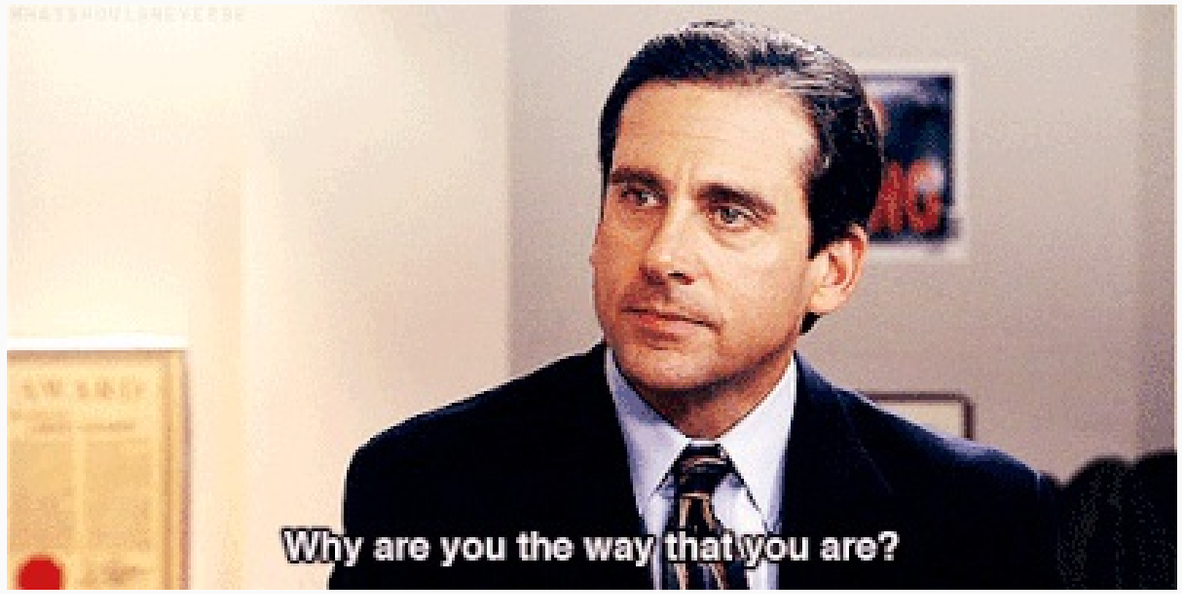
Data Analysis in R

Dealing (Learning to Love) With Factors

Michael E DeWitt Jr

2018-10-18 (updated: 2018-11-02)

Factors Are Frustrating To Novices



- Factors throw weird errors
- They get in the way of sorting (sometimes)

But Why Factors?

Categorical data types take on discrete values

But Why Factors?

Categorical data types take on discrete values

- Months of the Year
- Hair Color
- Likert Scales
- Political Party Affiliation
- etc

But Why Factors?

Categorical data types take on discrete values

- Months of the Year
- Hair Color
- Likert Scales
- Political Party Affiliation
- etc

Factors help R represent *categorical* variables

Factor Representation in R

Factors have a class of `factor`

```
race <- factor(x = c("White", "Non-White"))
class(race)
```

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

Factor Representation in R

Factors have a class of `factor`

```
race <- factor(x = c("White", "Non-White"))
class(race)
```

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

Factors have "levels" which represent the values that the factor can take on

```
levels(race)
```

```
## [1] "Non-White" "White"
```

Coercion

You can can make any vector/ column a factor using `factor` or `as.factor`

```
mtcars %>%
  mutate(cyl_fact = factor(cyl)) %>%
  select(cyl_fact, cyl)
```

```
## # A tibble: 32 x 2
##    cyl_fact   cyl
##    <fct>    <dbl>
##  1 6            6
##  2 6            6
##  3 4            4
##  4 6            6
##  5 8            8
##  6 6            6
##  7 8            8
##  8 4            4
##  9 4            4
## 10 6            6
## # ... with 22 more rows
```

As always, it is best to be explicit

If you do not specify the levels R will alphabetise and set the order accordingly

```
months <- factor(c("Jan", "Feb", "Mar"))
```

As always, it is best to be explicit

If you do not specify the levels R will alphabetise and set the order accordingly

```
months <- factor(c("Jan", "Feb", "Mar"))
```

```
months
```

```
## [1] Jan Feb Mar
## Levels: Feb Jan Mar
```

Levels vs Labels, huh?

