Factorial Models

Neil J. Hatfield

4/15/2022

In this tutorial, we are going to explore Factorial Treatment Structures/Factorial Designs in R. For our purposes here, we will restrict our attention to [full] factorial models with all Fixed Factor Effects. (We will allow for our measurement units to be the only random effect term.) The general structure for this guide will be:

- Setting Up R
  - Loading Packages, Setting Options, and Additional Tools
- Data Contexts
  - Load! Aim! Ready! Release! (Gummy Bear Catapult)
  - Battery Manufacturing
- Exploring the Factorial Treatment Structure
- Fit the Models
  - Appropriateness of ANOVA
  - Check Interactions
  - Form the Model
- Assessing Assumptions
  - Gaussian Residuals
  - Homoscedasticity
  - Independence of Observations
- Results
  - Omnibus
  - Point Estimates
  - Post Hoc–Pairwise
  - Post Hoc–Contrasts
  - Effect Sizes
- Dealing with Imbalanced Designs

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.

I've also added in the `psych` package to demonstrate an approach you can use for getting values of descriptive statistics in accordance with the factorial treatment structure. You'll also see the `openxlsx` package in the list; this happens to be my preferred way to read in XLSX files.

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", "DescTools", "emmeans",
```

```
              "openxlsx")
lapply(packages, library, character.only = TRUE)

## Set options and constraint
options(knitr.kable.NA = "")
options(contrasts = c("contr.sum", "contr.poly"))

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

Data Contexts

For this guide/tutorial, we'll make use of two contexts: **Load! Aim! Ready! Release! (Gummy Bears)** and **Battery Manufacturing (Batteries)**.

Load! Aim! Ready! Release! (Gummy Bears)

This data comes from the design that we put together in class to explore the impacts of launch angle and launch position for our spoon-apults for launching gummy bears to see how far they will fly.

```
# Demo code for loading and cleaning data
## Loading gummy bear data
catapultData <- openxlsx::readWorkbook(
  xlsxFile = "https://raw.github.com/neilhatfield/STAT461/master/dataFiles/gummyBears2022.xlsx",
  sheet = 1,
  colNames = TRUE,
  rowNames = FALSE
)

## Change variables
names(catapultData)[which(names(catapultData) == "Launch.Angle")] <- "Angle"
names(catapultData)[which(names(catapultData) == "Launch.Position")] <- "Position"
names(catapultData)[which(names(catapultData) == "Measurement.Order")] <- "Order"

## Fix the data
## R will treat high and High as TWO separate levels
## Same with low & Low, back & Back, and front & Front
catapultData <- catapultData %>%
  dplyr::mutate(
    Angle = dplyr::recode_factor(
      Angle,
      "high" = "High", # Old Value = New Value
      "High" = "High",
      "low" = "Low",
      "Low" = "Low"
    ),
    Position = dplyr::recode_factor(
      Position,
      "back" = "Back",
      "Back" = "Back",
      "front" = "Front",
      "Front" = "Front"
    ),
    ## This is optional but I'm going to standardize team names
    Team = dplyr::case_when(
      !grepl(pattern = "Team", Team) ~ paste("Team", Team),
      TRUE ~ Team
```

```
    )
  )
```

Notice that I used a couple of functions from `dplyr` to help me clean and take control of the data. The `mutate` function allows you to change an entire column of a data frame in a consistent and relatively speedy way. The `recode_factor` function combines two things: alter values for consistency AND tell `R` to treat the attribute as a factor. Note: the order you list the new values will be the order `R` uses for the factor levels.

Battery Manufacturing

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—15°F, 70°F, and 125°F—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?

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

## Clean data
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"
)
```

Exploring the Factorial Treatment Structure

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.

Box Plots Revisited

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.

