Lesson Notes | Factors in R

GRAPH Network & WHO, supported by the Global Fund to fight HIV, TB & Malaria

October 2023

This document is a draft of a lesson made by the GRAPH Network, a non-profit headquartered at the University of Geneva Global Health Institute, in collaboration with the World Health Organization, under a Global Fund 2023 grant to create e-learning modules to build in-country data capacity for epidemiological and impact analysis for National HIV, TB and malaria programs

EDITS EXAMPLEasfkdjhaskfhlajfhalskf

Introduction

Factors are an important data class for representing and working with categorical variables in R. In this lesson, we will learn how to create factors and how to manipulate them with functions from the `forcats` package, a part of the tidyverse. Let's dive in!

Learning Objectives

- You understand what factors are and how they differ from characters in R.
- You are able to modify the **order** of factor levels.
- You are able to modify the **value** of factor levels.

Packages

```
# Load packages
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse,
               here)
```

Dataset: HIV Mortality

We will use a dataset containing information about HIV mortality in Colombia from 2010 to 2016, which is hosted on the open data platform 'Datos Abiertos Colombia.' You can learn more and access the full dataset here.

Each row corresponds to an individual who passed away from AIDS or AIDS-related-complications.

```
hiv_mort <- read_csv(here("data/colombia_hiv_deaths_2010_to_2016"))
```

```
## # A tibble: 5 × 25
##   municipality_type death_location  birth_date birth_year
##   <chr>             <chr>           <date>          <dbl>
## 1 Municipal head    Hospital/clinic 1956-05-26       1956
## 2 Municipal head    Hospital/clinic 1983-10-10       1983
## 3 Municipal head    Hospital/clinic 1967-11-22       1967
## 4 Municipal head    Home/address    1964-03-14       1964
## 5 Municipal head    Hospital/clinic 1960-06-27       1960
## # ℹ 21 more variables: birth_month <chr>, birth_day <dbl>,
## #   death_year <dbl>, death_month <chr>, death_day <dbl>, …
```

See the appendix at the bottom for the data dictionary describing all variables.

What are Factors?

Factors are an important data class in R used to represent categorical variables.

A categorical variable takes on a limited set of possible values or levels. For example, country, race or political affiliation. These differ from free-form string variables that take arbitrary values, like person names, book titles or doctor's comments.

RECAP

Review of the Main Data Classes in R

- `Numeric`: Represents continuous numerical data, including decimal numbers.
- `Integer`: Specifically for whole numbers without decimal places.
- `Character`: Used for text or string data.
- `Logical`: Represents boolean values (TRUE or FALSE).

RECAP

- `Factor`: Used for categorical data with predefined levels or categories.
- `Date`: Represents dates without times.

Factors have a few key advantages over character vectors for working with categorical data in R:

- Factors are stored in R slightly more efficiently than characters.
- Certain statistical functions, such as `lm()`, require categorical variables to be input as factors
- Factors allow control over the order of categories or levels. This allows properly sorting and plotting of categorical data.

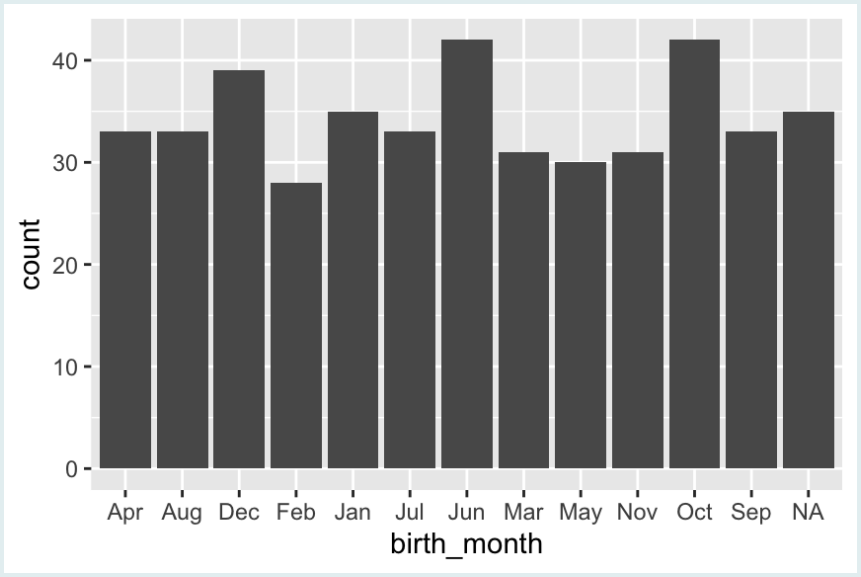
This last point, controlling the order of factor levels, will be our primary focus.

Factors in Action

Let's see a practical example of the value of factors using the `hiv_mort` dataset we loaded above.

Suppose you are interested in visualizing the patients in the dataset by their birth month. We can do this with ggplot:

```
ggplot(hiv_mort) +
  geom_bar(aes(x = birth_month))
```



However, there's a hiccup: the x-axis (representing the months) is arranged alphabetically, with April first on the left, then August, and so on. But months should follow a specific chronological order!

We can arrange the plot in the desired order by creating a factor using the `factor()` function:

```
hiv_mort_modified <-
  hiv_mort %>%
  mutate(birth_month = factor(x = birth_month,
                              levels = c("Jan", "Feb", "Mar", "Apr",
                                         "May", "Jun", "Jul", "Aug",
                                         "Sep", "Oct", "Nov", "Dec")))
```

The syntax is straightforward: the `x` argument takes the original character column, `birth_month`, and the `levels` argument takes in the desired sequence of months.

