Flexplot: Graphically-Based Data Analysis

Dustin Fife[1]

[1] Rowan University

Author Note

I wish to thank the dozens of students who very patiently discovered bugs in the flexplot software. Your efforts have resulted in a much more robust tool. Many of the ideas and information contained in this article have been disseminated elsewhere, including at APA's annual conference and on my YouTube channel. Furthermore, Flexplot is the engine driving the visualizations in JASP and Jamovi and various publications have referenced the ideas contained within Flexplot (including blog posts both on JASP and Jamovi's website, as well as a Behavioral Research Methods publication).

Correspondence concerning this article should be addressed to Dustin Fife, 201 Mullica Hill Road Glassboro, NJ 08028. E-mail: fife.dustin@gmail.com

Abstract

The human visual processing system has enormous bandwidth, able to interpret vast amounts of data in fractions of a second (Otten, Cheng, & Drewnowski, 2015). Despite this amazing ability, there is a troubling lack of graphics in scientific literature (Healy & Moody, 2014), and the graphics most traditionally used tend to bias perception in unintentional ways (Weissgerber, Milic, Winham, & Garovic, 2015). I suspect the reason for the underuse and misuse of graphics is because sound graphs are difficult to produce with existing software (Wainer, 2010). While `ggplot2` allows immense flexibility in creating graphics, its learning curve is quite steep, and even basic graphics require multiple lines of code. `flexplot` is an R package that aims to address these issues by providing a formula-based suite of tools that simplifies and automates much of the graphical decision-making. Additionally, `flexplot` pairs well with statistical modeling, making it easy for researchers to produce graphs that map onto statistical procedures. With one-line functions, users can visualize bivariate statistical models (e.g., scatterplots for regression, beeswarm plots for ANOVA/t-tests), multivariate statistical models (e.g., ANCOVA and multiple regression), and even more sophisticated models like multi-level models and logistic regressions. Further, this package utilizes old tools (e.g., added variable plots and coplots) as well as introduces new tools for complex visualizations, including ghost lines and point sampling.

Keywords: flexplot, statistical assumptions, statistical modeling, graphics, exploratory data analysis, graphical data analysis

Flexplot: Graphically-Based Data Analysis

Introduction

Graphics offer many advantages over traditional methods that rely on tables and reported statistics. First, graphs are perhaps the most important tool available to enhance transparency, as they provide a medium to display the data in their entirety (Pastore, Lionetti, & Altoè, 2017; Tay, Parrigon, Huang, & LeBreton, 2016). Audiences can see, at a glance, the appropriateness of the model, the size of the effect, and the degree of uncertainty.

Furthermore, graphs improve communication between the scientific community and the public (Otten, Cheng, & Drewnowski, 2015). Lay people can understand relatively sophisticated statistical procedures when that information is presented graphically (Correll, 2015), which may mitigate the recent uptick in scientific mistrust (Camargo & Grant, 2015).

Another advantage of graphics is that they highlight problems with models that are masked by traditional statistical procedures, such as nonlinearity, outliers, and heteroscedasticity (Healy & Moody, 2014; Levine, 2018). As such, graphics serve as an important diagnostic check, one that is overwhelmingly ignored by most researchers (Hoekstra, Kiers, & Johnson, 2012).[1] As noted by Wilkinson, "If you assess hypotheses without examining your data, you risk publishing nonsense" (Wilkinson & Task Force on Statistical Inference, 1999, p. 597).

Finally, and perhaps most importantly, graphics improve encoding (Hansen, Chen, Johnson, Kaufman, & Hagen, 2014). Nearly half of the brain is devoted to visual processing,

[1] It is possible that, while graphs are underused for *presentation* of results, they might be heavily used for *analysis*. Plots for analyses are often less polished and can be used to guide analytic decisions, while plots for presentation are more refined and are designed for clear communication. Both plots are important, though (arguably) the latter is more important since it allows audiences to evaluate the merits of the statistical model. Flexplot aims to have plots that are near-presentation worthy, though some tweaks may be required (e.g., changing the axis labels).

and human visual processing can encode information in as little as a tenth of a second (Abbott, Do, & Byrne, 2012; Marieb, 1992; Otten, Cheng, & Drewnowski, 2015; Semetko & Scammell, 2012). Consider the image displayed in Figure 1, taken from Correll (2015). The table on the left presents the same information as the one on the right, though the encoding of the information on the right is much easier because that information is conveyed *visually.*

While visualizations have many advantages, they can also be used to mislead, sometimes intentionally, and sometimes not (Correll, 2015; Kosslyn, 2006; Pandey, Rall, Satterthwaite, Nov, & Bertini, 2015). For example, when means/standard errors are presented as barcharts, people tend to judge values below the mean (i.e., within the confines of the bar) as far more likely than values above the mean, even when the underlying distribution is symmetric (Correll, 2015). To further complicate matters, the default images in most point-and-click statistical software violate important visualization heuristics (Fife, Tremoulet, & Longo, 2019; Healy & Moody, 2014; Wainer, 2010). For example, in `SPSS` it is impossible (as far as I know) to produce standard error plots that display raw data (jittered or otherwise). Also, in standard error plots, the axes are scaled to the means, not the range of the actual data, which visually inflates the size of the effect. In addition, producing some types of graphics (e.g., Skew-Location plots) requires more effort than many are willing to perform (the user must model the data, export the residuals, then produce a scatterplot).

This paper introduces `flexplot`, an R package specifically designed to remove obstacles to sound visualizations. The graphics produced by `flexplot` were developed using empirically-derived heurstics (see Fife, Longo, Correll, & Tremoulet, 2020 for a review). Also, the graphics produced are simple to generate and permit analysts to quickly shift between statistical modeling and graphical interpretation.

In the following section, I begin by outlining the guiding philosophy behind `flexplot`, then introduce its grammar. I then spend the remainder of the paper demonstrating how to produce intuitive graphics with `flexplot` and illustrate how to pair these tools with

statistical modeling.

Guiding Philosophy of Flexplot

The traditional approach to data analysis requires a great deal of decision-making up-front, but requires little effort to interpret results. Initially, the analyst must choose between regression, ANOVA, *t*-tests, MANOVA, etc. If the analyst then chooses to graph these results, they must *again* decide whether a boxplot, histogram, bar chart, scatterplot, etc. is most appropriate. By the time the analyst interprets the data, a great deal of effort has already been expended, though the traditional approach to data-analysis makes interpreting the results unambiguous; one simply needs to determine whether the p-value dips below the critical 0.05 threshold.

This approach has been routinely criticized (Cohen, 1994; Gigerenzer, 2004; Tyron, 1998), particularly in how results are interpreted (significant versus not). While the traditional approach requires great effort to choose the analysis and little effort to interpret it, `flexplot` takes the opposite approach: little effort is required to *choose* the analysis, leaving more resources to *interpret* the analysis. This is as it should be.

To accomplish this goal, `flexplot` is based on the following principles (Fife, Longo, Correll, & Tremoulet, 2020):

1. Minimize obstacles to producing graphics. The easier it is to produce graphics, the more likely they will be used, and the more resources the researcher will have available to interpret the results. Technology companies spend millions of dollars attempting to make the interaction between humans and technology as seamless as possible. One-click purchasing, voice-activated personal assistants like Siri, movies-on-demand, and audible app notifications are all innovations that are successful because they make it easy for humans to use their technology. Likewise, if producing a graphic is as simple (or simpler) than performing a statistical analysis, they too will become heavily utilized (and, dare I say, addictive?).

Furthermore, the less effort required to produce them, the more resources available to invest in *interpreting* graphics. To make producing graphics as simple as possible, `flexplot` automates much of the decision-making in the background, such as choosing between types of graphics (e.g., histograms versus bar charts) and how those graphics are displayed.

2. Design graphics that leverage human strenths and mitigate human biases. Successful technology capitalizes on human strengths. A mobile phone, for example, leverages our advanced finger tactile sensitivity and dexterity. Sending text messages with one's toes would be a very poor choice. Likewise, a computer that sends olefactory information might work well for a dog, but not a human. Visualization technology ought to be designed with the same principles in mind. Unfortunately, standard statistical analyses do not capitalize on human strengths. It takes a great deal of training to understand even basic statistics, and even then results are frequently misinterpreted (Gigerenzer, 2004). To put it in the words of Tyron (1998), traditional analyses have a "human factors" problem. To overcome misconceptions about statistical analyses, some of the tools within `flexplot` create visual representations of the statistical models. These representations highlight uncertainty, reveal whether chosen models are appropriate, and improve encoding of statistical information.

`ggplot2` Versus `flexplot`

Hadley Wickham, the author of `ggplot2`, developed a grammar of graphics (Wickham, 2010), or a set of rules that guide the construction of visualizations. Furthermore, Wickham's grammar is "layered," which means that plotting elements can be stacked atop one another. For example, one could "layer" dots over an X/Y axis, layer a regression line over the dots, layer a correlation coefficient over the regression line, etc. Wickham's grammar allows a great deal of flexibility in the design of graphics. However, this flexibility comes at a cost. Very often the grammar necessary to produce a graphic requires a great deal of coding to produce. For example, consider the code to create jittered mean plots:

```
require(ggplot2)
require(flexplot)
plot = ggplot(data = exercise_data, aes(x=therapy.type, y=weight.loss)) +
      # x/y axis layer
  geom_jitter(width = .2, alpha = .4) +
      # point layer
  stat_summary(fun.y = 'mean', geom = 'point',
        size = 3, position = position_dodge(width = .2)) +
      # summary point layer
  stat_summary(geom = 'errorbar', fun.ymin = function(z){mean(z)-1.96*sd(z)},
        fun.ymax=function(z) {mean(z)+1.96*sd(z)},
        size = 1.25, width = .2, position = position_dodge(width = .2))
      # "errorbar" layer
```

I have personally spoken to many veteren R users who have been extremely reluctant to adopt `ggplot2`, simply because the approach and sytax are elaborate, if not complicated. For those less experienced, the prospect of leveraging `ggplot2` is even more daunting, which means that few will likely abandon point-and-click software to produce graphics.

As noted earlier, the more difficult it is to produce a graphic, the more likely it is someone will simply not use it. A similar graphic can be produced with only one line of code using the `flexplot` function:

```
plot = flexplot(weight.loss ~ therapy.type, data = exercise_data)
```

Naturally, this simplicity comes at a cost; `flexplot` is more limited than `ggplot2`. However, it was not designed to be able to produce any graphic conceivable. Rather it was designed to visualize statistical models with ease, and will cover the majority of graphics analysts will use for modeling. However, in the end, graphics produced through `flexplot` are still `ggplot2` objects. As such, they can be edited and/or layered for further

customization, which I will demonstrate throughout this paper.

The Grammar of the General Linear Model

`flexplot`'s grammar is also a layered grammar (but incidentally, because it was developed within `ggplot2`'s grammar), though its grammar is actually based on the linear model (LM).[2] Recall that most statistical procedures are subsumed within the LM, which is essentially regression. For example, a *t*-test is simply a regression where the intercept is the mean of the referent group (e.g., the control group) and the slope is simply the difference between the treatment and control groups. The base `R` function `lm` utilizes LMs to do various sorts of modeling, and all this is accomplished using a simple formula (e.g., `y ~ x1 + x2`). Likewise, `flexplot` adopts the same convention, utilizing a similar formula to produce graphics. The advantage of this approach is that there is notational consistency from modeling to visualization. Very often the exact code used to produce a fitted model can also be used to visualize the data, as in the following example:

```
model =       lm(A ~ B + C, data = d)
plot  = flexplot(A ~ B + C, data = d)
```

This simplifies the choice of graphics immensely; one only needs to specify the predictor variable(s) (and sometimes make some choices of paneling). Otherwise, `flexplot` handles the decision-making in the background.

Figure 3 is an illustration that identifies which components of a `flexplot` equation produce each element of a `flexplot`-style graphic. The first variable in the equation ($X1$) specifies which variable is displayed on the X axis. The second variable ($X2$) is displayed as different colors/symbols/lines. The third and fourth elements are shown as column and row panels, respectively.

---

[2] The package `vizreg` (Breheny & Burchett, 2017) is another R package that provides seamless integration with linear models.

By following the grammar of a LM, there is consistency between statistical modeling and graphing, with a few notable exceptions. Most obviously, a LM formula (e.g., `plot(y ~ x1 + x2 + x3, data = data)`) does not have any vertical pipes (`|`) as `flexplot` does (though other modeling procedures and R packages do, such as mixed models in `lmer`; coplots in the `car` packages; and the `visreg` package). The vertical pipe is necessary in `flexplot` to allow more specificity in paneling. Additionally, with LMs, one must explicitly specify interaction (e.g., `X1:X2`) and polynomial terms (e.g., `I(X2^2)`). This is not necessary with `flexplot`; the raw data are displayed exactly as they are and if interactions are present, the graph will show it.

The base `plot()` function in `R` follows similar conventions as `flexplot` (i.e., users can specify a formula, such as `plot(y ~ x)`), though `flexplot` is more intelligent in its choice of displays.[3] Also, `plot()` only allows the user to visualize one variable at a time. Another function, `coplot()`, allows some multivariate visualizations, yet it is limited in the types of data allowed and, like `plot()` is not flexible in the types of visualization decisions it makes. The `flexplot` package, on the other hand, offers great flexibility and automates much of the decision-making.

In the following section, I will demonstrate how decisions are made in `flexplot`. I begin by showing how to produce univariate graphics, then bivariate graphics, then multivariate graphics. I will then follow that up with various functions and techniques for combining graphs with models, then conclude with a brief summary.

---

[3] R's `plot()` function is a S3 method, so it is actually offers a great deal of flexibility for visualizing all sorts of R objects, much like flexplot's `visualize()` method. However, its flexibility is limited to package developers who must create a class to leverage the `plot()` method. The point I'm making here is not about the S3 `plot()` method, but about R's generic `plot()` function.

Univariate Graphics

In `lm()`, one can fit an "intercept only" model, using the code `lm(y ~ 1)`. This is equivalent to estimating the mean of y. `flexplot` follows a similar convention for graphing univariate distributions. Alternatively, one can also write this as `flexplot(y~y, ...)`. The type of graphic displayed depends on the type of variable inputted into the function. `flexplot` graphs numeric variables in histograms and categorical variables as barcharts. For example, in the code below, notice that `flexplot` recognizes whether the variable is categorical or numeric, and plots accordingly (see Figure 4).

```
require(patchwork) #### for combining plots into one
require(flexplot)
data(exercise_data) #### these are simulated data available
                    #### in flexplot.
a = flexplot(weight.loss ~ 1, data = exercise_data)
b = flexplot(therapy.type ~ 1, data = exercise_data)
a + b
```

`flexplot` follows standard conventions when producing histograms. When producing barcharts, on the other hand, `flexplot` deviates from defaults. The order of the x-axis is typically sorted alphabetically in `ggplot2`, which violates the "principle of relevance" for visual processing (Kosslyn, 2006). This principle suggests that all features of a graphic should present meaningful information. Humans are primed to attend to visual differences, and stacking large and small N categories side-by-side willy nilly taxes the visual processing system into believing these differences are meaningful. `flexplot` automatically sorts the categories by sample size (the category with the largest N is left-most on the X axis), which also enhances the ability to "chunk" pieces of information meaningfully (Kosslyn, 2006), as in Figure 5. If the categories do have a meaningful order (e.g., if the variables are ordinal), the user can convert the variable of interest to an ordered factor.

