Manipulating categorical variables

Abhijit Dasgupta, PhD

Categorical variables

What are categorical variables?

Categorical variables are variables that

- have values defining categories of things
- typically have a few unique values
- may or may not be ordered
- are not interval-scaled, i.e., their differences don't make sense *per se*

What are categorical variables?

Non-ordered

1. Race (White, Black, Hispanic, Asian, Native American)
2. Gender (Male, Female, Other)
3. Geographic regions (Africa, Asia, Europe, North America, South America)
4. Genes/Proteins

Ordered

1. Income levels (< $10K, $10K - $25K, $25K - $75K, $75K - $100K)
2. BMI categories (Underweight, Normal, Overweight, Obese)
3. Number of bedrooms in houses (1 BR, 2BR, 3BR, 4BR)

Cateogical variables in R

The `factor` data type

R stores categorical variables as type `factor`.

- You can coerce a character or numeric object into a factor using `as.factor`.
- You can check if an object is a factor with `is.factor`.
- You can create a factor with the function `factor`.

The `factor` data type

> `factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA)`
>
> factor returns an object of class "factor" which has a set of integer codes the length of x with a "levels" attribute of mode character and unique

---

- Internally, each level of a factor is coded as an integer
- Each such integer has a corresponding `level` which is a character, describing the level.
- You can add `labels` to each level to change the printed form of the factor.

The factor data type

```
x <- c('Maryland','Virginia', 'District', 'Maryland','Virginia')  # a character vector
xf <- as.factor(x)
xf
```

```
[1] Maryland Virginia District Maryland Virginia
Levels: District Maryland Virginia
```

There are three levels, that by default are in alphabetical order

---

```
as.integer(xf)
```

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

- District = 1, Maryland = 2, Virginia = 3

```
as.character(xf)
```

```
[1] "Maryland" "Virginia" "District" "Maryland" "Vi
```

- Get original characters back

The factor data type

```
y <- c(5, 3, 9, 4, 5, 3)
yf <- as.factor(y)
yf
```

```
[1] 5 3 9 4 5 3
Levels: 3 4 5 9
```

Levels are still in alphanumeric order

```
as.numeric(yf)
```

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

- Note, we don't get original integers back!!
- 3 = 1, 4 = 2, 5 = 3, 9 = 4

```
as.numeric(as.character(yf))
```

```
[1] 5 3 9 4 5 3
```

- This is how you get numbers back

The factor data type

```
x <- c('MD','DC','VA','MD','DC')
xf <- factor(x)
unclass(xf)
```

```
[1] 2 1 3 2 1
attr(,"levels")
[1] "DC" "MD" "VA"
```

```
x <- c('MD','DC','VA','MD','DC')
xf <- factor(x, levels = c('MD','DC','VA'))
unclass(xf)
```

```
[1] 1 2 3 1 2
attr(,"levels")
[1] "MD" "DC" "VA"
```

- If I change the level designation, the underlying coding changes
- This is important when a factor is an independent variable in a regression model

The `factor` data type

The `drv` variable in the `mpg` dataset tells us the kind of drive (front, rear or 4-wheel) each car has. However it's coded as `f`, `r`, and `4`, which is not great for display purposes. We can re-label these levels, but we have to be a bit careful

```
x <- mpg$drv
xf <- factor(x,
             levels = c('4-wheel','Front wheel',
                        'Rear wheel'))
head(xf)
```

```
  [1] <NA> <NA> <NA> <NA> <NA> <NA>
  Levels: 4-wheel Front wheel Rear wheel
```

```
x <- mpg$drv
xf <- factor(x,
             levels = c('4', 'f','r'),
             labels = c('4-wheel','Front wheel',
                        'Rear wheel'))
head(xf)
```

```
  [1] Front wheel Front wheel Front wheel Front wheel
  Levels: 4-wheel Front wheel Rear wheel
```

Levels have to match what's actually in the original data, but you can re-label the levels.

