STAT 210
Applied Statistics and Data Analysis
Objects and Data I

Joaquín Ortega

joaquin.ortegasanchez@kaust.edu.sa

Objects and Data

Objects and Data

Data objects are composed of elements, which in simple objects correspond to individual data and in more complex objects can comprise other data objects.

Every element has a **mode** which can be

- *Logical*, with values `TRUE` of `FALSE`.
- *Numerical*: Floating point numbers. They can be written as integers (4, -24), in decimal notation (3.14, -2.717) or in scientific notation (4e12).
- *Complex*: Complex numbers of the form a + bi, where a and b are numerical (e.g., 3.6 + 2.4i).
- *Characters*: Sequences of letters or characters limited by single or double quotation marks (e.g. 'letter' 'test').

Objects and Data

We have listed the modes progressing from the least informative to the most informative.

This order is important when considering data objects created using elements in different modes since there are data objects that do not allow combining different modes.

The number of elements of an object determines its **length**.

Vectors are the simplest data objects and are classified according to their mode and length. To create vectors we use the function `c`, as we have seen.

Objects and Data

It is important to observe that if elements of different modes are combined in the same vector, they will all be assigned the most informative mode among those present:

```
c(T,F,12.3)
```

```
## [1]  1.0  0.0 12.3
```

```
c(T,8.3,12+3i)
```

```
## [1]  1.0+0i  8.3+0i 12.0+3i
```

```
c(F,12.3,'hi')
```

```
## [1] "FALSE" "12.3"  "hi"
```

Objects and Data

Mode and *length* are **atributes** of an object.

All objects in `R` have these two attributes, which are known as implicit attributes. Vectors only have these two.

Other attributes refer to the structure of more complicated objects.

The functions `mode` and `length` give the atributes of an object:

```
mode(letters); length(letters)
```

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

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

Objects and Data

We can introduce a multidimensional structure to the elements of a vector by adding the attribute **dim**, creating an object called an **array**.

The dim attribute is a numeric vector that specifies how many elements should be placed in each dimension. The length of the dim attribute determines how many dimensions there are.

If dim has length two, the array is called a **matrix**. In this case, the first element determines the number of rows and the second, the number of columns.

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

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

```
dim(x) <- c(2,5)
x
```

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

Objects and Data

The table below lists some of the different types of objects available in R and some of their characteristics.

Objet	Modes	Several modes
Vector	Numerical, character, complex or logical	No
Factor	Numerical or character (categorical)	No
Matrix	Numerical, character, complex or logical	No
Array	Numerical, character, complex or logical	No
Data frame	Numerical, character, complex or logical	Yes
ts	Numerical, character, complex or logical	No
List	Any mode, any object	Yes

Table 1 Types of objects.

Objects and Data

- A *factor* is a categorical variable.
- A *matrix* is a two-dimensional arrangement of elements that must all be of the same mode.
- An *array* is a k-dimensional arrangement of elements that must all be of the same mode.
- A *data frame* is a table that allows elements of different modes. All columns must have the same length.
- A *ts* corresponds to a time series. It has additional attributes, such as frequency and dates.
- Finally, one of the most useful objects is the *list*, which can be used to combine any collection of objects into a single object, including other lists.

Factors

Factors

Some variables take non-numerical values, such as gender or the country a person was born in. These are known as **categorical variables** or **factors**, and their values are known as **levels**.

When you create a data frame by reading a file using a command such as `read.table`, all variables containing one or more character strings will be converted automatically into factors.

The function `factor` can be used to create a factor:

```
gender <- factor(c('male', 'female','female',
                   'male', 'female' ))
```

Some important functions for dealing with factors are the following.

Factors

```r
attach(iris)
is.factor(Species)
```

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

```r
is.factor(Sepal.Length)
```

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

```r
levels(Species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

```r
nlevels(Species)
```

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

```r
length(levels(Species))
```

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

Factors

Suppose a factor has n levels. Internally, this factor consists of two items, a vector of integers between 1 and n, corresponding to the levels of the factor, and a vector of length n, containing strings describing what the levels are.

```
opinion <- c(1,2,2,2,1,0,4,4)
fopinion <- factor(opinion,levels=0:4)
levels(fopinion) <- c('awful','bad','regular',
                      'good', 'excellent')
```

The first command creates a numerical vector `opinion` that corresponds to the graded opinions of 8 users of a webpage. The second command creates a factor `fopinion` with this data and five levels, from 0 to 4. The final line assigns the names for the levels.

Factors

Now we look at the objects we have created:

```
opinion
```

```
## [1] 1 2 2 2 1 0 4 4
```

```
fopinion
```

```
## [1] bad       regular   regular   regular
## [5] bad       awful     excellent excellent
## Levels: awful bad regular good excellent
```