When we inspect the data type of the birth_month variable, we can see its transformation:

```
# Modified dataset
class(hiv_mort_modified$birth_month)
```

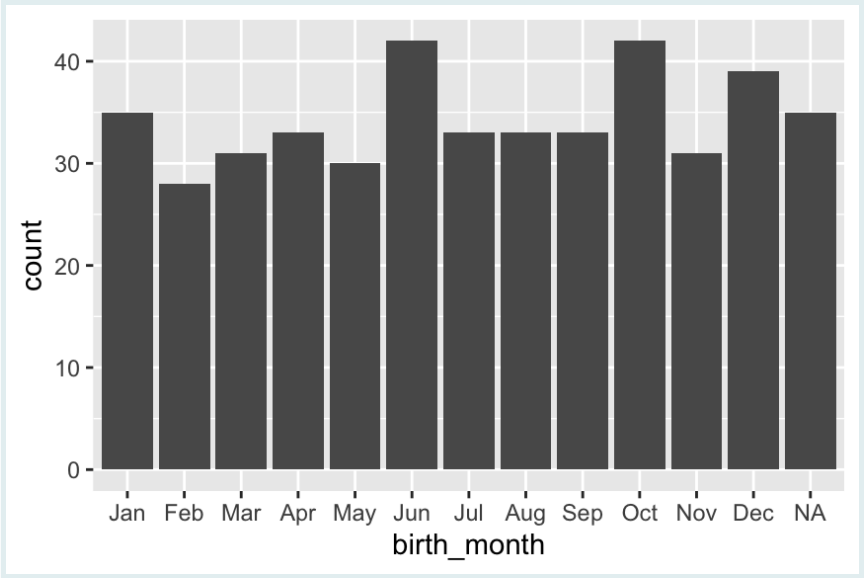
```
## [1] "factor"
```

```
# Original dataset
class(hiv_mort$birth_month)
```

```
## [1] "character"
```

Now we can regenerate the ggplot with the modified dataset:

```
ggplot(hiv_mort_modified) +
  geom_bar(aes(x = birth_month))
```



The months on the x-axis are now displayed in the order we specified.

The new factor variable will respect the defined order in other contexts as well. For example, compare how the `count()` function displays the two frequency tables below:

```
# Original dataset
count(hiv_mort, birth_month)
```

```
## # A tibble: 13 × 2
##    birth_month     n
##    <chr>       <int>
##  1 Apr            33
##  2 Aug            33
##  3 Dec            39
##  4 Feb            28
##  5 Jan            35
##  6 Jul            33
##  7 Jun            42
##  8 Mar            31
##  9 May            30
## 10 Nov            31
## 11 Oct            42
## 12 Sep            33
## 13 <NA>           35
```

```
# Modified dataset
count(hiv_mort_modified, birth_month)
```

```
## # A tibble: 13 × 2
##    birth_month     n
```

```
##    <fct>       <int>
##  1 Jan            35
##  2 Feb            28
##  3 Mar            31
##  4 Apr            33
##  5 May            30
##  6 Jun            42
##  7 Jul            33
##  8 Aug            33
##  9 Sep            33
## 10 Oct            42
## 11 Nov            31
## 12 Dec            39
## 13 <NA>           35
```

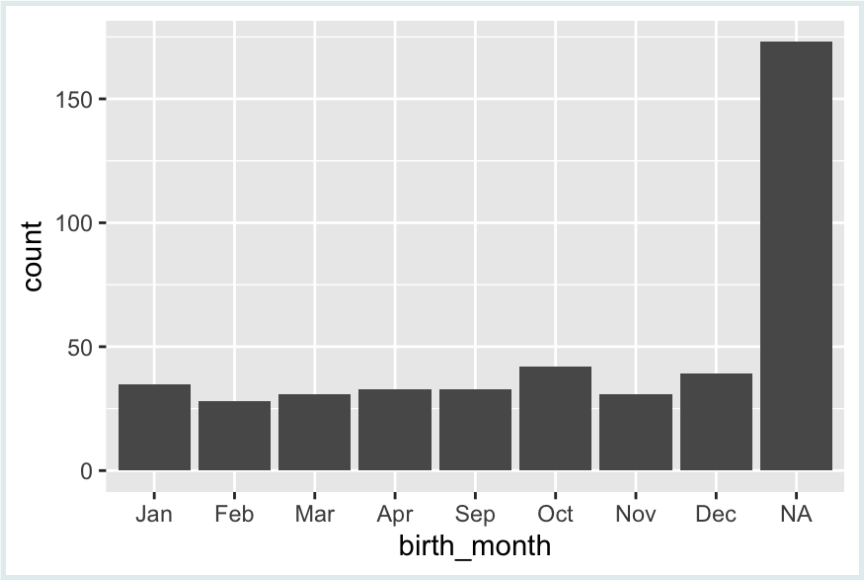
Be mindful when creating factor levels! Any values in the variable that are *not* included in the set of levels provided to the `levels` argument will be converted to NA.

For instance, if we missed some months in our example:

```
hiv_mort_missing_months <-
  hiv_mort %>%
  mutate(birth_month = factor(x = birth_month,
                              levels = c("Jan", "Feb", "Mar",
         "Apr",
                                         # missing months
                                        "Sep", "Oct", "Nov",
         "Dec")))
```

We end up with a lot of NA values:

```
ggplot(hiv_mort_missing_months) +
  geom_bar(aes(x = birth_month))
```

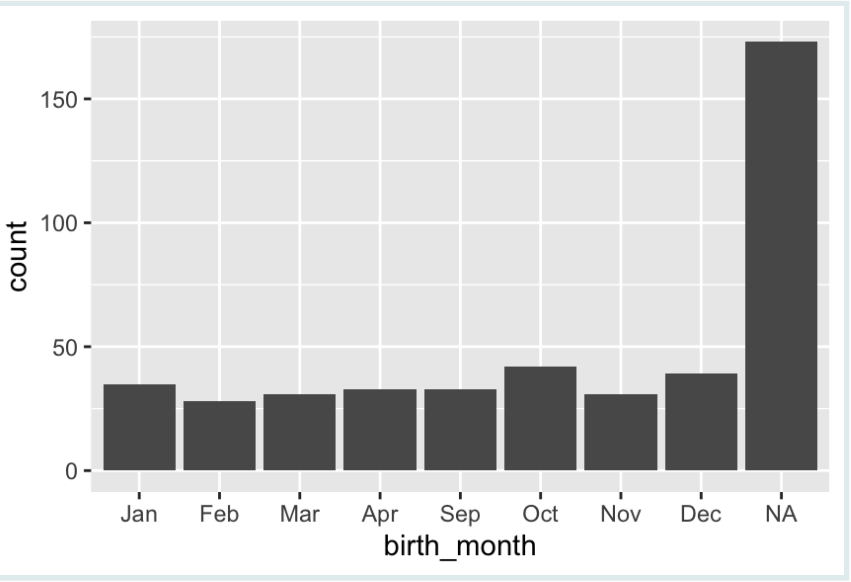


You will have the same problem if there are typographical errors:

```
hiv_mort_with_typos <-
  hiv_mort %>%
  mutate(birth_month = factor(x = birth_month,
                              levels = c("Jan", "Feb", "Mar",
         "Apr",
                                         "Moy", "Jon", "Jol",
         "Aog", # typos
                                         "Sep", "Oct", "Nov",
         "Dec")))
```

```
ggplot(hiv_mort_with_typos) +
  geom_bar(aes(x = birth_month))
```

WATCH OUT



SIDE NOTE

You can use factor without levels. It just uses default (alphabetical) arrangement of levels

```
hiv_mort_default_factor <- hiv_mort %>%
  mutate(birth_month = factor(x = birth_month))
```

```
class(hiv_mort_default_factor$birth_month)
```

```
## [1] "factor"
```

```
levels(hiv_mort_default_factor$birth_month)
```

```
##  [1] "Apr" "Aug" "Dec" "Feb" "Jan" "Jul" "Jun" "Mar" "May"
"Nov" "Oct" "Sep"
```

Q: Gender factor

Using the `hiv_mort` dataset, convert the `gender` variable to a factor with the levels "Female" and "Male", in that order.

Q: Error spotting

What errors are you able to spot in the following code chunk? What are the consequences of these errors?

```
hiv_mort <-
  hiv_mort %>%
  mutate(birth_month = factor (x = birth_month,
                               levels = c("Jan", "Feb", "Mar", "Apr",
                                          "Mai", "Jun", "Jul", "Sep",
                                          "Oct", "Nov.", "Dec")))
```

Q: Advantage of factors

What is one main advantage of using factors over characters for categorical data in R?

a. It is easier to perform string manipulation on factors.
b. Factors allow better control over ordering of categorical data.
c. Factors increase the accuracy of statistical models.

Manipulating Factors with `forcats`

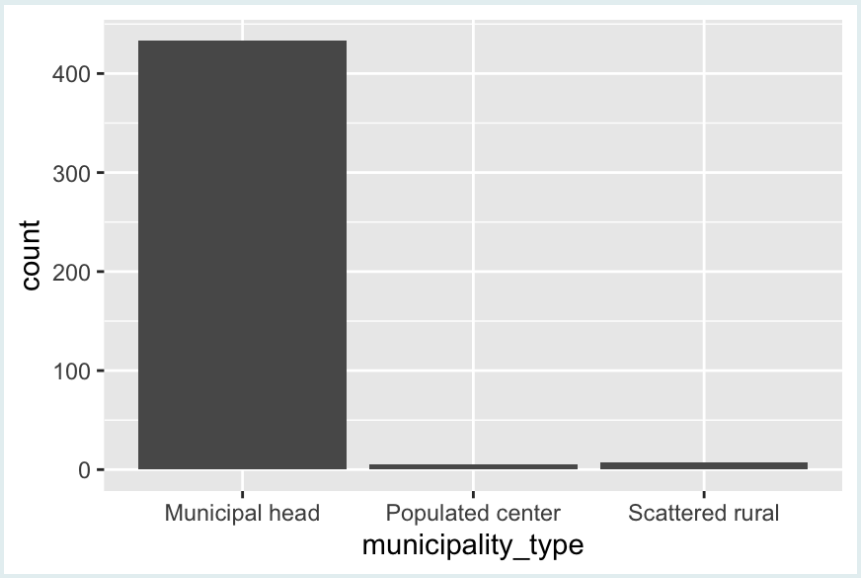
Factors are very useful, but they can sometimes be a little tedious to manipulate using base R functions alone. Thankfully, the `forcats` package, a member of the tidyverse, offers a set of functions that make factor manipulation much simpler. We'll consider four functions here, but there are many others, so we encourage you to explore the *forcats* website on your own time here!

fct_relevel

The `fct_relevel()` function is used to manually change the order of factor levels.

For example, let's say we want to visualize the frequency of individuals in our dataset by municipality type. When we create a bar plot, the values are ordered alphabetically by default:

```
ggplot(hiv_mort) +
  geom_bar(aes(x = municipality_type))
```



But what if we want a specific value, say "Populated center", to appear first in the plot?

