UPDATE – June 2025: Some of the underlying packages have changed, and this guide no longer works out of the box. If you'd like to run the original code, see this post (or the README in the repo) for instructions on how to recreate the older package environment. TLDR: use `renv` and run `renv::restore()`.

UPDATE: It seems that GitHub pages is less than pleased with my liberal use of formatting in this lengthy blog post. To view this guide in it's intended form please check out the pdf version here.

Diverging and 100% stacked bar charts are an effective way to visualize Likert Scale data. In R there are two main packages -- *HH* and *likert* -- that turn Likert Scale data into pretty charts. I have used both extensively to make pretty plots and, personally, I like the *likert* package more because it works with *ggplot* objects and functions but I will run through a quick tutorial of both here.

Please click here to see the associated repo that contains all of the code and data needed to run these examples.

HH Package

Preparing Your Data

The *HH* package will only accept data in a summary table. With this option for data input, you must have a data frame that represents the Likert data in a pre-summarized form. Meaning, that the first column must be the items and the remaining columns are the levels for each item in an ordinal sequence.

I have elected to create my own summary table instead of summarizing existing long format data. However, **be aware that the *HH* package overloads tidyverse's `select()` function.** An easy fix for this is to specify the library via `dplyr::select()`.

Below is a sample summary table that I created. The `Neutral` column is omitted because it's just a vector of zeros.

Item	Strongly Disagree	Disagree	Agree	Strongly Agree
"Oatmeal Raisin is The Best Type of Cookie"	0.6	0.07	0.03	0.3
"Chocolate Chip is The Best Type of Cookie"	0.2	0.25	0.15	0.4
"Snickerdoodle is The Best Type of Cookie"	0.1	0.47	0.38	0.05

It doesn't matter what you name the first column. I named it `Item` to make the dataset play well with the *likert* package as well.

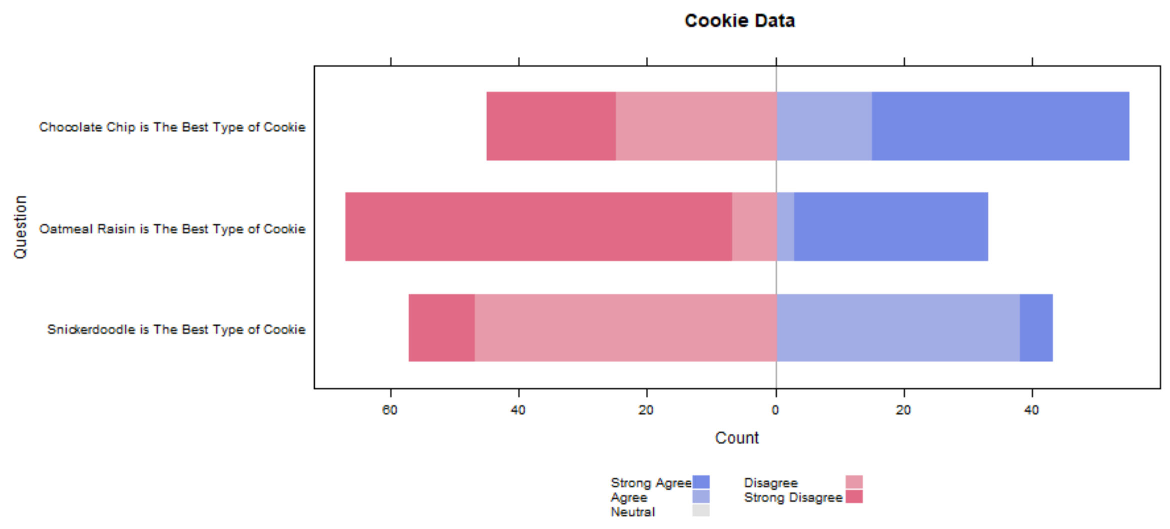
To graph the data simply use the `likert()` function:

> `likert(Item~., df, arg3,...)`

Where the `Item` is the column with the questions and the `.` means to sum each of the other columns for the graph. In fact `Item~.` is the same as typing `Item~Strong_Disagree+Disagree+neutral+Agree+Strong_Agree`. The argument `df` is the data frame you want to pull data from and `arg3` is a stand in for all the optional arguments you can add.

A simple graph would look like the following:

```r
load(file = "../data/sample-likert-data.rda") #our data, see likert-data-generation.Rmd for more info
likert(Item~., cookie_data, ReferenceZero=3,  ylab = "Question", main = list("Cookie Data", x=unit(.62, "npc")), auto.key = list(columns = 2, reverse.rows = T))
```



Note how I set the `ReferenceZero` to 3. This is because there are five levels in this graph ranging from "Strong Disagree" to "Strong Agree." Hence, the neutral is the third level (counting here, like in all of R, starts at 1). The `auto.key` argument is covered in the section entitled *Change Your Viz Via* `likert()` *Arguments* and the remaining optional arguments are to label the graph. The `main` argument is the title and accepts just a string for the title. However, if you want to center the title you need to use `list("GRAPH TITLE",x=unit(.7, "npc"))`. Where the `x` argument dictates where on the x-axis your title is placed. The domain is between [0,1).

Saving Your Graph

To save a graph created in the *HH* package you need to use functions such as `png()` . You can read the documentation for yourself here. Below is how I save images created via the *HH* package.

```r
p1 <- likert(Item~., cookie_data, ReferenceZero=3,  ylab = "Question", main = list("Cookie Data", x=unit(.62, "npc")), auto.key = list(columns = 2, reverse.rows = T))

p1 #show image

## Save Image
png("../imgs/HH_basic.png",
     height=720, width=1080)
p1
dev.off()
```

Note that `png` defaults to unit of pixels. If you wanted to use inches you would do the following:

```r
png("../imgs/HH_basic_in.png",
     height=5, width=7, units = "in", res = 720)
```

When using inches you need to specify the resolution via `res`.

Changing How Data Is Displayed

We can group data via:

> `likert(Item~. |grouping, df)`

Working with our cookie data frame, we can group the cookies via "Chunky" and "Smooth." We reflect this in our data by creating a vector that categorizes each cookie type as smooth or chunky and then using `cbind` to add this vector to the existing data frame.

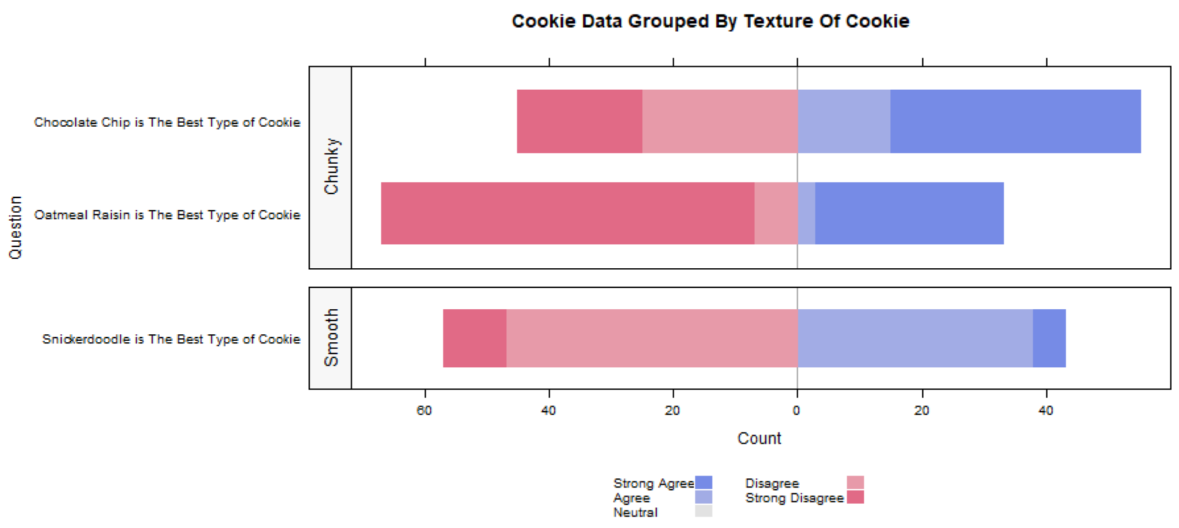
```r
## Create New df with Groups
type <- c("Chunky", "Chunky", "Smooth")
new_cookie_data <- cbind(cookie_data, type)
```

Our new data frame (with `Neutral` omitted) looks like this:

Item	Strongly Disagree	Disagree	Agree	Strongly Agree	type
"Oatmeal Raisin is The Best Type of Cookie"	0.6	0.07	0.03	0.3	"Chunky"
"Chocolate Chip is The Best Type of Cookie"	0.2	0.25	0.15	0.4	"Chunky"
"Snickerdoodle is The Best Type of Cookie"	0.1	0.47	0.38	0.05	"Smooth"

We then use the following code to plot the new data frame:

```r
## Plot Data
likert(Item~. | type, new_cookie_data, ReferenceZero=3, main = list("Cookie Data Grouped By Texture Of Cookie", x=unit(.6, "npc")), layout=c(1,2), auto.key = list(columns = 2, reverse.rows = T),
 scales=list(y=list(relation="free")), between=list(y=1), strip.left=TRUE, strip = FALSE,
 par.strip.text=list(cex=1.1, lines=2), ylab="Question")
```



There is a lot going on in the above code snippet. The following arguments are covered in the next section:

- `layout`
- `between`
- `strip.left`
- `strip`
- `auto.key`

This just leaves us with `scales=list(y=list(relation="free"))` as the only new parameter related to displaying grouping data. A few things are nested within the `scales` argument; the gist is that `scales` expects its data packaged in multi-dimensional vector form and `list` does that without adding to much extra visual clutter. Within that, we want to set the y-axis scale to "free" so that each of the panels in the graph only contain scale elements for the data that they display.

Change Your Viz Via `likert()` Arguments

Color

To change what colors are used on the graph use the following argument:

> `col=c("level 1","level 2,...,"level n")`

Where each color corresponds to the levels in ordinal order from the top row of your summarized data frame. To generate your color pallets, use the diverging pallets in the *RColorBrewer* package. If you are new to the package I highly recommend checking that out here. With the *RColorBrewer* package, the vector is supplied for you. Thus to change the colors on your graph all you need is:

> `col = brewer.pal(n,"pName")`

Percentages

The following argument displays the Likert graph in a percentage

> `as.percent=TRUE`

It also, helpfully, adds the n value for each corresponding row of the data.

Legend Shape

The following argument modifies how the legend is displayed

> `auto.key = list(columns = 1, reverse.rows = T)`

Where `columns` is the number of columns you want to see in the legend. `reverse.rows` is a quick way to flip which end of the Likert scale (1 or n) appears at the top of the legend.

Augmenting How Graphs With Multiple Plots Are Labeled

By default, the *HH* package will put a label on each of the plots within an image.

The following argument turns *off* the label at the top of each plot:

> `strip = FALSE`

The following argument turns *on* the label at the left side of each plot:

> `strip.left = TRUE`

The following argument augments how the labels are presented.

> `par.strip.text = list(cex=1.1, lines=2)`

The `cex` argument dictates how large the text is where a value of 1 is normal size. The `lines` argument dictates how many lines there are in the label. Two lines gives a left label a nice buffer. This functionality is built off of the *lattice* package. There are more ways to augment the labels and you can learn more about it here.

Arranging Plots

The following argument changes how the plots are arranged:

> `layout = c(1,2)`

`layout` expects the format (column, row)

To change the space between the plots use the following code:

> `between = list(y=1)`

Where `y` changes the space on the y-axis between plots.

likert package

My workflow with the *likert* package is to use the `likert` function nested inside of the `plot` function. Like this:

> `plot(likert(arg1,arg2), arg3, ...)`

Preparing Your Data

The *likert* package will accept two types of input data:

1. data in a long format data frame or tibble
2. a summary table of the Likert data

Long Format Data

When your data is in the long format, ensure that each variable within the data frame is a factor. These variables will be their own category on the graph. Within each of these categories you can group the data by some other factor variable -- the nth variable in the data frame where $n-1$ variables are used as data for graphing.

