4: Factors

Introduction to R

This assignment consists of six parts:

- Intro to Basics
- Vectors
- Matrices
- Factors (this document)
- Data frames
- Lists

If you haven't already, download the answer sheet for this document (**Note:** right-click and save the file with the `.Rmd` extension to your `hw02` folder.)

After you complete each exercise, push the R Notebook to your remote repo. See Part 0 for instructions. Do *not* push this document.

4.1 What's a factor and why would you use it?

In this part you dive into the wonderful world of factors.

The term **factor** refers to a statistical data type used to store categorical variables. The difference between a categorical variable and a continuous variable is that a categorical variable can belong to a limited number of categories. A continuous variable, on the other hand, can correspond to an infinite number of values.

It is important that R knows whether it is dealing with a continuous or a categorical variable, as the statistical models you will develop in the future treat both types differently. (You will see later why this is the case.)

A good example of a categorical variable is sex. In many circumstances you can limit the sex categories to "Male" or "Female". (Sometimes you may need different categories. For example, you may need to consider chromosomal variation, hermaphroditic animals, or different cultural norms, but you will always have a finite number of categories.)

4.1 Instructions

- Assign to variable `theory` the value "factors for categorical variables".

```
# Assign to the variable theory what this chapter is about!


#
```

4.2 What's a factor and why would you use it? (2)

To create factors in R, you make use of the function `factor()`. First thing that you have to do is create a vector that contains all the observations that belong to a limited number of categories. For example, `sex_vector` contains the sex of 5 different individuals:

```
sex_vector <- c("Male","Female","Female","Male","Male")
```

It is clear that there are two categories, or in R-terms 'factor levels', at work here: "Male" and "Female".

The function `factor()` will encode the vector as a factor:

```
factor_sex_vector <- factor(sex_vector)
```

Instructions

- Convert the character vector `sex_vector` to a factor with `factor()` and assign the result to `factor_sex_vector`
- Print out `factor_sex_vector` and assert that R prints out the factor levels below the actual values.

```
# Sex vector
sex_vector <- c("Male", "Female", "Female", "Male", "Male")

# Convert sex_vector to a factor


# Print out factor_sex_vector


#
```

4.3 What's a factor and why would you use it? (3)

There are two types of categorical variables: a **nominal** categorical variable and an **ordinal** categorical variable.

A nominal variable is a categorical variable without an implied order. This means that it is impossible to say that 'one is worth more than the other'. For example, think of the categorical variable `animals_vector` with the categories "Elephant", "Giraffe", "Donkey" and "Horse". Here, it is impossible to say that one stands above or below the other. (Note that some of you might disagree ;-)).

In contrast, ordinal variables do have a natural ordering. Consider for example the categorical variable `temperature_vector` with the categories: "Low", "Medium" and "High". Here it is obvious that "Medium" stands above "Low", and "High" stands above "Medium".

Instructions

Run the script below to check how R constructs and prints nominal and ordinal variables. Do not worry if you do not understand all the code just yet, we will get to that.

```
# Animals
animals_vector <- c("Elephant", "Giraffe", "Donkey", "Horse")
factor_animals_vector <- factor(animals_vector)
factor_animals_vector

# Temperature
temperature_vector <- c("High", "Low", "High","Low", "Medium")
factor_temperature_vector <- factor(temperature_vector, order = TRUE, levels = c("Low", "Medium", "High
factor_temperature_vector
```

4.4 Factor levels

When you first get a data set, you will often notice that it contains factors with specific factor levels. However, sometimes you will want to change the names of these levels for clarity or other reasons. R allows you to do this with the function `levels()`:

```
levels(factor_vector) <- c("name1", "name2",...)
```

A good illustration is the raw data that is provided to you by a survey. A common question for every questionnaire is the sex of the respondent. Here, for simplicity, just two categories were recorded, "M" and "F". (You usually need more categories for survey data; either way, you use a factor to store the categorical data.)

```
survey_vector <- c("M", "F", "F", "M", "M")
```

Recording the sex with the abbreviations "M" and "F" can be convenient if you are collecting data with pen and paper, but it can introduce confusion when analyzing the data. At that point, you will often want to change the factor levels to "Male" and "Female" instead of `"M"` and `"F"` for clarity.

Watch out: the order with which you assign the levels is important. If you type `levels(factor_survey_vector)`, you'll see that it outputs `[1] "F" "M"`. If you don't specify the levels of the factor when creating the vector, R will automatically assign them alphabetically. To correctly map `"F"` to `"Female"` and `"M"` to `"Male"`, the levels should be set to `c("Female", "Male")`, in this order.

Instructions

- Check out the code that builds a factor vector from `survey_vector`. You should use `factor_survey_vector` in the next instruction.
- Change the factor levels of `factor_survey_vector` to `c("Female", "Male")`. Mind the order of the vector elements here.

```
# Code to build factor_survey_vector
survey_vector <- c("M", "F", "F", "M", "M")
factor_survey_vector <- factor(survey_vector)

# Specify the levels of factor_survey_vector
levels(factor_survey_vector) <- c("F", "M")

factor_survey_vector
```

4.5 Summarizing a factor

After finishing this course, one of your favorite functions in R will be `summary()`. This will give you a quick overview of the contents of a variable:

```
summary(my_var)
```

Going back to our survey, you would like to know how many `"Male"` responses you have in your study, and how many `"Female"` responses. The `summary()` function gives you the answer to this question.

Instructions

- Ask for a `summary()` of the `survey_vector` and `factor_survey_vector`. Interpret the results of both vectors. Are they both equally useful in this case?

```r
# Build factor_survey_vector with clean levels
survey_vector <- c("M", "F", "F", "M", "M")
factor_survey_vector <- factor(survey_vector)
levels(factor_survey_vector) <- c("Female", "Male")

# Generate summary for survey_vector


# Generate summary for factor_survey_vector


#
```

4.6 Battle of the sexes

You might wonder what happens when you try to compare elements of a factor. In `factor_survey_vector` you have a factor with two levels: `"Male"` and `"Female"`. But how does R value these relative to each other?

Instructions

- Read the code in the editor and then run it to test if `male` is greater than (`>`) `female`.

```r
# Build factor_survey_vector with clean levels
survey_vector <- c("M", "F", "F", "M", "M")
factor_survey_vector <- factor(survey_vector)
levels(factor_survey_vector) <- c("Female", "Male")

# Male
male <- factor_survey_vector[1]

# Female
female <- factor_survey_vector[2]

# Battle of the sexes: Male 'larger' than female?
male > female
```

4.7 Ordered factors

Since `"Male"` and `"Female"` are unordered (or nominal) factor levels, R returns a warning message, telling you that the greater than operator is not meaningful. As seen before, R attaches an equal value to the levels for such factors.

But this is not always the case! Sometimes you will also deal with factors that do have a natural ordering between its categories. If this is the case, we have to make sure that we pass this information to R.

Let us say that you are leading a research team of five data analysts and that you want to evaluate their performance. To do this, you track their speed, evaluate each analyst as `"slow"`, `"medium"` or `"fast"`, and save the results in `speed_vector`.

Instructions

As a first step, assign `speed_vector` a vector with 5 entries, one for each analyst. Each entry should be either `"slow"`, `"medium"`, or `"fast"`. Use the list below:

- Analyst 1 is medium,
- Analyst 2 is slow,
- Analyst 3 is slow,
- Analyst 4 is medium and
- Analyst 5 is fast.

No need to specify these are factors yet.

```
# Create speed_vector


#
```

4.8 Ordered factors (2)

`speed_vector` should be converted to an ordinal factor since its categories have a natural ordering. By default, the function `factor()` transforms `speed_vector` into an unordered factor. To create an ordered factor, you have to add two additional arguments: `ordered` and `levels`.

```
factor(some_vector,
       ordered = TRUE,
       levels = c("lev1", "lev2" ...))
```

By setting the argument `ordered` to `TRUE` in the function `factor()`, you indicate that the factor is ordered. With the argument `levels` you give the values of the factor in the correct order.

Instructions

- From `speed_vector`, create an ordered factor vector: `factor_speed_vector`. Set `ordered` to `TRUE`, and set `levels` to `c("slow", "medium", "fast")`.
- Print out `factor_speed_vector`
- Use `summary()` to summarize `factor_speed_vector`

```
# Create speed_vector
speed_vector <- c("medium", "slow", "slow", "medium", "fast")

# Convert speed_vector to ordered factor vector


# Print and summarize factor_speed_vector


#
```

4.9 Comparing ordered factors

Having a bad day at work, 'data analyst number two' enters your office and starts complaining that 'data analyst number five' is slowing down the entire project. Since you know that 'data analyst number two' has the reputation of being a smarty-pants, you first decide to check if his statement is true.

The fact that `factor_speed_vector` is now ordered enables us to compare different elements (the data analysts in this case). You can simply do this by using the well-known operators.

Instructions

- Use `[2]` to select from `factor_speed_vector` the factor value for the second data analyst. Store it as `da2`.
- Use `[5]` to select the `factor_speed_vector` factor value for the fifth data analyst. Store it as `da5`.
- Check if `da2` is greater than `da5`; simply print out the result. Remember that you can use the `>` operator to check whether one element is larger than the other.

```
# Create factor_speed_vector
speed_vector <- c("medium", "slow", "slow", "medium", "fast")
factor_speed_vector <- factor(speed_vector, ordered = TRUE, levels = c("slow", "medium", "fast"))

# Factor value for second data analyst


# Factor value for fifth data analyst


# Is data analyst 2 faster than data analyst 5?


#
```