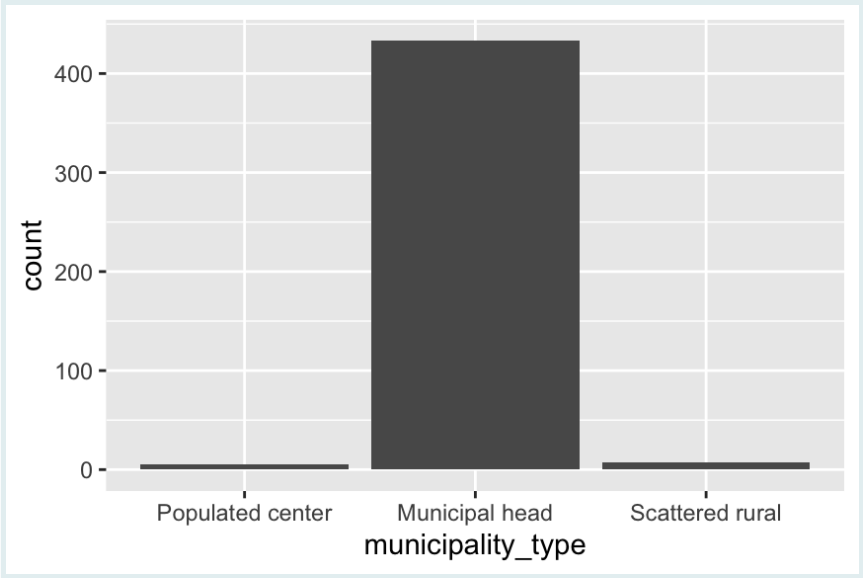
This can be achieved using `fct_relevel()`. Here's how:

```
hiv_mort_pop_center_first <-
  hiv_mort %>%
  mutate(municipality_type = fct_relevel(municipality_type, "Populated
         center"))
```

The syntax is straightforward: we pass the factor variable as the first argument, and the level we want to move to the front as the second argument.

Now when we plot:

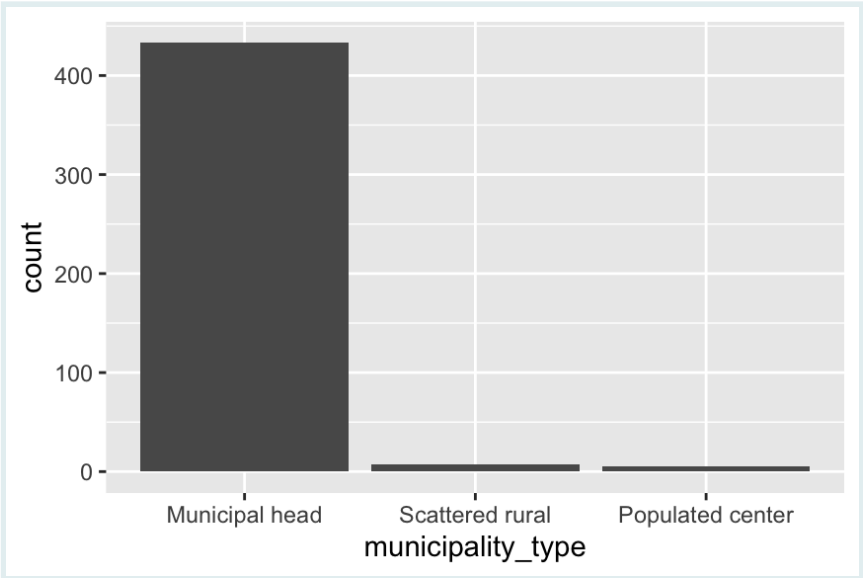
```
ggplot(hiv_mort_pop_center_first) +
  geom_bar(aes(x = municipality_type))
```



The "Populated center" level is now first.

We can move the "Populated center" level to a different position with the `after` argument:

```
hiv_mort %>%
  mutate(municipality_type = fct_relevel(municipality_type, "Populated
         center",
                                         after = 2)) %>%
 # pipe directly into to plot to visualize change
  ggplot() +
  geom_bar(aes(x = municipality_type))
```

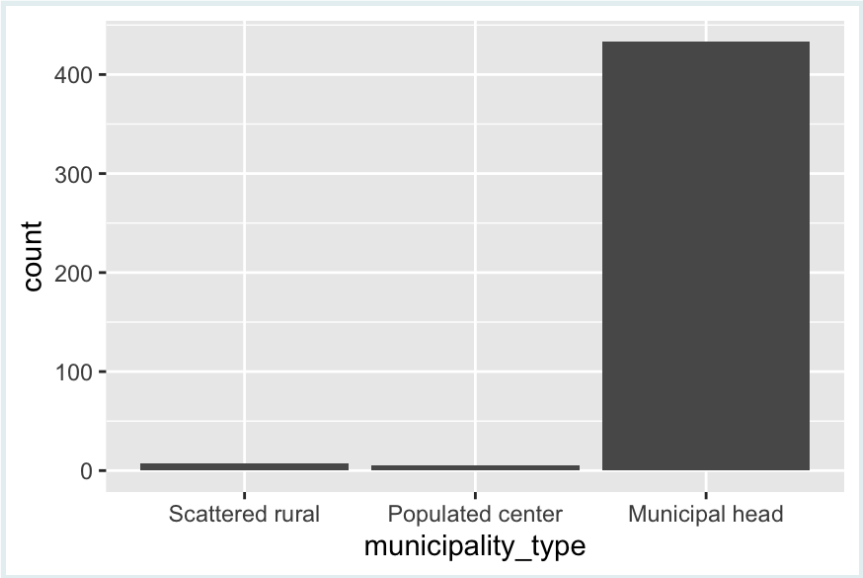


The syntax is: specify the factor, the level to move, and use the `after` argument to define what position to place it after.

