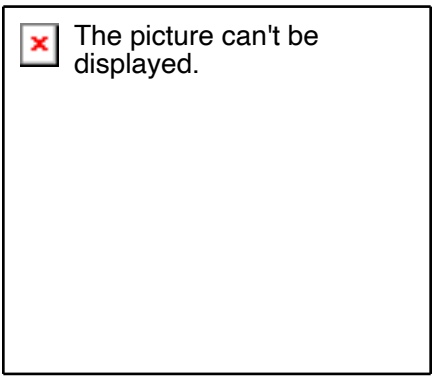


DATA TYPES EN R

Dept. Ciencias de la Computación e Inteligencia Artificial,
Universidad de Granada

Factors vectores para valores categóricos

It is common to have *categorical data* in statistical data analysis (e.g. Male/ Female). In R such variables are referred to as factors. This makes it possible to assign meaningful names to categories.

```
Pain <- c(0,3,2,2,1)

SevPain <- factor(c(0,3,2,2,1),
levels=c(0,1,2,3),labels=c("none","mild","medium","severe"))

> SevPain
[1] none   severe medium medium mild
Levels: none mild medium severe
```

Factors a special vector to work with categories

A factor has a set of levels and labels and this can be confusing.

Levels: refers to the input values

Labels: refers to the output values of the new factor.

```
Pain <- c(0,3,2,2,1)

SevPain <- factor(c(0,3,2,2,1),
levels=c(0,1,2,3),labels=c("none","mild","medium","severe"))

> SevPain
[1] none   severe medium medium mild
Levels: none mild medium severe
```

Factors a special vector to work with categories

The levels of the new factor does not contain the value **“West”** However, it makes sense to have all possible levels of your factor.

To add the missing level specify the **levels** arguments of **factor**:

```
> directions <- c("North", "East", "South", "South")

> factor(directions)
[1] North East  South South
Levels: East North South

> factor(directions, levels= c("North", "East", "South",
"West"),labels=c("N","E","S","W”))
[1] N E S S
Levels: N E S W
```

Summarizing factors

In factors values are repeated and it is interesting to have summarized information. `table()`

```
> head(state.region)
[1] South West  West  South West  West
Levels: Northeast South North Central West

> table(state.region)
state.region
    Northeast         South North Central          West
            9            16            12            13
```

Working with ordered factors

Sometimes categorical data has some kind of order. An example:

- Project status is described as low, medium, or high.
- A traffic light that can be red, yellow, or green.
- A gene underexpressed or overexpressed

The name for this type of data, where rank ordering is important is **ordinal data**. In R, there is a special data type for ordinal data called **ordered factors**

Working with ordered factors

```
> status <- c("Lo", "Hi", "Med", "Med", "Hi")
>ordered.status <- factor(status, levels=c("Lo", "Med",
"Hi"),ordered=TRUE)

> ordered.status
[1] Lo  Hi  Med Med Hi
Levels: Lo < Med < Hi

> table(status)
status
 Hi  Lo Med
  2   1   2

> table(ordered.status)
ordered.status
 Lo Med  Hi
  1   2   2
```

Dataframes: combining different types of values

- A data frame is a generalized matrix, where different columns can have different modes (numeric, character, factor, etc.).

- For example vectors and/or factors of the same length that are related "across", such that data in the same position come from the same experimental unit (subject, animal, etc).

Dataframes

The function data.frame() allows to create one from scratch

```
> S<-as.factor(c("F","M","M","F"))

> Patients <- data.frame(age=c(31,32,40,50),sex=S)

> Patients
  age sex
1  31   F
2  32   M
3  40   M
4  50   F
```

Creating a Dataframe from a matrix

- To create a data frame from a matrix use the function as.data.frame()

```
> m<-matrix(1:12, ncol=4, byrow=TRUE)

     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12

> m.df<-as.data.frame(m)
➢ m.df<-as.data.frame(t(m))
 V1 V2 V3
1  1  5  9
2  2  6 10
3  3  7 11
4  4  8 12
```

Creating a Dataframe from vectors

- To create a data frame from vectors use the function data.frame()

```
> employee <- c("John Doe","Peter Gynn","Jolie Hope")
> salary <- c(21000, 23400, 26800)
> startdate <- as.Date(c("2010-11-1","2008-3-25","2007-3-
14"))

> employ.data <- data.frame(employee, salary, startdate)
> str(employ.data)
'data.frame':  3 obs. of  3 variables:
 $ employee : Factor w/ 3 levels "John Doe","Jolie Hope",..:
1 3 2
 $ salary   : num  21000 23400 26800
 $ startdate: Date, format: "2010-11-01" "2008-03-25" "2007-
03-14"
```

