Factors

forcats package

Federica Fusi

MSCA, UIC

Updated: 2021-03-16

Final projects

How to set them up?

- Questions of interest to you but of relevance for public policy.

Outputs:

- a R source file with appropriate comments and documentation to explain your code. The R file should be perfectly reproducible by me and your TA.
- A final report containing both a descriptive analysis of data and appropriate data visualization.
- A methodology write-up (this can be an appendix to your report) detailing data source, data characteristics, and limitations.

You can produce either PDF documents or HTML pages created with R markdown.

Set up

Due March 29th: A one-page proposal

Due April 20th: Project workshop in class [presentations are cancelled]. Submit a preliminary draft of your work.

Due May 4th: Final submission

Question: group project vs pairs vs individual projects?

FACTORS

What are factors?

Factors are categorical variables. Categorical variables are variables that can only assume a fixed set of known possible values. Examples:

- How do you get to school? [Bus][Train][Car][Walking][Metro]
- What is your race? [White][Black/African American][Asian][Native American/Alaskan Native]
- What is your current position? [Undergraduate student][Graduate student][Professor][Staff]

In R, the possible categories are called levels. When you visualize a factor variable, you will see both the actual values of the variable AND its levels.

Quick overview

Let's create a factor storing information about someone's marital status.

```
fact_1 = as_factor(c("Married",
                     "Never Married",
                     "Divorced",
                     "Married",
                     "Widowed",
                     "Widowed",
                     "Divorced",
                     "Divorced"))
```

When we visualize the variable, we are going to see its values + its levels

```
fact_1
```

A quicker way to see the levels is by using the function levels.

```
# Note that levels are automatically organized in alphabetical order
levels(fact_1)
```

Factors

Why do factors exist?

Categorical variables are different than continuous variables within regression analysis. They are treated as a set of dummy variables.

Because of this, R was created with a *factor* variable type to facilitate regression. Several functions still convert character variables into factors.

What to do with factors?

So far, we have occasionally transformed factors into character or numeric variable. Remember the conversion system

Transforming factors

Transforming a factor into a character variable is not problematic.

We can directly transform a factor into a character variable.

```
fact_1_R = as.character(fact_1)

class(fact_1_R)
```

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

```
fact_1_R # All values are preserved
```

```
## [1] "Married"       "Never Married" "Divorced"      "Married"
## [5] "Widowed"       "Widowed"       "Divorced"      "Divorced"
```

Transforming factors

Transforming a factor into a numeric variable is tricky

```
#Let's create a factor with numbers in it
fact_2 = as_factor(c(4, 5, 6, 7, 4, 6, 7))

#We transform it directly into a numeric variable
fact_2_R = as.numeric(fact_2)

#Values are not preserved. Instead levels order is reported
fact_2_R
```

```
## [1] 1 2 3 4 1 3 4
```

```
# See what happened
levels(fact_2)
```

```
## [1] "4" "5" "6" "7"
```

Transforming factors

We need to use a different formula here...

```
fact_2_R = as.numeric(levels(fact_2))[fact_2]

fact_2_R
```

```
## [1] 4 5 6 7 4 6 7
```

Transforming factors into character or numeric variable was helpful as we hadn't talked yet about how to work with factors.

It can also be appropriate if your variables are, indeed, character or numberic variables.

Dataframes (remember them?) tend to transform character and numeric variables into factors by default. So you need to pay attention.

Tibbles address this issue by not transforming variables into factors by default.

forcats package

From now on, we are going to work with factors when working with categorical variables by using the forcats package.

forcats (anagram of factors!) is included in tidyverse, so you don't need to install it! (pro tip: ggplot2 and dplyr are also included in tidyverse).

Advantages of using factors:

- Appropriate when dealing with categorical variables.
- Prevent wrong categories from being inserted in the dataset
- Maintain a given order
- Easy to re-order for plots.

Data

We are going to use a new dataset from the stevedata package

```
library("stevedata")
data <- gss_wages

data
```

> "Wage data from the General Social Survey (1974-2018) to illustrate wage discrepancies by gender (while also considering respondent occupation, age, and education)."

A few basics

Let's have a look at the maritalcat variable which indicate the marital status of the respondent.

```
class(data$maritalcat) # this is a character variable
```

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

```
table(data$maritalcat) # Let's see the categories
```

```
##
##      Divorced       Married Never Married     Separated       Widowed
##          8254         31893         13434          2151          5938
```

Since there is a set of predefined categories, this is a factor not a real character (text) variable.