We can also move multiple levels at a time by providing these levels to `fct_relevel()`:

Below we arrange all the factor levels for municipality type in our desired order:

```
hiv_mort %>%
  mutate(municipality_type = fct_relevel(municipality_type,
                                         "Scattered rural",
                                         "Populated center",
                                         "Municipal head")) %>%
  ggplot() +
  geom_bar(aes(x = municipality_type))
```



This is similar to creating a factor from scratch with levels in that order:

```
hiv_mort %>%
  mutate(municipality = factor(municipality_type,
                               levels = c("Scattered rural",
                                          "Populated center",
                                          "Municipal head")))
```

Q: Using fct_relevel

Using the `hiv_mort` dataset, convert the `death_location` variable to a factor such that 'Home/address' is the first level. Then create a bar plot that shows the count of individuals in the dataset by `death_location`.

fct_reorder

`fct_reorder()` is used to reorder the levels of a factor based on the values of another variable.

To illustrate, let's make a summary table with number of deaths, mean and median age at death for each municipality:

```
summary_per_muni <-
  hiv_mort %>%
  group_by(municipality_name) %>%
  summarise(n_deceased = n(),
            mean_age_death = mean(age_at_death, na.rm = T),
            med_age_death = median(age_at_death, na.rm = T))

summary_per_muni
```

```
## # A tibble: 25 × 4
##    municipality_name n_deceased mean_age_death med_age_death
##    <chr>                  <int>          <dbl>         <dbl>
##  1 Aguadas                    2           42            42
##  2 Anserma                   15           37.4          37.5
##  3 Aranzazu                   2           37.5          37.5
##  4 Belalcázar                 4           38.8          41
##  5 Chinchiná                 62           43.6          42.5
##  6 Filadelfia                 5           42.6          43
##  7 La Dorada                 46           41.0          41
##  8 La Merced                  3           27            28
##  9 Manizales                199           41.0          41
## 10 Manzanares                 3           38.3          34
## # ℹ 15 more rows
```

When plotting one of the variables, we may want to arrange the factor levels by that numeric variable. For example, to order municipality by the mean age column:

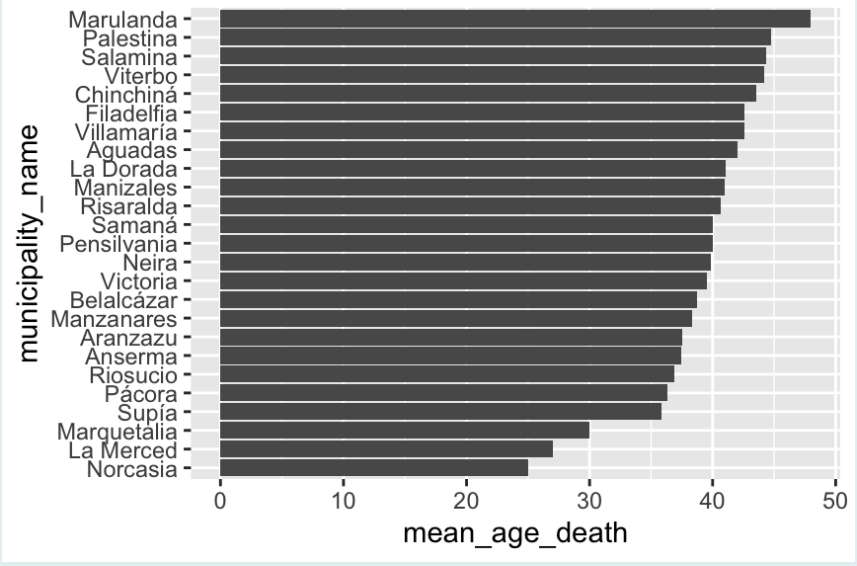
```
summary_per_muni_reordered <-
  summary_per_muni %>%
  mutate(municipality_name = fct_reorder(.f = municipality_name,
                                         .x = mean_age_death))
```

The syntax is:

- `.f` - the factor to reorder
- `.x` - the numeric vector determining the new order

We can now plot a nicely arranged bar chart:

```
ggplot(summary_per_muni_reordered) +
  geom_col(aes(y = municipality_name, x = mean_age_death))
```



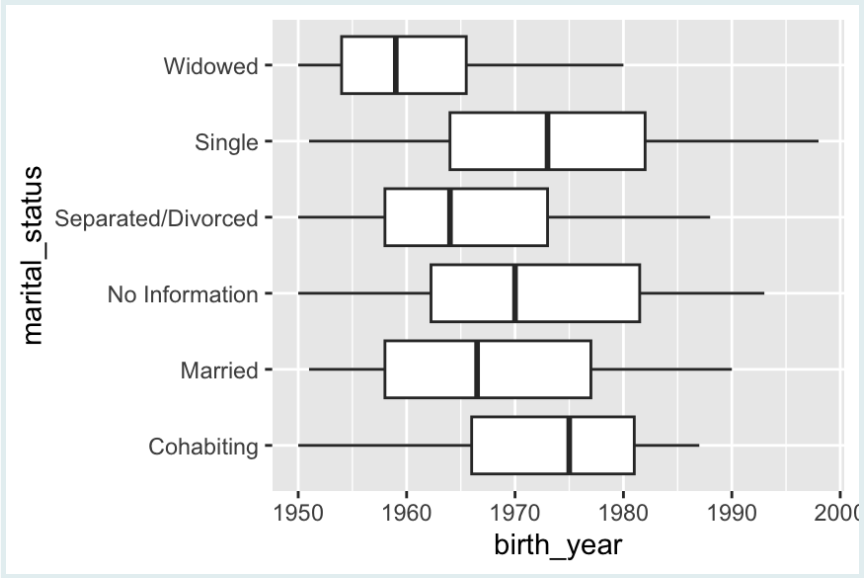
Q: Using fct_reorder

Starting with the `summary_per_muni` data frame, reorder the municipality (`municipality_name`) by the `med_age_death` column and plot the reordered bar chart.

