Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

flexplot: Graphically-Based Data Analysis

Dustin Fife
Rowan University

Abstract

The human visual processing system has enormous bandwidth, able to interpret vast amounts of data in fractions of a second (Otten, Cheng, and Drewnowski 2015). Despite this amazing ability, there is a troubling lack of graphics in scientific literature (Healy and Moody 2014), and the graphics most traditionally used tend to bias perception in unintentional ways (Weissgerber, Milic, Winham, and Garovic 2015). I suspect the reason for the underuse and misuse of graphics is because sound visuals are difficult to produce with existing software. 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 visuals that map onto statistical procedures. With one-line functions, users can visualize bivariate statistical models (e.g., scatterplots for regression, jittered density plots for ANOVA/t-tests), multivariate statistical models (e.g., ANCOVA and multiple regression), and even more sophisicated 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, sampling, and jittered-density plots.

Keywords: flexplot, statistical assumptions, statistical modeling, replication crisis, visualization, exploratory data analysis, graphical data analysis.

1. Introduction

In light of the recent "replication crisis" in science (Pashler and Wagenmakers 2012; Collaboration 2015), researchers are becoming increasingly concerned with the validity of science (Baker 2016). As such, many are pushing for greater transparency and reproducibility (Nelson, Simmons, and Simonsohn 2018; Nosek, Ebersole, DeHaven, and Mellor 2018). One way to accomplish greater transparency is through abudant use of data visualizations (Tay, Parrigon, Huang, and LeBreton 2016), particularly when data are presented in their raw form.

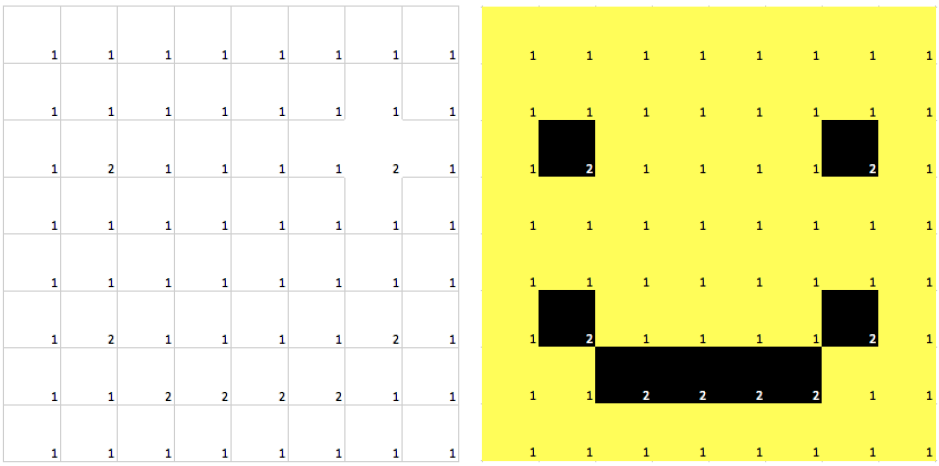


Figure 1: A table of numbers on the left, and a color-coded table on the right, where the 1's have been highlighted in yellow and the 2's in black. With the color, a pattern emerges that was not easy to see without the graphic. Figure used with permission from Correll (2015).

In addition to answering the call for greater transparency, graphics offer several additional advantages. First, they improve communication between the scientific community and the public (Otten *et al.* 2015). Lay people can understand relatively sophisticated statistical procedures when that information is presented graphically (Correll 2015). Additionally, graphics highlight problems with models that are masked by traditional statistical procedures, such as nonlinearity, outliers, and heteroscedasticity (Healy and Moody 2014; Levine 2018). As such, graphics serve as an important diagnostic check. Finally, graphics improve encoding (Hansen, Chen, Johnson, Kaufman, and Hagen 2014). 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. 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, and Longo 2019). 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) require more effort than many are willing to perform (the user must model the data, export the residuals, then produce a scatterplot).

flexplot was designed to address these limitations. It was developed using empirically-derived heurstics that maximize perceptual understanding, while minimizing perceptual biases (Fife *et al.* 2019). Also, the graphics produced are simple to generate and permit analysts to quickly

shift between statistical modeling and graphical interpretation.

2. Existing graphical software

This package builds upon **ggplot2** (Wickham 2011) and aims to convert R models to graphics. Other packages are designed to do similar things. For example, **visreg** (Breheny and Burchett 2017) also offers one-line functions for models, though lacks some of the tools unique to **flexplot** (such as ghost lines and advanced customization of how variables are visualized). Furthermore, it doesn't allow one to visually compare models like **flexplot** does and **visreg** can only visualize a smaller subset of variables than **flexplot** can. **ggfortify** (Tang, Horikoshi, and Li 2016) also offers the ability to visualize fitted models, but focuses on clustering and time series analyses. Finally, **tourr** (Wickham, Cook, Hofmann, and Buja 2011), like **flexplot**, offers multivariate visualizations through tours of multivariate data, but these are animated graphics, which can only be reproduced in a limited number of environments. No other package, that I know of, was derived using principles of human factors and visual processing.

2.1. Guiding philosophy of flexplot

The design of **flexplot** is based on the following principles:

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 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 a 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 overcome miconceptions about statistical analyses, some of the tools within **flexplot** create visual representation of the statistical models. These representations highlight uncertainty, reveal whether chosen models are appropriate, and improve encoding of statistical information.

3. A new grammar of graphics

Hadley Wickham, the author of **ggplot2**, developed a "layered" grammar of graphics (Wickham 2010). Wickham's grammar allows a great deal flexibility in the design of graphics. However, this flexibility comes at 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:

```
R> plot = ggplot(data = exercise_data, aes(x=therapy.type, y=weight.loss)) +
   geom_jitter(width = .2, alpha = .4) +
   stat_summary(fun.y = 'mean', geom = 'point',
         size = 3, position = position_dodge(width = .2)) +
   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))
```

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:

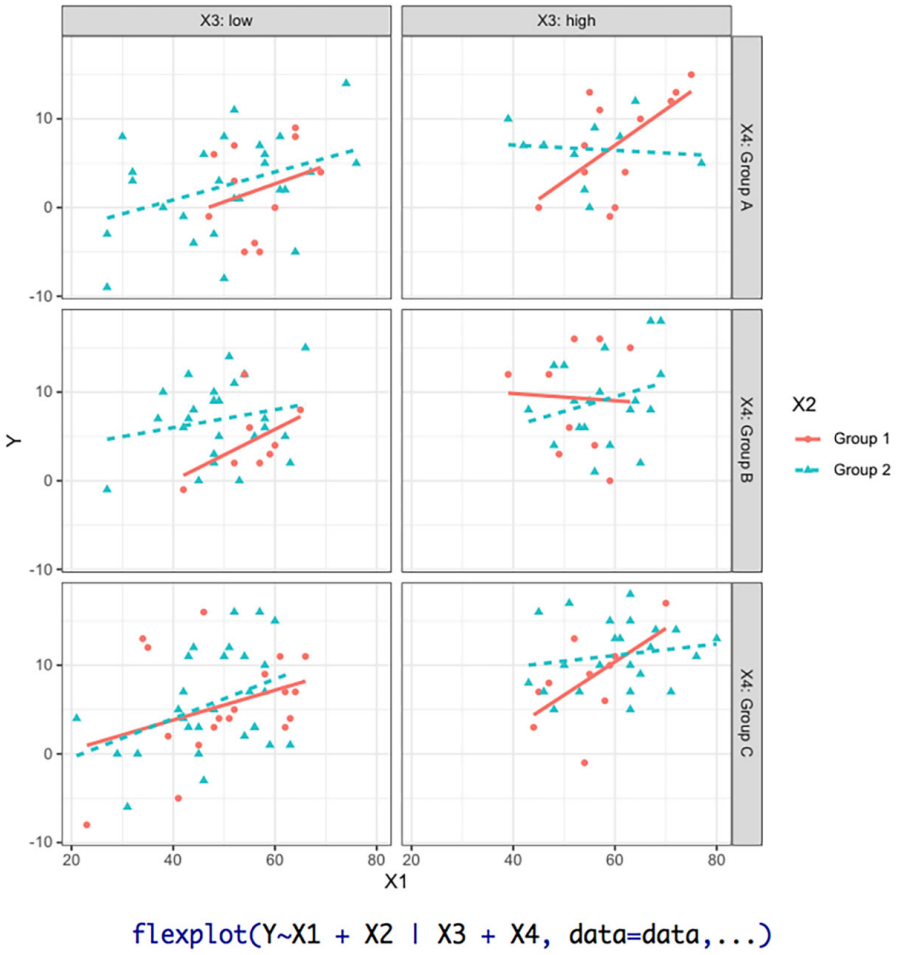
```
R> plot = flexplot(weight.loss ~ therapy.type, data = exercise_data)
```

