UiO PROMENTA
Det samfunnsvitenskapelige fakultet

Working with Factor Variables

E. F. Haghish

25/8/2022

Main idea

- Understanding factor variables
- It is a real nightmare
  - They are so confusing
  - You see string labels, but the data is stored in numbers!
  - We should learn to handle them properly
    - And convert them properly
- Create **ordinal factors (ordered)**

Factors

- Factor variables are categorical variables, either numeric or character variables
- factors are often take on a limited number of different values
- they need a special treatment in statistical modeling therefore should be stored as factors! Use the `factor` function to convert categorical variables to factors
- storing factors as factors reduces the required memory as well (less data to memorize)
- R stores factor as a vector of integer values with a corresponding set of character values for each value to use when the factor is displayed

```r
data <- c(1,2,2,3,1,2,3,3,1,2,3,3,1)
var <- factor(data)
var
```

```
##  [1] 1 2 2 3 1 2 3 3 1 2 3 3 1
## Levels: 1 2 3
```

- levels provides access to the levels attribute of a variable
- levels are always characters

```r
levels(var)
```

```
## [1] "1" "2" "3"
```

- levels can be indexed and labeled

```r
levels(var) <- c("low", "mid", "high")
var <- factor(data)
```

- another way of defining labels is using the `labels` arguments, which belongs to the `factor` function

```
factor(data, labels=c("I","II","III"))
```

```
##  [1] I   II  II  III I   II  III III I   II  III III I
## Levels: I II III
```

- the number of levels of a factor can be obtained using the `nlevels` function

```
nlevels(var)
```

```
## [1] 3
```

```
#class and mode
class(var)
```

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

```
mode(var)
```

```
## [1] "numeric"
```

Converting factors

- You can also convert factors to numeric or characters again
- You can't do arithmatic operations on factors:

```
var[1] + var[2]
```

```
## Warning in Ops.factor(var[1], var[2]): '+' not meaningful for factors

## [1] NA
```

- As noted, factors are stored with numeric values (`mode` function revealed that). To convert factors to numbers, `as.numeric` only returns how the variable is internally stored in R. **The level should be taken into consideration**

```
a <- c(10,20,20,50,10,20,10,50,20)
b <- factor(a)
as.numeric(b) #WRONG RESULTS!
```

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

- To convert a factor variable – **with numeric labels** – back to numeric, the levels should be used.

```
levels(b)[b]
```

```
## [1] "10" "20" "20" "50" "10" "20" "10" "50" "20"
```

```
as.numeric(levels(b)[b])
```

```
## [1] 10 20 20 50 10 20 10 50 20
```

- this is equal to converting the factor to a character variable and then, to a numeric variable which looks simpler

```
as.numeric(as.character(b))
```

```
## [1] 10 20 20 50 10 20 10 50 20
```

Ordered factors (Ordinal)

- Ordered factors differ from factors only in their class
- **methods and the model-fitting functions treat the two classes quite differently.**
  - **make sure your data is correctly destinguishing these variables**
- Use the argument of `ordered=TRUE` in the `factor` function to create ordered factor
- or use the `ordered()` function
- If argument `ordered` is true (or `ordered()` is used) the result has class `c("ordered", "factor")`

```
ordered(var)
```

```
##  [1] 1 2 2 3 1 2 3 3 1 2 3 3 1
## Levels: 1 < 2 < 3
```

```
class(ordered(4:1))
```

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

```
(b <- factor(a, ordered = TRUE))
```

```
## [1] 10 20 20 50 10 20 10 50 20
## Levels: 10 < 20 < 50
```

```
(b <- ordered(a))
```

```
## [1] 10 20 20 50 10 20 10 50 20
## Levels: 10 < 20 < 50
```

- a lot of calculations can be done with ordered factors
- for doing computations with ordered factors, they should first be turned to numeric values

Manipulating factors

- When a factor is first created, all of its levels are stored along with the factor
- if subsets of the factor are extracted, the subsets will retain all of the original levels, even if the levels are not existing in the subset
- This can create problems (e.g. when creating a model or interaction)
- **we have to get rid of the levels that are not used!**

Example

As an example, consider a random sample from the letters vector, which is part of the base R distribution

```r
lets <- sample(letters,size=100,replace=TRUE)
lets <- factor(lets)
table(lets[1:5])
```

```
##
## a b c d e f g h i j k l m n o p q r s t u v w x y z
## 2 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

- Use `drop=TRUE` argument to the subscripting operator. When used with factors, this argument will remove the unused levels

```r
table(lets[1:5,drop=TRUE])
```

```
##
## a d f k
## 2 1 1 1
```

- this could also acheived by passing the factors to the `table` function

```
table(factor(lets[1:5]))
```

```
##
## a d f k
## 2 1 1 1
```

- use the `exclude=` argument in the `factor` function to exclude a particular level
- use the `cut` function to create factors from continuous variables (p 72)
- when combining variables which are factors, because the `c` function will interpret the factors as integers, they should first be converted back to their original values (through the levels function), then catenated and converted to a new factor

```
fact1 <- factor(sample(letters,size=10,replace=TRUE))
fact2 <- factor(sample(letters,size=10,replace=TRUE))
fact1
```

```
##  [1] p l t p o m x n p f
## Levels: f l m n o p t x
```

```
fact2
```

```
##  [1] w w i j e h t i t o
## Levels: e h i j o t w
```

```
fact12 <- factor(c(levels(fact1)[fact1],levels(fact2)[fact2]))
fact12
```

```
##  [1] p l t p o m x n p f w w i j e h t i t o
## Levels: e f h i j l m n o p t w x
```