(a)
```

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

```
print(sum(200, 100))
```

```
## [1] 300
```

```
print(abs(-1000))
```

```
## [1] 1000
```

```
print(max(100, 200))
```

```
## [1] 200
```

```
print(min(100, 200))
```

```
## [1] 100
```

```
print(sqrt(81))
```

```
## [1] 9
```

```
num <- c(2, 8, 5, 9, 3)
print(sort(num))
```

```
## [1] 2 3 5 8 9
```

```
library(stringr) # calling library
```

```r
x <- c("why", "video", "cross", "extra", "deal", "authority")
str_length(x)
```

```
## [1] 3 5 5 5 4 9
```

```r
str_c(x, collapse = ", ")
```

```
## [1] "why, video, cross, extra, deal, authority"
```

```r
str_sub(x, start = 1, end = 2)
```

```
## [1] "wh" "vi" "cr" "ex" "de" "au"
```

```r
m_str <- format(100)
print(m_str)
```

```
## [1] "100"
```

```r
# str cat
a <- "R"
b <- "em"
c <- "8"
d <- "horas"

str_c(a,b, sep = " ")
```

```
## [1] "R em"
```

```r
paste(a,b,c,d)
```

```
## [1] "R em 8 horas"
```

```r
str <- "Engenharia Biomédica - UFU"
str_length(str)
```

```
## [1] 26
```

```r
str <- "ENGENHARIA BIOMÉDICA - UFU"
str_to_lower(str)
```

```
## [1] "engenharia biomédica - ufu"
```

```
str <- "engenharia biomédica - ufu"
str_to_upper(str)
```

```
## [1] "ENGENHARIA BIOMÉDICA - UFU"
```

```
str <- "engenharia biomédica - ufu"
str_to_title(str)
```

```
## [1] "Engenharia Biomédica - Ufu"
```

```
str <- "engenharia biomédica - ufu"
str_replace(str,'ufu', "UFU")
```

```
## [1] "engenharia biomédica - UFU"
```

Parte 2

Exercício 7

> Refaça todos os exemplos apresentados neste módulo Parte 2.

```
# integer vector
v.inteiro <- integer(10)
v.inteiro <- c(1:10)

# double vector
v.double <- numeric(10)
v.double <- c(2.5, c(1:9))

# logical vector
v.logical <- logical(10)
v.logical <- c(F, T, F, F, T, F, T, T, T, F)

# complex vector
v.complex <- complex(10)
v.complex <- c(1+2i, 3+1i, 5-4i, 9-0.1i, 10-54i,
               100-1i, -2-2i, 3+3i, 10-0i, 8)

# raw vector
v.raw <- raw(length = 5)
rawToChar(v.raw)
```

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

```
v.raw[1] <- as.raw(10)
rawToChar(v.raw[1])
```

```
## [1] "\n"
```

```
v.raw[2] <- as.raw(5)
rawToChar(v.raw[2])
```

```
## [1] "\005"
```

```
print("céu azul") # vector of chars
```

```
## [1] "céu azul"
```

```
print("98.1") # double
```

```
## [1] "98.1"
```

```
print(2:8) # integer vector
```

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

```
print(1.8:6.8) # double vector
```

```
## [1] 1.8 2.8 3.8 4.8 5.8 6.8
```

```
print (c(3,4,5)) # also integer vector
```

```
## [1] 3 4 5
```

```
print(c("apple", "banana", "cherry")) # string vector
```

```
## [1] "apple"  "banana" "cherry"
```

```
seq(from = 1, to=2, by = 0.1) # ranging vector
```

```
##  [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
```

```
v1 <- seq(from=0, to=2, by =0.1)
print(v1)
print(v1[1])
print(v1[c(1, 4)])
v2 <- c("maçã", "banana", "amora")
print(v2[2])
```

```r
print(v2[c(1, 3)])
v3 <- c("maçã", "banana", "amora")
print(v3[c(FALSE, TRUE, FALSE)]) # only true values are displayed
v2 <- c("maçã", "banana", "amora")
print(v2)
print(v2[-2]) # removing second element

v1 <- c(1, 2, 3)
v2 <- c(4, 5, 6)

print(v1 + v2) # vector sum
print(v1 - v2) # vector sub
print(v1 * v2) # vector mul
print(v1/v2) # vector div

v1 <- c(4,5,6,7,8,9)
v2 <- c(2,3)
r <- v1 + v2

print(r)print(v2 - v1)

v<-c(1:10)
print(sort(v, decreasing = TRUE))
a<-seq(from=-1, to=1, by = 0.1)
print(sort(a, decreasing = FALSE))
ml <- list("a", "b", "c", "d")
print(ml)
print(ml[2])
ml[5] <- 45.67
print(ml)
ml[5] <- NULL
print(ml)
ml[2] <- "atualizei o segundo elemento"
print(ml[2])
l1 <- list("a", "b", 2)
l2 <- list("d", "e", 3)
l12  <- c(l1, l2)
print(l12)
v12 <- unlist(l12)
print(v12)

v <- c(1:12)
vnames <- list(
  c("row1", "row2"), # row names
  c("col1", "col2", "col3","col4","col5","col6")) # column names

m <- matrix(v, nrow = 2, ncol = 6, byrow = TRUE, dimnames = vnames)

print(m)

m <- matrix(v, nrow = 2, ncol = 6, byrow = FALSE, dimnames = vnames)

print(m)
print(m[2,3])
print(m[1,3])
print(m[2,6])
print(m[1,4])
```

```r
m2 <- matrix(c(4,5,6,7,8,9), nrow = 2)
print(m2)

print(m2 + m2)
print(m2 - m2)
print(m2 * m2)
print(m2/3*m2)

rownames <- c("r1", "r2", "r3")
colnames <- c("c1", "c2", "c3",  "c4")

m1 <- matrix(c(1:12), nrow = 3, ncol = 4, dimnames = list(rownames, colnames))

print(m1)

M1 = matrix(c(46,  41,  85,  94,  10,  21,  27,  39,  89), nrow = 3)

M2 = matrix(c(26,  54,  39,  32,  62,  85,  19,  57,  38), nrow = 3)

## multiplication
M_prod = M1 %*% M2

print(M_prod); cat("\n")

M_transp = t(M1)

print(M_transp); cat("\n")
```

```
## Error: <text>:26:9: unexpected symbol
## 25:
## 26: print(r)print
##             ^
```

```r
M1 = matrix(c(46,  41,  85,  94,  10,  21,  27,  39,  89), nrow = 3)
M2 = matrix(c(26,  54,  39,  32,  62,  85,  19,  57,  38), nrow = 3)
det(M1) # determinant of matrix
```

```
## [1] -27833
```

```r
M_inversa = solve(M1)

print(M_inversa); cat("\n")
```

```
##               [,1]       [,2]        [,3]
## [1,] -0.0025509288  0.2802069 -0.12201344
## [2,]  0.0120001437 -0.0646355  0.02468293
## [3,] -0.0003952143 -0.2523623  0.12194158
```

```r
solve(M1)%*%M1
```

```
##               [,1]          [,2] [,3]
## [1,]  1.000000e+00 -4.440892e-16    0
## [2,]  0.000000e+00  1.000000e+00    0
## [3,] -1.776357e-15  4.440892e-16    1
```

```
sum(diag(M1))
```

```
## [1] 145
```

```
crossprod(M1,M2)   # M1'M2
```

```
##      [,1]  [,2] [,3]
## [1,] 6725 11239 6441
## [2,] 3803  5413 3154
## [3,] 6279 10847 6118
```

```
kronecker(M1, M2)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,] 1196 1472  874 2444 3008 1786  702  864  513
##  [2,] 2484 2852 2622 5076 5828 5358 1458 1674 1539
##  [3,] 1794 3910 1748 3666 7990 3572 1053 2295 1026
##  [4,] 1066 1312  779  260  320  190 1014 1248  741
##  [5,] 2214 2542 2337  540  620  570 2106 2418 2223
##  [6,] 1599 3485 1558  390  850  380 1521 3315 1482
##  [7,] 2210 2720 1615  546  672  399 2314 2848 1691
##  [8,] 4590 5270 4845 1134 1302 1197 4806 5518 5073
##  [9,] 3315 7225 3230  819 1785  798 3471 7565 3382
```

```
print(array(1:8))
```

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

```
print(array(1:8, dim = c(2,4)))
```

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

```
print(array(1:16, dim = c(2,4,2)))
```

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

```
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]    9   11   13   15
## [2,]   10   12   14   16
```

```
rowname <- c("r1", "r2", "r3")
colname <- c("c1", "c2", "c3")
matrixname <- c("m1", "m2")

marray <- array(1:18, dim = c(3,3,2),
                dimnames = list(rowname,
                                colname,
                                matrixname))

print(marray)
```

```
## , , m1
##
##    c1 c2 c3
## r1  1  4  7
## r2  2  5  8
## r3  3  6  9
##
## , , m2
##
##    c1 c2 c3
## r1 10 13 16
## r2 11 14 17
## r3 12 15 18
```

```
print( marray[2, 3 ,1] )
```

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

```
print(marray[,,1])
```

```
##    c1 c2 c3
## r1  1  4  7
## r2  2  5  8
## r3  3  6  9
```

```
print(marray[2,,1])
```

```
## c1 c2 c3
##  2  5  8
```

```r
vector1 <- c(1,2,3)
vector2 <- c(4,5,6,7,8,9)

marray <- array(c(vector1, vector2),
                dim = c(3,3,1))

print(marray)
```

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

```r
result1 <- apply(marray, c(1), sum)
print(result1)
```

```
## [1] 12 15 18
```

```r
result2 <- apply(marray, c(2), sum)
apply(marray, c(3), sum)
```

```
## [1] 45
```

```r
result3 <- apply(marray, c(3), sort)
print(result3)
```

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

```r
myfactor <- factor(1:5, levels=1:8, labels = c("a"))
print(myfactor)
```

```
## [1] a1 a2 a3 a4 a5
## Levels: a1 a2 a3 a4 a5 a6 a7 a8
```

```r
data <- c("A", "C", "B", "B", "C", "A")
myfactor <- factor(data)
print(myfactor)
```

```
## [1] A C B B C A
## Levels: A B C
```

```
myfactor <- factor(1:5, levels = 1:3,
                   labels=c("a", "b", "c"))

print(myfactor)
```

```
## [1] a    b    c    <NA> <NA>
## Levels: a b c
```

```
data <- c("A", "C", "B", "B", "C", "A")
myfactor <- factor(data, labels=c("maçã", "banana", "uva"))
print(myfactor)
```

```
## [1] maçã   uva    banana banana uva    maçã
## Levels: maçã banana uva
```

```
myfactor <- factor(c("A", "B","C"),
                   levels = c("C", "B", "A"))
print(myfactor)
```

```
## [1] A B C
## Levels: C B A
```

```
L <- gl(4,2, labels = c("A", "B", "C", "D"))
print(L)
```

```
## [1] A A B B C C D D
## Levels: A B C D
```

```
L <- gl(4,5, labels = c("A", "B", "C", "D"))
print(L)
```

```
##  [1] A A A A A B B B B B C C C C C D D D D D
## Levels: A B C D
```

```
sex <- factor(c("male", "female", "female", "male"))

levels(sex)
```

```
## [1] "female" "male"
```

```r
nlevels(sex)
```

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

```r
food <- factor(c("low", "high", "medium", "high", "low", "medium", "high"))
levels(food)
```

```
## [1] "high"   "low"    "medium"
```

```r
food <- factor(food, levels = c("low", "medium", "high"))
levels(food)
```

```
## [1] "low"    "medium" "high"
```

```r
food <- factor(food, levels = c("low", "medium", "high"), ordered = TRUE)
levels(food)
```

```
## [1] "low"    "medium" "high"
```

```r
exercise <- factor(c("l", "n", "n", "i", "l"), levels = c("n", "l", "i"), ordered = TRUE)
print(exercise)
```

```
## [1] l n n i l
## Levels: n < l < i
```

```r
df <- data.frame(
  id = c(1:5),
  name = c("Silvana", "João", "Maria", "Roberto", "Carla"),
  score = c(85,99,95,92,96),
  year = c("2009", "2002", "2008", "2006", "2007"),
  stringsAsFactors = FALSE
)

print(df)
```

```
##   id    name score year
## 1  1 Silvana    85 2009
## 2  2    João    99 2002
## 3  3   Maria    95 2008
## 4  4 Roberto    92 2006
## 5  5   Carla    96 2007
```

