Post 1: All About Factors

Catherine Li

October 15, 2017

All About Factors

Introduction

In class, Professor Gaston Sanchez introduced the concept of factors as follows:

> Factors are data structures exclusively designed to handle categorical data.

Motivation: I decided to write Post 1 on factors because I wanted to learn about the concept of factors in more depth. It seems like it would be useful when organizing and sorting data. Turns out (and you will learn later in this post) that factors can play a large role in data mining, which is interesting.

Audience: This post is intended for people who are interested in learning more about factors in R. In lecture, we learned the basic functions of factors, including how to create factors, how to convert vectors into factors, how R stores factors, and how to manipulate factors. In my post, I will provide a comprehensive guide to factors and introduce practical applications of factors through various examples.

Table of Contents:

Background: Basics of Factors

Let's begin by covering the basics of factors. In this section, I am going to introduce different functions of factors.

What's a factor?

- A factor is a variable in R that works with categorical data.
    - It is most useful in statistical modeling.
- The possible values of a factor are referred to as its levels. The levels of a factor are used when displaying the factor's values.
- The data type for ordinal and categorical vectors is factor.
- A categorical variable takes on a value that is a name or label.
    - Examples of categorical variables include countries, gender, race, age group, and educational level.

There are two types of categorical variables:

- nominal variable: a categorical variable without an order.
    - For example, types of cheese do not have a certain order.
- ordinal variable: a categorical variable that has an order.
    - For example, temperature (Low, Medium, High) or size (Small, Medium, Large) have a sense of order to them.

Create a factor

1. Create a vector with your observations using `c()`. For example, if you are observing gelato flavors in a gelato shop, a vector you create could look like the following. The six flavor choices are the six categories, or in R-terms 'factor levels'. Note: gelato flavors are nominal variables because there is no clear order between them.

```
# Create gelato_vector that contains gelato flavors.
gelato_vector <- c("Hazelnut", "Pistachio", "Stracciatella", "Mango", "Banana", "Coconut")

# Create container_vector that contains the two methods of getting gelato, cup or cone.
container_vector <- c("cup", "cone")
```

2. Use the function `factor()`, with a vector (numeric, character, or logical) as an argument, to encode the vector as a factor.

```
gelato_factor <- factor(gelato_vector)
container_factor <- factor(container_vector)
```

To see the levels of a factor, use the function `levels()`.

```
levels(gelato_factor)
```

```
## [1] "Banana"        "Coconut"       "Hazelnut"      "Mango"
## [5] "Pistachio"     "Stracciatella"
```

```
levels(container_factor)
```

```
## [1] "cone" "cup"
```

Convert to a factor

You have a vector, but don't know how to convert it to a factor? 1. Use the function `factor()`, with a vector (numeric, character, or logical) as an argument, to convert the vector into a factor.

```
# Starting cookies_vector that contains different cookie flavors, use factor() to convert the vector into a factor
.
cookies_vector <- c("Chocolate Chip", "Snickerdoodle", "Ginger", "Oatmeal", "Peanut Butter")
cookies_factor <- factor(cookies_vector)
```

A factor is internally stored using two arrays (R vectors)

- an integer array containing the values of the categories
- the other array is the "levels" which has the names of categories which are mapped to the integers

Use `storage.mode()` to confirm that the values of the categories are mapped as integers.

Checking whether or not an object is a factor

To check whether or not an object is a factor, use `is.factor()`.

```
#Is an object a factor?
is.factor(gelato_vector)
```

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

```
is.factor(gelato_factor)
```

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

Manipulating Factors

Access elements of a factor: Use bracket notation.

```
# Access first element
gelato_factor[1]
```

```
## [1] Hazelnut
## Levels: Banana Coconut Hazelnut Mango Pistachio Stracciatella
```

```
# Access second through fourth elements (second, third, fourth)
gelato_factor[2:4]
```

```
## [1] Pistachio     Stracciatella Mango
## Levels: Banana Coconut Hazelnut Mango Pistachio Stracciatella
```

```
# Access first and third elements
gelato_factor[c(1,3)]
```

```
## [1] Hazelnut      Stracciatella
## Levels: Banana Coconut Hazelnut Mango Pistachio Stracciatella
```

```
# Access all but the first element
gelato_factor[-1]
```

```
## [1] Pistachio     Stracciatella Mango         Banana        Coconut
## Levels: Banana Coconut Hazelnut Mango Pistachio Stracciatella
```

```
# Access elements using a logical vector
gelato_factor[c(TRUE, FALSE, FALSE, TRUE, TRUE, FALSE)]
```

```
## [1] Hazelnut Mango    Banana
## Levels: Banana Coconut Hazelnut Mango Pistachio Stracciatella
```

