ANOVA and factors

Stat 145

Using numbers to label categories

ANOVA and simple linear regression use the same function (`lm()` in R). R decides which to do based on the types of data supplied. Both expect a quantitative response variable.

- If the explanatory variable is quantitative, `lm()` fits a simple linear regression model.
- If the explanatory variable is categorical, `lm()` fits an ANOVA model.

R treats text data as categorical and numerical values as quantitative. But sometimes people use numbers to identify groups (e.g., 1, 2, 3). In this case, R would fit a linear regression model instead of an ANOVA… unless we do something to tell R otherwise.

```
Blisters <- read.csv("https://rpruim.github.io/s145/data/blisters.csv")
Blisters %>% sample_n(5) %>% pander()
```

days	treatment	group
8	A	1
6	A	1
9	B	2
13	P	3
10	B	2

The `factor()` function in R converts variables into factors – R's preferred way to deal with categorical data. Compare the results below. The values of `treatment` are A, B, and P. The values of group are 1, 2, and 3. The degrees of freedom in the ANOVA table alert us to the the problem if we use `group` as the explanatory variable. Wrapping in `factor()` treats these numeric values as category labels, matching the results we get when we use `treatement`.

```
lm(days ~ treatment, data = Blisters) %>% anova()
```

```
## Analysis of Variance Table
##
## Response: days
##           Df Sum Sq Mean Sq F value   Pr(>F)
## treatment  2 34.736 17.3681  6.4474 0.006256
## Residuals 22 59.264  2.6938
```

```
lm(days ~ group, data = Blisters) %>% anova()
```

```
## Analysis of Variance Table
##
## Response: days
##           Df Sum Sq Mean Sq F value   Pr(>F)
## group      1 34.531  34.531  13.355 0.001321
## Residuals 23 59.469   2.586
```

```r
lm(days ~ factor(group), data = Blisters) %>% anova()
```

```
## Analysis of Variance Table
##
## Response: days
##               Df Sum Sq Mean Sq F value   Pr(>F)
## factor(group)  2 34.736 17.3681  6.4474 0.006256
## Residuals     22 59.264  2.6938
```

Moral of the story: Always check the degrees of freedom to make sure they match what you are expecting. If you see degrees of freedom = 1 for the explanatory variable, you probably have a quantitative variable that needs to be converted to a categorical variable.

Ordering the levels

The possible values of a categorical variable are called the levels. By default, R puts the levels in alphabetical order. But we can use `factor()` to create a different ordering. This doesn't change the results of ANOVA, but it will rearrange the order in tables and plots.

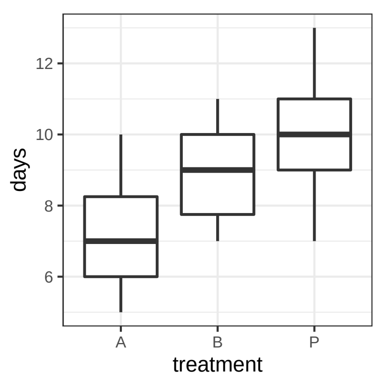
```r
Blisters2 <-
  Blisters %>%
  mutate(treatment2 = factor(treatment, levels = c("P", "A", "B")))

df_stats(days ~ treatment, data = Blisters)
```

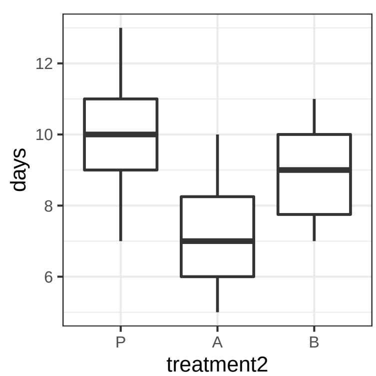
```
##   response treatment min   Q1 median    Q3 max     mean       sd n missing
## 1     days         A   5 6.00      7  8.25  10  7.25000 1.669046 8       0
## 2     days         B   7 7.75      9 10.00  11  8.87500 1.457738 8       0
## 3     days         P   7 9.00     10 11.00  13 10.11111 1.763834 9       0
```

```r
gf_boxplot(days ~ treatment, data = Blisters)
df_stats(days ~ treatment2, data = Blisters2)
```

```
##   response treatment2 min   Q1 median    Q3 max     mean       sd n missing
## 1     days          P   7 9.00     10 11.00  13 10.11111 1.763834 9       0
## 2     days          A   5 6.00      7  8.25  10  7.25000 1.669046 8       0
## 3     days          B   7 7.75      9 10.00  11  8.87500 1.457738 8       0
```

```r
gf_boxplot(days ~ treatment2, data = Blisters2)
```





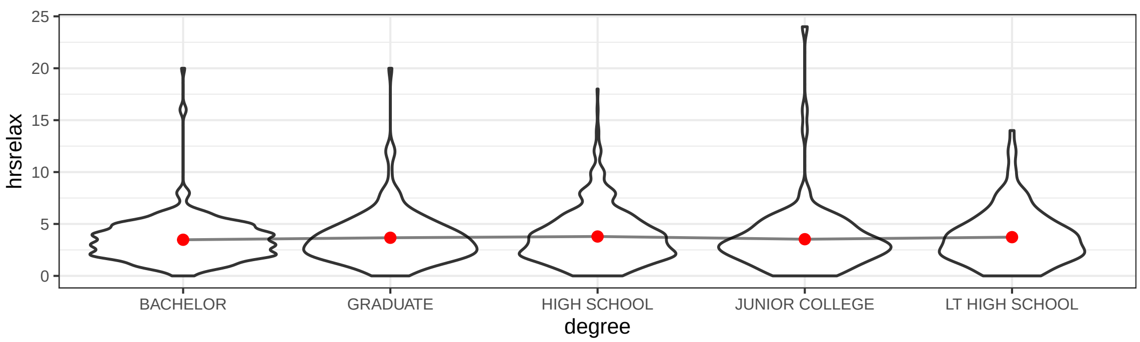
Here is another example.

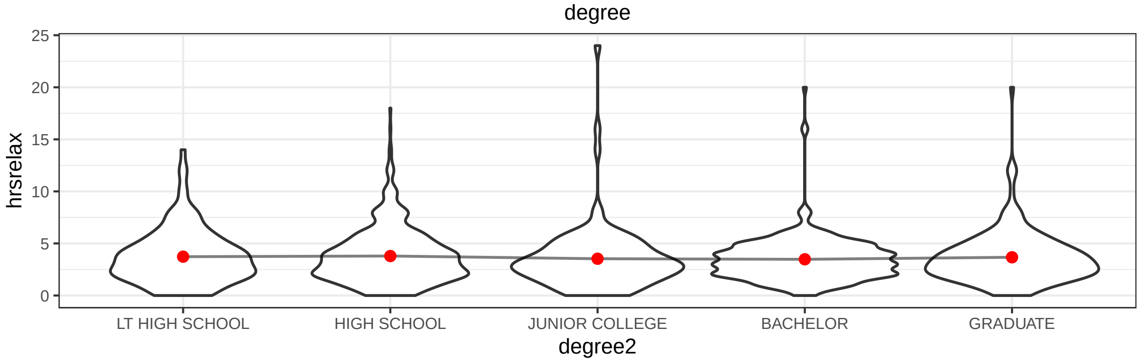
```r
library(openintro)
gss2010 <-  gss2010 %>%
  mutate(degree2 =  factor(degree, levels = c("LT HIGH SCHOOL", "HIGH SCHOOL",
                                              "JUNIOR COLLEGE",  "BACHELOR", "GRADUATE")
  ))
gf_violin(hrsrelax ~ degree, data = gss2010) %>%
  gf_summary(geom = "line", group = 1, alpha = 0.5) %>%
  gf_summary(geom = "point", color = "red")
```

```
## Warning: Removed 890 rows containing non-finite values (stat_ydensity).

## Warning: Removed 890 rows containing non-finite values (stat_summary).

## No summary function supplied, defaulting to `mean_se()`

## Warning: Removed 890 rows containing non-finite values (stat_summary).

## No summary function supplied, defaulting to `mean_se()`
```

```r
gf_violin(hrsrelax ~ degree2, data = gss2010) %>%
  gf_summary(geom = "line", group = 1, alpha = 0.5) %>%
  gf_summary(geom = "point", color = "red")
```

```
## Warning: Removed 890 rows containing non-finite values (stat_ydensity).

## Warning: Removed 890 rows containing non-finite values (stat_summary).

## No summary function supplied, defaulting to `mean_se()`

## Warning: Removed 890 rows containing non-finite values (stat_summary).

## No summary function supplied, defaulting to `mean_se()`
```





What are all those warning messages?

- `Removed 890 rows containing non-finite values`

    - If there are missing values in variables needed to make your plot, `ggformula` alerts you to this when making a plot so you don't mistakenly think that your plot is using all the rows of your data.

- `No summary function supplied, defaulting to mean_se()`
    - `gf_summary()` defaults to computing the mean and standard error, but you can provide other functions if you want other summaries (like median and IQR, for example). Points and lines only use one value (the mean here), but some other plots use multiple values. Here is an example.

```
gf_violin(hrsrelax ~ degree2, data = gss2010) %>%
  gf_summary(geom = "pointrange", group = 1, color = "steelblue")
```

```
## Warning: Removed 890 rows containing non-finite values (stat_ydensity).
```

```
## Warning: Removed 890 rows containing non-finite values (stat_summary).
```

```
## No summary function supplied, defaulting to `mean_se()`
```