```r
str(df)
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ id   : int  1 2 3 4 5
##  $ name : chr  "Silvana" "João" "Maria" "Roberto" ...
##  $ score: num  85 99 95 92 96
##  $ year : chr  "2009" "2002" "2008" "2006" ...
```

```
summary(df)
```

```
##        id        name               score          year
##  Min.   :1   Length:5           Min.   :85.0   Length:5
##  1st Qu.:2   Class :character   1st Qu.:92.0   Class :character
##  Median :3   Mode  :character   Median :95.0   Mode  :character
##  Mean   :3                      Mean   :93.4
##  3rd Qu.:4                      3rd Qu.:96.0
##  Max.   :5                      Max.   :99.0
```

```
print(df$id)
```

```
## [1] 1 2 3 4 5
```

```
print(df$name)
```

```
## [1] "Silvana" "João"    "Maria"   "Roberto" "Carla"
```

```
print(df$score)
```

```
## [1] 85 99 95 92 96
```

```
print(df$year)
```

```
## [1] "2009" "2002" "2008" "2006" "2007"
```

```
df$idade <- c(30,40,17,12,98)
print(df)
```

```
##   id    name score year idade
## 1  1 Silvana    85 2009    30
## 2  2    João    99 2002    40
## 3  3   Maria    95 2008    17
## 4  4 Roberto    92 2006    12
## 5  5   Carla    96 2007    98
```

```
newRows <- data.frame(

  id = c(6,7),
```

```
  name = c("Rui", "Alana"),
  score = c(100, 33),
  year = c("1999", "1997"),
  idade = c(43, 33),
  stringsAsFactors = FALSE

)

df <- rbind(df, newRows)
print(df)
```

```
##   id    name score year idade
## 1  1 Silvana    85 2009    30
## 2  2    João    99 2002    40
## 3  3   Maria    95 2008    17
## 4  4 Roberto    92 2006    12
## 5  5   Carla    96 2007    98
## 6  6     Rui   100 1999    43
## 7  7   Alana    33 1997    33
```

```
id <- c("001", "002", "003", "004")
name <- c("Anna", "Judy", "Tony", "Nacy")
age <- c(18, 16, 20, 19)

members <- cbind(id, name,age)
print(members)
```

```
##      id    name   age
## [1,] "001" "Anna" "18"
## [2,] "002" "Judy" "16"
## [3,] "003" "Tony" "20"
## [4,] "004" "Nacy" "19"
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(nycflights13)
dim(flights)
```

```
## [1] 336776     19
```

```
str(flights)
```

```
## tibble [336,776 x 19] (S3: tbl_df/tbl/data.frame)
##  $ year          : int [1:336776] 2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month         : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ day           : int [1:336776] 1 1 1 1 1 1 1 1 1 1 ...
##  $ dep_time      : int [1:336776] 517 533 542 544 554 554 555 557 557 558 ...
##  $ sched_dep_time: int [1:336776] 515 529 540 545 600 558 600 600 600 600 ...
##  $ dep_delay     : num [1:336776] 2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ arr_time      : int [1:336776] 830 850 923 1004 812 740 913 709 838 753 ...
##  $ sched_arr_time: int [1:336776] 819 830 850 1022 837 728 854 723 846 745 ...
##  $ arr_delay     : num [1:336776] 11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier       : chr [1:336776] "UA" "UA" "AA" "B6" ...
##  $ flight        : int [1:336776] 1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ tailnum       : chr [1:336776] "N14228" "N24211" "N619AA" "N804JB" ...
##  $ origin        : chr [1:336776] "EWR" "LGA" "JFK" "JFK" ...
##  $ dest          : chr [1:336776] "IAH" "IAH" "MIA" "BQN" ...
##  $ air_time      : num [1:336776] 227 227 160 183 116 150 158 53 140 138 ...
##  $ distance      : num [1:336776] 1400 1416 1089 1576 762 ...
##  $ hour          : num [1:336776] 5 5 5 5 6 5 6 6 6 6 ...
##  $ minute        : num [1:336776] 15 29 40 45 0 58 0 0 0 0 ...
##  $ time_hour     : POSIXct[1:336776], format: "2013-01-01 05:00:00" "2013-01-01 05:00:00"
...
```

```
filter(flights, month == 1, day == 1)
```

```
## # A tibble: 842 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 832 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
arrange(flights, dep_time, day, month, year)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     7     1        1           2029       212      236           2359
##  2  2013     4     5        1           2359         2      410            339
##  3  2013     3     8        1           2355         6      431            440
```

```
##  4  2013     4    10        1           1930       271      106           2101
##  5  2013     2    11        1           2100       181      111           2225
##  6  2013     1    13        1           2249        72      108           2357
##  7  2013    11    13        1           2359         2      442            440
##  8  2013     8    15        1           2359         2      336            340
##  9  2013    12    16        1           2359         2      447            437
## 10  2013     3    18        1           2128       153      247           2355
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
arrange(flights, dep_time, desc(day), month, year)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1    31        1           2100       181      124           2225
##  2  2013    12    30        1           2359         2      441            437
##  3  2013     7    28        1           2359         2      423            350
##  4  2013     7    27        1           2359         2      345            340
##  5  2013    12    26        1           2359         2      437            440
##  6  2013     5    25        1           2359         2      336            341
##  7  2013     6    25        1           2130       151      249             14
##  8  2013     2    24        1           2245        76      121           2354
##  9  2013     6    24        1           1950       251      105           2130
## 10  2013     5    22        1           1935       266      154           2140
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
select(flights, year, month, day)
```

```
## # A tibble: 336,776 x 3
##     year month   day
##    <int> <int> <int>
##  1  2013     1     1
##  2  2013     1     1
##  3  2013     1     1
##  4  2013     1     1
##  5  2013     1     1
##  6  2013     1     1
##  7  2013     1     1
##  8  2013     1     1
##  9  2013     1     1
## 10  2013     1     1
## # ... with 336,766 more rows
```

```
rename(flights, tail_num = tailnum)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
```

```
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tail_num <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
mutate(flights,
  gain = arr_delay - dep_delay,
  speed = distance / air_time * 60
)
```

```
## # A tibble: 336,776 x 21
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 13 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>,
## #   gain <dbl>, speed <dbl>
```

```
transmute(flights,
  gain = arr_delay - dep_delay,
  gain_per_hour = gain / (air_time / 60)
)
```

```
## # A tibble: 336,776 x 2
##     gain gain_per_hour
##    <dbl>         <dbl>
##  1     9          2.38
##  2    16          4.23
##  3    31         11.6
##  4   -17         -5.57
##  5   -19         -9.83
##  6    16          6.4
##  7    24          9.11
##  8   -11        -12.5
##  9    -5         -2.14
```

```
## 10    10          4.35
## # ... with 336,766 more rows
```

```
summarise(flights,
  delay = mean(dep_delay, na.rm = TRUE)
)
```

```
## # A tibble: 1 x 1
##   delay
##   <dbl>
## 1  12.6
```

```
sample_n(flights, 10)
```

```
## # A tibble: 10 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     3    10     1341           1345        -4     1457           1453
##  2  2013     5    23      657            700        -3      931            950
##  3  2013     9    28     1343           1345        -2     1625           1650
##  4  2013    10    11     1934           1934         0     2205           2226
##  5  2013     9     4      948            955        -7     1056           1110
##  6  2013     4    16     1627           1627         0     1823           1824
##  7  2013     8    17     1257           1300        -3     1451           1505
##  8  2013     5     9     2032           2012        20     2325           2330
##  9  2013    11    26     1700           1700         0     2020           2025
## 10  2013     3    31     1229           1235        -6     1539           1552
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
sample_frac(flights, 0.01)
```

```
## # A tibble: 3,368 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     3    15      748            655        53      850            813
##  2  2013     6     1      626            630        -4      900            916
##  3  2013     8     2      559            600        -1      711            711
##  4  2013     2    10     1329           1320         9     1659           1650
##  5  2013     2    14     1625           1635       -10     1744           1810
##  6  2013     6    19      758            756         2     1040           1037
##  7  2013    10    26     1505           1505         0     1720           1709
##  8  2013     7     6     1854           1856        -2     2035           2104
##  9  2013     7    15      823            829        -6      947           1028
## 10  2013    10    21     1001           1005        -4     1212           1217
## # ... with 3,358 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
by_tailnum <- group_by(flights, tailnum)

delay <- summarise(by_tailnum,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE))

delay <- filter(delay, count > 20, dist < 2000)

daily <- group_by(flights, year, month, day)
(per_day   <- summarise(daily, flights = n()))
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the `.groups`
argument.
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day flights
##    <int> <int> <int>   <int>
##  1  2013     1     1     842
##  2  2013     1     2     943
##  3  2013     1     3     914
##  4  2013     1     4     915
##  5  2013     1     5     720
##  6  2013     1     6     832
##  7  2013     1     7     933
##  8  2013     1     8     899
##  9  2013     1     9     902
## 10  2013     1    10     932
## # ... with 355 more rows
```

```r
select(flights, year)
```

```
## # A tibble: 336,776 x 1
##     year
##    <int>
##  1  2013
##  2  2013
##  3  2013
##  4  2013
##  5  2013
##  6  2013
##  7  2013
##  8  2013
##  9  2013
## 10  2013
## # ... with 336,766 more rows
```

```r
select(flights, 1)
```

```
## # A tibble: 336,776 x 1
##     year
```

```
##    <int>
##  1  2013
##  2  2013
##  3  2013
##  4  2013
##  5  2013
##  6  2013
##  7  2013
##  8  2013
##  9  2013
## 10  2013
## # ... with 336,766 more rows
```

```
select(flights, starts_with("dep"))
```

```
## # A tibble: 336,776 x 2
##    dep_time dep_delay
##       <int>     <dbl>
##  1      517         2
##  2      533         4
##  3      542         2
##  4      544        -1
##  5      554        -6
##  6      554        -4
##  7      555        -5
##  8      557        -3
##  9      557        -3
## 10      558        -2
## # ... with 336,766 more rows
```

```
# Piping
a1 <- group_by(flights, year, month, day)
a2 <- select(a1, arr_delay, dep_delay)
```

```
## Adding missing grouping variables: `year`, `month`, `day`
```

```
a3 <- summarise(a2,
  arr = mean(arr_delay, na.rm = TRUE),
  dep = mean(dep_delay, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the `.groups`
argument.
```

```
a4 <- filter(a3, arr > 30 | dep > 30)

# or...
flights %>%
  group_by(year, month, day) %>%
  select(arr_delay, dep_delay) %>%
  summarise(
    arr = mean(arr_delay, na.rm = TRUE),
```

```
    dep = mean(dep_delay, na.rm = TRUE)
  ) %>%
  filter(arr > 30 | dep > 30)
```

```
## Adding missing grouping variables: `year`, `month`, `day`
```

```
## `summarise()` has grouped output by 'year', 'month'. You can override using the `.groups`
argument.
```

```
## # A tibble: 49 x 5
## # Groups:   year, month [11]
##     year month   day   arr   dep
##    <int> <int> <int> <dbl> <dbl>
##  1  2013     1    16  34.2  24.6
##  2  2013     1    31  32.6  28.7
##  3  2013     2    11  36.3  39.1
##  4  2013     2    27  31.3  37.8
##  5  2013     3     8  85.9  83.5
##  6  2013     3    18  41.3  30.1
##  7  2013     4    10  38.4  33.0
##  8  2013     4    12  36.0  34.8
##  9  2013     4    18  36.0  34.9
## 10  2013     4    19  47.9  46.1
## # ... with 39 more rows
```

Exercício 8

> Faça um programa em R que avalie o nível de atividade física dos estudantes da disciplina de Processamento de Sinais Biomédicos.

As seguintes variáveis devem ser consideradas:

- Identificador do participante (id)
- Curso do participante (cp)
- Período do curso (pc)
- Idade (idade)
- Peso (peso)
- Altura (altura)
- Nível de atividade física (nat)
    - nat = “nula” - nenhuma vez por semana
    - nat = “baixa” - de uma a duas vezes por semana
    - nat = “moderada” - três a cinco vezes por semana
    - nat = “avançada” - acima de cinco vezes por semana

> Construa um data frame que armazene as variáveis coletadas e faça os seguintes cálculos:

- Número de participantes por cada uma das categorias de nível de atividade fisica (nat)
- Percentual de participantes por cada uma das categorias de nível de atividade fisica (nat)

- Valor médios das variáveis: idade, peso e altura (calcular os valores médios globais, por período de curso e por curso)