Modify a factor: Use simple assignments.

```
# Modify third element from "Stracchiatella" to "Mango"
gelato_factor[3] <- "Mango"
gelato_factor
```

```
## [1] Hazelnut  Pistachio Mango     Mango     Banana    Coconut
## Levels: Banana Coconut Hazelnut Mango Pistachio Stracciatella
```

```
# You cannot modify an element into a level that does not exist. For example,
# Modify third element from "Stracchiatella" to "White Chocolate"
gelato_factor[3] <- "White Chocolate"
```

```
## Warning in `[<-.factor`(`*tmp*`, 3, value = "White Chocolate"): invalid
## factor level, NA generated
```

```
gelato_factor
```

```
## [1] Hazelnut  Pistachio <NA>      Mango     Banana    Coconut
## Levels: Banana Coconut Hazelnut Mango Pistachio Stracciatella
```

```
# Assigns NA to the level
```

Summarizing Factors

The function `summary()` gives you an overview of a variable's contents.
For example, suppose you want to know which gelato flavor(s) are the most popular (i.e. which flavor the most people buy).

```
# Create a factor with all observations of gelato purchases in a given day.
gelato_purchased <- c("Banana", "Hazelnut", "Pistachio", "Stracciatella", "Mango", "Hazelnut", "Hazelnut", "Pistac
hio", "Hazelnut", "Hazelnut", "Stracciatella", "Stracciatella", "Stracciatella")
gelato_purchased_factor <- factor(gelato_purchased)
summary(gelato_purchased_factor)
```

```
##        Banana      Hazelnut         Mango     Pistachio Stracciatella
##             1             5             1             2             4
```

Interpretation: On this given day, the most popular flavor is hazelnut, followed by stracciatella in close second. The least popular flavor of the day is coconut because no one purchased coconut.
You can also use the function `table()` to create a table with each of the levels of a factor as its column names. The table also takes the name of the factor. In this case, it is gelato_purchased_factor.

```
#Create table
table(gelato_purchased_factor)
```

```
## gelato_purchased_factor
##        Banana      Hazelnut         Mango     Pistachio Stracciatella
##             1             5             1             2             4
```

Ordered Factors

We are continue into depth about a specific type of factors, ordered factors.

Ordering Factors

Up until now, we have been working with flavors or number purchased, which are nominal categorical variables. Factors become even more useful when ordinal categorical variables. By default, the `factor()` function converts a given vector (the argument of the function) into an unordered factor.

To create an ordered factor, you have to add the two arguments, ordered and levels.
`factor(a_vector, ordered = TRUE, levels = c("level1", "level2", "level3"))`

Arguments:

- ordered = TRUE indicates that the factor is ordered
- levels = c() gives the values of the factor in the correct order.

By specifying the levels for the function `factor()` to use, it becomes clear which vector contains the observations and which vector contains the levels.

```r
# Create a factor with all observations of gelato size purchases in a given day.
size_purchased <- c("medium", "small", "small", "large", "small", "small", "medium", "small", "small", "large", "small", "small", "large")
size_levels <- c("baby", "small", "medium", "large", "extra large")

# Ordered factor for size
size_ordered <- factor(size_purchased, ordered = TRUE, levels = size_levels)
size_ordered
```

```
##  [1] medium small  small  large  small  small  medium small  small  large
## [11] small  small  large
## Levels: baby < small < medium < large < extra large
```

```r
# Unordered factor for size
size_unordered <- factor(size_purchased, ordered = FALSE, levels = size_levels)
size_unordered
```

```
##  [1] medium small  small  large  small  small  medium small  small  large
## [11] small  small  large
## Levels: baby small medium large extra large
```

```r
table(size_ordered)
```

```
## size_ordered
##        baby       small      medium       large extra large
##           0           8           2           3           0
```

```r
table(size_unordered)
```

```
## size_unordered
##        baby       small      medium       large extra large
##           0           8           2           3           0
```