```
levels(fopinion)
```

```
## [1] "awful"     "bad"       "regular"
## [4] "good"      "excellent"
```

```
as.numeric(fopinion)
```

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

Factors

The function `as.numeric` extracts the numerical coding for the levels, while `levels` gives the names of the levels.

Observe that the original coding on a scale from 0 to 4 has been changed to a scale from 1 to 5: The internal representation always uses a scale starting at 1.

When using `factor` the option `levels` may be omitted and `R` will then define the levels using the values present in the argument.

However, beware that if a value in a scale is missing, it will not be included among the levels, as the next example shows

Factors

```r
test1 <- factor(c(0,1,2,3,4,5))
test2 <- factor(c(0,1,2,3,5))
levels(test1)
```

```
## [1] "0" "1" "2" "3" "4" "5"
```

```r
levels(test2)
```

```
## [1] "0" "1" "2" "3" "5"
```

Ordered Factors

Ordered Factors

There are two kinds of factors, with an order, known as **ordinals**, and without order, known as **nominal** factors.

Ordered factors have a natural order such a length scales (short, medium, or long) or satisfaction scales (excellent, good, normal, bad, awful).

For the nominal factors, there is no natural order. Examples are the type of soil or the make of a car.

There are two commands for creating a factor, `factor`, and `ordered`. We have already seen examples of the use of the `factor` function. Let's now see some examples of the use of `ordered` to explore the differences between the two.

We use these functions to create a random vector with the values `high`, `medium`, and `low`.

Ordered Factors

```r
(xx <- sample(c('high','medium','low'),10,replace=T))
```

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

```r
(yy <- factor(xx))
```

```
##  [1] high   high   high   medium low    high
##  [7] low    high   low    low
## Levels: high low medium
```

```r
(zz <- ordered(xx))
```

```
##  [1] high   high   high   medium low    high
##  [7] low    high   low    low
## Levels: high < low < medium
```

Ordered Factors

We see some differences in the way the data are presented.

In the first case (`xx`), the elements are of character mode, since they are between quotation marks.

In the other two cases, the elements are levels of a factor.

The levels for `yy` are presented unordered while those for `zz` are (alphabetically) ordered.

We can verify the class of each of these objects:

Ordered Factors

```r
class(xx)
```

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

```r
class(yy)
```

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

```r
class(zz)
```

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

Ordered Factors

```r
is.factor(xx)
```

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

```r
is.factor(yy)
```

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

```r
is.factor(zz)
```

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

Ordered Factors

For the ordered factor `zz`, the alphabetical order is not the natural order among the levels, which would be `low < medium < high`. If we try to change this using the function `levels`, the result is not perhaps what you would expect:

```
levels(zz)
```

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

```
zz
```

```
##  [1] high   high   high   medium low    high
##  [7] low    high   low    low
## Levels: high < low < medium
```

```
levels(zz) <- c('low','medium','high')
zz
```

```
##  [1] low    low    low    high   medium low
##  [7] medium low    medium medium
## Levels: low < medium < high
```

Ordered Factors

The outcome is that everything changes, not only the order of the levels but also the values.

What this command does is to 'translate' everything, values, and levels of the object, to new labels for the levels.

Levels should be assigned when the object is created and for that we may use the function `ordered`:

Ordered Factors

```
xx
```

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

```
(ww <- ordered(xx,levels=c('low','medium','high')))
```

```
##  [1] high   high   high   medium low    high
##  [7] low    high   low    low
## Levels: low < medium < high
```

Tables

Tables

The function `table` is useful for summing up in a table the frequency of factor values in a vector:

```
(fact1 <- sample(c('f','m'),10,replace=T))
```

```
##  [1] "m" "f" "f" "f" "f" "m" "m" "m" "f" "f"
```

```
table(fact1)
```

```
## fact1
## f m
## 6 4
```

Tables

It can also be used to count repeated values in a vector:

```r
table(rpois(100,3))
```

```
##
##  0  1  2  3  4  5  6  7  8
##  4 22 18 23 13 10  6  3  1
```

Tables

Let's see another example using the dataset `mtcars`, that has data on fuel consumption and 10 other variables for 32 cars, 1973-74 models, taken from the 1974 *Motor Trend* magazine.

```r
with(mtcars, table(cyl, gear))
```

```
##    gear
## cyl  3  4  5
##   4  1  8  2
##   6  2  4  1
##   8 12  0  2
```

Tables

More than two factors can be used to build up the table:

```r
with(mtcars, table(cyl, gear,am))
```

```
## , , am = 0
##
##    gear
## cyl  3  4  5
##   4  1  2  0
##   6  2  2  0
##   8 12  0  0
##
## , , am = 1
##
##    gear
## cyl  3  4  5
##   4  0  6  2
##   6  0  2  1
##   8  0  0  2
```

Tables

A nicer output is obtained with the function `ftable`

```
with(mtcars, ftable(cyl, gear,am))
```

```
##          am  0  1
## cyl gear
## 4   3        1  0
##     4        2  6
##     5        0  2
## 6   3        2  0
##     4        2  2
##     5        0  1
## 8   3       12  0
##     4        0  0
##     5        0  2
```