Sometimes, `flexplot` will make a wrong guess, if, for example, a categorical variable is recorded as a number (e.g., 1 = Group 1, 2 = Group 2, 3 = Group 3). To force `flexplot` to produce a barchart, one can either convert the variable to a factor (e.g., `data$group = factor(data$group, levels = 1:3, labels = c("Group1", "Group2", "Group3"))`). Alternatively, one can use the `factor` function within a `flexplot` function (e.g., `flexplot(y~factor(group), data=d)`).

Bivariate Graphics

As with univariate graphics, `flexplot` will automatically produce an appropriate bivariate graphic, depending on the type of predictors and outcome variable: a numeric predictor/outcome will produce a scatterplot, a numeric predictor/categorical outcome = logistic curve graph, categorical predictor/numeric outcome = beeswarm plot, and categorical predictor/categorical outcome = association plot.

With all graphics produced by `flexplot` it will display the raw data. Although Tufte (2001) advocated for plots that minimize the "data to ink ratio," subsequent investigation of visual perception have shown little evidence that minimizing ink in a graph improves visual perception (Inbar, Tractinsky, & Meyer, 2007). Rather, raw data is essential for sound visual interpretation. Vastly different patterns can produce identical summary statistics such as means, variances, slopes, intercepts, etc. (Anscombe, 1973). Raw data allow one to determine, at a glance whether these summary statistics accurately reflect the raw data. Furthermore, research has shown that humans are quite adept at visually aggregating information from raw data without summary information (see Correll, 2015 for a review), while they cannot accurately surmise raw data from summary statistics. In other words, if anything is to be omitted, it should be the summary statistics (e.g., regression lines, whiskers in a boxplot, dots of means), not the raw data.

Visualizing raw data can become tricky, particularly when categorical variables are

involved. With categorical variables, there is bound to be a great deal of overlap (e.g., if the treatment group has 100 participants, 100 individuals will have identical scores on the X axis when plotted, which will tend to mean datapoints will overlap). In the next section, I will explain how `flexplot` handles overlapping datapoints from categorical predictors.

Categorical Predictor, Numeric Outcome (Beeswarm Plots)

There are many different ways to graph a categorical predictor/numeric outcome relationship, including bar plots of means, box plots, violin plots, gradient plots, etc. Some are misleading (e.g., barplots of means and standard error plots). Others are mediocre (e.g., boxplots). Finally, some perform exceptionally well in human testing, including violin plots and gradient plots (Correll, 2015). As mentioned in the previous section, raw data are essential. To minimize overlap, a common strategy is to utilize "jittering," which means that categorical variables are first converted to values (e.g., control group = 1, treatment group = 2), then random noise is added to each participant's score (e.g., a 1 may become 1.012 or 0.097). This reduces overlap between datapoints. However, simply jittering values uniformly robs one of a valuable opportunity to convey additional information in a graphic. `flexplot` instead jitters data proportional to the density of the data. In other words, regions of little density will have very little jittering and regions of high density (e.g., at the mode) will have more jittering. These sorts of plots are essentially violin plots with raw data, where the maximum amount of jittering is the outline of the violin plot. Traditionally, these were called "textured dot strips," which were invented by Tukey and Tukey (1990; see also Wilkinson, 1999). Others (e.g., Eklund, 2012) call them "bee swarm" plots. I am not too fond of the original name, so I'll refer to them as bee swarm plots throughout this text.

While violin plots partially solve the problem mentioned earlier (i.e., that summary statistics can be generated from a diverse set of patterns of raw data), they too have their limitations. For one, it is impossible to tell the difference between a dataset with 15,000 versus 15 observations. For example, the two distributions in the left image in Figure 6 look

essentially the same, while the same data, plotted as beeswarm plots, very clearly show the sample size.

```
group1 = c(0,1,2,2,3,3,3,3,3,3,4,4,5,6)
group2 = rnorm(10000,3,1)
d = data.frame(score = c(group1, group2),
               group = factor(c(rep("group 1", times = length(group1)),
                        rep("group 2", times = length(group2))), ordered=T))
a = ggplot2::ggplot(data = d, aes(x = group, y = score)) +
  geom_violin() + theme_bw()
b = flexplot(score ~ group, data = d)
a + b
```

In `flexplot`, one can control the amount of jittering. The amount can be specified in multiple ways: as a boolean (`TRUE` means it will jitter, `FALSE` it will not), as a number (e.g., `0.2`), or as a vector (e.g., `c(.2, .4)`, which will indicate .2 jittering for X and .4 for Y). Just as it is in `geom_jitter()`, this number refers to the amount of jittering on either side. However, the value refers to the *maximum* amount the computer will jitter the data. So, 0.2 (the default) will jitter up to 0.1 points on the right, but only at the highest density and 0.1 on the left at the highest density.

Users can also specify what the "whiskers" mean for the summary statistics. They default to the interquartile range (with the median as the center dot), but the user can also specify `sterr`, or `stdev`, to indicate the standard error or standard deviation (see Figure 7):

```
theme_update(axis.text.x = element_text(angle=90, hjust=100))
a = flexplot(weight.loss ~ therapy.type, data = exercise_data,
             jitter = F, spread = "quartile") +
             labs(x="")
b = flexplot(weight.loss ~ therapy.type, data = exercise_data,
```

```
              jitter = c(.4,.5), spread = "sterr") +
              labs(x="")
c = flexplot(weight.loss ~ therapy.type, data = exercise_data,
              jitter = .2, spread = "stdev") +
              labs(x="")
a + b + c + plot_layout(nrow = 1)
```

Numeric Predictor, Numeric Outcome (Scatterplots)

The indisputed king of numeric on numeric visualization is the scatterplot. Once again, `flexplot` is smart enough to choose a scatterplot when it is passed a numeric predictor and numeric outcome. Except for severe departures, people tend to believe the fit of a line overlaid on raw data, even when the line is not appropriate (Fife, Tremoulet, & Longo, 2019). For this reason, `flexplot` defaults to graphing a loess line as the summary statistic, so as to highlight deviations from linearity. However, the user can specify other sorts of fits, such as `"lm"` (for regression), `"quadratic"`, `"cubic"`, `"logistic"`, "`poisson`," "`gamma`," and `"rlm"` (robust linear model) in the `MASS` package (Ripley et al., 2013). The user can also choose to remove the confidence interval by specifying `se = F`, as well as jitter one or both variables, as shown below and in Figure 8.

```
a = flexplot(weight.loss ~ satisfaction, data = exercise_data) +
  theme_minimal() ### using layering to change theme
b = flexplot(weight.loss ~ satisfaction, data = exercise_data,
              method = "lm", se = F)
c = flexplot(weight.loss ~ satisfaction, data = exercise_data,
              method = "polynomial", jitter = .4)
a + b + c + plot_layout(nrow = 1)
```

Though `flexplot` defaults to a loess line, if the analyst models the data using another fitted function (e.g., regression, cubic, robust), the final display should reflect that (Umanath

& Vessey, 1994; Vessey, 1991). This process is seamless when one uses the `visualize` function, discussed shortly.

Numeric Predictor, Categorical Outcome (Logistic Curves)

`flexplot` also has some ability to model categorical outcomes. One common situation might be when one is attempting to model a binary outcome, as they would in a logistic regression. In this situation, it is critical that the graph match the analysis (Fife, 2020). This aligns with the principle of "cognitive fit," which suggests that the type of display matches the type of information conveyed (Umanath & Vessey, 1994; Vessey, 1991). Because logistic regressions utilize ogive curves to model the data, the graphs ought to reflect that.

Any binary variable can be graphed as a logistic regression in `flexplot`, except when the axis variable (i.e., the variable occupying the first slot in the `flexplot` equation) is also categorical. To model logistic curves, the user only needs to specify `logistic` as the method (see Figure 9).

```
data("tablesaw.injury") ### also simulated data available
                        ### in flexplot package
flexplot(injury ~ attention, data = tablesaw.injury,
             method = "logistic", jitter = c(0, .05))
```

Categorical Outcome, Categorical Predictor (Association Plots)

Sometimes the analyst may wish to graph the relationship between two categorical variables. Once again, `flexplot` is smart enough to determine that information from the formula, provided the user supplies two factors. In this situation, `flexplot` will generate an "association" plot, which plots the deviation of each cell from its expected frequencies (divided by the expected values within that cell). The reason for an association plot (as opposed to a traditional barplot) is because it best maps into what sorts of questions viewers are interested in asking. When users model the association between categorical variables,

they traditionally use a χ^2 test, which compares observed versus expected frequencies. An association plot displays observed (height of bar) versus expected (y axis at zero) frequencies, thus following the principle of cognitive fit (Umanath & Vessey, 1994; Vessey, 1991; see also Kosslyn, 2006 for a similar principle, the "principle of compatibility").

In the example below, I had to convert `injury` to a factor to get a barplot. The graphic (Figure 10) shows that females are less likely to be injured than males, relatively speaking.

```
tablesaw.injury = within(tablesaw.injury, {injury = factor(injury,
                    levels=c(0, 1), labels=c("all good", "ouch"))})
flexplot(injury ~ gender, data = tablesaw.injury)
```

Repeated Measures Data (Related *t*-test)

One of the guiding tenets of `flexplot` is that every statistical analysis ought to be accompanied by a graphic that closely matches the analysis. This not only improves encoding of statistical information, but it also highlights uncertainty and reveals the appropriateness of the model. With a related *t*-test, the existing graphics will not accurately represent this model because a related *t* actually models the *difference* between scores (e.g., from Time 1 to Time 2). As such, `flexplot` allows an additional option (`related = TRUE`) that tells `flexplot` to plot the differences, rather than the groups. To do so, `flexplot` requires "tidy" data, or data where time is indicated in one column and the score in the other. Also, there must be equal numbers of observations in each group. Once in this format, it simply plots the differences (Figure 11). For example:

```
data("plant_growth")
flexplot(Diameter ~ Soil.Type, data = plant_growth, related=T) +
  theme(axis.title=element_text(size=12,face="bold"))
```

(Note, this dataset didn't actually contain repeated measures data. This is merely for illustrative purposes).

Unfortunately, plotting difference scores only works with two timepoints. When there are more than two timepoints, I recommend graphing these relationships with the `visualize()` function using mixed models. (I'll address `visualize()` shortly). In the plant growth graphic, the differences seem centered around zero, indicating that the type of potting soil used (store-bought potting soil versus a "secret" custom mix I found online) didn't make a difference in seedling diameter.

Avoiding Overlap

If it wasn't yet apparent, let me be less subtle: I think all graphics should include raw data. Showing raw data allows readers to determine whether the chosen model is appropriate, and it communicates the degree of uncertainty about the model. However, when there are a large number of datapoints, it increases cognitive load and masks salient characteristics (Kosslyn, 2006). This makes it quite difficult to see any patterns; areas of high density look just as crowded as areas of lower density, relatively speaking (although having bee swarm plots makes it clear which areas are most dense; see the example in top-left image in Figure 12). To address such overlap, `flexplot` offers three options (aside from making the plot bigger, of course). The first is to suppress raw data (`raw.data = F`, right-top in Figure 12). I don't recommend that, but it can be done.

A second option is to reduce the transparency (e.g., bottom-left in Figure 12). This will draw more attention to the fit of the model than the raw data (e.g., users will attend to the regression line rather than the raw data; see Kosslyn, 2006), which may or may not be a good thing. Perhaps the best option is to *sample* (bottom-right in Figure 12). Sampling allows the visual-processing system to not be overly influenced by the fit, but without overwhelming the visual processing system. However, it is important that the visual display of fit (e.g., median + IQR, loess line, regression line) not be estimated from the sampled data. Rather, the fit should correspond to the entire dataset. `flexplot` performs this operation in the background. In Figure 12, notice how the medians/interquartile ranges do

not change, despite having different numbers of datapoints.

```
data("nsduh")
a = flexplot(distress ~ major.dep, data = nsduh)
b = flexplot(distress ~ major.dep, data = nsduh, raw.data = F)
c = flexplot(distress ~ major.dep, data = nsduh, alpha = .005)
d = flexplot(distress ~ major.dep, data = nsduh, sample = 200)
a + b + c + d + plot_layout(nrow = 2)
```

Multivariate Graphics

Graphing multivariate relationships can become quite tricky. It is very easy to induce cognitive overload, especially when attempting to visualize raw data (which is, again, a key characteristic of `flexplot`). Some might be inclined to create three-dimensional plots. However, these are difficult to interpret and they require the user to rotate the view. Even then, they can only show two predictors at a time. I find it much easier to use other strategies. `flexplot` utilizes four different strategies to visualize multivariate relationships: (1) plotting a dimension as different colors/lines/shapes, (2) plotting a dimension in row or column panels, (3) visualizing conditional relationships with added variable plots, and (4) overlaying ghost lines.

Added Variable Plots (AVPs) With `added.plot()`

AVPs are underused, yet extremely useful.[4] Essentially, an AVP shows the relationship between a predictor of interest and the *residuals* of an existing model. (Alternatively, one can use partial residual plots, which are based on a similar principle. See Fife, 2021). For example, if one wanted to understand the relationship between `therapy.type` and `weight.loss` after controlling for `motivation`, that person could build a model predicting

[4] The `car` package allows users to construct AVPs, but does so with R's native plots. This means one cannot leverage the advantages of `ggplot` (e.g., layering).

`weight.loss` from `motivation`, residualize that relationship, then show a beeswarm plot of the residuals for each type of therapy. This is what AVPs do (see Figure 16). These reduce cognitive load substantially, since users only need to interpret two dimensions.

`flexplot's` version of AVPs have a slightly different flavor. In my experience, AVPs can be confusing to lay audiences because the scale of the outcome variable has changed to be centered on zero; if one is expecting the outcome variable to range from 0 to 30, yet the graph shows scores from -15 to +15, this will violate users' intuitions, which will create obstacles to proper interpretation of graphics (Kosslyn, 2006). To counter this confusion, `flexplot` adds the mean back into the residuals so the Y-axis retains the original scale. The notation for `added.plot()` is similar to `flexplot`, though the vertical pipes aren't necessary.