Transform into factors

If we want to make sure that no category is inserted by mistake, we can set up our levels (categories) first and then transform the variable into a factor.

```
# Create a vector containing all these categories
marital_levels = c("Divorced",
                   "Married",
                   "Never Married",
                   "Separated",
                   "Widowed")

# Convert the variable into a factor with those levels
data$maritalcat_F <- factor(data$maritalcat,
                            levels = marital_levels)
```

Transform into factors

See what happens if we omit one category

```
marital_levels = c("Divorced",
                   "Married",
                   "Never Married",
                   "Separated")

# It gets replaced by NAs
data$maritalcat_F2 <- factor(data$maritalcat,
                             levels = marital_levels)
```

How do you check levels?

```
# look at the level
levels(data$maritalcat_F)
```

```
## [1] "Divorced"      "Married"       "Never Married" "Separated"
## [5] "Widowed"
```

```
# Table in dplyr
data %>%
 count(maritalcat_F)
```

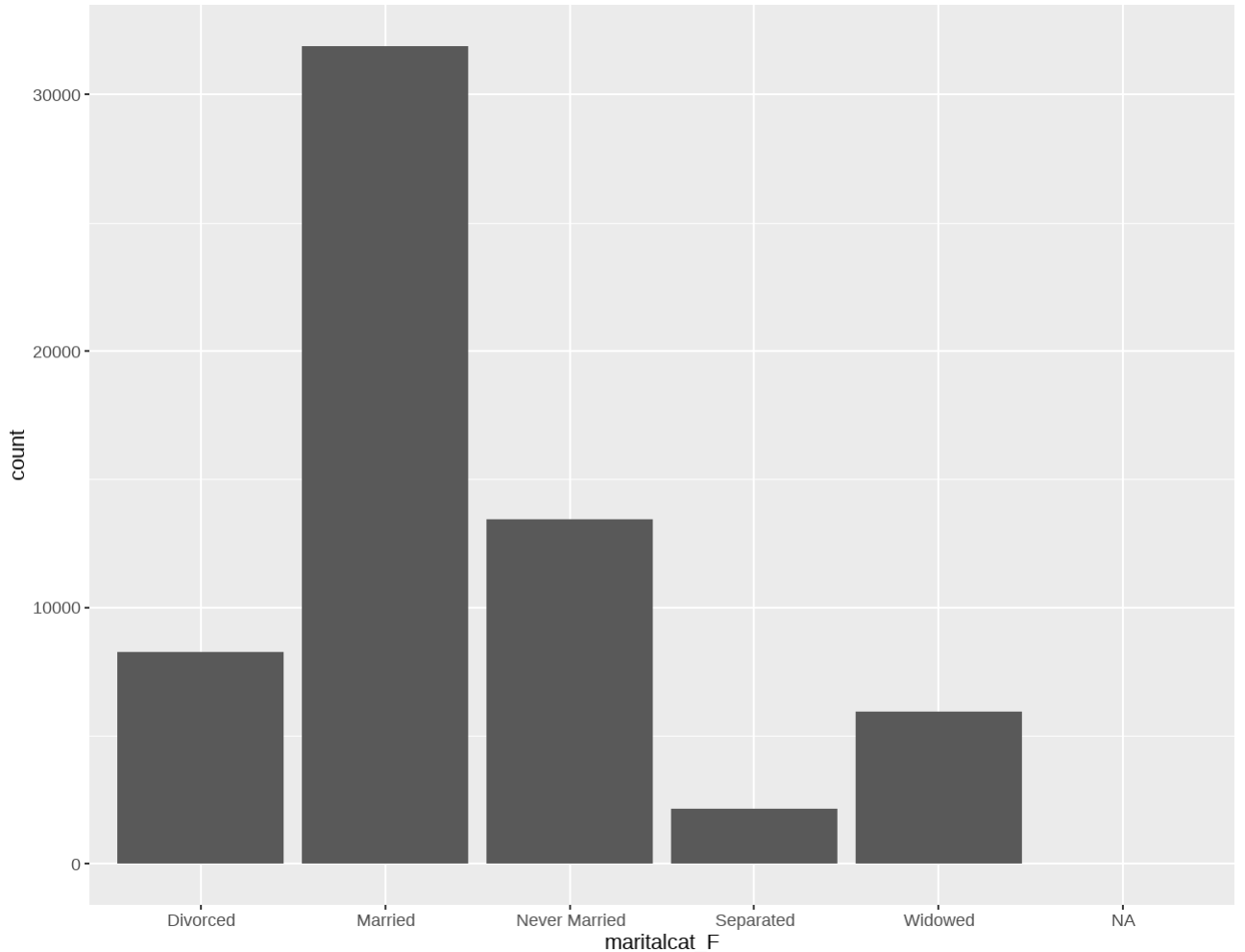
```
## # A tibble: 6 x 2
##   maritalcat_F      n
## * <fct>         <int>
## 1 Divorced       8254
## 2 Married       31893
## 3 Never Married 13434
## 4 Separated      2151
## 5 Widowed        5938
## 6 <NA>             27
```

Note that levels are again presented in the alphabetical order.