Dataframe: keeping character as char

- The original vector employee was a character vector but R converted it in a factor the data frame

```
> str(employ.data)
```

```
> str(employ.data)
data.frame':   3 obs. of  3 variables:
 $ employee : chr  "John Doe" "Peter Gynn" "Jolie Hope"
 $ salary   : num  21000 23400 26800
 $ startdate: Date, format: "2010-11-01" "2008-03-25" "2007-
03-14"
```

```
> employ.data <- data.frame(employee, salary, startdate,
+                           stringsAsFactors=FALSE)
```

Looking at a Dataframe

- Structure: str()
- Number of variables: ncols() and length()
- Number of observations: nrow()

```
> m.df<-as.data.frame(t(m))
 V1 V2 V3
1  1  5  9
2  2  6 10
3  3  7 11
4  4  8 12
> str(m.df)
'data.frame':  4 obs. of  3 variables:
 $ V1: int  1 2 3 4
 $ V2: int  5 6 7 8
 $ V3: int  9 10 11 12
> ncol(m.df)
[1] 3
> length(m.df)
[1] 3
> nrow(m.df)
[1] 4
```

Data frames

```
# Get the structure of the data frame.
> str(emp.data)
'data.frame': 5 obs. of 4 variables:
$ emp_id : int 1 2 3 4 5
$ emp_name : chr "Rick" "Dan" "Michelle" "Ryan" ...
$ salary : num 623 515 611 729 843

# Get the statistical summary of the data with  summary()
>summary(emp.data)
emp_id         emp_name            salary
Min. :1        Length:5            Min. :515.2
1st Qu.:2      Class :character    1st Qu.:611.0
Median :3      Mode :character     Median :623.3
Mean :3                            Mean :664.4
3rd Qu.:4                          3rd Qu.:729.0 3rd
Max. :5                            Max. :843.2
```

Indexing a data frame

- A data frame is a generalized matrix and work as such for data indexing

```
> S<-as.factor(c("F","M","M","F"))

> Patients <- data.frame(age=c(31,32,40,50),sex=S)

> Patients
  age sex
1  31   F
2  32   M
3  40   M
4  50   F

> Patients[1,]
  Age gender
1 31      F

> Patients[2,]
  Age gender
2  32      M
```

Accessing a data frame

- When looking at the result of str() we see that variables are preceded by a $ sign

```
> str(Patients)
'data.frame':    4 obs. of  2 variables:
 $ age: num  31 32 40 50
 $ sex: Factor w/ 2 levels "F","M": 1 2 2 1

> Patients$age
[1] 31 32 40 50

> Patients$sex
[1] F M M F
Levels: F M
```

Adding rows

```
# Add a new row
> rbind(Patients,c(60,"F"))
  age sex
1  31   F
2  32   M
3  40   M
4  50   F
5  60   F
```

Remember: The two data frames must have the same variables. If dataframe1 has variables that dataframe2 does not have, do one of the following things before joining:

. Delete the extra variables in dataframe1

. Create the additional variables in dataframe2 with value NA (missing)

Lists

- Lists can be used to combine objects (of possibly different kinds/sizes) into a larger composite object.
- The components of the list are named according to the arguments used.
- Components can be extracted with the double bracket operator [[ ]]
- Alternatively, named components can be accessed with the "$" separator.

```
Create a list containing strings, numbers, vectors and a logical
values.

list_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)
print(list_data)
[[1]]
[1] "Red"
[[2]]
[1] "Green”
[[3]]
[1] 21 32 11
[[4]]
[1] TRUE …..
```

Giving names to Lists elements

- Lists can be used to combine objects (of possibly different kinds/sizes) into a larger composite object.
- The components of the list are named according to the arguments used.

```
Create a list containing strings, numbers, vectors and a logical
values.

list_data <- list("Red", "Green", c(21,32,11), 51.23, 119.1)
names (list_data)<-c("Colors", "Age", "Time")
list_data
$Colors
[1] "Red"   "Green"

$Age
[1] 21 32 11

$Time
[1] 51.23
```

