

# Introduction to R Programming

## Data Frames

Pedro Fonseca

01 Maio 2020

## Before we start

The `ifelse()` function performs vectorized if-then-else statements.

```
ifelse(test, x, y)
```

- ▶ `test` can be anything that returns a logical vector or matrix
- ▶ `ifelse()` returns `x` in the positions where the corresponding element of `test` is `TRUE` and `y` in the positions where the corresponding element of `test` is `FALSE`

## Before we start

```
vec <- 1:5
```

```
ifelse(vec > 2, 2, 1)
```

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

```
ifelse(vec > 2, "large", "small")
```

```
## [1] "small" "small" "large" "large" "large"
```

```
ifelse(vec > 2, vec - 1, vec + 1)
```

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

## Before we start

You can chain multiple `ifelse()` calls:

```
heights <- c(1.66, 1.88, 1.76, 1.68, 1.7, 1.9)

shirts <- ifelse(heights >= 1.9, "XL",
                 ifelse(heights > 1.8, "L",
                         ifelse(heights > 1.75, "M",
                                "S")))

shirts

## [1] "S" "L" "M" "S" "S" "XL"
```

# Datasets

In a typical dataset:

- ▶ Rows represent observations
- ▶ Columns represent variables

	age	smoker	marital
1	20	TRUE	M
2	38	FALSE	S
3	50	FALSE	W
4	19	TRUE	S
5	15	FALSE	S

Figure 1: A rectangular dataset

# What is a data frame?

Data frames are designed specifically for rectangular datasets.

- ▶ Why not matrices? We may need different data types for different columns.
- ▶ Why not lists? Not very practical nor visually appealing.

## My first data frame

You can create a data frame by supplying name-vector pairs to `data.frame()`:

```
df <- data.frame(  
  age = c(20, 38, 50, 19),  
  smoker = c(TRUE, FALSE, FALSE, FALSE),  
  marital = factor(c("M", "S", "W", "S"))  
)  
df
```

```
##   age smoker marital  
## 1  20   TRUE      M  
## 2  38  FALSE      S  
## 3  50  FALSE      W  
## 4  19  FALSE      S
```

Data frame rows are numbered by default.

## Some utilities

```
summary(df)
```

##	age	smoker	marital
##	Min. :19.00	Mode :logical	M:1
##	1st Qu.:19.75	FALSE:3	S:2
##	Median :29.00	TRUE :1	W:1
##	Mean :31.75		
##	3rd Qu.:41.00		
##	Max. :50.00		

Try this:

```
View(df)
```



# What is a data frame?

```
str(df)
```

```
## 'data.frame':    4 obs. of  3 variables:  
## $ age      : num  20 38 50 19  
## $ smoker   : logi  TRUE FALSE FALSE FALSE  
## $ marital: Factor w/ 3 levels "M","S","W": 1 2 3 2
```

- ▶ Similar to the structure of a list?
- ▶ Actually, data frames are build on top of lists.
- ▶ A data frame is a list of atomic vectors with equal length.

# What is a data frame?

A data frame is a named list of vectors with attributes for column names, row.names, and its own class:

```
typeof(df)
```

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

```
class(df)
```

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

# What is a data frame?

Due to their rectangular structure, data frames share the properties of both matrices and lists:

- ▶ A data frame has `rownames()` and `colnames()`. The `names()` of a data frame are the column names.
- ▶ A data frame has `nrow()` rows and `ncol()` columns. The `length()` of a data frame is the number of columns.

## My second data frame

```
name <- c("Anne", "Pete", "Frank", "Julia", "Cath")
age <- c(28, 30, 21, 39, 35)
child <- c(FALSE, TRUE, TRUE, FALSE, TRUE)

df2 <- data.frame(name, age, child)
df2
```

```
##      name age child
## 1  Anne  28 FALSE
## 2  Pete  30  TRUE
## 3 Frank  21  TRUE
## 4 Julia  39 FALSE
## 5  Cath  35  TRUE
```

Unlike `list()`, `data.frame()` keeps the names of the vectors.

## Assign names() to a data frame

Data frame columns always have names. If you don't provide them, default names will be assigned:

```
df3 <- data.frame(  
  c(1, 4, 6),  
  c(6, 7, 9)  
)
```

```
df3
```

```
##      c.1..4..6. c.6..7..9.  
## 1             1           6  
## 2             4           7  
## 3             6           9
```

## Assign names() to a data frame

```
names(df3) <- c("a-name", "some-other-name")  
df3
```

```
##   a-name some-other-name  
## 1      1                6  
## 2      4                7  
## 3      6                9
```

## Row names

`data.frame()` allows you to label row indexes. You can provide a vector of unique values to the `row.names` argument:

```
df4 <- data.frame(  
  age = c(35, 27, 18),  
  hair = c("blond", "brown", "black"),  
  row.names = c("Bob", "Susan", "Sam") # row names!  
)
```

df4

```
##      age  hair  
## Bob    35 blond  
## Susan  27 brown  
## Sam    18 black
```

```
rownames(df4)
```

```
## [1] "Bob"    "Susan"  "Sam"
```

## Row names

You can also set (or overwrite) row names after the data frame is created:

```
rownames(df4) <- c("Bob M.", "Susan B.", "Sam L.")  
df4
```

```
##           age  hair  
## Bob M.     35 blond  
## Susan B.   27 brown  
## Sam L.     18 black
```



## stringsAsFactors

```
df5 <- data.frame(  
  person = c("John", "Matt", "Mary"),  
  age = c(20, 38, 50),  
  smoker = c(TRUE, FALSE, FALSE),  
  marital = factor(c("M", "S", "S"))  
)  
  
str(df5)
```

```
## 'data.frame':    3 obs. of  4 variables:  
## $ person : Factor w/ 3 levels "John","Mary",...: 1 3 2  
## $ age      : num  20 38 50  
## $ smoker   : logi  TRUE FALSE FALSE  
## $ marital  : Factor w/ 2 levels "M","S": 1 2 2
```

The person variable is a factor! Why?

## stringsAsFactors

The `stringsAsFactors` argument of `data.frame` is set to `TRUE` by default.

```
df6 <- data.frame(  
  person = c("John", "Matt", "Mary"),  
  age = c(20, 38, 50),  
  smoker = c(TRUE, FALSE, FALSE),  
  marital = factor(c("M", "S", "S")),  
  stringsAsFactors = FALSE  
)  
str(df6)
```

```
## 'data.frame':    3 obs. of  4 variables:  
## $ person : chr  "John" "Matt" "Mary"  
## $ age : num  20 38 50  
## $ smoker : logi  TRUE FALSE FALSE  
## $ marital: Factor w/ 2 levels "M","S": 1 2 2
```

## Testing and coercing

To check if an object is a data frame use `is.data.frame()`:

```
is.data.frame(df6)
```

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

You can coerce an object to a data frame with `as.data.frame()`:

```
M <- matrix(1:9, ncol = 3)
as.data.frame(M)
```

```
##   V1 V2 V3
## 1  1  4  7
## 2  2  5  8
## 3  3  6  9
```

# Subsetting

Data frames inherit subsetting syntax from both matrices and lists.  
You can subset a data frame:

- ▶ Like a 2D structure (behaves like a matrix)
- ▶ Like a 1D structure (behaves like a list)

## Subsetting

```
name <- c("Anne", "Pete", "Frank", "Julia", "Cath")
age <- c(28, 30, 21, 39, NA)
child <- c(FALSE, FALSE, TRUE, FALSE, TRUE)

people <- data.frame(name, age, child,
                     stringsAsFactors = FALSE)

people
```

```
##      name age child
## 1  Anne  28 FALSE
## 2  Pete  30 FALSE
## 3 Frank  21  TRUE
## 4 Julia  39 FALSE
## 5  Cath  NA  TRUE
```

## Subsetting with matrix syntax

```
people[3, ] # returns a data frame
```

```
##      name age child  
## 3 Frank  21  TRUE
```

```
people[c(1, 3), ] # data frame
```

```
##      name age child  
## 1  Anne  28 FALSE  
## 3 Frank  21  TRUE
```

```
people[3, 2] # vector
```

```
## [1] 21
```

```
people[3, "age"] # vector
```

```
## [1] 21
```

## Subsetting with matrix syntax

```
people[, "age"] # vector
```

```
## [1] 28 30 21 39 NA
```

```
people[, 1] # vector
```

```
## [1] "Anne" "Pete" "Frank" "Julia" "Cath"
```

```
people[c(3, 5), c("age", "child")] # data frame
```

```
##   age child
```

```
## 3  21  TRUE
```

```
## 5  NA  TRUE
```

## Subsetting with matrix syntax

See the pattern?

- ▶ When you subset a data frame like a 2D object with `[]` you get a vector if only one column is extracted. You get a data frame otherwise.
- ▶ This is a frequent source of bugs when using `[]` in a function, unless you always remember to use `drop = FALSE`.



## Subsetting with matrix syntax

```
people[, "age", drop = FALSE] # data frame!
```

```
##    age  
## 1  28  
## 2  30  
## 3  21  
## 4  39  
## 5  NA
```

Setting `drop = FALSE` prevents [ from coercing to a lower dimension.

## Subsetting with matrix syntax

You can also subset using logical conditions.

Show name and child for the people with known age:

```
people[!is.na(people[, "age"]), c("name", "child")]
```

```
##      name child
## 1  Anne FALSE
## 2   Pete FALSE
## 3  Frank  TRUE
## 4  Julia FALSE
```

## Subsetting with matrix syntax

Show name and child for the people who are older than 21:

```
people[!is.na(people[, "age"]) & people[, "age"] > 21,  
       c("name", "child")]
```

```
##      name child  
## 1  Anne FALSE  
## 2   Pete FALSE  
## 4  Julia FALSE
```

## Subsetting with matrix syntax

If available, row names can be used for subsetting:

```
df4
```

```
##           age  hair
## Bob M.     35 blond
## Susan B.   27 brown
## Sam L.     18 black
```

```
df4["Bob M.", ]
```

```
##           age  hair
## Bob M.     35 blond
```

```
df4["Bob M.", "hair"]
```

```
## [1] blond
## Levels: black blond brown
```

## Subsetting with list syntax

```
people[2] # data frame
```

```
##    age  
## 1  28  
## 2  30  
## 3  21  
## 4  39  
## 5  NA
```

```
people["age"] # data frame
```

```
##    age  
## 1  28  
## 2  30  
## 3  21  
## 4  39  
## 5  NA
```

## Subsetting with list syntax

```
people[c(1, 3)] # data frame
```

```
##      name child
## 1  Anne FALSE
## 2  Pete FALSE
## 3 Frank  TRUE
## 4 Julia FALSE
## 5  Cath  TRUE
```

```
people[c("name", "child")] # data frame
```

```
##      name child
## 1  Anne FALSE
## 2  Pete FALSE
## 3 Frank  TRUE
## 4 Julia FALSE
## 5  Cath  TRUE
```

## Subsetting with list syntax

```
people["age"] # data frame
```

```
##    age  
## 1  28  
## 2  30  
## 3  21  
## 4  39  
## 5  NA
```

```
people[["age"]] # vector
```

```
## [1] 28 30 21 39 NA
```

```
people$age # vector
```

```
## [1] 28 30 21 39 NA
```

## Subsetting with list syntax

```
people[["age"]][1]
```

```
## [1] 28
```

```
people[["age"]][c(1, 4)]
```

```
## [1] 28 39
```

```
people$age[1]
```

```
## [1] 28
```

```
people$age[c(1, 4)]
```

```
## [1] 28 39
```



## Subsetting with list syntax

```
people[[2]] # same as people[["age"]]
```

```
## [1] 28 30 21 39 NA
```

```
people[[2]][1]
```

```
## [1] 28
```

```
people[[2]][c(1, 4)]
```

```
## [1] 28 39
```

## Subsetting with list syntax

Since data frames are lists of vectors, when you subset a data frame like a 1D object:

- ▶ with `[` you get a data frame.
- ▶ with `[[` or `$` you get a vector.

## Partial matching with \$

When you attempt to extract a column from `df` with `df$name` and there is no column `name`, a data frame will instead select any variable that starts with `name`. If no variable starts with `name` (or if more than one do), `df$name` will return `NULL`.

```
names(df)
```

```
## [1] "age"      "smoker"   "marital"
```

```
df$a
```

```
## [1] 20 38 50 19
```

By default, `[[` does not do partial matching. Why? Run `?"["` and check what is the default value of the `exact` argument.

`df$name` is equivalent to `df[["name", exact = FALSE]]`

## Add columns with \$

So far our data frame `people` has 5 rows and 3 columns.

```
str(people)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ name : chr  "Anne" "Pete" "Frank" "Julia" ...
##  $ age  : num  28 30 21 39 NA
##  $ child: logi  FALSE FALSE TRUE FALSE TRUE
```

## Add columns with \$

Let's add a column with the people's height:

```
height <- c(175, 170, 166, 182, 172)
```

```
people$height <- height
```

```
str(people)
```

```
## 'data.frame':    5 obs. of  4 variables:
```

```
## $ name   : chr  "Anne" "Pete" "Frank" "Julia" ...
```

```
## $ age    : num  28 30 21 39 NA
```

```
## $ child  : logi  FALSE FALSE TRUE FALSE TRUE
```

```
## $ height: num  175 170 166 182 172
```

## Add columns with cbind()

Now let's add a column with the people's weight:

```
weight <- c(86, 63, 68, 55, 56)
people <- cbind(people, weight)
```

```
str(people)
```

```
## 'data.frame':    5 obs. of  5 variables:
## $ name   : chr   "Anne" "Pete" "Frank" "Julia" ...
## $ age    : num   28 30 21 39 NA
## $ child  : logi  FALSE FALSE TRUE FALSE TRUE
## $ height: num   175 170 166 182 172
## $ weight: num    86 63 68 55 56
```

## Modify columns

We can overwrite existing columns.

Lets convert height from centimeters to meters:

```
people$height <- people$height/100
```

```
people
```

##	name	age	child	height	weight
## 1	Anne	28	FALSE	1.75	86
## 2	Pete	30	FALSE	1.70	63
## 3	Frank	21	TRUE	1.66	68
## 4	Julia	39	FALSE	1.82	55
## 5	Cath	NA	TRUE	1.72	56

## Calculations with existing columns

We can create new columns using the values of existing columns.

Let's use `weight` and `height` to calculate the body mass index:

```
people$bmi <- round(  
  people$weight/((people$height)^2)  
)  
people
```

	##	name	age	child	height	weight	bmi
##	1	Anne	28	FALSE	1.75	86	28
##	2	Pete	30	FALSE	1.70	63	22
##	3	Frank	21	TRUE	1.66	68	25
##	4	Julia	39	FALSE	1.82	55	17
##	5	Cath	NA	TRUE	1.72	56	19



## Calculations with existing columns

Now let's use the body mass index to calculate the weight status:

```
people$weight_status <- ifelse(  
  people$bmi < 18.5, "underweight",  
    ifelse(people$bmi < 24.9, "normal",  
      ifelse(people$bmi < 29.9, "overweight",  
        "obese")  
    )  
  )  
)
```

## Calculations with existing columns

```
people
```

##	name	age	child	height	weight	bmi	weight_status
## 1	Anne	28	FALSE	1.75	86	28	overweight
## 2	Pete	30	FALSE	1.70	63	22	normal
## 3	Frank	21	TRUE	1.66	68	25	overweight
## 4	Julia	39	FALSE	1.82	55	17	underweight
## 5	Cath	NA	TRUE	1.72	56	19	normal

## Drop columns

Lets drop the bmi and weight\_status columns. By name:

```
people$bmi <- NULL  
people$weight_status <- NULL
```

Lets drop the weight column. By position:

```
people <- people[, -5]  
people
```

```
##      name age child height  
## 1  Anne  28 FALSE   1.75  
## 2  Pete  30 FALSE   1.70  
## 3 Frank  21  TRUE   1.66  
## 4 Julia  39 FALSE   1.82  
## 5  Cath  NA  TRUE   1.72
```

## Add a row

```
tom <- data.frame("Tom", 37, FALSE, 1.83)
```

```
rbind(people, tom)
```

```
## Error in match.names(clabs, names(xi)) : names do  
## not match previous names
```

## Add a row

The default names do not match the names of the people data frame:

```
names(tom)
```

```
## [1] "X.Tom." "X37"    "FALSE." "X1.83"
```

```
names(people)
```

```
## [1] "name"    "age"     "child"   "height"
```

## Add a row

```
tom <- data.frame(name = "Tom", age = 37,  
                  child = FALSE, height = 1.83)  
rbind(people, tom)
```

```
##   name age child height  
## 1  Anne  28 FALSE   1.75  
## 2  Pete  30 FALSE   1.70  
## 3 Frank  21  TRUE   1.66  
## 4 Julia  39 FALSE   1.82  
## 5  Cath  NA  TRUE   1.72  
## 6   Tom  37 FALSE   1.83
```

## Add a row

You can also add rows using vectors, as long as they have appropriate names and the correct length:

```
new_obs <- t(c(name = "Peter", age = 30,  
               child = FALSE, height = 1.86))
```

```
rbind(people, new_obs)
```

```
##      name  age child height  
## 1  Anne   28 FALSE   1.75  
## 2  Pete   30 FALSE    1.7  
## 3 Frank   21  TRUE   1.66  
## 4 Julia   39 FALSE   1.82  
## 5 Cath <NA>  TRUE   1.72  
## 6 Peter   30 FALSE   1.86
```

## Combining matrix and list syntax

Show the name and age of the people without children:

```
people[people$child == FALSE, c("name", "age")]
```

```
##      name age  
## 1  Anne  28  
## 2   Pete  30  
## 4 Julia  39
```



## Combining matrix and list syntax

Show the name and age of the people without children who are taller than 1.70:

```
people[people$child == FALSE & people$height > 1.70,  
       c("name", "age")]
```

```
##      name age  
## 1  Anne  28  
## 4 Julia  39
```

# Sorting

```
people$age
```

```
## [1] 28 30 21 39 NA
```

```
sort(people$age)
```

```
## [1] 21 28 30 39
```

```
ranks <- order(people$age)  
ranks
```

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

```
people$age[ranks]
```

```
## [1] 21 28 30 39 NA
```

# Sorting

Sort the rows by ascending age:

```
people[ranks, ]
```

```
##      name age child height
## 3 Frank  21  TRUE   1.66
## 1  Anne  28 FALSE   1.75
## 2  Pete  30 FALSE   1.70
## 4 Julia  39 FALSE   1.82
## 5  Cath  NA  TRUE   1.72
```

# Sorting

Sort the rows by descending age:

```
people[order(-people$age), ]
```

```
##      name age child height
## 4 Julia  39 FALSE   1.82
## 2 Pete   30 FALSE   1.70
## 1 Anne   28 FALSE   1.75
## 3 Frank  21  TRUE   1.66
## 5 Cath   NA  TRUE   1.72
```

# Sorting

Sort the rows by descending height:

```
people[order(people$height, decreasing = TRUE), ]
```

```
##      name age child height
## 4 Julia  39 FALSE   1.82
## 1  Anne  28 FALSE   1.75
## 5  Cath  NA  TRUE   1.72
## 2  Pete  30 FALSE   1.70
## 3 Frank  21  TRUE   1.66
```

# Sorting

Sort the rows by name:

```
people[order(people$name), ]
```

```
##      name age child height
## 1  Anne  28 FALSE   1.75
## 5  Cath  NA  TRUE   1.72
## 3 Frank  21  TRUE   1.66
## 4 Julia  39 FALSE   1.82
## 2  Pete  30 FALSE   1.70
```