To graph the whole data frame:

> plot(likert(myData), arg2, ...)

To create a graph with groupings within each category where the nth column is the grouping variable:

> `plot(likert(myData[,c(1:n-1)], grouping = myData[,n]), arg3, ...)`

I tried to execute this with tidy via `select()` but the *likert* package refused to comply. So, I am using some old fashioned indexing. Recall that data frames can be indexed [row, column] and if you want to grab a bunch of columns you need to put it in vector `c(start:stop)` where start and stop are inclusive. Meaning, in the above code snippet we are grabbing the first column (indexing starts at 1 in R!) through the second to last column for graphing and using the last column for grouping.

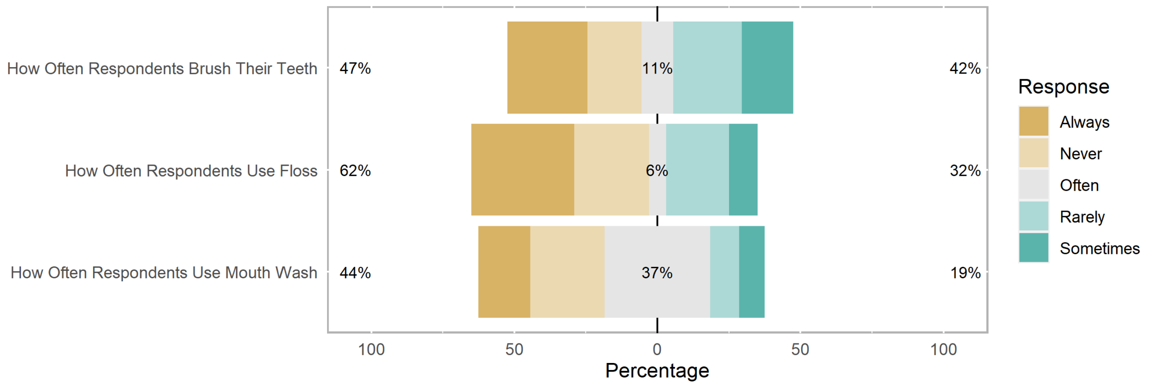
There may be a more elegant way to do it but here is how I create my data frames for plotting:

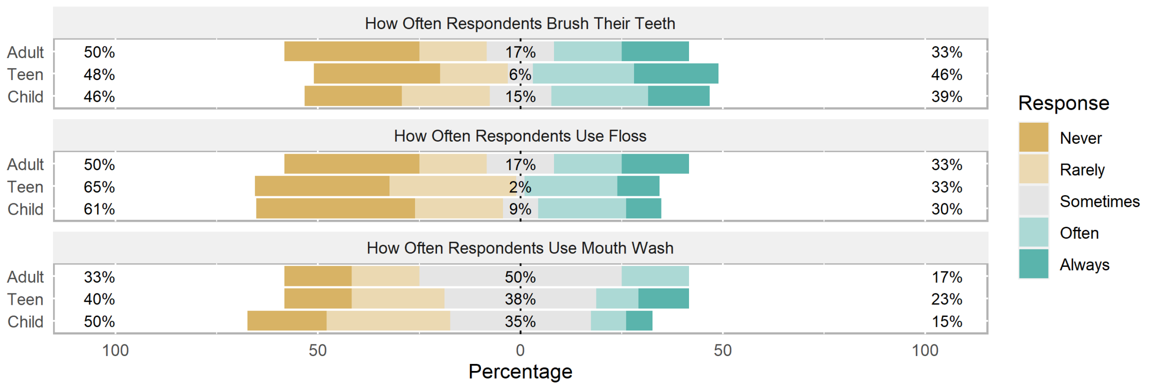
```r
load(file = "../data/sample-likert-data.rda") #our data, see likert-data-generation.Rmd for more info

## Rename Cols
dental_hyg  <- dental_hyg  %>%
  rename("How Often Respondents Use Mouth Wash" = mouth_wash,
         "How Often Respondents Brush Their Teeth" = tooth_brush,
         "How Often Respondents Use Floss" = floss)

## New df for not grouped
dh <- dental_hyg %>% select(-age)

## Not Grouped
# plot(likert(dh)) # basic not grouped
plot(likert(dh), legend.position="right")

## Grouped
```

```
# plot(likert(dental_hyg[,c(1:3)], grouping = dental_hyg[,4])) # basic grouped
plot(likert(dental_hyg[,c(1:3)], grouping = dental_hyg[,4]), legend.position="right")
```

Notice here that I rename the variables in the data frame because their names will be displayed in the Likert graph. I then use tidyverse functions to create a second data frame of the data I am interested in graphing.

The resulting plots for this code are a slightly prettier version of the basic likert plot as the legend has been moved. You can read more about this in the section entitled *Change Your Viz Via* `plot()` *Arguments*.





Summarized Data

With this option for data input, you must have a data frame that represents the Likert data in a pre-summarized form. Meaning, that the first column must be the items and the remaining columns are the levels for each item.

Item	Strongly Disagree	Disagree	Agree	Strongly Agree
"Oatmeal Raisin is The Best Type of Cookie"	0.6	0.07	0.03	0.3
"Chocolate Chip is The Best Type of Cookie"	0.2	0.25	0.15	0.4
"Snickerdoodle is The Best Type of Cookie"	0.1	0.47	0.38	0.05

It is crucial that the variable name for your first column is called `Item`. The entries in this column will be the label for the data contained in the row on the graph. Moreover, the data frame needs low, neutral, and high columns. (Neutral has been omitted from the table but is in the code snippet below).