How do you check levels?

```
plot_1 =
data %>%
ggplot() +
  geom_bar(mapping = aes(maritalcat_F))
```

```
plot_1
```



Note that the order of the bars corresponds to the order of the levels.

Order of levels

fct_relevel

In some cases, you might want the levels to have a specific order.

```
data =
  data %>%
  mutate(maritalcat_F =
           fct_relevel(maritalcat_F,
                       "Married",
                       "Separated",
                       "Divorced",
                       "Widowed",
                       "Never Married"))

levels(data$maritalcat_F)
```

```
## [1] "Married"       "Separated"     "Divorced"      "Widowed"
## [5] "Never Married"
```

fct_relevel

Or simply move one level to a specific position - let's say the last one:

```
data =
  data %>%
  mutate(maritalcat_F =
           fct_relevel(maritalcat_F,
                       "Married",
                       after = 5))

levels(data$maritalcat_F)
```

```
## [1] "Separated"     "Divorced"      "Widowed"       "Never Married"
## [5] "Married"
```

fct_relevel

Now we want to get back to the alphabetical order

```
data =
  data %>%
  mutate(maritalcat_F =
           fct_relevel(maritalcat_F,
                       sort))

levels(data$maritalcat_F)
```

```
## [1] "Divorced"      "Married"       "Never Married" "Separated"
## [5] "Widowed"
```

fct_inorder

In some cases, you might match your levels with their order of appearance in the data.

```
data =
data %>%
  mutate(maritalcat_F2 =
           fct_inorder(maritalcat_F))

levels(data$maritalcat_F)
```

```
## [1] "Divorced"      "Married"       "Never Married" "Separated"
## [5] "Widowed"
```

```
levels(data$maritalcat_F2)
```

```
## [1] "Married"       "Widowed"       "Never Married" "Divorced"
## [5] "Separated"
```

fct_infreq

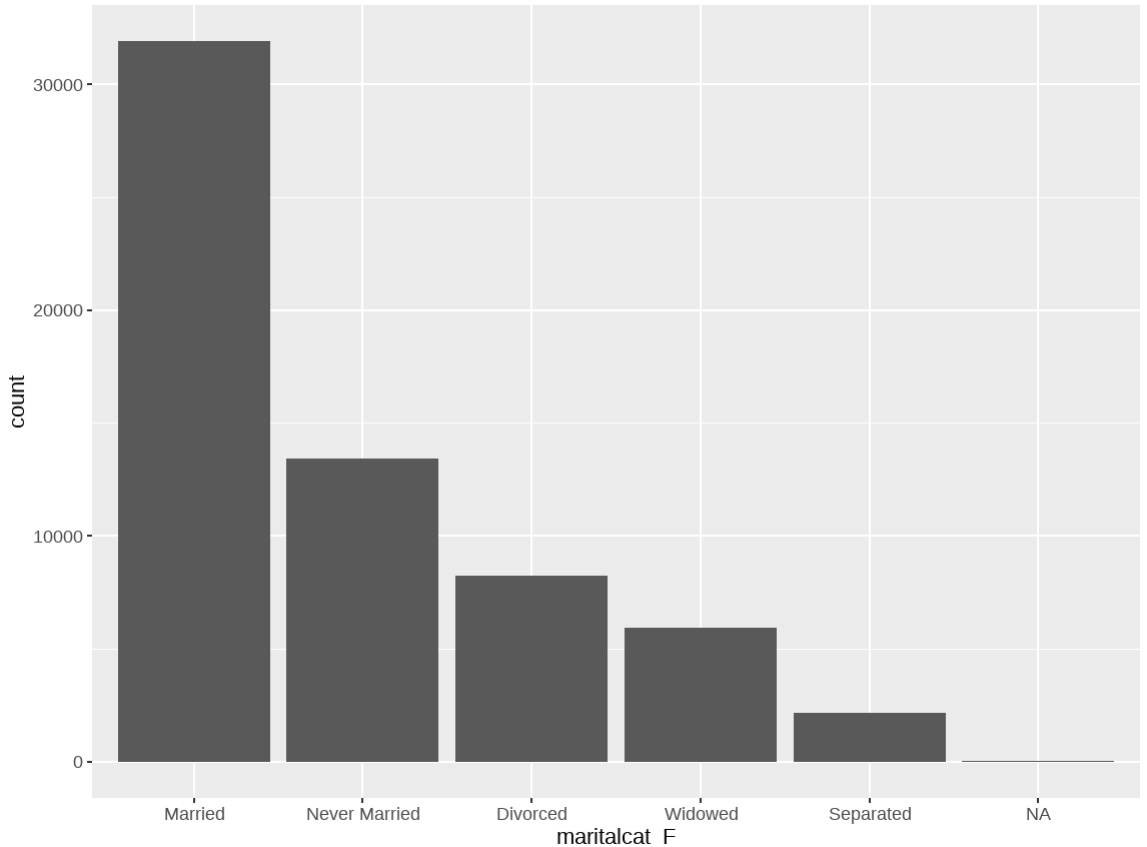
In others, you might want to order levels from the most frequent to the least frequent category (or vice versa)

```
# From most frequent to least frequent

plot_2 =
data %>%
  mutate(maritalcat_F =
           fct_infreq(maritalcat_F)) %>%
  ggplot() +
  geom_bar(mapping = aes(maritalcat_F))
```