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.

flexplot's grammar is also a layered grammar (but incidentally, because it developed within **ggplot2**'s grammar), though its grammar is actually based on the general linear model (GLM). Recall that most statistical procedures are subsumed within the GLM, 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 GLMs to do various sorts of modeling, and all this is accomplished using a simple equation (e.g., `y ~ x1 + x2`). Likewise, **flexplot** adopts the same convention, utilizing a similar equation to produce graphics. Figure 2 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. By following the grammar of a GLM, there is consistency between statistical modeling and visualization, with a few notable exceptions. Most obviously, a GLM equation (e.g., `plot(y ~ x1 + x2 + x3, data = data)`) does not have any vertical pipes (`|`) as `flexplot()` does. This is necessary in `flexplot()` to allow more specificity in paneling. Additionally, with GLMs, 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 visual will show it.

The base `plot()` function in R follows similar conventions as **flexplot** (i.e., users can specify an equation, such as `plot(y ~ x)`), though `flexplot()` is more intelligent in its choice

Figure 2: 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.



of displays. 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 visuals with models, then conclude with a brief summary.

4. 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 with visualizing univariate

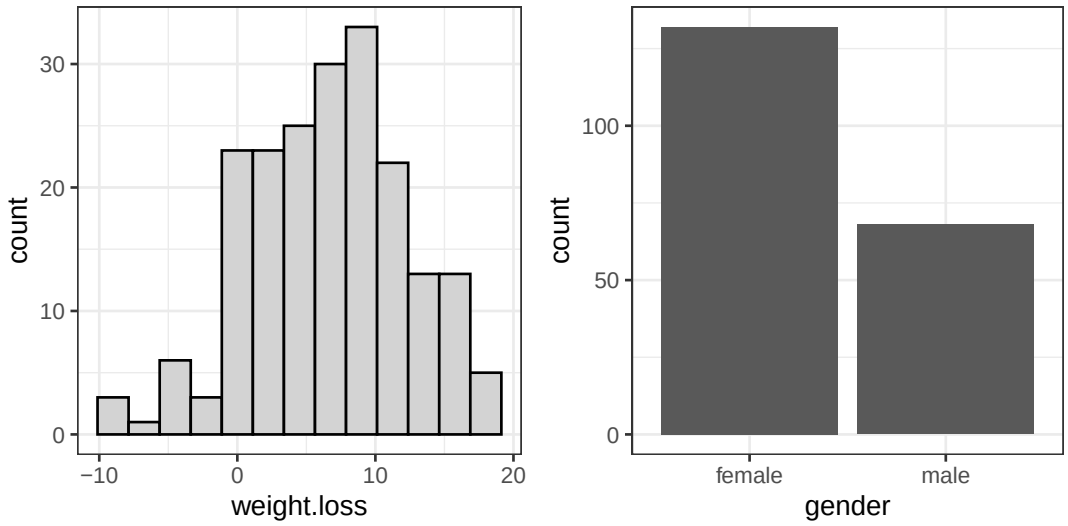


Figure 3: A histogram (left) and barchart (right) produced within **flexplot**.

distributions. However, the type of graphic displayed depends on the type of variable inputted into the function. For example, in the code below, notice that `flexplot()` recognizes whether the variable is categorical or numeric, and plots accordingly.

```
R> require(flexplot)
R> data(exercise_data) #### these are simulated data available
R>                     #### in flexplot. Please don't base your
R>                     #### weight loss program on this dataset
R> a = flexplot(weight.loss ~ 1, data = exercise_data)
R> b = flexplot(gender ~ 1, data = exercise_data)
R> cowplot::plot_grid(a , b)
```

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, simply convert the variable to a factor (e.g., `data$group = factor(data$group, levels = 1:3, labels = c("Group1", "Group2", "Group3"))`).

5. 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 = jittered density plot, and categorical predictor/categorical outcome = association plot.

5.1. Categorical predictor, numeric putcome (jittered-density plots)

There are many different ways to visualize a categorical predictor/numeric outcome relationship, including bar plots of means, box plots, violin plots, gradient plots, etc. Some are

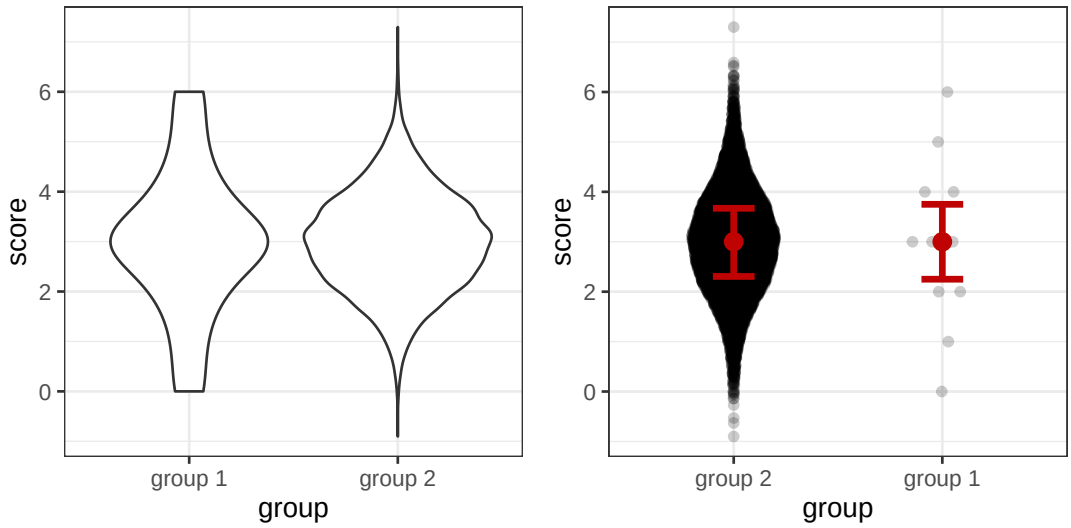


Figure 4: Violin plots with 15,000 versus 15 datapoints. The outlines look the same in the left image, but the right image overlays the raw data, which makes the differing sample sizes much more apparent.

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). `flexplot()` defaults to a "textured dot strip," which was invented by Tukey and Tukey (1990) (see also Wilkinson 1999). However, I am not too fond on the name. It lacks pizzazz. I have taken the liberty of renaming them "jittered density plots," or JD plots for short. I leave it to the reader to decide if this is pizzazzy enough. These are essentially violin plots with raw data. It is important to display raw data because without, 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 4 look essentially the same, while the same data, plotted as JD plots, very clearly show the sample size.

```
R> group1 = c(0,1,2,2,3,3,3,3,3,3,4,4,5,6)
R> group2 = rnorm(10000,3,1)
R> d = data.frame(score = c(group1, group2),
                  group = c(rep("group 1", times = length(group1)),
                            rep("group 2", times = length(group2))))
R> a = ggplot2::ggplot(data = d, aes(x = group, y = score)) +
   geom_violin() + theme_bw()
R> b = flexplot(score ~ group, data = d)
R> cowplot::plot_grid(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 10% of the way to the right only at the highest density and 10% to

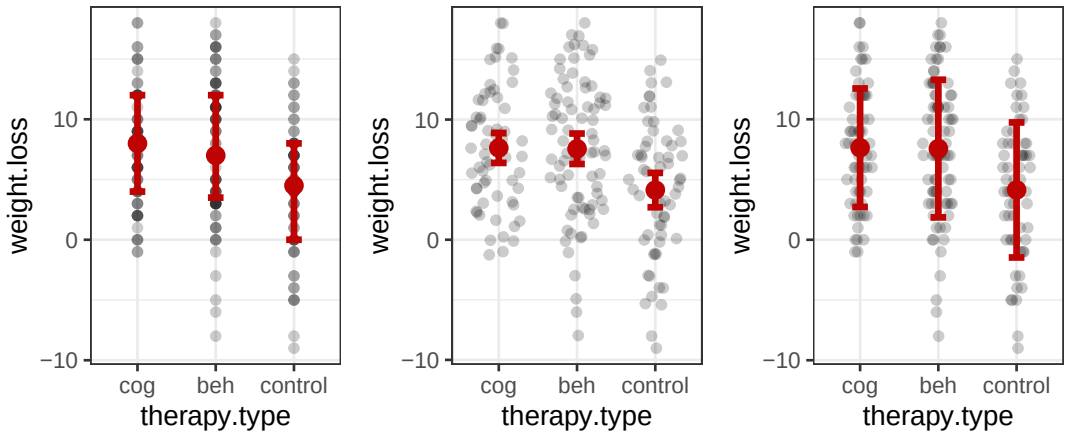


Figure 5: Dot plots with the interquartile range and no jittering (left), JD plot with mean + standard errors and jittering on X and Y (middle), and JD plot with mean + standard deviation with jittering on only X (right).

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:

```
R> a = flexplot(weight.loss ~ therapy.type, data = exercise_data,
                jitter = F, spread = "quartile")
R> b = flexplot(weight.loss ~ therapy.type, data = exercise_data,
                jitter = c(.4,.5), spread = "sterr")
R> c = flexplot(weight.loss ~ therapy.type, data = exercise_data,
                jitter = .2, spread = "stdev")
R> cowplot::plot_grid(a, b, c, nrow = 1)
```

5.2. 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. The fit of the graph defaults to a loess line, so as to highlight deviations from linearity. However, the user can specify other sorts of fits, such as `"lm"` (for regression), `"polynomial"`, `"cubic"`, and `"rlm"` (robust linear model) in the **MASS** package (Venables and Ripley 2002). 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.

```
R> a = flexplot(weight.loss ~ satisfaction, data = exercise_data) +
   theme_minimal() ### using layering to change theme
R> b = flexplot(weight.loss ~ satisfaction, data = exercise_data,
                method = "lm", se = F)
R> c = flexplot(weight.loss ~ satisfaction, data = exercise_data,
```

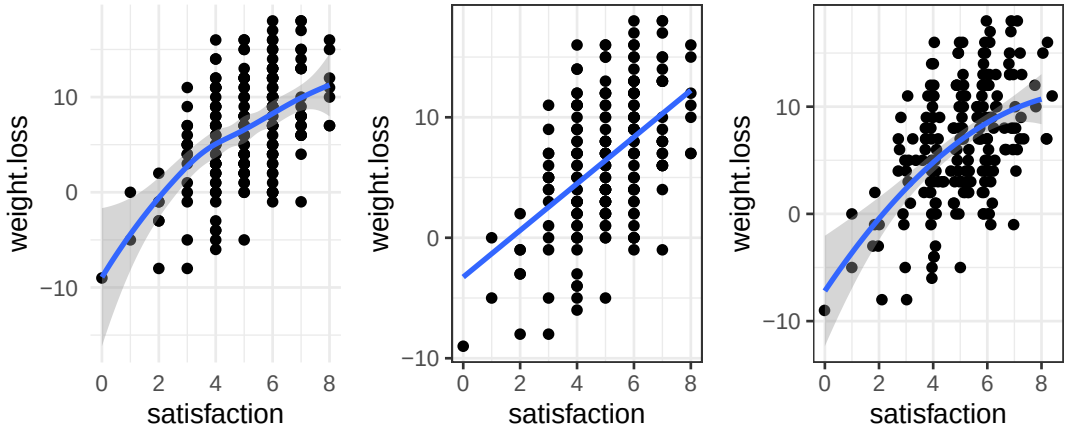


Figure 6: Scatterplot with different options of fit: `loess` (default), `lm` (regression), `polynomial`. Also, the data in the far right plot has been jittered.

```
              method = "polynomial", jitter = .4)
R> cowplot::plot_grid(a, b, c, nrow = 1)
```

5.3. 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. Any binary variable can be visualized as a logistic regression in **flexplot**, except when the axis variable is also categorical (i.e., the variable occupying the first slot in the `flexplot()` equation, which in the case of Figure 7 is `attention`). To do so, the user only needs to specify `logistic` as the method.

```
R> data("tablesaw.injury") ### also simulated data available
R>                         ### in flexplot package
R>                         ### always remember to be safe
R>                         ### and attentive when woodworking
R> flexplot(injury ~ attention, data = tablesaw.injury,
              method = "logistic", jitter = c(0, .05))
```

5.4. Categorical outcome, categorical predictor (association plots)

Sometimes the analyst may wish to visualize 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). In the example below, I had to convert `injury` to a factor to get a barplot. The graphic (Figure 8) shows that females are less likely to be injured than males, relatively speaking.

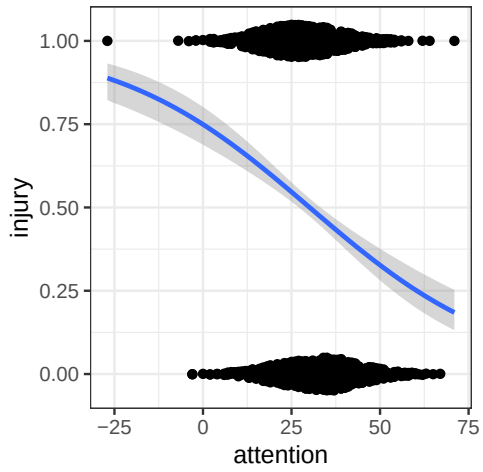


Figure 7: Example of a logistic plot in the **flexplot** package.

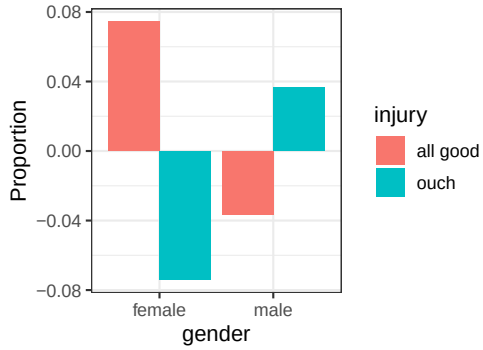


Figure 8: Example of an association plot for categorical predictor/categorical outcome.

```
R> tablesaw.injury$injury = factor(tablesaw.injury$injury,
                    levels=c(0, 1), labels=c("all good", "ouch"))
R> flexplot(injury ~ gender, data = tablesaw.injury)
```

5.5. Repeated measures data (related t-test)

As I mentioned previously, one of the guiding tenets of **flexplot** is that every statistical analysis ought to be accompanied by a graphic that closely matches the analysis. 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 (Wickham 2017), 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 9). For example:

```
R> data("plant_growth")
R> flexplot(Diameter ~ Soil.Type, data = plant_growth, related=T)
```

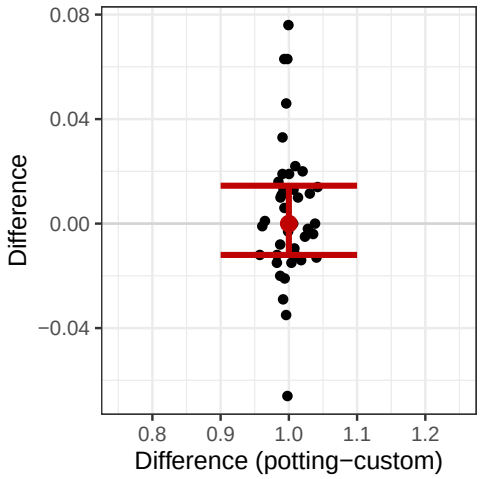


Figure 9: A plot of repeated measures data.

(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 visualizing 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.

5.6. 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 becomes quite difficult to see any patterns; areas of high density look just as crowded as areas of lower density, relatively speaking (although having JD plots makes it clear which areas are most dense; see the example in top-left image in Figure 10). To address such overlap, `flexplot()` offers three options. The first is to suppress raw data (`raw.data = F`, right-top in Figure 10). I don't recommend that, but it can be done. A second option is to reduce the transparency (e.g., bottom-left in Figure 10). Perhaps the best option is to *sample* (bottom-right in Figure 10). 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 10, notice how the medians/interquartile ranges do not change, despite having different numbers of datapoints.

```
R> data("nsduh")
R> a = flexplot(distress ~ major.dep, data = nsduh)
R> b = flexplot(distress ~ major.dep, data = nsduh, raw.data = F)
R> c = flexplot(distress ~ major.dep, data = nsduh, alpha = .005)
```

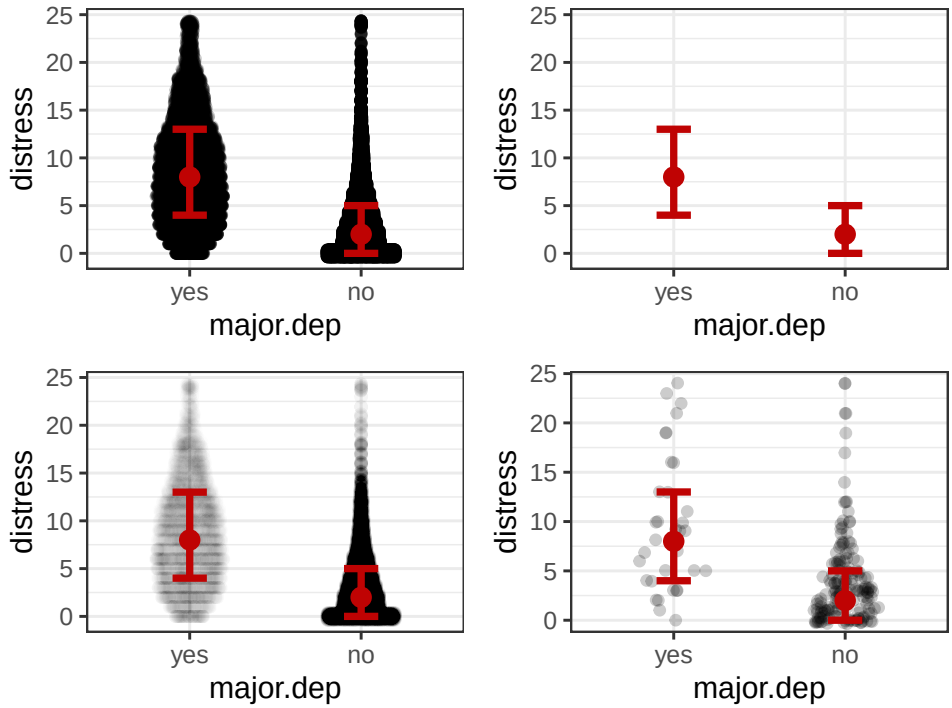


Figure 10: Four graphics showing different ways to handle overlapping datapoints. The top-left image does nothing. The top-right does omits raw data. The bottom-left reduces the opacity of the points. The bottom-right samples datapoints.

```
R> d = flexplot(distress ~ major.dep, data = nsduh, sample = 200)
R> cowplot::plot_grid(a, b, c, d, nrow = 2)
```

6. Multivariate graphics

Visualizing 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.

6.1. Added variable plots (AVPs) with `added.plot()`

AVPs are underused, yet extremely useful. Essentially, an AVP shows the relationship between a predictor of interest and the *residuals* of an existing model. 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 `weight.loss` from `motivation`,

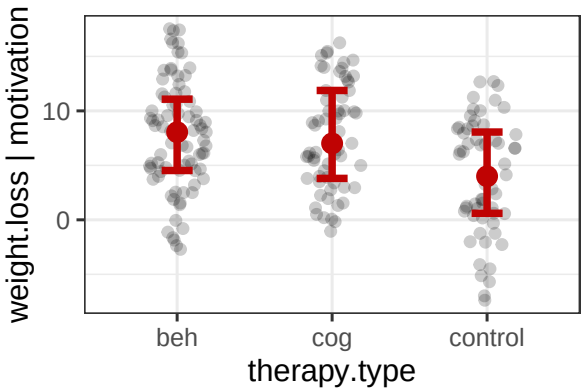


Figure 11: An Added Variable Plot (AVP) showing the relationship between `therapy.type` and `weight.loss`, after controlling for `motivation`.

residualize that relationship, then show a JD plot of the residuals for each type of therapy. This is what AVPs do (see Figure 11). **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. To counter this confusing, **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. What `flexplot()` will do is take the *last* variable entered (`therapy.type` in the following example) 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`). Other arguments can be passed to `added.plots()` as well (such as `alpha`, `sample`, `method`, etc.)

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

6.2. Colors/lines/shapes

As shown in Figure 2, the second slot in the `flexplot()` formula (`X2` in Figure 2) controls which variable is displayed as different colors/symbols/lines. Figure 12 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).

```
R> a = flexplot(weight.loss ~ motivation + gender,
     data = exercise_data, se = F, alpha = .3)
R> b = flexplot(weight.loss ~ therapy.type + gender,
     data = exercise_data, se = F, alpha = .3)
R> cowplot::plot_grid(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. In my experience, having a variable with more

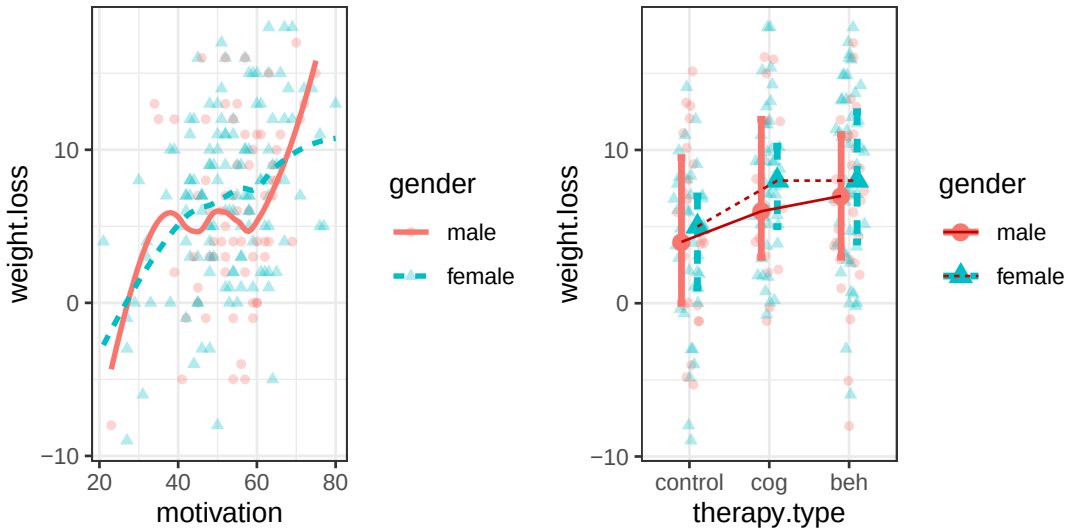


Figure 12: Two multivariate graphs illustrating how the second slot in a **flexplot** formula controls the visualization. The left image demonstrates what happens to the second slot variable ($X2$) when a numeric predictor is on the X-axis, while the right image demonstrates what happens to the second slot variable when a categorical predictor is on the X-axis.

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.

6.3. Paneling

An alternative (or additional) strategy for plotting multivariate data is paneling. As shown in Figure 2, 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 13 shows the same relationships in Figure 12, 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 visualized, 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).[1]

```
R> a = flexplot(weight.loss ~ motivation | gender,
                data = exercise_data)
R> b = flexplot(weight.loss ~ therapy.type | gender,
                data = exercise_data)
R> c = flexplot(weight.loss ~ motivation |  gender + therapy.type,
                data = exercise_data) +
```

[1] 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.

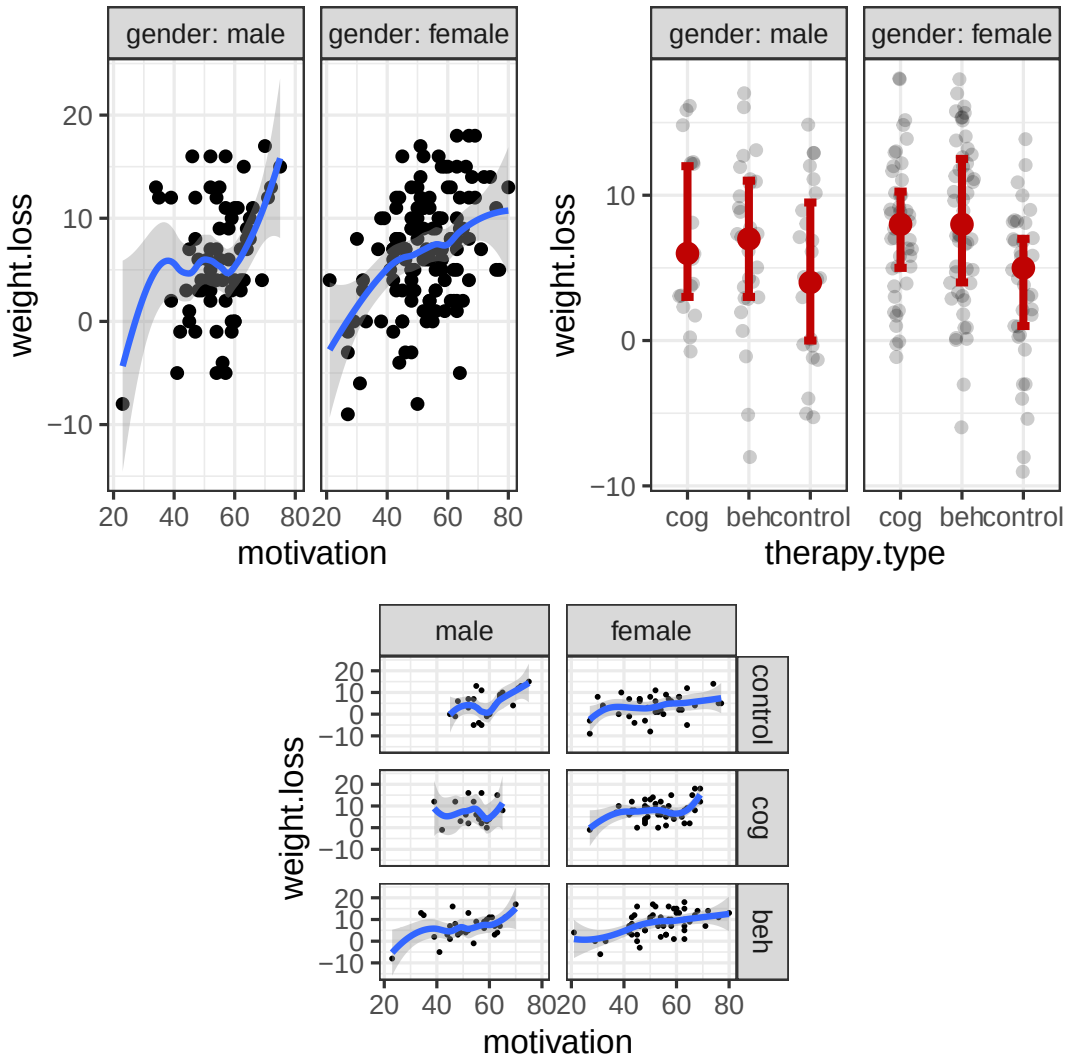


Figure 13: A multivariate plot where `therapy.type` and `gender` are now shown in panels.

```
        ggplot2::facet_grid(therapy.type ~ gender,
                labeller = ggplot2::labeller(therapy.type = label_value))
R> #### edit point size
R> c = ggplot2::ggplot_build(c)
R> c$data[[1]]$size = .25
R> c = ggplot2::ggplot_gtable(c)
```

6.4. 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 may 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 14 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.

```
R> a = flexplot(weight.loss ~ motivation | satisfaction,
                data = exercise_data,
                breaks = list(satisfaction=c(3,7))) +
   ggplot2::facet_grid( ~ therapy.type,
                 labeller = ggplot2::labeller(therapy.type = label_value))
R> b = flexplot(weight.loss ~ motivation + satisfaction,
                data = exercise_data,
                breaks = list(satisfaction=c(3,7)),
                labels = list(satisfaction=c("low", "medium", "high")))
R> c = flexplot(weight.loss ~ motivation + satisfaction,
                data = exercise_data,
                bins = 2)
```

6.5. 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 15 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 15, 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.

```
R> flexplot(weight.loss ~ motivation | satisfaction,
                data = exercise_data, method = "lm",
                bins = 3, ghost.line = "red")
```

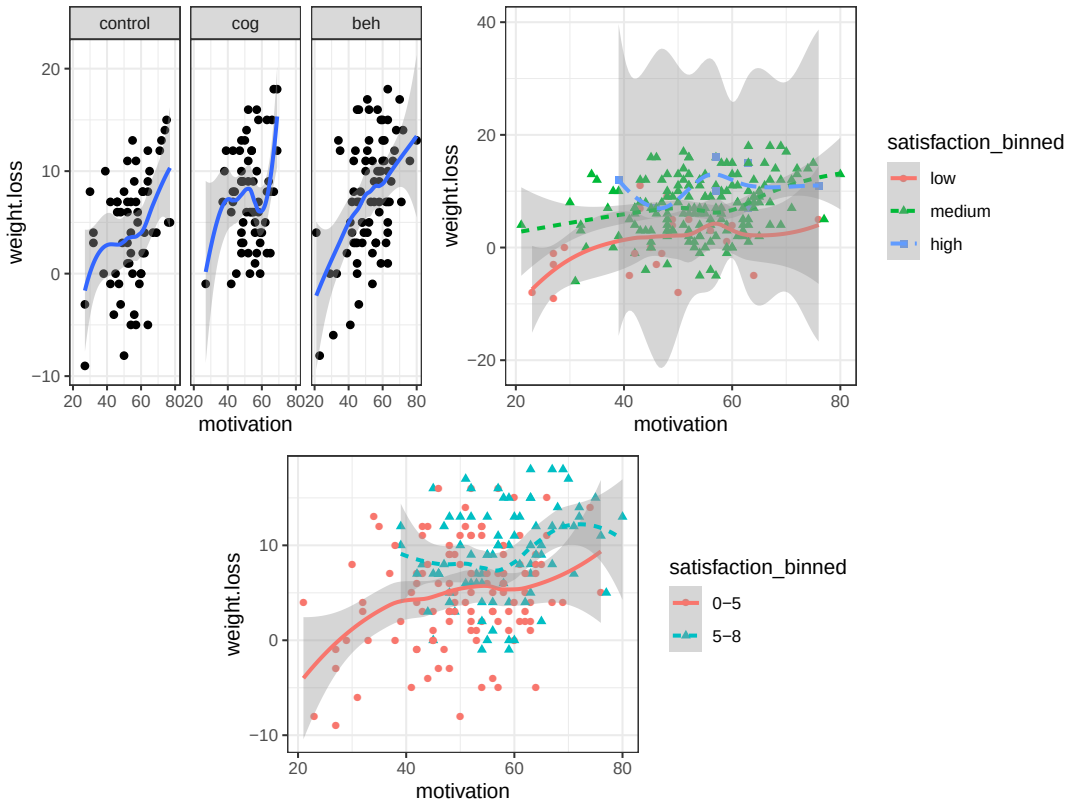


Figure 14: Plots reflecting choices of different break points (left-most plot), labels (right plot), and bins (bottom plot).

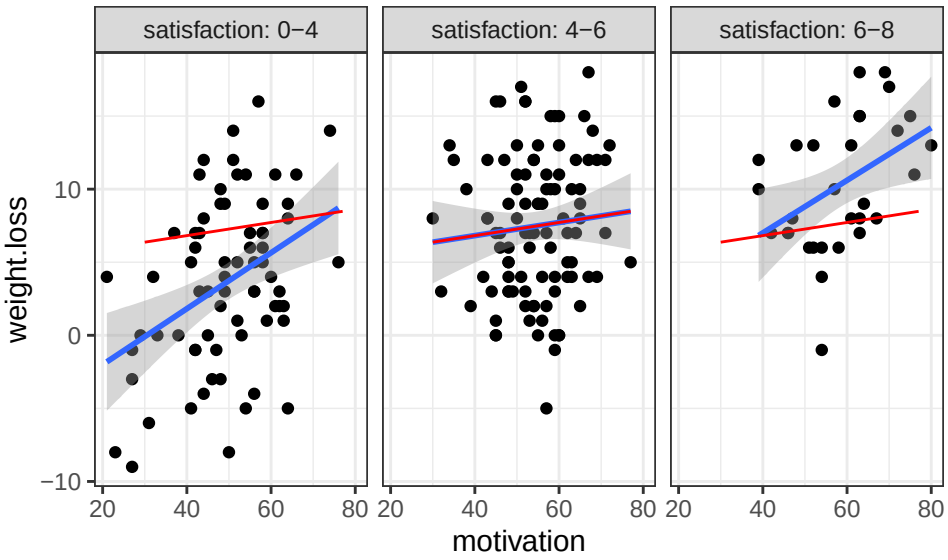


Figure 15: Ghost lines repeat the pattern from one panel to the others, making it easier to compare across panels.

6.6. General strategy for plotting multivariate relationships

It is easy for multivariate visuals to become unnecessarily complicated. For example, the top image in Figure 16 shows a graphic where all four `flexplot()` slots are occupied. This is very difficult to interpret what is going on. To simplify things, we could utilize several 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.[2] 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 16 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 16, 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 visuals with statistical modeling.

```
R> a = flexplot(weight.loss ~ motivation + gender | satisfaction + health,
     data = exercise_data)
R> b = flexplot(weight.loss ~ motivation + gender | satisfaction + health,
     data = exercise_data,
     method = "lm", se = F, bins = 2, ghost.line = "black", alpha = .2,
     ghost.reference = list(satisfaction = 0, health = 10, gender = "male"))
R> cowplot::plot_grid(a, b, ncol = 1)
```

7. Combining modeling and visualizations

Visuals 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 16, 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 visualizations, 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.

7.1. The `visualize()` function

[2] 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.

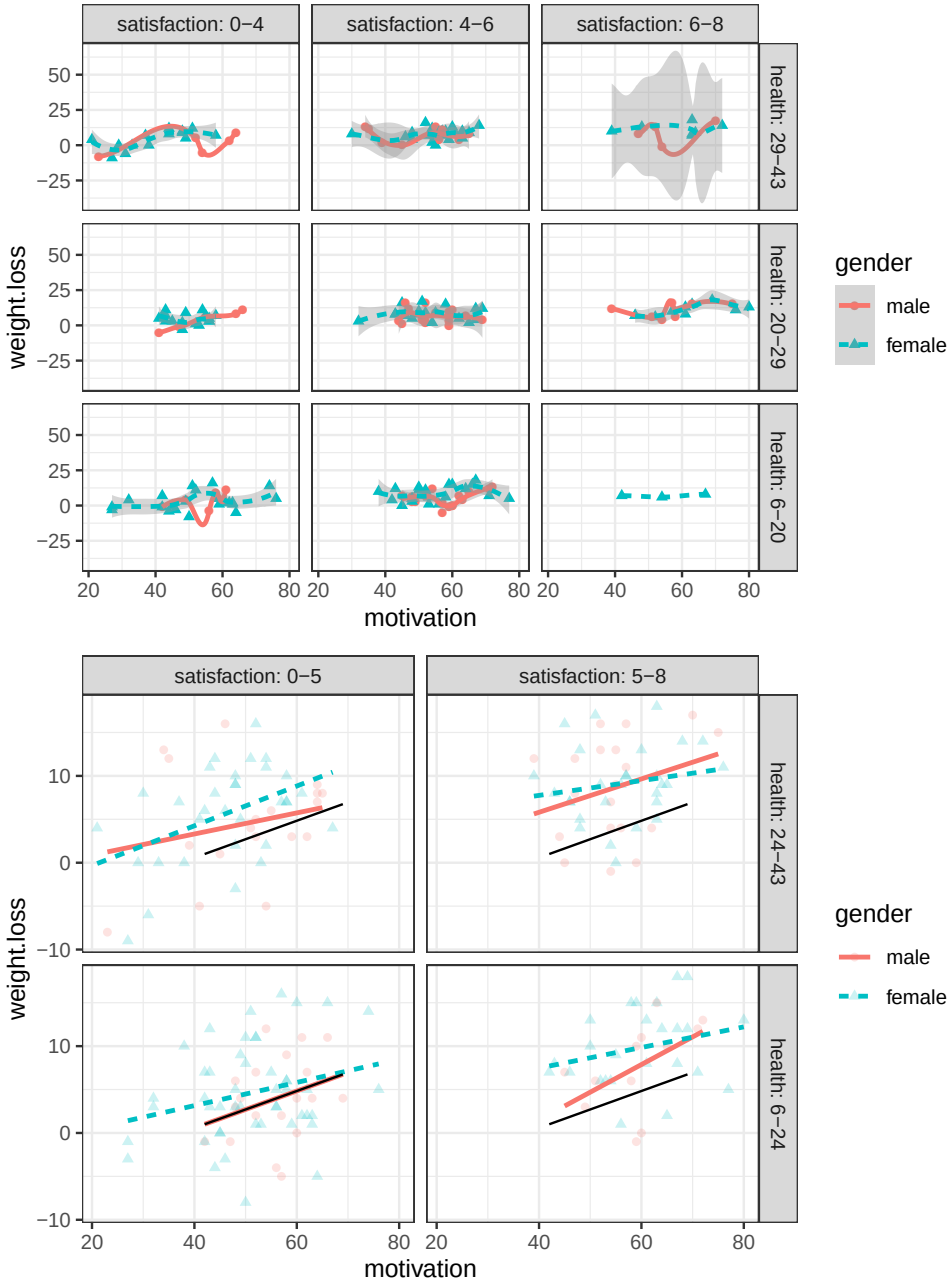


Figure 16: Multivariate relationship between five variables. Each **flexplot** slot is occupied and it is difficult to interpret what is going on in the top figure, though the use of regression lines instead of loess lines, removing standard errors, reducing transparency of the datapoints, adding ghost lines, and reducing the number of bins have made it easier to interpret (bottom image).

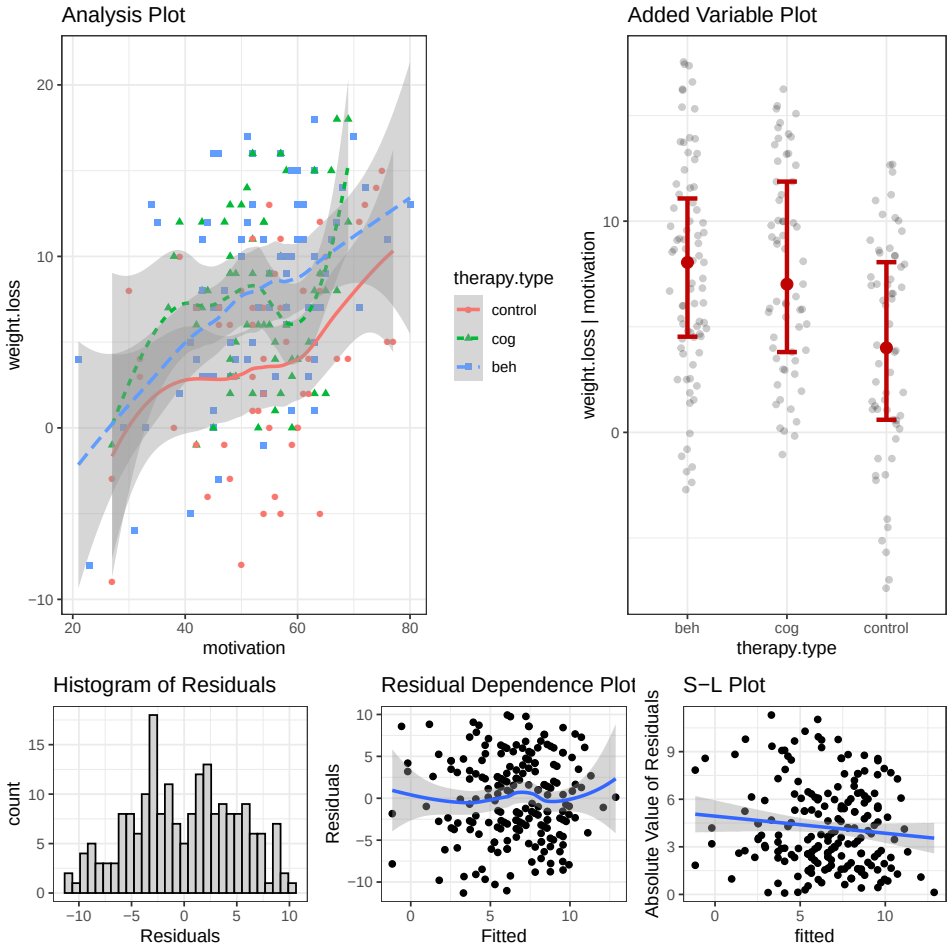


Figure 17: Demonstration of the `visualize` function on a `lm` object. The top row shows two different representations of the statistical model. The bottom row shows diagnostic plots.

As I've mentioned repeatedly, one of the primary purposes of **flexplot** is 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 visualize it as follows:

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

For multivariate data, visualize will default to producing two graphics to match the analysis: one AVP (top right graphics in Figure 17) and one other 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,`

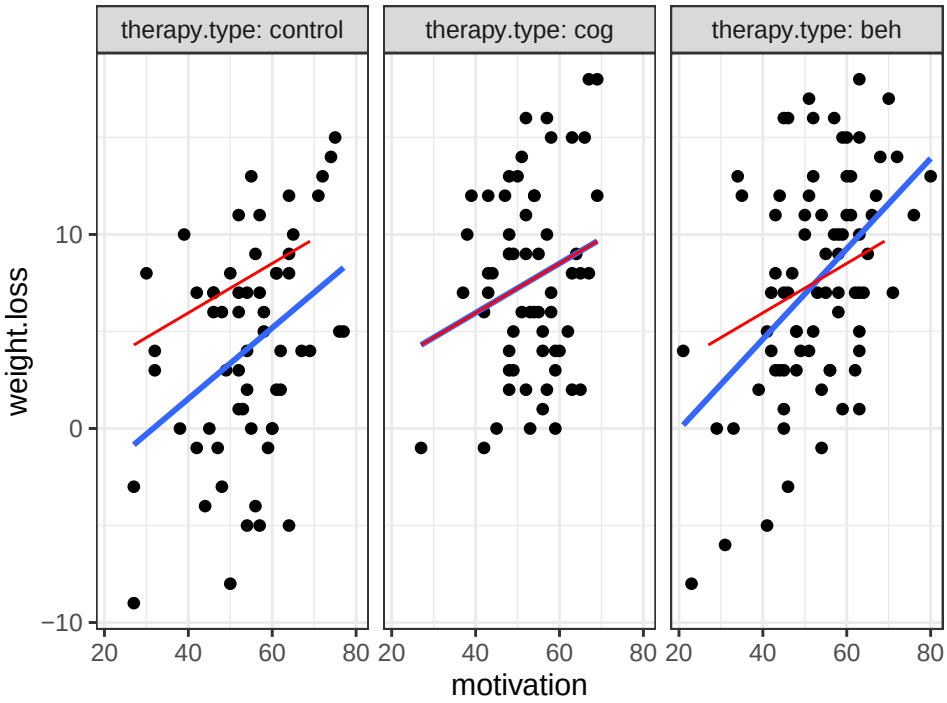


Figure 18: Demonstration of the visualize function on a `lm` object, but with **flexplot** arguments controlling the output (as well as suppressing residuals).

`"residuals"`)). Additionally, visualize can take `flexplot()` arguments and even a `flexplot()` formula.

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

Figure 19 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 visualize a subset of the subjects in a mixed model to ensure the model chosen is appropriate.

```
R> require(lme4)
R> data(math)
R> model = lmer(MathAch ~ Sex + SES + (SES|School), data = math)
R> visualize(model,
   plot = "model",
   formula = MathAch ~  Sex + School| SES,
   sample = 30)
```

7.2. The `compare.fits()` function

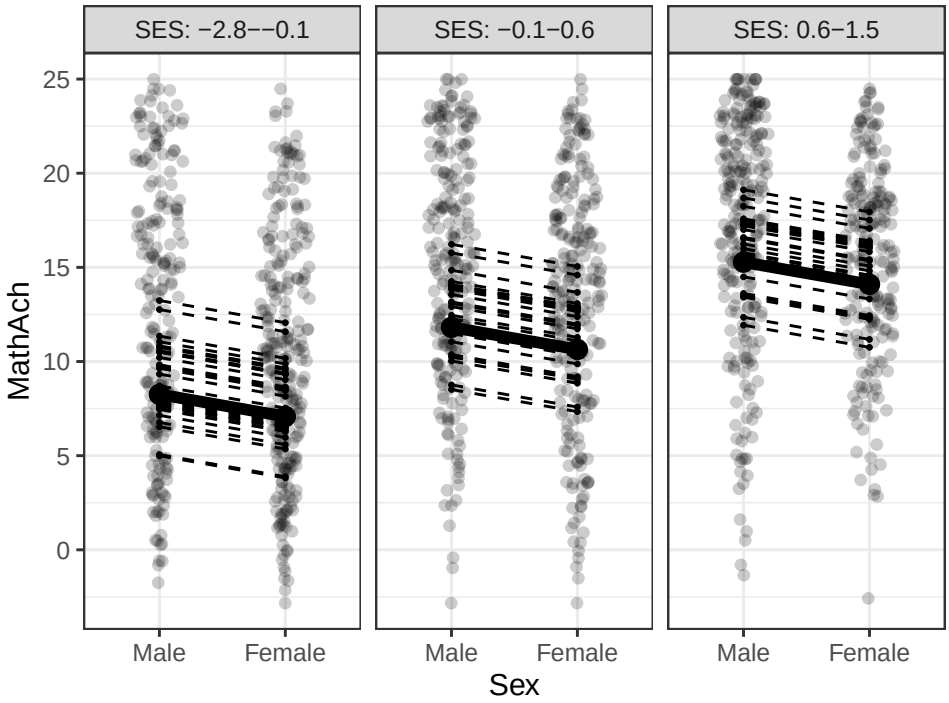


Figure 19: Demonstration of the `visualize` function for a mixed model. In this graphic, each dashed line represents the fit of a particular school.

Very often in statistical modeling, we are interested in comparing two models, 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 visuals 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 20 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.

```
R> model.me = lm(weight.loss ~ motivation + therapy.type, data = exercise_data)
R> model.int = lm(weight.loss ~ motivation * therapy.type, data = exercise_data)
R> compare.fits(weight.loss ~ motivation | therapy.type,
              data = exercise_data, model.me, model.int, ghost.line = "black")+
      ggplot2::facet_grid( ~ therapy.type,
          labeller = ggplot2::labeller(therapy.type = label_value))
```

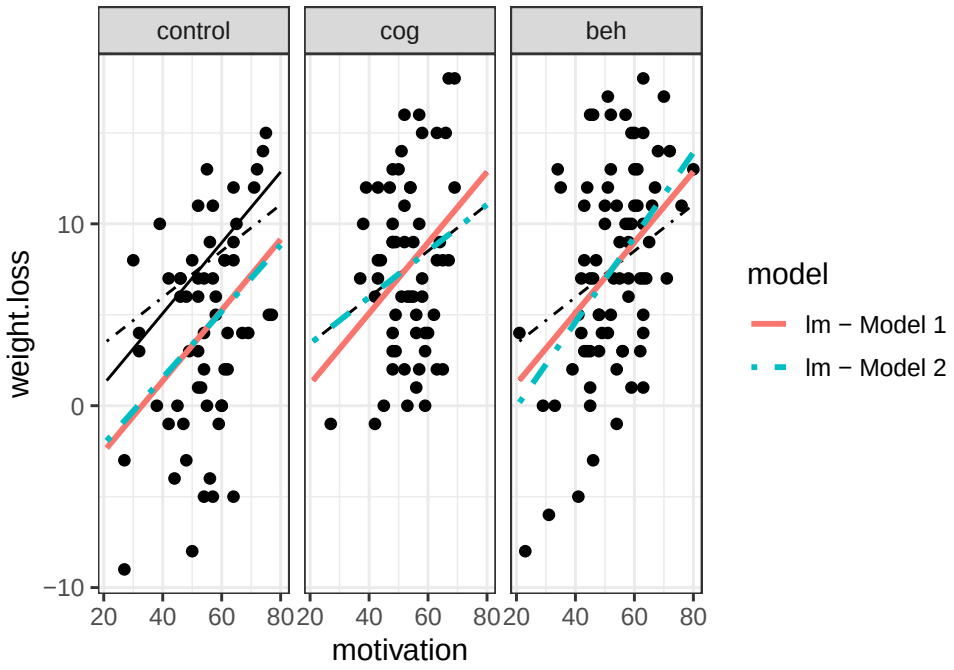


Figure 20: Demonstration of the `compare.fits()` function, comparing a main effects and an interaction model. Ghost lines have been added from the `cog` condition for easier comparison across panels.

8. 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 visuals. These functions include the `estimates()` method and `model.comparison()`.

8.1. The `estimates()` function

The `estimates()` function 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).

```
R> estimates(model.int)

Model R squared:
0.222 (0.12, 0.32)
```

```
Semi-Partial R squared:
motivation:therapy.type
                  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 17-18, as well as Figure 20, namely that the addition of the interaction likely is not improving the fit enough to consider keeping.

8.2. 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, in standardized units. 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 a third of a standard deviation, while the median difference is only about 0.05 standard deviations, 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.

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

$statistics
               aic      bic bayes.factor p.value r.squared
model.int 1223.041 1246.129        0.010   0.487     0.222
```

```
model.me  1220.523 1237.015       95.294             0.217

$pred.difference
   0%   25%   50%   75%  100%
0.001 0.017 0.050 0.078 0.348
```

9. Summary

This paper has sought to demonstrate the utility of the **flexplot** package. This visual approach to data analysis has many advantages, including emphasizing uncertainty, improving encoding, and increasing communication with lay audiences. The formula-based, layered grammar of graphics makes many basic and advanced plotting functions much easier to produce, and does so using scientifically-derived principals of human perception. Additionally, **flexplot** allows tools that permit seamless integration of statistical analysis with visual interpretation.

10. Computational details

All analyses were conducted in RStudio (RStudio Team 2016), version 1.1.463. Styling was produced with RMarkdown (Xie, Allaire, and Grolemund 2018), v 1.16 and rticles (Allaire, Xie, R Foundation, Wickham, Journal of Statistical Software, Vaidyanathan, Association for Computing Machinery, Boettiger, Elsevier, Broman, Mueller, Quast, Pruim, Marwick, Wickham, Keyes, Yu, Emaasit, Onkelinx, Gasparini, Desautels, Leutnant, MDPI, Taylor and Francis, Öğreden, Hance, Nüst, Uvesten, Campitelli, Muschelli, Kamvar, and Ross 2019), v 0.11.1. This document used **flexplot** version 0.6.2, which is available at http://www.github.com/dustinfife/flexplot. Modifications to the displayed graphics were created using ggplot2 3.2.1.9000 (Wickham 2011) and **cowplot** (Wilke 2016). Mixed models were fit using **lme4** (Bates, Mächler, Bolker, and Walker 2015).

References

Allaire JJ, Xie Y, R Foundation, Wickham H, Journal of Statistical Software, Vaidyanathan R, Association for Computing Machinery, Boettiger C, Elsevier, Broman K, Mueller K, Quast B, Pruim R, Marwick B, Wickham C, Keyes O, Yu M, Emaasit D, Onkelinx T, Gasparini A, Desautels MA, Leutnant D, MDPI, Taylor and Francis, Öğreden O, Hance D, Nüst D, Uvesten P, Campitelli E, Muschelli J, Kamvar ZN, Ross N (2019). ***rticles:*** *Article Formats for R Markdown*. URL https://github.com/rstudio/rticles.

Baker M (2016). "1,500 scientists lift the lid on reproducibility." *Nature*, **533**(7604), 452–454. ISSN 0028-0836. doi:10.1038/533452a. URL http://www.nature.com/articles/533452a.

Bates D, Mächler M, Bolker B, Walker S (2015). "Fitting Linear Mixed-Effects Models Using **lme4**." *Journal of Statistical Software*, **67**(1), 1–48. doi:10.18637/jss.v067.i01.

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

Cleveland WS (1994). "Coplots, nonparametric regression, and conditionally parametric fits." *Lecture Notes-Monograph Series*, pp. 21–36. ISSN 0749-2170.

Collaboration OS (2015). "Estimating the reproducibility of psychological science." *Science*, **349**(6251), aac4716. ISSN 0036-8075, 1095-9203. doi:10.1126/science.aac4716. URL http://science.sciencemag.org/content/349/6251/aac4716.

Correll MA (2015). *Visual Statistics*. Doctoral dissertation, University of Wisconsin-Madison.

Fife DA, Tremoulet P, Longo G (2019). *Developing and Empirically Validating **flexplot**: A Tool for Mapping Statistical Analyses into Graphical Presentation*. Paper presented at the American Psychological Association, Chicago, IL.

Gigerenzer G (2004). "Mindless statistics." *The Journal of Socio-Economics*, **33**(5), 587–606. ISSN 1053-5357. doi:10.1016/J.SOCEC.2004.09.033. URL https://www.sciencedirect.com/science/article/pii/S1053535704000927.

Hansen CD, Chen M, Johnson CR, Kaufman AE, Hagen H (2014). *Mathematics and Visualization: Scientific Visualization Uncertainty, Multifield, Biomedical, and Scalable Visualization*. Springer, London. URL http://www.springer.com/series/4562.

Healy K, Moody J (2014). "Data Visualization in Sociology." *https://doi.org/10.1146/annurev-soc-071312-145551*, **40**(1), 105–128. doi:10.1146/ANNUREV-SOC-071312-145551. URL http://www.annualreviews.org/doi/10.1146/annurev-soc-071312-145551.

Levine SS (2018). "Show us your data: Connect the dots, improve science." doi:10.1017/mor.2018.19.

Nelson LD, Simmons JP, Simonsohn U (2018). "Psychology's Renaissance." *Annual Review of Psychology*, **69**, 511–545. doi:10.1146/annurev-psych-122216. URL https://doi.org/10.1146/annurev-psych-122216-.

Nosek BA, Ebersole CR, DeHaven AC, Mellor DT (2018). "The preregistration revolution." *Proceedings of the National Academy of Sciences*. ISSN 0027-8424. doi:10.1073/pnas.1708274114.

Otten JJ, Cheng K, Drewnowski A (2015). "Infographics And Public Policy: Using Data Visualization To Convey Complex Information." *Health Affairs*, **34**(11), 1901–1907. ISSN 0278-2715. doi:10.1377/hlthaff.2015.0642. URL http://www.healthaffairs.org/doi/10.1377/hlthaff.2015.0642.

Pashler H, Wagenmakers EJ (2012). "Editors' Introduction to the Special Section on Replicability in Psychological Science." *Perspectives on Psychological Science*, **7**(6), 528–530. ISSN 1745-6916. doi:10.1177/1745691612465253. URL http://journals.sagepub.com/doi/10.1177/1745691612465253.

RStudio Team (2016). *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA. URL http://www.rstudio.com/.

Tang Y, Horikoshi M, Li W (2016). "**ggfortify**: unified interface to visualize statistical results of popular R packages." *The R Journal*, **8**(2), 474–489.

Tay L, Parrigon S, Huang Q, LeBreton JM (2016). "Graphical Descriptives: A Way to Improve Data Transparency and Methodological Rigor in Psychology." *Perspectives on Psychological Science*, **11**(5), 692–701. ISSN 1745-6916. doi:10.1177/1745691616663875. URL https://doi.org/10.1177/1745691616663875.

Tukey JW, Tukey PA (1990). *Strips Displaying Empirical Distributions: I. Textured Dot Strips*. Bellcore.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Fourth edition. Springer, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Weissgerber TL, Milic NM, Winham SJ, Garovic VD (2015). "Beyond Bar and Line Graphs: Time for a New Data Presentation Paradigm." *PLoS Biology*, **13**(4). ISSN 15457885. doi:10.1371/journal.pbio.1002128.

Wickham H (2010). "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics*, **19**(1), 3–28. doi:10.1198/jcgs.2009.07098. URL https://vita.had.co.nz/papers/layered-grammar.pdf.

Wickham H (2011). "**ggplot2**." *Wiley Interdisciplinary Reviews: Computational Statistics*, **3**(2), 180–185. ISSN 1939-5108. doi:10.1002/wics.147.

Wickham H (2017). ***tidyverse**: Easily Install and Load the 'Tidyverse'*. R package version 1.2.1, URL https://CRAN.R-project.org/package=tidyverse.

Wickham H, Cook D, Hofmann H, Buja A (2011). "Journal of Statistical Software **tourr**: An R Package for Exploring Multivariate Data with Projections." *Technical Report 2*. URL http://www.jstatsoft.org/.

Wilke CO (2016). "**cowplot**: streamlined plot theme and plot annotations for **ggplot2**." URL http://www.cran.r-project.org/package=cowplot.

Wilkinson L (1999). "Dot plots." *The American Statistician*, **53**(3), 276–281. ISSN 0003-1305.

Xie Y, Allaire JJ, Grolemund G (2018). *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, Boca Raton, Florida. URL https://bookdown.org/yihui/rmarkdown.

Affiliation:

Dustin Fife
Rowan University
201 Mullica Hill Road Glassboro, NJ 08028
E-mail: fife.dustin@gmail.com

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd