Research Report 1 - Sample Code

Professor Noyce

- Get organized.
- Load data file and pivot longer
- Initial data visualization
- Descriptive stats
- Inferential stats

Get organized.

Create a new directory for lab 1, and a new R project for lab 1. Load any necessary packages.

```
library(tidyverse) # tidy data
library(here) # easier path management
here::i_am("2-analysis/research-report-1_sample-code.Rmd")
library(afex) # ANOVAs
library(performance) # model checks
```

Load data file and pivot longer

While R can read many common data formats, here we use one of the simplest: a comma separated values file. We create a variable specifying the filename and path, and then use `read.csv` to create a data frame.

You'll also notice that I regularly ask R to show me information about a data frame using either `head()` (the first 6 rows), `summary()` (basic statistics such as averages and counts), or `glimpse` .

```
# make a variable with the filename
datafile <- here("1-data", "skittles-taste_2025-01-22.csv") # make sure to set your data
file name appropriately!

# read it in
raw_data <- read.csv(datafile)

# check it
head(raw_data)
```

```
##   Participant_ID class_year orange_classic orange_gummy purple_classic
## 1              1     senior              6            4              7
## 2              2     senior              4            6              3
## 3              3     junior              5            3              4
## 4              4     senior              7            4              4
## 5              5     senior              4            5              2
## 6              6  sophomore              6            4              6
##   purple_gummy  X X.1
## 1            5 NA
## 2            3 NA
## 3            2 NA
## 4            2 NA
## 5            1 NA   `
## 6            2 NA
```

This data is in a *wide* format, and we use `pivot_longer()` to rearrange it into *long* format. Note that because the column headings are all in the format **color-underscore-texture**, we can tell `pivot_longer` to use the names_sep option to split those at the underscore. While we're cleaning this up, we can use `mutate()` to force the color and texture variables to be stored as factors rather than as character strings.

```
# New variable data will be old variable "raw_data", piped to...
data <- raw_data  %>%

  # which function will get just the columns we need?
  ____("Participant_ID",
       starts_with("purple"),
       # which other columns do we need?
       starts_with("____"))  %>%

  # piped to pivot_longer
  pivot_longer(
   cols=2:5,
   names_to = c("color","texture"),
   names_sep = "_",
   values_to = "rating")  %>%

  # piped to...
  # which function lets you modify a variable?
  ____(
    color = factor(color),
    texture = factor(texture),
    rating = as.numeric(rating)
    )
```

Check on your processed data using `head()` or `glimpse()`.

```
____(data)
```

Initial data visualization

A histogram of the ratings for each condition lets us make sure that all values are in the expected range. `ggplot()` has its own syntax which can be frustrating, but the plots are very pretty so we put up with it.

Because we want histograms, we need the rating scale on the x axis, so we call `ggplot` with `aes(x = rating)`. ("aes" is short for "aesthetics").

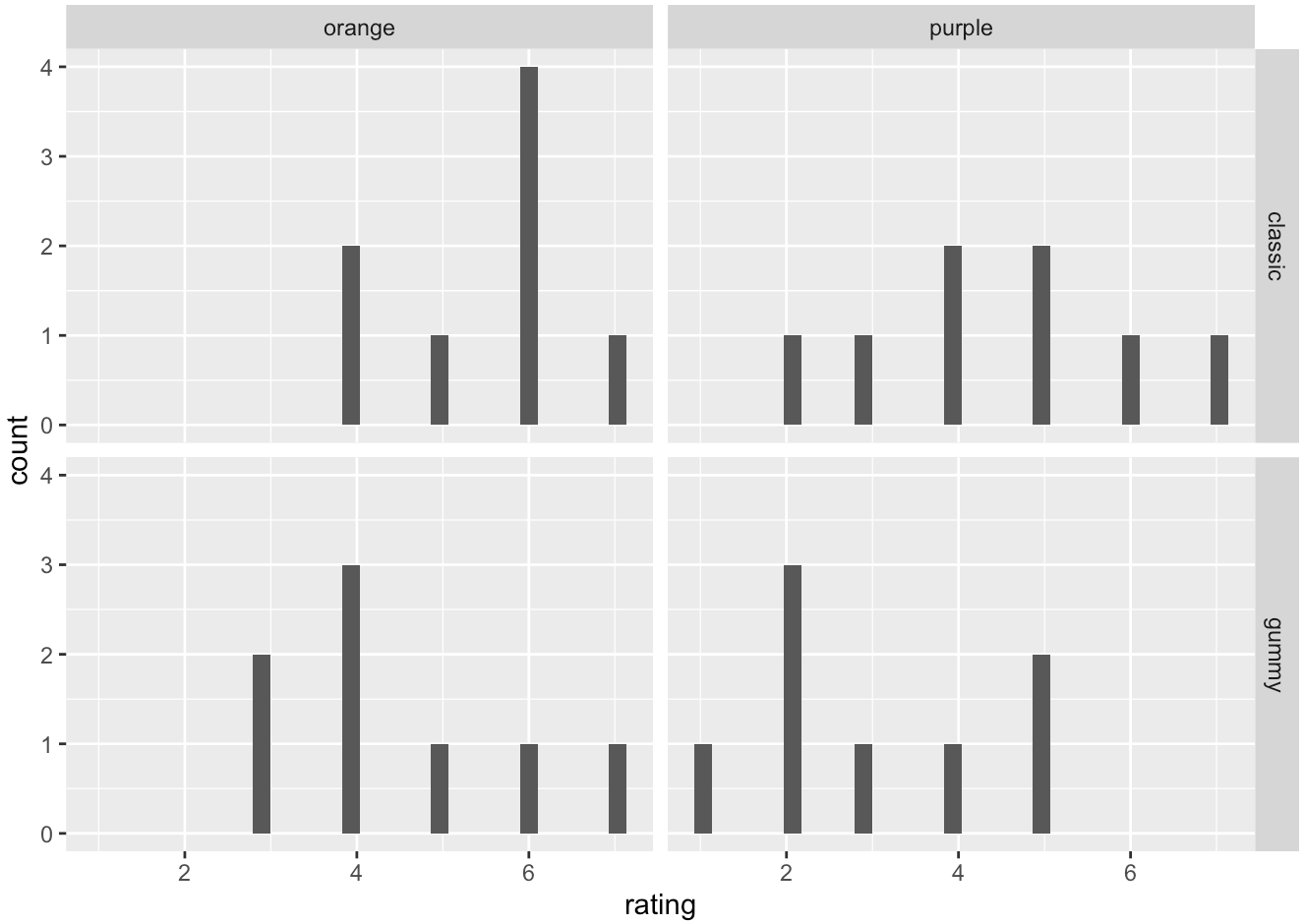
Because we have a 2 x 2 factorial research design, with two levels of color and two levels of texture, it's most appropriate to create a 2x2 grid of histograms using `facet_grid()`. The `~` operator is used in R to describe one variable across values of another.

