Repeated Measures

Neil J. Hatfield

Last Updated: December 03, 2023

In this tutorial we will take a look at Repeated Measures ANOVA models. Keep in mind that there are **two** different types of Repeated Measures models: applying every treatment to each subject/measurement unit (a.k.a. "Within Subjects") and measuring each subject/measurement unit multiple times but each only gets one treatment (a.k.a. "Nested Repeated Measures").

1 Important Starting Considerations

There are a few important considerations you need to think through before you get too far into the process of analyzing Repeated Measures designs.

1.1 Conventions

In Repeated Measure designs, our measurement units are not as straightforward as in our other designs. To help signal this, we often use the term *subjects* to denote the living being (or object) who produces the multiple observations. This has two important consequences.

First, our last node in the Hasse diagram will no longer be the measurement units (i.e., objects/living beings) but rather the observations. Second, our subjects are almost always presumed to be randomly sampled from a broader population. Thus, they are *random effects* and need to be marked as such in our Hasse diagram.

1.2 Identify Type of Repeated Measures

Repeated Measures Designs are almost exclusively used in experimental settings. To decide on which type of Repeated Measures Design you're facing, ask yourself the following questions:

- Who are the subjects?
- What are the treatments?
- How many different treatments does each subject get?
  - Answer: All of them → **Within Subjects Repeated Measures**
  - Answer: Only one → **Nested Repeated Measures**

1.3 Wide and Long Formats of Data Frames

In both types of Repeated Measures Designs, we will need the data to be arranged in two formats: "wide" and "long". These terms refer to the construction of the data frame.

A "long data frame" is what we're must used to working with. Here, each row represents a unique combination of Subject & Treatment or Subject & Time Point. If we have n subjects, and g treatments (or t time points of measurement), we should have a total of $n \cdot g$ rows (alternatively, $n \cdot t$ rows). Our response is a single column. Visually, imagine your data frame as a rectangle that is taller than it is wide.

A "wide data frame" is a re-arrangement. Here, each subject gets one and only one row. Instead of a single response column, we have multiple response columns. In fact, we'll have a separate response column for each of the g treatments (or t time points of measurement). Now imagine your data frame as a rectangle that is wider than it is tall.

One of the first challenges you must tackle in analyzing Repeated Measures data is identifying which of these formats your data is currently in. Then creating a new data frame that is in the other format. In these situations, I tend to not include `Data` on the end of my object name; rather, I use either `Long` or `Wide` so that I have a reminder of which format I'm calling.

1.3.1 Transforming Data Frame Formats

Thankfully, we have some useful functions to help us. As part of the `tidyverse`, the package `tidyr` gives us the functions `pivot_wider` and `pivot_longer`. Given the imagery of the rectangles, you can imagine turning (pivoting) the long rectangle into the wide rectangle and vice versa. This imagery can help you keep in mind that `pivot_wider` takes a long data frame and makes a wide data frame. The `pivot_longer` function starts with a wide data frame and returns a long data frame.

```
# Generic Demo Code for creating a wide data frame ----
# Note: this code assumes you already read in a long format data frame

#IMPORTANT: you will need to the code to match the names in your actual situation

dataWide <- pivot_wider(
  data = dataLong, # "dataLong" is the name of the long format data frame
  names_from = group, #"group" is the name of the column that contains your treatments
  values_from = response #"response" is the name of the column with the response values
)
```

```
# Generic Demo Code for creating a long data frame ----
# Note: this code assumes you already read in a wide format data frame

#IMPORTANT: you will need to the code to match the names in your actual situation

dataLong <- pivot_longer(
  data = dataWide, # "dataWide" is the name of the wide format data frame
  cols = !subject, # Says to not use the "subject" column
  names_to = "group", # This is the new column you want the treatments to go to
  names_transform = list(group = as.factor), # Makes the treatment column a factor
  values_to = "response" # A new column that will contain all of the response values
)
```

You'll be able to see the above code examples in action in the examples below.

2 Setting Up R

Just as in the prior guides/tutorials, we have to first ensure that `R` is properly configured and prepared for our work. We will want to ensure that we load all of the appropriate packages, set our constraint, and load in any additional tools. Many of the packages that we've used throughout the semester will continue to assist us now. [New] Packages that we will need to make sure that we load for Repeated Measures designs: `lme4`, `nlme`, and `rstatix`.

As a reminder, the following code does all of these things:

```
# Demo code to set up R ----
## Load packages
packages <- c("tidyverse", "knitr", "kableExtra",
              "parameters", "hasseDiagram", "car",
              "psych", "emmeans", "rstatix", "lme4",
              "nlme")
lapply(packages, library, character.only = TRUE, quitely = TRUE)

options(knitr.kable.NA = "")
options(contrasts = c("contr.sum", "contr.poly"))

source("https://raw.github.com/neilhatfield/STAT461/master/rScripts/ANOVATools.R")
```

3 Within Subjects Repeated Measures

The Within Subjects Repeated Measures Design has the hallmark that each subject will be given each and every treatment. To prevent an order effect, we should randomize the order in which each subject gets the treatments.

3.1 Example Context–Taste Testing Beer

Beer is big business; the craft brewing industry contributed $76.3 billion to the US Economy in 2021 and 490,000+ jobs. Getting a craft beer scored can be quite the achievement. In a single blind tasting, judges are given a chilled, properly poured beer and told the style category. They then judge the beer on Aroma (24 pts), Appearance (6 pts), Flavor (40 pts), Mouthfeel (10 pts), and Overall Impression (20 pts).

We have decided to put several State College beers to the test:

- Barnstormer (IPA, Happy Valley Brewing Company)
- Craftsman (Brown, HVBC)
- Red Mo (Red, Otto's)
- King Richard Red (Amber, Robin Hood)

For this study, we have used a lottery to select six individuals to act as judges. Each judge will be presented with samples of the four beers and they will score each beer. The order in which each judge samples/scores the beers will be determined by the research team drawing labeled tokens without replacement.

3.2 Fit the Model

To assess the appropriateness of ANOVA methods, we will want to turn towards our knowledge of the study design as well as the Hasse diagram.

3.2.1 Is ANOVA Appropriate?

For this particular study, we can express the Repeated Measures-Within Subjects design with the following Hasse diagram (Figure 1).

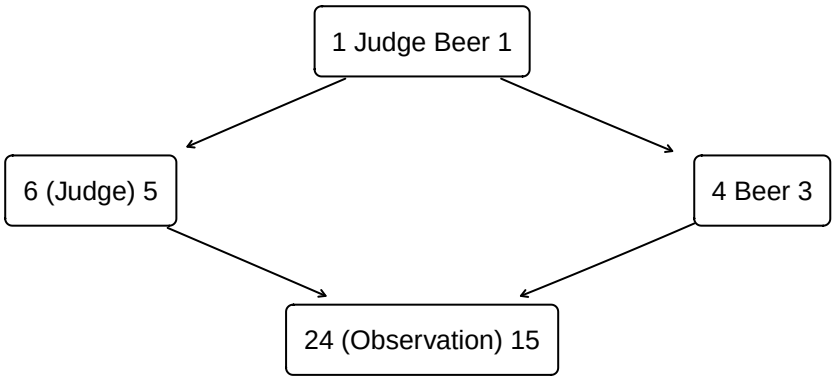

Figure 1: Hasse Diagram for Beer Judging

In looking at the Hasse diagram (Figure 1), we can see that we have sufficient *degrees of freedom* to estimate all effects and residuals/errors. From the design we know that we have a continuous response (beer rating) as well as a categorical factor (the beer). We are also using the same judges for each beer (i.e., our "subjects"), thus we are in a Within Subjects Repeated Measures design. (We also have an additive model.)

3.2.2 Get Data Ready

To form the models, we will first read in our data and then form both types of data frames. The following are the data for this situation:

```
# Demo Code for loading in Beer Data ----
## The data are stored in a wide format
beerWide <- read.table(
  file = "https://raw.github.com/neilhatfield/STAT461/master/dataFiles/beerJudging.csv",
  header = TRUE,
  sep = ","
)

## The order column contains the first letter of each
## beer in the order the judge tasted them.
```

```r
## Convert to a long format AND unpack the order column

beerLong <- pivot_longer(
  data = beerWide,
  cols = where(is.numeric),
  names_to = "beer",
  names_transform = list(beer = as.factor),
  values_to = "score",
) %>%
  rowwise() %>% # rowwise tells R to make changes only row by row
  mutate(
    order = str_locate(
      string = order,
      pattern = str_sub(string = beer, start = 1, end = 1)
    )[1] #[1] Gives us the correct index
  ) %>%
  mutate( # Optional: Get rid of periods in beer names
    beer = str_replace_all(string = beer, pattern = "\\.", replacement = " ")
  ) # Note: this will undo the factor type of beer

beerLong$beer <- as.factor(beerLong$beer)
beerLong$judge <- as.factor(beerLong$judge)
```

For assessing compound symmetry (sphericity), we will need the wide format. For fitting the actual model, we'll need the long format.

3.2.3 Form the Models

We are going to need to fit three (3) models for Within Subjects designs:

1) We will fit a One-way ANOVA + Block (RCBD) model for our ANOVA table.
2) We will fit a Mixed Effects model for estimating effects and assess assumptions about the judge factor (uses the `lme4` package).
3) We will use a Nested Approach via the `rstatix` package for assessing Compound Symmetry/Sphericity assumption.

```r
# Demo Code for Within Subject Repeated Measures ANOVA ----
## Omnibus Model (for our ANOVA table)
beerOmni <- aov(
  formula = score ~ judge + beer,
  data = beerLong
)

## Random Effect Model (Assumption Checking and Point Estimation)
beerMixed <- lme4::lmer(
  formula = score ~ (1|judge) + beer,
  data = beerLong
)

## Compound Symmetry/Sphericity Assessment
beerSphere <- rstatix::anova_test(
  data = beerLong,
  formula = score ~ beer + Error(judge %in% beer)
)
## The %in% tells R that judge should be treated as nested in beer
```

3.3 Assessing Assumptions

For Within Subjects Repeated Measures designs, we have the following assumptions:

1) Our residuals follow a Gaussian distribution,
2) Subject effects follow a Gaussian distribution,
3) Homoscedasticity around the model,

4) Independence of Subjects
5) No interaction between Subjects and Factor (just like RCBDs), and
6) We have Sphericity.

To assess these assumptions, we'll make use of many of the same tools that we've been using throughout the semester.