There are multiple ways to specify the display of added variable plots. By default, `flexplot` will do is take the *last* variable entered (`therapy.type` in the first example below) and plot that on the X axis, while plotting the residuals of the model on the Y axis (i.e., the residuals of the model `weight.loss ~ motivation`). Another option is to specify either a number (indicating the ordinal position of the variable they wish to plot on the X axis) or the variable name in quotes (see the latter two examples). Other arguments can be passed to `added.plot()` as well (such as `alpha`, `sample`, `method`, etc.). A final option is to specify explicitly how the residuals are computed (using the `lm_formula` argument), separately from how the data are displayed (via the formula argument). This final method allows one to specify a multivariate plot, as in Figure 16.

```
added.plot(weight.loss ~ motivation + therapy.type, data = exercise_data)
```

```
added.plot(weight.loss ~ motivation + therapy.type, data = exercise_data, x=1)
```

```
added.plot(weight.loss ~ motivation + therapy.type, data = exercise_data, x="motivation")
```

```
added.plot(weight.loss~motivation + therapy.type, data=exercise_data,
           lm_formula = weight.loss~health)
```

Colors/Lines/Shapes

As shown in Figure 3, the second slot in the `flexplot` formula (`X2` in Figure 3) controls which variable is displayed as different colors/symbols/lines. Figure 17 shows two examples of this: one where a numeric predictor is on the X axis, and one where a categorical predictor is on the X-axis. When categorical variables are shown on the X axis, `flexplot` draws lines connecting the medians (or means).

```
a = flexplot(weight.loss ~ motivation + gender,
    data = exercise_data, se = F, alpha = .3)
b = flexplot(weight.loss ~ therapy.type + gender,
    data = exercise_data, se = F, alpha = .3) +
    theme(axis.text.x = element_text(angle=90, hjust=1, vjust=.2))
a + b
```

One limitation of colors/shapes/symbols is the increase in cognitive load. When plotting different symbols/colors/lines on the same graph, there is often a great deal of overlap, which makes it more difficult to pick out patterns. While research suggests we can conceptualize up to four unique elements (Kosslyn, 2006), in my experience, having a variable with more than two levels in the second slot of a `flexplot` equation becomes challenging to interpret, particularly when there are more than a handful of datapoints.

Paneling

An alternative (or additional) strategy for plotting multivariate data is paneling. As shown in Figure 3, the third and fourth slots control paneling in columns and rows, respectively. The panels follow many conventions developed by William Cleveland (1994), such as having values increase from left to right and bottom to top (just as they do on the X

and Y axis, respectively).

Figure 18 shows the same relationships in Figure 17, but with the second variable in panels instead. The bottom image also displays panels for three variables simultaneously. Paneled variables are easy to conceptualize with categorical variables, but what about numeric variables? These can still be graphed, but the values must be binned. Also notice that I have taken advantage of the fact that `flexplot` returns a `ggplot2` object that can be edited. In this case, I am both layering (modifying the behavior of the labels to prevent cutting them off) and modifying the `ggplot2` object itself (reducing the size of the points in the final graphic).[5] This modification is done using flexplot's `modify_points` function, which allows users to modify the shape, size, or color of points.

```
theme_update(axis.text.x = element_text(angle=90, hjust=1, vjust=.2))
a = flexplot(weight.loss ~ motivation | gender,
             data = exercise_data)
b = flexplot(weight.loss ~ therapy.type | gender,
             data = exercise_data)
c = flexplot(weight.loss ~ motivation |  gender + therapy.type,
             data = exercise_data) +
      ggplot2::facet_grid(therapy.type ~ gender,
               labeller = ggplot2::labeller(therapy.type = label_value))
#### edit point size
c = modify_points(c, size=.25)
```

[5] I also do not show the code where I actually plot the graphics. This required some advanced manipulation of the layout and I didn't want to detract from what `flexplot` is doing. However, interested readers are welcome to see the source code of this document at https://github.com/dustinfife/flexplot/tree/master/vignettes.

Binning

Within `flexplot`, any numeric predictor, with the exception of the variable in the first slot, will be binned into discrete categories. These bins will then be represented in panels (if the variable is in the third or fourth slot) or as colors/symbols/lines (if the variable is in the second slot). The user has the option of specifying the number of bins (e.g., 2 or 3), or the user can specify breakpoints at which to bin. When the user specifies bins, `flexplot` will attempt to have an equal number of datapoints in each bin. However, `flexplot` may be unable to bin into the specified number of bins. For example, if a user specifies four bins, it is possible the scores at the 50th and 75th percentile are the same. In these cases, `flexplot` will choose a smaller bin number, though it will report such to the user. `flexplot` defaults to three bins.

When specifying breakpoints, the panels can have different sample sizes in each bin. A user may wish to do this if these breakpoints are meaningful (e.g., when particular scores are clinically meaningful, such as Beck Depression Inventory scores above 29 are considered severly depressed).

Whether using breakpoints or bins, the user can also specify labels for the bins. Figure 19 shows three plots of the same variables: the first specifies two breakpoints, the second specifies breakpoints with labels, and the third just specifies the number of bins.

```
a = flexplot(weight.loss ~ motivation | satisfaction,
             data = exercise_data,
             breaks = list(satisfaction=c(3,7)))  +
  # change font size to eliminate overlap
    theme(axis.text.x =
            element_text(size = 12, angle = 90, hjust=0.95,vjust=0.5))
b = flexplot(weight.loss ~ motivation + satisfaction,
             data = exercise_data,
```

```
             breaks = list(satisfaction=c(3,7)),
             labels = list(satisfaction=c("low", "medium", "high")))
c = flexplot(weight.loss ~ motivation + satisfaction,
             data = exercise_data,
             bins = 2)
```

Ghost Lines

There are advantages and disadvantages to plotting a variable as a color/symbol/line versus panels. Panels reduce clutter, but make it harder to make comparisons because the eye has to travel further to make such comparisons. On the other hand, plotting in the same panel with different colors/lines/symbols means less visual distances to travel, but then there is too much clutter. One obvious solution is to stick with colors/symbols/lines and reduce transparency (or sample). However, there is a more innovative way of resolving this difficulty, by using something I call ghost lines. Ghost lines repeat the relationship from one panel to the other panels to make it easier to compare. Figure 20 demonstrates how to use ghost lines. By default, `flexplot` chooses the middle panel for odd numbers of panels, otherwise it chooses a panel close to the middle. The second line of code below specifies the referent panel by picking a value in the range of the referent panel. In Figure 20, the ghost lines makes it clear that the relationship between `motivation` and `weight.loss` is stronger both at low and high levels of `satisfaction`, but less so at medium levels.

```
# change font size to eliminate overlap
theme_update(axis.text.x =
             element_text(size = 14))
flexplot(weight.loss ~ motivation | satisfaction,
             data = exercise_data, method = "lm",
             bins = 3, ghost.line = "red")
```

General Strategy for Plotting Multivariate Relationships

It is easy for multivariate graphs to become unnecessarily complicated. For example, the top image in Figure 21 shows a graphic where all four `flexplot` slots are occupied. This is very difficult to interpret. To simplify things, we could utilize several strategies. (See Fife, 2021 for a more detailed review of these and other strategies). First, we can specify two bins instead of the default three. We can also remove confidence intervals, reduce the opacity of the data, and plot regression lines.[6] Also, I generally try to have only two levels for the variable in the second slot (in this case, Male versus Female). Finally, we could add ghost lines. The bottom image in Figure 21 utilizes all these strategies and simplifies the visual interpretation immensely.

Another strategy is to mentally block out all panels but the diagonal (the bottom-left to top-right). The diagonal reflects the influence of *both* variables as they increase (or decrease). In other words, they are a rough approximation of the average effect of the $X1/Y$ relationship as you increase (or decrease) the paneled variables. If there is a general pattern of the lines consistently getting steeper (or shallower) as they move up the diagonal, that indicates there may be a three-way interaction effect. In Figure 21, the lines seem pretty parallel going from bottom-left to top-right. The one thing that *does* seem to change is that the colored lines go from below the ghost line to above it (indicating a main effect of `health` and/or `satisfaction`). In other words, we may be safe to do AVPs. However, before doing so, it may be best to combine graphs with statistical modeling.

```
a = flexplot(weight.loss ~ motivation + gender | satisfaction + health,
    data = exercise_data)
b = flexplot(weight.loss ~ motivation + gender | satisfaction + health,
```

[6] These are great strategies to use *after* one has determined that the model generally fits the data. If the data are curvilinear, for example, it would not make sense to plot straight lines or make the datapoints overly transparent.

```
    data = exercise_data,
    method = "lm", se = F, bins = 2, ghost.line = "black", alpha = .2,
    ghost.reference = list(satisfaction = 0, health = 10, gender = "male"))
a + b + plot_layout(ncol = 1)
```

Combining Modeling and Visualizations

Graphs are great for conveying general trends and patterns. However, it can be difficult to make decisions based on graphics, particularly when the pattern is not striking. For example, in Figure 21, I don't feel entirely comfortable rejecting the idea that there are no interactions present in the model. Statistics, on the other hand, put visual patterns into concrete numbers that assist with statistical decision-making. Furthermore, it is easy to engage in confirmation bias when viewing a graphic. Statistics provide a much-needed reality check. As such, the two, statistics and graphs, ought to proceed hand in hand. Fortunately, `flexplot` was designed to complement statistical analysis (and vice versa). More specifically, `flexplot` has two additional visualization functions that simplify modeling, as well as two functions dedicated to statistical analysis.

The `visualize()` Function

As I've mentioned repeatedly, one of the primary purposes of `flexplot` is to provide visualization for statistical models. The `visualize()` function is designed to do exactly that. Much like `summary()`, or `coef()`, `visualize()` is an R method within `flexplot` that can be applied to diverse sorts of models, including `lmer`, `lm`, and `glm`. `visualize()` attempts to generate a graphic that matches the formula used in a fitted model. Not only will `visualize()` plot a graphic to match the analysis, but it will also show diagnostic plots. For example, if we were to fit an ANCOVA model, we could graph it as follows:

```
model = lm(weight.loss ~ motivation + therapy.type,
           data = exercise_data)
```

```
visualize(model)
```

For multivariate data, `visualize` will create a plot that will use panels and/or colors/symbols/lines. It will also generate diagnostic plots (histogram of the residuals, residual dependence plots, and S-L plots). The user can specify *just* a plot of the model (`visualize(model1, "model")`), or *just* a plot of the diagnostics (`visualize(model1, "residuals")`). Additionally, visualize can take `flexplot` arguments and even a `flexplot` formula.

```
model = lm(weight.loss ~ motivation + therapy.type, data = exercise_data)
visualize(model, formula = weight.loss ~ motivation | therapy.type,
          ghost.line = "gray", method = "lm", plot = "model")
```

Figure 24 shows the `visualize()` function for a mixed model. With mixed models, `visualize()` randomly samples from the random effects (`Subject` in this case) and plots that as a variable in the graphic, and its location on the graph depends on whether the user specifies `formula`. If they do not, `visualize()` will default to placing it in the second slot (different lines/colors/shapes). This allows the user to graph a subset of the subjects in a mixed model to ensure the model chosen is appropriate.

```
require(lme4)
data(math)
model = lmer(MathAch ~ Sex + SES + (SES|School), data = math)
visualize(model,
  plot = "model",
  formula = MathAch ~  SES + School| Sex,
  sample = 3)
```

The `compare.fits()` Function

Very often in statistical modeling, we are interested in comparing two models (Rodgers, 2010), such as one with and one without an interaction term. There are many statistics available that allow easy comparison between models, such as the AIC, BIC, Bayes Factor, R^2, p-values, etc. However, it is again important to *see* how the two models differ in terms of fit. On multiple occasions, I have found various statistics show preference for one model, yet the graphs show the two models differ in only trivial ways.

That is where `compare.fits()` comes in. It is simply a wrapper for the `predict()` function, combined with the graphing capabilities of `flexplot`. More specifically, `compare.fits()` will overlay the fit of both models onto the raw data. For example, Figure 25 shows the fit of two different models, one that includes an interaction and the other that does not. The arguments are very similar to `flexplot`, but with the addition of the model objects. Likewise, `compare.fits()` takes many of the same arguments. In this example, I've overlaid a black ghost line. Notice that the two lines (from `lm` and `interaction`) generate very similar predictions across the range of data, suggesting that a main effects model may be sufficient.

```
model.me = lm(weight.loss ~ motivation + therapy.type, data = exercise_data)
model.int = lm(weight.loss ~ motivation * therapy.type, data = exercise_data)
compare.fits(weight.loss ~ motivation | therapy.type,
             data = exercise_data, model.me, model.int, ghost.line = "black")+
  # get rid of variable name in panels (just put in the level)
      ggplot2::facet_grid( ~ therapy.type,
          labeller = ggplot2::labeller(therapy.type = label_value))
```

Functions Devoted to Estimation

The `flexplot` package specializes in visualization, providing easy-to-use tools for graphical data analysis. However, it also has a small collection of non-visual functions that

can be used hand-in hand with graphs. These functions include the `estimates()` method and `model.comparison()`.

The `estimates()` Method

The `estimates()` method was designed to report paremeter estimates and effect sizes. Much like `flexplot`, many of the decisions are made in the background. And like `visualize()`, `estimates()` takes a fitted object as input. `estimates()` will then determine which estimates are most appropriate. For grouping variables, `estimates()` will report means, mean differences, and cohen's d, as well as 95% confidence intervals. For numeric variables, `estimates()` will report the intercept, slopes, and standardized slopes, also with corresponding confidence intervals. Additionally, it will report the model R^2, as well as the semi-partial R^2 associated with each effect in the model, except when there are interactions in the model. (With interactions present, it does not make sense to interpret main effects, which is why `estimates()` only reports the semi-partial for the interaction effect).

```
estimates(model.int)
```

```
## Model R squared:
## 0.222 (0.12, 0.32)
##
## Semi-Partial R squared:
##              motivation            therapy.type motivation:therapy.type
##                   0.125                   0.091                   0.006
##
## Estimates for Factors:
##      variables  levels estimate lower upper
## 1 therapy.type control     4.09  2.81  5.38
## 2                  beh     7.86  6.74  8.99
## 3                  cog     7.74  6.46  9.02
```

```
##
##
## Mean Differences:
##      variables  comparison difference lower upper cohens.d
## 1 therapy.type beh-control       3.77  0.71  6.84     0.75
## 2              cog-control       3.65  0.77  6.54     0.72
## 3                  cog-beh      -0.12 -2.99  2.75    -0.02
##
##
## Estimates for Numeric Variables =
##     variables estimate  lower upper std.estimate std.lower std.upper
## 1 (Intercept)    -5.77 -11.84  0.31         0.00      0.00      0.00
## 2  motivation     0.18   0.07  0.29         0.35     -0.33      1.03
```

The estimates shown support the conclusions gleaned from Figures 22-23, as well as Figure 25, namely that the addition of the interaction likely is not improving the fit enough to consider keeping.

The `model.comparison()` Function

The final function I will mention is the `model.comparison()` function. This is similar to the `anova()` function in base `R`, but it includes additional estimates, including AIC, BIC, and the BIC-derived Bayes Factor. Additionally, `model.comparison()` works on both nested and non-nested functions. When used on non-nested functions, it will only compute AIC, BIC, and Bayes Factor. Also, the `model.comparison()` function will report the quantiles of the *differences* in prediction. The `model.comparison()` below compares the main effects and the interaction model. The last reported numbers indicate that the maximum difference in prediction between the two models is only about 1.76 pounds, while the median difference is only about a quarter of a pound, suggesting the predictions are quite similar. Also, all statistics (AIC, BIC, Bayes Factor, p-value, and probably R^2)

support the more simplified main effects model.

```
model.comparison(model.int, model.me)
```

```
## $statistics
##                aic      bic bayes.factor     p   rsq
## model.int 1223.041 1246.129        0.010 0.487 0.222
## model.me  1220.523 1237.015       95.294       0.217
##
## $predicted_differences
##    0%   25%   50%   75%  100%
## 0.003 0.087 0.252 0.395 1.757
```

Discussion

There's little doubt that psychology is undergoing a methodological revolution. Scholars are pushing with increased fervor for preregistration, larger sample sizes, open science practices, etc. This is a very good thing, but inevitably insufficient. Data can often be deceptive, especially when tortured into submission by standard analyses. Alongside emerging post-replication crisis practices, I advocate for greater use of visualizations. Graphs are the best, and sometimes the only way to allow data to speak for itself.

Traditionally, producing graphs has been a cumbersome and extraneous procedure, quite divorced from the actual analyses. In this paper, I have introduced `flexplot`. `flexplot` aims to automate much of the graphical analysis that accompany data analysis. These graphs, which are based on scientifically-derived principles of visual perception, promise to promote sound data analysis and enhance data decision-making.

To use `flexplot`, users can download and install the R package through github, at www.github.com/dustinfife/flexplot. Alternatively, a point-and-click version of flexplot can be accessed via JASP (through the visual modeling module) or through Jamovi (through the

flexplot module). For additional information on using flexplot, visit the following YouTube playlist: https://bit.ly/2Qn4yoi

References

Abbott, G. R., Do, M., & Byrne, L. K. (2012). Diminished subjective wellbeing in schizotypy is more than just negative affect. *Personality and Individual Differences*, *52*(8), 914–918. https://doi.org/10.1016/j.paid.2012.01.018

Anscombe, F. J. (1973). Graphs in statistical analysis. *The American Statistician*, *27*(1), 17–21.

Breheny, P., & Burchett, W. (2017). Visualization of Regression Models Using visreg. *The R Journal*, 56–71. https://doi.org/10.32614/rj-2017-046

Camargo, K., & Grant, R. (2015). Public health, science, and policy debate: being right is not enough. *American Journal of Public Health*, *105*(2), 232–235. https://doi.org/10.2105/AJPH.2014.302241

Cleveland, W. S. (1994). Coplots, nonparametric regression, and conditionally parametric fits. *Lecture Notes-Monograph Series*, 21–36.

Cohen, J. (1994). The earth is round ($p < .05$). *American Psychologist*, *49*(12), 997–1003. https://doi.org/10.1037/0003-066X.49.12.997

Correll, M. A. (2015). *Visual Statistics* (Doctoral Dissertation). University of Wisconsin-Madison.

Eklund, A. (2012). Beeswarm: The bee swarm plot, an alternative to stripchart. *R Package Version 0.1*, *5*.

Fife, D. A. (2020). The Eight Steps of Data Analysis: A Graphical Framework to Promote Sound Statistical Analysis. *Perspectives on Psychological Science*, *15*. https://doi.org/https://doi.org/10.1177/1745691620917333

Fife, D. A. (2021). Visual partitioning for multivariate models: An approach for identifying and visualizing complex multivariate dataset. *PsyArxiv*. https://doi.org/10.31234/osf.io/avu2n

Fife, D. A., Longo, G., Correll, M., & Tremoulet, P. (2020). A graph for every analysis: Mapping visuals onto common analyses. *Behavioral Research Methods*.

Fife, D. A., Tremoulet, P., & Longo, G. (2019). *Developing and Empirically Validating Flexplot: A Tool for Mapping Statistical Analyses into Graphical Presentation*. Chicago, IL: American Psychological Association.

Gigerenzer, G. (2004). Mindless statistics. *The Journal of Socio-Economics*, *33*(5), 587–606. https://doi.org/10.1016/J.SOCEC.2004.09.033

Hansen, C. D., Chen, M., Johnson, C. R., Kaufman, A. E., & Hagen, H. (2014). *Scientific Visualization: Uncertainty, Multifield, Biomedical, and Scalable Visualization*. London: Springer. Retrieved from http://www.springer.com/series/4562

Healy, K., & Moody, J. (2014). Data Visualization in Sociology. *Annual Review of Sociology*, *40*(1), 105–128. https://doi.org/10.1146/ANNUREV-SOC-071312-145551

Hoekstra, R., Kiers, H., & Johnson, A. (2012). Are Assumptions of Well-Known Statistical Techniques Checked, and Why Not? *Frontiers in Psychology*, *3*(137). https://doi.org/10.3389/fpsyg.2012.00137

Inbar, O., Tractinsky, N., & Meyer, J. (2007). Minimalism in information visualization: Attitudes towards maximizing the data-ink ratio. In *ECCE* (Vol. 7, pp. 185–188).

Kosslyn, S. M. (2006). *Graph design for eye and mind*. New York, NY: Oxford University Press.

Levine, S. S. (2018). Show us your data: Connect the dots, improve science. *Management and Organization Review*, *14*(2), 433–437. https://doi.org/10.1017/mor.2018.19

Marieb, E. N. (1992). Human anatomy and physiology. Redwood city. *CA: Benjamin/Cummings Publishing Company, Ine*, *1*(992), 306–307.

Otten, J. J., Cheng, K., & Drewnowski, A. (2015). Infographics And Public Policy: Using Data Visualization To Convey Complex Information. *Health Affairs*, *34*(11), 1901–1907. https://doi.org/10.1377/hlthaff.2015.0642

Pandey, A. V., Rall, K., Satterthwaite, M. L., Nov, O., & Bertini, E. (2015). How deceptive are deceptive visualizations?: An empirical analysis of common distortion techniques. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems* (pp. 1469–1478). ACM.

Pastore, M., Lionetti, F., & Altoè, G. (2017). When one shape does not fit all: A commentary essay on the use of graphs in psychological research. Frontiers Media S.A. https://doi.org/10.3389/fpsyg.2017.01666

Ripley, B., Venables, B., Bates, D. M., Hornik, K., Gebhardt, A., Firth, D., & Ripley, M. B. (2013). Package 'mass.' *Cran R*, *538*.

Rodgers, J. L. (2010). The epistemology of mathematical and statistical modeling: a quiet methodological revolution. *The American Psychologist*, *65*(1), 1–12. https://doi.org/10.1037/a0018326

Semetko, H. A., & Scammell, M. (2012). *The SAGE handbook of political*

communication. Sage Publications.

Tay, L., Parrigon, S., Huang, Q., & LeBreton, J. M. (2016). Graphical Descriptives: A Way to Improve Data Transparency and Methodological Rigor in Psychology. *Perspectives on Psychological Science*, *11*(5), 692–701. https://doi.org/10.1177/1745691616663875

Tufte, E. R. (2001). *The visual display of quantitative information*. Cheshire, CT: Graphics press.

Tukey, J. W., & Tukey, P. A. (1990). *Strips Displaying Empirical Distributions: I. Textured Dot Strips*. Bellcore.

Tyron, W. W. (1998). The inscrutable null hypothesis. *American Psychologist*, *53*(7), 796–796. https://doi.org/10.1037/0003-066X.53.7.796.b

Umanath, N. S., & Vessey, I. (1994). Multiattribute data presentation and human judgment: A cognitive fit perspective. *Decision Sciences*, *25*(5-6), 795–824.

Vessey, I. (1991). Cognitive fit: A theory-based analysis of the graphs versus tables literature. *Decision Sciences*, *22*(2), 219–240.

Wainer, H. (2010). Prelude. In J. Berkson (Ed.), *Semiology of graphics: Diagrams, networks, maps* (2nd ed., pp. ix–x). Redlands, CA: ESRI Press.

Weissgerber, T. L., Milic, N. M., Winham, S. J., & Garovic, V. D. (2015). Beyond Bar and Line Graphs: Time for a New Data Presentation Paradigm. *PLoS Biology*, *13*(4). https://doi.org/10.1371/journal.pbio.1002128

Wickham, H. (2010). A Layered Grammar of Graphics. *Journal of Computational and Graphical Statistics*, *19*(1), 3–28. https://doi.org/10.1198/jcgs.2009.07098

Wilkinson, L. (1999). Dot plots. *The American Statistician*, *53*(3), 276–281.

Wilkinson, L., & Task Force on Statistical Inference. (1999). Statistical Methods in Psychology Journals: Guidelines and Explanations. *American Psychologist*, *54*(8), 594–601.

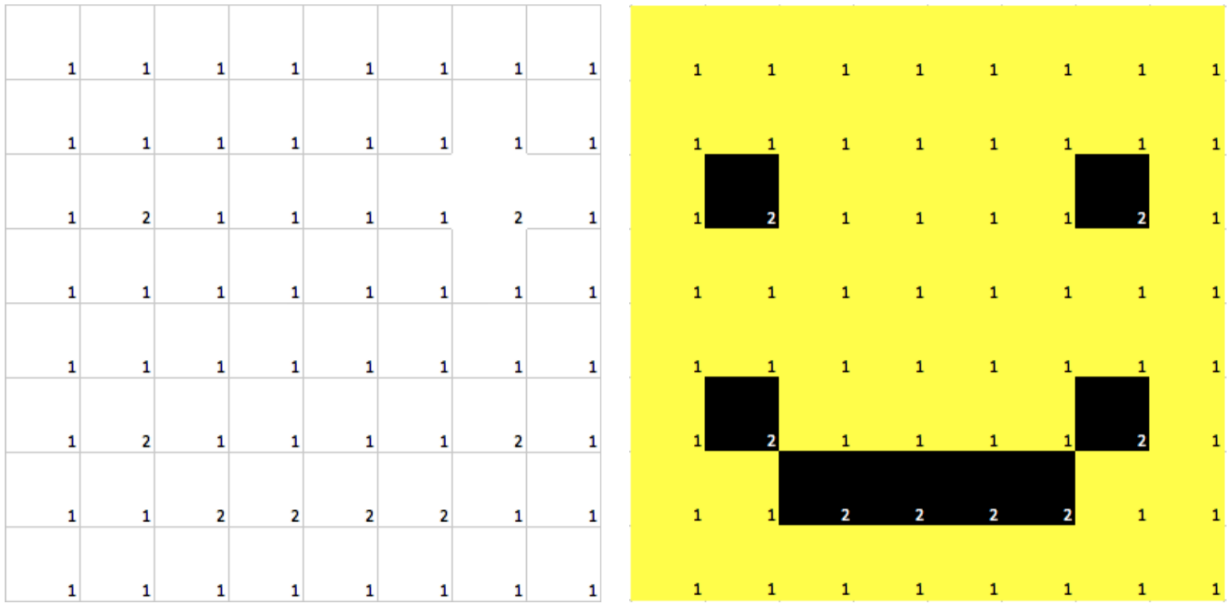

Figure 1. A table of numbers on the left, and a color-coded table on the right, where the 2's have been highlighted in yellow. With the color, a pattern emerges that was not easy to see without the graphic. Figure used with permission from Correll (2015).

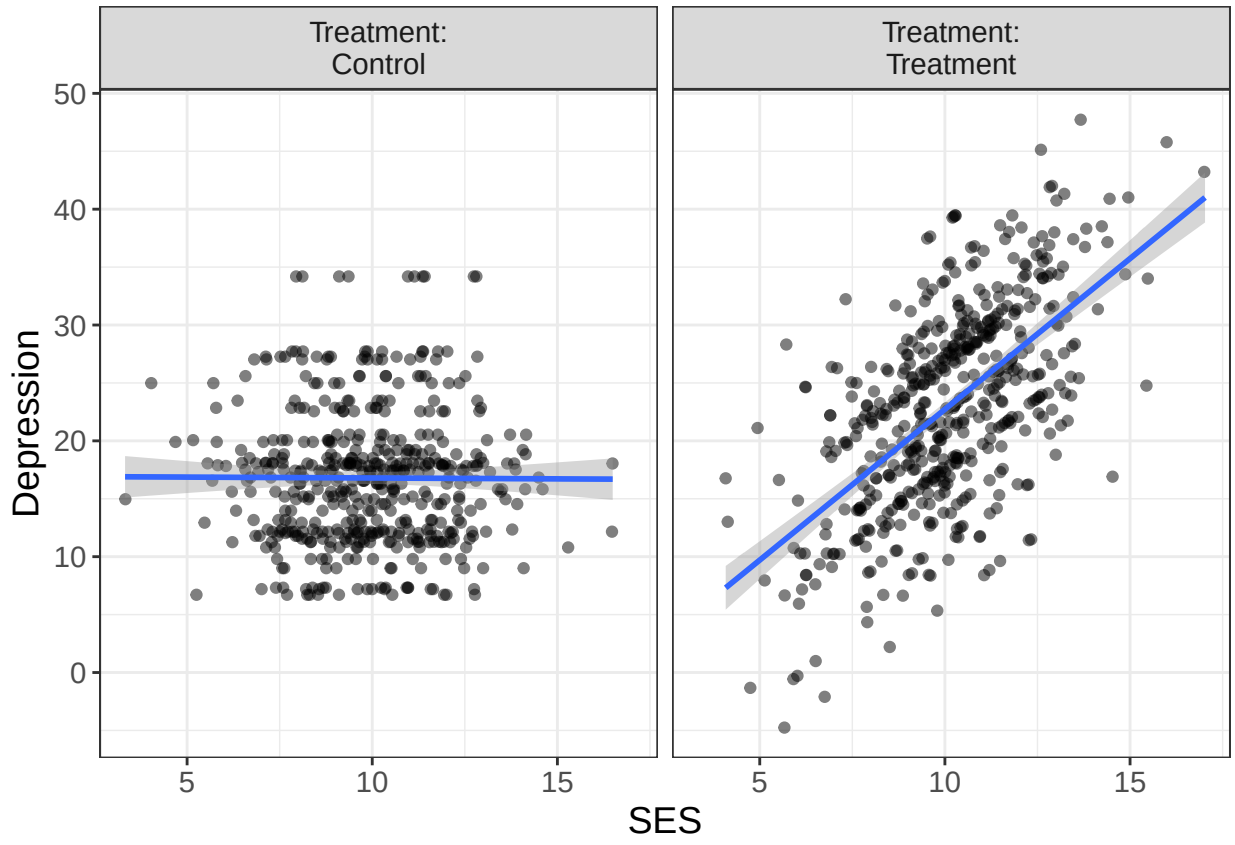

Figure 2. Simulated relationship between `Treatment`, `Depression`, and `SES`. The graph makes it clear there's an interaction effect, which would be masked if one simply computed a main effects model.

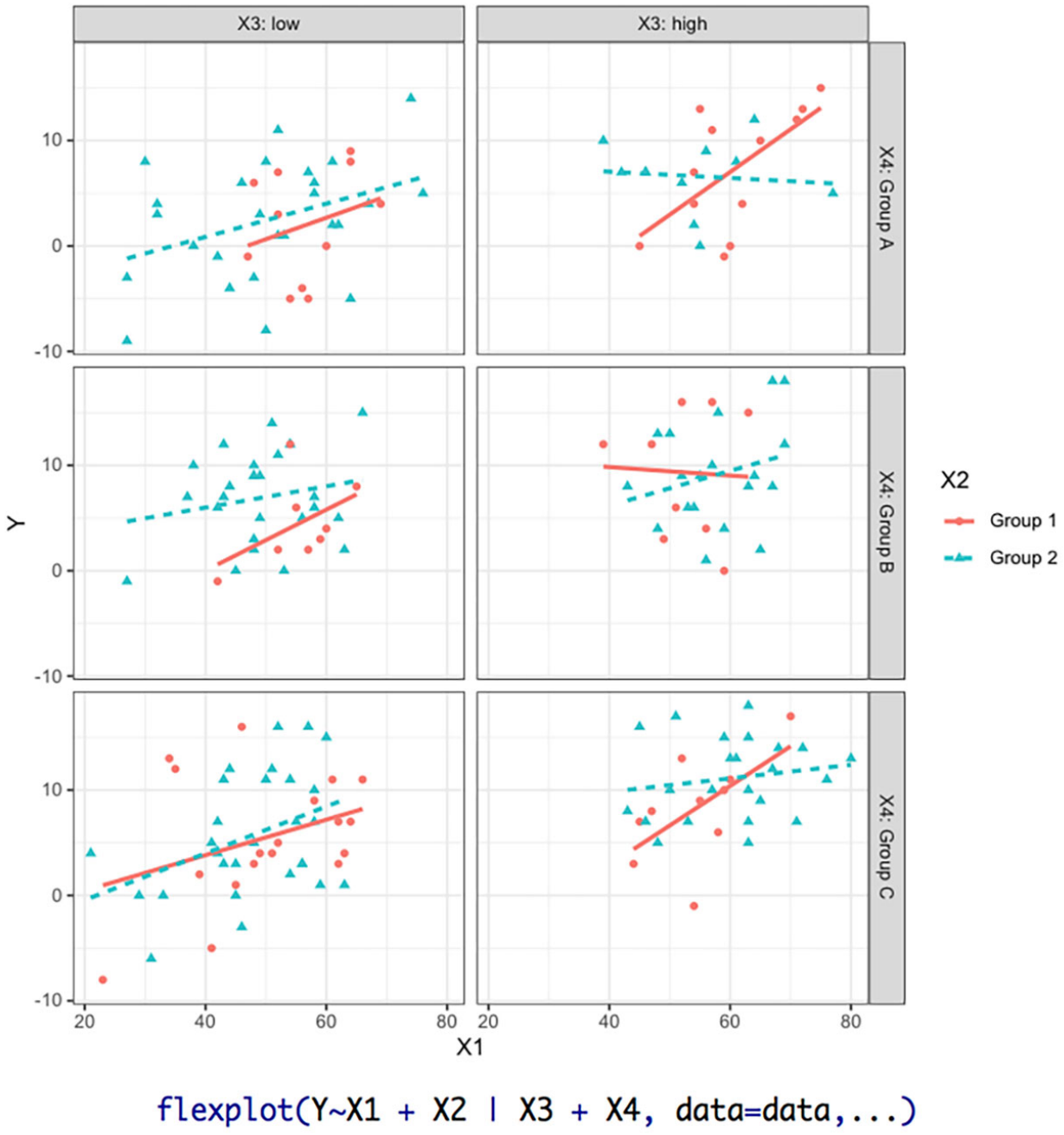

Figure 3. A diagram showing how elements of Flexplot's graphics are represented in a plot. $X1$ is shown on the X axis, $X2$ shows up as different colors/symbols/lines, $X3$ panels in columns, and $X4$ panels in rows.

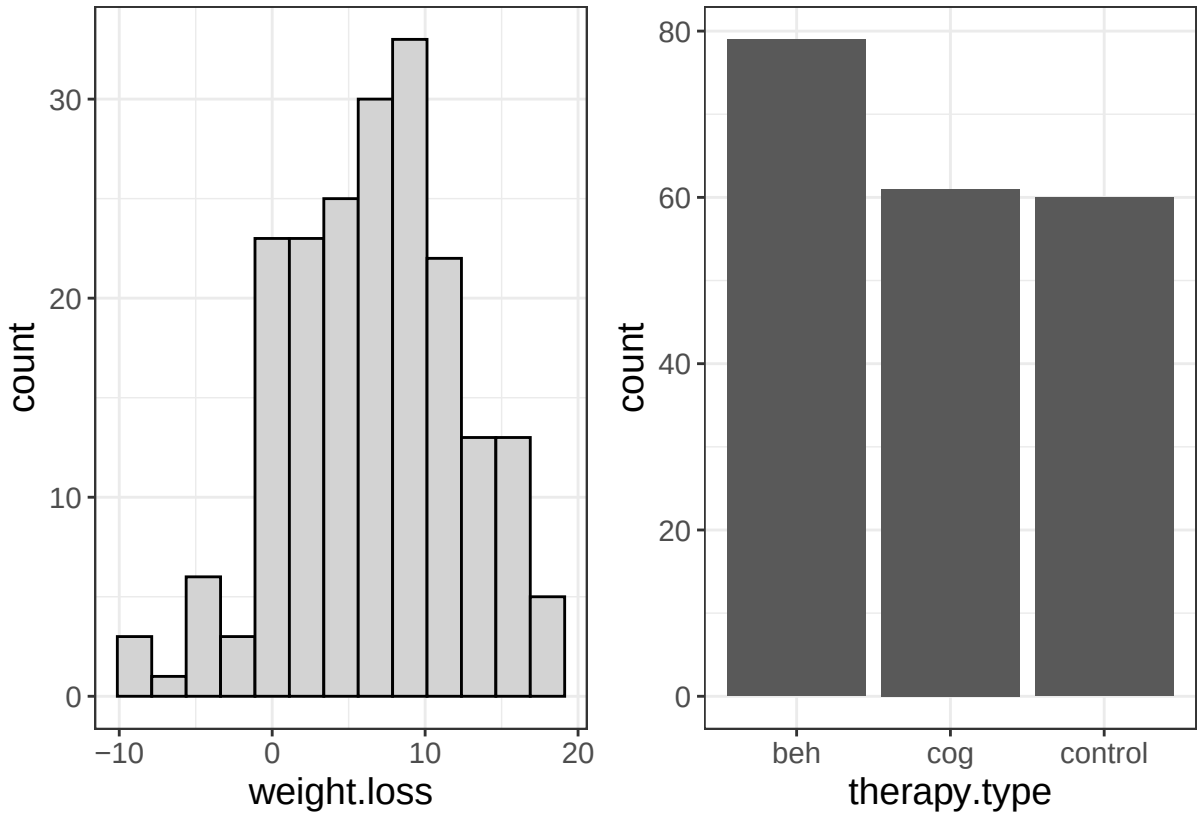

Figure 4. A histogram (left) and barchart (right) produced within `flexplot` in R.

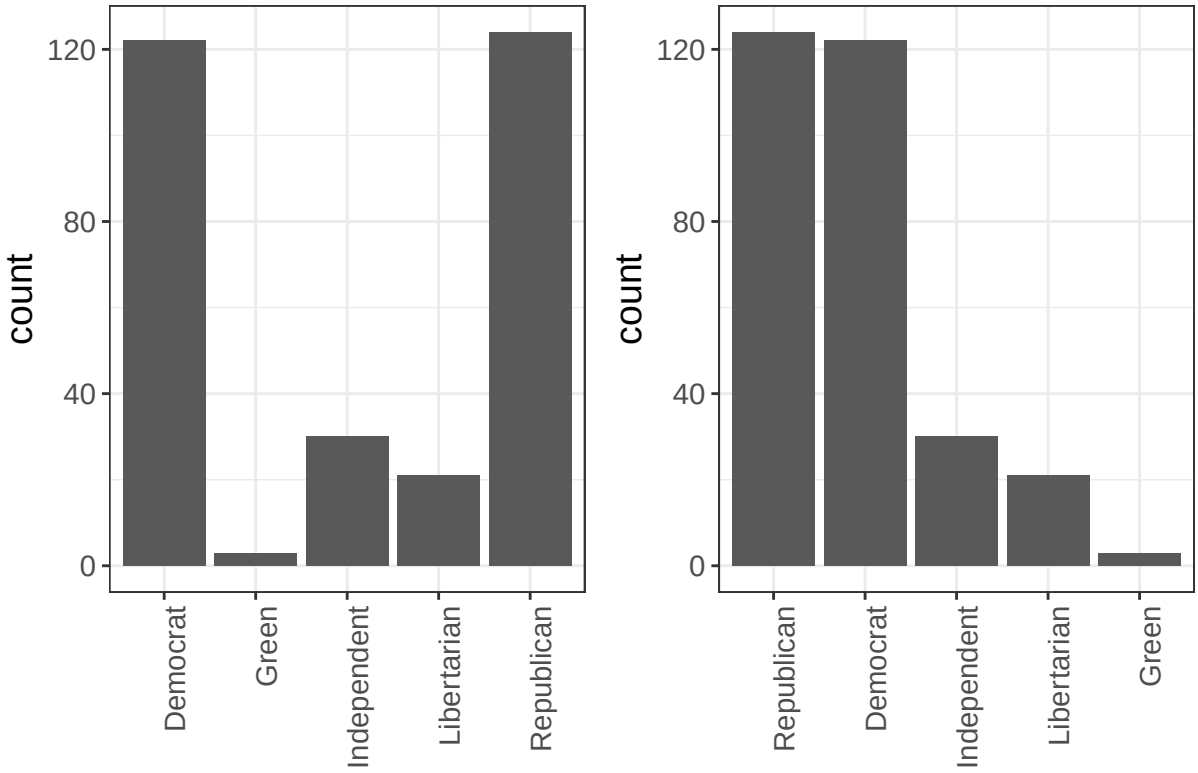

Figure 5. Two displays of the same data. In the left image, categories are sorted alphabetically (which is the default of `ggplot2`). In `flexplot` the categories are sorted by sample size, which enables better chunking of information.

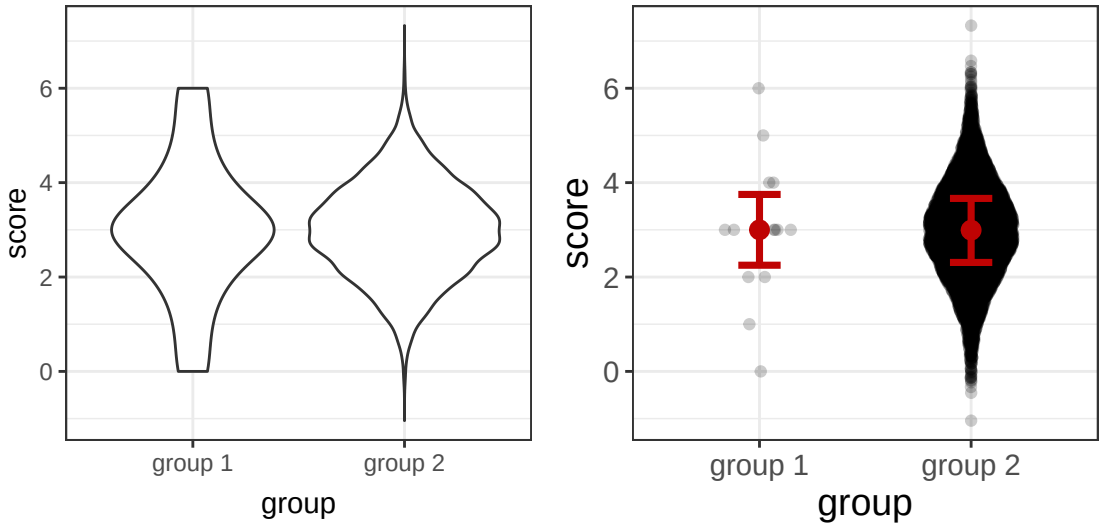

Figure 6. Violin plots with 15,000 versus 15 datapoints. The outlines look the same in the left image, but the right image overlays the raw data, which makes the differing sample sizes much more apparent.

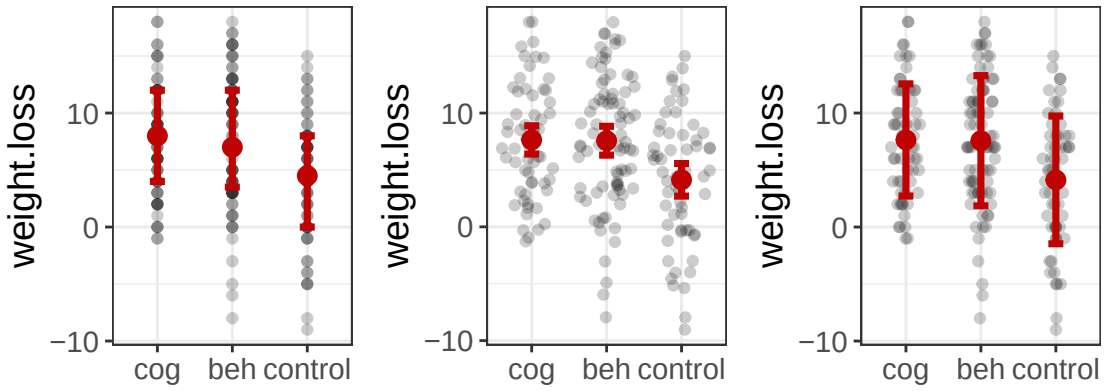

Figure 7. Dot plots with the interquartile range and no jittering (left), beeswarm plot with mean + standard errors and jittering on X and Y (middle), and beeswarm plot with mean + standard deviation with jittering on only X (right).

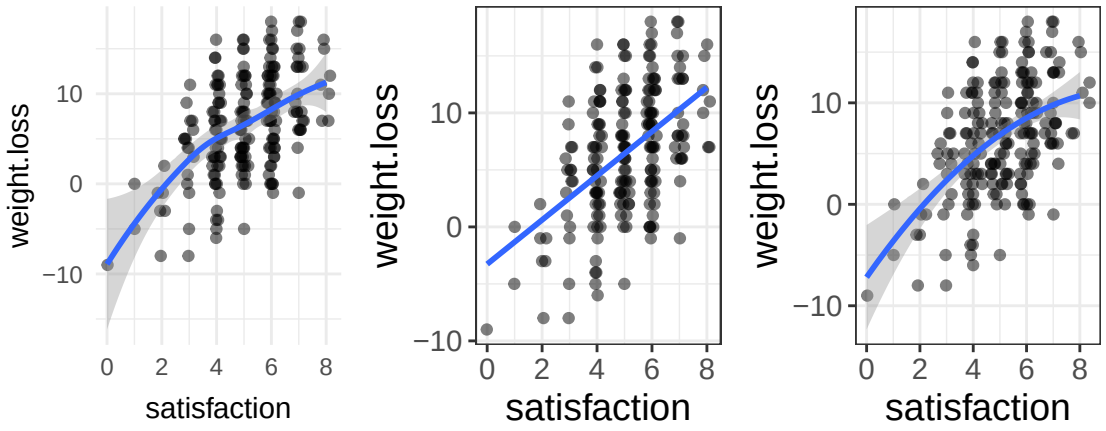

Figure 8. Scatterplot with different options of fit: loess (default), lm (regression), and quadratic Also, the data in the far right plot has been jittered.

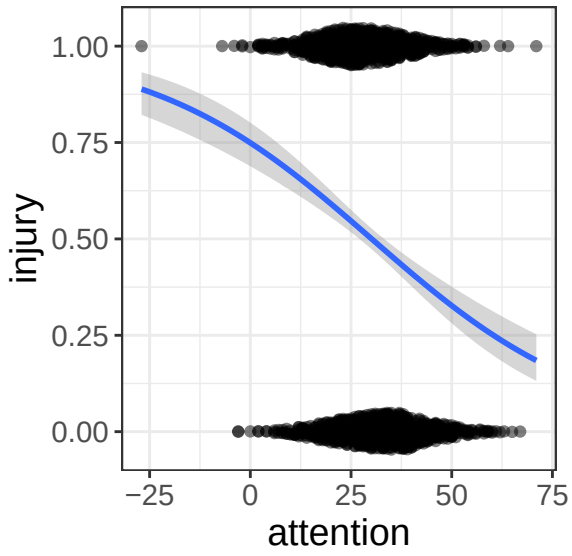

Figure 9. Example of a logistic plot in the `flexplot` package.

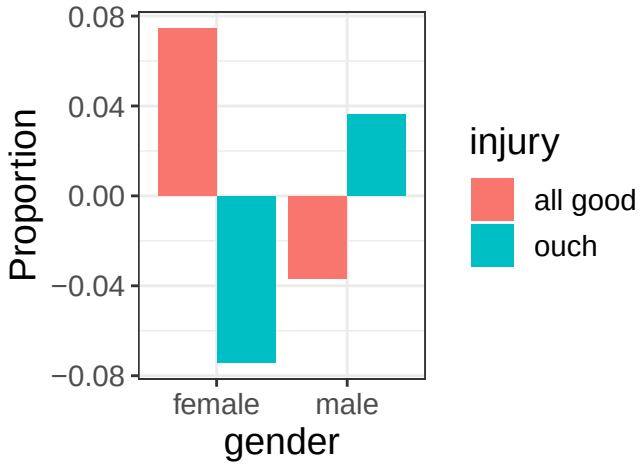

Figure 10. Example of an association plot for categorical predictor/categorical outcome.

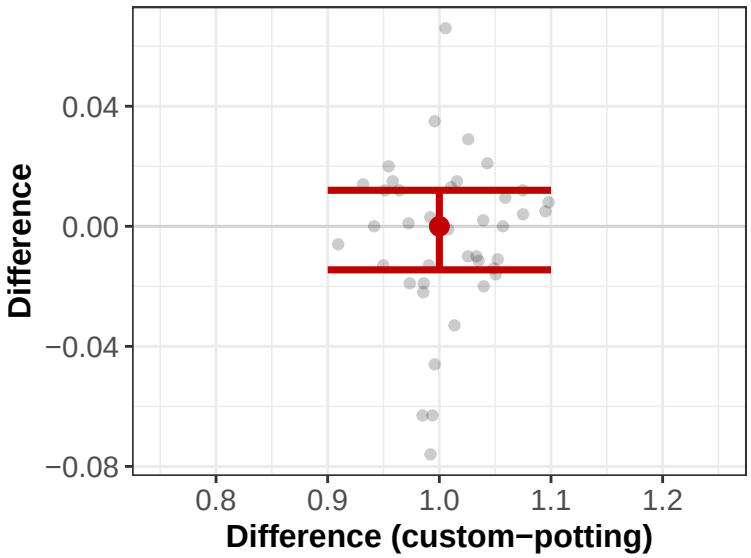

Figure 11. A plot of repeated measures data.

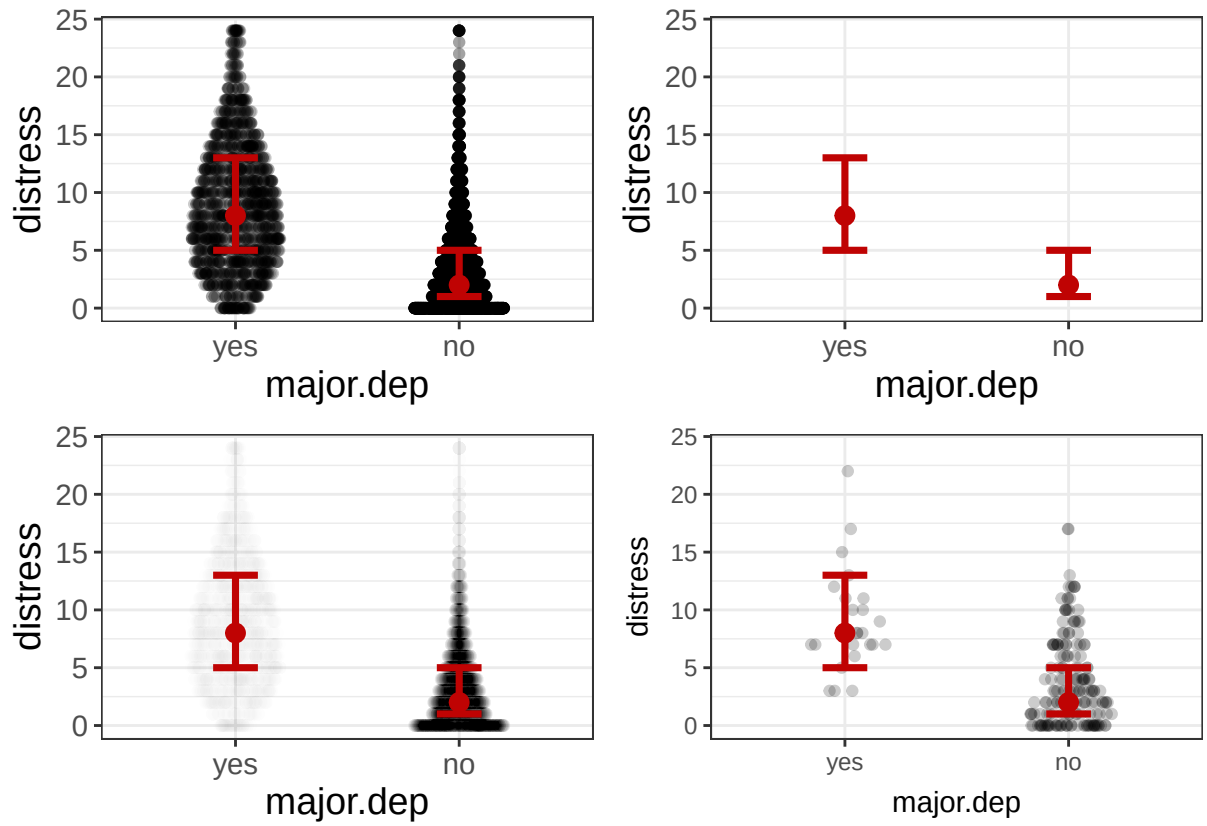

Figure 12. Four graphics showing different ways to handle overlapping datapoints. The top-left image does nothing. The top-right omits raw data. The bottom-left reduces the opacity of the points. The bottom-right samples datapoints.

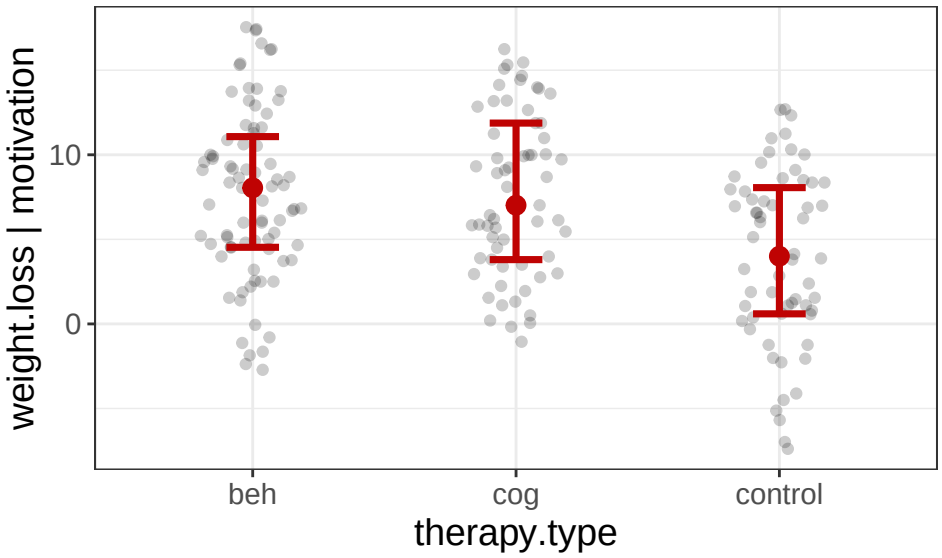

Figure 13. An Added Variable Plot (AVP) showing the relationship between `therapy.type` and `weight.loss`, after controlling for `motivation`.

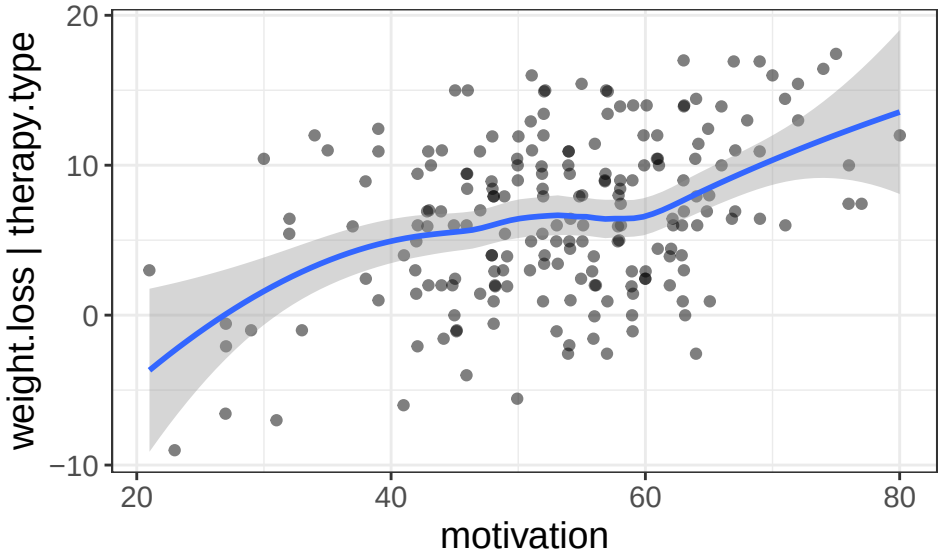

Figure 14. An Added Variable Plot (AVP) showing the relationship between `therapy.type` and `weight.loss`, after controlling for `motivation`.

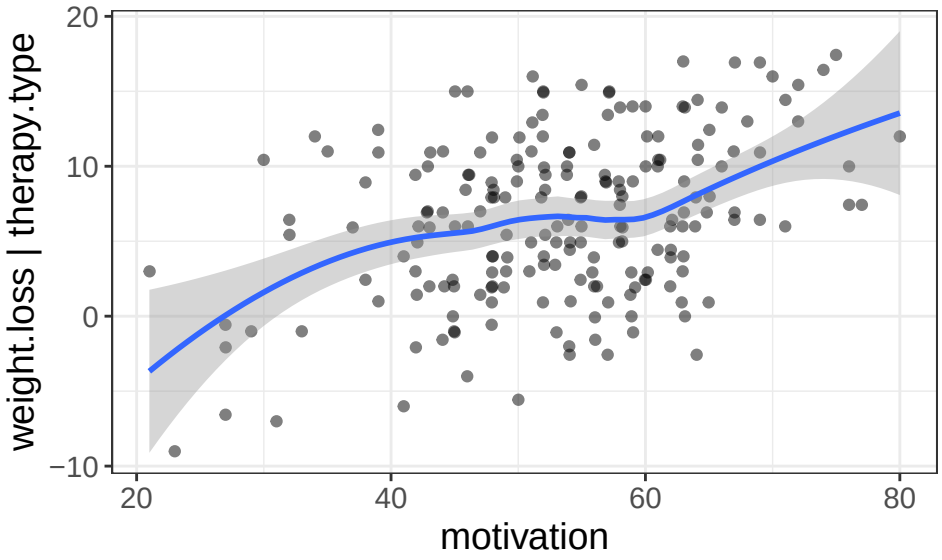

Figure 15. An Added Variable Plot (AVP) showing the relationship between `therapy.type` and `weight.loss`, after controlling for `motivation`.

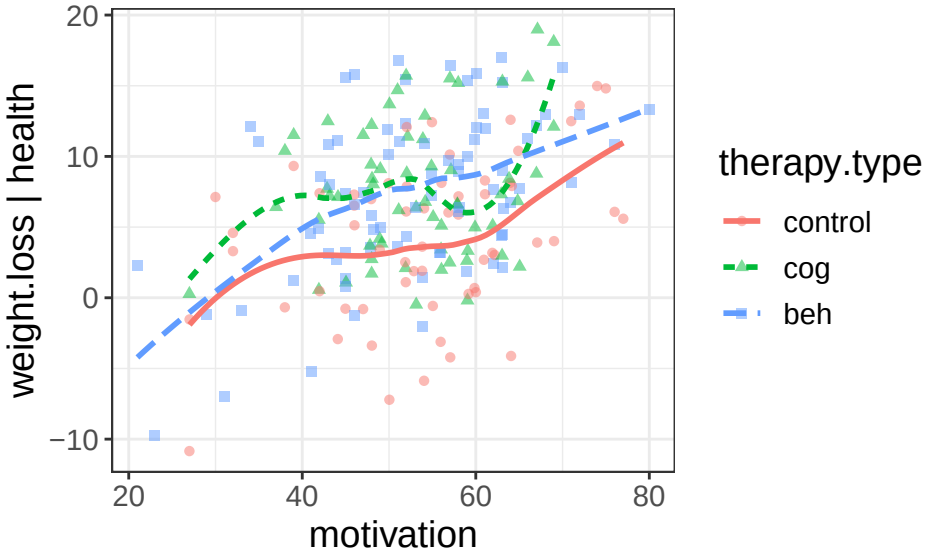

Figure 16. An Added Variable Plot (AVP) showing the relationship between `therapy.type` and `weight.loss`, after controlling for `motivation`.

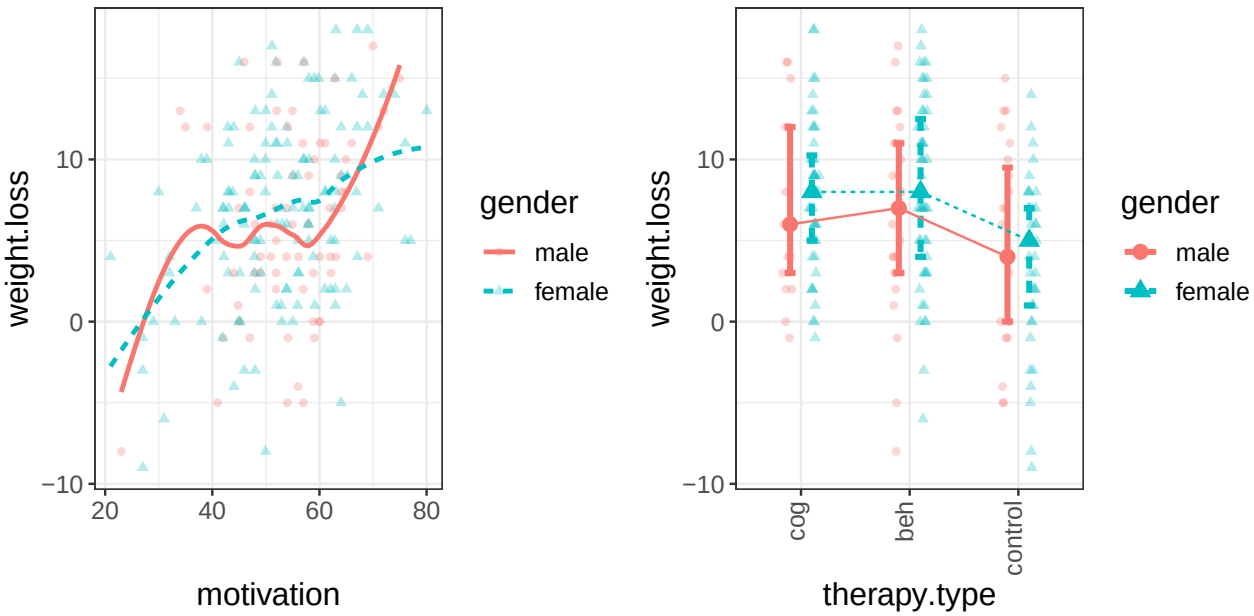

Figure 17. Two multivariate graphs illustrating how the second slot in a `flexplot` formula controls the graph. The left image demonstrates what happens to the second slot variable ($X2$) when a numeric predictor is on the X-axis, while the right image demonstrates what happens to the second slot variable when a categorical predictor is on the X-axis.

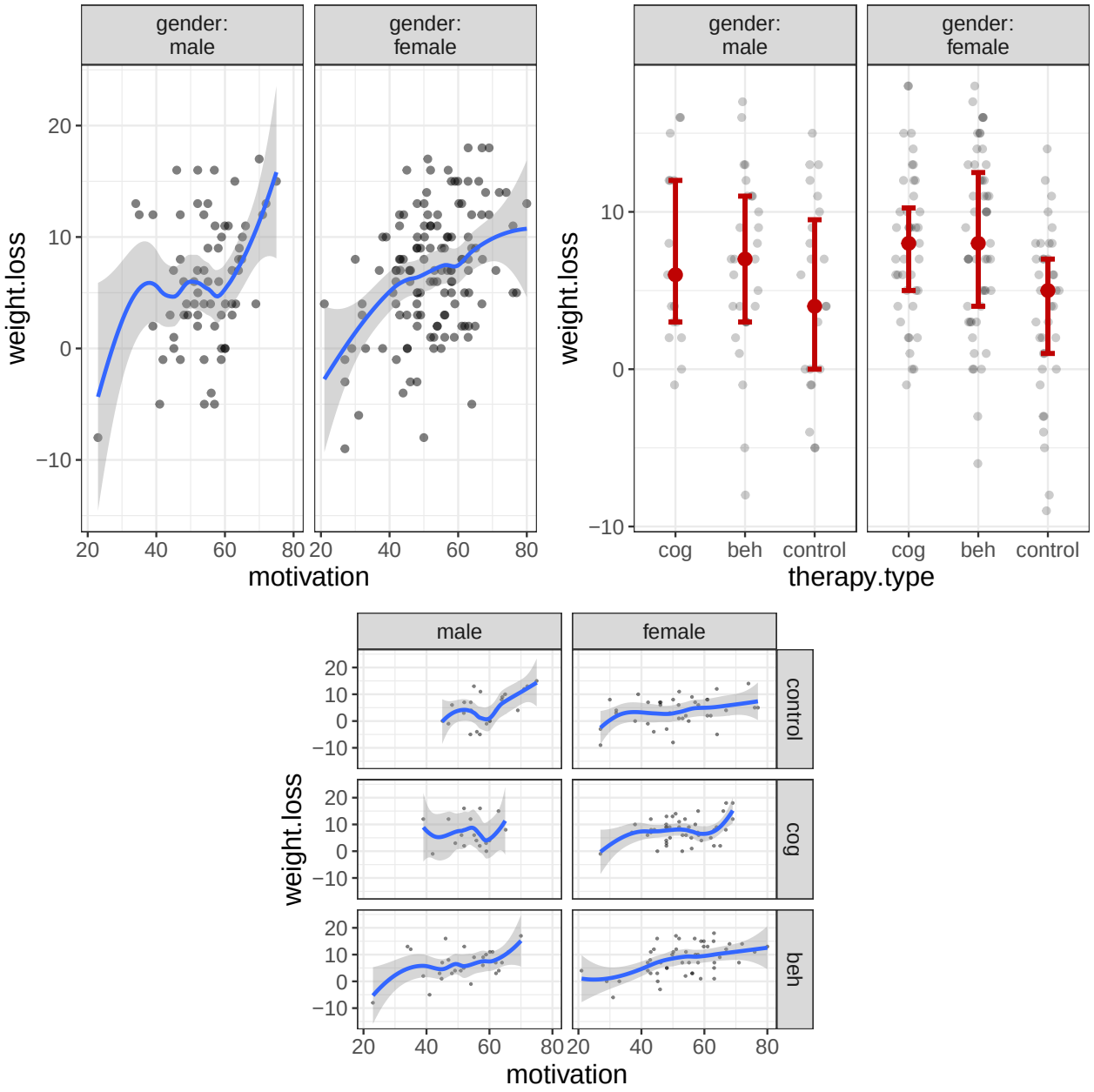

Figure 18. A multivariate plot where `therapy.type` and `gender` are now shown in panels.

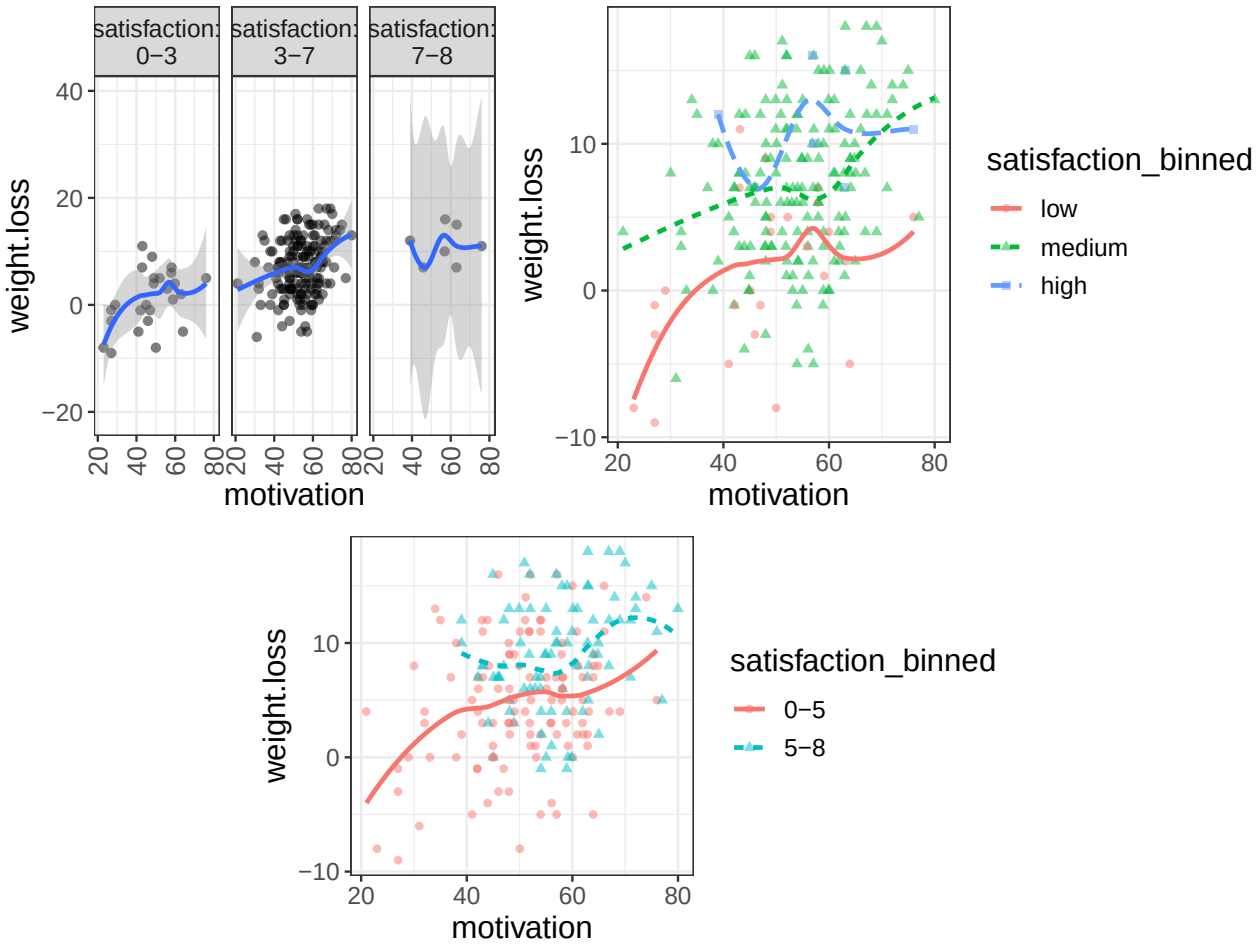

Figure 19. Plots reflecting choices of different break points (left-most plot), labels (right plot), and bins (bottom plot).

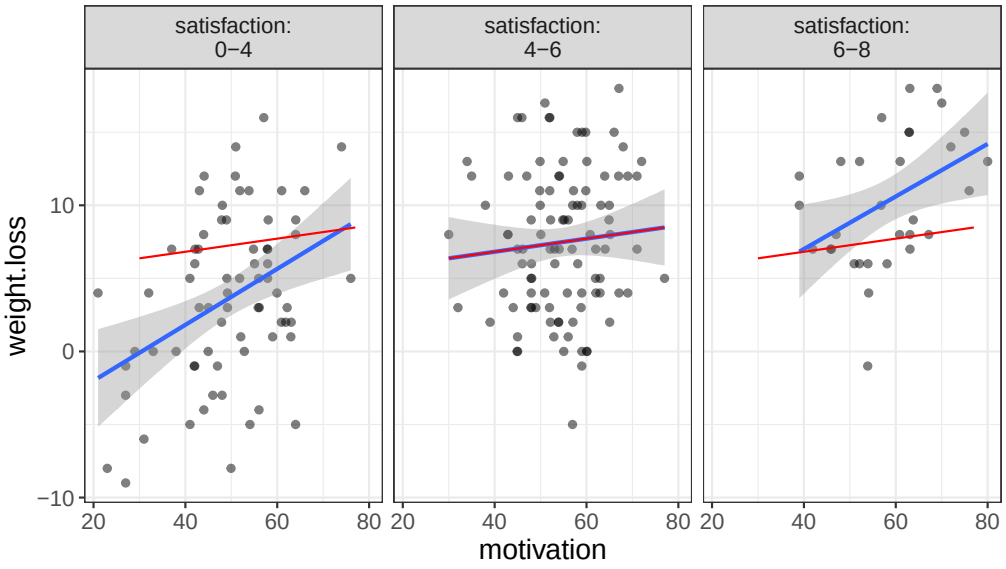

Figure 20. Ghost lines repeat the pattern from one panel to the others, making it easier to compare across panels. In this case, the line from the middle panel (Satisfaction = 4-6) is repeated in red across the other panels.

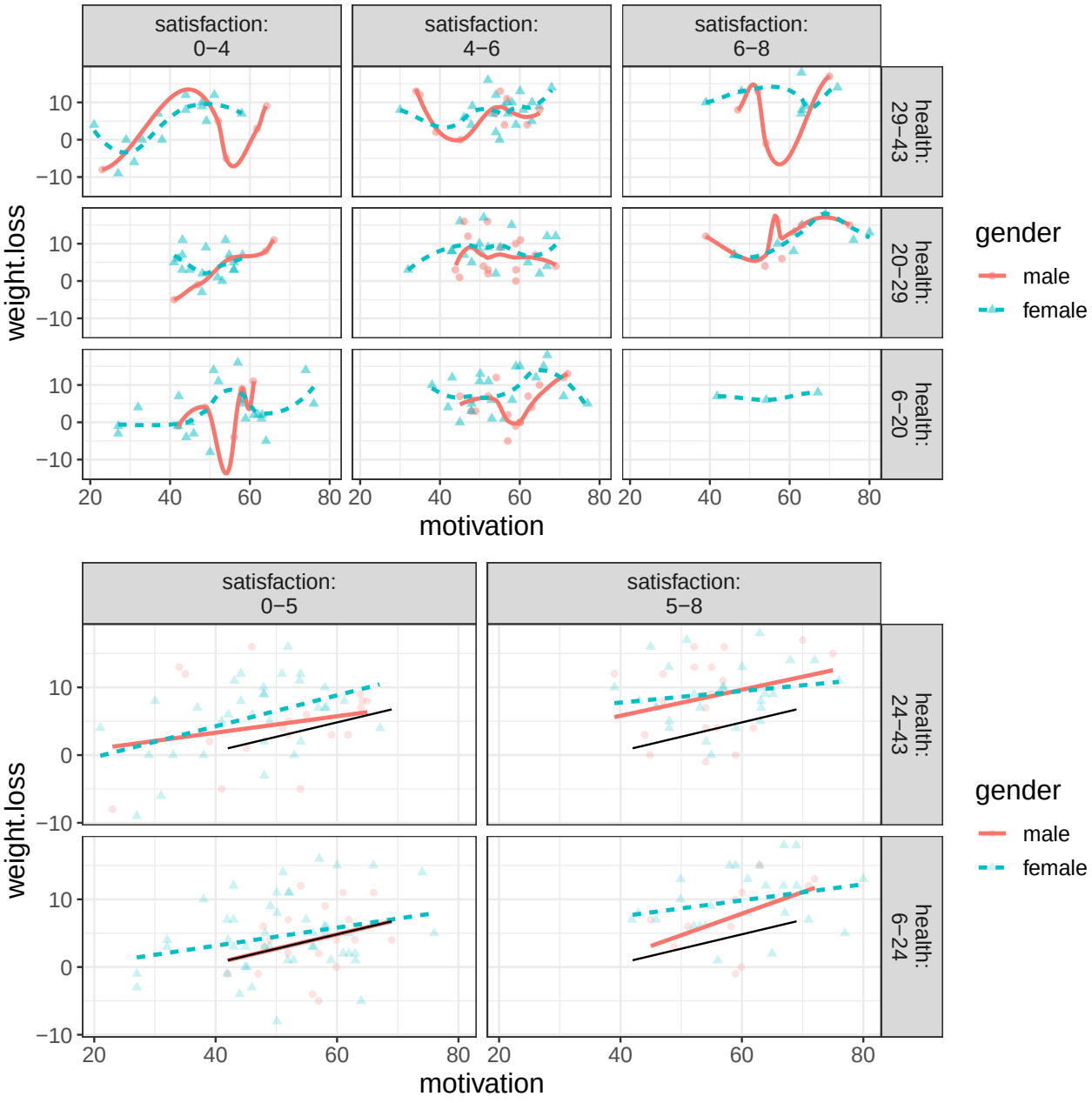

Figure 21. Multivariate relationship between five variables. Each `flexplot` slot is occupied and it is difficult to interpret what is going on in the top figure, though the use of regression lines instead of loess lines, removing standard errors, reducing transparency of the datapoints, adding ghost lines, and reducing the number of bins have made it easier to interpret (bottom image).

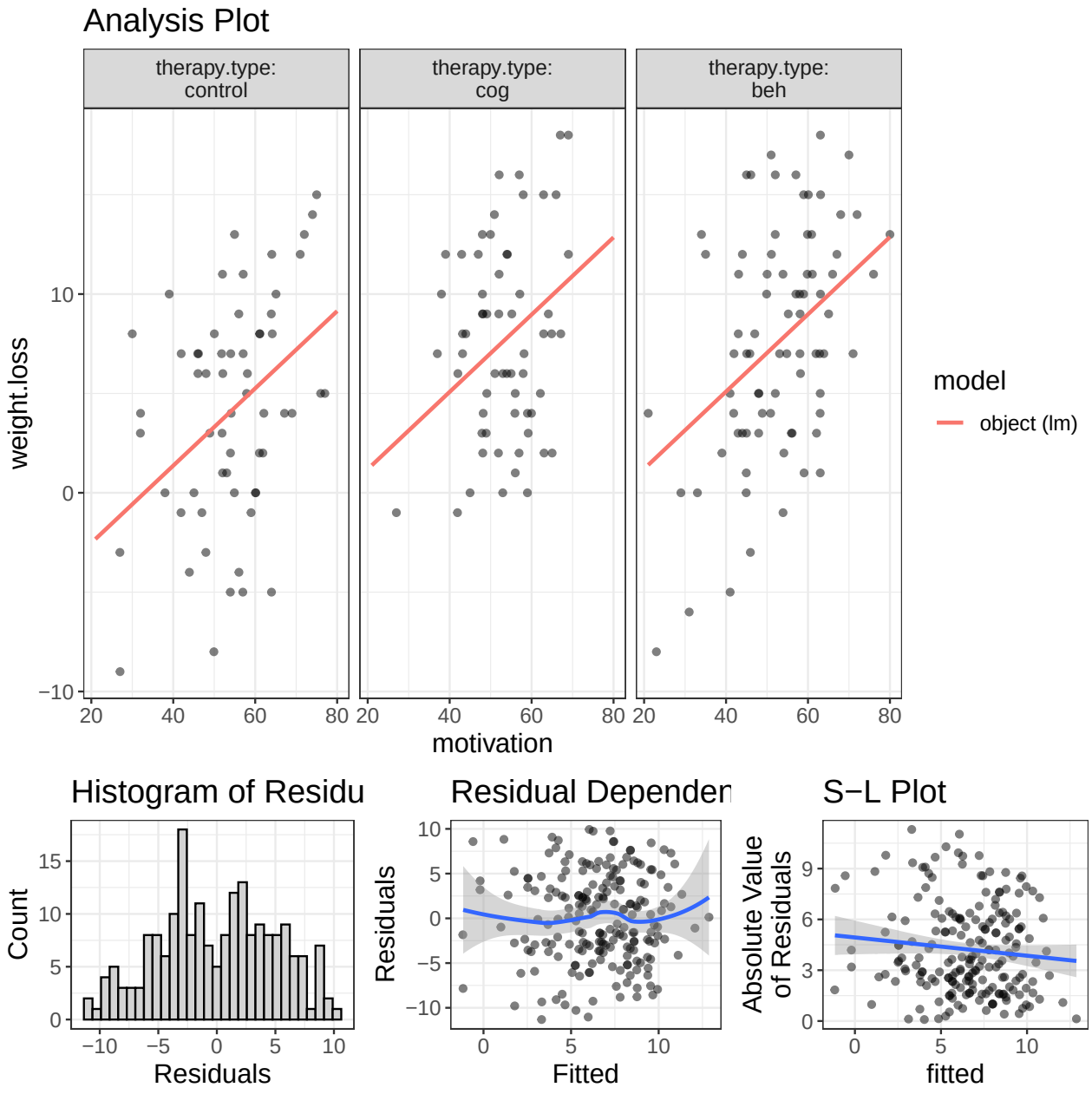

Figure 22. Demonstration of the `visualize` function on a `lm` object. The top row shows a representation of the statistical model. The bottom row shows diagnostic plots.

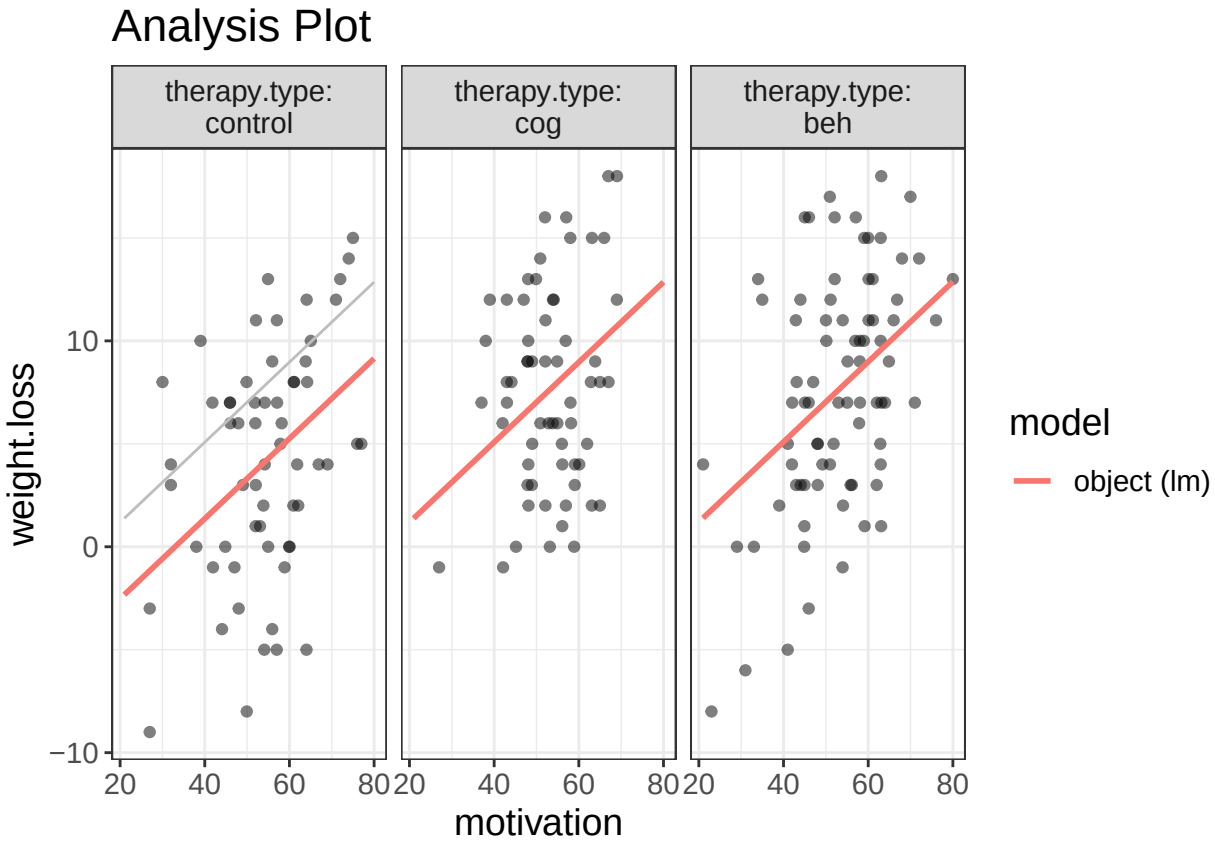

Figure 23. Demonstration of the `visualize` function on a `lm` object, but with `flexplot` arguments controlling the output (as well as suppressing residuals).

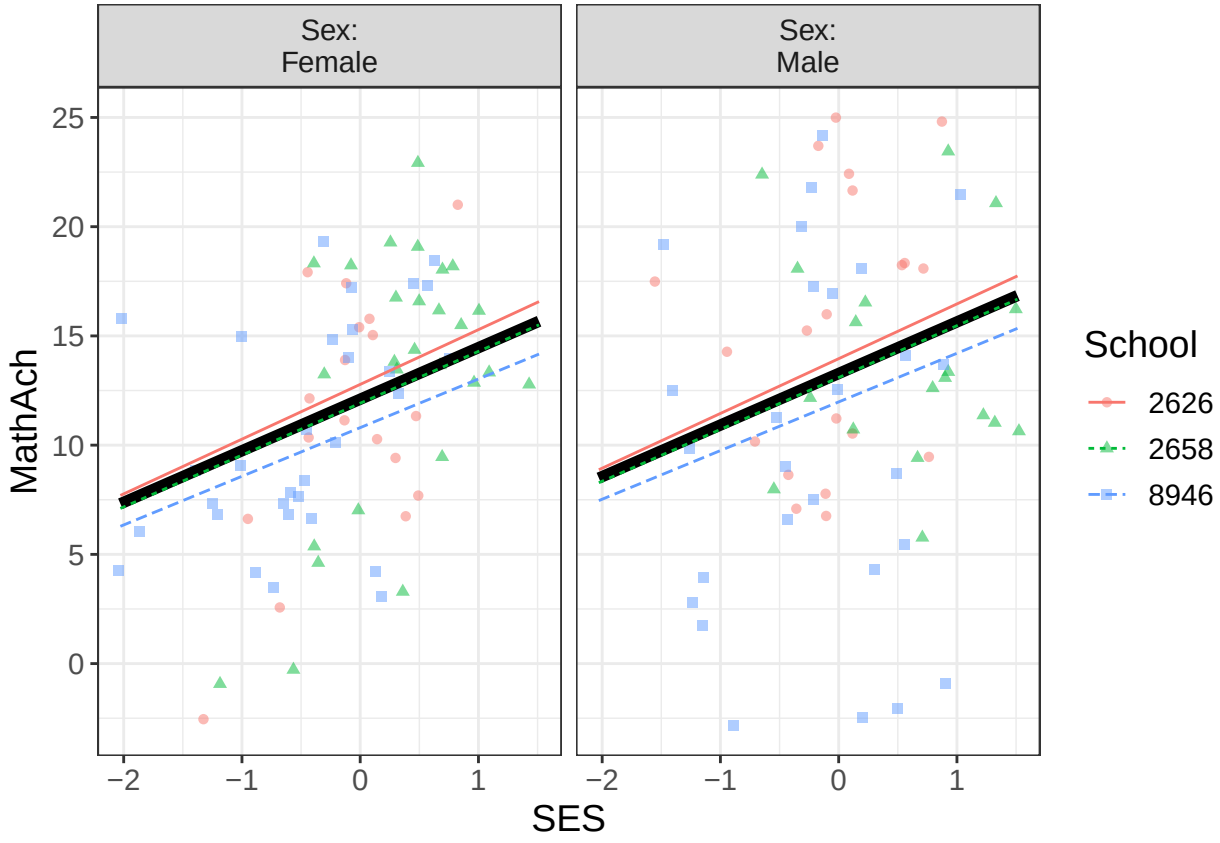

Figure 24. Demonstration of the `visualize` function for a mixed model. In this graphic, each thin line represents the fit of a particular school.

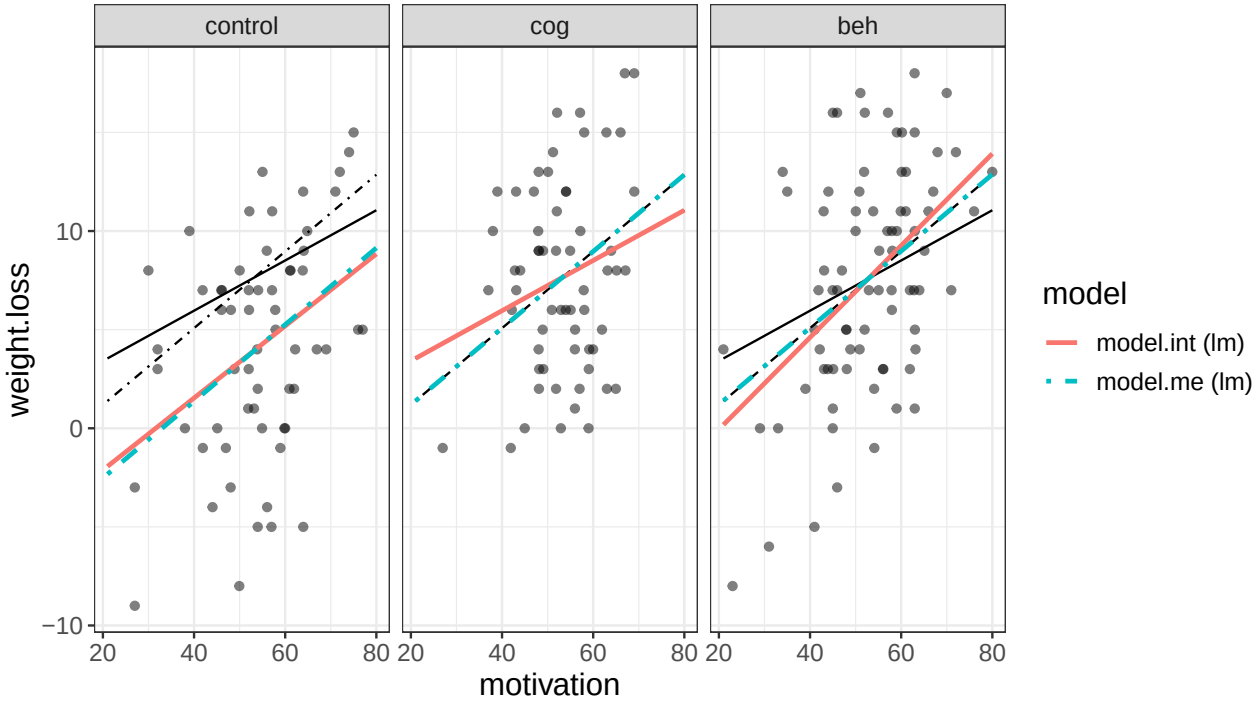

Figure 25. Demonstration of the `compare.fits()` function, comparing a main effects and an interaction model. Ghost lines have been added from the `cog` condition for easier comparison across panels.