# Part 4: Factors

Introduction to R

This assignment consists of six parts:

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

Create a script called `hw02-4.R` and save it in your `hw02` folder.

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

---

## 4.1 What is a factor?

In R, a **factor** is vector data used for categorical variables, also called discrete data. Discrete data can belong only to one of a limited number of categories, such as plant, animal, fungus, or bacteria. The Linnean classification hierarchy used by biologists is another example of categorical data. A species can belong to only one genus, one family, one order, and so on. Factors are different from continuous data that can have an unlimited number of possible values. R handles factors and continuous variables differently as you will learn during this course.

To create factors in R, you first create a vector with the names of your discrete categories. You then pass that vector to the `factor()` function. Study this example.

```
# Genotype can be homozygous or heterozyous. This vector contains
# the genotype of six individuals.
genotype_vector <- c("Heterozygote", "Heterozygote", "Homozygote", "Heterozygote", "Homozygote", "Heter

genotype_factor <- factor(genotype_vector)

genotype_factor
```

```
## [1] Heterozygote Heterozygote Homozygote   Heterozygote Homozygote
## [6] Heterozygote
## Levels: Heterozygote Homozygote
```

The output lists the elements of the vector and also **Levels.** Levels are the unique categories that `factor()` found in the vector.

### 4.1 Instructions

For this exercise, you will assume that you counted the number and species of bumble bees (genus *Bombus*) that visitied a single flowering plant in one hour. Your notes show the order of the visitors:

- kirbiellus
- kirbiellus
- flavifrons
- kirbiellus
- bifarius
- flavifrons
- kirbiellus

Add code to your script to

- Make a character vector called `bee_visitors_vector` with these seven visitors. Your vector must reflect the visiting order listed above.
- Use `factor()` to make a factor from `bee_visitors_vector`. Assign the new factor to `bee_visitors_factor`
- Print out `bee_visitors_factor` to be sure you have the order and three levels. A incorrect number of levels suggests you might have spelled a name incorrectly.

```
# Make a character vector called `bee_visitors_vector`


# Make a `bee_visitors_factor` factor from `bee_visitors_vector`.


# Print out bee_visitors_factor


#
```

Your results should look like this. Notice that the levels are arranged alphabetically. R does this automatically, which may not be what you want. You'll learn later how to arrange factors in the order you want.

```
## [1] kirbiellus kirbiellus flavifrons kirbiellus bifarius   flavifrons kirbiellus
## Levels: bifarius flavifrons kirbiellus
```

## 4.2 Factor types

Categorical variables can be **ordinal** or **nominal.** Ordinal categories are ordered. For example, the bee species listed above have proboscis lengths that can be categorized as short, intermediate, or long An order is implied because short < intermediate < long lengths. You can choose to order the categories from short to long or long to short. What's important is that an order is implied.

Nominal categories (nomen = name) do not have an implied order. The three species of bees are nominal categories. There is no implied order to the names of the three species. *Bombus kirbiellus* is not any more or less a bumble bee than *B. flavifrons* or *B. bifarius.* They're just different.

Study the example below. The *Bacillus* factor is nominal because no order is implied for the bacteria. `factor()` defaults to alphabetical order for the levels. The allozyme factor is ordinal because allozymes move through an electrophoretic gel at different speeds (here, slow, medium, fast). Notice that the order of the levels is shown as `Slow < Medium < Fast`.

```
# Bacillus bacteria
bacillus_vector <- c("subtilis", "cereus", "pasteurii", "sphaericus")
bacillus_factor <- factor(bacillus_vector)
bacillus_factor
```

```
## [1] subtilis   cereus     pasteurii  sphaericus
```

```
## Levels: cereus pasteurii sphaericus subtilis
```

```
# Allozymes are slow, medium, or fast
allozyme_vector <- c("Medium", "Slow", "Fast", "Slow", "Medium")
allozyme_factor <- factor(allozyme_vector, order = TRUE, levels = c("Slow", "Medium", "Fast"))
allozyme_factor
```

```
## [1] Medium Slow   Fast   Slow   Medium
## Levels: Slow < Medium < Fast
```

Pay attention to the arguments for `factor()` in the allozyme example. In addition to the vector, the `order = TRUE` argument tells the function that order matters. The `levels = c("Slow", "Medium", "Fast")` argument tells the function the specific order of the levels.

```
allozyme_vector <- c("Medium", "Slow", "Fast", "Slow", "Medium")
allozyme_factor <- factor(allozyme_vector, order = TRUE, levels = c("Slow", "Medium", "Fast"))
```

How would you rewrite the call to `factor()` to arrange the allozyme levels from fast to slow?

### 4.2 Instructions

Here are the proboscis lengths for the three species of *Bombus* you surveyed.

- kirbiellus: Long
- flavifrons: Intermediate
- bifarius: Short

Add code to your script to

- Create a vector of seven lengths that matches the order of the bees in your nominal vector above (the order you observed them). Name that vector `proboscis_vector`.
- Create an ordinal factor called `proboscis_factor`, with the lengths ordered from long to short.
- Display `proboscis factor` to be sure you have it correct.

**Hint:** Remember that case matters.

```
# Create `proboscis_vector` with seven elements


# Create an ordered `proboscis_factor` ordered from long to short.


# Display the contents of `proboscis_factor


#
```

```
# Create `proboscis_vector` with seven elements
proboscis_vector <- c("Long", "Long", "Intermediate", "Long", "Short", "Intermediate", "Long")

# Create an ordered `proboscis_factor` ordered from long to short.
proboscis_factor <- factor(proboscis_vector, ordered = TRUE, levels = c("Long", "Intermediate", "Short"))

# Display the contents of `proboscis_factor
proboscis_factor
```

```
## [1] Long         Long         Intermediate Long         Short
## [6] Intermediate Long
```

```
## Levels: Long < Intermediate < Short
```

```
#
```

**Note:** The default for `factor()` is to create unordered (nominal) factors. You must specify levels and pass the `ordered = TRUE` argument to the function to create ordinal factors.

## 4.3 Summarizing a factor

A useful function that you will use often is `summary()` which provides a summary of the information contained in a variable. The example below shows how to apply `summary()` to `allozyme_factor` to summarize the number of observations of each level.

```
summary(allozyme_factor)
```

```
##   Slow Medium   Fast
##      2      2      1
```

### 4.3 Instructions

Add code to your script to

- apply the `summary()` function to `bee_visitors_factor` and `proboscis_factor`. You do not have to save the results to a variable.

## 4.4 Extracting from and comparing factors

Factors are simply categorical vectors. If you want to extract a particular element, you can use square brackets `[]` like you would with other vectors. Study these examples.

```
# Show all the elements for review
allozyme_factor

# Extract specific elements
allozyme_factor[4]     # Get the fourth element
allozyme_factor[2:4]   # Get elements 2 through 4
allozyme_factor[c(1,3,5)] # Get elements 1, 3, and 5

# Extract slow allozyme elements using a boolean vector
slow_allozymes <- allozyme_factor == "Slow"
allozyme_factor[slow_allozymes]
```

You have to be careful using boolean operators like `>` and `<=` because they only work on ordered factors where some level is greater than or less than another level. Nominal factors are unordered so "greater than" and "less than" do not have any context. Study these example.

```
# Show bacillus (unordered) and allozyme (ordered) factors for review
bacillus_factor
```

```
## [1] subtilis   cereus     pasteurii  sphaericus
## Levels: cereus pasteurii sphaericus subtilis
```

```
allozyme_factor
```

```
## [1] Medium Slow   Fast   Slow   Medium
## Levels: Slow < Medium < Fast
```

```r
# Extract allozymes that are faster than slow. This is just one method.
not_slow_allozymes <- allozyme_factor > "Slow"
allozyme_factor[not_slow_allozymes]
```

```
## [1] Medium Fast   Medium
## Levels: Slow < Medium < Fast
```

```r
# Test to see which is greater: medium or fast

az1 <- allozyme_factor[1] # medium allozyme
az3 <- allozyme_factor[3] # fast allozyme

# FALSE because allozyme level is Slow < Medium < Fast.
# Medium is less than Fast.
az1 > az3
```

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

```r
# The reverse is TRUE
az3 > az1  # Could also test as az1 < az3
```

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

```r
## Now, test with underordered bacillus factor
##
# This works because there is a level for "cereus"
bacillus_factor == "cereus"
```

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

```r
# This does not because one species is not greater than another
bacillus_factor > "cereus"
```

```
## Warning in Ops.factor(bacillus_factor, "cereus"): '>' not meaningful for factors
```

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

### 4.4 Instructions

Add code to your script to do the following.

- Extract from `bee_visitors_factor` the elements that are *not* kirbiellus, using a boolean technique. If necessary, review 02_vectors to review different boolean operators. Try to use the shortest code possible.
- Extract from `bee_visitors_factor` the second, and fourth through sixth elements. Use the `:` for the range. You do not have to save this result to a variable.
- Test whether the first element of `proboscis_factor` is longer than the last element of `proboscis_factor`. Follow the example for `allozyme_factor` above. You do not have to save this result to a variable.
- **Optional challenge:** Redo the previous test of the first and last elements but use `length()` to obtain the position of the last element in `proboscis_factor`. Remember you can look up help for functions using `?`.

```r
# Extract from `bee_visitors_factor` the elements that are *not* kirbiellus


# Extract from `bee_visitors_factor` the second, and fourth through sixth elements. Use the colon for t
```

```
# Test whether the first element of `proboscis_factor` is longer than the last element of `proboscis_fa

# CHALLENGE

#
```

### Save to GitHub

Be sure you have used comments throughout your code to identify sections and to describe what you are doing, then commit and push your `hw02-4.R` script to GitHub.