`levels` are the numeric representation of a factor

`labels` are the pretty name that it is given

Levels vs Labels, huh?

`levels` are the numeric representation of a factor

`labels` are the pretty name that it is given

Practically

Labels are nice for printing

Levels are for ordering and doing math

A Practical use case

Likert Scale

```
likert_agree <- c("Strongly Disagree", "Disagree", "Neither Agree/ Nor Disagree", "Agree", "Strongly Agree")
```

A Practical use case

Likert Scale

```
likert_agree <- c("Strongly Disagree", "Disagree", "Neither Agree/ Nor Disagree", "Agree", "Strongly Agree")
```

We have some data

```
response_group <- data_frame(response = c(1 ,2 ,3 ,4 ,4 ,4, 2, 3, 4))
```

A Practical use case

Likert Scale

```
likert_agree <- c("Strongly Disagree", "Disagree", "Neither Agree/ Nor Disagree", "Agree", "Strongly Agree")
```

We have some data

```
response_group <- data_frame(response = c(1 ,2 ,3 ,4 ,4 ,4, 2, 3, 4))
```

And we want to do some analysis

```
mean(response_group$response)
```

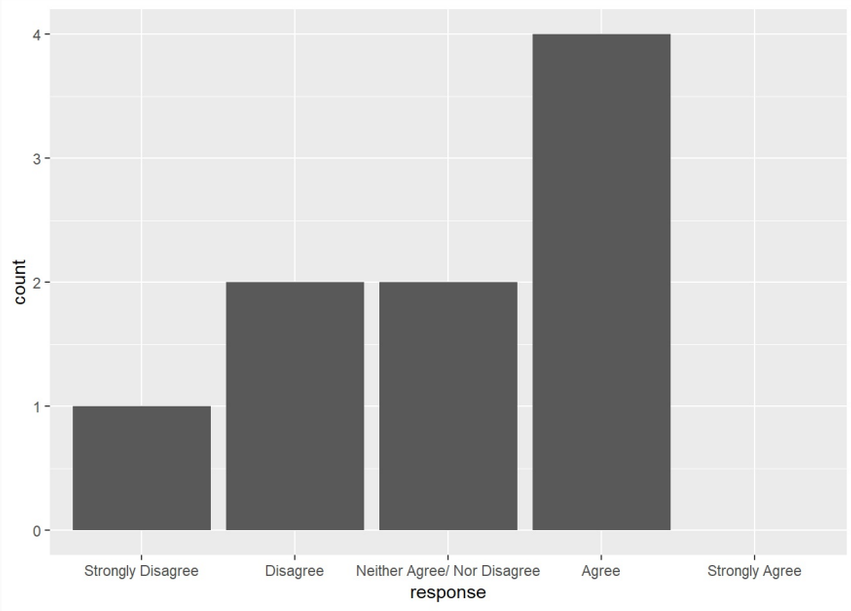
```
## [1] 3
```

A Practical Use Case

```
response_group %>%
  mutate(response = factor(response,
                           likert_agree,
                           levels = 1:5)) %>%
  count(response)
```

```
## # A tibble: 4 x 2
##   response                       n
##   <fct>                      <int>
## 1 Strongly Disagree              1
## 2 Disagree                       2
## 3 Neither Agree/ Nor Disagree    2
## 4 Agree                          4
```

```
ggplot(response_group, aes(response)) +
  geom_bar() +
  scale_x_discrete(drop = FALSE)
```



So how do we fix the defaults?