Accessing Lists elements

- Components can be extracted with the double bracket operator `[[ ]]`
- Alternatively, named components can be accessed with the "$" separator.

```
# Access the first element of the list.

print(list_data[1])
$Colors
[1] "Red"   "Green"

# Access the list element using the name of the
element.

print(list_data$Colors)
[1] "Red"   "Green"
```

Convert Lists to vectors (unlist)

- A list can be converted to a vector so that the elements of the vector can be used for further manipulation.
- All the arithmetic operations on vectors can be applied after the list is converted into vectors.

```
# Create lists.
list1 <- list(1:5)
print(list1)
[[1]]
[1] 1 2 3 4 5

list2 <-list(10:14)
print(list2)
[[1]]
[1] 10 11 12 13 14
```

```
# Convert the lists to vectors.
v1 <- unlist(list1)
[1] 1 2 3 4 5

v2 <- unlist(list2)
 [1] 1 2 3 4 5

# Now add the vectors
result <- v1+v2
print(result)
[1] 11 13 15 17 19
```

Predefined Lists

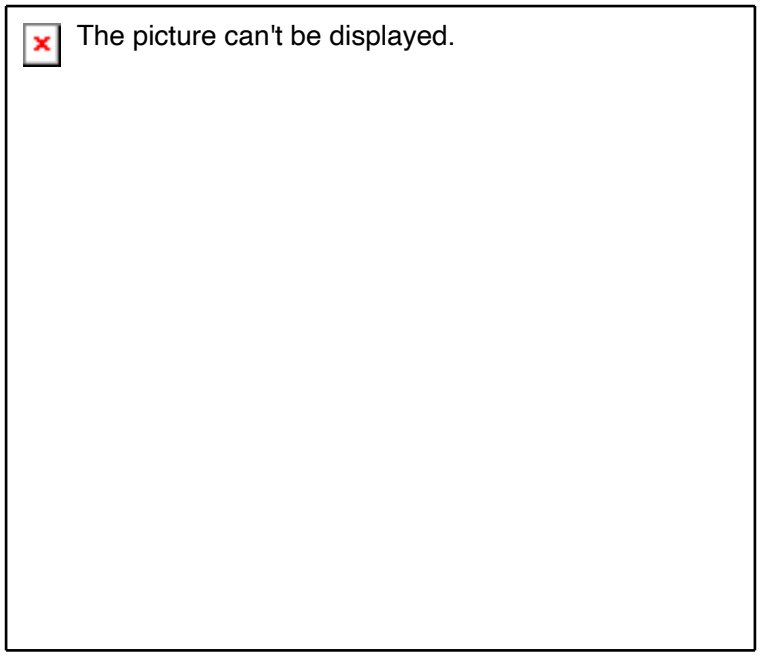
```
#Author
letters
LETTERS
month.abb
month.name
```

Merging Lists

```
num_list <- list(1,2,3,4,5)
day_list <- list("Mon","Tue","Wed", "Thurs", "Fri")

merge_list <- c(num_list, day_list)
merge_list
```

Gracias…



Adding columns: merge() one-to-one

```
d1
 id sex tc
 1 Nam 4.0
 2  Nu 3.5
 3  Nu 4.7
 4 Nam 7.7
 5 Nam 5.0
 6  Nu 4.2
 7 Nam 5.9
 8 Nam 6.1
 9 Nam 5.9
10  Nu 4.0
```

```
d2
 id sex tg
 1 Nam 1.1
 2  Nu 2.1
 3  Nu 0.8
 4 Nam 1.1
 5 Nam 2.1
 6  Nu 1.5
 7 Nam 2.6
 8 Nam 1.5
 9 Nam 5.4
10  Nu 1.9
11  Nu 1.7
```

```
d <- merge(d1, d2, by="id", all=TRUE)
d
   id sex.x  tc sex.y  tg
1   1   Nam 4.0   Nam 1.1
2   2    Nu 3.5    Nu 2.1
3   3    Nu 4.7    Nu 0.8
4   4   Nam 7.7   Nam 1.1
5   5   Nam 5.0   Nam 2.1
6   6    Nu 4.2    Nu 1.5
7   7   Nam 5.9   Nam 2.6
8   8   Nam 6.1   Nam 1.5
9   9   Nam 5.9   Nam 5.4
10 10    Nu 4.0    Nu 1.9
11 11  <NA>  NA    Nu 1.7
```

In most cases, two data frames are joined by one or more common key variables, (e.g. “id”)

Adding columns: merge()

In most cases, two data frames are joined by one or more common key variables, (e.g. “id”)

d1

year	country	gdp_pc
1994	USA	17314
1995	USA	2522
1996	USA	2475
1994	China	8464
1995	China	11035
1996	China	6331
1994	Sudan	19057
1995	Sudan	1880
1996	Sudan	2448

d2

country	year	demo_score
USA	1994	3
China	1994	16
Sudan	1994	18
USA	1995	3
China	1995	5
Sudan	1995	5
USA	1996	6
China	1996	14
Sudan	1996	5

Adding columns: merge()

In most cases, two data frames are joined by one or more common key variables, (e.g. “country”, “year”)

```
d <- merge(d1, d2, by = c("country", "year"))
country year demo_score gdp_pc
China   1994       16    8464
China   1995        5    11035
China   1996       14    6331
Sudan   1994       18    19057
Sudan   1995        5    1880
Sudan   1996        5    2448
USA     1994        3    17314
USA     1995        3    2522
USA     1996        6    2475
```

#keep only matching observations

Adding columns: merge()

```
d1 <- d1[-c(5, 9), ] #eliminamos las filas 5 & 9 para explicarlos siguientes
                     parametros
```

```
d1
year country gdp_pc
1994     USA  17314
1995     USA   2522
1996     USA   2475
1994   China   8464
1996   China   6331
1994   Sudan  19057
1995   Sudan   1880
```

```
d2
 country year demo_score
USA      1994      3
China    1994     16
Sudan    1994     18
USA      1995      3
China    1995      5
Sudan    1995      5
USA      1996      6
China    1996     14
Sudan    1996      5
```

Adding columns: merge()

```
d1 <- d1[-c(5, 9), ] #eliminamos las filas 5 & 9 para explicarlos siguientes
                     parametros
dim(d1)
dim(d2)   # diferentes dimensiones
```

d1

year	country	gdp_pc
1994	USA	17314
1995	USA	2522
1996	USA	2475
1994	China	8464
1996	China	6331
1994	Sudan	19057
1995	Sudan	1880

d2

country	year	demo_score
USA	1994	3
China	1994	16
Sudan	1994	18
USA	1995	3
China	1995	5
Sudan	1995	5
USA	1996	6
China	1996	14
Sudan	1996	5

Adding columns: merge()

```
merge(x, y, by = c("country", "year"), all.x = TRUE)
#keep all observations in 'd2'
```

- **Natural join:** Sólo mantiene las filas coincidentes entre los dos data sets. `all=FALSE.`
- **Full outer join:** Mantiene todas las filas de ambos data sets. `all=TRUE.`
- **Left outer join:** Mantiene todas las filas del dataframe x y sólo aquellas del dataframe y que coincide con ellas `all.x=TRUE.`
- **Right outer join:** Mantiene todas las filas del dataframe y, y sólo aquellas del dataframe x que coincide con ellas `all.y=TRUE`

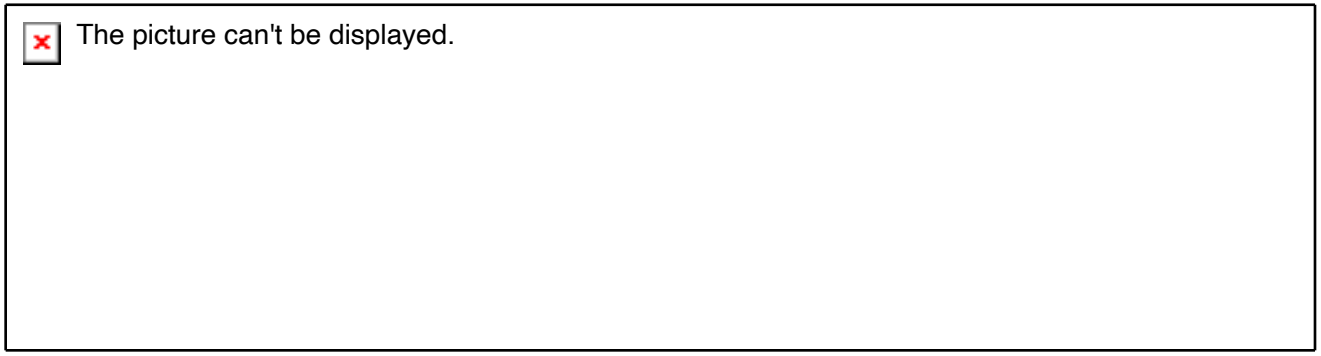


Image from: Book R for dummies https://www.dummies.com/programming/r/