fct_infreq

```
plot_2
```



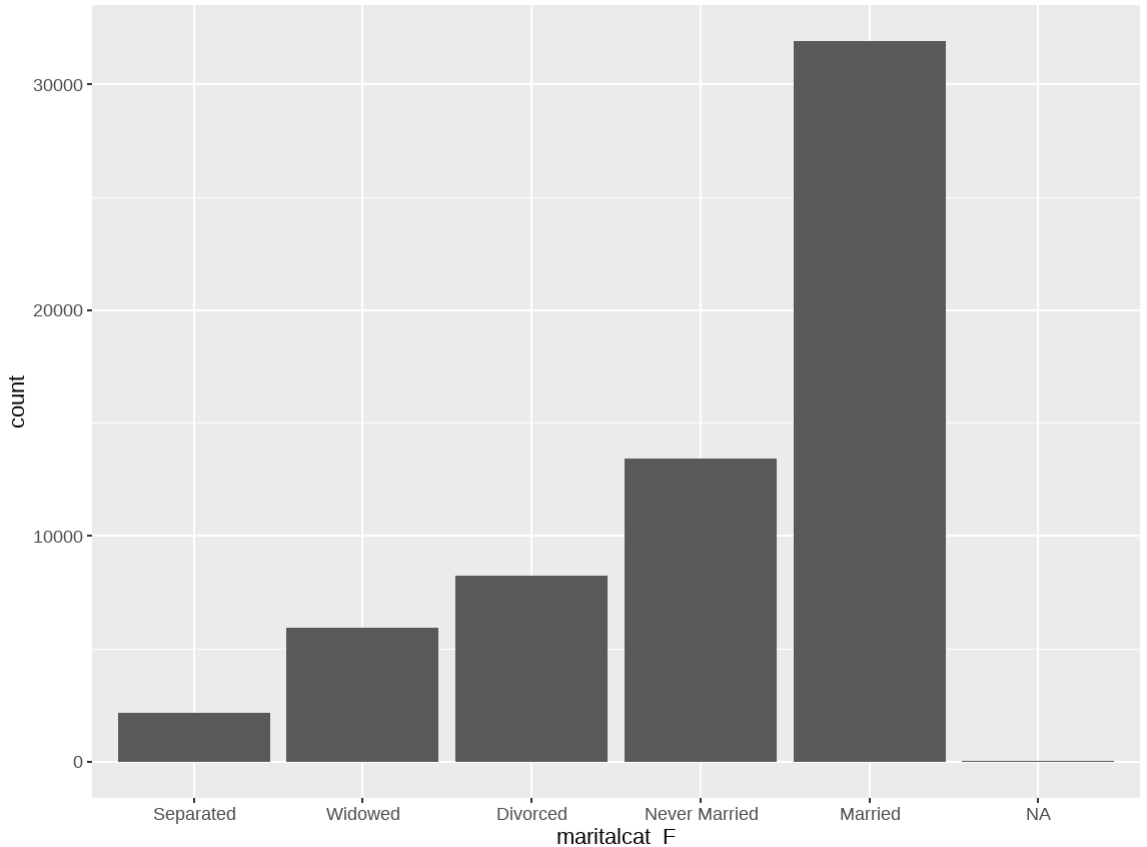
fct_inv

Or, from least frequent to most frequent:

```
plot_3 = data %>%
  mutate(maritalcat_F =
           fct_rev(
             fct_infreq(data$maritalcat_F))) %>%
  ggplot() +
  geom_bar(mapping = aes(maritalcat_F))
```

fct_inv

```
plot_3
```



Order by variables

When working with plots, it happens pretty often to want to order a factor's level based off another variable.

For instance, we might want to plot the average level of income (a continuous variable) by education level (a categorical variable). We could use a barplot here.

Give it a try using factors as much as possible (i.e., convert character variables into factors when appropriate).

fct_reorder

```
#Create a vector containing all these categories
educcat_levels = c("Bachelor",
                   "Graduate",
                   "High School",
                   "Junior College",
                   "Less Than High School")

# Convert the variable into a factor with those levels
data$educcat_F <- factor(data$educcat,
                         levels = educcat_levels)

# Create my plot
plot_4 =
data %>%
  filter(!is.na(educcat_F)) %>%
  group_by(educcat_F) %>%
  summarize(income_mean = mean(realrinc, na.rm = T)) %>%
  ggplot() +
  geom_bar(aes(income_mean, educcat_F),
           stat = "identity")
```

Note: stat = "identity"

In a barplot, the default is to plot the frequency of a given category on the y axis. The y axis is predefined and you do not need to set it (in fact, you just indicate the x in aes).

Here, we don't want to represent the frequency of the income (e.g., how many people have a given income) but the value of income based on education level.

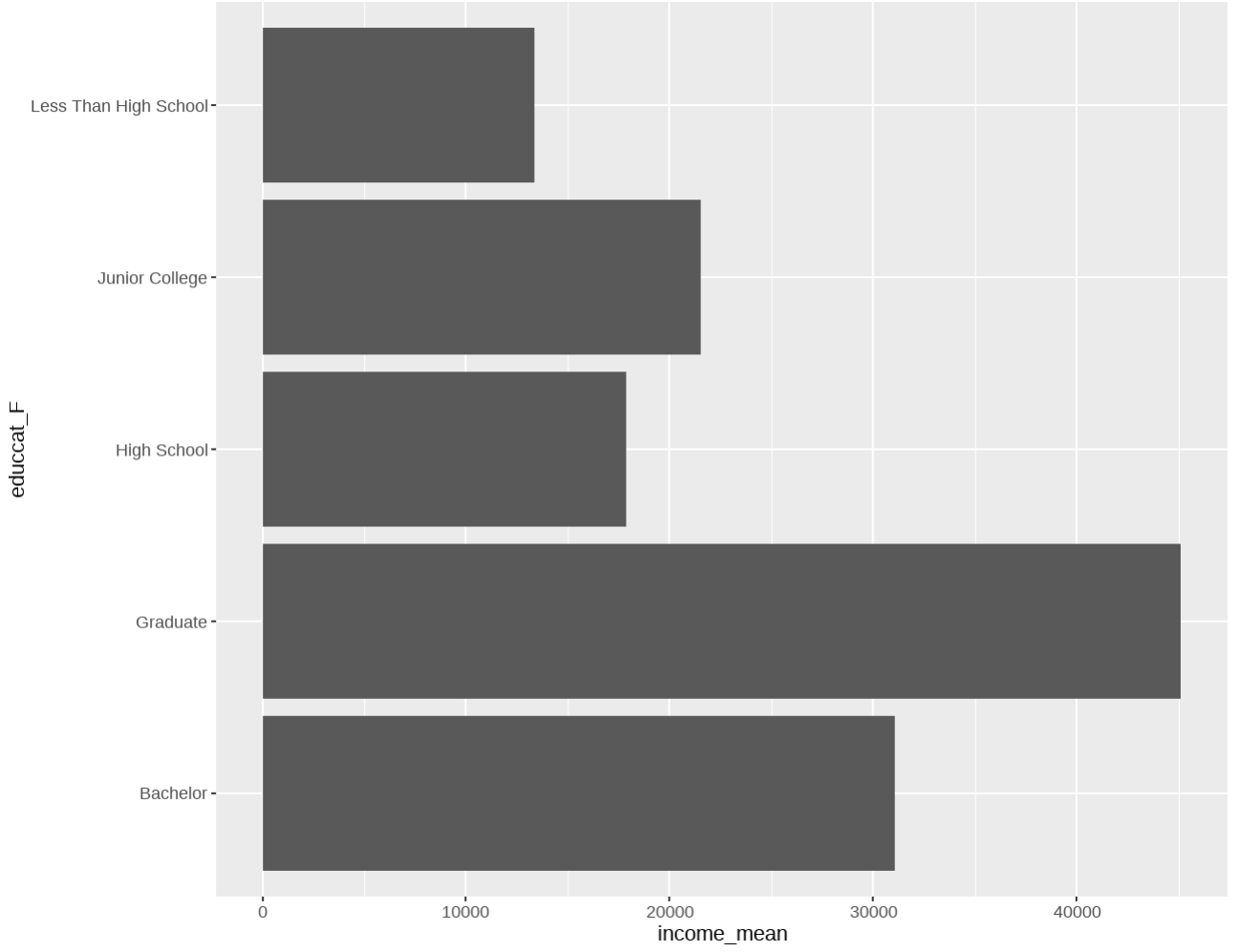
So, we specified the y axis to be equal to the average_income variable.