To check whether or not an object is a factor, use `is.factor()`.

```r
#Is the factor ordered?
is.ordered(size_ordered)
```

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

```r
is.ordered(size_unordered)
```

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

If you would like order of the levels to match the order of the observations, use `unique()`. The function reorders the factors levels by first appearance or frequency.

```r
#Use unique()
size_1 <- factor(size_purchased, levels = unique(size_purchased))
size_1
```

```
##  [1] medium small  small  large  small  small  medium small  small  large
## [11] small  small  large
## Levels: medium small large
```

Re-computing a factor's levels

In the tables above for `size_ordered` and `size_unordered`, there are levels that show up after using `table()` or `summary()` that contains the value, 0. To clean up the data, it may be useful to remove these levels that aren't represented in the data.
For a single factor object (for both ordered and unordered factors):

```r
# Remove the extra levels from size_ordered
size_ordered <- factor(size_ordered)
size_ordered
```

```
##  [1] medium small  small  large  small  small  medium small  small  large
## [11] small  small  large
## Levels: small < medium < large
```

```r
summary(size_ordered)
```

```
##  small medium  large
##      8      2      3
```

```r
# Remove the extra levels from size_unordered
size_unordered <- factor(size_unordered)
size_unordered
```

```
##  [1] medium small  small  large  small  small  medium small  small  large
## [11] small  small  large
## Levels: small medium large
```

```r
summary(size_unordered)
```

```
##  small medium  large
##      8      2      3
```

Comparing Ordered Factors

Ordered factors are useful because you can compare them with each other. However, it is important to note than it is only meaningful to compare ordered factors (factors that were created with an argument `ordered = TRUE`), not unordered factors.
Example: Suppose you work at a gelato shop. On a given day, your boss tells you that customers are more inclined to eat and buy more scoops of gelato towards the end of the day. You want to see if what she says is true, so you decide to compare the size of gelato the first customer of the day ordered and the size of gelato the last customer of the day ordered.

```r
# Use bracket notation to choose the values you want to compare. Use operators to compare the respective ordered f
actors.
#Remember, the ORDERED factor for gelato sizes we created earlier is called size_ordered.

# Factor value for second data analyst
size_ordered_1 <- size_ordered[1]

# Factor value for fifth data analyst
size_ordered_2 <- size_ordered[13]

# Is data analyst 2 faster data analyst 5?
size_ordered_1 < size_ordered_2
```

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

Note: If you try this with an unordered factor, you will receive the message: “not meaningful for factors”.

```r
# Remember, the UNORDERED factor for gelato sizes we created earlier is called size_ordered.

# Factor value for second data analyst
size_unordered_1 <- size_unordered[1]

# Factor value for fifth data analyst
size_unordered_2 <- size_unordered[13]

# Is data analyst 2 faster data analyst 5?
size_unordered_1 < size_unordered_2
```

```
## Warning in Ops.factor(size_unordered_1, size_unordered_2): '<' not
## meaningful for factors
```

```
## [1] NA
```

Changing the order of a factor’s levels

Use `relevel()` to make a particular level of an ordered factor first in the list. The first argument is the factor and the second argument is the particular level that you want to move to the first in the list. The function will not work with ordered factors.

```r
# Start with a factor with its levels in the wrong order
size_wrong <- factor(c("large", "small", "medium", "baby", "large"))
size_wrong
```

```
## [1] large  small  medium baby   large
## Levels: baby large medium small
```

```r
# Change the order
size_wrong <- relevel(size_wrong, "medium")
size_wrong <- relevel(size_wrong, "small")
size_wrong <- relevel(size_wrong, "baby")
size_wrong
```

```
## [1] large  small  medium baby   large
## Levels: baby small medium large
```

To reverse the order of levels in a factor:

```r
# Start with a factor with its levels in the wrong order
size_wrong <- factor(c("large", "small", "medium", "large"))
size_wrong
```

```
## [1] large  small  medium large
## Levels: large medium small
```

```r
# Reverse the order
size_reverse <- factor(size_wrong, levels=rev(levels(size_wrong)))
size_reverse
```

```
## [1] large  small  medium large
## Levels: small medium large
```