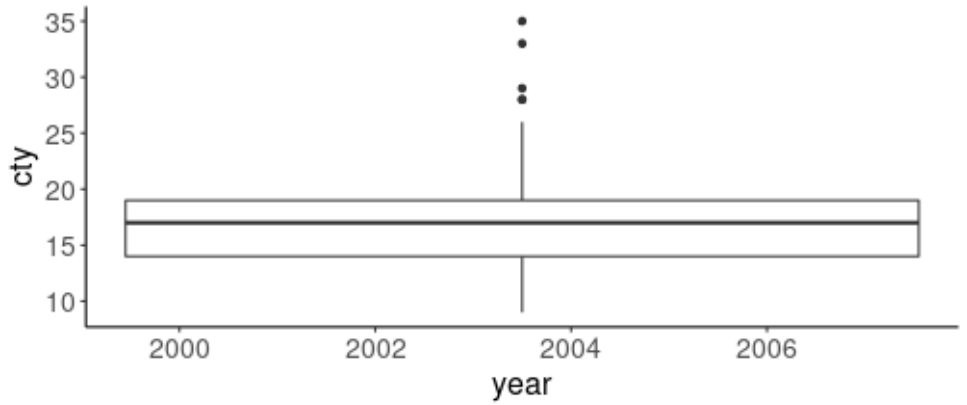
Why factors?

Factors are R's discrete data type

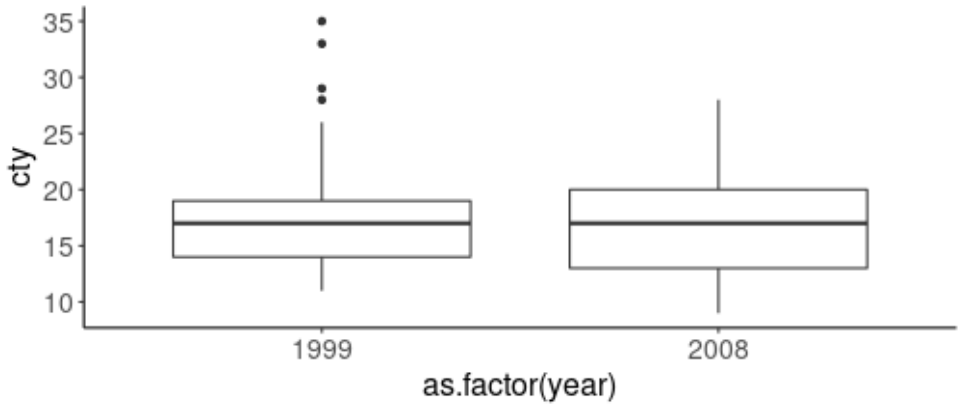
- Factors are interpreted as discrete by R's functions

```
ggplot(mpg,
       aes(year, cty))+
  geom_boxplot()
```

```
  Warning: Continuous x aesthetic -- did you forget a
```




```
ggplot(mpg,
       aes(as.factor(year), cty))+
  geom_boxplot()
```



Dummy variables are automatically created from factors

```
model.matrix(~species, data = palmerpenguins::penguin
```

```
   (Intercept) speciesChinstrap speciesGentoo
1            1                0             0
2            1                0             0
3            1                1             0
4            1                1             0
5            1                0             1
6            1                0             1
attr(,"assign")
[1] 0 1 1
attr(,"contrasts")
attr(,"contrasts")$species
[1] "contr.treatment"
```

- If a factor has *n* levels, you get *n-1* dummy variables
- The level corresponding to integer code 1 is omitted as the reference level

---

Changing the base level (integer code 1) changes model interpretation since it changes the reference level against which all other levels are compared.

Manipulating factors

The `forcats` package (part of `tidyverse`)

Effect in models

```
m <- lm(body_mass_g ~ species, data = penguins)
broom::tidy(m)
```

```
# A tibble: 3 x 5
  term             estimate std.error statistic   p
  <chr>               <dbl>     <dbl>     <dbl>
1 (Intercept)        3701.       37.6    98.4   2.4
2 speciesChinstrap     32.4      67.5     0.480 6.3
3 speciesGentoo      1375.       56.1    24.5   5.4
```

Compare with Adele

```
p1 <- penguins %>%
  mutate(species = fct_relevel(species, 'Gentoo'))
m1 <- lm(body_mass_g ~ species, data=p1 )
broom::tidy(m1)
```