Since we specified y, we also needed to specify stat = "identity" to tell R not to calculate the frequency (which is the default) but to report the absolute values of our y variable on the y axis.

In a barplot:

- standard option is stat = "bin" which calculates the frequency;
- stat = "identity" allows you to specify a different y.

plot_4


Less Than High School
Junior College
High School
Graduate
Bachelor
educcat_F
0
10000
20000
30000
40000
income_mean

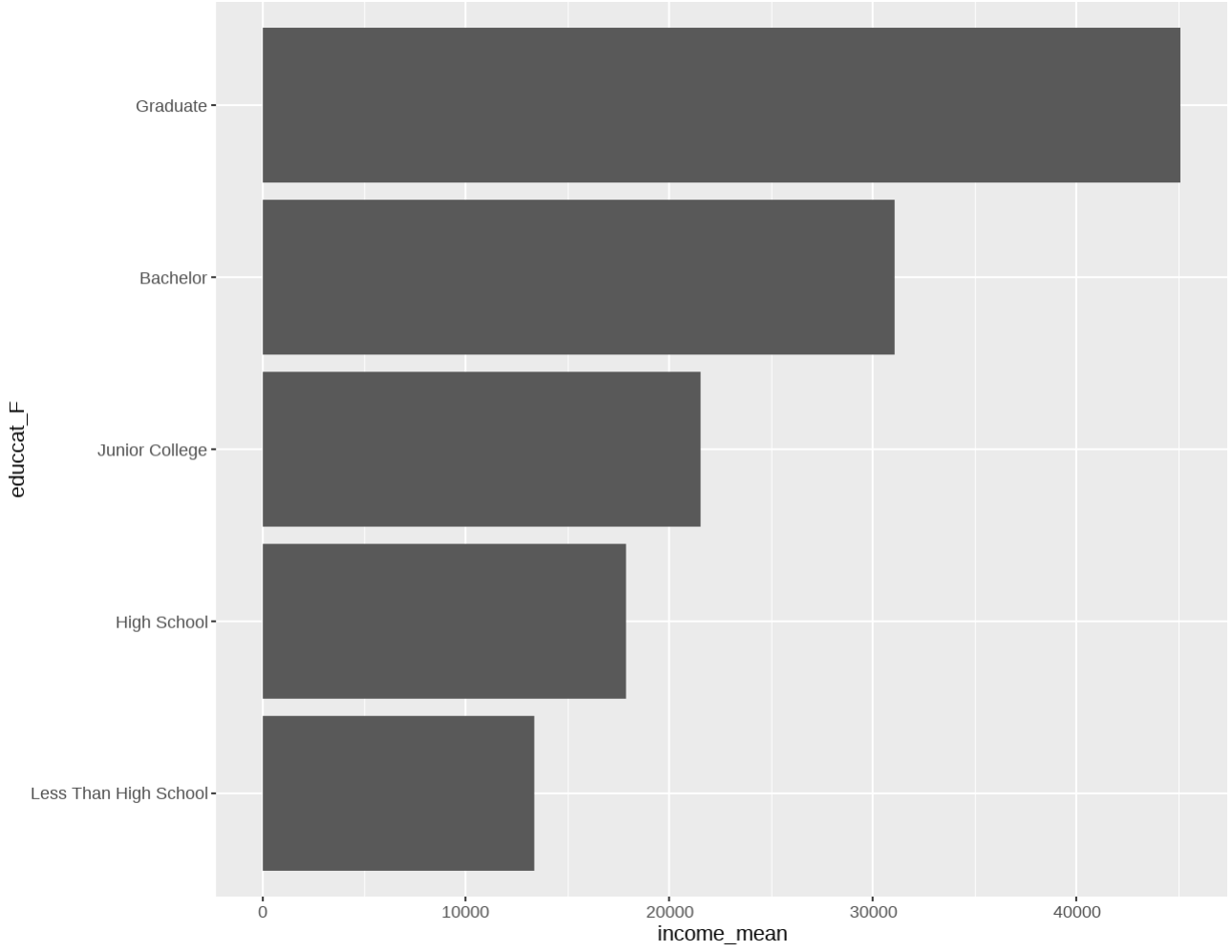
fct_reorder

Now, if we want the bars to be ordered by average income (e.g., education levels with a higher average income should go first), we will use fct_reorder.

```
plot_5 =
data %>%
  filter(!is.na(educcat_F)) %>%
  group_by(educcat_F) %>%
  summarize(income_mean = mean(realrinc, na.rm = T)) %>%
