Factorial Models

Neil J. Hatfield

4/9/2021

In this tutorial, we are going to explore Factorial Treatment Structures/Factorial Designs in R.

New Package

For Factorial Designs, we will need to add the `emmeans` package to our list of packages to load with the `library` call. Later on in this document (mixed effects), we will also look at the `lmerTest` package.

Two-way Fixed ANOVA Context

An engineer is designing a battery for use in a device that will people will use in some extreme temperatures. Unfortunately, the engineer may only alter one design parameter: the plate material for the battery of which he has three choices.

The device his batteries are for gets manufactured separately and is then shipped to the field, where the engineer has no control over the temperature the device will encounter. His experiences lead him to believe that environmental temperature will affect the battery life. He can control the temperature in the lab for product development testing.

He decides to test all three plate materials at three temperature levels—15ºF, 70ºF, and 125ºF—as these temperatures are consistent with reported end-use environments.

His questions:

1) What effects do material type and temperature have on life of battery?
2) Is there a choice of material that would give uniformly long life regardless of temperature?

Examine the Hasse Diagram

Remember to look at a Hasse diagram and to make use of the diagram to justify why a factorial design is appropriate.

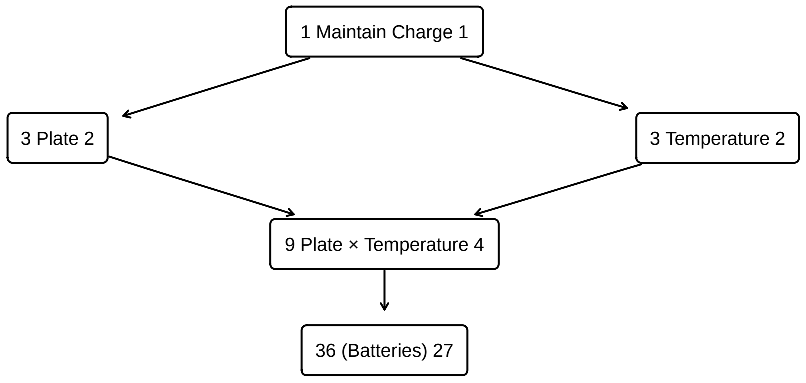

Figure 1: Hasse Diagram for Battery Design Study

Data

For this example, you'll want to import the data as shown below. You'll notice that I'm using the `recode_factor` function from the `dplyr` package to translate the integers for both temperature and plate into more meaningful values (plus this tells R to treat those as factors).

```r
# Load battery data
battery <- read.table(
  file = "https://raw.github.com/neilhatfield/STAT461/master/dataFiles/batteryLife.dat",
  header = TRUE,
  sep = ","
)

battery$temperature <- dplyr::recode_factor(
  battery$temperature,
  `15` = "15ºF",
  `70` = "70ºF",
  `125` = "125ºF"
)
battery$material <- dplyr::recode_factor(
  battery$material,
  `1` = "Plate 1",
  `2` = "Plate 2",
  `3` = "Plate 3"
)
```

Explore the data

Exploring the data in factorial settings becomes much more important as now you have many more ways to think about slicing up the data resulting in more ways to help people (and yourself) think about the data. Remember, data visualizations are some of your strongest and most helpful tools here.

You can use the multiple factors in a variety of ways in your data visualizations. For example, rather than looking at a side-by-side box plots along one factor, you could do a set for each factor or by the interaction. R's base `boxplot` function allows for you explore interactions by using the formula argument.

```r
# Boxplot Example with interaction of factors
boxplot(
  formula = life ~ temperature:material,
  data = battery,
  ylab = "Life (hrs)",
  xlab = "Temp (ºF) x Material"
)
```

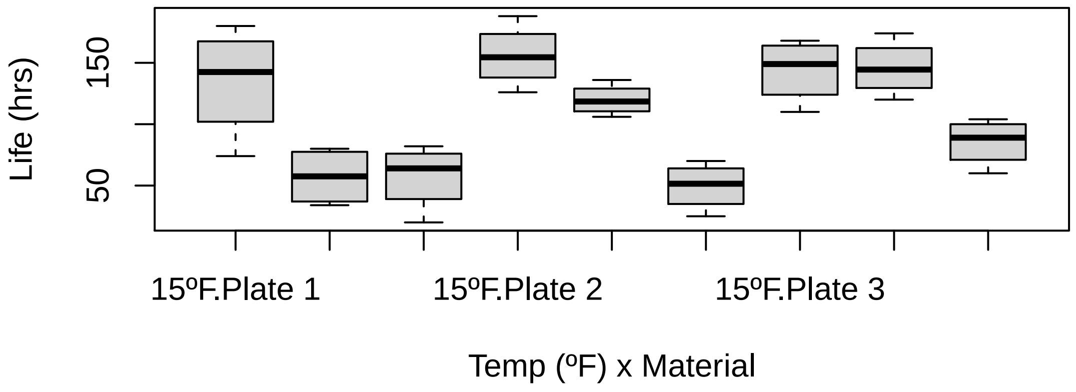

Figure 2: Box Plot of Batter Life Spans by Temperature and Plate Material

While this box plot is okay to look at, we could improve this plot greatly for professional work. The easiest method would be to use `ggplot2`.

```
## Ggplot box plot with interaction of factors
ggplot(
  data = battery,
  mapping = aes(
    x = temperature,
    y = life,
    fill = material
  )
) +
  geom_boxplot() +
  theme_bw() +
  xlab("Operating Temperature") +
  ylab("Life span (hours)") +
  labs(
    fill = "Material"
  ) +
  theme(
    legend.position = "top"
  )
```

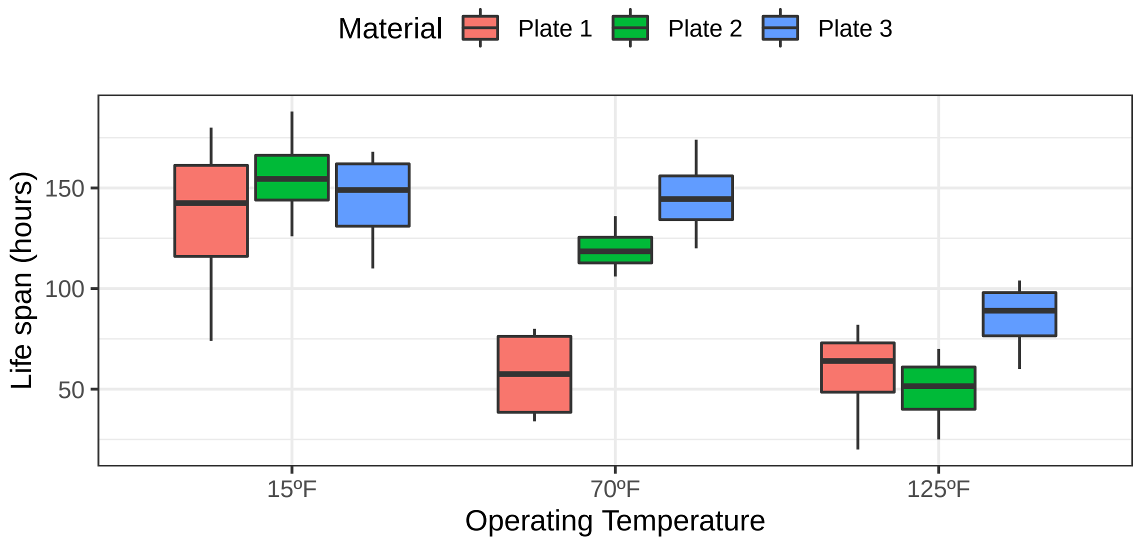

Figure 3: Box Plot With Multiple Factors

Descriptive Statistics

In addition to data visualizations, we also may make use of descriptive/incisive statistics. We've used the `describeBy` from the `psych` package in the past to break our response up into groups based upon our factor. We can do something similar in multi-factor situations as shown here:

```r
# Descriptive statistics by interactions of factors
batteryStats <- psych::describeBy(
  x = battery$life,
  group = paste(battery$temperature, battery$material, sep = " x "),
  na.rm = TRUE,
  skew = TRUE,
  ranges = TRUE,
  quant = c(0.25, 0.75),
  IQR = TRUE,
  mat = TRUE,
  digits = 4
)

batteryStats %>%
  tibble::remove_rownames() %>%
  tibble::column_to_rownames(
    var = "group1"
  ) %>%
  dplyr::select(
    n, min, Q0.25, median, Q0.75, max, mad, mean, sd, skew, kurtosis
  ) %>%
  knitr::kable(
    caption = "Summary Statistics for Battery Life Spans",
    digits = 3,
    format.args = list(big.mark = ","),
    align = rep('c', 11),
    col.names = c("n", "Min", "Q1", "Median", "Q3", "Max", "MAD", "SAM", "SASD",
                  "Sample Skew","Sample Ex. Kurtosis"),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    font_size = 12,
    latex_options = c("HOLD_position", "scale_down")
  )
```

Table 1: Summary Statistics for Battery Life Spans

	n	Min	Q1	Median	Q3	Max	MAD	SAM	SASD	Sample Skew	Sample Ex. Kurtosis
125ºF x Plate 1	4	20	48.50	64.0	73.00	82	17.791	57.50	26.851	-0.466	-1.864
125ºF x Plate 2	4	25	40.00	51.5	61.00	70	18.532	49.50	19.261	-0.195	-2.015
125ºF x Plate 3	4	60	76.50	89.0	98.00	104	16.309	85.50	19.279	-0.319	-2.001
15ºF x Plate 1	4	74	116.00	142.5	161.25	180	37.065	134.75	45.353	-0.331	-1.938
15ºF x Plate 2	4	126	144.00	154.5	166.25	188	24.463	155.75	25.617	0.105	-1.917
15ºF x Plate 3	4	110	131.00	149.0	162.00	168	22.239	144.00	25.974	-0.308	-2.047
70ºF x Plate 1	4	34	38.50	57.5	76.25	80	29.652	57.25	23.599	-0.006	-2.397
70ºF x Plate 2	4	106	112.75	118.5	125.50	136	11.861	119.75	12.659	0.197	-1.968
70ºF x Plate 3	4	120	134.25	144.5	156.00	174	22.239	145.75	22.544	0.114	-1.956

If you are using `dplyr`'s `summarize` function, you can achieve similar results by first calling `group_by` and then listing all of your factors. In this case we would want `dplyr::group_by(temperature, material)`.

Fit the Model

There are a couple of different ways that you can specify factorial designs in R: you can manually type in the main the effects and interactions in the order you wish OR you can let R fill in all of the terms for you.

For R, to specify a main effect, you simply type the name of the factor in the formula just as we have been doing all semester.

For an interaction, you'll type the names of **all** main effects involved in the interaction, separating each name with a colon (:). For example, if we wanted the two-way interaction of A and B, we would type `A:B`; for a three-way interaction of A, B, and C, we would type `A:B:C`.

To have R automatically fill in all terms, you simply list each main effect and use `*` to separate terms. Thus, typing `y ~ A*B` is the same as `y ~ A + B + A:B`.

For this example, I'm going to write out the model myself.

```
# Fitting the Two-way ANOVA model
batteryModel <- aov(
  formula = life ~ temperature + material + temperature:material,
  data = battery
)
```

Check Assumptions

Just as with One-way ANOVA with a Block, we still have our core three assumptions to check: Residuals are consistent with following a Gaussian distribution, homoscedasticity, and independence of observations.

Gaussian Residuals

Use a QQ plot like usual:

```
# QQ plot for residuals
car::qqPlot(
  x = residuals(batteryModel),
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
  ylab = "Residuals (hours)"
)
```

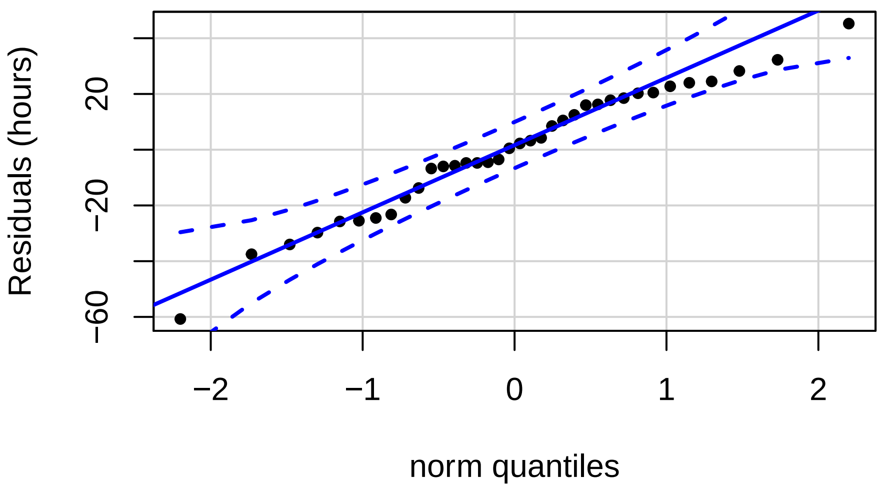

Figure 4: QQ Plot for Residuals

There is very little to be concerned about in our QQ plot; we will go ahead and proceed as if our residuals follow a Gaussian distribution.

Homoscedasticity

Just as in the One-way ANOVA with a Block, we will want to look at a Tukey-Anscombe plot rather than a strip chart for our factorial designs.

```
ggplot(
  data = data.frame(
    residuals = residuals(batteryModel),
    fitted = fitted.values(batteryModel)
  ),
  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,
    size = 0.5
  ) +
  theme_bw() +
  xlab("Fitted values (hours)") +
  ylab("Residuals (hours)")
```

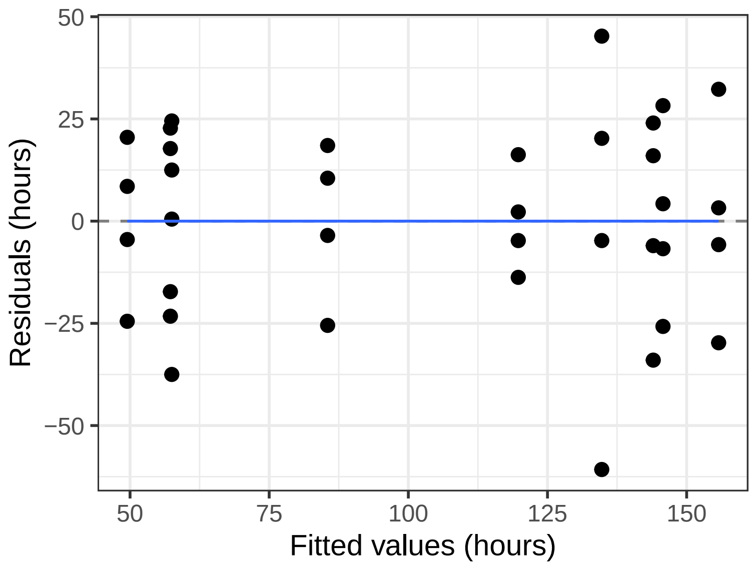

Figure 5: Tukey-Anscombe Plot for Battery Life Span Study

The first thing that I notice in the Tukey-Anscombe plot is that the fourth strip from the left shows the least amount of variation while the fifth strip (from the left) shows the most. The fifth used more than twice the vertical space as the fourth, however, this is the only aspect that causes me a moment of hesitation. There are no discernible patterns to the plot and the blue reference line is perfectly horizontal indicating that we have homoscedasticity.

Independence of Observations

Unfortunately, we don't know measurement order so index plots are not going to be useful here. However, we can think through the study design and reach the decision that we have independent observations.

(I'm leaving this to each of you to practice and come up with a justification for why we can say that we have independence of observations.)

Interaction Plots

With a [Full] Factorial Design, we no longer have a truly additive model. The interaction term in some ways is a measure of how far our model departs from additivity. We want to see whether interactions are important or unimportant: data visualizations are our key to detect this. However, unlike with One-way ANOVA with a Block, we will be okay if we see interactions.

There are several ways that we can look at interactions.

Base R The first method is to use the `interaction.plot` function included in base R.

```
# Using base R to make interaction plot
interaction.plot(
  x.factor = battery$temperature, # First Factor
  trace.factor = battery$material, # Second Factor
  response = battery$life, # Response
  fun = mean,
  type = "b", # Both points and lines
  col = c("black","red","blue"), # Set colors for trace
  pch = c(19, 17, 15),  # Set symbols for trace
  fixed = TRUE,
  legend = TRUE,
  xlab = "Temperature",
  ylab = "Life (hours)",
  trace.label = "Material")
```

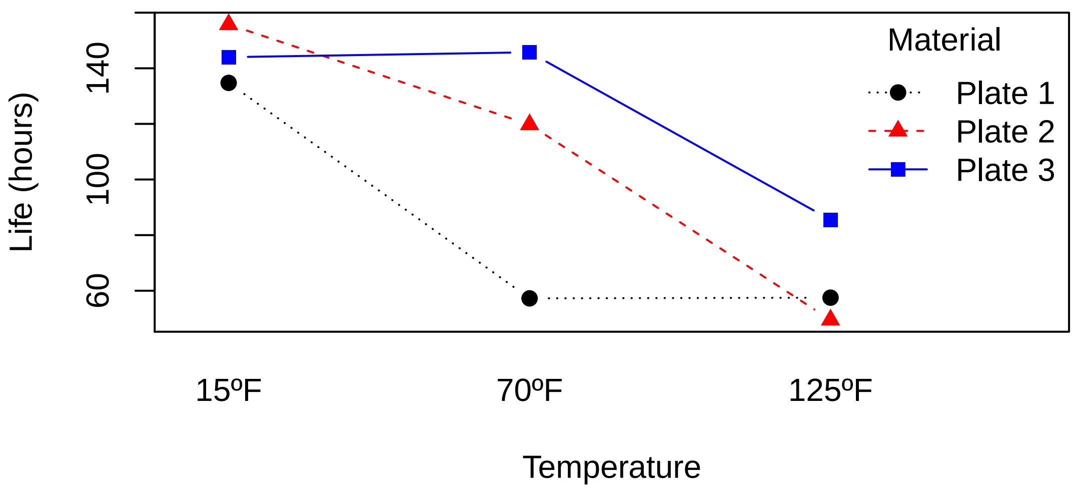

Figure 6: Interaction Plot using base R

GGplot2 We can also use `ggplot2` to create an interaction plot.

```
# Using ggplot to make interaction plot
ggplot(
  data = battery,
  mapping = aes(
    x = temperature,
    y = life,
    color = material,
    group = material
    )
) +
  stat_summary(fun = "mean", geom = "point") +
  stat_summary(fun = "mean", geom = "line") +
  geom_jitter(width = 0.1, height = 0.1, shape = 5) +
  ggplot2::theme_bw() +
  xlab("Temperature") +
  ylab("Life (hours)") +
  labs(color = "Material")
```

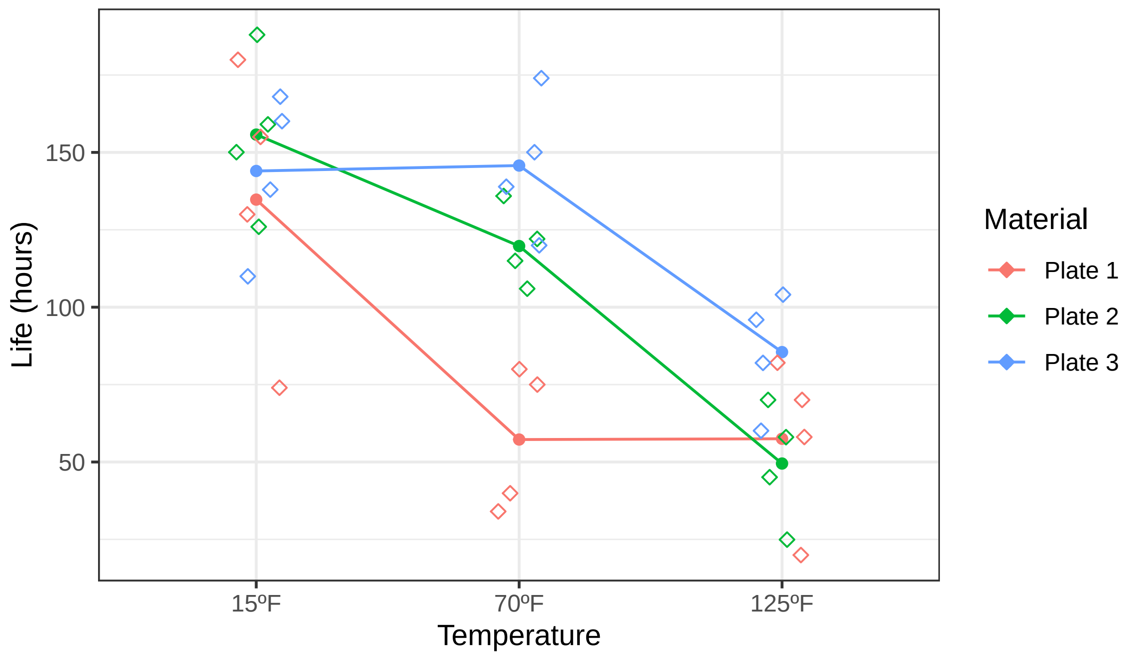

Figure 7: Interaction Plot using ggplot2

Emmeans We can also use the `emmip` function from the `emmeans` package to create an interaction plot.

```
# Using emmeans to make interaction plot

emmeans::emmip(
  object = batteryModel, # our ANOVA model
  # How do we want to arrange our factors
  # formula = color/trace factor ~ horizontal axis
  formula = material ~ temperature
) +
  theme_bw() + # Notice that we can add on ggplot contols
  xlab("Temperature") +
  ylab("Life span (hours)") +
  labs(
    color = "Material"
  )
```

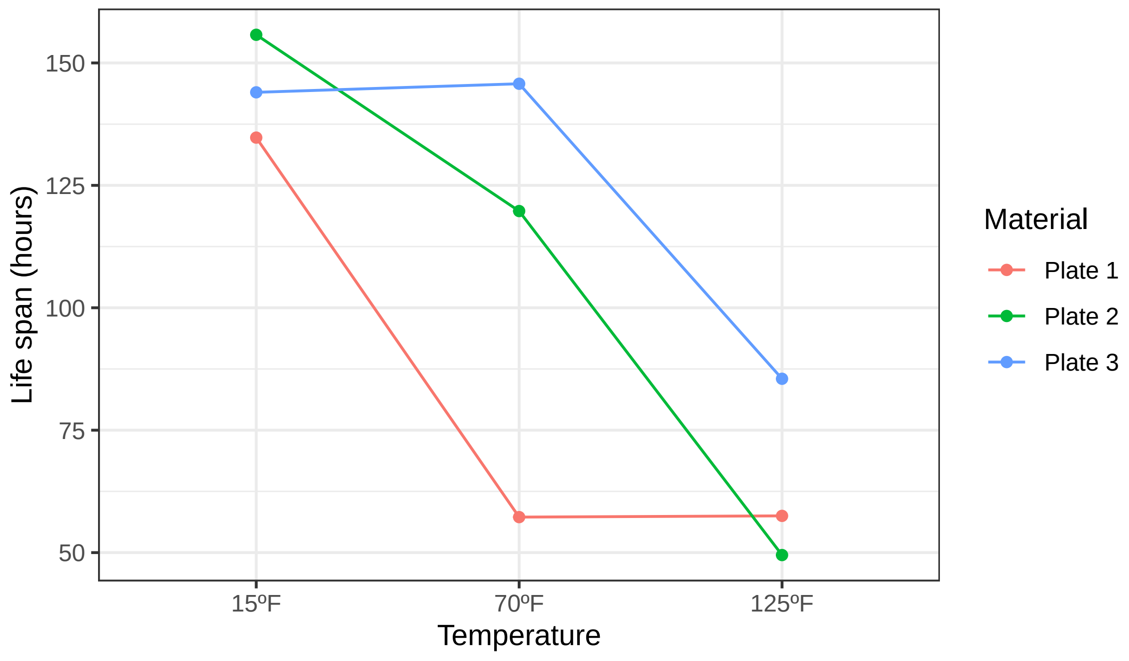

Figure 8: Interaction Plot using emmeans

Each of these three methods have their strengths and their weaknesses. The choice is really up to you and what you want to show in the plot.

Interaction Write Up We will note that there does appear to be some worthwhile interactions between the operating temperature and the plate material. (If there weren't we would anticipate seeing perfectly parallel lines.)

Results

Remember, there are essentially two parts to results: the omnibus test and the post hoc analysis.

Omnibus Results

In this particular situation, we have a **balanced** design, thus we do not need to worry about different types of Sums of Squares.

```r
# Omnibus Test/Modern ANOVA Table
parameters::model_parameters(
    model = batteryModel,
    omega_squared = "partial",
    eta_squared = "partial",
    epsilon_squared = "partial"
) %>%
  knitr::kable(
    digits = 4,
  col.names = c("Source", "SS", "df", "MS", "F", "p-value",
                "Partial Omega Sq.", "Partial Eta Sq.", "Partial Epsilon Sq."),
  caption = "ANOVA Table for Batter Life Span 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 Batter Life Span Study

Source	SS	df	MS	F	p-value	Partial Omega Sq.	Partial Eta Sq.	Partial Epsilon Sq.
temperature	39118.722	2	19559.361	28.9677	0.0000	0.6084	0.6821	0.6586
material	10683.722	2	5341.861	7.9114	0.0020	0.2774	0.3695	0.3228
temperature:material	9613.778	4	2403.444	3.5595	0.0186	0.2214	0.3453	0.2483
Residuals	18230.750	27	675.213					

We treat this type just like we have before, except that now we're interested in **ALL** of the rows and thus we will need to talk about each of the main effects and interactions. The partial effect sizes are still interpreted as proportion of the variation in the response by just that main effect/interaction (all others are dropped).

Post Hoc Analysis

Point Estiamtes I want to quickly remind you that you can get point estimates for your main effects and treatment effects using the `dummy.coef` function. If you need confidence intervals for these, you can use the `confint` function (don't forget to provide an *adjusted* confidence level).

```
# Point Estimates for Battery Factorial Model
## Don't use raw output in your reports, make a nice table
dummy.coef(batteryModel)
```

```
## Full coefficients are
##
## (Intercept):                   105.5278
## temperature:                       15ºF         70ºF         125ºF
##                               39.305556     2.055556    -41.361111
## material:                       Plate 1      Plate 2       Plate 3
##                              -22.361111     2.805556     19.555556
## temperature:material:    15ºF:Plate 1 70ºF:Plate 1 125ºF:Plate 1 15ºF:Plate 2
##                               12.277778   -27.972222     15.694444     8.111111
##
## (Intercept):
## temperature:
##
## material:
##
## temperature:material:    70ºF:Plate 2 125ºF:Plate 2 15ºF:Plate 3 70ºF:Plate 3
##                                9.361111    -17.472222   -20.388889    18.611111
##
## (Intercept):
## temperature:
##
## material:
##
## temperature:material:    125ºF:Plate 3
##                                 1.777778
```

Pairwise Comparisons While you *could* use the pairwise comparison functions we've previously used, a better approach is to embrace our Factorial Design and look at the *estimated marginal means*. These will hold certain factors constant and let others vary. To do this, we will need to use the `emmeans` package.

```
# Pairwise Comparisons
batteryPH <- emmeans::emmeans(
  object = batteryModel,
  # The order of factors does not really matter
  specs = pairwise ~ temperature | material,
  adjust = "tukey",
  level = 0.9
)

as.data.frame(batteryPH$emmeans) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Temperature", "Plate Material", "Marginal Mean","SE", "DF",
                  "Lower Bound","Upper Bound"),
    caption = "Marginal Means-Tukey 90\\% Adjustment",
    align = c("l","l", rep("c", 5)),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("scale_down", "HOLD_position")
  )
```

Table 3: Marginal Means-Tukey 90% Adjustment

Temperature	Plate Material	Marginal Mean	SE	DF	Lower Bound	Upper Bound
15ºF	Plate 1	134.75	12.9924	27	112.6201	156.8799
70ºF	Plate 1	57.25	12.9924	27	35.1201	79.3799
125ºF	Plate 1	57.50	12.9924	27	35.3701	79.6299
15ºF	Plate 2	155.75	12.9924	27	133.6201	177.8799
70ºF	Plate 2	119.75	12.9924	27	97.6201	141.8799
125ºF	Plate 2	49.50	12.9924	27	27.3701	71.6299
15ºF	Plate 3	144.00	12.9924	27	121.8701	166.1299
70ºF	Plate 3	145.75	12.9924	27	123.6201	167.8799
125ºF	Plate 3	85.50	12.9924	27	63.3701	107.6299

The `adjust` argument of `emmeans` allows for the following values for confidence intervals: `"bonferroni"`, `"tukey"`, `"scheffe"`, and `"sidak"`. If you do not want confidence intervals you may use values of `"holm"`, `"hochberg"`, `"hommel"`, `"BH"` (Benjamini and Hochberg), and `"fdr"`.

Effect Sizes

Unfortunately, my `anova.PostHoc` function does not currently work with with factorial models. However, the `emmeans` package provides us with a way to get Cohen's *d*, which then allows us to my `probSup` function to get the Probability of Superiority.

```r
# We want to first narrow our focus and store the marginal means
## You could change the specs to material
tempEMM <- emmeans::emmeans(
  object = batteryModel,
  specs = "temperature"
)

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

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

Table 4: Effect Sizes for Temperature

Comparison	Cohen's d	Probability of Superiority
15ºF - 70ºF	1.434	0.845
15ºF - 125ºF	3.104	0.986
70ºF - 125ºF	1.671	0.881

```r
## Doing the same for material
matEMM <- emmeans::emmeans(
  object = batteryModel,
  specs = "material"
)
cohenMat <- emmeans::eff_size(
  object = matEMM,
  sigma = sigma(batteryModel),
  edf = df.residual(batteryModel)
)
```

```r
as.data.frame(cohenMat) %>%
  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 Material",
    booktab = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = "HOLD_position"
  )
```

Table 5: Effect Sizes for Material

Comparison	Cohen's d	Probability of Superiority
Plate 1 - Plate 2	-0.969	0.247
Plate 1 - Plate 3	-1.613	0.127
Plate 2 - Plate 3	-0.645	0.324

Imbalanced Designs

As mentioned in class, when you have imbalanced designs for factorial models, we have to make a decision about which type of Sums of Squares we want to use.

Type I Sums of Squares–Sequential

If you really want Type I SSQs in a factorial setting (which is often not consistent with most typical SRQs), then you do not need to do anything different than what you have. Just make sure that your `formula` argument of the `aov` call is in the order you want. (Note: I recommend manually entering the model rather than letting R automatically fill in the formula whenever you might have the tinest bit of hestiation over what your model is.)

Type II and Type III Sums of Squares

The vast majority of the time, SRQs for factorial designs revolve around Type II (for model building) and Type III (for testing differences in factor levels) Sums of Squares. To get these sums of squares, I recommend using the `car` package's `Anova` function.

For this example, I'm going to use a data set on different training methods' (fixed, 2 levels) and engery drink's (fixed, 2 levels) impacts on the time to run around a particular track.

```r
# Load Running Data
running <- read.table(
  file = "http://stat.ethz.ch/~meier/teaching/data/running.dat",
  header = TRUE
)

running$method <- as.factor(running$method)
running$drink <- as.factor(running$drink)

# Fit the anova model--same as usual
runningModel <- aov(
  formula = y ~ method*drink, # R interprets this as y ~ method + drink + method:drink
  data = running
)


# Type I Example
## From stats (base) R
anova(runningModel)
```

```
## Analysis of Variance Table
##
## Response: y
##              Df  Sum Sq Mean Sq  F value    Pr(>F)
## method        1 2024.02 2024.02 263.7191 < 2.2e-16 ***
## drink         1  455.25  455.25  59.3164 9.053e-11 ***
## method:drink  1   29.09   29.09   3.7908   0.05579 .
## Residuals    66  506.54    7.67
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
## Remember, we don't want raw output in a professional report

# Type II Example
car::Anova(
  mod = runningModel,
  type = 2
)
```

```
## Anova Table (Type II tests)
##
## Response: y
##               Sum Sq Df  F value    Pr(>F)
## method       1333.41  1 173.7365 < 2.2e-16 ***
## drink         455.25  1  59.3164 9.053e-11 ***
## method:drink   29.09  1   3.7908   0.05579 .
## Residuals     506.54 66
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Notice that the SSQ for Method is different in II than I

# Type III Example
car::Anova(
  mod = runningModel,
  type = 3
)
```

```
## Anova Table (Type III tests)
```

```
##
## Response: y
##               Sum Sq Df    F value    Pr(>F)    
## (Intercept)   136968  1 17846.2330 < 2.2e-16 ***
## method          1352  1   176.2110 < 2.2e-16 ***
## drink            484  1    63.0935 3.347e-11 ***
## method:drink      29  1     3.7908   0.05579 .  
## Residuals        507 66                         
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Notice that the SSQ for Method and drink are different for III than in II and I
```

You will want to present the results in a much more professional way than what I have just done. (I'm just trying to show how you do the coding for different types of SSQs.)

Random Effects Factorial Models

Just as we can have a One-way Random Effects model, we may have a Random Effects Factorial Model. We will want to make sure that we use the `lme4` package's `lmer` function to fit our model for parameter estimation and assumption checking. For the omnibus tests, we may use the the `aov` function. . . with a slight modification.

Context

For this example, we will turn towards industry and look at a common problem in manufacturing. During manufacturing, we often use various types of gauges to take measurements. For example, we might use a pressure gauge to ensure that there is an appropriate amount of air in a closed system or that we've applied the appropriate amount of force. To ensure that our manufacturing is consistent, we might undertake a gauge capability (repeatability & reproducibility) study.

In such a study we'll want to explore our usage of a new gauge. We will use a lottery to select three operators to work with the new gauge and 60 parts (20 for each operator selected via a second lottery). We want to know whether there is significant variance in the gauge readings due

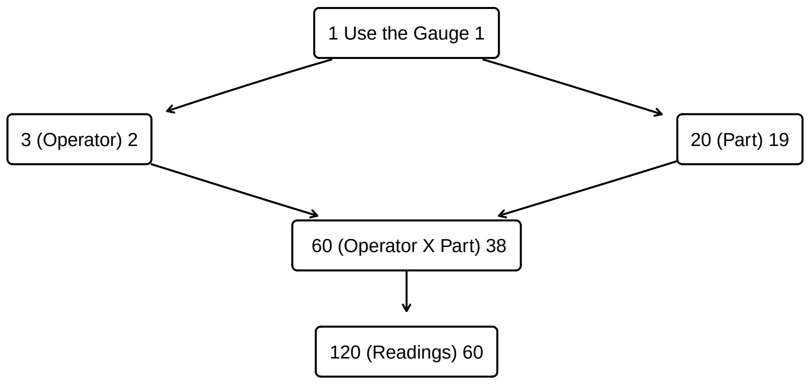

Figure 9: Hasse Diagram for Gauge Study

Figure 9 show the Hasse diagram for the Gauge study. Notice that the parts **are not** our measurement units but the gauge readings are. The factors in this study are Operator and Part, both of which are randomly selected from larger populations, thus this is a Two-way Random Effects model.

Data

The following will be our data for this example:

```
gaugeData <- data.frame(
  partNum = as.factor(rep(c(1:20), 6)),
  operator = as.factor(sort(rep(LETTERS[1:3], 40))),
  measure = c(
    21, 24, 20, 27, 19, 23, 22, 19, 24, 25, 21, 18, 23, 24, 29, 26, 20, 19, 25, 19,
    20, 23, 21, 27, 18, 21, 21, 17, 23, 23, 20, 19, 25, 24, 30, 26, 20, 21, 26, 19,
    20, 24, 19, 28, 19, 24, 22, 18, 25, 26, 20, 17, 25, 23, 30, 25, 19, 19, 25, 18,
    20, 24, 21, 26, 18, 21, 24, 20, 23, 25, 20, 19, 25, 25, 28, 26, 20, 19, 24, 17,
    19, 23, 20, 27, 18, 23, 22, 19, 24, 24, 21, 18, 25, 24, 31, 25, 20, 21, 25, 19,
    21, 24, 22, 28, 21, 22, 20, 18, 24, 25, 20, 19, 25, 25, 30, 27, 20, 23, 25, 17
  )
)
```

Fitting the Models

As a reminder we will fit TWO models one using the `lme4` package for assumptions and estimation and one using the `aov` call for omnibus testing.

```
# Omnibus testing
gaugeOmni <- aov(
  formula = measure ~ operator*partNum,
  data = gaugeData
)

# Random Effects Model
## Note: the * operator IS NOT available with random effects
gaugeModel <- lme4::lmer(
  formula = measure ~ (1|operator) + (1|partNum) + (1|operator:partNum),
  data = gaugeData,
  REML = TRUE
)
```

```
## boundary (singular) fit: see ?isSingular
```

You'll notice that I've allowed for an important message to get printed: "boundry (singular)". When you are fitting Factorial Models with Random Effects, we will want to keep an eye out of these. This message should appear if you run the code chunk as a preview and if you set the `message` chunk option to `TRUE`. This is a good sign that there is an aspect of the model which is unnecessary. Let's check the interaction term.

```
# Interaction Plot for Gauge Study
emmeans::emmip(
  object = gaugeOmni, #Notice our use of aov output
  formula = operator ~ partNum) +
  theme_bw() +
  xlab("Part Number") +
  ylab("Reading") +
  labs(
    color = "Operator"
  )
```

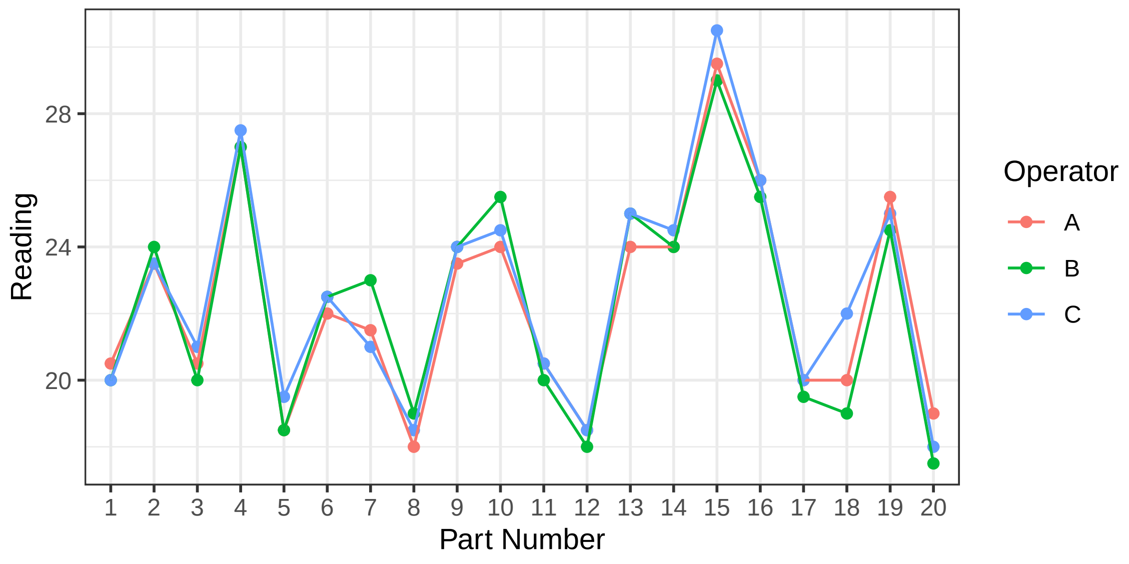

Figure 10: Interaction Plot for Gauage Study

Notice that with only a few exceptions (Parts 2-3, 6-7, 13-14, 17-18) we have fairly parallel line segments. Even those segments which we've called out aren't excessively far off from being parallel. This suggests that we might be able to fit a reduced model dropping the interaction term.

Checking Assumptions

You will check assumptions just as you would for a One-way Random Effects model. The only care that you'll want to take is checking that the treatment effects *for each factor* follow a Gaussian distribution.

```
# Operator Effects
car::qqPlot(
  x = unlist(
    lme4::ranef(
      object = gaugeModel,
      whichel = c("operator")
    )
  ),
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
  ylab = "Operator Effects"
)
```

```
# Part Number Effects
car::qqPlot(
  x = unlist(
    lme4::ranef(
      object = gaugeModel,
      whichel = c("partNum")
    )
  ),
  distribution = "norm",
  envelope = 0.90,
```

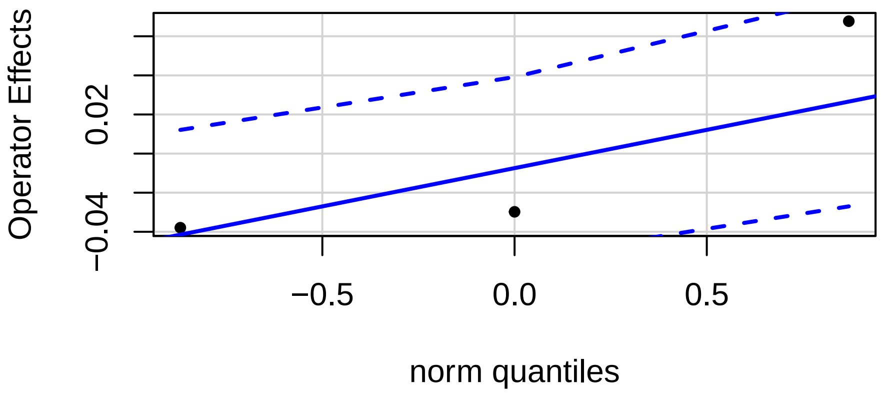

Figure 11: QQ Plots for Each Factor's Treatment Effects

```
  id = FALSE,
  pch = 20,
  ylab = "Part Number Effects"
)
```

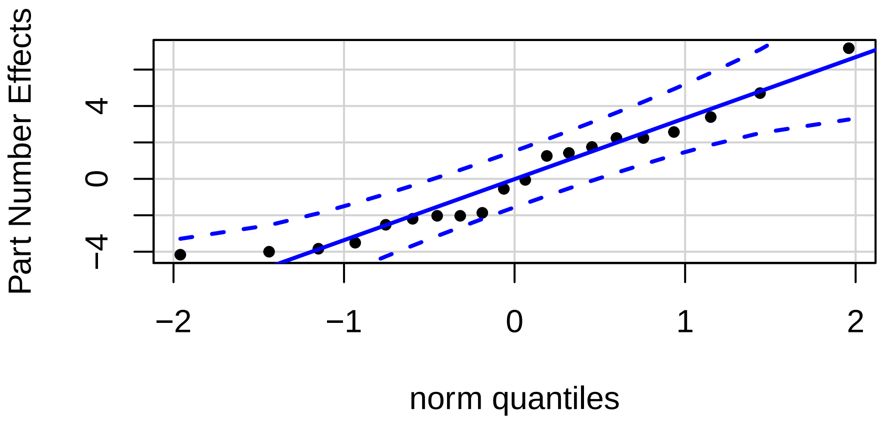

Figure 12: QQ Plots for Each Factor's Treatment Effects

Omnibus Test

Remember that for factorial which are Random or Mixed Effects models, our denominator isn't always the Residual/error term. Unfortunately, R always uses the Residual term for forming F ratios. To help us out, I have created a function, `anovaFixer` that you can use. (You'll need to use the `source` function first just as you do for the `anova.PostHoc` function.)

You'll want to use the `anovaFixer` function first before creating your ANOVA table. Note: the `anovaFixer` function will print messages to the console and will end up in your knitted document. You can suppress this with the `quietly` function from the `purrr` package (see the Post Hoc resource guide for details).

WARNING! The `anovaFixer` function ONLY currently works for Two-way Random Effects models.

```
## [1] "Using the interaction term for F Ratios..."
```

Table 6: ANOVA Table for a Two-way Random

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
operator	2	2.617	1.308	1.838	0.173
partNum	19	1185.425	62.391	87.647	0.000
operator:partNum	38	27.050	0.712	0.718	0.861
Residuals	60	59.500	0.992		

Post Hoc Analysis

Remember that for Random Effects models, we are interested in the variance components and the Grand Mean rather than the treatment effects. Thus, you'll want to make use of the same methods we used in the One-way situation.

Point Estimates

```
# Summary function for quick look
summary(gaugeModel)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: measure ~ (1 | operator) + (1 | partNum) + (1 | operator:partNum)
##    Data: gaugeData
##
## REML criterion at convergence: 409.4
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.0313 -0.6595  0.1270  0.5374  2.7345
##
## Random effects:
##  Groups           Name        Variance Std.Dev.
##  operator:partNum (Intercept)  0.00000 0.0000
##  partNum          (Intercept) 10.25130 3.2018
##  operator         (Intercept)  0.01063 0.1031
##  Residual                      0.88316 0.9398
## Number of obs: 120, groups:  operator:partNum, 60; partNum, 20; operator, 3
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  22.3917     0.7235   30.95
## optimizer (nloptwrap) convergence code: 0 (OK)
## boundary (singular) fit: see ?isSingular
```

Remember, for a report you'll want to present the point estimates more professionally.

Confidence Intervals

Don't forget that you'll need to pass the adjusted/individualized Type I Error Rate to the `confint` function.

```
# Confidence intervals for our random effects parameters
intervals <- confint(
  object = gaugeModel,
  level = 0.98, # Use the adjusted/individualized level here
  oldNames = FALSE
)

row.names(intervals) <- c(
  "Operator-Part Interaction SD",
  "Part SD",
  "Operator SD",
  "Residual SD",
  "Grand Mean"
)

knitr::kable(
  intervals,
  digits = 3,
  caption = "Upper and Lower Confidence Bounds-90\\% SCI, Bonferroni Adj.",
  align = c('l',rep('c',2)),
  booktab = TRUE
) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = "HOLD_position"
  )
```

Table 7: Upper and Lower Confidence Bounds-90% SCI, Bonferroni Adj.

	1 %	99 %
Operator-Part Interaction SD	0.000	0.483
Part SD	2.232	4.764
Operator SD	0.000	0.990
Residual SD	0.803	1.121
Grand Mean	20.662	24.121

Mixed Effects Factorial Models

The last example I'm going to include here is a Three-way Mixed Effects model.

Context

When working with turbines, we often want to track the drop in pressure across an expansion valve. Our client has identified three important attributes which influence pressure drop readings: the temperature (ºF) of the gas on the inlet side, the operator, and the specific pressure gauge used.

Our client has communicated that there is a pool of operators from which we can select 4, a pool of gauges where we can select 3, and that we can set the inlet gas temperature to 60ºF, 75ºF, and 90ºF. We will replicate each possible combination twice.

The client would like to know several things:

1. How should we design the experiment?
2. Are the three attributes actually important?
3. Are any of the interactions of the attributes important?

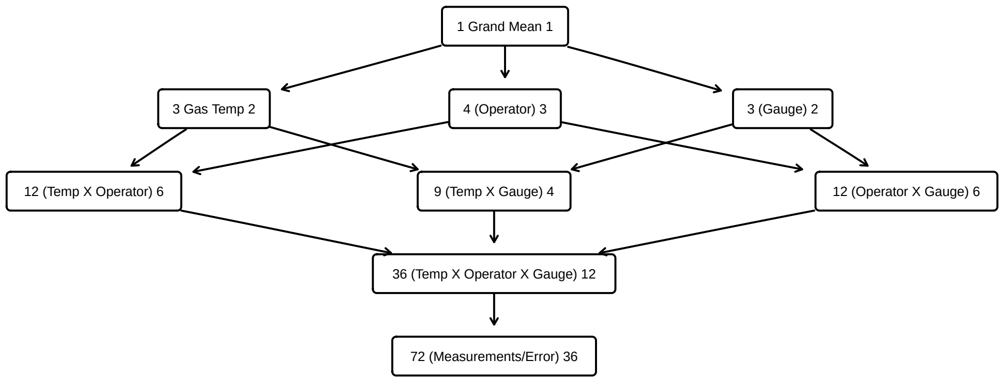

Figure 13: Hasse Diagram for the Pressure Study

Data

```r
pressure <- data.frame(
  gasTemp = as.factor(rep(c(rep(60, 4), rep(75, 4), rep(90, 4)), 6)),
  gauge = as.factor(sort(rep(letters[24:26], 24))),
  operator = as.factor(rep(LETTERS[1:4], 18)),
  coded = c(
    -2, 0, -1, 4, 14, 6, 1, -7, -8, -2, -1, -2,
    -3, -9, -8, 4, 14, 0, 2, 6, -8, 20, -2, 1,
    -6, -5, -8, -3, 22, 8, 6, -5, -8, 1, -9, -8,
    4, -1, -2, -7, 24, 6, 2, 2, 3, -7, -8, 3,
    -1, -4, 0, -2, 20, 2, 3, -5, -2, -1, -4, 1,
    -2, -8, -7, 4, 16, 0, 0, -1, -1, -2, -7, 3
  )
)
```

Fitting the Model

There are two approaches we can take here. The first approach is to fit two models just as we did for Random Effects (a "fixed" model for omnibus testing and a random effects model for estimation). The second approach is to use the `lmerTest` package.

In either case, you'll need to use the `lmer` function. For fixed effects, you'll include those terms in the model as you would in `aov`. For the random effects, you'll need to enclose each term (main effects and interactions) in their own set of parentheses with the `1|`.

Option 1–Two Models

```r
# Fitting the Omnibus Model
pressureOmni <- aov(
  formula = coded ~ gasTemp*operator*gauge,
  data = pressure
)

## Random Effects for Estimation
## Can also be used for Omnibus Test via Conf. Intervals
pressureModel <- lme4::lmer(
  formula = coded ~ gasTemp + (1|operator) + (1|gauge) +
    (1|gasTemp:operator) + (1|gasTemp:gauge) +
    (1|operator:gauge) + (1|gasTemp:operator:gauge),
  data = pressure,
  REML = TRUE
)
```

```
## boundary (singular) fit: see ?isSingular
```

Option 2—`lmerTest`

For this second option, you'll need to load the `lmerTest` package.

```r
# Use the lmerTest package
pressModel <- lmerTest::lmer(
  formula = coded ~ gasTemp + (1|operator) + (1|gauge) +
    (1|gasTemp:operator) + (1|gasTemp:gauge) +
    (1|operator:gauge) + (1|gasTemp:operator:gauge),
  data = pressure
)
```

```
## boundary (singular) fit: see ?isSingular
```

Results

I"m going to present the raw output from these three models so that we can talk about them.

"Fixed" Omnibus

```r
# Omnibus table for "Fixed" three-way ANOVA
car::Anova(
  mod = pressureOmni,
  type = 3
)
```

```
## Anova Table (Type III tests)
##
## Response: coded
##                         Sum Sq Df F value    Pr(>F)
## (Intercept)               8.68  1  0.4056  0.528250
## gasTemp                1023.36  2 23.9072 2.476e-07 ***
## operator                423.82  3  6.6007  0.001141 **
## gauge                     7.19  2  0.1681  0.845952
## gasTemp:operator       1211.97  6  9.4378 3.108e-06 ***
## gasTemp:gauge           137.89  4  1.6106  0.192749
## operator:gauge          209.47  6  1.6312  0.166923
## gasTemp:operator:gauge  166.11 12  0.6468  0.788202
## Residuals               770.50 36
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notice that I used the `car` package's `Anova` function with Type III Sums of Squares. I did this so that I would be covered in the event that I have an imbalanced design.

The biggest issue in using this table is that the denominators are not necessarily correct. However, this table does contain all of the information you would need to calculate the correct denominators. Use your Hasse diagram and this table to construct your appropriate F ratios. You can then get the p-value using the following code:

```r
# Calculating p-values for F-ratios
pf(
  q = f-ratio, # This is your new/corrected f ratio
  df1 = numDF, # Degrees of freedom for numerator
  df2 = denomDF, # Degrees of freedom for denominator,
  lower.tail = FALSE
)
```

Mixed Model

```r
summary(pressureModel)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula:
## coded ~ gasTemp + (1 | operator) + (1 | gauge) + (1 | gasTemp:operator) +
##     (1 | gasTemp:gauge) + (1 | operator:gauge) + (1 | gasTemp:operator:gauge)
##    Data: pressure
##
## REML criterion at convergence: 438.1
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.7201 -0.6468 -0.1159  0.6592  4.0073
##
## Random effects:
##  Groups                 Name        Variance  Std.Dev.
##  gasTemp:operator:gauge (Intercept) 5.708e-09 7.555e-05
##  operator:gauge         (Intercept) 1.268e+00 1.126e+00
##  gasTemp:operator       (Intercept) 2.674e+01 5.171e+00
##  gasTemp:gauge          (Intercept) 7.072e-01 8.409e-01
##  operator               (Intercept) 2.221e-08 1.490e-04
##  gauge                  (Intercept) 0.000e+00 0.000e+00
##  Residual                           1.999e+01 4.471e+00
## Number of obs: 72, groups:
## gasTemp:operator:gauge, 36; operator:gauge, 12; gasTemp:operator, 12; gasTemp:gauge, 9; operator, 4;
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   0.3472     1.6403   0.212
## gasTemp1     -2.9722     2.2737  -1.307
## gasTemp2      5.3194     2.2737   2.340
##
## Correlation of Fixed Effects:
##          (Intr) gsTmp1
## gasTemp1  0.000
## gasTemp2  0.000 -0.500
## optimizer (nloptwrap) convergence code: 0 (OK)
## boundary (singular) fit: see ?isSingular
```

The `summary` call here will produce raw output that you can then draw from to build a professional looking table. All of the tools you would use with Random Effects are still viable here.

lmerTest Results

```
# Results from the lmerTest package
anova(pressModel)
```

```
## Type III Analysis of Variance Table with Satterthwaite's method
##         Sum Sq Mean Sq NumDF  DenDF F value Pr(>F)
## gasTemp 109.94  54.967     2 9.0821  2.7494 0.1165
```

Notice that by using the `lmerTest` package, we can use the base `anova` call and we only get the ANOVA table for the fixed effects. These F ratios should already be appropriately adjusted using Satterthwaite's method since there are three leading, eligible random terms (see Hasse diagram; Satterthwaite's is beyond this course).

To get the rest of the results (i.e., for the random effects), you'll want to use the `summary` function.

```
summary(pressModel)
```

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula:
## coded ~ gasTemp + (1 | operator) + (1 | gauge) + (1 | gasTemp:operator) +
##     (1 | gasTemp:gauge) + (1 | operator:gauge) + (1 | gasTemp:operator:gauge)
##    Data: pressure
##
## REML criterion at convergence: 438.1
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.7201 -0.6468 -0.1159  0.6592  4.0073
##
## Random effects:
##  Groups                 Name        Variance  Std.Dev.
##  gasTemp:operator:gauge (Intercept) 5.708e-09 7.555e-05
##  operator:gauge         (Intercept) 1.268e+00 1.126e+00
##  gasTemp:operator       (Intercept) 2.674e+01 5.171e+00
##  gasTemp:gauge          (Intercept) 7.072e-01 8.409e-01
##  operator               (Intercept) 2.221e-08 1.490e-04
##  gauge                  (Intercept) 0.000e+00 0.000e+00
##  Residual                           1.999e+01 4.471e+00
## Number of obs: 72, groups:
## gasTemp:operator:gauge, 36; operator:gauge, 12; gasTemp:operator, 12; gasTemp:gauge, 9; operator, 4;
##
## Fixed effects:
##             Estimate Std. Error      df t value Pr(>|t|)
## (Intercept)   0.3472     1.6403  9.5184   0.212   0.8368
## gasTemp1     -2.9722     2.2737  9.0821  -1.307   0.2233
## gasTemp2      5.3194     2.2737  9.0821   2.340   0.0438 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##          (Intr) gsTmp1
## gasTemp1  0.000
## gasTemp2  0.000 -0.500
## optimizer (nloptwrap) convergence code: 0 (OK)
## boundary (singular) fit: see ?isSingular
```

My recommendation is that if you are using a mixed model of any size, use the `lmerTest` package.

Assumptions

You'll notice that I haven't said anything about assumptions for the mixed model. This is because you'll do the exact same things as for the the Fixed and Random Effects. This is really just a rinse and repeat type of situation.

Interaction Plots

I do want to take a moment to mention that for interaction plots when you have more than 2 factors are a bit more complicated. You can't just look at one interaction plot; you'll need to look at several–switching around which factors are where each time. This will provide you with the best sense of what is going on with your data.

```
# Interaction of Operator, Gas Temp, and Gauge
emmeans::emmip(
  object = pressureOmni,
  formula = gasTemp ~ operator|gauge
) +
  theme_bw() +
  labs(
    color = "Gas Temp (F)"
  )
```

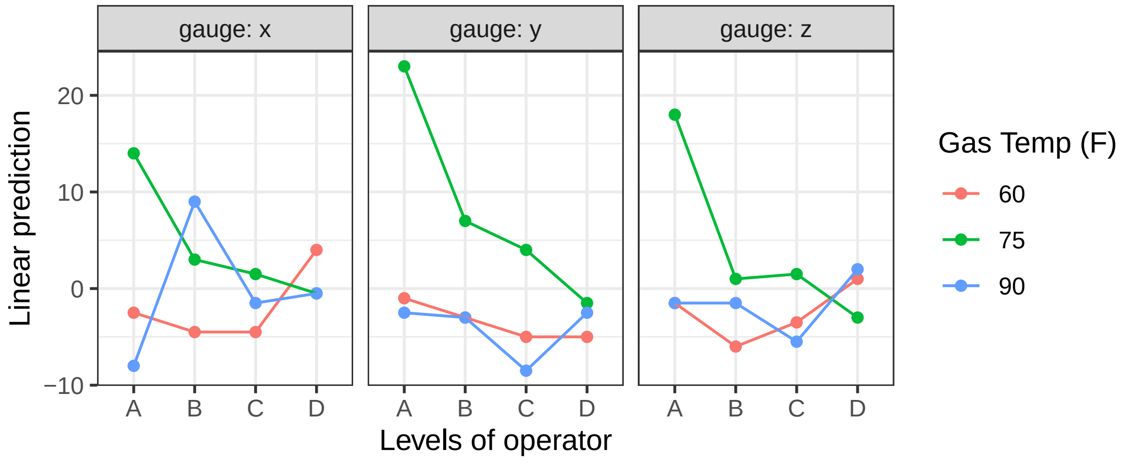

Figure 14: Interaction plot for Operator, Gas Temp, and Gauge

```
# Interaction of Gas Temp and Operator
emmeans::emmip(
  object = pressureOmni,
  formula = gasTemp:operator ~ gasTemp
) +
  theme_bw() +
  xlab("Gas Temperature (F)") +
  labs(
    color = "Operator"
  )
```

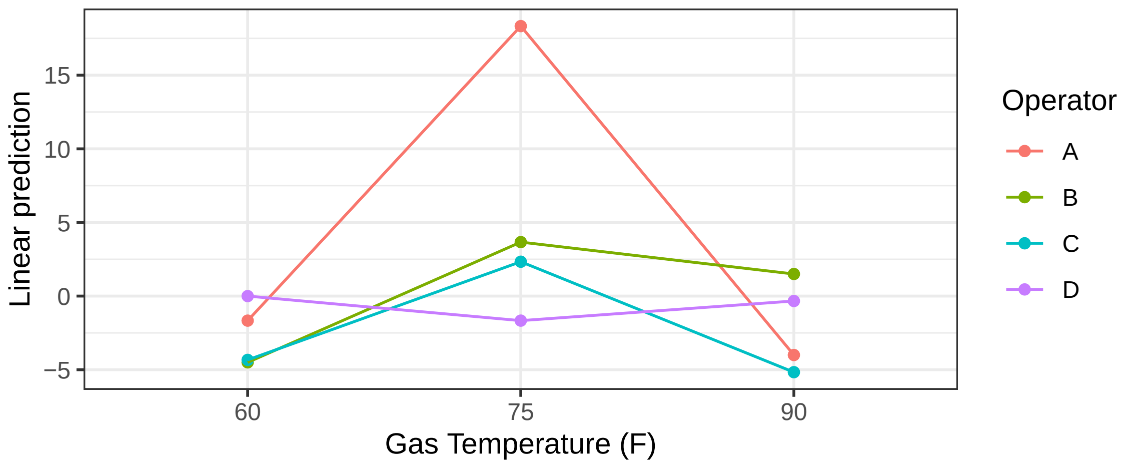

Figure 15: Interaction Plot for Gas Temp and Operator

```
# Interaction of Gas Temp and Gauge
emmeans::emmip(
  object = pressureOmni,
  formula = gasTemp:gauge ~ gasTemp
) +
  theme_bw() +
  xlab("Gas Temperature (F)") +
  labs(
    color = "Gauge"
  )
```

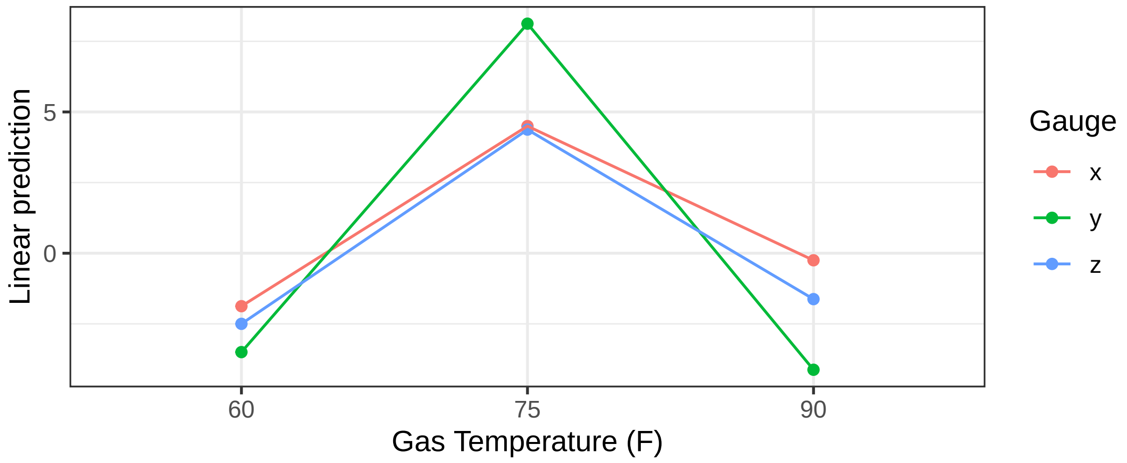

Figure 16: Interaction Plot for Gas Temp and Gauge

Code Appendix

```r
# 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", "DescTools", "emmeans",
              "lme4", "lmerTest")
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")

# Hasse Diagram
modelLabels <- c("1 Maintain Charge 1", "3 Plate 2", "3 Temperature 2",
                 "9 Plate × Temperature 4", "36 (Batteries) 27")
modelMatrix <- matrix(
  data = c(FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, TRUE,
           FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE, TRUE,
           TRUE, TRUE, FALSE),
  nrow = 5,
  ncol = 5,
  byrow = FALSE
)
hasseDiagram::hasse(
 data = modelMatrix,
 labels = modelLabels
)

# Load battery data
battery <- read.table(
  file = "https://raw.github.com/neilhatfield/STAT461/master/dataFiles/batteryLife.dat",
  header = TRUE,
  sep = ","
)

battery$temperature <- dplyr::recode_factor(
  battery$temperature,
  `15` = "15ºF",
  `70` = "70ºF",
  `125` = "125ºF"
)
battery$material <- dplyr::recode_factor(
  battery$material,
  `1` = "Plate 1",
  `2` = "Plate 2",
  `3` = "Plate 3"
)

# Boxplot Example with interaction of factors
boxplot(
  formula = life ~ temperature:material,
  data = battery,
  ylab = "Life (hrs)",
  xlab = "Temp (ºF) x Material"
)

## Ggplot box plot with interaction of factors
ggplot(
  data = battery,
  mapping = aes(
    x = temperature,
    y = life,
    fill = material
  )
) +
  geom_boxplot() +
  theme_bw() +
  xlab("Operating Temperature") +
  ylab("Life span (hours)") +
  labs(
    fill = "Material"
  ) +
  theme(
    legend.position = "top"
  )

# Descriptive statistics by interactions of factors
batteryStats <- psych::describeBy(
  x = battery$life,
  group = paste(battery$temperature, battery$material, sep = " x "),
  na.rm = TRUE,
  skew = TRUE,
  ranges = TRUE,
  quant = c(0.25, 0.75),
  IQR = TRUE,
  mat = TRUE,
  digits = 4
)

batteryStats %>%
  tibble::remove_rownames() %>%
  tibble::column_to_rownames(
    var = "group1"
  ) %>%
  dplyr::select(
    n, min, Q0.25, median, Q0.75, max, mad, mean, sd, skew, kurtosis
  ) %>%
  knitr::kable(
    caption = "Summary Statistics for Battery Life Spans",
    digits = 3,
    format.args = list(big.mark = ","),
    align = rep('c', 11),
    col.names = c("n", "Min", "Q1", "Median", "Q3", "Max", "MAD", "SAM", "SASD",
                  "Sample Skew","Sample Ex. Kurtosis"),
    booktabs = TRUE
  )  %>%
  kableExtra::kable_styling(
    font_size = 12,
    latex_options = c("HOLD_position", "scale_down")
  )

# Fitting the Two-way ANOVA model
batteryModel <- aov(
  formula = life ~ temperature + material + temperature:material,
  data = battery
)

# QQ plot for residuals
car::qqPlot(
  x = residuals(batteryModel),
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
  ylab = "Residuals (hours)"
)


ggplot(
  data = data.frame(
    residuals = residuals(batteryModel),
    fitted = fitted.values(batteryModel)
  ),
  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,
    size = 0.5
  ) +
  theme_bw() +
  xlab("Fitted values (hours)") +
  ylab("Residuals (hours)")
```

```r
# Using base R to make interaction plot
interaction.plot(
  x.factor = battery$temperature, # First Factor
  trace.factor = battery$material, # Second Factor
  response = battery$life, # Response
  fun = mean,
  type = "b", # Both points and lines
  col = c("black","red","blue"), # Set colors for trace
  pch = c(19, 17, 15),  # Set symbols for trace
  fixed = TRUE,
  legend = TRUE,
  xlab = "Temperature",
  ylab = "Life (hours)",
  trace.label = "Material")

# Using ggplot to make interaction plot
ggplot(
  data = battery,
  mapping = aes(
    x = temperature,
    y = life,
    color = material,
    group = material
    )
) +
  stat_summary(fun = "mean", geom = "point") +
  stat_summary(fun = "mean", geom = "line") +
  geom_jitter(width = 0.1, height = 0.1, shape = 5) +
  ggplot2::theme_bw() +
  xlab("Temperature") +
  ylab("Life (hours)") +
  labs(color = "Material")

# Using emmeans to make interaction plot

emmeans::emmip(
  object = batteryModel, # our ANOVA model
  # How do we want to arrange our factors
  # formula = color/trace factor ~ horizontal axis
  formula = material ~ temperature
) +
  theme_bw() + # Notice that we can add on ggplot contols
  xlab("Temperature") +
  ylab("Life span (hours)") +
  labs(
    color = "Material"
  )

# Omnibus Test/Modern ANOVA Table
parameters::model_parameters(
    model = batteryModel,
    omega_squared = "partial",
    eta_squared = "partial",
    epsilon_squared = "partial"
) %>%
  knitr::kable(
    digits = 4,
  col.names = c("Source", "SS", "df", "MS", "F", "p-value",
                "Partial Omega Sq.", "Partial Eta Sq.", "Partial Epsilon Sq."),
  caption = "ANOVA Table for Batter Life Span 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")
  )

# Point Estimates for Battery Factorial Model
## Don't use raw output in your reports, make a nice table
dummy.coef(batteryModel)

# Pairwise Comparisons
batteryPH <- emmeans::emmeans(
  object = batteryModel,
  # The order of factors does not really matter
  specs = pairwise ~ temperature | material,
  adjust = "tukey",
  level = 0.9
)

as.data.frame(batteryPH$emmeans) %>%
  knitr::kable(
    digits = 4,
    col.names = c("Temperature", "Plate Material", "Marginal Mean","SE", "DF",
                  "Lower Bound","Upper Bound"),
    caption = "Marginal Means-Tukey 90\\% Adjustment",
    align = c("l","l", rep("c", 5)),
    booktabs = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = c("scale_down", "HOLD_position")
  )

# We want to first narrow our focus and store the marginal means
## You could change the specs to material
tempEMM <- emmeans::emmeans(
  object = batteryModel,
  specs = "temperature"
)

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

# 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 Temperature",
    booktab = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = "HOLD_position"
  )

## Doing the same for material
matEMM <- emmeans::emmeans(
  object = batteryModel,
  specs = "material"
)
cohenMat <- emmeans::eff_size(
  object = matEMM,
  sigma = sigma(batteryModel),
  edf = df.residual(batteryModel)
)

as.data.frame(cohenMat) %>%
  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 Material",
    booktab = TRUE
  ) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = "HOLD_position"
  )

# Load Running Data
running <- read.table(
  file = "http://stat.ethz.ch/~meier/teaching/data/running.dat",
  header = TRUE
)

running$method <- as.factor(running$method)
running$drink <- as.factor(running$drink)

# Fit the anova model--same as usual
runningModel <- aov(
  formula = y ~ method*drink, # R interprets this as y ~ method + drink + method:drink
  data = running
)


# Type I Example
## From stats (base) R
anova(runningModel)

## Remember, we don't want raw output in a professional report

# Type II Example
car::Anova(
  mod = runningModel,
  type = 2
)

# Notice that the SSQ for Method is different in II than I

# Type III Example
car::Anova(
  mod = runningModel,
  type = 3
)

# Notice that the SSQ for Method and drink are different for III than in II and I


sysLabels <- c("1 Use the Gauge 1", "3 (Operator) 2",
               "20 (Part) 19", " 60 (Operator X Part) 38",
               "120 (Readings) 60")
sysMat <- matrix(data = F, nrow = 5, ncol = 5)
sysMat[1, c(2:5)] = sysMat[c(2:3), c(4:5)] = sysMat[4, 5] = T
hasseDiagram::hasse(sysMat, sysLabels)

gaugeData <- data.frame(
  partNum = as.factor(rep(c(1:20), 6)),
  operator = as.factor(sort(rep(LETTERS[1:3], 40))),
  measure = c(
    21, 24, 20, 27, 19, 23, 22, 19, 24, 25, 21, 18, 23, 24, 29, 26, 20, 19, 25, 19,
    20, 23, 21, 27, 18, 21, 21, 17, 23, 23, 20, 19, 25, 24, 30, 26, 20, 21, 26, 19,
    20, 24, 19, 28, 19, 24, 22, 18, 25, 26, 20, 17, 25, 23, 30, 25, 19, 19, 25, 18,
    20, 24, 21, 26, 18, 21, 24, 20, 23, 25, 20, 19, 25, 25, 28, 26, 20, 19, 24, 17,
    19, 23, 20, 27, 18, 23, 22, 19, 24, 24, 21, 18, 25, 24, 31, 25, 20, 21, 25, 19,
    21, 24, 22, 28, 21, 22, 20, 18, 24, 25, 20, 19, 25, 25, 30, 27, 20, 23, 25, 17
  )
)

# Omnibus testing
gaugeOmni <- aov(
  formula = measure ~ operator*partNum,
  data = gaugeData
)

# Random Effects Model
## Note: the * operator IS NOT available with random effects
gaugeModel <- lme4::lmer(
  formula = measure ~ (1|operator) + (1|partNum) + (1|operator:partNum),
  data = gaugeData,
  REML = TRUE
)

# Interaction Plot for Gauge Study
emmeans::emmip(
  object = gaugeOmni, #Notice our use of aov output
  formula = operator ~ partNum) +
  theme_bw() +
  xlab("Part Number") +
  ylab("Reading") +
  labs(
    color = "Operator"
  )

# Operator Effects
car::qqPlot(
  x = unlist(
    lme4::ranef(
      object = gaugeModel,
      whichel = c("operator")
    )
  ),
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
  ylab = "Operator Effects"
)


# Part Number Effects
car::qqPlot(
  x = unlist(
    lme4::ranef(
      object = gaugeModel,
      whichel = c("partNum")
    )
  ),
  distribution = "norm",
  envelope = 0.90,
  id = FALSE,
  pch = 20,
  ylab = "Part Number Effects"
)

anovaFixer(
  aov.obj = gaugeOmni,
  fixed = NULL, # character vector of Fixed Effects
  random = c("partNum", "operator"), # character vector of random effects
  type = "unrestricted" # restricted or unrestricted
) %>%
  knitr::kable(
  digits = 3,
  #col.names = c("Df", "SS", "MS", "F", "p-value"),
  caption = "ANOVA Table for a Two-way Random",
 # align = c('l',rep('c',5)),
  booktabs = TRUE
) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = "HOLD_position"
  )

# Summary function for quick look
summary(gaugeModel)

# Confidence intervals for our random effects parameters
intervals <- confint(
  object = gaugeModel,
  level = 0.98, # Use the adjusted/individualized level here
  oldNames = FALSE
)

row.names(intervals) <- c(
  "Operator-Part Interaction SD",
  "Part SD",
  "Operator SD",
  "Residual SD",
  "Grand Mean"
)

knitr::kable(
  intervals,
  digits = 3,
  caption = "Upper and Lower Confidence Bounds-90\\% SCI, Bonferroni Adj.",
```

```r
  align = c('l',rep('c',2)),
  booktab = TRUE
) %>%
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "condensed"),
    font_size = 12,
    latex_options = "HOLD_position"
  )

# Hasse Diagram for Pressure Study
pressureLab <- c("1 Grand Mean 1", "3 Gas Temp 2",
               "4 (Operator) 3", "3 (Gauge) 2",
               "12 (Temp X Operator) 6", "9 (Temp X Gauge) 4",
               "12 (Operator X Gauge) 6", "36 (Temp X Operator X Gauge) 12",
               "72 (Measurements/Error) 36")
pressureMat <- matrix(data = F, nrow = 9, ncol = 9)
pressureMat[1, c(2:9)] = pressureMat[2, c(5, 6, 8, 9)] = T
pressureMat[3, c(5, 7, 8, 9)] = pressureMat[4, c(6, 7, 8, 9)] = T
pressureMat[c(5:7), c(8:9)] = pressureMat[8, 9] = T
hasseDiagram::hasse(pressureMat, pressureLab)

pressure <- data.frame(
  gasTemp = as.factor(rep(c(rep(60, 4), rep(75, 4), rep(90, 4)), 6)),
  gauge = as.factor(sort(rep(letters[24:26], 24))),
  operator = as.factor(rep(LETTERS[1:4], 18)),
  coded = c(
    -2, 0, -1, 4, 14, 6, 1, -7, -8, -2, -1, -2,
    -3, -9, -8, 4, 14, 0, 2, 6, -8, 20, -2, 1,
    -6, -5, -8, -3, 22, 8, 6, -5, -8, 1, -9, -8,
    4, -1, -2, -7, 24, 6, 2, 2, 3, -7, -8, 3,
    -1, -4, 0, -2, 20, 2, 3, -5, -2, -1, -4, 1,
    -2, -8, -7, 4, 16, 0, 0, -1, -1, -2, -7, 3
  )
)

# Fitting the Omnibus Model
pressureOmni <- aov(
  formula = coded ~ gasTemp*operator*gauge,
  data = pressure
)

## Random Effects for Estimation
## Can also be used for Omnibus Test via Conf. Intervals
pressureModel <- lme4::lmer(
  formula = coded ~ gasTemp + (1|operator) + (1|gauge) +
    (1|gasTemp:operator) + (1|gasTemp:gauge) +
    (1|operator:gauge) + (1|gasTemp:operator:gauge),
  data = pressure,
  REML = TRUE
)

# Use the lmerTest package
pressModel <- lmerTest::lmer(
  formula = coded ~ gasTemp + (1|operator) + (1|gauge) +
    (1|gasTemp:operator) + (1|gasTemp:gauge) +
    (1|operator:gauge) + (1|gasTemp:operator:gauge),
  data = pressure
)

# Omnibus table for "Fixed" three-way ANOVA
car::Anova(
  mod = pressureOmni,
  type = 3
)

# Calculating p-values for F-ratios
pf(
  q = f-ratio, # This is your new/corrected f ratio
  df1 = numDF, # Degrees of freedom for numerator
  df2 = denomDF, # Degrees of freedom for denominator,
  lower.tail = FALSE
)

summary(pressureModel)

# Results from the lmerTest package
anova(pressModel)
summary(pressModel)

# Interaction of Operator, Gas Temp, and Gauge
emmeans::emmip(
  object = pressureOmni,
  formula = gasTemp ~ operator|gauge
) +
  theme_bw() +
  labs(
    color = "Gas Temp (F)"
  )

# Interaction of Gas Temp and Operator
emmeans::emmip(
  object = pressureOmni,
  formula = gasTemp:operator ~ gasTemp
) +
  theme_bw() +
  xlab("Gas Temperature (F)") +
  labs(
    color = "Operator"
  )

# Interaction of Gas Temp and Gauge
emmeans::emmip(
  object = pressureOmni,
  formula = gasTemp:gauge ~ gasTemp
) +
  theme_bw() +
  xlab("Gas Temperature (F)") +
  labs(
    color = "Gauge"
  )
```