Those two initial lines just set up the data framework to make a figure. In order to draw the actual plot, we use `geom_histogram()` to tell ggplot what kind of plot we want to make, and because these data are discrete-valued, we tell it to make fewer bins than its default, only `bins = 4`.

Here's the simplest grid of histograms:

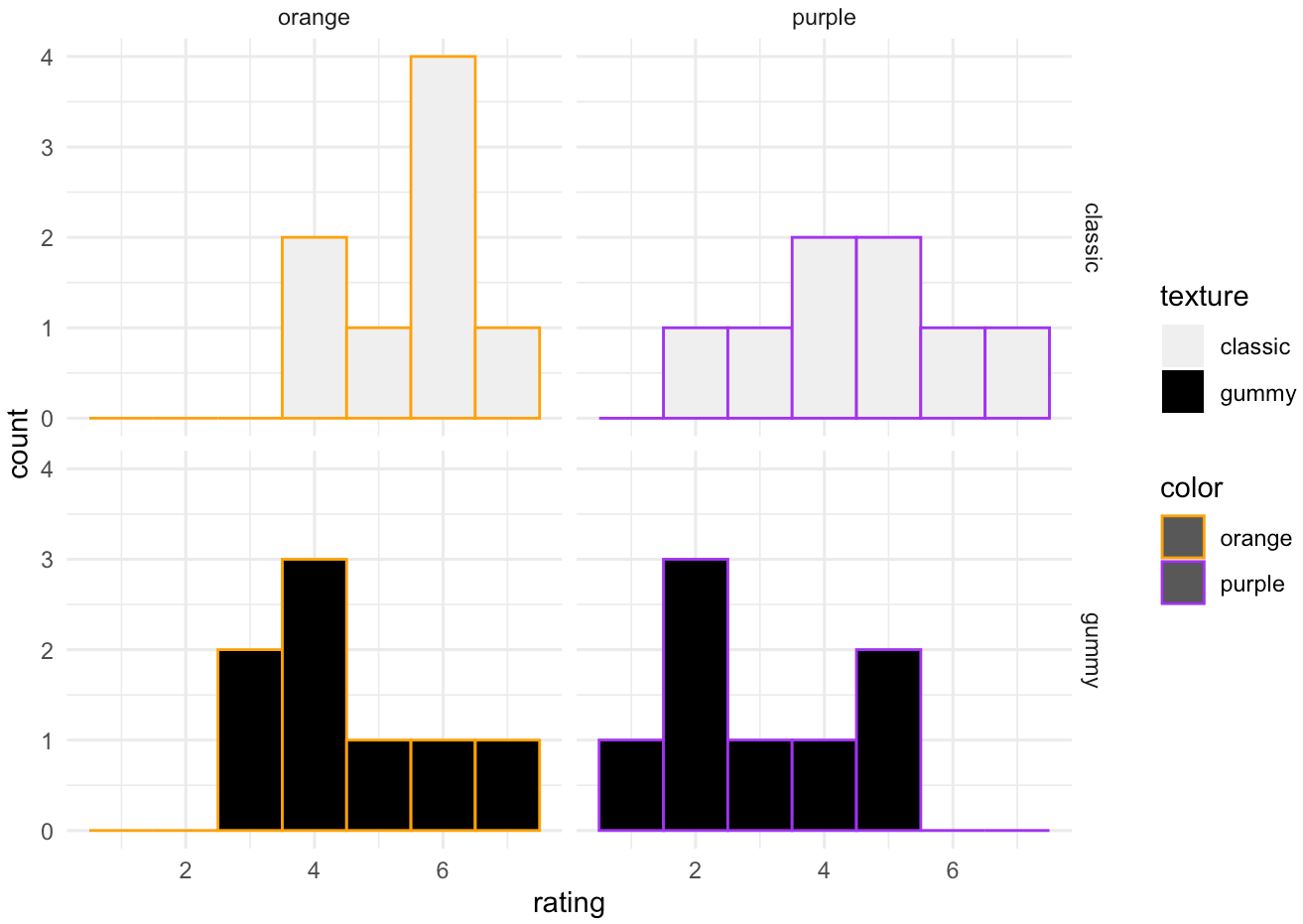
```
# Tell ggplot which data to use and how variables map to visual features (aes)
ggplot(data=data, aes(x = rating)) +
  # Facet across levels of texture and color
  facet_grid(texture ~ color) +
  # make a histogram
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



To make it a bit prettier, we can add color to the base aesthetics call. (If you're going to use colors from your data like this, triple-check that you've called them in the right order! Don't plot your purples in orange and your orange in purple.)