The `.fun` argument

Sometimes we want the categories in our plot to appear in a specific order that is determined by a summary statistic. For example, consider the box plot of `birth_year` by `marital_status`:

```
ggplot(hiv_mort, aes(y = marital_status, x = birth_year)) +
  geom_boxplot()
```



The boxplot displays the median `birth_year` for each category of marital status as a line in the middle of each box. We might want to arrange the marital_status categories in order of these medians. But if we create a summary table with medians, like we did before with `summary_per_muni`, we can't create a box plot with it (go look at the `summary_per_muni` data frame to verify this yourself).

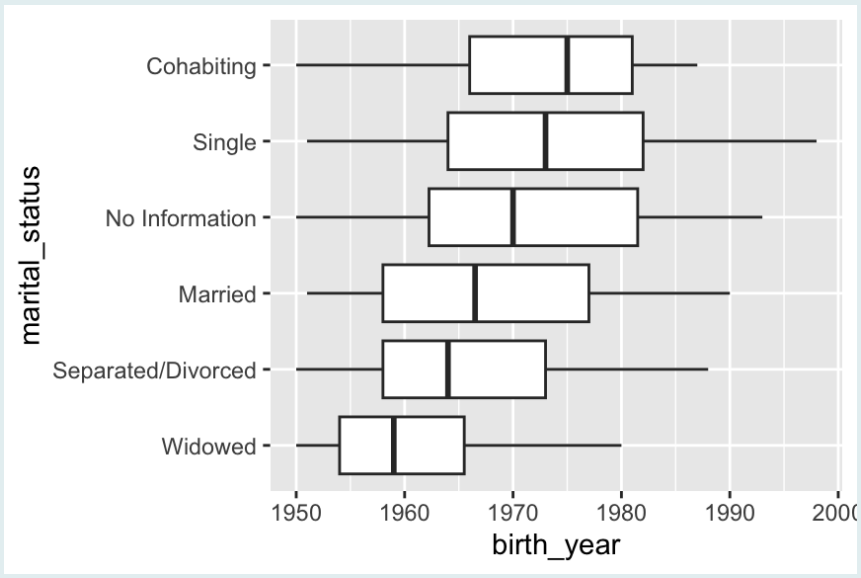
This is where the `.fun` argument of `fct_reorder()` comes in. The `.fun` argument allows us to specify a summary function that will be used to calculate the new order of the levels:

```
hiv_mort_arranged_marital <-
  hiv_mort %>%
  mutate(marital_status = fct_reorder(.f = marital_status,
                                      .x = birth_year,
                                      .fun = median,
                                      na.rm = TRUE))
```

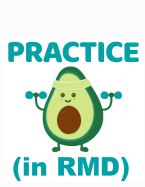
In this code, we are reordering the `marital_status` factor based on the median of `birth_year`. We include the argument `na.rm = TRUE` to ignore NA values when calculating the median.

Now, when we create our box plot, the `marital_status` categories are ordered by the median `birth_year`:

```
ggplot(hiv_mort_arranged_marital, aes(y = marital_status, x = birth_year)) +
  geom_boxplot()
```



We can see that individuals with the marital status "cohabiting" tend to be the youngest (they were born in the latest years).



Q: Using .fun

Using the `hiv_mort` dataset, make a boxplot of `birth_year` by `health_insurance_status`, where the `health_insurance_status` categories are arranged by the median `birth_year`.

fct_recode

The `fct_recode()` function allows us to manually change the values of factor levels. This function can be especially helpful when you need to rename categories or when you want to merge multiple categories into one.

For example, we can rename 'Municipal head' to 'City' in the `municipality_type` variable:

```
hiv_mort_muni_recode <- hiv_mort %>%
  mutate(municipality_type = fct_recode(municipality_type,
                                        "City" = "Municipal head"))

# View the change
levels(hiv_mort_muni_recode$municipality_type)
```

```
## [1] "City"             "Populated center" "Scattered rural"
```

In the above code, `fct_recode()` takes two arguments: the factor variable you want to change (`municipality_type`), and the set of name-value pairs that define the recoding. The new level ("City") is on the left of the equals sign, and the old level ("Municipal head") is on the right.