  mutate(educcat_F = fct_reorder(educcat_F, income_mean)) %>%
  ggplot() +
  geom_bar(aes(income_mean, educcat_F), stat = "identity")
```

Note that I prefer changing the factor using mutate instead of doing the transformation within the aes function. It keeps the code more organize (first, data wraggling, then visualization).

plot_5


Graduate
Bachelor
Junior College
High School
Less Than High School
educcat_F
0
10000
20000
30000
40000
income_mean

fct_reorder2

It is also possible that you might want to re-order the factors based on two other variables. This is common when the factor variable is used to highlight groups in your plot (e.g., as part of the color, fill, shape functions).

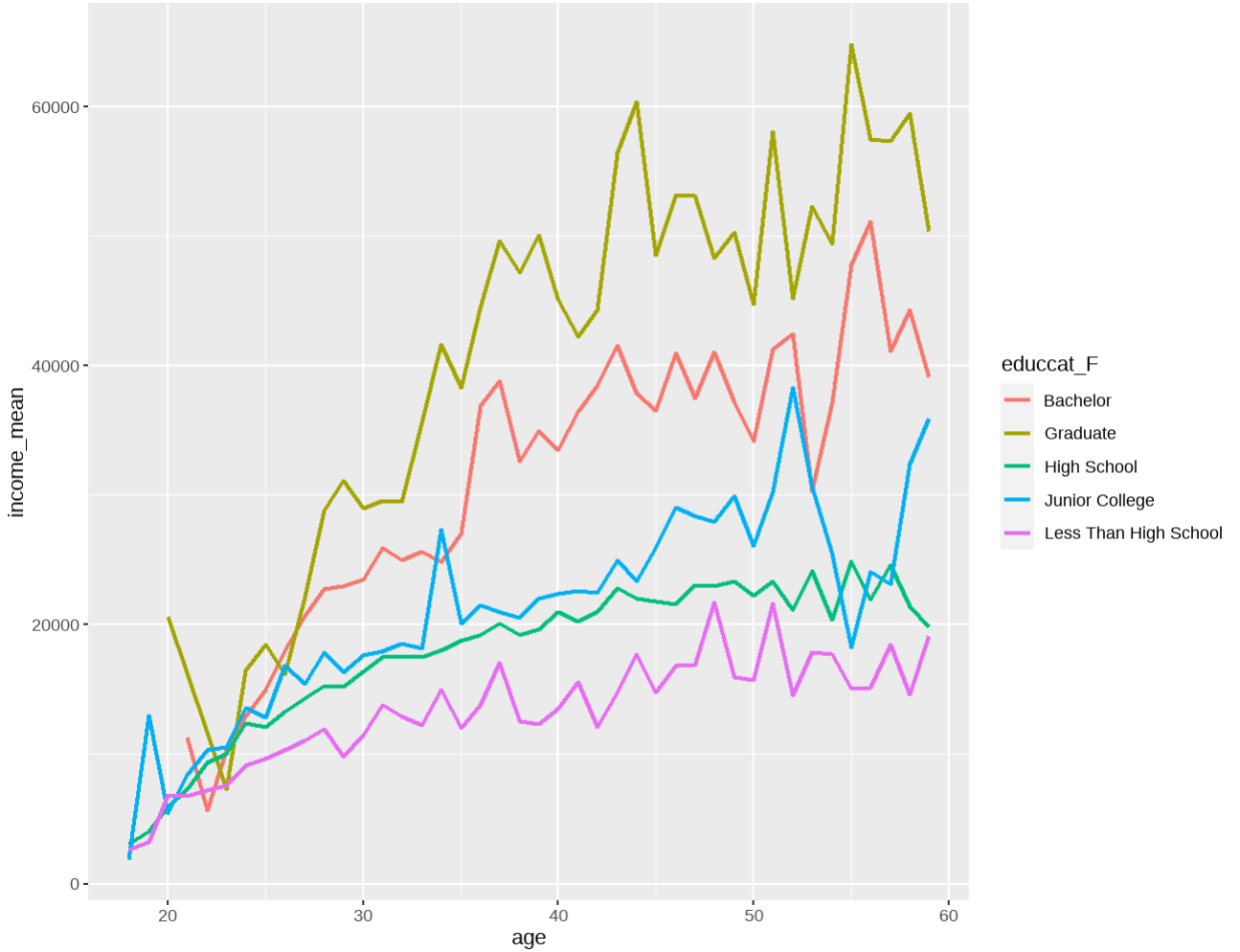
Let's create a line graph showing how the average income changes by age and education level.