Enter the `forcats` package

- Tidyverse package[1]
- (almost) All functions begin with the `fct_` prefix
- Designed to help make working with factors easier[2]

[1] Package Page here

[2] More details in Hadley Wickham's *R for Data Science* here

A Practical Example

An Extract from the General Social Survey is provided in `forcats`

```
gss_cat
```

```
## # A tibble: 21,483 x 9
##     year marital     age race  rincome   partyid   relig   denom   tvhours
##    <int> <fct>     <int> <fct> <fct>     <fct>     <fct>   <fct>     <int>
##  1  2000 Never ma~    26 White $8000 to~ Ind,near~ Protes~ Southe~      12
##  2  2000 Divorced     48 White $8000 to~ Not str ~ Protes~ Baptis~      NA
##  3  2000 Widowed      67 White Not appl~ Independ~ Protes~ No den~       2
##  4  2000 Never ma~    39 White Not appl~ Ind,near~ Orthod~ Not ap~       4
##  5  2000 Divorced     25 White Not appl~ Not str ~ None    Not ap~       1
##  6  2000 Married      25 White $20000 -~ Strong d~ Protes~ Southe~      NA
##  7  2000 Never ma~    36 White $25000 o~ Not str ~ Christ~ Not ap~       3
##  8  2000 Divorced     44 White $7000 to~ Ind,near~ Protes~ Luther~      NA
##  9  2000 Married      44 White $25000 o~ Not str ~ Protes~ Other         0
## 10  2000 Married      47 White $25000 o~ Strong r~ Protes~ Southe~       3
## # ... with 21,473 more rows
```

Let's take a peak

```
glimpse(gss_cat)
```

```
## Observations: 21,483
## Variables: 9
## $ year    <int> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, ...
## $ marital <fct> Never married, Divorced, Widowed, Never married, Divor...
## $ age     <int> 26, 48, 67, 39, 25, 25, 36, 44, 44, 47, 53, 52, 52, 51...
## $ race    <fct> White, White, White, White, White, White, White, White...
## $ rincome <fct> $8000 to 9999, $8000 to 9999, Not applicable, Not appl...
## $ partyid <fct> Ind,near rep, Not str republican, Independent, Ind,nea...
## $ relig   <fct> Protestant, Protestant, Protestant, Orthodox-christian...
## $ denom   <fct> Southern baptist, Baptist-dk which, No denomination, N...
## $ tvhours <int> 12, NA, 2, 4, 1, NA, 3, NA, 0, 3, 2, NA, 1, NA, 1, 7, ...
```

Let's Examine the Religion Section A Little Closer

```
gss_cat %>%
  count(relig)
```

```
## # A tibble: 15 x 2
##    relig                      n
##    <fct>                  <int>
##  1 No answer                 93
##  2 Don't know                15
##  3 Inter-nondenominational  109
##  4 Native american           23
##  5 Christian                689
##  6 Orthodox-christian        95
##  7 Moslem/islam             104
##  8 Other eastern             32
##  9 Hinduism                  71
## 10 Buddhism                 147
## 11 Other                    224
## 12 None                    3523
## 13 Jewish                   388
## 14 Catholic                5124
## 15 Protestant             10846
```

Enter `forcats` for aggregation

`fct_lump` will help us by grouping infrequent groups of our specifcation together

```
gss_cat %>%
  mutate(relig = fct_lump(relig, n = 4)) %>%
  count(relig)
```

```
## # A tibble: 5 x 2
##   relig          n
##   <fct>      <int>
## 1 Christian    689
## 2 None        3523
## 3 Catholic    5124
## 4 Protestant 10846
## 5 Other       1301
```

`fct_other` will replace non-specified with others

```
gss_cat %>%
  mutate(relig = fct_other(relig,
                           keep = c("Christian", "Moslem/i
  count(relig)
```

```
## # A tibble: 4 x 2
##   relig            n
##   <fct>        <int>
## 1 Christian      689
## 2 Moslem/islam   104
## 3 Jewish         388
## 4 Other        20302
```

Some additional functions

`fct_collapse` can be used when you want to collapse a specific factor together

```
gss_cat %>%
  count(partyid)
```

```
## # A tibble: 10 x 2
##    partyid               n
##    <fct>             <int>
##  1 No answer           154
##  2 Don't know            1
##  3 Other party         393
##  4 Strong republican  2314
##  5 Not str republican 3032
##  6 Ind,near rep       1791
##  7 Independent        4119
##  8 Ind,near dem       2499
##  9 Not str democrat   3690
## 10 Strong democrat    3490
```

```
gss_cat %>%
  mutate(party_id_rd = fct_collapse(partyid,
    Rep = c("Strong republican", "Not str republican"),
    Dem = c("Strong democrat", "Not str democrat"),
    Ind = c("Ind,near rep", "Independent", "Ind,near dem")
  count(party_id_rd)
```

```
## # A tibble: 6 x 2
##   party_id_rd     n
##   <fct>       <int>
## 1 No answer     154
## 2 Don't know      1
## 3 Other party   393
## 4 Rep          5346
## 5 Ind          8409
## 6 Dem          7180
```

We can also recode the levels completely

`fct_recode` will allows us to recode based on our own wishes

Syntax

```
data %>%
  mutate( x = fct_recode(x,
                         "New name1" = "Old Name1",
                         "New name2" = "Old Name2")
```

Example

```
gss_cat %>%
  mutate( rincome = fct_recode(rincome,
                               "Too much to say" = "Refused",
                               "None" = "Not applicable")) %>%
  select(rincome)
```

```
## # A tibble: 21,483 x 1
##    rincome
##    <fct>
##  1 $8000 to 9999
##  2 $8000 to 9999
##  3 None
##  4 None
##  5 None
##  6 $20000 - 24999
##  7 $25000 or more
##  8 $7000 to 7999
##  9 $25000 or more
## 10 $25000 or more
## # ... with 21,473 more rows
```

Sometimes we want to change the ordering of factors

`fct_relevel` Reset the factor orders

`fct_inorder` Orders based on appearence in the data

`fct_infreq` Orders from most common to leave common

`fct_rev` Reverses the orders of the levels

True Motivating Need

Modeling and analysis!!!

True Motivating Need

Modeling and analysis!!!

Analysis of Variance for Example...

```
aov(tvhours ~ rincome, data = gss_cat) %>%
  summary()
```

```
##               Df Sum Sq Mean Sq F value Pr(>F)
## rincome       15   4876   325.1   51.83 <2e-16 ***
## Residuals   11321  71000     6.3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 10146 observations deleted due to missingness
```

One Note

IF you create a factor and you specify a value that is not in the levels you will get an `NA` silently!

```
my_vector <- c("Yes", "No", "Maybe")

factor(my_vector, c("Yes", "No"))
```

```
## [1] Yes  No   <NA>
## Levels: Yes No
```

One Note

IF you create a factor and you specify a value that is not in the levels you will get an `NA` silently!

```
my_vector <- c("Yes", "No", "Maybe")

factor(my_vector, c("Yes", "No"))
```

```
## [1] Yes  No   <NA>
## Levels: Yes No
```

If you use `forcats` `parse_factor` you will be pased an NA

```
parse_factor(my_vector, c("Yes", "No"))
```

```
## Warning: 1 parsing failure.
## row # A tibble: 1 x 4 col     row   col expected           actual expected   <int> <int> <chr>              <chr>  actua
```

```
## [1] Yes  No   <NA>
## attr(,"problems")
## # A tibble: 1 x 4
##     row   col expected           actual
##   <int> <int> <chr>              <chr>
## 1     3    NA value in level set Maybe
## Levels: Yes No
```

Aside History of Factors

R developed as a statistical computing environment

Statisticians do experiements (often in blocks)

Factors make sense in this context

Additionally, it saved memory because factors could be represented with `numbers` rather than `characters`

Recap

Factors are useful for

- Summarising Categorical Data
- Representing Categorical Data in Analysis
- Organising Graphs
- `forcats` provides a `tidyverse` approach to factors

Now let's code