FACTORS

Taiwo

3/10/2021

WHAT IS FACTOR AND HOW IS IT CREATED IN R

Basically, factor in R is a variable used to categorize and store data. Most often, it is used to represent categorical variables that stores both integers and strings data values as levels.

Also, it is possible to create a factor as a vector but with an additional information. The supplementary information consists of records of distinct values in that vector, called levels.

The following example further illustrates how a factor is created using vector and a factor() function.

```
# A vector of integers with unique values
x <- c(5, 12, 13, 12, 5)
# Return a unique values called levels
xf <- factor(x)
xf
[1] 5  12 13 12 5
Levels: 5 12 13
```

With this simple example, we could see that the distinct values in xf are 5, 12 and 13 respectively. In addition, these unique values are called levels.

Taking a look at the structure of the xf object, it is a factor object with three levels.

```
str(xf)
 Factor w/ 3 levels "5","12","13": 1 2 3 2 1
```

The length of a factor is equivalent to the length of the vector that creates it and it is 5.

In the same vein, we can anticipate the future levels of a factor, as seen here:

```
x <- c(5, 12, 13, 12)
xff <- factor(x, levels = c(5, 12, 13, 88))
xff
[1] 5  12 13 12
Levels: 5 12 13 88
xff[2] <- 88
xff
[1] 5  88 13 12
Levels: 5 12 13 88
```

We can see that xff object does not contain value 88 at the first instance, defining it, we allow for the future possibility.

COMMON FUNCTIONS USED WITH FACTORS

There are four functions that are synonymous with factor in R and they are: tapply(), split() and by() function respectively. We shall be illustrating each of them one by one

The tapply() function

Suppose we have a vector that contains the ages of six voters and another vector that contains their political affiliation (Democrat, Republican and Unaffiliated). If we want to find the mean ages in x within each of the party clusters, then it is possible to apply tapply() function to achieve this purpose.

In the normal usage, the call tapply(x, y, z) has x has a vector, f as a factor or list and z as a function. Also, it is possible to group by two factors or more, for instance say you want to group by party and gender, then x will contain two factors; party and gender respectively.

The operation performed by tapply() function is to temporarily split x into clusters or groups, each group corresponding to level of factor and then apply z() function to the resulting subvectors of x. Here is an example to illustrates the idea:

```
voter_ages <- c(30, 25, 36, 45, 59, 70)
political_afil <- c("R", "D", "D", "R", "D", "U")
tapply(voter_ages, political_afil, mean)
   D    R    U
40.0 37.5 70.0
```

Furthermore, this second example shows how voters ages are group by political affiliation and gender. For example, we have the following example expatiates this point:

```
voter_ages <- c(30, 25, 36, 45, 59, 70)
voter_gender <- c("f", "m", "m", "f", "f", "m")
political_afil <- c("R", "D", "D", "R", "D", "R")
y <- data.frame(voter_gender, political_afil)
tapply(voter_ages, y, mean)
            political_afil
voter_gender    D    R
           f 59.0 37.5
           m 30.5 70.0
```

The result shows the mean age of voters according to their political affiliation. Also, as regards to political affiliation, female voters from democratic party has the highest mean age but referring to overall mean age, the male mean age is higher than the female mean age.

The Split() Function

The split() function is opposite to the tapply() function in the sense that its splits a vector into groups and apply a specified function on each group.

The main form of the split function is split(x, f) where x stands vector of data frame and f is a factor or a list of factors. For instance, let's say we create a data frame (income_info) with four variables: gender, age, income and over25 income such as:

```
# Create a data frame with four variables
income_info <- data.frame(gender = c("M", "F", "M", "F", "M",
    "F"), age = c(20, 30, 22, 50, 23, 55), income = c(70000,
    80000, 60000, 40000, 23000, 120000))
# Create a categorical variable that returns one for #each
# age greater than 25 and 0 other wise.
income_info$over25 <- ifelse(income_info$age > 25, 1, 0)
```

```
income_info
  gender age income over25
1      M  20  70000      0
2      F  30  80000      1
3      M  22  60000      0
4      F  50  40000      1
5      M  23  23000      0
6      F  55 120000      1
```

What follows now is to split income variable by the two categorical variables, gender and over25 as follows:

```
split(income_info$income, list(income_info$gender, income_info$over25))
$F.0
numeric(0)

$M.0
[1] 70000 60000 23000

$F.1
[1]  80000  40000 120000

$M.1
numeric(0)
```

The outcome of the split function is a list whose components is denoted by a dollar sign.

by() function

The by() function is an object-oriented wrapper for tapply() applied to data frames. The function argument is by(data, INDICES, FUN, . . . , simplify = TRUE).

The Abalone data set is about predicting the age of abalone from physical measurement. The data set contains four thousand and one hundred and seventy seven observations with nine variables respectively. Click this: abalone to gain access to the data set.

At first, import the data with read.csv() like this:

```
abalone <- read.csv(file = "abalone.csv", header = TRUE)

head(abalone)
  Sex Length Diameter Height Whole.weight Shucked.weight Viscera.weight
1   M  0.455    0.365  0.095       0.5140         0.2245         0.1010
2   M  0.350    0.265  0.090       0.2255         0.0995         0.0485
3   F  0.530    0.420  0.135       0.6770         0.2565         0.1415
4   M  0.440    0.365  0.125       0.5160         0.2155         0.1140
5   I  0.330    0.255  0.080       0.2050         0.0895         0.0395
6   I  0.425    0.300  0.095       0.3515         0.1410         0.0775
  Shell.weight Rings
1        0.150    15
2        0.070     7
3        0.210     9
4        0.155    10
5        0.055     7
6        0.120     8
```

Thereafter, we can call the by() function here. Its works like tapply() but it is applied to objects rather than vectors. Since the input data set is a data frame not a vector, we can call the by() function to perform prediction operation we are looking for.

```
by(abalone, abalone$Sex, function(m) lm(m[, 2] ~ m[, 3]))
abalone$Sex: F

Call:
lm(formula = m[, 2] ~ m[, 3])

Coefficients:
(Intercept)       m[, 3]
    0.04288      1.17918

------------------------------------------------------------
abalone$Sex: I

Call:
lm(formula = m[, 2] ~ m[, 3])

Coefficients:
(Intercept)       m[, 3]
    0.02997      1.21833

------------------------------------------------------------
abalone$Sex: M

Call:
lm(formula = m[, 2] ~ m[, 3])

Coefficients:
(Intercept)       m[, 3]
    0.03653      1.19480
```

We could see from the output that the defined function regresses the second column of the data frame argument m against the third column. Also, the function was called three times - one for each level of sex variable - thus producing the three regression analysis.

Working with Tables

A contingency table is a tabulation of counts and/or percentages for one or more variables. In R, table can be created by the table() function. The table function is mostly use to count the number of unique values or levels of a categorical variable.

To use the table() function with data frame, reference the specific variable such as data frame$variable. For instance, to get the number of unique level of the sex variable in the Abalone dataframe, do the following:

```
table(abalone$Sex)

   F    I    M 
1307 1342 1528 
```