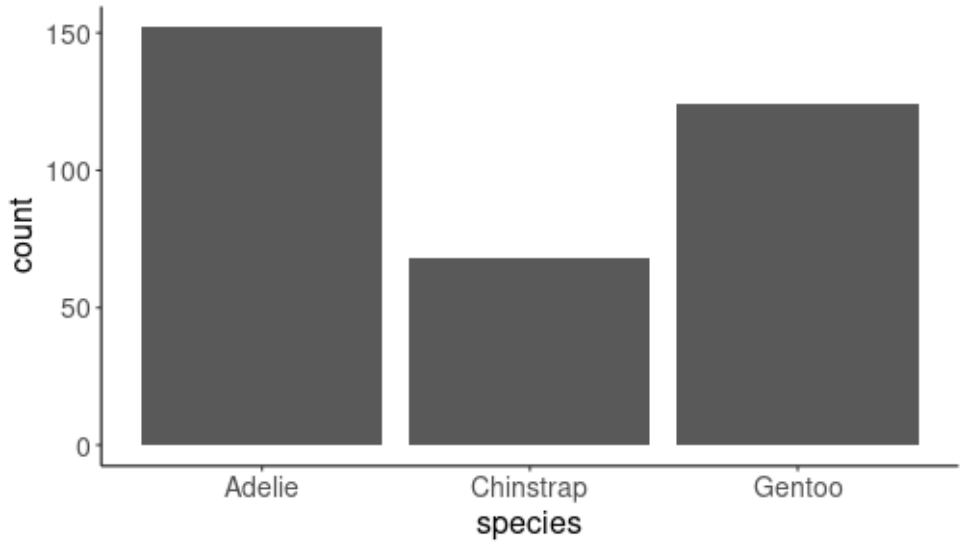
```
# A tibble: 3 x 5
  term             estimate std.error statistic   p
  <chr>               <dbl>     <dbl>     <dbl>
1 (Intercept)        5076.       41.7     122.  6.8
2 speciesAdelie     -1375.       56.1     -24.5 5.4
3 speciesChinstrap  -1343.       69.9     -19.2 3.2
```

Compare with Gentoo

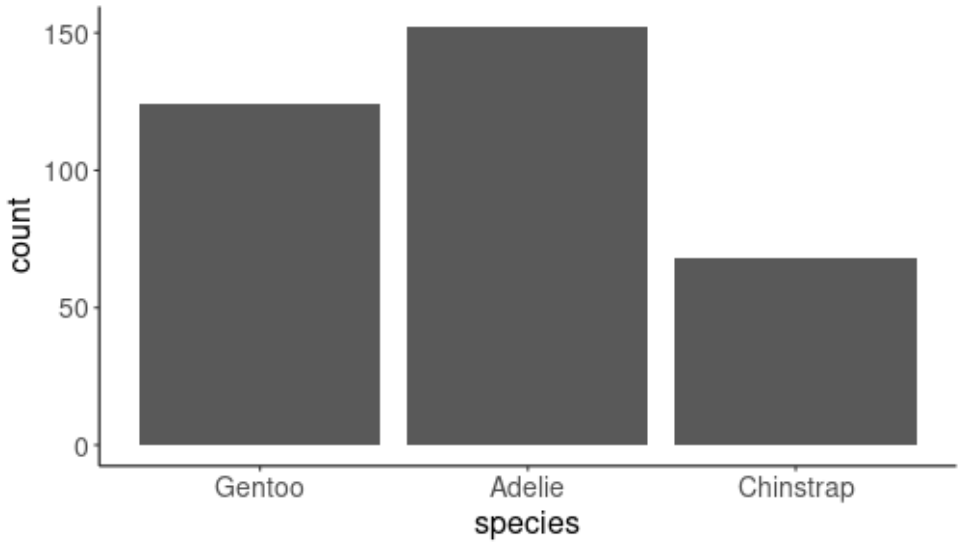
Providing only one level to `fct_relevel` makes that the base level (integer code 1).
You can also fully specify all the levels in order, or partially specify them. If you partially specify them, the remaining levels will be put in alphabetical order after the ones you specify.