```
# Demo code for box plots using factorial treatment structure
## Battery Manufacturing
boxplot(
  formula = life ~ temperature:material,
  data = battery,
  ylab = "Life (hrs)",
  xlab = "Temp (ºF) x Material"
)
```

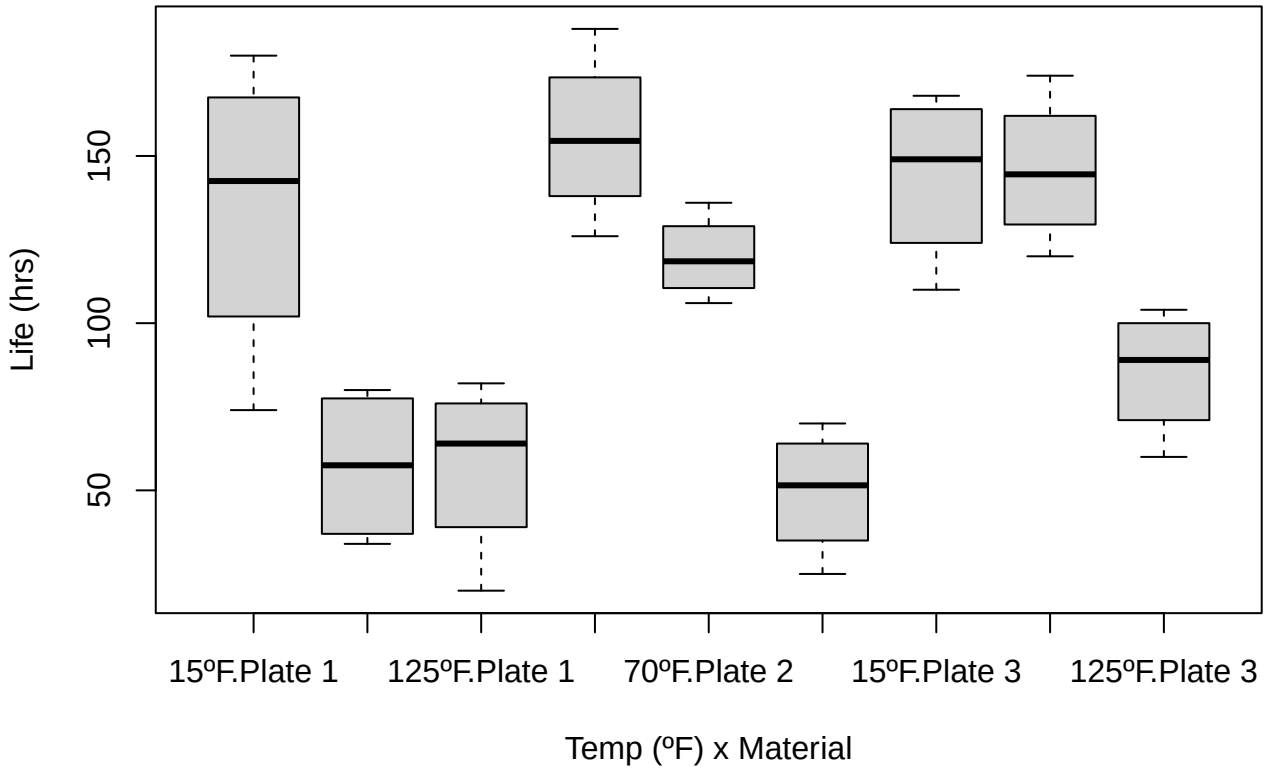

Figure 1: Box Plot of Battery Life Spans by Temperature and Plate Material

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`.

```
# Demo code of box plots incorporating factorial treatment structure
## ggplot2 and Gummy Bears Study
ggplot(
  data = catapultData,
  mapping = aes(
    x = Angle,
    y = Distance,
    fill = Position
  )
) +
  geom_boxplot() +
  theme_bw() +
  xlab("Launch Angle") +
  ylab("Distance (in)") +
  labs(
    fill = "Launch Position"
  ) +
  theme(
    legend.position = "right",
```

```
    text = element_text(size = 14)
  )
```

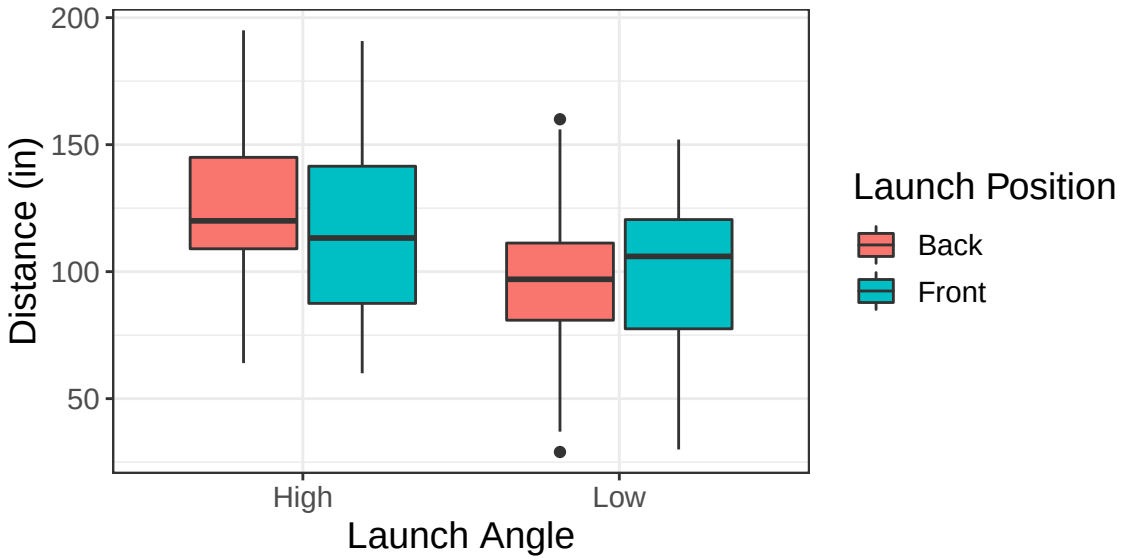

Figure 2: Box Plot With Multiple Factors–Gummy Bears Study

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:

```
# Demo code for descriptive statistics for factorial treatment structure
## Battery Manufacturing
batteryStats <- psych::describeBy(
  x = battery$life,
  # Notice how we're getting the factorial treatments
  group = paste(battery$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
125°F x Plate 1	4	20	48.50	64.0	73.00	82	17.791	57.50	26.851	-0.466	-1.864
125°F x Plate 2	4	25	40.00	51.5	61.00	70	18.532	49.50	19.261	-0.195	-2.015
125°F x Plate 3	4	60	76.50	89.0	98.00	104	16.309	85.50	19.279	-0.319	-2.001
15°F x Plate 1	4	74	116.00	142.5	161.25	180	37.065	134.75	45.353	-0.331	-1.938
15°F x Plate 2	4	126	144.00	154.5	166.25	188	24.463	155.75	25.617	0.105	-1.917
15°F x Plate 3	4	110	131.00	149.0	162.00	168	22.239	144.00	25.974	-0.308	-2.047
70°F x Plate 1	4	34	38.50	57.5	76.25	80	29.652	57.25	23.599	-0.006	-2.397
70°F x Plate 2	4	106	112.75	118.5	125.50	136	11.861	119.75	12.659	0.197	-1.968
70°F 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)`.

```
# Demo code for descriptive statistics for factorial treatment structure
## Gummy Bears
catapultData %>%
  dplyr::group_by(Angle, Position) %>%
  summarize(
    n = n(),
    min = min(Distance),
    Q1 = quantile(Distance, probs = c(0.25)),
    med = median(Distance),
    Q3 = quantile(Distance, probs = c(0.75)),
    max = max(Distance),
    mad = mad(Distance),
    sam = mean(Distance),
    sd = sd(Distance),
    skew = psych::skew(Distance),
    kurtosis = psych::kurtosi(Distance)
  ) %>%
  knitr::kable(
    caption = "Summary Statistics for Gummy Bear Study",
    digits = 3,
    format.args = list(big.mark = ","),
    align = rep('c', 13),
    col.names = c("Angle", "Position", "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 2: Summary Statistics for Gummy Bear Study

Angle	Position	n	Min	Q1	Median	Q3	Max	MAD	SAM	SASD	Sample Skew	Sample Ex. Kurtosis
High	Back	75	64	109.000	120.00	145.00	195.0	26.687	125.190	28.451	0.265	-0.474
High	Front	75	60	87.500	113.25	141.50	190.8	38.918	114.935	31.050	0.284	-0.915
Low	Back	75	29	80.875	97.00	111.25	160.0	24.092	96.390	26.295	-0.114	0.098
Low	Front	75	30	77.500	106.00	120.50	152.0	25.204	98.473	27.877	-0.630	-0.479

Either approach (`psych::describeBy` or `dplyr::group_by` and `dplyr::summarize`) works for getting values of descriptive statistics in accordance to the factorial treatment structure.

Fit the Models

Before we write code to fit the factorial model in `R`, we should do two things: 1) double check that a factorial ANOVA approach is appropriate, and 2) check for interactions.

Appropriateness of ANOVA

As we have been doing since Unit 3, checking for whether ANOVA methods are appropriate for answering our SRQ comes down to the following:

- Do we have a quantitative response?
- Do we have two or more categorical/qualitative factors? (If only one, then we're not factorial ANOVA.)
- Do we have enough *Degrees of Freedom* to be able to estimate the Main Effects and Interactions?
- Do we have enough *Degrees of Freedom* for estimating residuals/errors?
- (For Main Effects Only Models: do we have an additive model?)

As before, our knowledge of the study design and the Hasse diagram can help us out.

Figure 3 shows the Hasse diagram for the Battery Manufacturing study. We know that the response is quantitative (number of hours of life). From the Hasse diagram, we have two factors (Plate Material and Temperature) which are both categorical. We are doing a full factorial structure (we have all possible interactions). Further, since we have positive *Degrees of Freedom* for each node in the Hasse diagram, we know that we should be able to estimate all main effects, interactions, and the residuals/errors.

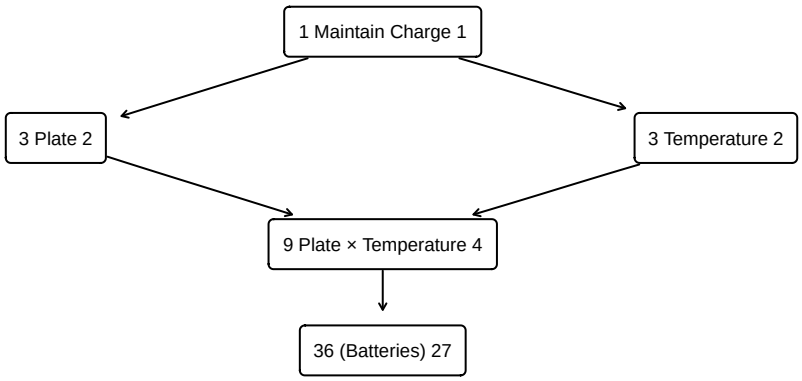

Figure 3: Hasse Diagram for Battery Manufacturing Study

Your Turn

Create a similar paragraph as I did but for the Gummy Bears situation.

Check Interactions

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 interaction plots.

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")
```

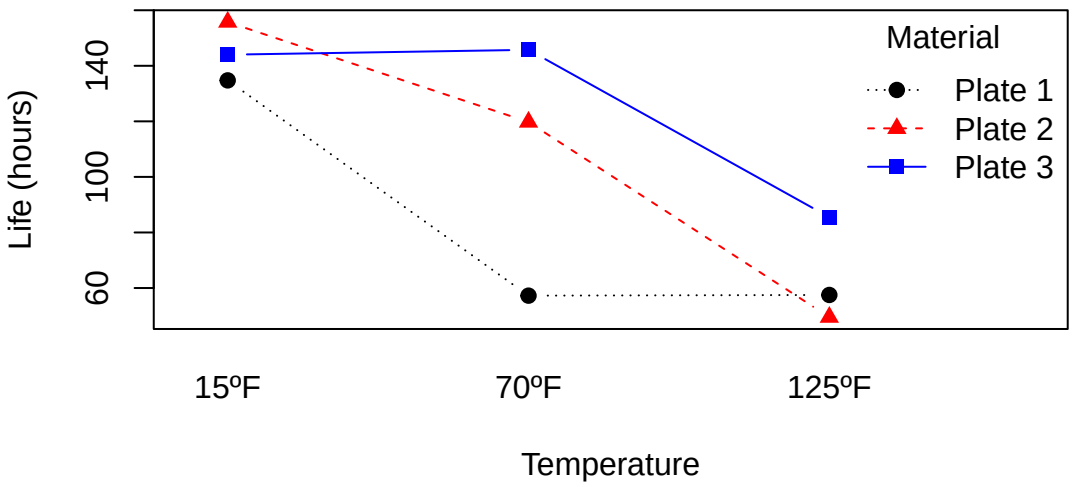

Figure 4: Interaction Plot using base R

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")
```

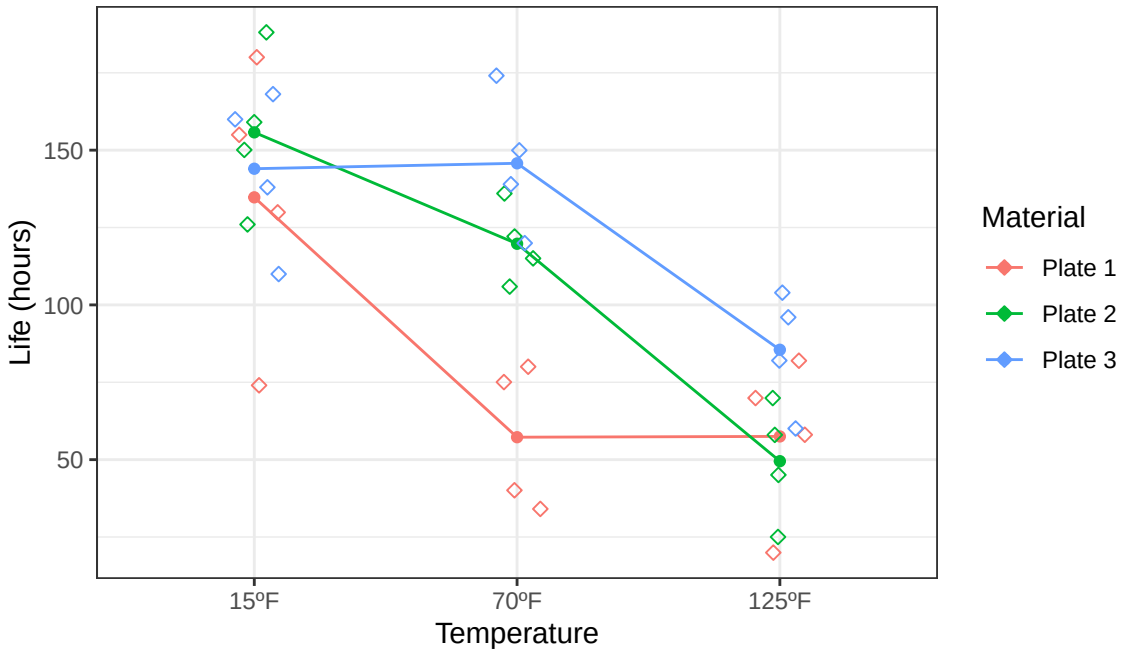

Figure 5: Interaction Plot using ggplot2

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.)

- Form the Model
- Assessing Assumptions
  - Gaussian Residuals
  - Homoscedasticity
  - Independence of Observations
- Results
  - Omnibus
  - Point Estimates
  - Post Hoc–Pairwise
  - Post Hoc–Contrasts
  - Effect Sizes
- Dealing with Imbalanced Designs

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)"
)
```

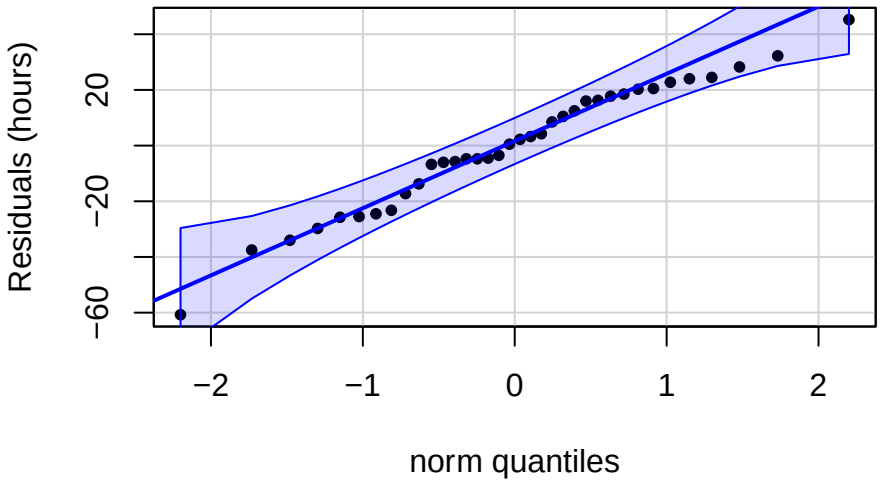

Figure 6: QQ Plot for Residuals

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)")
```

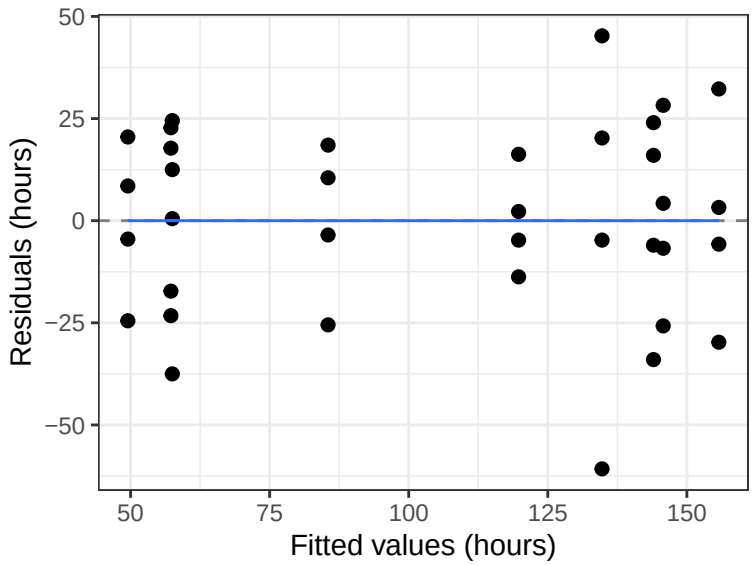

Figure 7: Tukey-Anscombe Plot for Battery Life Span Study

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.)

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.

```
# 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 3: 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 Estimates 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 4: Marginal Means-Tukey 90% Adjustment

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

```
# 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 5: Effect Sizes for Temperature

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

```
## 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"
  )
```

Table 6: 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.

```
# 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
```

```
## 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
```

```
# 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.)

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", "DescTools", "emmeans")
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")

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

## Set options and constraint
options(knitr.kable.NA = "")
options(contrasts = c("contr.sum", "contr.poly"))

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

# Demo code for loading and cleaning data
## Loading gummy bear data
catapultData <- openxlsx::readWorkbook(
  xlsxFile = "https://raw.github.com/neilhatfield/STAT461/master/dataFiles/gummyBears2022.xlsx",
  sheet = 1,
  colNames = TRUE,
  rowNames = FALSE
)

## Change variables
names(catapultData)[which(names(catapultData) == "Launch.Angle")] <- "Angle"
names(catapultData)[which(names(catapultData) == "Launch.Position")] <- "Position"
names(catapultData)[which(names(catapultData) == "Measurement.Order")] <- "Order"

## Fix the data
## R will treat high and High as TWO separate levels
## Same with low & Low, back & Back, and front & Front
catapultData <- catapultData %>%
  dplyr::mutate(
    Angle = dplyr::recode_factor(
      Angle,
      "high" = "High", # Old Value = New Value
      "High" = "High",
      "low" = "Low",
```

```
      "Low" = "Low"
    ),
    Position = dplyr::recode_factor(
      Position,
      "back" = "Back",
      "Back" = "Back",
      "front" = "Front",
      "Front" = "Front"
    ),
    ## This is optional but I'm going to standardize team names
    Team = dplyr::case_when(
      !grepl(pattern = "Team", Team) ~ paste("Team", Team),
      TRUE ~ Team
    )
  )

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

## Clean data
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"
)

# Demo code for box plots using factorial treatment structure
## Battery Manufacturing
boxplot(
  formula = life ~ temperature:material,
  data = battery,
  ylab = "Life (hrs)",
  xlab = "Temp (ºF) x Material"
)

# Demo code of box plots incorporating factorial treatment structure
## ggplot2 and Gummy Bears Study
ggplot(
  data = catapultData,
  mapping = aes(
    x = Angle,
    y = Distance,
    fill = Position
  )
) +
  geom_boxplot() +
  theme_bw() +
```

```
  xlab("Launch Angle") +
  ylab("Distance (in)") +
  labs(
    fill = "Launch Position"
  ) +
  theme(
    legend.position = "right",
    text = element_text(size = 14)
  )

# Demo code for descriptive statistics for factorial treatment structure
## Battery Manufacturing
batteryStats <- psych::describeBy(
  x = battery$life,
  # Notice how we're getting the factorial treatments
  group = paste(battery$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")
  )

# Demo code for descriptive statistics for factorial treatment structure
## Gummy Bears
catapultData %>%
  dplyr::group_by(Angle, Position) %>%
  summarize(
    n = n(),
    min = min(Distance),
    Q1 = quantile(Distance, probs = c(0.25)),
    med = median(Distance),
    Q3 = quantile(Distance, probs = c(0.75)),
    max = max(Distance),
    mad = mad(Distance),
```

```
    sam = mean(Distance),
    sd = sd(Distance),
    skew = psych::skew(Distance),
    kurtosis = psych::kurtosi(Distance)
  ) %>%
  knitr::kable(
    caption = "Summary Statistics for Gummy Bear Study",
    digits = 3,
    format.args = list(big.mark = ","),
    align = rep('c', 13),
    col.names = c("Angle", "Position", "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")
  )

# 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
)

# 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")

# 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)")

# 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
```