```
library(dplyr)
df <- data.frame( # it was used generic data
  id = 1:5,
  cp = c("curso 1", "curso 1", "curso 2", "curso 1", "curso 2"),
  pc = c(5, 2, 4, 9, 9),
  peso = c(40, 80, 90, 50, 60),
  idade = c(20, 22, 21, 19, 18),
  altura = c(180, 129, 160, 170, 180),
  nat = c("avançada", "nula", "nula", "moderada", "baixa")
)
```

Segue abaixo o número de participantes por cada nível de atividade física:

```
partPerNat <- df %>% count(nat)
print(partPerNat)
```

```
##        nat n
## 1 avançada 1
## 2    baixa 1
## 3 moderada 1
## 4     nula 2
```

Agora o dado anterior com a opção em formato percentual inserida:

```
numberOfParticipants <- sum(partPerNat$n)
partPerNat$percentual <- partPerNat$n * (100/numberOfParticipants)
print(partPerNat)
```

```
##        nat n percentual
## 1 avançada 1         20
## 2    baixa 1         20
## 3 moderada 1         20
## 4     nula 2         40
```

O próximo passo é calcular a média dos valores numéricos peso, idade e altura do *data frame* para várias situações diferentes. Será utilizado o vetor abaixo para indexar as colunas desejadas:

```
idxCol <- c("peso", "idade", "altura") # selected columns
```

A primeira consiste nos valoes médios globais, resultando em:

```
df_globalMean <- colMeans(df[idxCol], na.rm=TRUE)
print(df_globalMean)
```

```
##   peso  idade altura
##   64.0   20.0  163.8
```

Outro quesito é realizar a média baseados em valores filtrados por período de curso, resultando em (nesse caso será exemplificado considerando o nono período):

```
selectedPeriod <- 9
df_sortedByPeriod <- df %>% filter(pc == selectedPeriod)
df_periodMean <- colMeans(df_sortedByPeriod[idxCol], na.rm=TRUE)
print(df_periodMean)
```

```
##   peso  idade altura
##   55.0   18.5  175.0
```

Por fim, filtrando por curso (nesse caso será exemplificado considerando o curso 1):

```
selectedCourse <- "curso 1"
df_sortedByCourse <- df %>% filter(cp == selectedCourse)
df_courseMean <- colMeans(df_sortedByCourse[idxCol], na.rm=TRUE)
print(df_courseMean)
```

```
##      peso     idade    altura
##  56.66667  20.33333 159.66667
```

Parte 3

Exercício 9

> Faça o tutorial sobre o package dplyr disponível em https://www.listendata.com/2016/08/dplyr-tutorial.html

```
library(dplyr) # load dplyr
mydata = read.csv("https://raw.githubusercontent.com/deepanshu88/data/master/sampledata.csv")
# loading data from .csv hosted in url
print(sample_n(mydata,3)) # random rows can be selected for visualization
```

```
##   Index    State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008
## 1     M Missouri 1221316 1858368 1773451 1573967 1374863 1486197 1735099
## 2     A  Alabama 1296530 1317711 1118631 1492583 1107408 1440134 1945229
## 3     O   Oregon 1794912 1726665 1805445 1133510 1502242 1419251 1482786
##     Y2009   Y2010   Y2011   Y2012   Y2013   Y2014   Y2015
## 1 1800620 1164202 1425363 1800052 1698105 1767835 1996005
## 2 1944173 1237582 1440756 1186741 1852841 1558906 1916661
## 3 1862351 1103794 1935687 1905378 1522129 1509171 1893515
```

```
print(sample_frac(mydata,0.1)) # visualize  randomly percentage of rows
```

```
##   Index         State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008
## 1     N    New Jersey 1605532 1141514 1613550 1181452 1541327 1156804 1568034
## 2     N New Hampshire 1419776 1854370 1195119 1990062 1645430 1286967 1762936
## 3     C   Connecticut 1610512 1232844 1181949 1518933 1841266 1976976 1764457
```

```
## 4     M      Missouri 1221316 1858368 1773451 1573967 1374863 1486197 1735099
## 5     A        Alaska 1170302 1960378 1818085 1447852 1861639 1465841 1551826
##     Y2009   Y2010   Y2011   Y2012   Y2013   Y2014   Y2015
## 1 1357418 1443718 1390010 1202326 1100990 1850165 1183568
## 2 1763211 1265642 1704297 1131298 1197576 1242623 1963313
## 3 1972730 1968730 1945524 1228529 1582249 1503156 1718072
## 4 1800620 1164202 1425363 1800052 1698105 1767835 1996005
## 5 1436541 1629616 1230866 1512804 1985302 1580394 1979143
```

```
x1 = distinct(mydata) # eliminate duplicates in data on all columns
x2 = distinct(mydata, Index, .keep_all= TRUE) # same as before but maintains all other column
s
x2 = distinct(mydata, Index, Y2010, .keep_all= TRUE) # remove duplicates rows based in multip
les columns/variables
mydata2 = select(mydata, Index, State:Y2008) # select some columns of a data frame
select(mydata, -Index, -State) # drop selected columns
```

```
##      Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009   Y2010
## 1  1296530 1317711 1118631 1492583 1107408 1440134 1945229 1944173 1237582
## 2  1170302 1960378 1818085 1447852 1861639 1465841 1551826 1436541 1629616
## 3  1742027 1968140 1377583 1782199 1102568 1109382 1752886 1554330 1300521
## 4  1485531 1994927 1119299 1947979 1669191 1801213 1188104 1628980 1669295
## 5  1685349 1675807 1889570 1480280 1735069 1812546 1487315 1663809 1624509
## 6  1343824 1878473 1886149 1236697 1871471 1814218 1875146 1752387 1913275
## 7  1610512 1232844 1181949 1518933 1841266 1976976 1764457 1972730 1968730
## 8  1330403 1268673 1706751 1403759 1441351 1300836 1762096 1553585 1370984
## 9  1111437 1993741 1374643 1827949 1803852 1595981 1193245 1739748 1707823
## 10 1964626 1468852 1419738 1362787 1339608 1278550 1756185 1818438 1198403
## 11 1929009 1541565 1810773 1779091 1326846 1223770 1773090 1630325 1145473
## 12 1461570 1200280 1213993 1245931 1459383 1430465 1919423 1928416 1330509
## 13 1353210 1438538 1739154 1541015 1122387 1772050 1335481 1748608 1436809
## 14 1508356 1527440 1493029 1261353 1540274 1747614 1871645 1658551 1422021
## 15 1776918 1734104 1269927 1204117 1848073 1129546 1139551 1883976 1999102
## 16 1499269 1444576 1576367 1388924 1554813 1452911 1317983 1150783 1751389
## 17 1509054 1290700 1522230 1532094 1104256 1863278 1949478 1561528 1550433
## 18 1813878 1448846 1800760 1250524 1137913 1911227 1301848 1956681 1350895
## 19 1584734 1110625 1868456 1751920 1233709 1920301 1185085 1124853 1498662
## 20 1582720 1678622 1208496 1912040 1438549 1330014 1295877 1969163 1627262
## 21 1579713 1404700 1849798 1397738 1310270 1789128 1112765 1967225 1486246
## 22 1647582 1686259 1620601 1777250 1531641 1380529 1978904 1567651 1761048
## 23 1295635 1149931 1601027 1340716 1729449 1567494 1990431 1575185 1267626
## 24 1729921 1675204 1903907 1561839 1985692 1148621 1328133 1890633 1995304
## 25 1983285 1292558 1631325 1943311 1354579 1731643 1428291 1568049 1383227
## 26 1221316 1858368 1773451 1573967 1374863 1486197 1735099 1800620 1164202
## 27 1877154 1540099 1332722 1273327 1625721 1983568 1251742 1592690 1350619
## 28 1885081 1309769 1425527 1240465 1500594 1278272 1140598 1270585 1128711
## 29 1426117 1114500 1119707 1758830 1694526 1765826 1903270 1231480 1526066
## 30 1419776 1854370 1195119 1990062 1645430 1286967 1762936 1763211 1265642
## 31 1605532 1141514 1613550 1181452 1541327 1156804 1568034 1357418 1443718
## 32 1819239 1226057 1935991 1124400 1723493 1475985 1237704 1820856 1801430
## 33 1395149 1611371 1170675 1446810 1426941 1463171 1732098 1426216 1604531
## 34 1616742 1292223 1482792 1532347 1158716 1827420 1267737 1116168 1791535
## 35 1618807 1510193 1876940 1443172 1425030 1868788 1720352 1671468 1534571
## 36 1802132 1648498 1441386 1670280 1534888 1314824 1516621 1511460 1585465
## 37 1173918 1334639 1663622 1798714 1312574 1708245 1256746 1853142 1673831
## 38 1794912 1726665 1805445 1133510 1502242 1419251 1482786 1862351 1103794
```

```
## 39 1320191 1446723 1218591 1122030 1971479 1563062 1274168 1571032 1433835
## 40 1501744 1942942 1266657 1961923 1835983 1234040 1151409 1993136 1983569
## 41 1631522 1803455 1425193 1458191 1538731 1825195 1250499 1864685 1345102
## 42 1159037 1150689 1660148 1417141 1418586 1279134 1171870 1852424 1554782
## 43 1811867 1485909 1974179 1157059 1786132 1399191 1826406 1326460 1231739
## 44 1520591 1310777 1957713 1907326 1873544 1655483 1785986 1827503 1447457
## 45 1771096 1195861 1979395 1241662 1437456 1859416 1939284 1915865 1619186
## 46 1146902 1832249 1492704 1579265 1332048 1563537 1123567 1618583 1326369
## 47 1134317 1163996 1891068 1853855 1708715 1197698 1803330 1590043 1516758
## 48 1977749 1687136 1199490 1163092 1334864 1621989 1545621 1555554 1179331
## 49 1677347 1380662 1176100 1888948 1922085 1740826 1238174 1539322 1539603
## 50 1788920 1518578 1289663 1436888 1251678 1721874 1980167 1901394 1648755
## 51 1775190 1498098 1198212 1881688 1750527 1523124 1587602 1504455 1282142
##      Y2011   Y2012   Y2013   Y2014   Y2015
## 1  1440756 1186741 1852841 1558906 1916661
## 2  1230866 1512804 1985302 1580394 1979143
## 3  1130709 1907284 1363279 1525866 1647724
## 4  1928238 1216675 1591896 1360959 1329341
## 5  1639670 1921845 1156536 1388461 1644607
## 6  1665877 1491604 1178355 1383978 1330736
## 7  1945524 1228529 1582249 1503156 1718072
## 8  1318669 1984027 1671279 1803169 1627508
## 9  1353449 1979708 1912654 1782169 1410183
## 10 1497051 1131928 1107448 1407784 1170389
## 11 1851245 1850111 1887157 1259353 1725470
## 12 1902816 1695126 1517184 1948108 1150882
## 13 1456340 1643855 1312561 1713718 1757171
## 14 1751422 1696729 1915435 1645465 1583516
## 15 1559924 1905760 1129794 1988394 1467614
## 16 1992996 1501879 1173694 1431705 1641866
## 17 1465812 1882929 1410249 1930090 1385528
## 18 1512894 1916616 1878271 1722762 1913350
## 19 1210385 1234234 1287663 1908602 1403857
## 20 1706080 1437088 1318546 1116792 1529233
## 21 1872327 1175819 1314343 1979529 1569566
## 22 1658538 1482203 1731917 1669749 1963337
## 23 1274673 1709853 1815596 1965196 1646634
## 24 1575533 1910216 1972021 1515366 1864553
## 25 1629132 1988270 1907777 1649668 1991232
## 26 1425363 1800052 1698105 1767835 1996005
## 27 1520064 1185225 1465705 1110394 1125903
## 28 1187207 1569665 1690920 1459243 1802211
## 29 1143343 1980195 1283813 1225348 1903804
## 30 1704297 1131298 1197576 1242623 1963313
## 31 1390010 1202326 1100990 1850165 1183568
## 32 1653384 1475715 1623388 1533494 1868612
## 33 1683687 1500089 1718837 1619033 1367705
## 34 1553750 1472258 1104893 1596452 1229085
## 35 1271132 1430978 1529024 1563898 1604118
## 36 1887714 1227303 1840898 1880804 1573117
## 37 1822933 1674707 1900523 1956742 1307678
## 38 1935687 1905378 1522129 1509171 1893515
## 39 1483292 1290329 1475344 1931500 1668232
## 40 1781016 1909119 1531212 1990412 1611730
## 41 1116203 1532332 1591735 1188417 1110655
## 42 1647245 1811156 1147488 1302834 1136443
## 43 1469785 1849041 1560887 1349173 1162164
## 44 1978374 1882532 1698698 1646508 1705322
```