Effect in plots

```
ggplot(penguins,
       aes(x = species))+
  geom_bar()
```




```
ggplot(p1,
       aes(x = species))+
  geom_bar()
```



Changes the order in which bars are plotted

Extra levels

```
x <- factor(str_split('statistics', '')[[1]],
            levels = letters)
x
```

```
 [1] s t a t i s t i c s
Levels: a b c d e f g h i j k l m n o p q r s t u v
```

```
p1 <- penguins %>% filter(species != 'Gentoo')
fct_count(p1$species)
```

```
# A tibble: 3 x 2
  f             n
  <fct>     <int>
1 Adelie      152
2 Chinstrap    68
3 Gentoo        0
```

```
fct_drop(x)
```

```
 [1] s t a t i s t i c s
Levels: a c i s t
```

```
p1 <- p1 %>% mutate(species = fct_drop(species))
fct_count(p1$species)
```

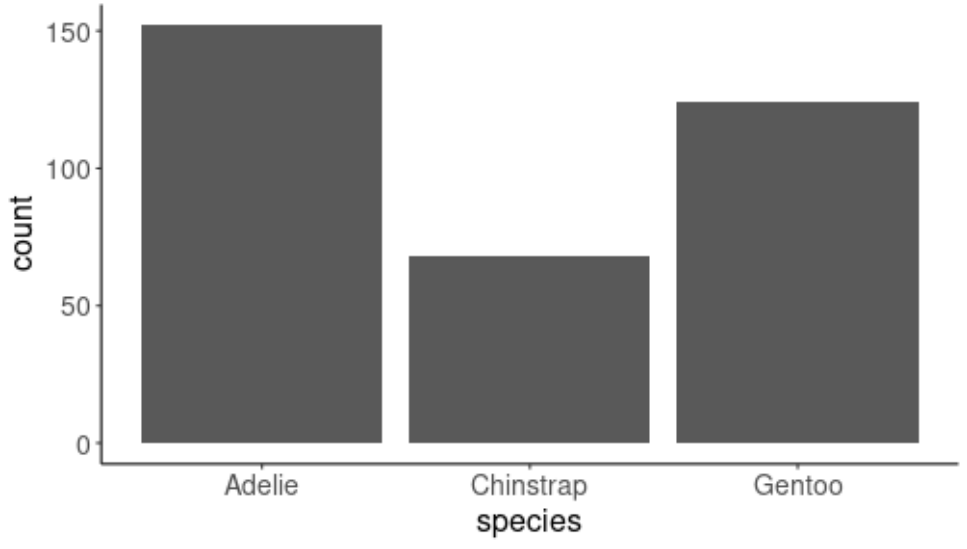
```
# A tibble: 2 x 2
  f             n
  <fct>     <int>
1 Adelie      152
2 Chinstrap    68
```

Getting rid of extra levels

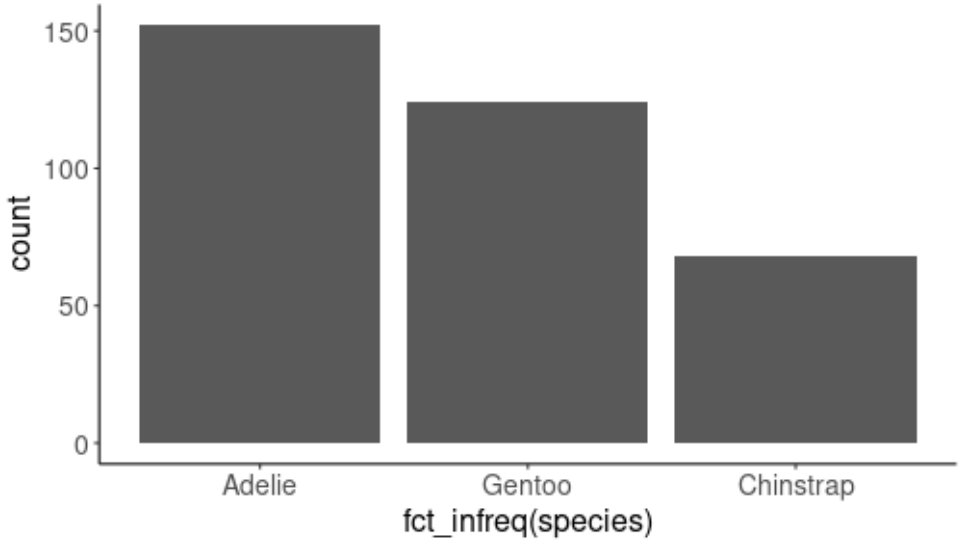
Sometimes levels with no data show up in summaries or plots

Ordering levels by frequency

```
ggplot(penguins,
       aes(x = species))+
  geom_bar()
```



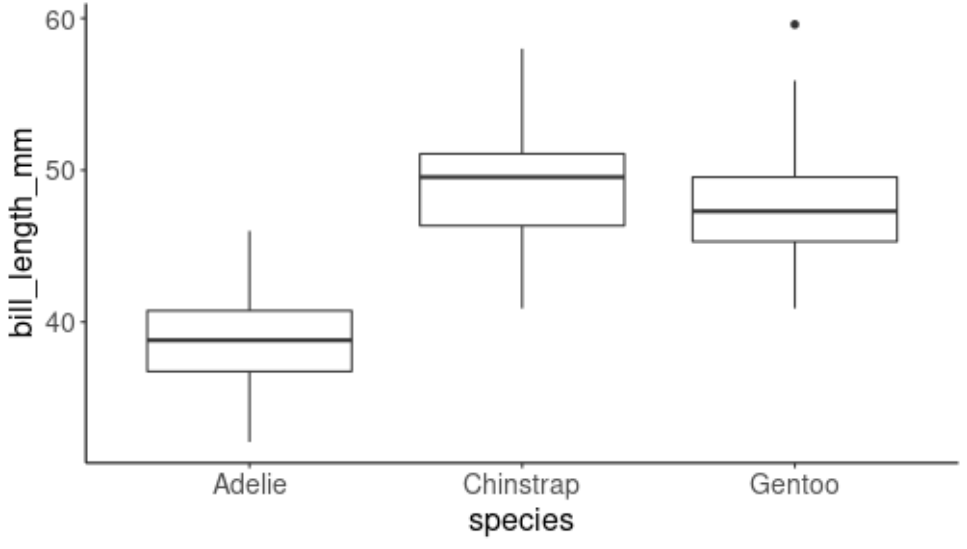

```
ggplot(penguins,
       aes(x = fct_infreq(species)))+
  geom_bar()
```



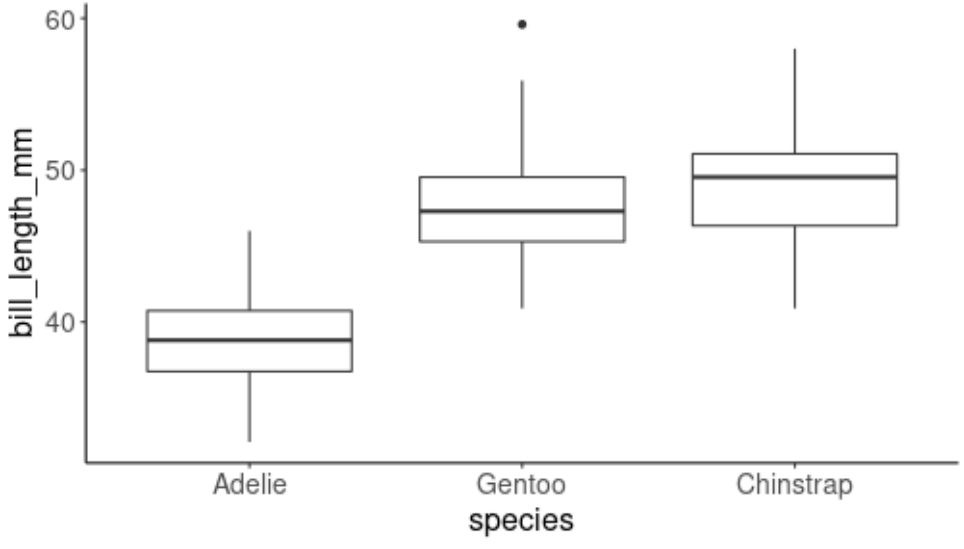
Ordering levels from most to least frequent

Ordering levels by values of another variable

```
ggplot(penguins,
       aes(x = species,
           y = bill_length_mm))+
  geom_boxplot()
```




```
ggplot(penguins,
       aes(x = fct_reorder(species, bill_length_mm,
                           .fun=median, na.rm=T),
           y = bill_length_mm))+
  geom_boxplot() + labs(x = 'species')
```

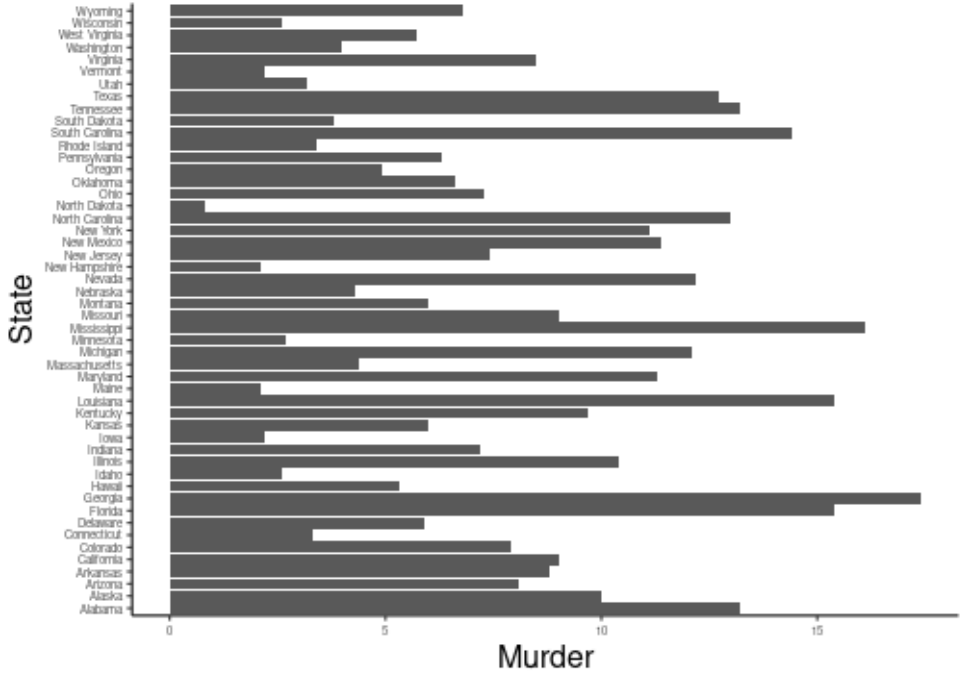


`fct_reorder` is useful for ordering a plot by ascending or descending levels. This makes the plot easier to read.

Ordering levels by values of another variable

```
USArrests <- USArrests %>% rownames_to_column('State')
```

```
ggplot(USArrests, aes(x=State, y = Murder))+
  geom_bar(stat = 'identity') +
  theme(axis.text = element_text(size=6))+
  coord_flip()
```




```
ggplot(USArrests, aes(
  x = fct_reorder(State, Murder),
  y = Murder))+
  geom_bar(stat = 'identity')+
  theme(axis.text = element_text(size=6))+
  coord_flip()
```

Order levels based on last values when plotting 2 variables

The level ordering also shows up in the order of levels in the legends of plots. Suppose you are plotting two variables, grouped by a factor.

```
ggplot(iris, aes(
  x = Sepal.Length,
  y = Sepal.Width,
  color = Species))+
  geom_smooth(se=F)
```

```
ggplot(iris, aes(
  x = Sepal.Length,
  y = Sepal.Width,
  color = fct_reorder2(Species,
                       Sepal.Length, Sepal.Width)))+
  geom_smooth(se=F) + labs(color = 'Species')
```

Further exploration

1. forcats cheatsheet
2. Chapter 15 of R4DS

Pie chart

```
penguins %>% count(species) %>%
  ggplot(aes(x=1,y=n, fill=species))+
    geom_bar(stat='identity')+
    coord_polar('y') +
    theme(axis.text = element_blank(),
          axis.title = element_blank(),
          axis.ticks=element_blank(),
          panel.grid = element_blank(),
          axis.line = element_blank())
```

See here and here for more examples.