Lecture 1.9 – Factors, Special Values, and Class Coercion

Specific Learning Objectives:

1.1.9 – Create vectors, arrays, matrices, lists, and data frames.

1.1.10 – Understand vectors and vectorized calculations.

1.1.11 – Understand the data classes of R.

1.1.12 – Learn how to index vectors, arrays, matrices, lists, and data frames.

So far we've learned...

- A lot of different types of data and ways to organize them!

    - Types of data: numeric, integer, complex, logical, character.

    - Useful ways of organizing them (classes): matrix, data frame, list

- Today we'll learn an additional type of data/specialized class (factor).

- We'll also learn about special numbers which don't quite fit into other types.

- Finally, we'll learn how to turn one type of data or class into another (called coercion).

Factors

- Factor is a class of data that is special to R.
  - Factors convert data into categorical data.

```
> colors <- c("white","blue","red", "blue", "red", "red")
> color.fac <- factor(colors)
> color.fac
[1] white blue  red   blue  red   red
Levels: blue red white
```

1 2 3

Values	
color.fac	Factor w/ 3 levels "blue","red","white": 3 …
colors	chr [1:6] "white" "blue" "red" "blue" "red"…

- Each category is represented by a name and encoded with integers.

```
> as.numeric(color.fac)
[1] 3 1 2 1 2 2
```

This saves a lot of memory with large data sets!

Creating Factors

- Factor creation is easy.
  - Create factored data *de novo* with **`factor()`**:

```
> pets <- factor(c("dog", "cat", "squirrel", "dog", "cat", "dog", "dog", "bear", "cat",
+                  "fish"))
```

    - **`factor()`** takes a vector as its first argument.

Text, numbers, etc must be exact matches for `factor()` to work properly!

    - **`factor()`** takes **`levels`** as an argument if you want to assign specific levels.

- You can also use an existing vector with **`factor()`**:

```
> colors <- c("white","blue","red", "blue", "red", "red")
> color.fac <- factor(colors)
```

Note: factors are only generated in vector form!

Creating Factors

- Factors can be either *unordered* or *ordered*.
  - Unordered factors: blue, red, white
  - Ordered factors: good, better, best

- Create ordered factors also using **`factor()`**: use argument **`ordered=TRUE`** and add **`levels`** to specify order.

```
> pet.quality <- c("great", "good", "ok", "great", "good", "great", "great", "poor", "good",
+                  "ok")
> pet.quality <- factor(pet.quality, levels=c("poor", "ok", "good", "great"), ordered=T)
```

Here we are overwriting `pet.quality` with the new factored vector!

```
> pet.quality
 [1] great good  ok    great good  great great poor  good  ok
Levels: poor < ok < good < great
```

Here is the order of your levels!

Factor Info

- **`levels()`** will return the levels of your factor

```
> levels(pets)
[1] "bear"     "cat"      "dog"      "fish"     "squirrel"
> levels(pets)[1]
[1] "bear"
```

You can select a single level by specifying a number in square brackets after the **`levels()`** function.

```
> levels(pet.quality)
[1] "poor"  "ok"    "good"  "great"
```

On ordered data, **`levels()`** will report in their order.

- You can access the elements of factored vector the same as other vectors! As are:
  - Replacing elements
  - Subsetting
  - Basically they are just like regular vectors

Check Your Understanding

Create a data frame that contains:

- **a column of class character describing something in your current surroundings**

- **a column of class factor describing the color of those items**

When data doesn't quite fit (or isn't there)…

- There are sometimes numbers (or non-numbers) that pop up in data sets and calculations. They should be treated specially by R, so they have special values.

  - Infinity
  - Data not available
  - Not-a-numbers
  - NULL values

Infinity

- Infinity occurs when *R* is forced to limit the size of a number. When it is too large, it is considered by *R* to be infinite (mathematical symbol: ∞).

    - Infinity is **`Inf`** in R

    - Infinity can be positive (**`Inf`**) or negative (**`-Inf`**)

    - Once an element is infinite, all operations on that element will result in another infinite number.

    - If you divide a number by **`Inf`**, the result is 0 (zero).

    - You can check an R object for infinities using **`is.infinite()`** and/or **`is.finite()`** which will return a logical for each element.

Not Available

- When you have data that is unknown, uncollected, or otherwise “not available”, you want a special value to indicate this in a data set.