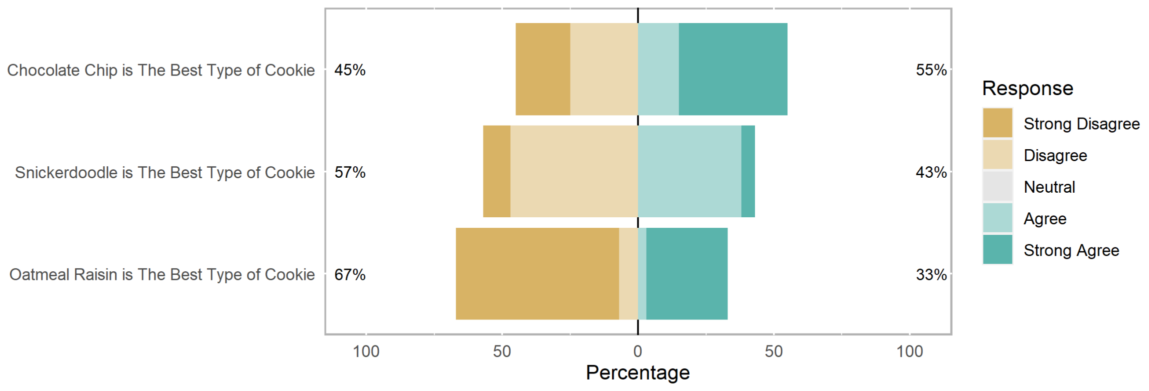
When calling the `likert` function, you will need to specify that this is pre-summarized data via the keyword argument `summary`. So, your function call should be:

> `plot(likert(summary = myData), arg3, ...)`

Put all together it looks like this:

```r
## Create df
Item <- c("Oatmeal Raisin is The Best Type of Cookie", "Chocolate Chip is The Best Type of Cookie", "Snickerdoodle is The Best Type of Cookie")
strong_disagree <- c(60, 20, 10)
disagree <- c(7, 25, 47)
neutral <- c(0,0,0)
agree <- c(3, 15, 38)
strong_agree  <- c(30, 40, 05)
df <- data.frame(Item, strong_disagree, disagree, neutral, agree, strong_agree)

## Rename Cols (for legend)
df <- df  %>%
  rename("Strong Disagree" = strong_disagree,
          "Disagree" = disagree,
          "Agree" = agree,
         "Strong Agree" = strong_agree)

## Basic Plot (not image below)
plot(likert(summary = df))

## Pretty Plot (Image Below)
plot(likert(summary = df), plot.percent.neutral=FALSE, legend.position="right")
```

The code under "Pretty Plot" yields the graph below. This graph is the same as the "Basic Plot" except the legend has been moved and the neutral percents have been suppressed. You can read more about this in the section entitled Change Your Viz Via `plot()` Arguments.



Changing How Data Is Displayed

Regardless of which type of data you feed into the `likert` function, you may want to modify how the levels of your categorical data are presented. Most of the code snippets below will be working with the `dh` data frame created earlier in the post. As a reminder, the creation of `dh` looks like this:

```r
load(file = "../data/sample-likert-data.rda") #our data, see likert-data-generation.Rmd for more info

## Rename Cols
dental_hyg  <- dental_hyg  %>%
  rename("How Often Respondents Use Mouth Wash" = mouth_wash,
          "How Often Respondents Brush Their Teeth" = tooth_brush,
          "How Often Respondents Use Floss" = floss)

## New df for not grouped
dh <- dental_hyg %>% select(-age)
```

Changing The Color Scheme and Color Ordering On The Graph

Personally, I am a HUGE fan of the *RColorBrewer* package because it provides pre-generated color pallets that are aesthetically pleasing and do not mislead/confuse the viewer. datanovia.com has a wonderful writeup on the color pallets and how to use functions within *RColorBrewer*. If you are new to the package I highly recommend checking that out here.

In case the website is down or the link breaks in the future, the gist is:

- Use `scale_fill_brewer()` for box plots, bar charts, and other shapes with area to fill
- Use `scale_color_brewer()` for points, lines, and other shapes with no area to fill
- You can view all of the palettes via: `display.brewer.all()`
    - Looking at just the colorblind friendly palettes via `display.brewer.all(colorblindFriendly = TRUE)` is highly recommended
- You can view a single palettes via: `display.brewer.pal(n, pName)` where `n` is the number of colors and `pName` is the name of the palette

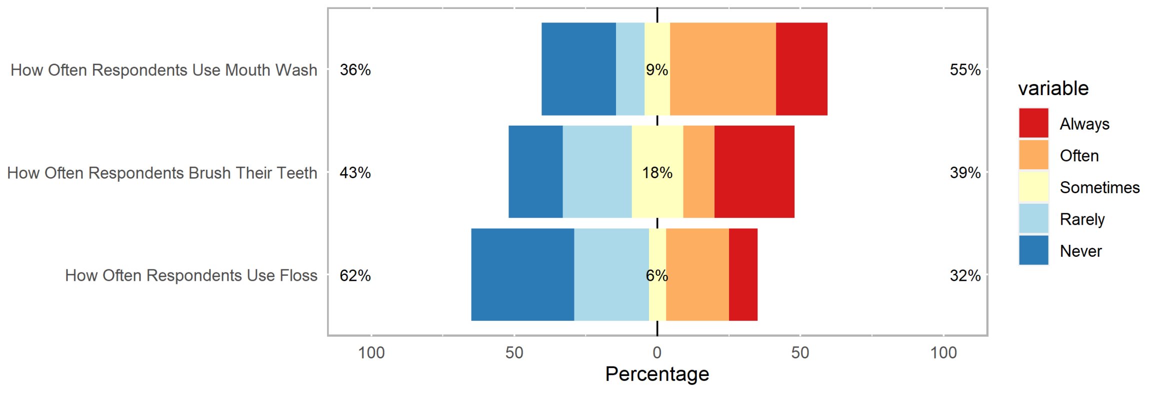
For the *likert* package the I use the following line to manipulate colors on the graph:

> `scale_fill_manual(values = brewer.pal(n,pName), breaks = c("level 1","level 2",...,"level n"))`

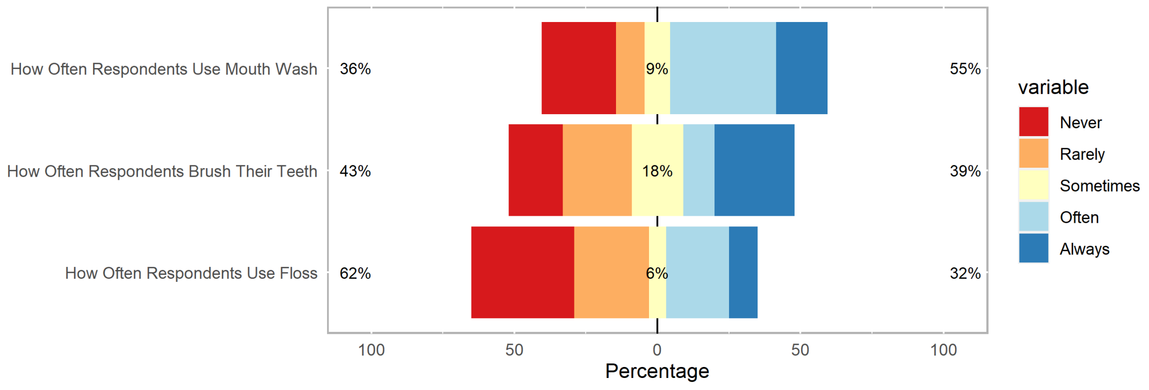
Let's look at an example where we create two graphs. The first where "Never" is blue and "Always" is red and the second where "Never" is red and "Always" is blue:

```r
## Blue-Never, Red-Always
plot(likert(dh), legend.position="right") +
  scale_fill_manual(values = brewer.pal(n=5,"RdYlBu"), breaks = c("Always", "Often", "Sometimes","Rarely", "Never")) #order and color the likert boxes
```

Note that `n` must be equal to the number of levels in your categorical variable. We use the `breaks` keyword argument to create a one-to-one mapping between the n colors and the levels of the categorical variable. Thus, the length of the `breaks` vector must be equal to `n`.



```r
## Red-Never, Blue-Always
plot(likert(dh), legend.position="right") +
  scale_fill_manual(values = brewer.pal(n=5,"RdYlBu"), breaks = c("Never",
"Rarely", "Sometimes", "Often", "Always")) #order and color the likert boxes
```



Before we move onto the next section, observe how the legend's order matches the vector passed into the `breaks` keyword argument.

Reversing The Levels In The Legend

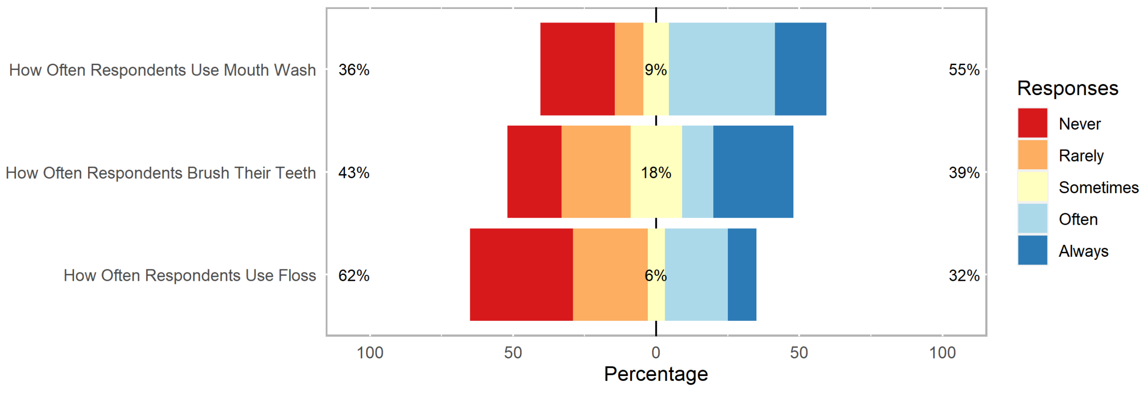
In the previous section we observed how the levels in the legend have the same order as the vector passed into the `breaks` keyword argument. We can change this via the *ggplot* function `guides`.

> guides(fill = guide_legend(title="Responses", reverse = TRUE))

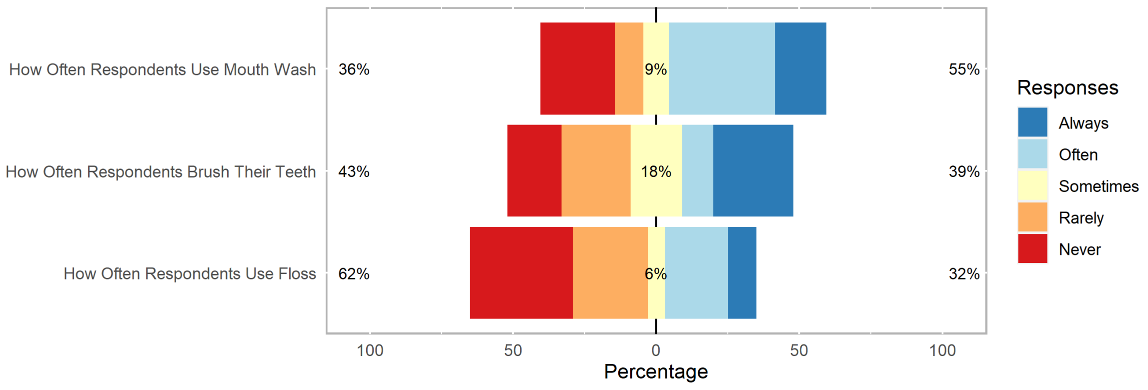
Technically, we only need `reverse = TRUE` but we can also set the guide legend here as well since I won't use the *ggplot* function `labs()` in these code snippets to reduce visual clutter.

Observe how the legend changes between these two code snippets:

```r
## Red-Never, Blue-Always
plot(likert(dh), legend.position="right") +
  scale_fill_manual(values = brewer.pal(n=5,"RdYlBu"), breaks = c("Never",
"Rarely", "Sometimes", "Often", "Always")) +
  guides(fill = guide_legend(title="Responses", reverse = FALSE)) #to reverse
the order in the legend
```



```
## Red-Never, Blue-Always
plot(likert(dh), legend.position="right") +
  scale_fill_manual(values = brewer.pal(n=5,"RdYlBu"), breaks = c("Never",
"Rarely", "Sometimes", "Often", "Always")) +
  guides(fill = guide_legend(title="Responses", reverse = TRUE)) #to reverse
the order in the legend
```



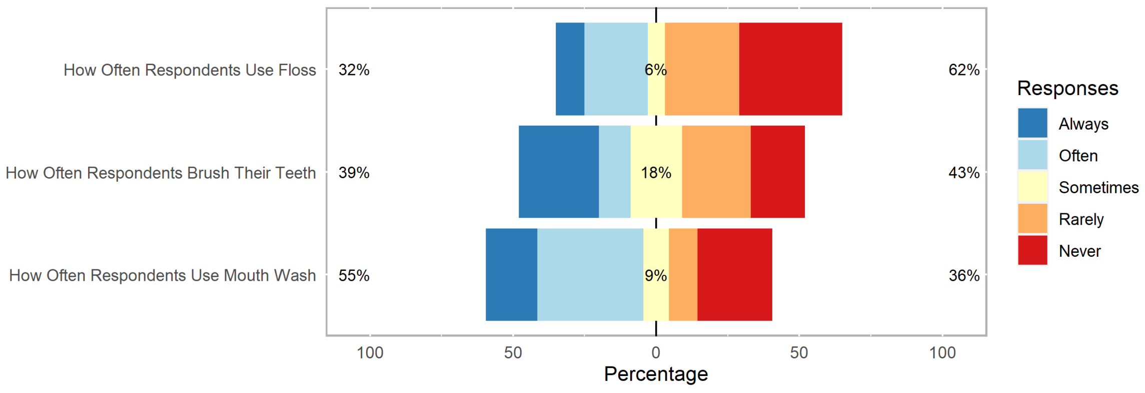
Reversing The Levels On The Graph

If, for whatever reason, you find yourself needing to reverse the levels on your graph -- meaning that you want the graph to flip around the midpoint line -- there is a simple way to accomplish this. Simply create a new data frame via:

> `df.reverse <- reverse.levels(dh)`

The result of this can be seen in the example below which reverses the last image in the *Reversing The Levels In The Legend* section.

```
## Reverse Levels
dh.reverse <- reverse.levels(dh)

## Red-Never, Blue-Always reversed
plot(likert(dh.reverse), legend.position="right") +
  scale_fill_manual(values = brewer.pal(n=5,"RdYlBu"), breaks = c("Never",
"Rarely", "Sometimes", "Often", "Always")) +
  guides(fill = guide_legend(title="Responses", reverse = TRUE))
```



Note that the reversed data frame also reversed the order of which the categorical variables are displayed. The next section will discuss how to reorder the display of the categorical variables.

Changing The Ordering Of The Categorical Variables (Not Grouped)

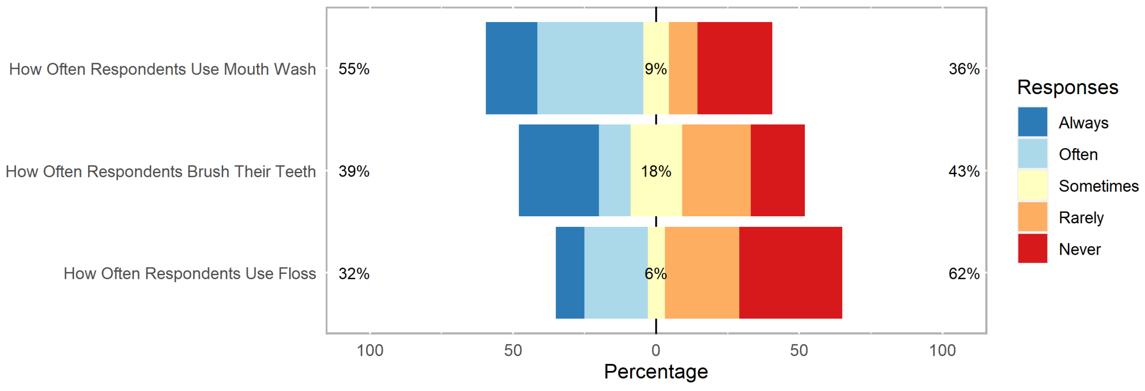
Sometimes you want to change the order of the categorical variables. Take, for example, the image in the last section where the order of the categorical variables is reversed from all other images. This can be rectified with a keyword argument to `plot()`

> `group.order = names(df)`

Recall that `names()` returns a character vector with the names of each variable in a data frame. Starting with the first column and ending with the last column. Thus, the ordering of the variables in your data frame will match the ordering on the graph. If this is not what you want, simply pass `group.order` a character vector with the names of each variable in your data frame arranged in the order you want them displayed on the graph.

With this in mind, let's now fix the incorrect ordering of the reversed graph from the last section.

```r
## Reverse Levels
dh.reverse <- reverse.levels(dh)

plot(likert(dh.reverse), group.order=names(dh), legend.position="right") +
#group.order changes the order of the variables
  scale_fill_manual(values = brewer.pal(n=5,"RdYlBu"), breaks = c("Never",
"Rarely", "Sometimes", "Often", "Always")) +
  guides(fill = guide_legend(title="Responses", reverse = TRUE))
```



Changing The Ordering Of The Categorical Variables (Grouped)

You may also want to change the order of the categorical variables for grouped data. There is a pretty quick fix for this -- simply reorder the columns in the data frame to match the order you want to see. This can be accomplished by reordering the columns by index.

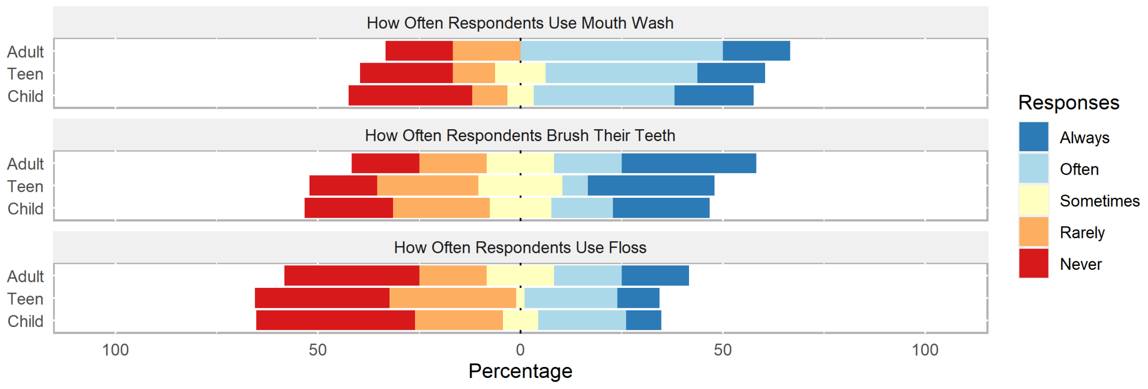
The following code snippet shows an example where you wanted the graph to display the variable at column index 3, then the variable at column index 2, then the variable at column index 1, and leave the last column untouched for grouping:

> df <- df[, c(3,2,1,4)] #an example where there are four columns in the data frame.

If you are confused by this syntax please refer to the Long Format Data section.

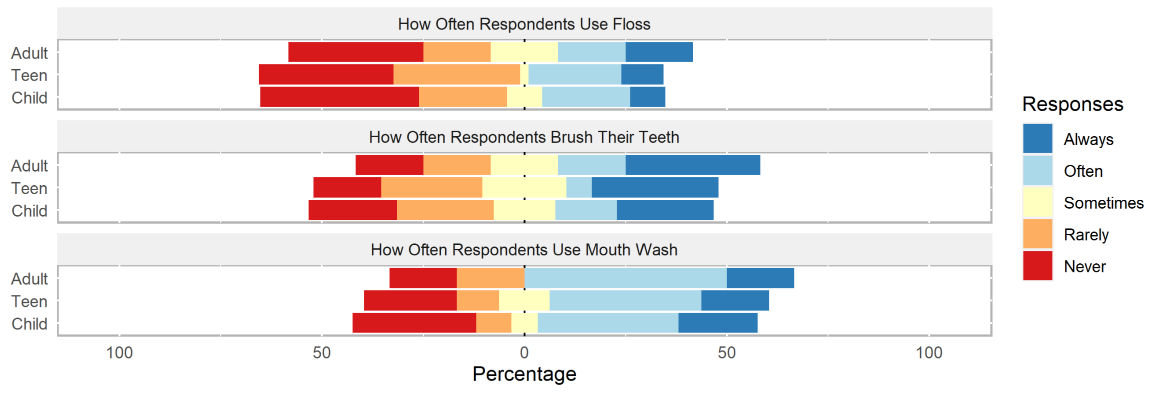
Now let's see a real example with our data. First, below is the unmodified graph.

```r
## Load dental_hyg
load(file = "../data/sample-likert-data.rda")

## Normal Grouped Plot
plot(likert(dental_hyg[,c(1:3)], grouping = dental_hyg[,4]), plot.percent.low = FALSE, plot.percent.high = FALSE, plot.percent.neutral = FALSE, legend.position="right") +
  scale_fill_manual(values = brewer.pal(n=5,"RdYlBu"), breaks = c("Never", "Rarely", "Sometimes", "Often", "Always")) +
  guides(fill = guide_legend(title="Responses", reverse = TRUE))

```



Now let's swap the first and third group so it goes "Floss" first "Mouthwash" last.

```r
## Load dental_hyg
load(file = "../data/sample-likert-data.rda")

## Reorder Levels
dental_hyg.reorder <- dental_hyg[,c(3,2,1,4)] #swap col 1 and 3

## Reorderd Group Plot
plot(likert(dental_hyg.reorder[,c(1:3)], grouping = dental_hyg.reorder[,4]), plot.percent.low = FALSE, plot.percent.high = FALSE, plot.percent.neutral = FALSE, legend.position="right") +
  scale_fill_manual(values = brewer.pal(n=5,"RdYlBu"), breaks = c("Never", "Rarely", "Sometimes", "Often", "Always")) +
  guides(fill = guide_legend(title="Responses", reverse = TRUE))
```



As desired, the groups switched.

There is, however, one very important caveat to this technique. There is a bug in the *likert* package that changes the variable ordering from column ordering to alphabetical if any of the percent arguments are set to `TRUE`. These arguments are:

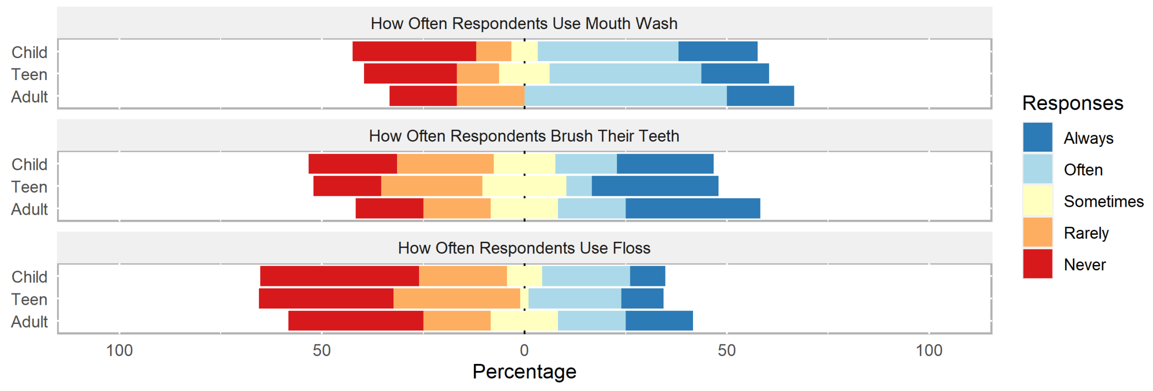
- `plot.percent.low`
- `plot.percent.high`
- `plot.percent.neutral`

This bug has been documented here. Unfortunately, all three of these parameters are set to `TRUE` by default so you must set them all to false each time you want to make a grouped plot with a custom ordering of the groups.

Reverse The Ordering Of Grouping Within Facets (grouped data only)

To reverse the order of the variables within each group you must simply provide the `grouping` argument a version of the column with the grouping data with reversed levels. Observe:

```
## Reverse Levels
dental_hyg.reverse <- reverse.levels(dental_hyg)

plot(likert(dental_hyg[,c(1:3)], grouping = dental_hyg.reverse[,4]),
plot.percent.low = FALSE, plot.percent.high = FALSE, plot.percent.neutral =
FALSE, legend.position="right") + #note grouping = dental_hyg.reverse
  scale_fill_manual(values = brewer.pal(n=5,"RdYlBu"), breaks = c("Never",
"Rarely", "Sometimes", "Often", "Always")) +
  guides(fill = guide_legend(title="Responses", reverse = TRUE))
```

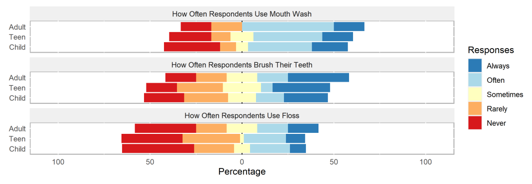
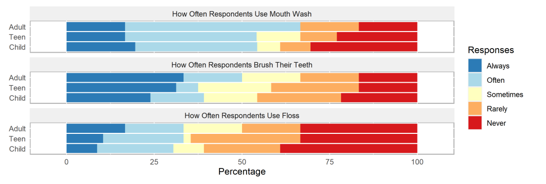


Now the ordering is "Child", "Teen", "Adult" instead of "Adult" , "Teen", "Child".

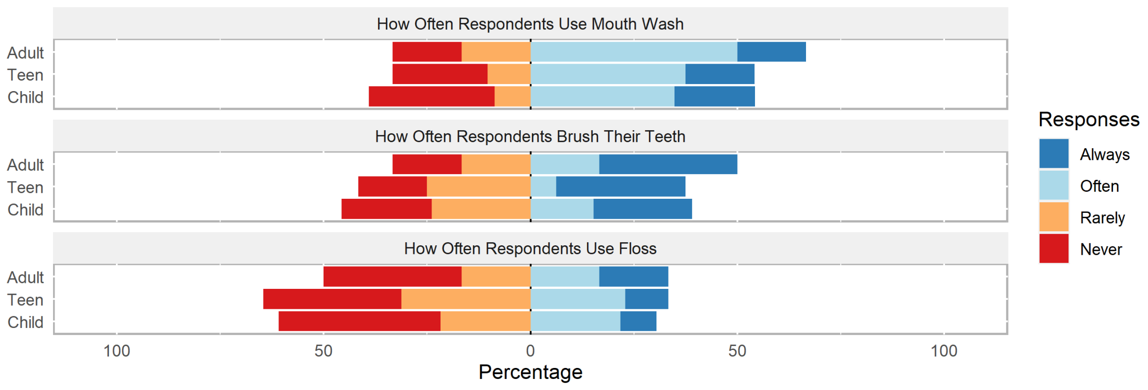
This technique can be generalized to reordering instead of reversing if you simply refactor the grouping variable, but I leave that as an exercise to the reader :)

Diverging VS 100% Stacked Bar Chart

There is an [excellent blog post from Datawrapper](#) making the case against diverging stacked bar charts and offering 100% stacked bar charts as the better alternative. The article argues that "the main problem with diverging bars, however, is comparability." On the whole I agree; in the case that there are two or four levels on the Likert scale, however, I think the diverging stacked bar chart is the better choice.[1] As you can see, the image below on the right is better for easy comparison than the image on the left.

Incorrect	Correct
	

Moreover, the diverging Likert scale below is also acceptable as there is an even number of levels.



By default `plot()` assumes that you want a diverging stacked bar chart. To toggle the 100% stacked bar chart, you must use the following key word argument:

> `plot(likert(myData), centered = FALSE)`

In the case that you are using a diverging bar chart and the centerline isn't quite where you want it to be, you may use the `center` key word argument to move the centerline.

> `plot(likert(myData), center = X)` (where x ranges between [low+.5, high-.5] in .5 increments)

You cannot have `centered = TRUE` and `center = x` in `plot()` at the same time.

Change Your Viz Via `plot()` Arguments

Percentages

By default the *likert* package slaps percentages on each of your graph. These percentages display how much for your data is coded as low, neutral, or high. The following code snippet has all three percentages turned off:

> `plot(likert(myData), plot.percent.neutral=FALSE, plot.percent.low=FALSE, plot.percent.high=FALSE)`

To ensure that the percentage calculation of high and low are correct, you must correctly set the neutral level via the `center` keyword argument in the `plot()` function.

> `plot(likert(myData), center = X)` (where x ranges between [low+.5, high-.5] in .5 increments)

If you wish to remove the neutral values from your graph you may use the following code snippet. This will not affect the percentage calculations nor change where the center of the graph is. The low values and high values will still be split on the line.

> `plot(likert(myData),include.center=FALSE)`

Legend Position

To position the legend where you want it, use the following code:

> `plot(likert(myData), legend.position="right")`

You can set the keyword argument `legend.position` to "right", "left", "top", or "bottom". Use this in combination with the ggplot method `+ guides(fill = guide_legend(title="LEGEND TITLE", reverse = TRUE))` to title and reverse the order of the legend.

Change the Label Color

When you have facets on your plot you can change the color behind the label of each category on the graph with the following code snippet:

> `plot(likert(myData[,c(1:n-1)], grouping = myData[,n]), panel.strip.color = "#FFA500")`

As mentioned earlier, the *likert* package plays well with *ggplot* so you can modify the plot with theme and scale elements just like a normal ggplot object. For example to change the legend title and order of the elements in the legend you could do:

> `plot(likert(myData)) + guides(fill = guide_legend(title="LEGEND TITLE", reverse = TRUE))`

You can also modify the visualization with arguments to the `plot()` function.

Links Referenced

HH Package

- https://cran.r-project.org/web/packages/HH/HH.pdf
- https://cran.r-project.org/web/packages/HH/HH.pdf
- https://stat.ethz.ch/R-manual/R-devel/library/lattice/html/strip.default.html

likert Package

- https://github.com/jbryer/likert
- https://github.com/jbryer/likert/blob/master/demo/likert.R
- https://lmudge13.github.io/sample_code/likert_graphs.html
- https://rpubs.com/m_dev/likert_summary

---

1. In this blog post I elected not to show 100% bar charts thus far because I didn't want to add too much too quickly. ↩