`fct_recode()` is particularly useful for compressing multiple categories into fewer levels:

We can explore this using the education_level variable. Currently it has six categories:

```
count(hiv_mort, education_level)
```

```
## # A tibble: 6 × 2
##   education_level     n
##   <chr>           <int>
## 1 No information     88
## 2 None               22
## 3 Post-secondary     29
## 4 Preschool           3
## 5 Primary           187
## 6 Secondary         116
```

For simplicity, let's group them into just three categories - primary & below, secondary & above and other:

```
hiv_mort_educ_simple <-
  hiv_mort %>%
  mutate(education_level = fct_recode(education_level,
                                      "primary & below" = "Primary",
                                      "primary & below" = "Preschool",
                                      "secondary & above" = "Secondary",
                                      "secondary & above" = "Post-
        secondary",
                                      "others" = "No information",
                                      "others" = "None"))
```

This condenses the categories nicely:

```
count(hiv_mort_educ_simple, education_level)
```

```
## # A tibble: 3 × 2
##   education_level       n
##   <fct>             <int>
## 1 others              110
## 2 secondary & above   145
## 3 primary & below     190
```

For good measure, we can arrange the levels in a reasonable order, with "others" as the last level:

```
hiv_mort_educ_sorted <-
  hiv_mort_educ_simple %>%
  mutate(education_level = fct_relevel(education_level,
                                       "primary & below",
                                       "secondary & above",
                                       "others"))
```

This condenses the categories nicely:

```
count(hiv_mort_educ_sorted, education_level)
```

```
## # A tibble: 3 × 2
##   education_level       n
##   <fct>             <int>
## 1 primary & below     190
## 2 secondary & above   145
## 3 others              110
```

Q: Using fct_recode

Using the `hiv_mort` dataset, convert death_location to a factor.

Then use `fct_recode()` to rename 'Public way' in death_location to 'Public place'. Plot the frequency counts of the updated variable.

fct_recode vs case_when/if_else

You might question why we need `fct_recode()` when we can utilize `case_when()` or `if_else()` or even `recode()` to substitute specific values. The issue is that these other functions can disrupt your factor variable.

To illustrate, let's say we choose to use `case_when()` to make a modification to the `education_level` variable of the `hiv_mort_educ_sorted` data frame.

As a quick reminder, that the `education_level` variable is a factor with three levels, arranged in a specified order, with "primary & below" first and "others" last:

```
count(hiv_mort_educ_sorted, education_level)
```

```
## # A tibble: 3 × 2
##   education_level       n
##   <fct>             <int>
## 1 primary & below     190
## 2 secondary & above   145
## 3 others              110
```

Say we wanted to replace the "others" with "other", removing the "s". We can write:

SIDE NOTE

```
hiv_mort_educ_other <-
  hiv_mort_educ_sorted %>%
  mutate(education_level = if_else(education_level ==
         "others",
                                   "other", education_level))
```

After this operation, the variable is no longer a factor:

```
class(hiv_mort_educ_other$education_level)
```

```
## [1] "character"
```

If we then create a table or plot, our order is disrupted and reverts to alphabetical order, with "other" as the first level:

```
count(hiv_mort_educ_other, education_level)
```

```
## # A tibble: 3 × 2
##   education_level       n
##   <chr>             <int>
## 1 other               110
## 2 primary & below     190
## 3 secondary & above   145
```

However, if we had used `fct_recode()` for recoding, we wouldn't face this issue:

```
hiv_mort_educ_other_fct <-
  hiv_mort_educ_simple %>%
  mutate(education_level = fct_recode(education_level, "other"
         = "others"))
```

The variable remains a factor:

```
class(hiv_mort_educ_other_fct$education_level)
```

SIDE NOTE

```
## [1] "factor"
```

And if we create a table or a plot, our order is preserved: primary, secondary, then other:

```
count(hiv_mort_educ_other_fct, education_level)
```

```
## # A tibble: 3 × 2
##   education_level       n
##   <fct>             <int>
## 1 other               110
## 2 secondary & above   145
## 3 primary & below     190
```

fct_lump

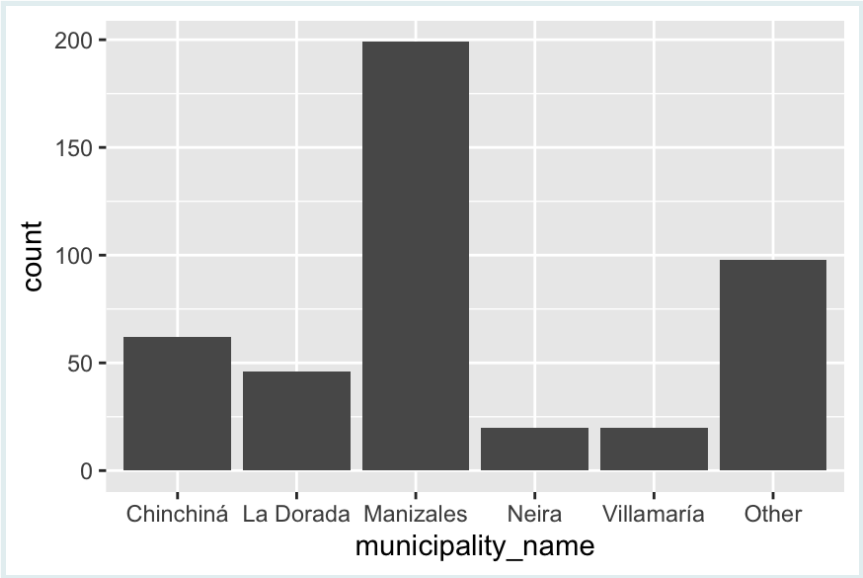
Sometimes, we have too many levels for a display table or plot, and we want to lump the least frequent levels into a single category, typically called 'Other'.

This is where the convenience function `fct_lump()` comes in.