    - R represents not-available data with the value **`NA`**

    - Elements having **`NA`** will not accept any manipulation other than replacing the element! R will completely ignore this element.

- **`NA`** serves as a useful placeholder

- You can check for the presence of **`NA`** by using **`is.na()`** which will return a logical for each element in an R object.

- You can remove any observations with an **`NA`** by using **`na.omit()`**.

Not-a-number

- When you perform a calculation that results in a value that is not a number, such as **`Inf/Inf`** or **`0/0`**.

    - R represents not-a-number elements with the value **`NaN`**

    - Elements having **`NaN`** will not accept any manipulation other than replacing the element! R will completely ignore this element.

- **`NaN`** is functionally the same as **`NA`** but holds a different meaning.

- You can check for the presence of **`NaN`** by using **`is.nan()`** which will return a logical for each element in an R object.

NULL

- **`NULL`** indicates an empty entity (value, vector, etc).
  - **`NULL`** is different than **`NA`**:

```
> c(NA,NA,NA)
[1] NA NA NA
```

```
> c(NULL,NULL,NULL)
NULL
```

    - In this example, **`NA`** represents three missing values, so they exist or could be known, but are not recorded. This is reflected in the output, recording three spots of **`NA`**!
    - However, **`NULL`** in three empty places (or emptiness three times) is interpreted as a single empty entity.

- Check for **`NULL`** using **`is.null()`**

Special values in practical use

- When to use each:
  - **`Inf`** - do not enter
  - **`NA`** - use this when data is missing
  - **`NaN`** - do not enter
  - **`NULL`** - Use this as a place holder only for entire objects
- What each practically means:
  - **`Inf`** / **`NULL`** / **`NaN`** - a calculation did something unexpected (divided by zero), you should go back and check to see what went wrong
  - **`NA`** - data is missing, will screw up some functions. Options:
    - Use **`na.rm`** argument

```
> avec <- c(1, 2, 3, NA, 5)
> mean(avec)
[1] NA
> mean(avec, na.rm = TRUE)
[1] 2.75
```

    - Use **`na.omit()`** to clean data frames, will remove observations with any **`NA`** (in other words, rows!)

Check Your Understanding

What would you do in the following scenarios?

a) **You have a data set in a data frame. It is very large, and you want to know whether or not it has missing data values. You'd also like to remove those values so that you only have a set of complete observations.**

b) **You have a data frame in which you use your own function with vectorized calculations. When you look at the output, it is `NULL`. What has happened? What should you do?**

Class and Data-type Coercion

- Sometimes you will need data types or classes to be converted into another class or type in order to perform some action. Converting one type or class of data into another is called **coercion**.

    - Coercion can happen automatically, such as logical to numeric with **`sum()`** or integer to numeric with many different functions.

- Sometimes you will have to coerce data manually, this is most often accomplished with **`as.`** functions:

    - **`as.character()`**
    - **`as.numeric()`**
    - **`as.integer()`**
    - **`as.factor()`**
    - **`as.logical()`**
    - **`as.data.frame()`**

Class and Data-type Coercion

- Lists are a special case since they are often comprised of members of different classes.
  - **`as.list()`** will coerce data into list form.
  - **`unlist()`** will coerce list members into atomic vectors.

```
> unlisted.cov <- unlist(ability.cov)
> unlisted.cov
   cov1    cov2    cov3    cov4    cov5    cov6    cov7    cov8    cov9   cov10   cov11   cov12   cov13
 24.641   5.991  33.520   6.023  20.755  29.701   5.991   6.700  18.137   1.782   4.936   7.204  33.520
  cov14   cov15   cov16   cov17   cov18   cov19   cov20   cov21   cov22   cov23   cov24   cov25   cov26
 18.137 149.831  19.424  31.430  50.753   6.023   1.782  19.424  12.711   4.757   9.075  20.755   4.936
  cov27   cov28   cov29   cov30   cov31   cov32   cov33   cov34   cov35   cov36 center1 center2 center3
 31.430   4.757  52.604  66.762  29.701   7.204  50.753   9.075  66.762 135.292   0.000   0.000   0.000
center4 center5 center6   n.obs
  0.000   0.000   0.000 112.000
```

Check Your Understanding

Which command can accomplish the following coercions:

a) a logical vector into a numeric vector

b) a numeric vector into a character vector

c) two vectors into a data frame

d) a list into a vector

Action Items

1. **Complete Assignments 1.12 and 1.13.**

2. **Prepare for your first Skill Check!**