3.3.1 Gaussian Residuals

Just as in other situations, we'll use a QQ Plot to assess whether our residuals follow a Gaussian distribution. Notice that we will use the `beerMixed` model (using the `lme4` package).

```
# Demo Code for QQ plot for residuals ----
car::qqPlot(
  x = residuals(beerMixed), # Notice which model gets used here
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
  ylab = "Residuals (score)"
)
```

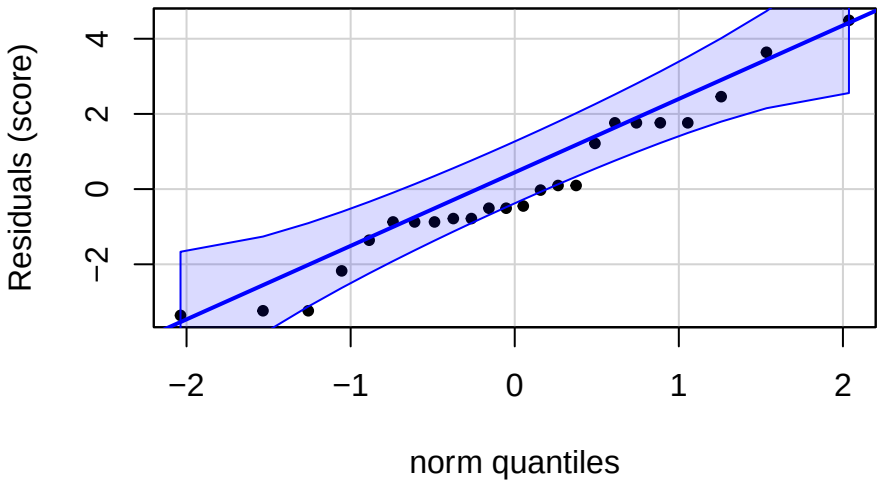

Figure 2: QQ Plot of Beer Judging Residuals

While there are roughly three points outside of the envelope (see Figure 2), this is only about 12% of the observations. I would supplement this plot with the values of the *Sample Skewness* and *Sample Excess Kurtosis* to help make my final decision. However, I would lean towards treating this assumption as satisfied.

3.3.2 Gaussian Subject Effects

We will also use a QQ plot for assessing our assumption of a Gaussian subject effect. However, rather than looking at the residuals, we need to get the effects of our subjects. To do this we'll need to use the `ranef` function ("random effects") from the `lme4` package on the `beerMixed` model.

```
# Demo Code QQ Plot of Judge Effects ----
car::qqPlot(
  x = unlist(
    lme4::ranef(
      object = beerMixed, ## Notice which model is getting used
      whichel = c("judge")
    )
  ),
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
```

```
  ylab = "Judge Effects"
)
```

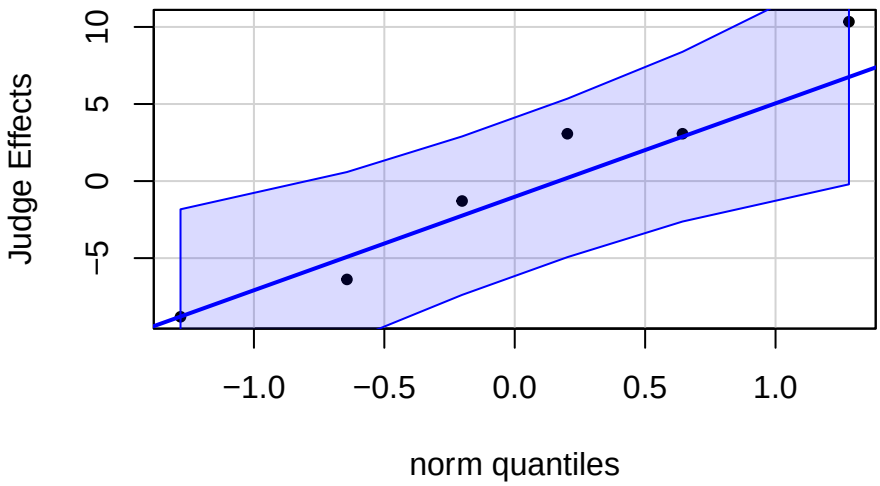

Figure 3: QQ Plot for Judge Effects

From Figure 3, we can be satisfied with this assumption.

3.3.3 Homoscedasticity

For assessing homoscedasticity in a Within Subjects design, we'll make use of *two* visualizations: the Tukey-Anscombe plot and a residual dot plot. In both cases, we will use the `beerMixed` model.

```
# Demo Code for the Tukey-Anscombe Plot ----
ggplot(
  data = data.frame(
    residuals = residuals(beerMixed), # Notice which model
    fitted = fitted.values(beerMixed)
  ),
  mapping = aes(x = fitted, y = residuals)
) +
  geom_point(size = 2) +
  geom_hline(
    yintercept = 0,
    linetype = "dashed",
    color = "grey50"
  ) +
  geom_smooth(
    formula = y ~ x,
    method = stats::loess,
    method.args = list(degree = 1),
    se = FALSE,
    linewidth = 0.5
  ) +
  theme_bw() +
  xlab("Fitted values (score)") +
  ylab("Residuals (score)")
```

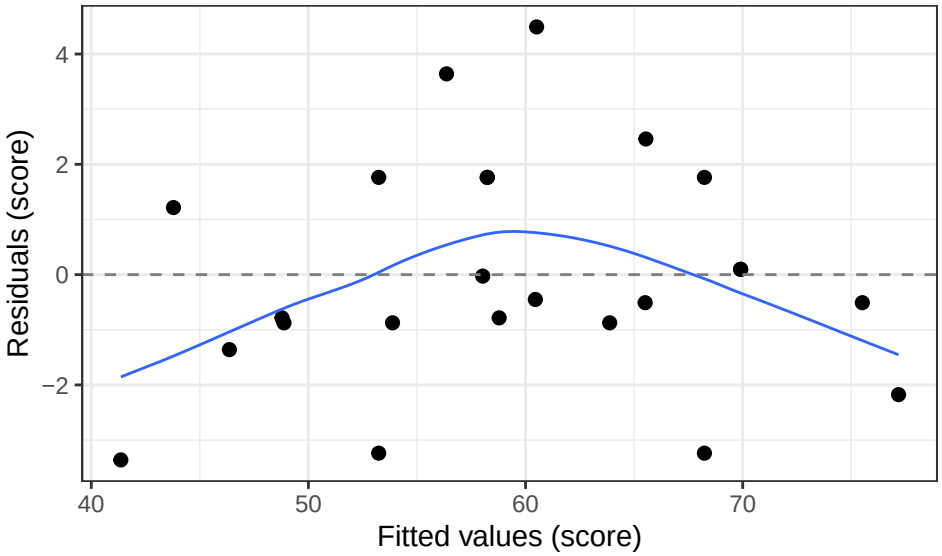

Figure 4: Tukey-Anscombe Plot for Beer Judging Study

3.3.3.1 Tukey-Anscombe Plot From Figure 4, I'm a little concerned about heteroscedasticity but not overly much.

3.3.3.2 Residual Dot Plots We can also use a residual dot plot (as shown in Figure 5) to look at whether we have homoscedasticity. This plot is similar to a strip chart and can help us make better informed decisions about the extent of issues shown in a Tukey-Asncombe plot.

```
# Demo Code for the the Residual Dot Plot ----
ggplot(
  data = data.frame(
    beer = beerLong$beer,
    residuals = residuals(beerMixed)
  ),
  mapping = aes(x = residuals)
) +
  geom_dotplot(
    method = "histodot",
    binwidth = 0.1,
    right = FALSE,
    origin = 0,
    dotsize = 2,
    stackratio = 1.1,
    binpositions = "all"
  ) +
  xlab("Residual (score)") +
  theme_bw() +
  facet_wrap(
    facets = vars(beer),
    ncol = 1,
    strip.position = "right",
    labeller = label_wrap_gen(width = 13)
  ) +
  theme(
    text = element_text(size = 12),
    axis.title.y = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks.y = element_blank(),
    strip.background = element_rect(
      linewidth = 25,
      color = "white",
      fill = "black")
```

```
  )
```

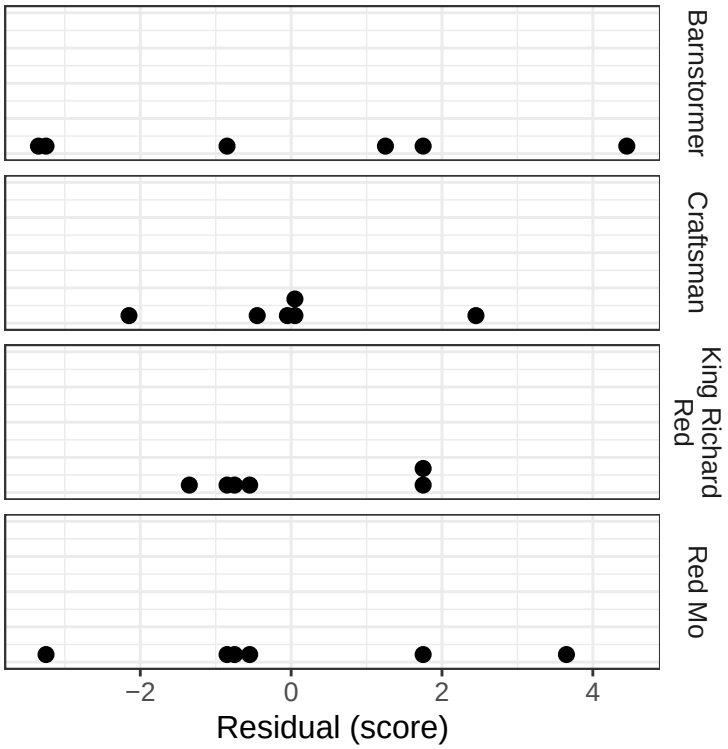

Figure 5: Residual Dot Plot for Beer Judging Study

As we look at Figure 5, we can see that we're right on the cusp of a potential issue. The variation (as spread) for the King Richard Red is just under half of that of the Barnstormer. The narrowing of King Richard Red and of Craftsman could explain the parabolic shape in the Tukey-Anscombe plot (Figure 4). With this information, I'm now less concerned about homoscedasticity being violated.

3.3.4 Independence of Subjects

As with Independence of Observations, the best place to begin assessing this assumption is the study design. Review how researchers selected the subjects and how they collected the data to see if there might be any potential issues. If you know measurement order, then you can look at an index plot. However, be sure that you incorporate your subject identifiers so that you can rule out any patterns that are the result of just switching to a new subject (i.e., switching of the fields in our Barley block example).

3.3.5 Interaction of Subject and Factor

We will want to make an interaction plot to look for consistency between the judges and the beers; again, think about what we would do with a RCBD.

```
# Demo Code for an Interaction Plot ----
ggplot(
  data = beerLong,
  mapping = aes(
    x = beer,
    y = score,
    color = judge,
    shape = judge,
    group = judge
    )
) +
  geom_point(size = 2) +
  geom_line() +
  ggplot2::theme_bw() +
```

```
  xlab("Beer") +
  ylab("Score") +
  labs(
    color = "Judge",
    shape = "Judge"
  ) +
  scale_color_manual(values = boastUtils::boastPalette) +
  scale_x_discrete(
    labels = label_wrap_gen(width = 12)
  )
```

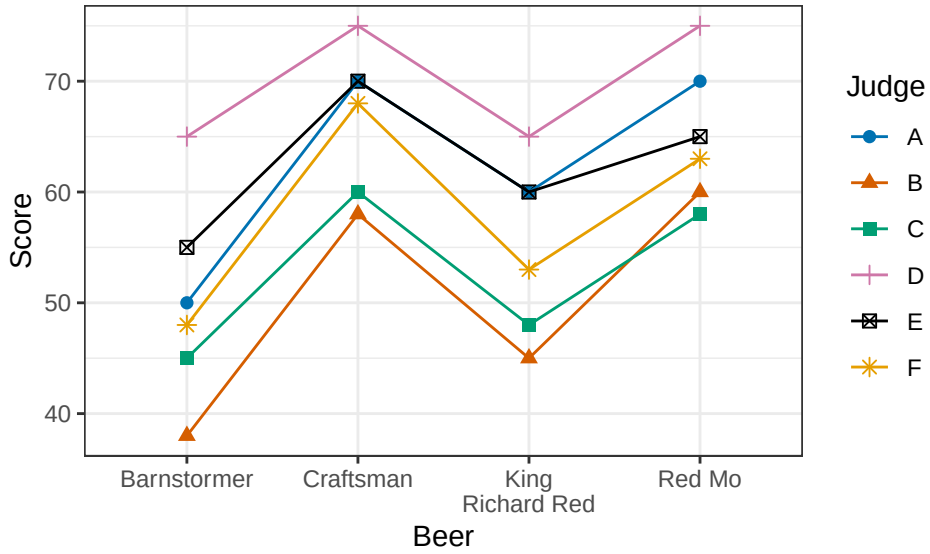

Figure 6: Interaction Plot for Beer and Judge

In Figure 6, we can see that as we move from beer to beer, the same general trend holds true for all judges. This indicates that there is not an interaction between judge (subject; our block) and beer (our factor).

In the event that you have more than 6-8 subjects, line plots may be challenging to make sense of; for example, think about the line plot for our snicker doodle study. In such cases, you might create a set of plots ("small multiple") through the use of a faceting attribute (e.g., breaking the 18 taste testers up into two sections) or use an alternative visualization (e.g., a heat map).

```
# Demo Code for Heat Map ----
ggplot(
  data = beerLong,
  mapping = aes(x = judge, y = beer, weight = score)
) +
  geom_bin_2d() +
  theme_bw() +
  xlab("Judge") +
  ylab("Beer") +
  scale_y_discrete(
    labels = label_wrap_gen(width = 10)
  ) +
  scale_fill_gradient2(name = "Score")
```

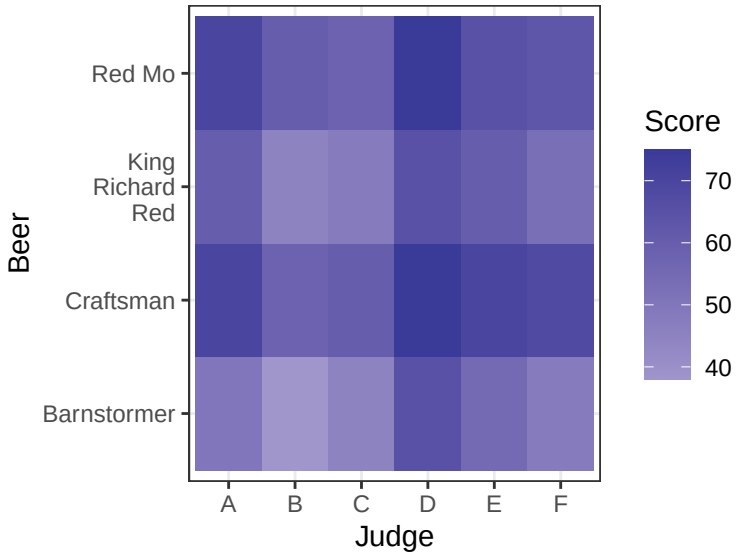

Figure 7: Heat Map Interaction Plot for Beer Judging Study

While in the interaction plot (Figure 6) we're looking for consistency via parallelism of line segments, we want to look for consistency in Figure 7. Instead of [near] parallel lines, we want to see the same general pattern as we move through the columns. For example, Judge A's column has Red Mod and Craftsman as darker (i.e., higher scores) while King Richard Red and Barnstromer are lighter (i.e., lower scores). That pattern of which beers are darker/lighter holds across each of the judges. This speaks to there not being an interaction between judge and beer that would need to be modeled.

Taking both Figures 6 and 7, we can be assured that we don't have any interaction effects that we need to model.

3.3.6 Sphericity

The idea behind sphericity is that we have essentially the same levels of variation for the *differences between treatments*. While there is a visual method we can use here, we do need to do so with some caution. Much like looking for homoscedasticity, we'll want to see if any difference has *excessively different* variation than another difference. Unlike homoscedasticity **there is no rule of thumb/guideline** (e.g., more than twice) for sphericity. Thus, this is one assumption where we will supplement with a formal test: Mauchly's Test of Sphericity.

```
# Demo Code for Sphericity Plot ----
# Note: to use color = "boast" or "psu" you'll need to install the boastUtils package
# devtools::install_github("EducationShinyAppTeam/boastUtils")
sphericityPlot(
  dataWide = beerWide, # Data needs to be in wide format
  subjectID = "judge", # character name of the subject column
  colsIgnore = c("order"),
  colors = "boast" # Optional; "default", "boast", "psu"
)
```

The `sphericityPlot` function comes from my `ANOVATools.R` script and requires ***wide format*** data. The call returns a plot like you see in Figure 8. The horizontal placement of the points does not mean much as there is some horizontal jitter in place so that the points don't all lie on top of one another. The vertical placements are insightful. These related to the differences in treatments. Much like a strip chart, we're looking to see if any comparison uses up a different amount of vertical space. As I look through Figure 8, I see that King Richard Red vs. Craftsman uses the least amount of vertical space, but not excessively so given the others. My initial thought would be that sphericity is satisfied.

To supplement the plot, we'll turn to Mauchly's test. The null hypothesis for Mauchly's Test is that there is ***no*** violation of Sphericity (Compound Symmetry); under this hypothesis, Mauchly's Test Statistic, W, follows a χ^2 with 2 *degrees of freedom*. To see the results of Mauchly's test, we will use the following code:

```
# Demo Code for Mauchly's Test ----
beerSphere$`Mauchly's Test for Sphericity` %>%
  dplyr::select(Effect, W, p) %>%
  knitr::kable(
    digits = 4,
```

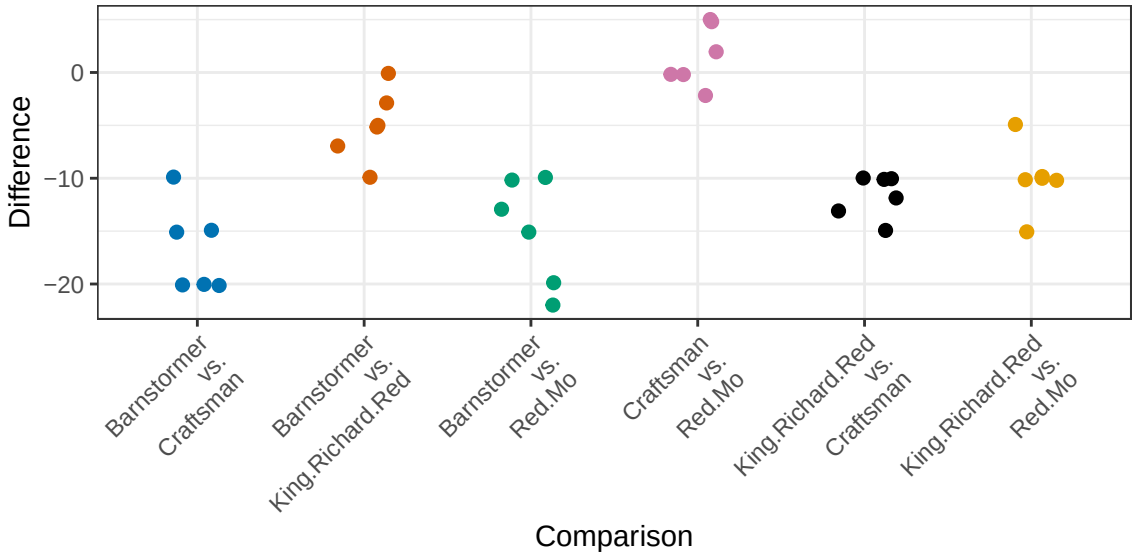

Figure 8: Sphericity Plot for Beer Judging

```
  col.names = c("Effect", "Mauchly's W", "p"),
  caption = "Mauchly's Sphericity Test",
  align = c('l',"c","c"),
  booktab = TRUE
) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )
```

Table 1: Mauchly's Sphericity Test

Effect	Mauchly's W	p
beer	0.365	0.597

Thus, we have $W = 0.365$ with a p-value of 0.597 (use your overall Type I Error Risk to set the Unusualness Threshold here). This is one of those cases where we *want to fail to reject* the null hypothesis.

Taken together, we will say that the sphericity is satisfied.

3.3.7 Within Subjects Interference Checks

While not an assumption of the parametric shortcut, checking for interference effects (i.e., order effects, carryover effects) when you're assessing assumptions is a wise move. To do this, we'll make use of a Residual Sequence Plot.

```
# Demo Code for Residual Sequence Plot ----
cbind(
  beerLong,
  residuals = residuals(beerMixed) ## Notice which model is getting used
) %>%
  ggplot(
    mapping = aes(x = order, y = residuals)
  ) +
  geom_point() +
  geom_line() +
  geom_hline(
    yintercept = 0,
```

```
    linetype = "dashed",
    color = "grey50"
  ) +
  theme_bw() +
  xlab("Tasting Order") +
  ylab("Residuals (score)") +
  facet_wrap(facets = vars(judge))
```

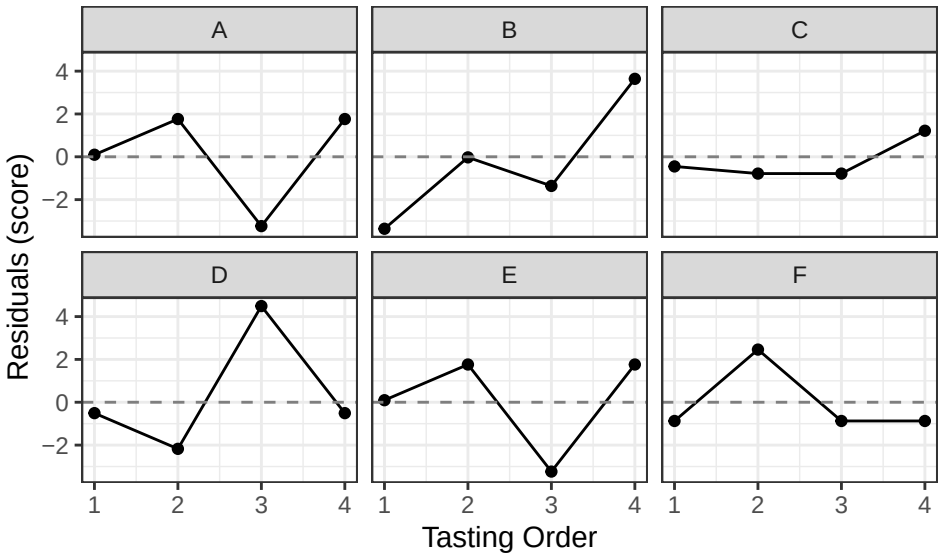

Figure 9: Residual Sequence Plot for Beer Judging Study

Figure 9 shows the residuals for each judge's scores, ordered by how the judge's tasted the beers. We are looking for anything that would suggest that there might be a relationship between the residuals and the tasting order. When I look at this plots, I don't see anything that would make me thing that we have significant interference effects to deal with.

3.4 Results

As we have been doing all semester, we can divide our results into an Omnibus portion, a Point Estimates portion, and Post Hoc portion.

3.4.1 Omnibus Test

For our omnibus test, we will use the same approach as we did for the RCBDs.

```
# Demo code of omnibus test ----
parameters::model_parameters(
  model = beerOmni, # Notice which model we're using here
  effectsize_type = c("eta", "omega", "epsilon")
) %>%
  dplyr::mutate(
    p = ifelse(
      test = is.na(p),
      yes = NA,
      no = pvalRound(p)
    )
  ) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Source", "SS", "df", "MS", "F", "p-value",
                  "Partial Eta Sq.", "Partial Omega Sq.", "Partial Epsilon Sq."),
    caption = "ANOVA Table for Beer Judging Study",
    align = c('l',rep('c',8)),
    booktab = TRUE
```

```
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("scale_down", "HOLD_position")
  )
```

Table 2: ANOVA Table for Beer Judging Study

Source	SS	df	MS	F	p-value	Partial Eta Sq.	Partial Omega Sq.	Partial Epsilon Sq.
judge	1045.833	5	209.1667	32.8534	< 0.0001	0.9163	0.869	0.8884
beer	1150.000	3	383.3333	60.2094	< 0.0001	0.9233	0.881	0.9080
Residuals	95.500	15	6.3667					

Notice that I slipped the `pvalRound` function in to take care of the fact that the p-values are excessively small and were coming out as 0 after rounding. We interpret the terms in this table exactly as we have been all semester.

3.4.1.1 Efficiency of Repeated Measures

Since the One-way Within Subjects subjects is like a RCBD, we can get a measure of the efficiency of such a design versus a completely randomized (one-way) design (CRD).

```
# Use the block relative efficiency function ----
block.RelEff(
  aov.obj = beerOmni,
  blockName = "judge",
  trtName = "beer"
)
```

```
## [1] "The relative efficiency of the block, judge, is 7.715."
```

Thus, we would need 8 times as many ratings for each beer as what we used in order to get the same level of information. That would mean that we would need around 48 scores for each beer.

3.4.1.2 Example Write Up

> For our Within Subjects Repeated Measures design, we can see that there is a statistically significant difference in the scores that the judges gave due to the type of beer ($F(3, 15) \approx 60.2$, $p < 0.0001$). The beer type accounted for just over 60 times as much variation as left unexplained, even after accounting for judge effects. Under the null hypothesis of no effect due to beer, we would only anticipate seeing such an extreme F ratio, less than 1/100th of a percent of the time. Further, we can see from the rather large effect sizes, that beer type accounts for around 90% of the variation in the judges' final scores (see Table 2). The relative efficiency of our Within Subjects design is approximately 7.7; thus, we would need 8 times as many scores for each beer as what we currently have to get the same level of information.

3.4.2 What if Sphericity is Violated?

If Sphericity is violated (i.e., Mauchly's Test leads you to reject the null hypothesis), we are not out of luck. When we fit the model to check for Sphericity, we also automatically got two corrected tests: the Greenhouse-Geisser and the Huynh-Feldt corrections:

```
# Demo Code for Corrected Omnibus p-values ----
correctedTable <- beerSphere$`Sphericity Corrections` %>%
  dplyr::select(GGe, `p[GG]`, HFe, `p[HF]`)
correctedTable$`p[GG]` <- lapply(
  X = correctedTable$`p[GG]`,
  FUN = pvalRound
)
correctedTable$`p[HF]` <- lapply(
  X = correctedTable$`p[HF]`,
  FUN = pvalRound
)
```

```
knitr::kable(
  x = correctedTable,
  digits = 4,
  col.names = c("Greenhouse-Geisser", "p-value", "Huynh-Feldt", "p-value"),
  caption = "Sphericity Corrections",
  align = "c",
  booktab = TRUE
) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )
```

Table 3: Sphericity Corrections

Greenhouse-Geisser	p-value	Huynh-Feldt	p-value
0.613	< 0.0001	0.952	< 0.0001

For the Corrections, the p-values are the adjusted p-values for the **omnibus** test. Thus, we would say that there is an effect due to our repeated measures model (i.e., there is a difference in the scores of the beer). If sphericity is violated, use these p-values instead of the ones from the `beerOmni` model.

3.5 Point Estimates

Due to the mixed effects model (i.e., the random effect of our subjects (judges)), we cannot use the `dummy.coef` call to get point estimates. We need to use the `emmeans` package.

```
# Demo Code for Point Estimates using emmeans package ----
beerPH <- emmeans::emmeans(
  object = beerMixed, # Notice the use of the mixed model here
  specs = pairwise ~ beer,
  adjust = "tukey",
  level = 0.92
)

## Point Estimates
as.data.frame(beerPH$emmeans) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Type of Beer", "Marginal Mean","SE", "DF",
                  "Lower Bound","Upper Bound"),
    caption = "Marginal Means-Tukey 92\\% Adjustment",
    align = c("l", rep("c", 5)),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )
```

Table 4: Marginal Means-Tukey 92% Adjustment

Type of Beer	Marginal Mean	SE	DF	Lower Bound	Upper Bound
Barnstormer	50.1667	3.084	5.9383	43.6635	56.6698
Craftsman	66.8333	3.084	5.9383	60.3302	73.3365
King Richard Red	55.1667	3.084	5.9383	48.6635	61.6698
Red Mo	65.1667	3.084	5.9383	58.6635	71.6698

Example statements:

- The Craftsman beer accumulated points at a rate of 66.83 points per judge.
- Red Mo's total score was 65.16 times as large as the number of judges who scored the beer.

3.5.1 Post Hoc Analysis–Pairwise Comparisons

For Post Hoc Analysis, we will make use of the `emmeans` package and make sure that we're looking at the correct aspect of our model. We already assume that there's some difference in the judges, thus we are really just after the marginals of our other factor(s). In this situation, the type of beer.

You can also do the standard pairwise comparisons of the beer types.

```
# Demo Code for Post Hoc Pairwise Comparisons ----
## Notice we're using the beerPH object from the point estimates code
as.data.frame(beerPH$contrasts) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Comparison", "Difference","SE", "DF",
                  "t Statistic","p-value"),
    caption = "Marginal Means-Tukey 92\\% Adjustment",
    align = c("l", rep("c", 5)),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )
```

Table 5: Marginal Means-Tukey 92% Adjustment

Comparison	Difference	SE	DF	t Statistic	p-value
Barnstormer - Craftsman	-16.6667	1.4568	15	-11.4407	0.0000
Barnstormer - King Richard Red	-5.0000	1.4568	15	-3.4322	0.0174
Barnstormer - Red Mo	-15.0000	1.4568	15	-10.2966	0.0000
Craftsman - King Richard Red	11.6667	1.4568	15	8.0085	0.0000
Craftsman - Red Mo	1.6667	1.4568	15	1.1441	0.6692
King Richard Red - Red Mo	-10.0000	1.4568	15	-6.8644	0.0000

Example Statements (see effect size section too)

> Amongst the four beers, there appear to be significant differences in how the judges scored them with one notable exception: Craftsman and Red Mo. The judges did not seem to coalesce around one of these beers being higher or lower rated than the other (p-value of 0.67).

3.5.2 Effect Sizes

For Post Hoc Effect Sizes, you'll need to use the `emmeans` package and my `probSup` function:

```r
# Demo Code for Post Hoc Effect Sizes ----
tempEMM <- emmeans::emmeans(
  object = beerMixed, # Notice we're using the beerMixed model
  specs = "beer"
)

# Pass the stored marginals into the effect size function
cohenTemp <- emmeans::eff_size(
  object = tempEMM,
  sigma = sigma(beerMixed),
  edf = df.residual(beerMixed)
)

# Create a data frame, add on the probability of superiority
# Send that data frame into a nice table
as.data.frame(cohenTemp) %>%
  dplyr::mutate(
    ps = probSup(effect.size),
    .after = effect.size
  ) %>%
  dplyr::select(contrast, effect.size, ps) %>%
  knitr::kable(
    digits = 3,
    col.names = c("Comparison", "Cohen's d", "Probability of Superiority"),
    align = "lcc",
    caption = "Effect Sizes for Beer",
    booktab = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = "HOLD_position"
  )
```

Table 6: Effect Sizes for Beer

Comparison	Cohen's d	Probability of Superiority
Barnstormer - Craftsman	-6.605	0.000
Barnstormer - King Richard Red	-1.982	0.081
Barnstormer - Red Mo	-5.945	0.000
Craftsman - King Richard Red	4.624	0.999
Craftsman - Red Mo	0.661	0.680
King Richard Red - Red Mo	-3.963	0.003

Example Statements

> There appears to be a clear difference between Craftsman and King Richard Red: almost 100% of the time we give a judge a sample of both of these beers to score, they will give the higher score to the Craftsman.

4 Nested Repeated Measures Design

The second kind of Repeated Measures ANOVA design deals with taking multiple measurements from our measurement units on the same attribute over time. Unlike the Within Subjects design, each measurement unit here only gets **ONE** treatment. A somewhat handy way to help you decide if you're in a Nested Repeated Measures design is to see if you can think about the situation as being like a Pre-/Post-Testing situation. This classic situation involves testing/measuring everyone before we apply treatments, then apply the treatments (each person only gets one), and then testing/measuring everyone again afterwards. If you can fit the situation into the pre/post design, you're a Nested Repeated Measures design.

4.1 Example Context–Shoe Advertizing and Sales

For this example, we are going to look at the impact of two advertising campaigns on the volume of sales of athletic shoes over time. Ten similar test markets were selected at random to participate in this study. The two advertising campaigns were similar in all respects except that a different national sports personality was used in each. Sales data were collected for three two-week periods (before-t1, during-t2, and after-t3)

4.2 Fit the Model

To assess the appropriateness of ANOVA methods, we will want to turn towards our knowledge of the study design as well as the Hasse diagram.

4.2.1 Is ANOVA Appropriate?

Checking the appropriateness of ANOVA methods, including the Nested Repeated Measures designs, follows all of the same patterns as before. However, if you use the Hasse Diagram app, you'll need to watch out for a couple of things:

1) You'll need to remove the interaction of Time Point X Market Nested Campaign. (Use Markets X Time as your measurement unit.)
2) The *degrees of freedom* will be off for the Markets. The app isn't subtracting the *degrees of freedom* for Campaign. Thus, you'll have to manually adjust the code until I can get a fix in place.
3) Remember to carry your *degrees of freedom* fix through to the final node.

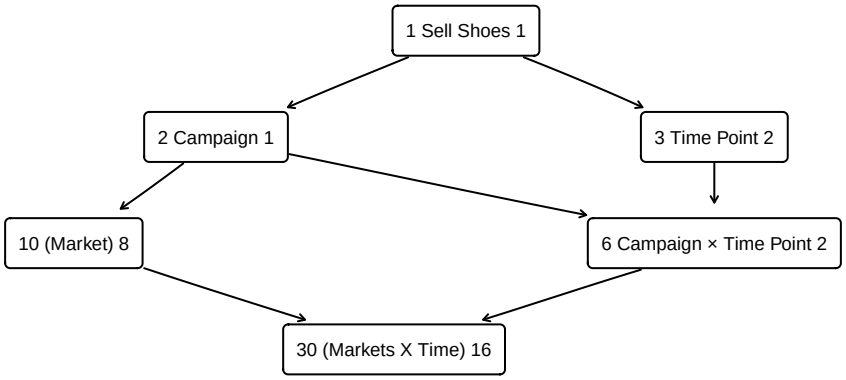

Figure 10: Hasse Diagram for Shoe Advertisement Study

4.2.2 Get the Data Ready

The shoe sales data comes to use in the wide format. We will want to make a long format version as well.

```
# Demo Code for reading in data ----
shoesWide <- read.table(
  file = "https://raw.github.com/neilhatfield/STAT461/master/dataFiles/shoes.csv",
  header = TRUE,
  sep = ","
)

shoesWide$campaign <- as.factor(shoesWide$campaign)
shoesWide$market <- as.factor(shoesWide$market)

# Make a long version of the data
shoesLong <- tidyr::pivot_longer(
  data = shoesWide,
  cols = dplyr::starts_with("t"),
  names_to = "time",
  names_ptypes = list("time" = factor()),
  values_to = "sales"
) %>%
  mutate(
    time = case_match(
```

```
      .x = time,
      "t1" ~ "before",
      "t2" ~ "during",
      "t3" ~ "after",
      .ptype = factor(levels = c("before", "during", "after"))
    )
  )
# Recode times to before, during, after and puts an ordering


## For dplyr version less than 1.1.0
# shoesLong$time <- dplyr::recode_factor(
#   shoesLong$time,
#   "t1" = "before",
#   "t2" = "during",
#   "t3" = "after",
#   .ordered = TRUE
# )
```

4.2.3 Fit the Models

We will fit two models: one that we'll use for the omnibus test and one we'll use for sphericity checking.

Let's begin with our omnibus testing model. Unfortunately, there is not a quick, easy way to get a well organized table. We are going to have to do some manipulations to get that table as we'll see in the results section.

The key here is that you will want to make sure that you listen/watch for a particular warning message: `Error() model is singular`. This is because our final interaction term (subject x time) uses up all remaining *degrees of freedom* so we will not have a traditional Residuals/Error term. That is the crux of this particular message.

```
# Demo Code for Omnibus Model ----
shoesModel <- aov(
  formula = sales ~ campaign * time + Error(market %in% campaign),
  data = shoesLong
)
```

```
## Warning in aov(formula = sales ~ campaign * time + Error(market %in% campaign),
## : Error() model is singular
```

While we want to see this error message, you don't want the message to show up in your final report.

Due to the lack of a typical Residuals/Error term, we will need to use an alternative route for getting things like residuals or fitted values for our assumption checking. For our second model, we will turn to the `nlme` package.

```
# Demo code for nlme package ----
## Use the nlme package to fit a model that we can use for assumption checking
shoesAssumptions <- nlme::lme(
  data = shoesLong,
  fixed = sales ~ campaign * time,
  random = ~ 1|market
)
```

4.3 Assessing the Assumptions

The methods here are the same as for the Within Subjects Repeated Measures design. Remember, we will use the `shoesAssumptions` model object from the `nlme` package.

4.3.1 Gaussian Residuals

Use a QQ plot:

```
# Demo Code for QQ plot for residuals ----
car::qqPlot(
  x = residuals(shoesAssumptions), # Notice which model is getting used
```

```
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
  ylab = "Residuals (sales)"
)
```

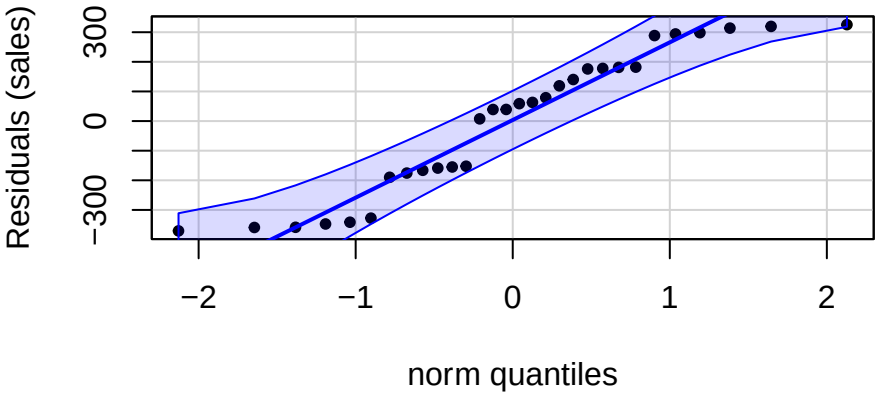

Figure 11: QQ Plot of Shoe Sales Study

4.3.2 Gaussian Treatment Effects

Again, use a QQ plot:

```
# Demo Code for Market Effects ----
car::qqPlot(
  x = unlist(
    lme4::ranef( # Notice the use of the lme4 package
      object = shoesAssumptions, # Notice which model is getting used
      whichel = c("market")
    )
  ),
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
  ylab = "Market"
)
```

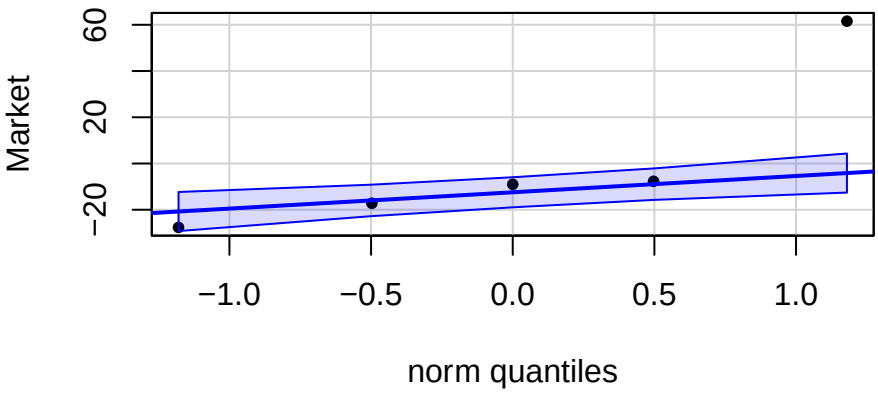

Figure 12: QQ Plot for Market Effects

4.3.3 Homoscedasticity

Use a Tukey-Anscombe Plot:

```r
# Demo Code for Tukey-Anscombe Plot ----
ggplot(
  data = data.frame(
    residuals = residuals(shoesAssumptions), # Notice which model gets used
    fitted = fitted.values(shoesAssumptions)
  ),
  mapping = aes(x = fitted, y = residuals)
) +
  geom_point(size = 2) +
  geom_hline(
    yintercept = 0,
    linetype = "dashed",
    color = "grey50"
  ) +
  geom_smooth(
    formula = y ~ x,
    method = stats::loess,
    method.args = list(degree = 1),
    se = FALSE,
    linewidth = 0.5
  ) +
  theme_bw() +
  xlab("Fitted values (sales)") +
  ylab("Residuals (sales)")
```

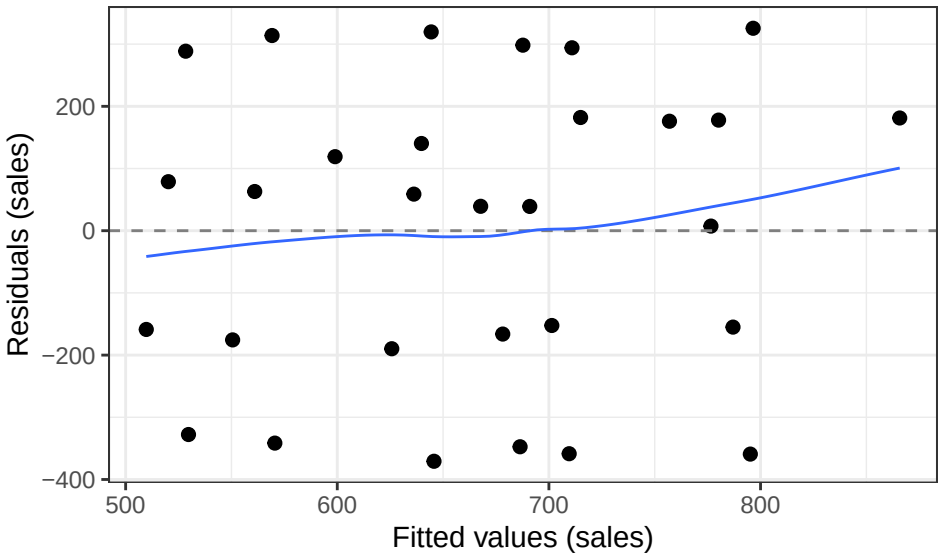

Figure 13: Tukey-Anscombe Plot for Shoe Sales Study

4.3.4 Independence of Subjects

One of the interesting things about Nested Repeated Measures is that we inherently know at least *some* of the measurement order. We might not know which store got measured before which other store, but we know the sequence of measurements for each store. Thus, we can make the following plot:

```r
# Demo Code Independence of Subjects ----
ggplot(
  data = shoesLong,
  mapping = aes(
    x = time,
    y = sales,
    color = paste(campaign, market, sep = ":"),
    group = paste(campaign, market, sep = ":")
  )
```

```
) +
  geom_point() +
  geom_line() +
  theme_bw() +
  xlab("Time Point (relative to campaign)") +
  ylab("Sales (coded)") +
  labs(
    color = "Campagin:Market"
  )
```

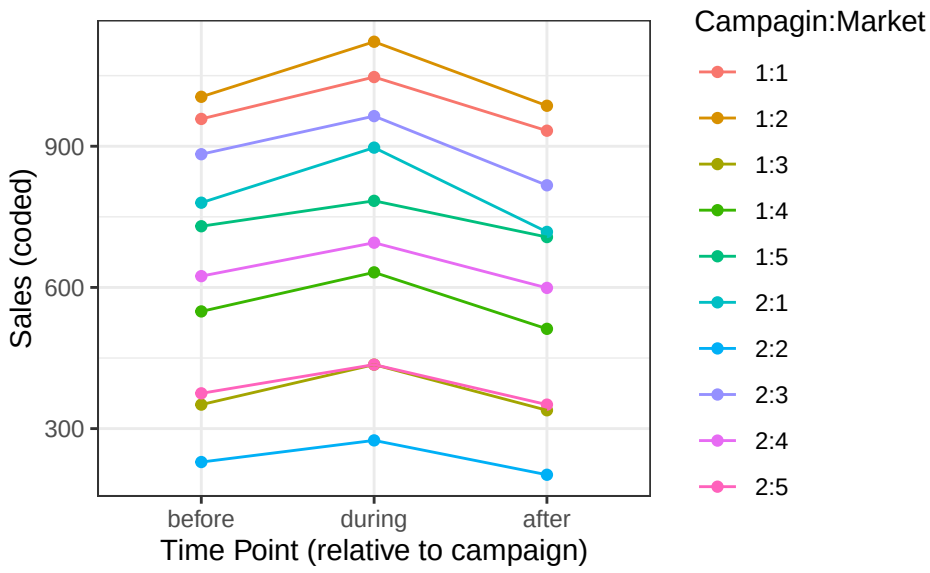

Figure 14: Shoe Sales by Time and Store

Now, this plot (Figure 14) is great for letting us compare the effects over time. This is not necessarily the greatest for letting us see if our subjects are independent of one another. The consistency of the effects over time do indicate that the inherent dependency of each subject's measurements *is* consistent across time. If a particular market had a huge increase in sales after the campaign, that would suggest something strange happened.

For Nested Repeated Measures we will ultimately want to fall back on our study design to make a solid justification. This is why I've been pressing you all course long on including details/being specific.

4.3.5 Interaction of Subject and Factor

A similar plot to the previous Time Series plot is to construct a plot known as "Growth Curves". In essence, we want to see how the response changes for each subject over time... but we're going to separate the data a bit more cleanly so we can see if and how the factor might interact with our subjects.

```
# Demo Code for Growth Curves ----
ggplot(
  data = shoesLong,
  mapping = aes(
    x = time,
    y = sales,
    color = market,
    group = market
    )
) +
  geom_point(size = 2) +
  geom_line() +
  facet_wrap(facets = ~campaign) +
  ggplot2::theme_bw() +
  xlab("Time Period") +
  ylab("Sales (coded)") +
```

```
  labs(
    color = "Market"
  )
```

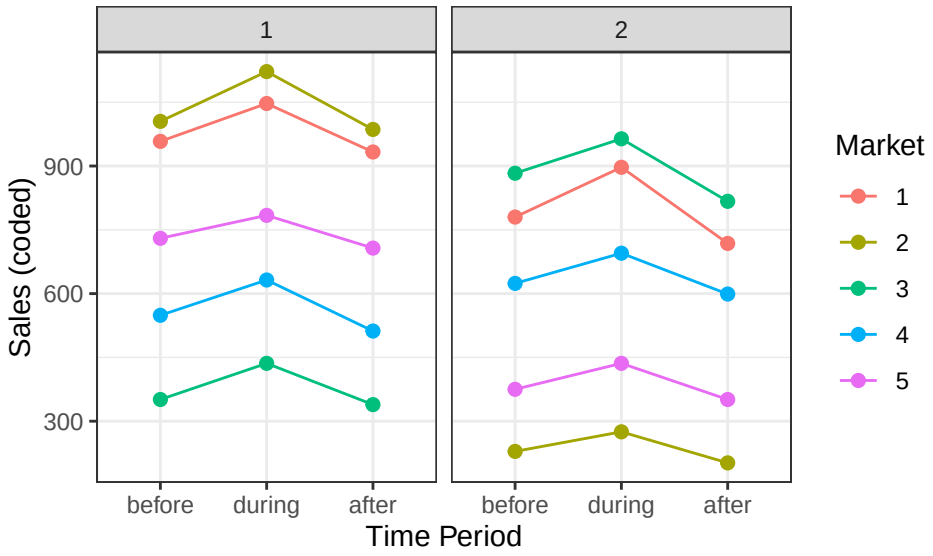

Figure 15: Growth Curves for Shoe Sales Study

Notice that we used the `facet_wrap` on campaign to split the time series plot into separate panels/facets for each campaign. If we see the same behaviors in both facets, then there is no worrisome interaction between subjects and our factor (Figure 15).

Keep in mind that Market 3 in Campaign 1 is ***not*** the same market as Market 3 in Campaign 2. They just happen to be the *third* market inside each campaign.

4.3.6 Sphericity

The idea behind sphericity is that we have essentially the same levels of variation for the *differences between treatments*. While there is a visual method we can use here, we do need to do so with some caution. Much like looking for homoscedasticity, we'll want to see if any difference has *excessively different* variation than another difference. Unlike homoscedasticity **there is no rule of thumb/guideline** (e.g., more than twice) for sphericity.

```
# Demo Code for Sphericity Plot ----
sphericityPlot(
  dataWide = shoesWide,
  subjectID = c("market", "campaign"),
  colsIgnore = NULL
)
```

The `nlme` package does not support Mauchly's Test for Sphericity. Thus, we will have to rely on the plot in Figure 16.

4.4 Results

As we have been doing all semester, we can divide our results into an Omnibus portion, a Point Estimates portion, and Post Hoc portion.

4.4.1 Omnibus Test

As mentioned, getting a nice looking table is not as straightforward with the Nested Repeated Measures ANOVA problems.

```
# Demo Code for Omnibus ANOVA Table ----

## We have to custom build the ANOVA table
shoesTemp <- summary(shoesModel)
shoesOmni <- rbind(
  shoesTemp$`Error: market:campaign`[[1]],
  shoesTemp$`Error: Within`[[1]]
```

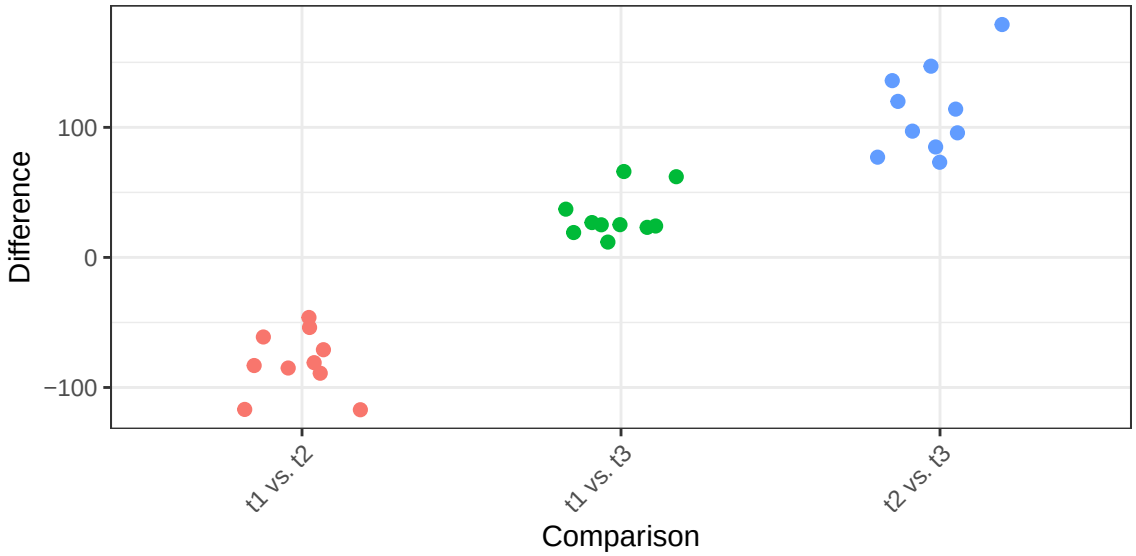

Figure 16: Sphericity Plot for Shoe Sales Study

```
)

row.names(shoesOmni) <- c("campaign", "market", "time","campaign:time", "market:time")

shoesOmni["market", "F value"] <- shoesOmni["market", "Mean Sq"] /
  shoesOmni["market:time", "Mean Sq"]
shoesOmni["market", "Pr(>F)"] <- pf(
  q = shoesOmni["market", "F value"],
  df1 = shoesOmni["market", "Df"],
  df2 = shoesOmni["market:time", "Df"],
  lower.tail = FALSE
)

shoesOmni %>%
  tibble::rownames_to_column(
    var = "Source"
  ) %>%
  dplyr::mutate(
    `Pr(>F)` = ifelse(
      test = is.na(`Pr(>F)`),
      yes = NA,
      no = pvalRound(`Pr(>F)`)
    )
  ) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Source", "df", "SS", "MS", "F", "p-value"),
    caption = "ANOVA Table for Athletic Shoes Study",
    align = c('l',rep('c',5)),
    booktab = TRUE,
    format.args = list(big.mark = ",")
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )
```

Table 7: ANOVA Table for Athletic Shoes Study

Source	df	SS	MS	F	p-value
campaign	1	168,150.5333	168,150.5333	0.7336	0.4166
market	8	1,833,680.9333	229,210.1167	640.3113	< 0.0001
time	2	67,073.0667	33,536.5333	93.6862	< 0.0001
campaign:time	2	391.4667	195.7333	0.5468	0.5892
market:time	16	5,727.4667	357.9667		

We will not worry about effect sizes here.

4.4.2 Point Estimates

For point estimates, you might want to look at both Campaign and Time Effects:

```
# Demo Code for Using emmeans ----
shoesCampaignPH <- emmeans::emmeans(
  object = shoesModel,
  specs = pairwise ~ campaign,
  adjust = "tukey",
  level = 0.99
)

shoesTimePH <- emmeans::emmeans(
  object = shoesModel,
  specs = pairwise ~ time,
  adjust = "tukey",
  level = 0.99
)
```

You can get point estimates for both effects:

```
# Demo Code Point Estimates ----
## Campaign Effects
as.data.frame(shoesCampaignPH$emmeans) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Campaign", "Marginal Mean","SE", "DF",
                  "Lower Bound","Upper Bound"),
    caption = "Marginal Means-Tukey 99\\% Adjustment",
    align = c("l", rep("c", 5)),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )
```

Table 8: Marginal Means-Tukey 99% Adjustment

Campaign	Marginal Mean	SE	DF	Lower Bound	Upper Bound
1	739.4000	123.615	8	324.6237	1154.176
2	589.6667	123.615	8	174.8904	1004.443

```
# Demo Code Point Estimates ----
## Time Point Effects
as.data.frame(shoesTimePH$emmeans) %>%
```

```
  knitr::kable(
    digits = 4,
    col.names = c("Time Point", "Marginal Mean","SE", "DF",
                  "Lower Bound","Upper Bound"),
    caption = "Marginal Means-Tukey 99\\% Adjustment",
    align = c("l", rep("c", 5)),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )
```

Table 9: Marginal Means-Tukey 99% Adjustment

Time Point	Marginal Mean	SE	DF	Lower Bound	Upper Bound
before	648.4	87.5454	8.05	355.1807	941.6193
during	728.8	87.5454	8.05	435.5807	1022.0193
after	616.4	87.5454	8.05	323.1807	909.6193

4.4.3 Post Hoc-Pairwise Comparisons

You can also do the standard pairwise comparisons:

```
# Demo Code Point Estimates ----
## Pairwise on Campaign
as.data.frame(shoesCampaignPH$contrasts) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Comparison", "Difference","SE", "DF",
                  "t Statistic","p-value"),
    caption = "Campaign Comparison-Tukey 99\\% Adjustment",
    align = c("l", rep("c", 5)),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )
```

Table 10: Campaign Comparison-Tukey 99% Adjustment

Comparison	Difference	SE	DF	t Statistic	p-value
campaign1 - campaign2	149.7333	174.818	8	0.8565	0.4166

```
# Demo Code Point Estimates ----
## Pairwise on Time Point
as.data.frame(shoesTimePH$contrasts) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Comparison", "Difference","SE", "DF",
                  "t Statistic","p-value"),
    caption = "Time Point-Tukey 99\\% Adjustment",
    align = c("l", rep("c", 5)),
    booktabs = TRUE
```

```
) %>%
kableExtra::kable_styling(
  bootstrap_options = c("striped", "condensed"),
  font_size = 12,
  latex_options = c("HOLD_position")
)
```

Table 11: Time Point-Tukey 99% Adjustment

Comparison	Difference	SE	DF	t Statistic	p-value
before - during	-80.4	8.4613	16	-9.5021	0.0000
before - after	32.0	8.4613	16	3.7819	0.0044
during - after	112.4	8.4613	16	13.2840	0.0000

4.4.4 Effect Sizes

We will not worry about effect sizes here.

5 Code Appendix

```
# Setting Document Options ----
knitr::opts_chunk$set(
  echo = FALSE,
  warning = FALSE,
  message = FALSE,
  fig.align = "center"
)

packages <- c("tidyverse", "knitr", "kableExtra",
              "parameters", "hasseDiagram", "car",
              "psych", "lme4", "nlme", "emmeans",
              "rstatix")
lapply(packages, library, character.only = TRUE)

options(knitr.kable.NA = "")
options(contrasts = c("contr.sum", "contr.poly"))

source("https://raw.github.com/neilhatfield/STAT461/master/rScripts/ANOVATools.R")

# Generic Demo Code for creating a wide data frame ----
# Note: this code assumes you already read in a long format data frame

#IMPORTANT: you will need to the code to match the names in your actual situation

dataWide <- pivot_wider(
  data = dataLong, # "dataLong" is the name of the long format data frame
  names_from = group, #"group" is the name of the column that contains your treatments
  values_from = response #"response" is the name of the column with the response values
)

# Generic Demo Code for creating a long data frame ----
# Note: this code assumes you already read in a wide format data frame

#IMPORTANT: you will need to the code to match the names in your actual situation

dataLong <- pivot_longer(
  data = dataWide, # "dataWide" is the name of the wide format data frame
  cols = !subject, # Says to not use the "subject" column
  names_to = "group", # This is the new column you want the treatments to go to
  names_transform = list(group = as.factor), # Makes the treatment column a factor
  values_to = "response" # A new column that will contain all of the response values
)

# Demo code to set up R ----
## Load packages
packages <- c("tidyverse", "knitr", "kableExtra",
              "parameters", "hasseDiagram", "car",
              "psych", "emmeans", "rstatix", "lme4",
              "nlme")
lapply(packages, library, character.only = TRUE, quitely = TRUE)

options(knitr.kable.NA = "")
options(contrasts = c("contr.sum", "contr.poly"))

source("https://raw.github.com/neilhatfield/STAT461/master/rScripts/ANOVATools.R")

# Demo Code for Hasse Diagram for Beer Judging ----
modelLabels <- c("1 Judge Beer 1", "6 (Judge) 5", "4 Beer 3", "24 (Observation) 15")
```

```r
modelMatrix <- matrix(
  data = c(FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE,
           FALSE, TRUE, TRUE, TRUE, FALSE),
  nrow = 4,
  ncol = 4,
  byrow = FALSE
)
hasseDiagram::hasse(
 data = modelMatrix,
 labels = modelLabels
)

# Demo Code for loading in Beer Data ----
## The data are stored in a wide format
beerWide <- read.table(
  file = "https://raw.github.com/neilhatfield/STAT461/master/dataFiles/beerJudging.csv",
  header = TRUE,
  sep = ","
)

## The order column contains the first letter of each
## beer in the order the judge tasted them.

## Convert to a long format AND unpack the order column

beerLong <- pivot_longer(
  data = beerWide,
  cols = where(is.numeric),
  names_to = "beer",
  names_transform = list(beer = as.factor),
  values_to = "score",
) %>%
  rowwise() %>% # rowwise tells R to make changes only row by row
  mutate(
    order = str_locate(
      string = order,
      pattern = str_sub(string = beer, start = 1, end = 1)
    )[1] #[1] Gives us the correct index
  ) %>%
  mutate( # Optional: Get rid of periods in beer names
    beer = str_replace_all(string = beer, pattern = "\\.", replacement = " ")
  ) # Note: this will undo the factor type of beer

beerLong$beer <- as.factor(beerLong$beer)
beerLong$judge <- as.factor(beerLong$judge)

# Demo Code for Within Subject Repeated Measures ANOVA ----
## Omnibus Model (for our ANOVA table)
beerOmni <- aov(
  formula = score ~ judge + beer,
  data = beerLong
)

## Random Effect Model (Assumption Checking and Point Estimation)
beerMixed <- lme4::lmer(
  formula = score ~ (1|judge) + beer,
  data = beerLong
)
```

```r
## Compound Symmetry/Sphericity Assessment
beerSphere <- rstatix::anova_test(
  data = beerLong,
  formula = score ~ beer + Error(judge %in% beer)
)
## The %in% tells R that judge should be treated as nested in beer

# Demo Code for QQ plot for residuals ----
car::qqPlot(
  x = residuals(beerMixed), # Notice which model gets used here
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
  ylab = "Residuals (score)"
)

# Demo Code QQ Plot of Judge Effects ----
car::qqPlot(
  x = unlist(
    lme4::ranef(
      object = beerMixed, ## Notice which model is getting used
      whichel = c("judge")
    )
  ),
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
  ylab = "Judge Effects"
)

# Demo Code for the Tukey-Anscombe Plot ----
ggplot(
  data = data.frame(
    residuals = residuals(beerMixed), # Notice which model
    fitted = fitted.values(beerMixed)
  ),
  mapping = aes(x = fitted, y = residuals)
) +
  geom_point(size = 2) +
  geom_hline(
    yintercept = 0,
    linetype = "dashed",
    color = "grey50"
  ) +
  geom_smooth(
    formula = y ~ x,
    method = stats::loess,
    method.args = list(degree = 1),
    se = FALSE,
    linewidth = 0.5
  ) +
  theme_bw() +
  xlab("Fitted values (score)") +
  ylab("Residuals (score)")

# Demo Code for the the Residual Dot Plot ----
ggplot(
```

```
  data = data.frame(
    beer = beerLong$beer,
    residuals = residuals(beerMixed)
  ),
  mapping = aes(x = residuals)
) +
  geom_dotplot(
    method = "histodot",
    binwidth = 0.1,
    right = FALSE,
    origin = 0,
    dotsize = 2,
    stackratio = 1.1,
    binpositions = "all"
  ) +
  xlab("Residual (score)") +
  theme_bw() +
  facet_wrap(
    facets = vars(beer),
    ncol = 1,
    strip.position = "right",
    labeller = label_wrap_gen(width = 13)
  ) +
  theme(
    text = element_text(size = 12),
    axis.title.y = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks.y = element_blank(),
    strip.background = element_rect(
      linewidth = 25,
      color = "white",
      fill = "black")
  )

# Demo Code for an Interaction Plot ----
ggplot(
  data = beerLong,
  mapping = aes(
    x = beer,
    y = score,
    color = judge,
    shape = judge,
    group = judge
    )
) +
  geom_point(size = 2) +
  geom_line() +
  ggplot2::theme_bw() +
  xlab("Beer") +
  ylab("Score") +
  labs(
    color = "Judge",
    shape = "Judge"
  ) +
  scale_color_manual(values = boastUtils::boastPalette) +
  scale_x_discrete(
    labels = label_wrap_gen(width = 12)
  )
```

```r
# Demo Code for Heat Map ----
ggplot(
  data = beerLong,
  mapping = aes(x = judge, y = beer, weight = score)
) +
  geom_bin_2d() +
  theme_bw() +
  xlab("Judge") +
  ylab("Beer") +
  scale_y_discrete(
    labels = label_wrap_gen(width = 10)
  ) +
  scale_fill_gradient2(name = "Score")
# Demo Code for Sphericity Plot ----
# Note: to use color = "boast" or "psu" you'll need to install the boastUtils package
# devtools::install_github("EducationShinyAppTeam/boastUtils")
sphericityPlot(
  dataWide = beerWide, # Data needs to be in wide format
  subjectID = "judge", # character name of the subject column
  colsIgnore = c("order"),
  colors = "boast" # Optional; "default", "boast", "psu"
)


# Demo Code for Mauchly's Test ----
beerSphere$`Mauchly's Test for Sphericity` %>%
  dplyr::select(Effect, W, p) %>%
  knitr::kable(
    digits = 4,
  col.names = c("Effect", "Mauchly's W", "p"),
  caption = "Mauchly's Sphericity Test",
  align = c('l',"c","c"),
  booktab = TRUE
) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )

# Demo Code for Residual Sequence Plot ----
cbind(
  beerLong,
  residuals = residuals(beerMixed) ## Notice which model is getting used
) %>%
  ggplot(
    mapping = aes(x = order, y = residuals)
  ) +
  geom_point() +
  geom_line() +
  geom_hline(
    yintercept = 0,
    linetype = "dashed",
    color = "grey50"
  ) +
  theme_bw() +
  xlab("Tasting Order") +
  ylab("Residuals (score)") +
  facet_wrap(facets = vars(judge))
```

```r
# Demo code of omnibus test ----
parameters::model_parameters(
  model = beerOmni, # Notice which model we're using here
  effectsize_type = c("eta", "omega", "epsilon")
) %>%
  dplyr::mutate(
    p = ifelse(
      test = is.na(p),
      yes = NA,
      no = pvalRound(p)
    )
  ) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Source", "SS", "df", "MS", "F", "p-value",
                  "Partial Eta Sq.", "Partial Omega Sq.", "Partial Epsilon Sq."),
    caption = "ANOVA Table for Beer Judging Study",
    align = c('l',rep('c',8)),
    booktab = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("scale_down", "HOLD_position")
  )

# Use the block relative efficiency function ----
block.RelEff(
  aov.obj = beerOmni,
  blockName = "judge",
  trtName = "beer"
)

# Demo Code for Corrected Omnibus p-values ----
correctedTable <- beerSphere$`Sphericity Corrections` %>%
  dplyr::select(GGe, `p[GG]`, HFe, `p[HF]`)
correctedTable$`p[GG]` <- lapply(
  X = correctedTable$`p[GG]`,
  FUN = pvalRound
)
correctedTable$`p[HF]` <- lapply(
  X = correctedTable$`p[HF]`,
  FUN = pvalRound
)
knitr::kable(
  x = correctedTable,
  digits = 4,
  col.names = c("Greenhouse-Geisser", "p-value", "Huynh-Feldt", "p-value"),
  caption = "Sphericity Corrections",
  align = "c",
  booktab = TRUE
) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )

# Demo Code for Point Estimates using emmeans package ----
```

```r
beerPH <- emmeans::emmeans(
  object = beerMixed, # Notice the use of the mixed model here
  specs = pairwise ~ beer,
  adjust = "tukey",
  level = 0.92
)

## Point Estimates
as.data.frame(beerPH$emmeans) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Type of Beer", "Marginal Mean","SE", "DF",
                  "Lower Bound","Upper Bound"),
    caption = "Marginal Means-Tukey 92\\% Adjustment",
    align = c("l", rep("c", 5)),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )

# Demo Code for Post Hoc Pairwise Comparisons ----
## Notice we're using the beerPH object from the point estimates code
as.data.frame(beerPH$contrasts) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Comparison", "Difference","SE", "DF",
                  "t Statistic","p-value"),
    caption = "Marginal Means-Tukey 92\\% Adjustment",
    align = c("l", rep("c", 5)),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )

# Demo Code for Post Hoc Effect Sizes ----
tempEMM <- emmeans::emmeans(
  object = beerMixed, # Notice we're using the beerMixed model
  specs = "beer"
)

# Pass the stored marginals into the effect size function
cohenTemp <- emmeans::eff_size(
  object = tempEMM,
  sigma = sigma(beerMixed),
  edf = df.residual(beerMixed)
)

# Create a data frame, add on the probability of superiority
# Send that data frame into a nice table
as.data.frame(cohenTemp) %>%
  dplyr::mutate(
    ps = probSup(effect.size),
    .after = effect.size
```

```
  ) %>%
  dplyr::select(contrast, effect.size, ps) %>%
  knitr::kable(
    digits = 3,
    col.names = c("Comparison", "Cohen's d", "Probability of Superiority"),
    align = "lcc",
    caption = "Effect Sizes for Beer",
    booktab = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = "HOLD_position"
  )

# Demo Code for Hasse Diagram for Shoe Advertisement Study ----
modelLabels <- c("1 Sell Shoes 1", "2 Campaign 1", "10 (Market) 8", "3 Time Point 2",
                 "6 Campaign × Time Point 2", "30 (Markets X Time) 16")
modelMatrix <- matrix(
  data = c(FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE,
           FALSE, TRUE, TRUE, FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE,
           FALSE, FALSE, TRUE, TRUE, FALSE, TRUE, FALSE, FALSE, TRUE, TRUE, TRUE,
           TRUE, TRUE, FALSE),
  nrow = 6,
  ncol = 6,
  byrow = FALSE
)
hasseDiagram::hasse(
 data = modelMatrix,
 labels = modelLabels
)

# Demo Code for reading in data ----
shoesWide <- read.table(
  file = "https://raw.github.com/neilhatfield/STAT461/master/dataFiles/shoes.csv",
  header = TRUE,
  sep = ","
)

shoesWide$campaign <- as.factor(shoesWide$campaign)
shoesWide$market <- as.factor(shoesWide$market)

# Make a long version of the data
shoesLong <- tidyr::pivot_longer(
  data = shoesWide,
  cols = dplyr::starts_with("t"),
  names_to = "time",
  names_ptypes = list("time" = factor()),
  values_to = "sales"
) %>%
  mutate(
    time = case_match(
      .x = time,
      "t1" ~ "before",
      "t2" ~ "during",
      "t3" ~ "after",
      .ptype = factor(levels = c("before", "during", "after"))
    )
  )
```

```r
# Recode times to before, during, after and puts an ordering


## For dplyr version less than 1.1.0
# shoesLong$time <- dplyr::recode_factor(
#   shoesLong$time,
#   "t1" = "before",
#   "t2" = "during",
#   "t3" = "after",
#   .ordered = TRUE
# )

# Demo Code for Omnibus Model ----
shoesModel <- aov(
  formula = sales ~ campaign * time + Error(market %in% campaign),
  data = shoesLong
)

# Demo code for nlme package ----
## Use the nlme package to fit a model that we can use for assumption checking
shoesAssumptions <- nlme::lme(
  data = shoesLong,
  fixed = sales ~ campaign * time,
  random = ~ 1|market
)

# Demo Code for QQ plot for residuals ----
car::qqPlot(
  x = residuals(shoesAssumptions), # Notice which model is getting used
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
  ylab = "Residuals (sales)"
)

# Demo Code for Market Effects ----
car::qqPlot(
  x = unlist(
    lme4::ranef( # Notice the use of the lme4 package
      object = shoesAssumptions, # Notice which model is getting used
      whichel = c("market")
    )
  ),
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
  ylab = "Market"
)

# Demo Code for Tukey-Anscombe Plot ----
ggplot(
  data = data.frame(
    residuals = residuals(shoesAssumptions), # Notice which model gets used
    fitted = fitted.values(shoesAssumptions)
  ),
  mapping = aes(x = fitted, y = residuals)
) +
```

```
  geom_point(size = 2) +
  geom_hline(
    yintercept = 0,
    linetype = "dashed",
    color = "grey50"
  ) +
  geom_smooth(
    formula = y ~ x,
    method = stats::loess,
    method.args = list(degree = 1),
    se = FALSE,
    linewidth = 0.5
  ) +
  theme_bw() +
  xlab("Fitted values (sales)") +
  ylab("Residuals (sales)")

# Demo Code Independence of Subjects ----
ggplot(
  data = shoesLong,
  mapping = aes(
    x = time,
    y = sales,
    color = paste(campaign, market, sep = ":"),
    group = paste(campaign, market, sep = ":")
  )
) +
  geom_point() +
  geom_line() +
  theme_bw() +
  xlab("Time Point (relative to campaign)") +
  ylab("Sales (coded)") +
  labs(
    color = "Campagin:Market"
  )

# Demo Code for Growth Curves ----
ggplot(
  data = shoesLong,
  mapping = aes(
    x = time,
    y = sales,
    color = market,
    group = market
    )
) +
  geom_point(size = 2) +
  geom_line() +
  facet_wrap(facets = ~campaign) +
  ggplot2::theme_bw() +
  xlab("Time Period") +
  ylab("Sales (coded)") +
  labs(
    color = "Market"
  )

# Demo Code for Sphericity Plot ----
sphericityPlot(
  dataWide = shoesWide,
```

```r
  subjectID = c("market", "campaign"),
  colsIgnore = NULL
)

# Demo Code for Omnibus ANOVA Table ----

## We have to custom build the ANOVA table
shoesTemp <- summary(shoesModel)
shoesOmni <- rbind(
  shoesTemp$`Error: market:campaign`[[1]],
  shoesTemp$`Error: Within`[[1]]
)

row.names(shoesOmni) <- c("campaign", "market", "time","campaign:time", "market:time")

shoesOmni["market", "F value"] <- shoesOmni["market", "Mean Sq"] /
  shoesOmni["market:time", "Mean Sq"]
shoesOmni["market", "Pr(>F)"] <- pf(
  q = shoesOmni["market", "F value"],
  df1 = shoesOmni["market", "Df"],
  df2 = shoesOmni["market:time", "Df"],
  lower.tail = FALSE
)

shoesOmni %>%
  tibble::rownames_to_column(
    var = "Source"
  ) %>%
  dplyr::mutate(
    `Pr(>F)` = ifelse(
      test = is.na(`Pr(>F)`),
      yes = NA,
      no = pvalRound(`Pr(>F)`)
    )
  ) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Source", "df", "SS", "MS", "F", "p-value"),
    caption = "ANOVA Table for Athletic Shoes Study",
    align = c('l',rep('c',5)),
    booktab = TRUE,
    format.args = list(big.mark = ",")
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )

# Demo Code for Using emmeans ----
shoesCampaignPH <- emmeans::emmeans(
  object = shoesModel,
  specs = pairwise ~ campaign,
  adjust = "tukey",
  level = 0.99
)

shoesTimePH <- emmeans::emmeans(
  object = shoesModel,
```

```
  specs = pairwise ~ time,
  adjust = "tukey",
  level = 0.99
)

# Demo Code Point Estimates ----
## Campaign Effects
as.data.frame(shoesCampaignPH$emmeans) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Campaign", "Marginal Mean","SE", "DF",
                  "Lower Bound","Upper Bound"),
    caption = "Marginal Means-Tukey 99\\% Adjustment",
    align = c("l", rep("c", 5)),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )

# Demo Code Point Estimates ----
## Time Point Effects
as.data.frame(shoesTimePH$emmeans) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Time Point", "Marginal Mean","SE", "DF",
                  "Lower Bound","Upper Bound"),
    caption = "Marginal Means-Tukey 99\\% Adjustment",
    align = c("l", rep("c", 5)),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )

# Demo Code Point Estimates ----
## Pairwise on Campaign
as.data.frame(shoesCampaignPH$contrasts) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Comparison", "Difference","SE", "DF",
                  "t Statistic","p-value"),
    caption = "Campaign Comparison-Tukey 99\\% Adjustment",
    align = c("l", rep("c", 5)),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("HOLD_position")
  )

# Demo Code Point Estimates ----
## Pairwise on Time Point
as.data.frame(shoesTimePH$contrasts) %>%
```

```
knitr::kable(
  digits = 4,
  col.names = c("Comparison", "Difference","SE", "DF",
                "t Statistic","p-value"),
  caption = "Time Point-Tukey 99\\% Adjustment",
  align = c("l", rep("c", 5)),
  booktabs = TRUE
) %>%
kableExtra::kable_styling(
  bootstrap_options = c("striped", "condensed"),
  font_size = 12,
  latex_options = c("HOLD_position")
)
```