Renaming a factor's levels

Renaming a factor's levels can come in handy when you are trying to organize the labels of your data. Because machine-readable code is more compact and powerful, it is important to have consistent labels when data mining. For example, if you want to represent countries, you can use abbreviations like "US" instead of "United States of America".
Method 1: Use revalue() or mapvalues() from the plyr package:

```r
# Download the plyr package using the function install.packages() in the console
# install.packages(plyr)

# Load the plyr package
library(plyr)

# Use revalue() to rename the levels.
revalue(size_reverse, c("small"="S", "medium"="M", "large"="L"))
```

```
## [1] L S M L
## Levels: S M L
```

```r
# Use mapvalues() to rename the levels.
mapvalues(size_wrong, from = c("small", "medium", "large"), to = c("S", "M", "L"))
```

```
## [1] L S M L
## Levels: L M S
```

Method 2: Use functions built into R.

```r
cookies_factor
```

```
## [1] Chocolate Chip Snickerdoodle  Ginger         Oatmeal
## [5] Peanut Butter
## Levels: Chocolate Chip Ginger Oatmeal Peanut Butter Snickerdoodle
```

```r
# Rename by name
levels(cookies_factor)[levels(cookies_factor)=="Snickerdoodle"] <- "Pumpkin Spice"
cookies_factor
```

```
## [1] Chocolate Chip Pumpkin Spice  Ginger         Oatmeal
## [5] Peanut Butter
## Levels: Chocolate Chip Ginger Oatmeal Peanut Butter Pumpkin Spice
```

```r
# Rename by position
levels(cookies_factor)[2] <- "Ginger Snap"
cookies_factor
```

```
## [1] Chocolate Chip Pumpkin Spice  Ginger Snap    Oatmeal
## [5] Peanut Butter
## 5 Levels: Chocolate Chip Ginger Snap Oatmeal ... Pumpkin Spice
```

```r
# Rename all levels
levels(cookies_factor) <- c("White Chocolate Macademia Nut","Gingerbread","Red Velvet", "Butter Pecan", "Chocolate Salted Caramel Pretzel", "Carrot Cake")
cookies_factor
```

```
## [1] White Chocolate Macademia Nut    Chocolate Salted Caramel Pretzel
## [3] Gingerbread                      Red Velvet
## [5] Butter Pecan
## 6 Levels: White Chocolate Macademia Nut Gingerbread ... Carrot Cake
```

Graphs of Factors

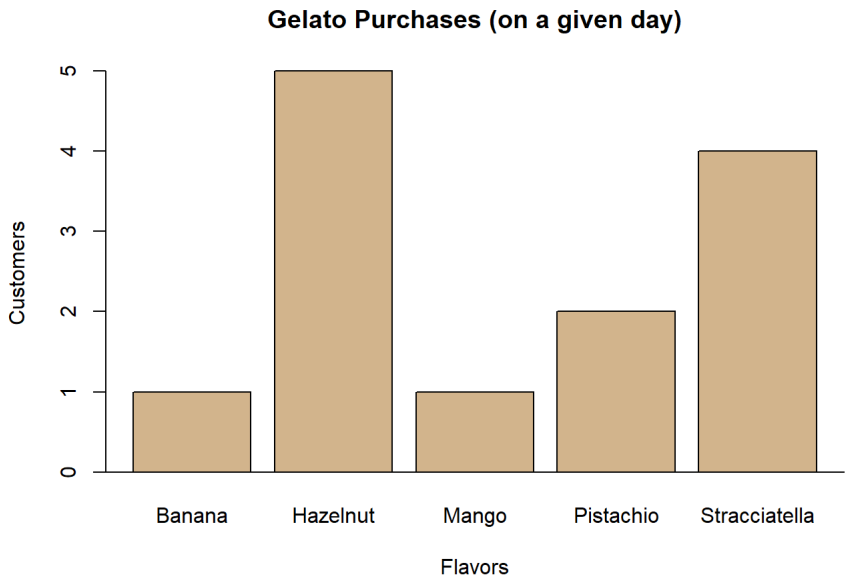
The function `table()` tabulates observations and can be used to create bar plots of factors.

To create a vertical bar graph:

```
table(gelato_purchased_factor)
```

```
## gelato_purchased_factor
##         Banana      Hazelnut         Mango     Pistachio Stracciatella
##              1             5             1             2             4
```

```
{
  barplot(table(gelato_purchased_factor), main = "Gelato Purchases (on a given day)", xlab = "Flavors", ylab = "Customers", col = "tan")
    abline(h=0)       #Add x axis line
}
```



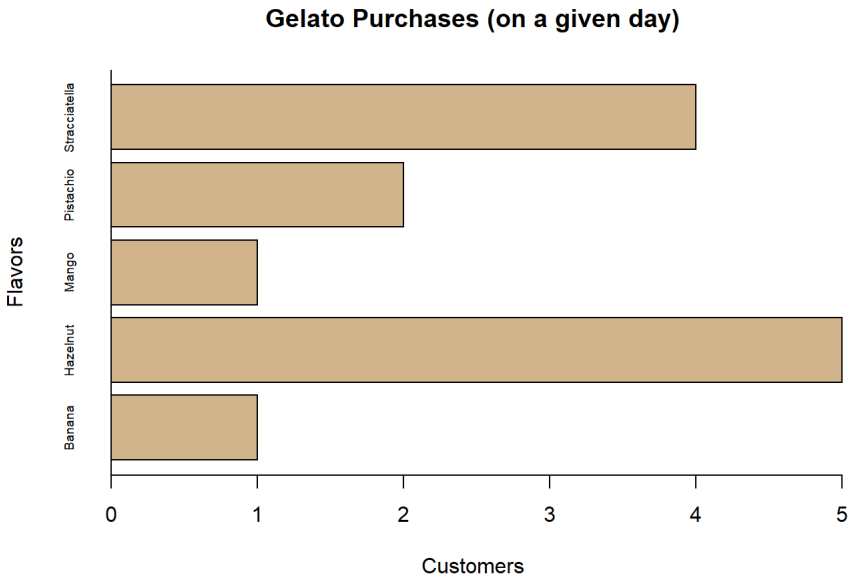
The vertical bar graph shows very clearly which flavors are most and least popular. In a given day, the most popular gelato flavor is hazelnut and the least popular gelato flavors are banana and mango, tied at one customer each. A bar graph provides an easy-to-understand depiction of factors and its levels.

To create a horizontal bar graph:

```
table(gelato_purchased_factor)
```

```
## gelato_purchased_factor
##         Banana      Hazelnut         Mango     Pistachio Stracciatella
##              1             5             1             2             4
```

```
{
  barplot(table(gelato_purchased_factor), horiz = TRUE, main = "Gelato Purchases (on a given day)", xlab = "Customers", ylab = "Flavors", col = "tan", cex.names = 0.58)
abline(v=0)     #Add the y axis line
  }
```



is obvious from the horizontal bar graph that the vertical bar graph is a much clearer depiction. To fit all of the flavor levels on the y axis, the size of the labels had to be reduced, thus making the labels harder to read. Often times with different data sets, you have to experiment with different visualizations to see which one(s) best suit the data you are trying to represent.

Conclusion

From all of this information, you may be wondering, how do factors relate to the bigger picture of R?
How do factors relate to vectors and data frames?

- Factors are stored as integer vectors. The levels are stored in a character vector and the elements are stored as indices.
- Factors are created when categorical/non-numerical data is read into a data frame. By default, the `data.frame()` function converts character vectors into factors. Use the argument `stringAsFactors = FALSE` to stop the automatic conversion.

Take-home message: Here is a crash course to the important information in this post.

- Factors are extremely useful when working with categorical data and statistical modeling.
- Create a factor: `factor()` with a vector (numeric, character, or logical) as an argument
- Convert to a factor: `factor()` with a vector (numeric, character, or logical) as an argument
- A factor is internally stored using two arrays (R vectors)
    1. an integer array containing the values of the categories
    2. the other array is the “levels” which has the names of categories which are mapped to the integers
        - `storage.mode()`: confirm that the values of the categories are mapped as integers

References

1. Factor() Documentation
2. Factors
3. R for Data Science: Factors
4. aRrgh: Factors
5. Factors in R
6. Programming with R: Understanding Factor
7. Data types, part 3: Factors!
8. How to convert a factor to an integerwithout a loss of information?
9. Factor Variables | R Learning Modules
10. Manipulating Data
11. Data Mining with R- Further Data Structures