fct_reorder2

```
plot_6 =
data %>%
  filter(!is.na(age),
         !is.na(educcat_F),
         age < 60,
         !(age == 19 & realrinc > 200000)) %>%
  group_by(age,
           educcat_F) %>%
  summarize(income_mean = mean(realrinc, na.rm = T)) %>%
  ggplot() +
  geom_line(aes(x = age,
                y = income_mean,
                color = educcat_F),
            size = 1)
```

```
plot_6
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```



fct_reorder2

```
plot_7 =
data %>%
  filter(!is.na(age),
         !is.na(educcat_F),
         age < 60,
         !(age == 19 & realrinc > 200000)) %>%
  group_by(age,
           educcat_F) %>%
  summarize(income_mean = mean(realrinc, na.rm = T)) %>%
  ggplot() +
  geom_line(aes(x = age,
                y = income_mean,
                color = fct_reorder2(educcat_F, age, income_mean)),
            size = 1)
```

```
## `summarise()` has grouped output by 'age'. You can override using the `.gro
```

fct_reorder

```
plot_7
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```

Recap - Change level orders

Function	When to use it
fct_relevel():	Reorder factor levels by hand
fct_inorder():	Reorder factor levels by first appearance
fct_infreq():	Reorder factor levels by frequency
fct_inseq():	Reorder factor levels by numeric order
fct_rev():	Reverse order of factor levels
fct_reorder():	Reorder factor levels by sorting along another variable (great fro cat-cont graphs)
fct_reorder2():	Reorder factor levels by sorting along 2 other variables

Levels' name

fct_recode

It is possible that you might want to change the name of the levels: names are not very descriptive, names are too long...

```
data %>%
  mutate(educcat_F = fct_recode(educcat_F,
                                "1" = "Less Than High School",
                                "2" = "High School",
                                "3" = "Junior College",
                                "4" = "Bachelor",
                                "5" = "Graduate")) %>%
  group_by(educcat_F) %>%
  summarize(count = n())
```

```
## # A tibble: 6 x 2
##   educcat_F count
## * <fct>     <int>
## 1 4          9219
## 2 5          4599
## 3 2         31713
## 4 3          3631
## 5 1         12400
## 6 <NA>        135
```

fct_recode

Note that you can also combine categories

```
data %>%
  mutate(educcat_F = fct_recode(educcat_F,
                                "1" = "Less Than High School",
                                "1" = "High School",
                                "2" = "Junior College",
                                "3" = "Bachelor",
                                "4" = "Graduate")) %>%
  group_by(educcat_F) %>%
  summarize(count = n())
```

```
## # A tibble: 5 x 2
##   educcat_F count
## * <fct>     <int>
## 1 3          9219
## 2 4          4599
## 3 1         44113
## 4 2          3631
## 5 <NA>        135
```

fct_collapse

If you aree combining several factor levels, you can also use fct_collapse

```
data %>%
  mutate(educcat_F =
           fct_collapse(educcat_F,
                        "1" = c("Less Than High School",
                                "High School",
                                "Junior College"),
                        "2" = c("Bachelor",
                                "Graduate"))) %>%
  group_by(educcat_F) %>%
  summarize(count = n())
```

```
## # A tibble: 3 x 2
##   educcat_F count
## * <fct>     <int>
## 1 2         13818
## 2 1         47744
## 3 <NA>        135
```

fct_lump

If the scope is just to simplify your categories, you might use fct_lump. It progressively combines small groups with one another (i.e., the smallest group with the second smallest group and so on).

```
data %>%
  mutate(educcat_F =
           fct_lump(educcat_F, 2)) %>%
  group_by(educcat_F) %>%
  summarize(count = n())
```

```
## # A tibble: 4 x 2
##   educcat_F             count
## * <fct>                 <int>
## 1 High School           31713
## 2 Less Than High School 12400
## 3 Other                 17449
## 4 <NA>                    135
```

Other functions

There are several other functions in the forcats package. These are the ones that I use the most often.

You can see a full list of functions in the forcats package here.