```
## 45 1288285 1108281 1123353 1801019 1729273
## 46 1792600 1714960 1146278 1282790 1565924
## 47 1171686 1262342 1647032 1706707 1850394
## 48 1150089 1775787 1273834 1387428 1377341
## 49 1872519 1462137 1683127 1204344 1198791
## 50 1940943 1729177 1510119 1701650 1846238
## 51 1881814 1673668 1994022 1204029 1853858
```

```
select(mydata, -c(Index,State)) # same results as before, but different syntax
```

```
##      Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009   Y2010
## 1  1296530 1317711 1118631 1492583 1107408 1440134 1945229 1944173 1237582
## 2  1170302 1960378 1818085 1447852 1861639 1465841 1551826 1436541 1629616
## 3  1742027 1968140 1377583 1782199 1102568 1109382 1752886 1554330 1300521
## 4  1485531 1994927 1119299 1947979 1669191 1801213 1188104 1628980 1669295
## 5  1685349 1675807 1889570 1480280 1735069 1812546 1487315 1663809 1624509
## 6  1343824 1878473 1886149 1236697 1871471 1814218 1875146 1752387 1913275
## 7  1610512 1232844 1181949 1518933 1841266 1976976 1764457 1972730 1968730
## 8  1330403 1268673 1706751 1403759 1441351 1300836 1762096 1553585 1370984
## 9  1111437 1993741 1374643 1827949 1803852 1595981 1193245 1739748 1707823
## 10 1964626 1468852 1419738 1362787 1339608 1278550 1756185 1818438 1198403
## 11 1929009 1541565 1810773 1779091 1326846 1223770 1773090 1630325 1145473
## 12 1461570 1200280 1213993 1245931 1459383 1430465 1919423 1928416 1330509
## 13 1353210 1438538 1739154 1541015 1122387 1772050 1335481 1748608 1436809
## 14 1508356 1527440 1493029 1261353 1540274 1747614 1871645 1658551 1422021
## 15 1776918 1734104 1269927 1204117 1848073 1129546 1139551 1883976 1999102
## 16 1499269 1444576 1576367 1388924 1554813 1452911 1317983 1150783 1751389
## 17 1509054 1290700 1522230 1532094 1104256 1863278 1949478 1561528 1550433
## 18 1813878 1448846 1800760 1250524 1137913 1911227 1301848 1956681 1350895
## 19 1584734 1110625 1868456 1751920 1233709 1920301 1185085 1124853 1498662
## 20 1582720 1678622 1208496 1912040 1438549 1330014 1295877 1969163 1627262
## 21 1579713 1404700 1849798 1397738 1310270 1789128 1112765 1967225 1486246
## 22 1647582 1686259 1620601 1777250 1531641 1380529 1978904 1567651 1761048
## 23 1295635 1149931 1601027 1340716 1729449 1567494 1990431 1575185 1267626
## 24 1729921 1675204 1903907 1561839 1985692 1148621 1328133 1890633 1995304
## 25 1983285 1292558 1631325 1943311 1354579 1731643 1428291 1568049 1383227
## 26 1221316 1858368 1773451 1573967 1374863 1486197 1735099 1800620 1164202
## 27 1877154 1540099 1332722 1273327 1625721 1983568 1251742 1592690 1350619
## 28 1885081 1309769 1425527 1240465 1500594 1278272 1140598 1270585 1128711
## 29 1426117 1114500 1119707 1758830 1694526 1765826 1903270 1231480 1526066
## 30 1419776 1854370 1195119 1990062 1645430 1286967 1762936 1763211 1265642
## 31 1605532 1141514 1613550 1181452 1541327 1156804 1568034 1357418 1443718
## 32 1819239 1226057 1935991 1124400 1723493 1475985 1237704 1820856 1801430
## 33 1395149 1611371 1170675 1446810 1426941 1463171 1732098 1426216 1604531
## 34 1616742 1292223 1482792 1532347 1158716 1827420 1267737 1116168 1791535
## 35 1618807 1510193 1876940 1443172 1425030 1868788 1720352 1671468 1534571
## 36 1802132 1648498 1441386 1670280 1534888 1314824 1516621 1511460 1585465
## 37 1173918 1334639 1663622 1798714 1312574 1708245 1256746 1853142 1673831
## 38 1794912 1726665 1805445 1133510 1502242 1419251 1482786 1862351 1103794
## 39 1320191 1446723 1218591 1122030 1971479 1563062 1274168 1571032 1433835
## 40 1501744 1942942 1266657 1961923 1835983 1234040 1151409 1993136 1983569
## 41 1631522 1803455 1425193 1458191 1538731 1825195 1250499 1864685 1345102
## 42 1159037 1150689 1660148 1417141 1418586 1279134 1171870 1852424 1554782
## 43 1811867 1485909 1974179 1157059 1786132 1399191 1826406 1326460 1231739
## 44 1520591 1310777 1957713 1907326 1873544 1655483 1785986 1827503 1447457
## 45 1771096 1195861 1979395 1241662 1437456 1859416 1939284 1915865 1619186
```

```
## 46 1146902 1832249 1492704 1579265 1332048 1563537 1123567 1618583 1326369
## 47 1134317 1163996 1891068 1853855 1708715 1197698 1803330 1590043 1516758
## 48 1977749 1687136 1199490 1163092 1334864 1621989 1545621 1555554 1179331
## 49 1677347 1380662 1176100 1888948 1922085 1740826 1238174 1539322 1539603
## 50 1788920 1518578 1289663 1436888 1251678 1721874 1980167 1901394 1648755
## 51 1775190 1498098 1198212 1881688 1750527 1523124 1587602 1504455 1282142
##      Y2011   Y2012   Y2013   Y2014   Y2015
## 1  1440756 1186741 1852841 1558906 1916661
## 2  1230866 1512804 1985302 1580394 1979143
## 3  1130709 1907284 1363279 1525866 1647724
## 4  1928238 1216675 1591896 1360959 1329341
## 5  1639670 1921845 1156536 1388461 1644607
## 6  1665877 1491604 1178355 1383978 1330736
## 7  1945524 1228529 1582249 1503156 1718072
## 8  1318669 1984027 1671279 1803169 1627508
## 9  1353449 1979708 1912654 1782169 1410183
## 10 1497051 1131928 1107448 1407784 1170389
## 11 1851245 1850111 1887157 1259353 1725470
## 12 1902816 1695126 1517184 1948108 1150882
## 13 1456340 1643855 1312561 1713718 1757171
## 14 1751422 1696729 1915435 1645465 1583516
## 15 1559924 1905760 1129794 1988394 1467614
## 16 1992996 1501879 1173694 1431705 1641866
## 17 1465812 1882929 1410249 1930090 1385528
## 18 1512894 1916616 1878271 1722762 1913350
## 19 1210385 1234234 1287663 1908602 1403857
## 20 1706080 1437088 1318546 1116792 1529233
## 21 1872327 1175819 1314343 1979529 1569566
## 22 1658538 1482203 1731917 1669749 1963337
## 23 1274673 1709853 1815596 1965196 1646634
## 24 1575533 1910216 1972021 1515366 1864553
## 25 1629132 1988270 1907777 1649668 1991232
## 26 1425363 1800052 1698105 1767835 1996005
## 27 1520064 1185225 1465705 1110394 1125903
## 28 1187207 1569665 1690920 1459243 1802211
## 29 1143343 1980195 1283813 1225348 1903804
## 30 1704297 1131298 1197576 1242623 1963313
## 31 1390010 1202326 1100990 1850165 1183568
## 32 1653384 1475715 1623388 1533494 1868612
## 33 1683687 1500089 1718837 1619033 1367705
## 34 1553750 1472258 1104893 1596452 1229085
## 35 1271132 1430978 1529024 1563898 1604118
## 36 1887714 1227303 1840898 1880804 1573117
## 37 1822933 1674707 1900523 1956742 1307678
## 38 1935687 1905378 1522129 1509171 1893515
## 39 1483292 1290329 1475344 1931500 1668232
## 40 1781016 1909119 1531212 1990412 1611730
## 41 1116203 1532332 1591735 1188417 1110655
## 42 1647245 1811156 1147488 1302834 1136443
## 43 1469785 1849041 1560887 1349173 1162164
## 44 1978374 1882532 1698698 1646508 1705322
## 45 1288285 1108281 1123353 1801019 1729273
## 46 1792600 1714960 1146278 1282790 1565924
## 47 1171686 1262342 1647032 1706707 1850394
## 48 1150089 1775787 1273834 1387428 1377341
## 49 1872519 1462137 1683127 1204344 1198791
## 50 1940943 1729177 1510119 1701650 1846238
## 51 1881814 1673668 1994022 1204029 1853858
```

```r
mydata3 = select(mydata, starts_with("Y")) # select columns that starts according to key cha
r/word
mydata33 = select(mydata, -starts_with("Y")) # same as before, but instead selecting it drops
any column that matches
mydata4 = select(mydata, contains("I")) # select columns that contains char/word
mydata5 = select(mydata, State, everything()) # keeps column State in front of every other on
e
rename(mydata, Index1=Index) # rename column
```

```
##    Index1                State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007
## 1       A              Alabama 1296530 1317711 1118631 1492583 1107408 1440134
## 2       A               Alaska 1170302 1960378 1818085 1447852 1861639 1465841
## 3       A              Arizona 1742027 1968140 1377583 1782199 1102568 1109382
## 4       A             Arkansas 1485531 1994927 1119299 1947979 1669191 1801213
## 5       C           California 1685349 1675807 1889570 1480280 1735069 1812546
## 6       C             Colorado 1343824 1878473 1886149 1236697 1871471 1814218
## 7       C          Connecticut 1610512 1232844 1181949 1518933 1841266 1976976
## 8       D             Delaware 1330403 1268673 1706751 1403759 1441351 1300836
## 9       D District of Columbia 1111437 1993741 1374643 1827949 1803852 1595981
## 10      F              Florida 1964626 1468852 1419738 1362787 1339608 1278550
## 11      G              Georgia 1929009 1541565 1810773 1779091 1326846 1223770
## 12      H               Hawaii 1461570 1200280 1213993 1245931 1459383 1430465
## 13      I                Idaho 1353210 1438538 1739154 1541015 1122387 1772050
## 14      I             Illinois 1508356 1527440 1493029 1261353 1540274 1747614
## 15      I              Indiana 1776918 1734104 1269927 1204117 1848073 1129546
## 16      I                 Iowa 1499269 1444576 1576367 1388924 1554813 1452911
## 17      K               Kansas 1509054 1290700 1522230 1532094 1104256 1863278
## 18      K             Kentucky 1813878 1448846 1800760 1250524 1137913 1911227
## 19      L            Louisiana 1584734 1110625 1868456 1751920 1233709 1920301
## 20      M                Maine 1582720 1678622 1208496 1912040 1438549 1330014
## 21      M             Maryland 1579713 1404700 1849798 1397738 1310270 1789128
## 22      M        Massachusetts 1647582 1686259 1620601 1777250 1531641 1380529
## 23      M             Michigan 1295635 1149931 1601027 1340716 1729449 1567494
## 24      M            Minnesota 1729921 1675204 1903907 1561839 1985692 1148621
## 25      M          Mississippi 1983285 1292558 1631325 1943311 1354579 1731643
## 26      M             Missouri 1221316 1858368 1773451 1573967 1374863 1486197
## 27      M              Montana 1877154 1540099 1332722 1273327 1625721 1983568
## 28      N             Nebraska 1885081 1309769 1425527 1240465 1500594 1278272
## 29      N               Nevada 1426117 1114500 1119707 1758830 1694526 1765826
## 30      N        New Hampshire 1419776 1854370 1195119 1990062 1645430 1286967
## 31      N           New Jersey 1605532 1141514 1613550 1181452 1541327 1156804
## 32      N           New Mexico 1819239 1226057 1935991 1124400 1723493 1475985
## 33      N             New York 1395149 1611371 1170675 1446810 1426941 1463171
## 34      N       North Carolina 1616742 1292223 1482792 1532347 1158716 1827420
## 35      N         North Dakota 1618807 1510193 1876940 1443172 1425030 1868788
## 36      O                 Ohio 1802132 1648498 1441386 1670280 1534888 1314824
## 37      O             Oklahoma 1173918 1334639 1663622 1798714 1312574 1708245
## 38      O               Oregon 1794912 1726665 1805445 1133510 1502242 1419251
## 39      P         Pennsylvania 1320191 1446723 1218591 1122030 1971479 1563062
## 40      R         Rhode Island 1501744 1942942 1266657 1961923 1835983 1234040
## 41      S       South Carolina 1631522 1803455 1425193 1458191 1538731 1825195
## 42      S         South Dakota 1159037 1150689 1660148 1417141 1418586 1279134
## 43      T            Tennessee 1811867 1485909 1974179 1157059 1786132 1399191
## 44      T                Texas 1520591 1310777 1957713 1907326 1873544 1655483
## 45      U                 Utah 1771096 1195861 1979395 1241662 1437456 1859416
```

```
## 46      V              Vermont 1146902 1832249 1492704 1579265 1332048 1563537
## 47      V             Virginia 1134317 1163996 1891068 1853855 1708715 1197698
## 48      W           Washington 1977749 1687136 1199490 1163092 1334864 1621989
## 49      W        West Virginia 1677347 1380662 1176100 1888948 1922085 1740826
## 50      W            Wisconsin 1788920 1518578 1289663 1436888 1251678 1721874
## 51      W              Wyoming 1775190 1498098 1198212 1881688 1750527 1523124
##      Y2008   Y2009   Y2010   Y2011   Y2012   Y2013   Y2014   Y2015
## 1  1945229 1944173 1237582 1440756 1186741 1852841 1558906 1916661
## 2  1551826 1436541 1629616 1230866 1512804 1985302 1580394 1979143
## 3  1752886 1554330 1300521 1130709 1907284 1363279 1525866 1647724
## 4  1188104 1628980 1669295 1928238 1216675 1591896 1360959 1329341
## 5  1487315 1663809 1624509 1639670 1921845 1156536 1388461 1644607
## 6  1875146 1752387 1913275 1665877 1491604 1178355 1383978 1330736
## 7  1764457 1972730 1968730 1945524 1228529 1582249 1503156 1718072
## 8  1762096 1553585 1370984 1318669 1984027 1671279 1803169 1627508
## 9  1193245 1739748 1707823 1353449 1979708 1912654 1782169 1410183
## 10 1756185 1818438 1198403 1497051 1131928 1107448 1407784 1170389
## 11 1773090 1630325 1145473 1851245 1850111 1887157 1259353 1725470
## 12 1919423 1928416 1330509 1902816 1695126 1517184 1948108 1150882
## 13 1335481 1748608 1436809 1456340 1643855 1312561 1713718 1757171
## 14 1871645 1658551 1422021 1751422 1696729 1915435 1645465 1583516
## 15 1139551 1883976 1999102 1559924 1905760 1129794 1988394 1467614
## 16 1317983 1150783 1751389 1992996 1501879 1173694 1431705 1641866
## 17 1949478 1561528 1550433 1465812 1882929 1410249 1930090 1385528
## 18 1301848 1956681 1350895 1512894 1916616 1878271 1722762 1913350
## 19 1185085 1124853 1498662 1210385 1234234 1287663 1908602 1403857
## 20 1295877 1969163 1627262 1706080 1437088 1318546 1116792 1529233
## 21 1112765 1967225 1486246 1872327 1175819 1314343 1979529 1569566
## 22 1978904 1567651 1761048 1658538 1482203 1731917 1669749 1963337
## 23 1990431 1575185 1267626 1274673 1709853 1815596 1965196 1646634
## 24 1328133 1890633 1995304 1575533 1910216 1972021 1515366 1864553
## 25 1428291 1568049 1383227 1629132 1988270 1907777 1649668 1991232
## 26 1735099 1800620 1164202 1425363 1800052 1698105 1767835 1996005
## 27 1251742 1592690 1350619 1520064 1185225 1465705 1110394 1125903
## 28 1140598 1270585 1128711 1187207 1569665 1690920 1459243 1802211
## 29 1903270 1231480 1526066 1143343 1980195 1283813 1225348 1903804
## 30 1762936 1763211 1265642 1704297 1131298 1197576 1242623 1963313
## 31 1568034 1357418 1443718 1390010 1202326 1100990 1850165 1183568
## 32 1237704 1820856 1801430 1653384 1475715 1623388 1533494 1868612
## 33 1732098 1426216 1604531 1683687 1500089 1718837 1619033 1367705
## 34 1267737 1116168 1791535 1553750 1472258 1104893 1596452 1229085
## 35 1720352 1671468 1534571 1271132 1430978 1529024 1563898 1604118
## 36 1516621 1511460 1585465 1887714 1227303 1840898 1880804 1573117
## 37 1256746 1853142 1673831 1822933 1674707 1900523 1956742 1307678
## 38 1482786 1862351 1103794 1935687 1905378 1522129 1509171 1893515
## 39 1274168 1571032 1433835 1483292 1290329 1475344 1931500 1668232
## 40 1151409 1993136 1983569 1781016 1909119 1531212 1990412 1611730
## 41 1250499 1864685 1345102 1116203 1532332 1591735 1188417 1110655
## 42 1171870 1852424 1554782 1647245 1811156 1147488 1302834 1136443
## 43 1826406 1326460 1231739 1469785 1849041 1560887 1349173 1162164
## 44 1785986 1827503 1447457 1978374 1882532 1698698 1646508 1705322
## 45 1939284 1915865 1619186 1288285 1108281 1123353 1801019 1729273
## 46 1123567 1618583 1326369 1792600 1714960 1146278 1282790 1565924
## 47 1803330 1590043 1516758 1171686 1262342 1647032 1706707 1850394
## 48 1545621 1555554 1179331 1150089 1775787 1273834 1387428 1377341
## 49 1238174 1539322 1539603 1872519 1462137 1683127 1204344 1198791
## 50 1980167 1901394 1648755 1940943 1729177 1510119 1701650 1846238
## 51 1587602 1504455 1282142 1881814 1673668 1994022 1204029 1853858
```

```
mydata7 = filter(mydata, Index == "A") # filter subset that matches condition in argument
mydata7 = filter(mydata, Index %in% c("A", "C")) # multiple filter criteria
mydata8 = filter(mydata, Index %in% c("A", "C") & Y2002 >= 1300000 ) # using AND in condition
mydata9 = filter(mydata, Index %in% c("A", "C") | Y2002 >= 1300000) # using OR in condition
mydata10 = filter(mydata, !Index %in% c("A", "C")) # using NOT in condition
mydata10 = filter(mydata, grepl("Ar", State)) # search for pattern matches that contains argu
ment
summarise(mydata, Y2015_mean = mean(Y2015), Y2015_med=median(Y2015)) # mean and median
```

```
##   Y2015_mean Y2015_med
## 1    1588297   1627508
```

```
summarise_at(mydata, vars(Y2005, Y2006), list(n=~n(), mean=mean, median=median)) # summarise
 multiples columns
```

```
##   Y2005_n Y2006_n Y2005_mean Y2006_mean Y2005_median Y2006_median
## 1      51      51    1522064    1530969      1480280      1531641
```

```
summarise_at(mydata, vars(Y2011, Y2012),funs(mean, median), na.rm = TRUE) # add additional ar
guments from code above
```

```
## Warning: `funs()` was deprecated in dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
```

```
##   Y2011_mean Y2012_mean Y2011_median Y2012_median
## 1    1574968    1591135      1575533      1643855
```

```
summarise_if(mydata, is.numeric, funs(n(),mean,median)) # summarise if attends to condition
```

```
##   Y2002_n Y2003_n Y2004_n Y2005_n Y2006_n Y2007_n Y2008_n Y2009_n Y2010_n
## 1      51      51      51      51      51      51      51      51      51
##   Y2011_n Y2012_n Y2013_n Y2014_n Y2015_n Y2002_mean Y2003_mean Y2004_mean
## 1      51      51      51      51      51    1566034    1509193    1540555
##   Y2005_mean Y2006_mean Y2007_mean Y2008_mean Y2009_mean Y2010_mean Y2011_mean
## 1    1522064    1530969    1553219    1538398    1658519    1504108    1574968
##   Y2012_mean Y2013_mean Y2014_mean Y2015_mean Y2002_median Y2003_median
## 1    1591135    1530078    1583360    1588297      1584734      1485909
##   Y2004_median Y2005_median Y2006_median Y2007_median Y2008_median Y2009_median
## 1      1522230      1480280      1531641      1563062      1545621      1658551
```

```
##   Y2010_median Y2011_median Y2012_median Y2013_median Y2014_median Y2015_median
## 1      1498662      1575533      1643855      1531212      1580394      1627508
```

```
numdata = mydata[sapply(mydata,is.numeric)] # other method
summarise_all(numdata, funs(n(),mean,median)) # other method
```

```
##   Y2002_n Y2003_n Y2004_n Y2005_n Y2006_n Y2007_n Y2008_n Y2009_n Y2010_n
## 1      51      51      51      51      51      51      51      51      51
##   Y2011_n Y2012_n Y2013_n Y2014_n Y2015_n Y2002_mean Y2003_mean Y2004_mean
## 1      51      51      51      51      51    1566034    1509193    1540555
##   Y2005_mean Y2006_mean Y2007_mean Y2008_mean Y2009_mean Y2010_mean Y2011_mean
## 1    1522064    1530969    1553219    1538398    1658519    1504108    1574968
##   Y2012_mean Y2013_mean Y2014_mean Y2015_mean Y2002_median Y2003_median
## 1    1591135    1530078    1583360    1588297      1584734      1485909
##   Y2004_median Y2005_median Y2006_median Y2007_median Y2008_median Y2009_median
## 1      1522230      1480280      1531641      1563062      1545621      1658551
##   Y2010_median Y2011_median Y2012_median Y2013_median Y2014_median Y2015_median
## 1      1498662      1575533      1643855      1531212      1580394      1627508
```

```
summarise_all(mydata["Index"], funs(nlevels(.), nmiss=sum(is.na(.)))) # check levels and coun
t missing observations
```

```
##   nlevels nmiss
## 1       0     0
```

```
arrange(mydata, Index, Y2011) # sort data by columns
```

```
##    Index                State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007
## 1      A              Arizona 1742027 1968140 1377583 1782199 1102568 1109382
## 2      A               Alaska 1170302 1960378 1818085 1447852 1861639 1465841
## 3      A              Alabama 1296530 1317711 1118631 1492583 1107408 1440134
## 4      A             Arkansas 1485531 1994927 1119299 1947979 1669191 1801213
## 5      C           California 1685349 1675807 1889570 1480280 1735069 1812546
## 6      C             Colorado 1343824 1878473 1886149 1236697 1871471 1814218
## 7      C          Connecticut 1610512 1232844 1181949 1518933 1841266 1976976
## 8      D             Delaware 1330403 1268673 1706751 1403759 1441351 1300836
## 9      D District of Columbia 1111437 1993741 1374643 1827949 1803852 1595981
## 10     F              Florida 1964626 1468852 1419738 1362787 1339608 1278550
## 11     G              Georgia 1929009 1541565 1810773 1779091 1326846 1223770
## 12     H               Hawaii 1461570 1200280 1213993 1245931 1459383 1430465
## 13     I                Idaho 1353210 1438538 1739154 1541015 1122387 1772050
## 14     I              Indiana 1776918 1734104 1269927 1204117 1848073 1129546
## 15     I             Illinois 1508356 1527440 1493029 1261353 1540274 1747614
## 16     I                 Iowa 1499269 1444576 1576367 1388924 1554813 1452911
## 17     K               Kansas 1509054 1290700 1522230 1532094 1104256 1863278
## 18     K              Kentucky 1813878 1448846 1800760 1250524 1137913 1911227
## 19     L            Louisiana 1584734 1110625 1868456 1751920 1233709 1920301
## 20     M              Michigan 1295635 1149931 1601027 1340716 1729449 1567494
## 21     M              Missouri 1221316 1858368 1773451 1573967 1374863 1486197
## 22     M               Montana 1877154 1540099 1332722 1273327 1625721 1983568
## 23     M             Minnesota 1729921 1675204 1903907 1561839 1985692 1148621
## 24     M           Mississippi 1983285 1292558 1631325 1943311 1354579 1731643
```

```
## 25     M        Massachusetts 1647582 1686259 1620601 1777250 1531641 1380529
## 26     M                Maine 1582720 1678622 1208496 1912040 1438549 1330014
## 27     M             Maryland 1579713 1404700 1849798 1397738 1310270 1789128
## 28     N               Nevada 1426117 1114500 1119707 1758830 1694526 1765826
## 29     N             Nebraska 1885081 1309769 1425527 1240465 1500594 1278272
## 30     N         North Dakota 1618807 1510193 1876940 1443172 1425030 1868788
## 31     N           New Jersey 1605532 1141514 1613550 1181452 1541327 1156804
## 32     N       North Carolina 1616742 1292223 1482792 1532347 1158716 1827420
## 33     N           New Mexico 1819239 1226057 1935991 1124400 1723493 1475985
## 34     N             New York 1395149 1611371 1170675 1446810 1426941 1463171
## 35     N        New Hampshire 1419776 1854370 1195119 1990062 1645430 1286967
## 36     O             Oklahoma 1173918 1334639 1663622 1798714 1312574 1708245
## 37     O                 Ohio 1802132 1648498 1441386 1670280 1534888 1314824
## 38     O               Oregon 1794912 1726665 1805445 1133510 1502242 1419251
## 39     P         Pennsylvania 1320191 1446723 1218591 1122030 1971479 1563062
## 40     R         Rhode Island 1501744 1942942 1266657 1961923 1835983 1234040
## 41     S       South Carolina 1631522 1803455 1425193 1458191 1538731 1825195
## 42     S         South Dakota 1159037 1150689 1660148 1417141 1418586 1279134
## 43     T            Tennessee 1811867 1485909 1974179 1157059 1786132 1399191
## 44     T                Texas 1520591 1310777 1957713 1907326 1873544 1655483
## 45     U                 Utah 1771096 1195861 1979395 1241662 1437456 1859416
## 46     V             Virginia 1134317 1163996 1891068 1853855 1708715 1197698
## 47     V              Vermont 1146902 1832249 1492704 1579265 1332048 1563537
## 48     W           Washington 1977749 1687136 1199490 1163092 1334864 1621989
## 49     W        West Virginia 1677347 1380662 1176100 1888948 1922085 1740826
## 50     W              Wyoming 1775190 1498098 1198212 1881688 1750527 1523124
## 51     W            Wisconsin 1788920 1518578 1289663 1436888 1251678 1721874
##      Y2008   Y2009   Y2010   Y2011   Y2012   Y2013   Y2014   Y2015
## 1  1752886 1554330 1300521 1130709 1907284 1363279 1525866 1647724
## 2  1551826 1436541 1629616 1230866 1512804 1985302 1580394 1979143
## 3  1945229 1944173 1237582 1440756 1186741 1852841 1558906 1916661
## 4  1188104 1628980 1669295 1928238 1216675 1591896 1360959 1329341
## 5  1487315 1663809 1624509 1639670 1921845 1156536 1388461 1644607
## 6  1875146 1752387 1913275 1665877 1491604 1178355 1383978 1330736
## 7  1764457 1972730 1968730 1945524 1228529 1582249 1503156 1718072
## 8  1762096 1553585 1370984 1318669 1984027 1671279 1803169 1627508
## 9  1193245 1739748 1707823 1353449 1979708 1912654 1782169 1410183
## 10 1756185 1818438 1198403 1497051 1131928 1107448 1407784 1170389
## 11 1773090 1630325 1145473 1851245 1850111 1887157 1259353 1725470
## 12 1919423 1928416 1330509 1902816 1695126 1517184 1948108 1150882
## 13 1335481 1748608 1436809 1456340 1643855 1312561 1713718 1757171
## 14 1139551 1883976 1999102 1559924 1905760 1129794 1988394 1467614
## 15 1871645 1658551 1422021 1751422 1696729 1915435 1645465 1583516
## 16 1317983 1150783 1751389 1992996 1501879 1173694 1431705 1641866
## 17 1949478 1561528 1550433 1465812 1882929 1410249 1930090 1385528
## 18 1301848 1956681 1350895 1512894 1916616 1878271 1722762 1913350
## 19 1185085 1124853 1498662 1210385 1234234 1287663 1908602 1403857
## 20 1990431 1575185 1267626 1274673 1709853 1815596 1965196 1646634
## 21 1735099 1800620 1164202 1425363 1800052 1698105 1767835 1996005
## 22 1251742 1592690 1350619 1520064 1185225 1465705 1110394 1125903
## 23 1328133 1890633 1995304 1575533 1910216 1972021 1515366 1864553
## 24 1428291 1568049 1383227 1629132 1988270 1907777 1649668 1991232
## 25 1978904 1567651 1761048 1658538 1482203 1731917 1669749 1963337
## 26 1295877 1969163 1627262 1706080 1437088 1318546 1116792 1529233
## 27 1112765 1967225 1486246 1872327 1175819 1314343 1979529 1569566
## 28 1903270 1231480 1526066 1143343 1980195 1283813 1225348 1903804
## 29 1140598 1270585 1128711 1187207 1569665 1690920 1459243 1802211
## 30 1720352 1671468 1534571 1271132 1430978 1529024 1563898 1604118
```

```
## 31 1568034 1357418 1443718 1390010 1202326 1100990 1850165 1183568
## 32 1267737 1116168 1791535 1553750 1472258 1104893 1596452 1229085
## 33 1237704 1820856 1801430 1653384 1475715 1623388 1533494 1868612
## 34 1732098 1426216 1604531 1683687 1500089 1718837 1619033 1367705
## 35 1762936 1763211 1265642 1704297 1131298 1197576 1242623 1963313
## 36 1256746 1853142 1673831 1822933 1674707 1900523 1956742 1307678
## 37 1516621 1511460 1585465 1887714 1227303 1840898 1880804 1573117
## 38 1482786 1862351 1103794 1935687 1905378 1522129 1509171 1893515
## 39 1274168 1571032 1433835 1483292 1290329 1475344 1931500 1668232
## 40 1151409 1993136 1983569 1781016 1909119 1531212 1990412 1611730
## 41 1250499 1864685 1345102 1116203 1532332 1591735 1188417 1110655
## 42 1171870 1852424 1554782 1647245 1811156 1147488 1302834 1136443
## 43 1826406 1326460 1231739 1469785 1849041 1560887 1349173 1162164
## 44 1785986 1827503 1447457 1978374 1882532 1698698 1646508 1705322
## 45 1939284 1915865 1619186 1288285 1108281 1123353 1801019 1729273
## 46 1803330 1590043 1516758 1171686 1262342 1647032 1706707 1850394
## 47 1123567 1618583 1326369 1792600 1714960 1146278 1282790 1565924
## 48 1545621 1555554 1179331 1150089 1775787 1273834 1387428 1377341
## 49 1238174 1539322 1539603 1872519 1462137 1683127 1204344 1198791
## 50 1587602 1504455 1282142 1881814 1673668 1994022 1204029 1853858
## 51 1980167 1901394 1648755 1940943 1729177 1510119 1701650 1846238
```

```
arrange(mydata, desc(Index), Y2011) # sort in descending order
```

```
##    Index                State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007
## 1      W           Washington 1977749 1687136 1199490 1163092 1334864 1621989
## 2      W        West Virginia 1677347 1380662 1176100 1888948 1922085 1740826
## 3      W              Wyoming 1775190 1498098 1198212 1881688 1750527 1523124
## 4      W            Wisconsin 1788920 1518578 1289663 1436888 1251678 1721874
## 5      V             Virginia 1134317 1163996 1891068 1853855 1708715 1197698
## 6      V              Vermont 1146902 1832249 1492704 1579265 1332048 1563537
## 7      U                 Utah 1771096 1195861 1979395 1241662 1437456 1859416
## 8      T            Tennessee 1811867 1485909 1974179 1157059 1786132 1399191
## 9      T                Texas 1520591 1310777 1957713 1907326 1873544 1655483
## 10     S       South Carolina 1631522 1803455 1425193 1458191 1538731 1825195
## 11     S         South Dakota 1159037 1150689 1660148 1417141 1418586 1279134
## 12     R         Rhode Island 1501744 1942942 1266657 1961923 1835983 1234040
## 13     P         Pennsylvania 1320191 1446723 1218591 1122030 1971479 1563062
## 14     O             Oklahoma 1173918 1334639 1663622 1798714 1312574 1708245
## 15     O                 Ohio 1802132 1648498 1441386 1670280 1534888 1314824
## 16     O               Oregon 1794912 1726665 1805445 1133510 1502242 1419251
## 17     N               Nevada 1426117 1114500 1119707 1758830 1694526 1765826
## 18     N             Nebraska 1885081 1309769 1425527 1240465 1500594 1278272
## 19     N         North Dakota 1618807 1510193 1876940 1443172 1425030 1868788
## 20     N           New Jersey 1605532 1141514 1613550 1181452 1541327 1156804
## 21     N       North Carolina 1616742 1292223 1482792 1532347 1158716 1827420
## 22     N           New Mexico 1819239 1226057 1935991 1124400 1723493 1475985
## 23     N             New York 1395149 1611371 1170675 1446810 1426941 1463171
## 24     N        New Hampshire 1419776 1854370 1195119 1990062 1645430 1286967
## 25     M             Michigan 1295635 1149931 1601027 1340716 1729449 1567494
## 26     M             Missouri 1221316 1858368 1773451 1573967 1374863 1486197
## 27     M              Montana 1877154 1540099 1332722 1273327 1625721 1983568
## 28     M            Minnesota 1729921 1675204 1903907 1561839 1985692 1148621
## 29     M          Mississippi 1983285 1292558 1631325 1943311 1354579 1731643
## 30     M        Massachusetts 1647582 1686259 1620601 1777250 1531641 1380529
## 31     M                Maine 1582720 1678622 1208496 1912040 1438549 1330014
```

```
## 32     M             Maryland 1579713 1404700 1849798 1397738 1310270 1789128
## 33     L            Louisiana 1584734 1110625 1868456 1751920 1233709 1920301
## 34     K               Kansas 1509054 1290700 1522230 1532094 1104256 1863278
## 35     K             Kentucky 1813878 1448846 1800760 1250524 1137913 1911227
## 36     I                Idaho 1353210 1438538 1739154 1541015 1122387 1772050
## 37     I              Indiana 1776918 1734104 1269927 1204117 1848073 1129546
## 38     I             Illinois 1508356 1527440 1493029 1261353 1540274 1747614
## 39     I                 Iowa 1499269 1444576 1576367 1388924 1554813 1452911
## 40     H               Hawaii 1461570 1200280 1213993 1245931 1459383 1430465
## 41     G              Georgia 1929009 1541565 1810773 1779091 1326846 1223770
## 42     F              Florida 1964626 1468852 1419738 1362787 1339608 1278550
## 43     D             Delaware 1330403 1268673 1706751 1403759 1441351 1300836
## 44     D District of Columbia 1111437 1993741 1374643 1827949 1803852 1595981
## 45     C           California 1685349 1675807 1889570 1480280 1735069 1812546
## 46     C             Colorado 1343824 1878473 1886149 1236697 1871471 1814218
## 47     C          Connecticut 1610512 1232844 1181949 1518933 1841266 1976976
## 48     A              Arizona 1742027 1968140 1377583 1782199 1102568 1109382
## 49     A               Alaska 1170302 1960378 1818085 1447852 1861639 1465841
## 50     A              Alabama 1296530 1317711 1118631 1492583 1107408 1440134
## 51     A             Arkansas 1485531 1994927 1119299 1947979 1669191 1801213
##      Y2008   Y2009   Y2010   Y2011   Y2012   Y2013   Y2014   Y2015
## 1  1545621 1555554 1179331 1150089 1775787 1273834 1387428 1377341
## 2  1238174 1539322 1539603 1872519 1462137 1683127 1204344 1198791
## 3  1587602 1504455 1282142 1881814 1673668 1994022 1204029 1853858
## 4  1980167 1901394 1648755 1940943 1729177 1510119 1701650 1846238
## 5  1803330 1590043 1516758 1171686 1262342 1647032 1706707 1850394
## 6  1123567 1618583 1326369 1792600 1714960 1146278 1282790 1565924
## 7  1939284 1915865 1619186 1288285 1108281 1123353 1801019 1729273
## 8  1826406 1326460 1231739 1469785 1849041 1560887 1349173 1162164
## 9  1785986 1827503 1447457 1978374 1882532 1698698 1646508 1705322
## 10 1250499 1864685 1345102 1116203 1532332 1591735 1188417 1110655
## 11 1171870 1852424 1554782 1647245 1811156 1147488 1302834 1136443
## 12 1151409 1993136 1983569 1781016 1909119 1531212 1990412 1611730
## 13 1274168 1571032 1433835 1483292 1290329 1475344 1931500 1668232
## 14 1256746 1853142 1673831 1822933 1674707 1900523 1956742 1307678
## 15 1516621 1511460 1585465 1887714 1227303 1840898 1880804 1573117
## 16 1482786 1862351 1103794 1935687 1905378 1522129 1509171 1893515
## 17 1903270 1231480 1526066 1143343 1980195 1283813 1225348 1903804
## 18 1140598 1270585 1128711 1187207 1569665 1690920 1459243 1802211
## 19 1720352 1671468 1534571 1271132 1430978 1529024 1563898 1604118
## 20 1568034 1357418 1443718 1390010 1202326 1100990 1850165 1183568
## 21 1267737 1116168 1791535 1553750 1472258 1104893 1596452 1229085
## 22 1237704 1820856 1801430 1653384 1475715 1623388 1533494 1868612
## 23 1732098 1426216 1604531 1683687 1500089 1718837 1619033 1367705
## 24 1762936 1763211 1265642 1704297 1131298 1197576 1242623 1963313
## 25 1990431 1575185 1267626 1274673 1709853 1815596 1965196 1646634
## 26 1735099 1800620 1164202 1425363 1800052 1698105 1767835 1996005
## 27 1251742 1592690 1350619 1520064 1185225 1465705 1110394 1125903
## 28 1328133 1890633 1995304 1575533 1910216 1972021 1515366 1864553
## 29 1428291 1568049 1383227 1629132 1988270 1907777 1649668 1991232
## 30 1978904 1567651 1761048 1658538 1482203 1731917 1669749 1963337
## 31 1295877 1969163 1627262 1706080 1437088 1318546 1116792 1529233
## 32 1112765 1967225 1486246 1872327 1175819 1314343 1979529 1569566
## 33 1185085 1124853 1498662 1210385 1234234 1287663 1908602 1403857
## 34 1949478 1561528 1550433 1465812 1882929 1410249 1930090 1385528
## 35 1301848 1956681 1350895 1512894 1916616 1878271 1722762 1913350
## 36 1335481 1748608 1436809 1456340 1643855 1312561 1713718 1757171
## 37 1139551 1883976 1999102 1559924 1905760 1129794 1988394 1467614
```

```
## 38 1871645 1658551 1422021 1751422 1696729 1915435 1645465 1583516
## 39 1317983 1150783 1751389 1992996 1501879 1173694 1431705 1641866
## 40 1919423 1928416 1330509 1902816 1695126 1517184 1948108 1150882
## 41 1773090 1630325 1145473 1851245 1850111 1887157 1259353 1725470
## 42 1756185 1818438 1198403 1497051 1131928 1107448 1407784 1170389
## 43 1762096 1553585 1370984 1318669 1984027 1671279 1803169 1627508
## 44 1193245 1739748 1707823 1353449 1979708 1912654 1782169 1410183
## 45 1487315 1663809 1624509 1639670 1921845 1156536 1388461 1644607
## 46 1875146 1752387 1913275 1665877 1491604 1178355 1383978 1330736
## 47 1764457 1972730 1968730 1945524 1228529 1582249 1503156 1718072
## 48 1752886 1554330 1300521 1130709 1907284 1363279 1525866 1647724
## 49 1551826 1436541 1629616 1230866 1512804 1985302 1580394 1979143
## 50 1945229 1944173 1237582 1440756 1186741 1852841 1558906 1916661
## 51 1188104 1628980 1669295 1928238 1216675 1591896 1360959 1329341
```

```
dt = mydata %>% select(Index, State) %>% sample_n(10) # piping example
```

```
t = mydata %>% filter(Index %in% c("A", "C","I")) %>% group_by(Index) %>% do(head( . , 2)) #
 filter data within categorical column
t = mydata %>% group_by(Index)%>% summarise(Mean_2014 = mean(Y2014, na.rm=TRUE), Mean_2015 =
mean(Y2015, na.rm=TRUE)) %>% arrange(desc(Mean_2015)) # summarise, group and sort together
mydata1 = mutate(mydata, change=Y2015/Y2014) # create new column with mutate
mydata12 = mutate_at(mydata, vars(Y2008:Y2010), funs(Rank=min_rank(.))) # ranking columns
```

Estudos de caso:

```
out = mydata %>% group_by(Index) %>% filter(min_rank(desc(Y2015)) == 1) %>% select(Index, Sta
te, Y2015) # select State that generated highest income among the column Index
out2 = mydata %>% group_by(Index) %>% mutate(Total=cumsum(Y2015)) %>% select(Index, Y2015, To
tal) # cumulative income of Index column
```

Fluxo normal:

```
df1 = data.frame(ID = c(1, 2, 3, 4, 5),
w = c('a', 'b', 'c', 'd', 'e'),
x = c(1, 1, 0, 0, 1),
y=rnorm(5),
z=letters[1:5])
df2 = data.frame(ID = c(1, 7, 3, 6, 8),
a = c('z', 'b', 'k', 'd', 'l'),
b = c(1, 2, 3, 0, 4),
c =rnorm(5),
d =letters[2:6])

# INNER JOIN
inner_join(df1, df2, by = c("ID"="ID"))
```

```
##   ID w x         y z a b          c d
## 1  1 a 1  1.972206 a z 1 -0.5240104 b
## 2  3 c 0 -1.307584 c k 3  0.3802837 d
```

```
# LEFT JOIN
left_join(df1, df2, by = "ID")
```

```
##   ID w x          y z    a  b          c    d
## 1  1 a 1  1.9722055 a    z  1 -0.5240104    b
## 2  2 b 1  0.9765914 b <NA> NA         NA <NA>
## 3  3 c 0 -1.3075839 c    k  3  0.3802837    d
## 4  4 d 0 -0.1302736 d <NA> NA         NA <NA>
## 5  5 e 1 -0.3150981 e <NA> NA         NA <NA>
```

```
mtcars$model <- rownames(mtcars)
first <- mtcars[1:20, ]
second <- mtcars[10:32, ]
intersect(first, second) # intersect between data common in both data frames
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## 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
##                                 model
## Merc 280                     Merc 280
## Merc 280C                   Merc 280C
## Merc 450SE                 Merc 450SE
## Merc 450SL                 Merc 450SL
## Merc 450SLC               Merc 450SLC
## Cadillac Fleetwood   Cadillac Fleetwood
## Lincoln Continental Lincoln Continental
## Chrysler Imperial     Chrysler Imperial
## Fiat 128                     Fiat 128
## Honda Civic               Honda Civic
## Toyota Corolla         Toyota Corolla
```

```
x=data.frame(ID = 1:6, ID1= 1:6)
y=data.frame(ID = 1:6,  ID1 = 1:6)
union(x,y)
```

```
##   ID ID1
## 1  1   1
## 2  2   2
## 3  3   3
## 4  4   4
## 5  5   5
## 6  6   6
```

```
union_all(x,y) # apply union
```

```
##    ID ID1
## 1   1   1
## 2   2   2
## 3   3   3
## 4   4   4
## 5   5   5
## 6   6   6
## 7   1   1
## 8   2   2
## 9   3   3
## 10  4   4
## 11  5   5
## 12  6   6
```

```
setdiff(first, second) # rows appears in one df but not in the other
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## 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
##                              model
## Mazda RX4                 Mazda RX4
## Mazda RX4 Wag         Mazda RX4 Wag
## Datsun 710               Datsun 710
## Hornet 4 Drive       Hornet 4 Drive
## Hornet Sportabout Hornet Sportabout
## Valiant                     Valiant
## Duster 360                Duster 360
## Merc 240D                  Merc 240D
## Merc 230                    Merc 230
```

```
df <- c(-10,2, NA)
if_else(df < 0, "negative", "positive", missing = "missing value") # if else with data frame
```

```
## [1] "negative"      "positive"      "missing value"
```

```
# nested if else
mydf =data.frame(x = c(1:5,NA))
mydf %>% mutate(newvar= if_else(is.na(x),"I am missing",
if_else(x==1,"I am one",
if_else(x==2,"I am two",
if_else(x==3,"I am three","Others")))))
```

```
##    x       newvar
## 1  1     I am one
```

```
## 2  2     I am two
## 3  3   I am three
## 4  4       Others
## 5  5       Others
## 6 NA I am missing
```

```
df = mydata %>% rowwise() %>% mutate(Max= max(Y2012,Y2013,Y2014,Y2015)) %>% select(Y2012:Y201
5,Max) # row wise operation
```

```
df1=data.frame(ID = 1:6,  x=letters[1:6])
df2=data.frame(ID = 7:12, x=letters[7:12])
xy = bind_rows(df1,df2) # combine data frames with rows
xy = bind_cols(x,y) # combine data frames with columns
```

```
## New names:
## * ID -> ID...1
## * ID1 -> ID1...2
## * ID -> ID...3
## * ID1 -> ID1...4
```

```
# percentile
mydata %>% group_by(Index) %>%
summarise(Pecentile_25=quantile(Y2015, probs=0.25),
Pecentile_50=quantile(Y2015, probs=0.5),
Pecentile_75=quantile(Y2015, probs=0.75),
Pecentile_99=quantile(Y2015, probs=0.99))
```

```
## # A tibble: 19 x 5
##    Index Pecentile_25 Pecentile_50 Pecentile_75 Pecentile_99
##    <chr>        <dbl>        <dbl>        <dbl>        <dbl>
##  1 A         1568128.     1782192.     1932282.     1977269.
##  2 C         1487672.     1644607      1681340.     1716603.
##  3 D         1464514.     1518846.     1573177.     1625335.
##  4 F         1170389      1170389      1170389      1170389
##  5 G         1725470      1725470      1725470      1725470
##  6 H         1150882      1150882      1150882      1150882
##  7 I         1554540.     1612691      1670692.     1753712.
##  8 K         1517484.     1649439      1781394.     1908072.
##  9 L         1403857      1403857      1403857      1403857
## 10 M         1559483.     1755594.     1970311.     1995671.
## 11 N         1333050      1703164.     1877410      1959147.
## 12 O         1440398.     1573117      1733316      1887107.
## 13 P         1668232      1668232      1668232      1668232
## 14 R         1611730      1611730      1611730      1611730
## 15 S         1117102      1123549      1129996      1136185.
## 16 T         1297954.     1433743      1569532.     1699890.
## 17 U         1729273      1729273      1729273      1729273
## 18 V         1637042.     1708159      1779276.     1847549.
## 19 W         1332704.     1611790.     1848143      1853629.
```

```
# divide data into N bins
x= data.frame(N= 1:10)
```

```
x = mutate(x, pos = ntile(x$N,5))
```

```
mydata2 = select_if(mydata, is.numeric) # select column if it is numeric
mydata3 = select_if(mydata, is.factor) # select factor column
summarise_if(mydata, is.factor, funs(nlevels(.))) # summarise to check number of levels in fa
ctor columns
```

```
## data frame with 0 columns and 1 row
```

```
mydata11 = mutate_if(mydata, is.numeric, funs("new" = .* 1000)) # multiply by 1000 to numeric
columns
k <- c("a", "b", "", "d")
na_if(k, "") # convert value to NA
```

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

```
iris %>% filter(Sepal.Length > 5.5) %>% pull(Species) # pull output from df to vector
```

```
##  [1] setosa     setosa     setosa     versicolor versicolor versicolor
##  [7] versicolor versicolor versicolor versicolor versicolor versicolor
## [13] versicolor versicolor versicolor versicolor versicolor versicolor
## [19] versicolor versicolor versicolor versicolor versicolor versicolor
## [25] versicolor versicolor versicolor versicolor versicolor versicolor
## [31] versicolor versicolor versicolor versicolor versicolor versicolor
## [37] versicolor versicolor versicolor versicolor versicolor versicolor
## [43] virginica  virginica  virginica  virginica  virginica  virginica
## [49] virginica  virginica  virginica  virginica  virginica  virginica
## [55] virginica  virginica  virginica  virginica  virginica  virginica
## [61] virginica  virginica  virginica  virginica  virginica  virginica
## [67] virginica  virginica  virginica  virginica  virginica  virginica
## [73] virginica  virginica  virginica  virginica  virginica  virginica
## [79] virginica  virginica  virginica  virginica  virginica  virginica
## [85] virginica  virginica  virginica  virginica  virginica  virginica
## [91] virginica
## Levels: setosa versicolor virginica
```

```
t = mydata %>% select(Index, Y2015) %>%
  group_by(Index) %>%
  mutate(rank = min_rank(desc(Y2015)))%>%
  arrange(Index, rank) # to use SQL rank over partition by
```

```
mtcars %>%
  group_by(carb) %>%
  summarise(across(mpg:qsec, mean)) # performs same operation across multiple columns
```

```
## # A tibble: 6 x 8
##    carb   mpg   cyl  disp    hp  drat    wt  qsec
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1  25.3  4.57  134.   86   3.68  2.49  19.5
```

```
## 2     2  22.4  5.6   208.  117.  3.70  2.86  18.2
## 3     3  16.3  8     276.  180   3.07  3.86  17.7
## 4     4  15.8  7.2   309.  187   3.60  3.90  17.0
## 5     6  19.7  6     145   175   3.62  2.77  15.5
## 6     8  15    8     301   335   3.54  3.57  14.6
```

```
mtcars %>%
  group_by(carb) %>%
  summarise(across(where(is.numeric), mean))
```

```
## # A tibble: 6 x 11
##    carb   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1  25.3  4.57  134.   86   3.68  2.49  19.5   1   0.571  3.57
## 2     2  22.4  5.6   208.  117.  3.70  2.86  18.2   0.5 0.4    3.8
## 3     3  16.3  8     276.  180   3.07  3.86  17.7   0   0      3
## 4     4  15.8  7.2   309.  187   3.60  3.90  17.0   0.2 0.3    3.6
## 5     6  19.7  6     145   175   3.62  2.77  15.5   0   1      5
## 6     8  15    8     301   335   3.54  3.57  14.6   0   1      5
```

```
mtcars %>%
  group_by(carb) %>%
  summarise(across(mpg:qsec, mean), across(vs:gear, n_distinct)) # multiple across
```

```
## # A tibble: 6 x 11
##    carb   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int> <int> <int>
## 1     1  25.3  4.57  134.   86   3.68  2.49  19.5     1     2     2
## 2     2  22.4  5.6   208.  117.  3.70  2.86  18.2     2     2     3
## 3     3  16.3  8     276.  180   3.07  3.86  17.7     1     1     1
## 4     4  15.8  7.2   309.  187   3.60  3.90  17.0     2     2     3
## 5     6  19.7  6     145   175   3.62  2.77  15.5     1     1     1
## 6     8  15    8     301   335   3.54  3.57  14.6     1     1     1
```

```
mtcars %>%
  group_by(carb) %>%
  mutate(across(where(is.numeric), mean)) # with mutate
```

```
## # A tibble: 32 x 12
## # Groups:   carb [6]
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb model
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1  15.8  7.2   309.  187   3.60  3.90  17.0   0.2 0.3    3.6      4 Mazda RX4
##  2  15.8  7.2   309.  187   3.60  3.90  17.0   0.2 0.3    3.6      4 Mazda RX4 ~
##  3  25.3  4.57  134.   86   3.68  2.49  19.5   1   0.571  3.57     1 Datsun 710
##  4  25.3  4.57  134.   86   3.68  2.49  19.5   1   0.571  3.57     1 Hornet 4 D~
##  5  22.4  5.6   208.  117.  3.70  2.86  18.2   0.5 0.4    3.8      2 Hornet Spo~
##  6  25.3  4.57  134.   86   3.68  2.49  19.5   1   0.571  3.57     1 Valiant
##  7  15.8  7.2   309.  187   3.60  3.90  17.0   0.2 0.3    3.6      4 Duster 360
##  8  22.4  5.6   208.  117.  3.70  2.86  18.2   0.5 0.4    3.8      2 Merc 240D
##  9  22.4  5.6   208.  117.  3.70  2.86  18.2   0.5 0.4    3.8      2 Merc 230
```

```
## 10  15.8  7.2   309.  187   3.60  3.90  17.0   0.2 0.3    3.6      4 Merc 280
## # ... with 22 more rows
```

Exercício 10

> Faça o tutorial sobre o package stringr disponível em https://cran.rstudio.com/web/packages/stringr/vignettes/stringr.html

```
print(str_length("abc")) # finding length of string
```

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

```
x <- c("abcdef", "ghifjk")
print(str_sub(x, 3, 3)) # accessing third character in each string in x
```

```
## [1] "c" "i"
```

```
print(str_sub(x, 2, -2)) # accessing second and second to last character in the strings in x
```

```
## [1] "bcde" "hifj"
```

```
str_sub(x, 3, 3) <- "X" # modifying strings at a given position
print(x)
```

```
## [1] "abXdef" "ghXfjk"
```

```
print(str_dup(x, c(2, 3))) # duplicating first string 2 times and second string 3 times
```

```
## [1] "abXdefabXdef"       "ghXfjkghXfjkghXfjk"
```

```
x <- c("abc", "defghi")
print(str_pad(x, 10)) # pad whitespace on left
```

```
## [1] "       abc" "    defghi"
```

```
print(str_pad(x, 10, "both")) # pad whitespace in both sides
```

```
## [1] "   abc    " "  defghi  "
```

```
print(str_pad(x, 4)) # won't make strin shorter
```

```
## [1] " abc"   "defghi"
```

```
x <- c("Short", "This is a long string")
# ensure strings have same size
x %>%
  str_trunc(10) %>%
  str_pad(10, "right")
```

```
## [1] "Short     " "This is..."
```

```
# ensure the opposite
x <- c("  a   ", "b   ",  "   c")
str_trim(x)
```

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

```
str_trim(x, "left")
```

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

```
jabberwocky <- str_c(
  "`Twas brillig, and the slithy toves ",
  "did gyre and gimble in the wabe: ",
  "All mimsy were the borogoves, ",
  "and the mome raths outgrabe. "
)
cat(str_wrap(jabberwocky, width = 40)) # ensure length of each line of a paragraph is similar as possible
```

```
## `Twas brillig, and the slithy toves did
## gyre and gimble in the wabe: All mimsy
## were the borogoves, and the mome raths
## outgrabe.
```

```
x <- "I like horses."
print(str_to_upper(x)) # upper case
```

```
## [1] "I LIKE HORSES."
```

```
print(str_to_title(x)) # title
```

```
## [1] "I Like Horses."
```

```
print(str_to_lower(x)) # lower case
```

```
## [1] "i like horses."
```

```
print(str_to_lower(x, "tr")) # lower case located to turkish
```

```
## [1] "i like horses."
```

```
x <- c("y", "i", "k")
print(str_order(x)) # order string and get indexes
```

```
## [1] 2 3 1
```

```
print(str_sort(x)) # order string and get elements
```

```
## [1] "i" "k" "y"
```

```
str_sort(x, locale = "lt") # lower case located to lithuanian
```

```
## [1] "i" "y" "k"
```

```
strings <- c(
  "apple",
  "219 733 8965",
  "329-293-8753",
  "Work: 579-499-7527; Home: 543.355.3679"
)
phone <- "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
print(str_detect(strings, phone)) # identify which strings have valid phone numbers
```

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

```
print(str_subset(strings, phone)) # detect if elements of a string matches a regex
```

```
## [1] "219 733 8965"
## [2] "329-293-8753"
## [3] "Work: 579-499-7527; Home: 543.355.3679"
```

```
print(str_count(strings, phone)) # count how many phone numbers have in each string
```

```
## [1] 0 1 1 2
```

```
print(str_locate(strings, phone)) # locate where phone number is located in string
```

```
##      start end
## [1,]    NA  NA
## [2,]     1  12
## [3,]     1  12
## [4,]     7  18
```

```
print(str_locate_all(strings, phone)) # locate all matches
```

```
## [[1]]
##      start end
##
## [[2]]
##      start end
## [1,]     1  12
##
## [[3]]
##      start end
## [1,]     1  12
##
## [[4]]
##      start end
## [1,]     7  18
## [2,]    27  38
```

```
print(str_extract(strings, phone)) # extract phone numbers to first match
```

```
## [1] NA             "219 733 8965" "329-293-8753" "579-499-7527"
```

```
print(str_extract_all(strings, phone)) # extract phone numbers for all matches
```

```
## [[1]]
## character(0)
##
## [[2]]
## [1] "219 733 8965"
##
## [[3]]
## [1] "329-293-8753"
##
## [[4]]
## [1] "579-499-7527" "543.355.3679"
```

```
print(str_extract_all(strings, phone, simplify = TRUE)) # extract phone numbers for all match
es with simplified structure
```

```
##      [,1]           [,2]
## [1,] ""             ""
## [2,] "219 733 8965" ""
## [3,] "329-293-8753" ""
## [4,] "579-499-7527" "543.355.3679"
```

```
print(str_match(strings, phone)) # return column for matches and other columns for subsets of the match
```

```
##      [,1]           [,2]  [,3]  [,4]
## [1,] NA             NA    NA    NA
## [2,] "219 733 8965" "219" "733" "8965"
## [3,] "329-293-8753" "329" "293" "8753"
## [4,] "579-499-7527" "579" "499" "7527"
```

```
print(str_match_all(strings, phone)) # same as before, but extract all matches
```

```
## [[1]]
##      [,1] [,2] [,3] [,4]
##
## [[2]]
##      [,1]           [,2]  [,3]  [,4]
## [1,] "219 733 8965" "219" "733" "8965"
##
## [[3]]
##      [,1]           [,2]  [,3]  [,4]
## [1,] "329-293-8753" "329" "293" "8753"
##
## [[4]]
##      [,1]           [,2]  [,3]  [,4]
## [1,] "579-499-7527" "579" "499" "7527"
## [2,] "543.355.3679" "543" "355" "3679"
```

```
print(str_replace(strings, phone, "XXX-XXX-XXXX")) # replace the occurrences of the first string by the second string for first match
```

```
## [1] "apple"
## [2] "XXX-XXX-XXXX"
## [3] "XXX-XXX-XXXX"
## [4] "Work: XXX-XXX-XXXX; Home: 543.355.3679"
```

```
print(str_replace_all(strings, phone, "XXX-XXX-XXXX")) # same as before but replaces all matches
```

```
## [1] "apple"
## [2] "XXX-XXX-XXXX"
## [3] "XXX-XXX-XXXX"
## [4] "Work: XXX-XXX-XXXX; Home: XXX-XXX-XXXX"
```

```
print(str_split("a-b-c", "-")) # split string into fixed number of pieces
```

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

```r
print(str_split_fixed("a-b-c", "-", n = 2)) # same as before, but to variable number of piece
s
```

```
##      [,1] [,2]
## [1,] "a"  "b-c"
```

```r
# since there are mutiples ways to describe a char...
a1 <- "\u00e1"
a2 <- "a\u0301"
print(c(a1, a2)) # this shows the same char, but with different codification
```

```
## [1] "á" "á"
```

```r
print(a1 == a2) # thus they are not equal
```

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

```r
print(str_detect(a1, fixed(a2)))
```

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

```r
print(str_detect(a1, coll(a2))) # using coll allow to compare as a human would do
```

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

```r
i <- c("I", "İ", "i", "ı")
print(i)
```

```
## [1] "I" "I" "i" "i"
```

```r
print(str_subset(i, coll("i", ignore_case = TRUE))) # fetch match using human-language collat
ion rules
```

```
## [1] "I" "I" "i" "i"
```

```r
print(str_subset(i, coll("i", ignore_case = TRUE, locale = "tr"))) # same as before for turki
sh language
```

```
## [1] "i" "i"
```

```r
x <- "This is a sentence."
print(str_split(x, boundary("word"))) # fetch boundaries between words and split them
```

```
## [[1]]
## [1] "This"     "is"       "a"        "sentence"
```

```
print(str_count(x, boundary("word"))) # print how many words were separated
```

```
## [1] 4
```

```
print(str_extract_all(x, boundary("word"))) # extract all matches
```

```
## [[1]]
## [1] "This"     "is"       "a"        "sentence"
```

```
print(str_split(x, "")) # the boundary character can be alternated
```

```
## [[1]]
##  [1] "T" "h" "i" "s" " " "i" "s" " " "a" " " "s" "e" "n" "t" "e" "n" "c" "e" "."
```

```
print(str_count(x, "")) # this results in the string beeing separated in every char
```

```
## [1] 19
```

Exercício 11

> Faça novamente o exercício 8 da Parte 2, utilizando a biblioteca dplyr

Já foi feito anteriormente.