```
# Tell ggplot which data to use and how variables map to visual features (aes)
ggplot(data=data, aes(x = rating, color = color, fill = texture)) +
  # which variables should we facet across?
  facet_grid(____ ~ ____) +
  # force it to use a certain number of bins
  geom_histogram(bins=____) +
  # specify outline colors
  scale_color_manual(values = c("orange","purple")) +
  # specify fills
  scale_fill_manual(values=c("gray95","black")) +
  # themes are just for personal preference; I like this one
  theme_minimal()
```



After checking the histograms for any unexpected values, take a minute to think about our upcoming analysis. What condition appears to have the highest mean? The lowest? What main effects or interactions do you anticipate?

Descriptive stats

For a results section, we generally need a measure of central tendency and a measure of spread. For *interval* level data, these should be the mean and standard deviation. For *ordinal* level data, a median and interquartile range may be more appropriate.

```
# Take our processed data, and pipe to...
descriptives <- data %>%

  summarize(mean_rating = mean(rating), median_rating = median(rating), sd_rating = sd(r
ating), n = n())

# print out descriptives
descriptives
```

```
## # A tibble: 1 × 4
##   mean_rating median_rating sd_rating     n
##         <dbl>         <dbl>     <dbl> <int>
## 1        4.38             4      1.62    32
```

However, we probably actually want these for each cell of our 2x2 design, so we first need to *group* the data by texture and color.

```
# Take our processed data and pipe to...
descriptives2 <- data %>%

  # specify which groups
  group_by(color,texture) %>%
  # which function summarizes a dataset?
  ____(mean_rating = mean(rating), median_rating = median(rating), sd_rating = sd(ratin
g), n = n()) %>%
  ungroup() # always a good idea to ungroup!

descriptives2
```

```
## `summarise()` has grouped output by 'color'. You can override using the
## `.groups` argument.
```

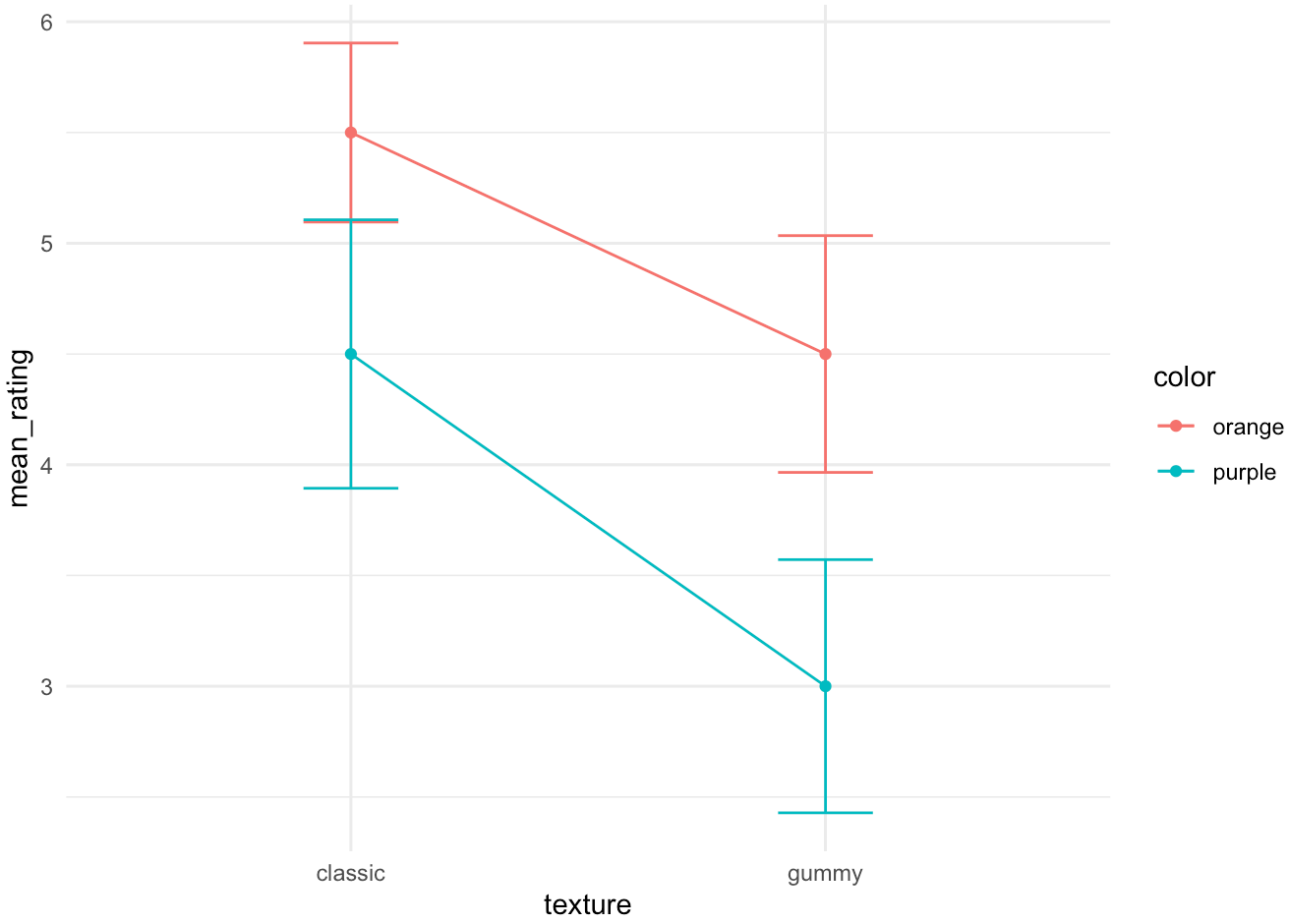
```
## # A tibble: 4 × 6
##   color  texture mean_rating median_rating sd_rating     n
##   <fct>  <fct>         <dbl>         <dbl>     <dbl> <int>
## 1 orange classic         5.5           6        1.07     8
## 2 orange gummy           4.5           4        1.41     8
## 3 purple classic         4.5           4.5      1.60     8
## 4 purple gummy           3             2.5      1.51     8
```

Finally, let's visualize these summary statistics.

```
# tell ggplot what data to use, and how to map variables to visual features.
# Put texture on the x axis, and use color to represent color.
ggplot(data = descriptives2, aes(___ = texture, ___ = color)) +

  # plot points representing the mean rating (as their y value), and a line connecting t
hem within each color
  geom_point(aes(___ = mean_rating)) +
  geom_line(aes(group = color, ___ = mean_rating)) +

  # add errorbars
  geom_errorbar(aes(ymin = mean_rating - (sd_rating/sqrt(n-1)), ymax = mean_rating + (sd
_rating/sqrt(n-1))), width = 0.2) +
  theme_minimal()
```



Inferential stats

So far, we have summarized the data we collected. One condition has the highest mean rating, and one has the lowest. The next question we ask is whether we can expect that *same pattern* to show up in the broader population. If we ran this experiment again, would we get the same result?

t-tests

To compare two means, we would use a *t*-test. (There's one handwave below that's not strictly accurate. Can you find it?)

```
# To test whether color has a significant effect, we run a t-test and save it into a mod
el variable

# To run a PAIRED ttest, we have to use different syntax than was in the example from th
e reading. Sorry, y'all.

model_t <- t.test(
  # compare ratings when color == color1 to ratings when color == color2
  data$rating[data$color=="____"], # data points in condition 1
  data$rating[data$color=="____"], # data points in condition 2
  # ratings are from the same participants
  paired = TRUE
)

# print the output
model_t
```

```
##
##  Paired t-test
##
## data:  data$rating[data$color == "orange"] and data$rating[data$color == "purple"]
## t = 3.371, df = 15, p-value = 0.0042
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  0.4596374 2.0403626
## sample estimates:
## mean difference
##            1.25
```

From the t-test results, make sure you understand whether we can expect color to affect taste rating in the real world.

ANOVA

More correctly, we have a *factorial* design. That is, two categorical predictor variables. That means we want to ask three difference-of-means questions at the same time:

1. Is there a **main effect of color**?
2. Is there a **main effect of texture**?
3. Is there an **interaction** between these factors?

We need a **factorial ANOVA**. The `aov_ez()` command lets us specify between- and within-subject factors. The sample code below assumes both `color` and `texture` were manipulated within-subjects; make sure your code matches your experimental design! `aov_ez()` specifies a linear model object; `anova()` pulls out the relevant pieces of that object and prints them as an ANOVA table.

```
model_1 <- aov_ez(
  data = data,
  id = "_____", # variable with participant IDs
  within = c("_____","_____"), # variables with within-subject factors
  dv = "_____" # variable with outcome values
)

anova(model_1)
```

```
## Anova Table (Type 3 tests)
##
## Response: rating
##                num Df den Df     MSE      F     ges  Pr(>F)
## color               1      7 1.71429 7.2917 0.18248 0.03063 *
## texture             1      7 2.28571 5.4688 0.18248 0.05196 .
## color:texture       1      7 0.57143 0.8750 0.00885 0.38071
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

What do you conclude about these effects from the ANOVA output?

Check assumptions

It's important to check these values, but don't stress too much about the answers. We know that, when there are the same number of samples in each group, ANOVA has good robustness to violations of these assumptions.

The `check_model()` function from the `performance` package doesn't work correctly for afex models

```
check_model(model_1,
            reformula = NA) # suppress a warning
```

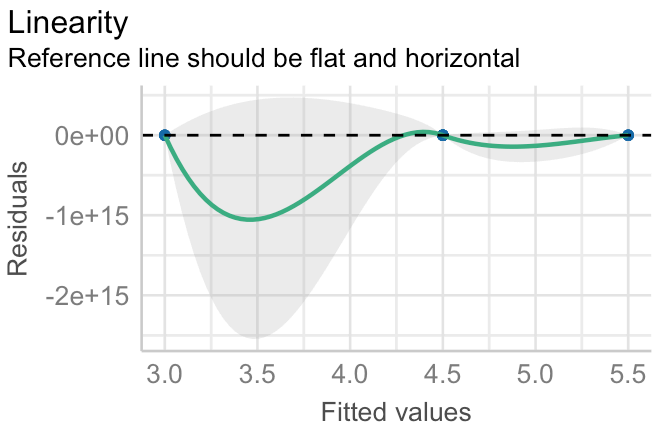
```
## Data was changed during ANOVA calculation. Thus, residuals cannot be added to origina
l data.
## residuals(..., append = TRUE) will return data and residuals.
```

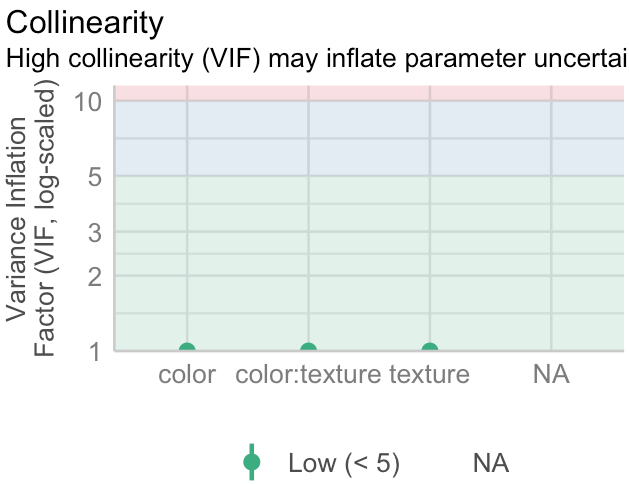
```
## Data was changed during ANOVA calculation. Thus, fitted values cannot be added to ori
ginal data.
## fitted(..., append = TRUE) will return data and fitted values.
```

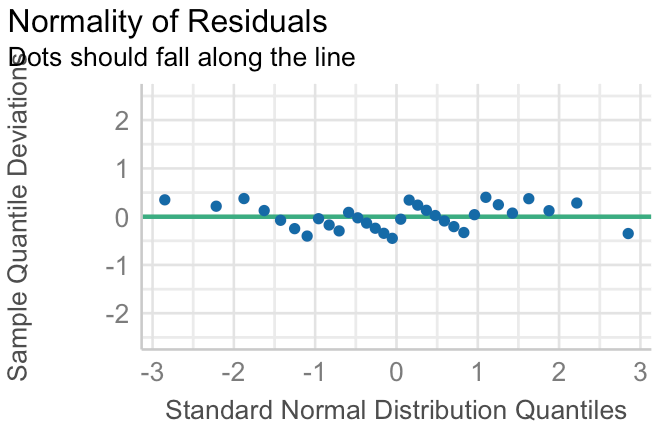

```
## Data was changed during ANOVA calculation. Thus, residuals cannot be added to origina
l data.
## residuals(..., append = TRUE) will return data and residuals.
```

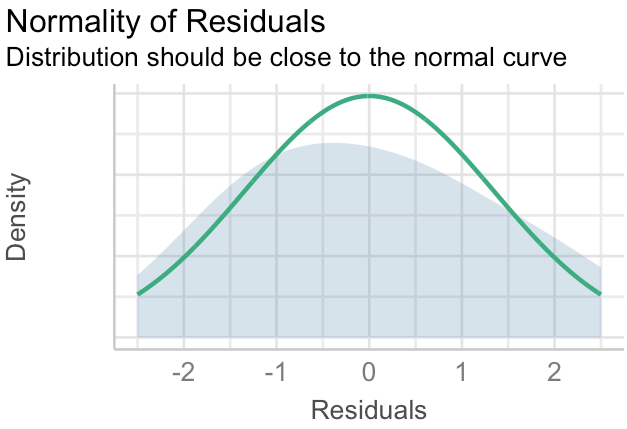
```
## Data was changed during ANOVA calculation. Thus, fitted values cannot be added to ori
ginal data.
## fitted(..., append = TRUE) will return data and fitted values.
```

```
## Data was changed during ANOVA calculation. Thus, residuals cannot be added to origina
l data.
## residuals(..., append = TRUE) will return data and residuals.
```










```
plot(check_heteroskedasticity(model_1))
```

```
## Data was changed during ANOVA calculation. Thus, fitted values cannot be added to ori
ginal data.
## fitted(..., append = TRUE) will return data and fitted values.
## Data was changed during ANOVA calculation. Thus, residuals cannot be added to origina
l data.
## residuals(..., append = TRUE) will return data and residuals.
```

```
## Data was changed during ANOVA calculation. Thus, fitted values cannot be added to ori
ginal data.
## fitted(..., append = TRUE) will return data and fitted values.
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
## : pseudoinverse used at 2.9875
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
## : neighborhood radius 1.5125
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
## : reciprocal condition number 4.3294e-18
```

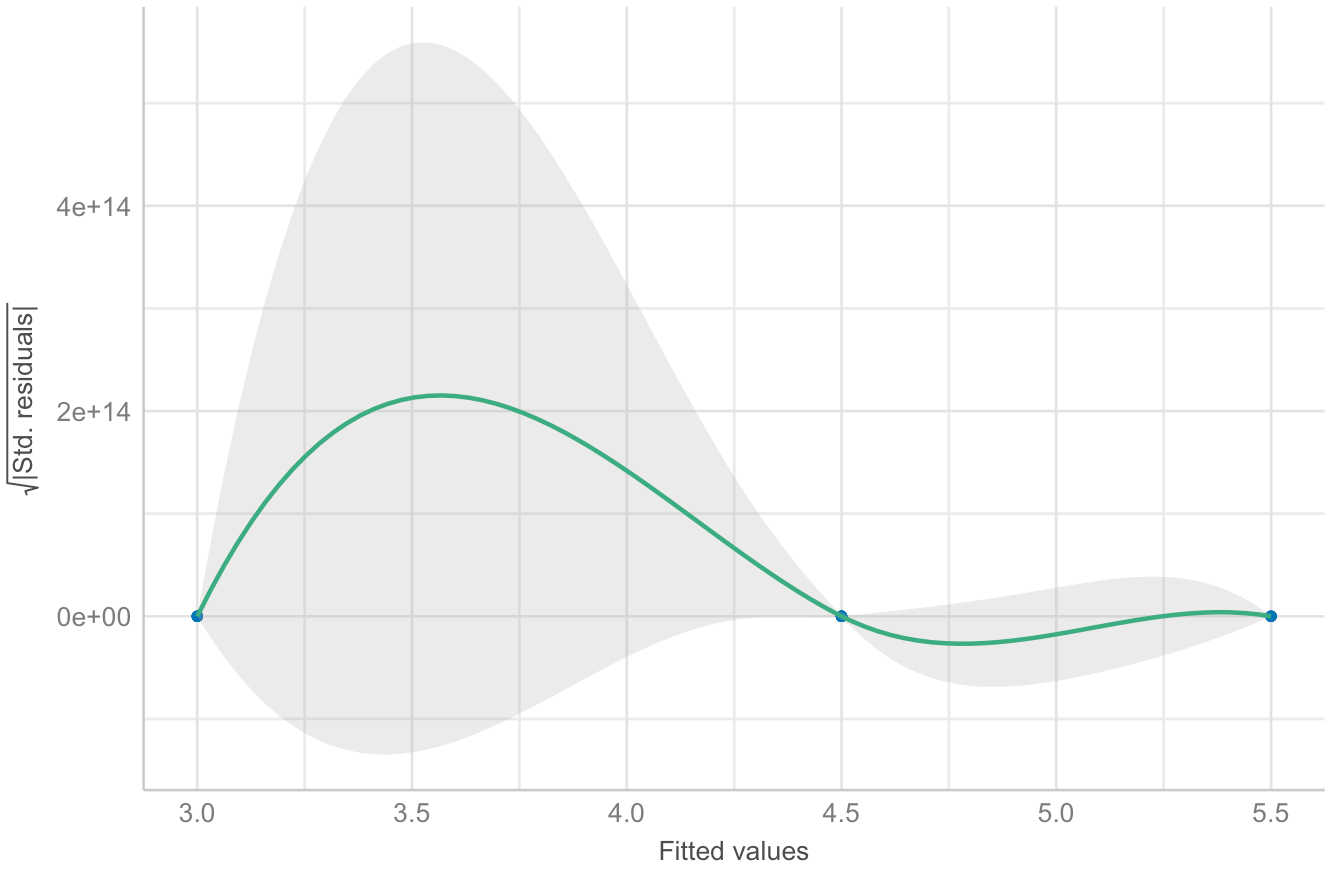
```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
## : There are other near singularities as well. 1.0252
```

```
## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
## else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at
## 2.9875
```

```
## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
## else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius
## 1.5125
```

```
## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
## else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition
## number 4.3294e-18
```

```
## Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
## else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : There are other near
## singularities as well. 1.0252
```

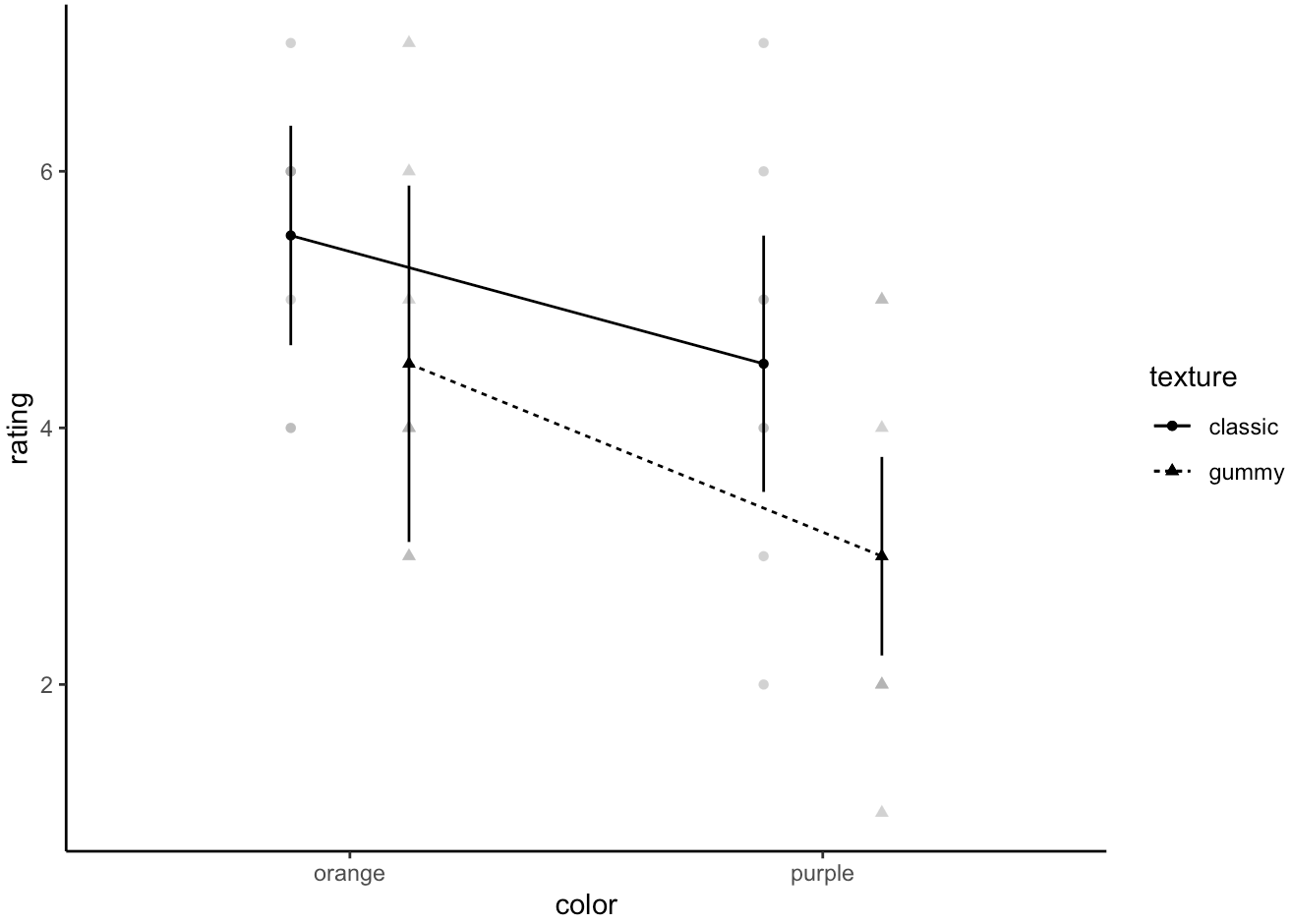


Plotting

`afex` has built-in plotting functionality:

```
afex_plot(model_1,           # model object
          x="color",         # assign a variable to X axis
          trace = "texture", # draw separate lines for a variable
          error_ci = TRUE,  # plot 95%CI error bars
          error = "within"  # draw within-subject error bars
          ) +

  theme_classic()
```

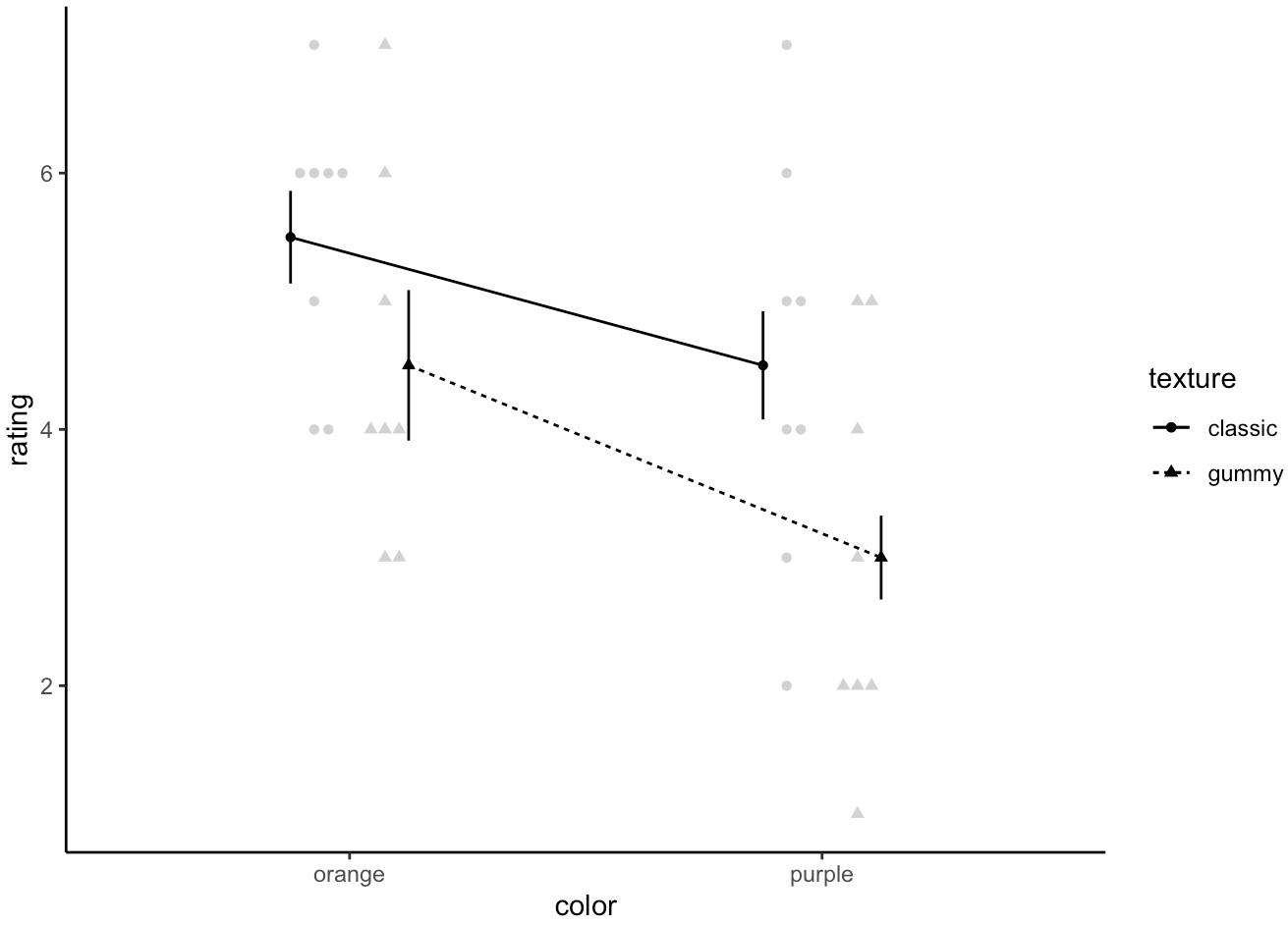


Because we have many data points with the same value, they're hard to see, so let's add some jitter so that the they don't land exactly on top of each other:

```
library(ggbeeswarm) # horizontal jitter for plots
dodge_value = 0.3 # if we change the geoms, we need to explicitly specify how far apart
to scoot the two conditions

afex_plot(model_1,           # model object
          x="color",         # assign a variable to X axis
          trace = "texture", # draw separate lines for a variable
          error_ci = FALSE,  # plot standard-error error bars
          error = "within",  # draw within-subject error bars
          data_geom = geom_beeswarm, # use geom_beeswarm to draw points
          data_arg = list(
                  dodge.width = dodge_value,  # needs to be same as dodge
                  cex = 1.5) # how much to spread
          ) +

  theme_classic()
```


rating
6
4
2
orange
purple
color
texture
classic
gummy