In the below example, we lump less frequent municipalities into 'Other', preserving just the top 5 most frequent municipalities:

```
hiv_mort_lump_muni <- hiv_mort %>%
  mutate(municipality_name = fct_lump(municipality_name, n = 5))
```

```
ggplot(hiv_mort_lump_muni, aes(x = municipality_name)) +  geom_bar()
```

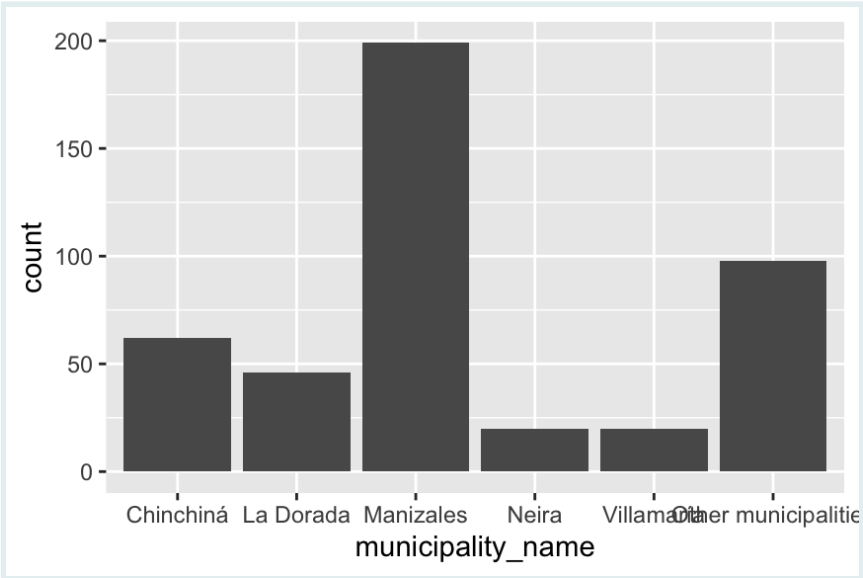


In the usage above, the parameter `n = 5` means that the five most frequent municipalities are preserved, and the rest are lumped into 'Other'.

We can provide a custom name for the other category with the `other_level` argument. Below we use the name `"Other municipalities"`.

```
hiv_mort_lump_muni_other_name <- hiv_mort %>%
  mutate(municipality_name = fct_lump(municipality_name, n = 5,
                                      other_level = "Other municipalities"))

ggplot(hiv_mort_lump_muni_other_name, aes(x = municipality_name)) +
  geom_bar()
```



In this way, `fct_lump()` is a handy tool for condensing factors with many infrequent levels into a more manageable number of categories.

Q: Using fct_lump

Starting with the `hiv_mort` dataset, use `fct_lump()` to create a bar chart with the frequency of the 10 most common occupations.

Lump the remaining occupation into an 'Other' category.

Put `occupation` on the y-axis, not the x-axis, to avoid label overlap.

Wrap up

Congrats on getting to the end. In this lesson, you learned details about the data class, **factors**, and how to manipulate them using basic operations such as `fct_relevel()`, `fct_reorder()` `fct_recode()`, and `fct_lump()`.

While these covered common tasks such as reordering, recoding, and collapsing levels, this introduction only scratches the surface of what's possible with the forcats package. Do explore more on the forcats website.

Now that you understand the basics of working with factors, you are equipped to properly represent your categorical data in R for downstream analysis and visualization.

Answer Key

Q: Gender factor

```
hiv_mort_q1 <- hiv_mort %>%
  mutate(gender = factor(x = gender,
                         levels = c("Female", "Male")))
```

Q: Error spotting

Errors:

- "Mai" should be "May".
- "Nov." has an extra period.
- "Aug" is missing from the list of months.

Consequences:

Any rows with the values "May", "Nov", or "Aug" for death_month will be converted to NA in the new death_month variable. If you create plots, ggplot will drop these levels with only NA values.

Q: Advantage of factors

b. Factors allow better control over ordering of categorical data.

The other two statements are not true.

If you want to apply string operations like substr(), strsplit(), paste(), etc., it's actually more straightforward to use character vectors than factors.

And while many statistical functions expect factors, not characters, for categorical predictors, this does not make them more "accurate".

Q: Using fct_relevel

Q: Using fct_reorder

Q: Using .fun

Q: Using fct_recode

Q: Using fct_lump

Appendix: Codebook

The variables in the dataset are:

- `municipality`: general municipal location of the patient [chr]
- `death_location`: location where the patient died [chr]

- `birth_date`: full date of birth, formatted "YYYY-MM-DD" [date]
- `birth_year`: year when the patient was born [dbl]
- `birth_month`: month when the patient was born [chr]
- `birth_day`: day when the patient was born [dbl]
- `death_year`: year when the patient died [dbl]
- `death_month`: month when the patient died [chr]
- `death_day`: day when the patient died [dbl]
- `gender`: gender of the patient [chr]
- `education_level`: highest level of education attained by patient [chr]
- `occupation`: occupation of patient [chr]
- `racial_id`: race of the patient [chr]
- `municipality_code`: specific municipal location of the patient [chr]
- `primary_cause_death_description`: primary cause of the patient's death [chr]
- `primary_cause_death_code`: code of the primary cause of death [chr]
- `secondary_cause_death_description`: secondary cause of the patient's death [chr]
- `secondary_cause_death_code`: code of the secondary cause of death [chr]
- `tertiary_cause_death_description`: tertiary cause of the patient's death [chr]
- `tertiary_cause_death_code`: code of the tertiary cause of death [chr]
- `quaternary_cause_death_description`: quaternary cause of the patient's death [chr]
- `quaternary_cause_death_code`: code of the quaternary cause of death [chr]

Contributors

The following team members contributed to this lesson:

CAMILLE BEATRICE VALERA

Project Manager and Scientific Collaborator, The GRAPH Network

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement