In this post, We will walk you through some examples that show off the major features of the ggforce package. The main goal is to share a few ideas about customizing visualizations that you may find useful in your everyday work.

The ggforce package is an extension to ggplot2 developed by [Thomas Pedersen](https://github.com/thomasp85/). Thanks to ggforce, you can enhance almost any ggplot by highlighting data groupings, and focusing attention on interesting features of the plot. The package contains geoms, stats, facets, and other ggplot functions. Among such functions, there are some for marking the [convex hull](https://en.wikipedia.org/wiki/Convex_hull) of a set of points, jittering data, and creating [Voronoi plots](https://en.wikipedia.org/wiki/Voronoi_diagram).

**Base ggplot**

The examples in this article will use data from the nycflights13 package. Most of the examples will build on the same basic ggplot that visualizes airports by geographical location. I am using this data set because it makes it easy to plot x/y coordinates without having to remember what they “mean”. This basic plot will be saved to a variable, and then that variable will be used as the base of the examples of enhancing the visualization using ggforce

library(tidyverse)

library(ggforce)

library(nycflights13)

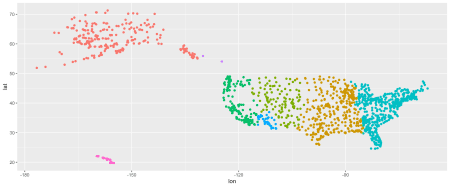
p <- airports %>%

filter(lon < 0, tzone != "\\N") %>%

ggplot(aes(lon, lat, color = tzone)) +

geom\_point(show.legend = FALSE)

p



**Make your mark with ggforce**

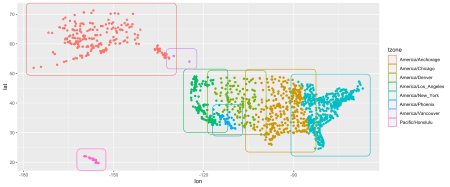
I have long been waiting for an easy way to draw an outline around groups of data. The geom\_mark\_...() family of functions does exactly that. There are four mark functions in ggforce, all different based on the shape they draw around the group:

1. geom\_mark\_circle()
2. geom\_mark\_ellipse()
3. geom\_mark\_hull()
4. geom\_mark\_rect()

Let’s start with geom\_mark\_rect(); it will draw a rounded rectangle around each time zone group.

p +

geom\_mark\_rect()



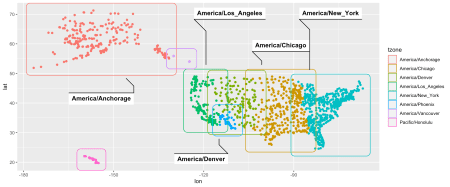
Like magic! The rectangles look amazing, even without modifying any arguments. Of course, more customization is possible via setting arguments. In this post, I will review some of the many great arguments available in ggforce functions, but I don’t want to rob you of the fun of trying it yourself and discovering all of the different options.

**Label, and an arrow!**

This next addition to our plot deserves its own subheading. Adding a label and an arrow pointing to a group would typically be a major undertaking. Without ggforce, this would require manually adding both the text and the arrow to the ggplot. But, with geom\_mark it is a simple as setting the label argument. So, without further ado, here is the label argument in action:

p +

geom\_mark\_rect(aes(label = tzone))



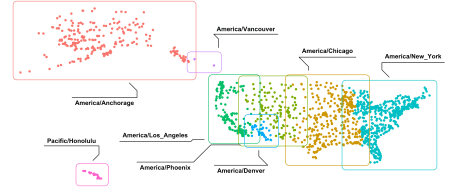
The labels and arrows are not only drawn, but they are also placed in an optimized location. In addition, the position will recalculate if the plot is re-sized! There are too many little details about this label argument to mention. The backdrop is automatically white, the indicator is not really an arrow, it is a simple line that also underlines the text, so it is easy for the eye to know which group belongs to which label.

It is now easy to finalize the plot by resetting the theme, and again suppressing the legend using show.legend.

p +

geom\_mark\_rect(aes(label = tzone), show.legend = FALSE) +

theme\_void()



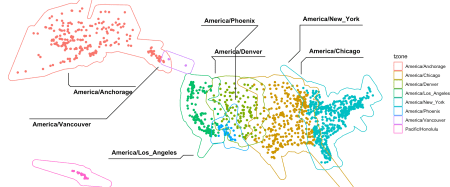
**Hull-k, enhance!**

There are many cases where drawing a rectangle or circle around the groups is not ideal, or even preferable. The geom\_mark\_hull() essentially traces a more complex polygon around the shape of the outline of the group.

p +

geom\_mark\_hull(aes(label = tzone)) +

theme\_void()

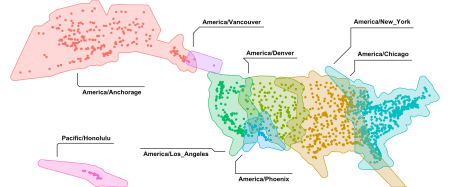


Again, without adding any arguments to the function, the traced outline already looks wonderful. Another option to add now is fill. And since the legend table is now redundant, it can be suppressed by setting show.legend to FALSE.

p +

geom\_mark\_hull(aes(label = tzone, fill = tzone), show.legend = FALSE) +

theme\_void()



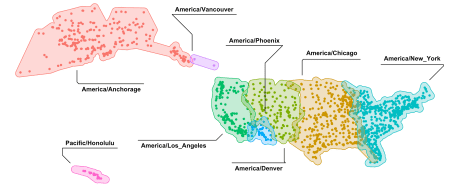
Notice that the fill color is not totally opaque; by default, ggforce has set the translucency lower to make sure that the dots are visible. This is something that I would have done anyway, usually by adding the alpha argument. In this case, it saves having to remember to add that argument.

Another adjustment that I thought was important for this plot was to modify the size of the hull, to change the padding around the outline of the group. The expand argument controls this aesthetic; it is possible to change it using the units() command.

p +

geom\_mark\_hull(aes(label = tzone, fill = tzone), show.legend = FALSE, expand = unit(3, "mm")) +

theme\_void()



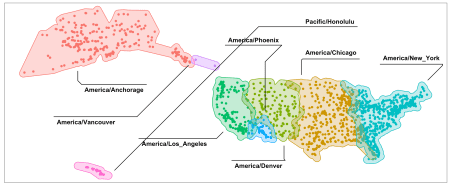
**Axe theme\_void()**

To finalize plots such as this one, it is necessary to remove most components from the default theme. Usually, theme\_void() does the trick. For printed or online articles with white backgrounds, which is essentially all of them, it is often hard to determine the margins of the plot. theme\_no\_axes() provides a great compromise by removing all but the one element.

p +

geom\_mark\_hull(aes(label = tzone, fill = tzone), show.legend = FALSE, expand = unit(3, "mm")) +

theme\_no\_axes()

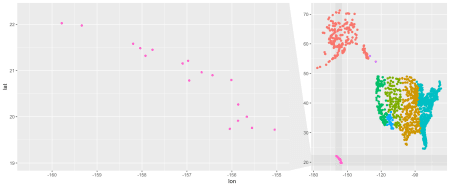


**Another facet of ggforce, and it’s magnify-cent**

It is common to produce two plots, one to show the full picture, and another to magnify or focus on a specific area. With facet\_zoom(), it is incredibly easy to show “macro” and “micro” in one plot by using the same xlim and ylim arguments to focus on an area of a plot.

p +

facet\_zoom(xlim = c(-155, -160.5), ylim = c(19, 22.3))

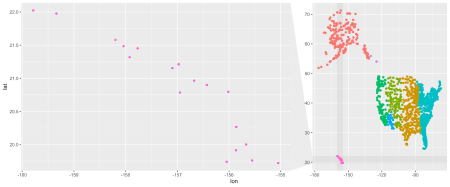


**Skip the coordinates**

Another cool feature of facet\_zoom() is the ability to set the zoom region based on a row selection. To do this, simply pass an expression that you would use in a function such as filter() to the facet. So instead of using coordinates, I just tell the facet to zoom on anything that has a Pacific/Honolulu time zone.

p +

facet\_zoom(xy = tzone == "Pacific/Honolulu")



**Putting it all together, with three lines of code**

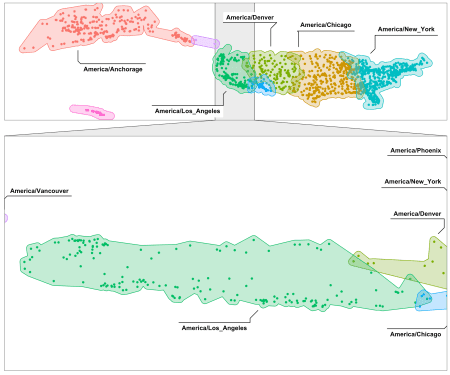
Using what has been covered so far, it is easy go from a very simple point plot to a sophisticated and nice-looking visualization with just three lines of code, thanks to ggforce.

p +

geom\_mark\_hull(aes(label = tzone, fill = tzone), show.legend = FALSE, expand = unit(3, "mm")) +

theme\_no\_axes() +

facet\_zoom(x = tzone == "America/Los\_Angeles")



**“What is a Voronoi?”**

This section title is based on my first reaction when I heard the word “Voronoi”. I have since learned about it, and can see why the [Voronoi Diagram](https://en.wikipedia.org/wiki/Voronoi_diagram) can be useful for very specific use cases. The good news is that if you encounter one of those use cases, you know that it is easy to draw it up in ggplot using geom\_voronoi\_segment().

The idea behind a Voronoi diagram is to split the area of the plot into as many sections as there are points. Unlike a grid or heat map, Voronoi draws custom shapes for each point based on the proximity to other points. It returns a plot that looks like stained glass. This can be good to determine the closest point inside each area. For example, a retailer can use it to see the area their store locations cover, and can help them make decisions to optimize their location based on the size of each Voronoi shape.

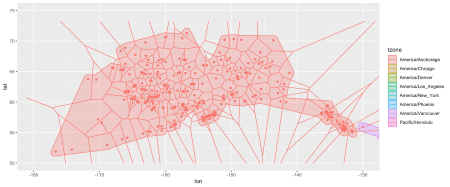
The following example will focus on airports in Alaska. The ggplot will zoom into that state’s general location, and then trace a hull shape. The hull will provide a quasi-map overlay. The final step is to add the Voronoi diagram layer by calling the function: geom\_voronoi\_segment()

p +

geom\_mark\_hull(aes(fill = tzone), expand = unit(3, "mm")) +

coord\_cartesian(xlim = c(-130, -180), ylim = c(50, 75)) +

geom\_voronoi\_segment()



**Parallel to alluvial**

The geom\_parallel... functions allow visualizing interactions between categorical variables. The implementation is generic enough to create Sankey or alluvial charts.

For this, I will use the Manufacturer and Engine data from the planes table inside nycflights13. In this case, some simple data preparation is needed first.

prep\_planes <- planes %>%

filter(year > 1998, year < 2005) %>%

filter(engine != "Turbo-shaft") %>%

select(manufacturer, engine) %>%

head(500)

prep\_planes

## # A tibble: 500 x 2

## manufacturer engine

##

## 1 EMBRAER Turbo-fan

## 2 AIRBUS INDUSTRIE Turbo-fan

## 3 AIRBUS INDUSTRIE Turbo-fan

## 4 EMBRAER Turbo-fan

## 5 AIRBUS INDUSTRIE Turbo-fan

## 6 AIRBUS INDUSTRIE Turbo-fan

## 7 AIRBUS INDUSTRIE Turbo-fan

## 8 AIRBUS INDUSTRIE Turbo-fan

## 9 AIRBUS INDUSTRIE Turbo-fan

## 10 EMBRAER Turbo-fan

## # … with 490 more rows

**Prep for plotting with one line**

The gather\_set\_data() is a convenience function that, just like gather(), creates a single line for each combination of categorical variables. The table contains three new columns – id, x, and y – which contain the combinations that each new row represents, and the row ID number from the original table.

prep\_planes %>%

gather\_set\_data(1:2)

## # A tibble: 1,000 x 5

## manufacturer engine id x y

##

## 1 EMBRAER Turbo-fan 1 manufacturer EMBRAER

## 2 AIRBUS INDUSTRIE Turbo-fan 2 manufacturer AIRBUS INDUSTRIE

## 3 AIRBUS INDUSTRIE Turbo-fan 3 manufacturer AIRBUS INDUSTRIE

## 4 EMBRAER Turbo-fan 4 manufacturer EMBRAER

## 5 AIRBUS INDUSTRIE Turbo-fan 5 manufacturer AIRBUS INDUSTRIE

## 6 AIRBUS INDUSTRIE Turbo-fan 6 manufacturer AIRBUS INDUSTRIE

## 7 AIRBUS INDUSTRIE Turbo-fan 7 manufacturer AIRBUS INDUSTRIE

## 8 AIRBUS INDUSTRIE Turbo-fan 8 manufacturer AIRBUS INDUSTRIE

## 9 AIRBUS INDUSTRIE Turbo-fan 9 manufacturer AIRBUS INDUSTRIE

## 10 EMBRAER Turbo-fan 10 manufacturer EMBRAER

## # … with 990 more rows

The ggplot is primed with x for x, and then new aesthetics: id, split and value. For id, we pass the id column, split takes y, and finally, value is fixed to 1. The value is used to express the amount of “thickness” to add to that particular relationship; using 1 means that all combinations are weighted the same. At this point, the only argument to pass geom\_parallel\_sets() will be the color fill; in this case we will use engine.

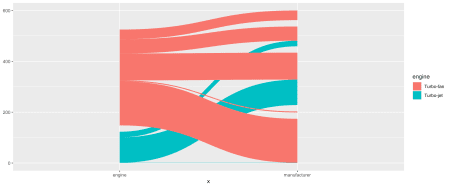
**Plotting with parallel**

prep\_planes %>%

gather\_set\_data(1:2) %>%

ggplot(aes(x, id = id, split = y, value = 1)) +

geom\_parallel\_sets(aes(fill = engine))



The plot shows how a specific plane’s engine relates to each of the manufacturers. Next geom\_parallel\_sets\_axes() provides a terminal box; the axis.width argument is the only one necessary to use at this stage, and we will set it to 0.1. The labels are added by using geom\_parallel\_sets\_labels(), and they are automatically rotated.

prep\_planes %>%

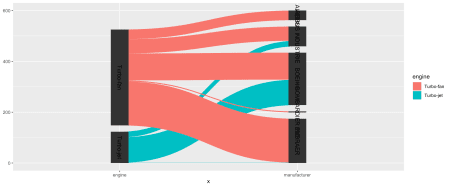
gather\_set\_data(1:2) %>%

ggplot(aes(x, id = id, split = y, value = 1)) +

geom\_parallel\_sets(aes(fill = engine)) +

geom\_parallel\_sets\_axes(axis.width = 0.1) +

geom\_parallel\_sets\_labels()



The following is done to finalize the plot:

* geom\_parallel\_sets() – Hide the legend and lower the alpha
* geom\_parallel\_sets\_axes() – Change the fill color and font color
* geom\_parallel\_sets\_labels() – Remove the rotation of the label

prep\_planes %>%

gather\_set\_data(1:2) %>%

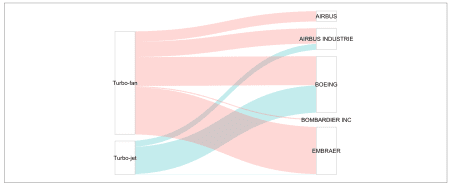
ggplot(aes(x, id = id, split = y, value = 1)) +

geom\_parallel\_sets(aes(fill = engine), show.legend = FALSE, alpha = 0.3) +

geom\_parallel\_sets\_axes(axis.width = 0.1, color = "lightgrey", fill = "white") +

geom\_parallel\_sets\_labels(angle = 0) +

theme\_no\_axes()



**Danger zone!**

When visualizing the combination of a continuous and a categorical variable, it is common practice to resort to a bar or column plot. Cases that require representing this in a single circle shape usually involve modifying a polar bar in ggplot. But, this is much easier now with ggforce. I start with the total number of planes by engine planes:

planes %>%

count(engine)

## # A tibble: 6 x 2

## engine n

##

## 1 4 Cycle 2

## 2 Reciprocating 28

## 3 Turbo-fan 2750

## 4 Turbo-jet 535

## 5 Turbo-prop 2

## 6 Turbo-shaft 5

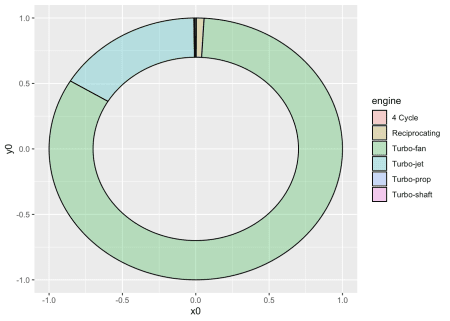
and then pipe those results into ggplot using geom\_arc\_bar() to create the circle-shaped plot. The new aesthetics employed here are: x0, y0, r0, r, amount, and explode. The x, y, and r aesthetics refer to the position and the radius of the circle. Since only one plot is needed, I fix x and y to 0. For radius, the r0 refers to the inside of the circle, and r to the outside. Setting r0 to 0.7 and r to 1 will create a sort of doughnut with a 0.3 thickness. Finally, I use “pie” as the stat.

planes %>%

count(engine) %>%

ggplot() +

geom\_arc\_bar(aes(x0 = 0, y0 = 0, r0 = 0.7, r = 1, amount = n, fill = engine), alpha = 0.3, stat = "pie")



Another cool thing this geom does is to make it east to “break-away” one or several segments of the plot. The explode aesthetic controls that. To break away the “Turbo-jet” results, I create a new column called focus, setting it to 0.2 if it is part of that engine group, and to 0 if it is not, then finish up with theme\_no\_axes().

planes %>%

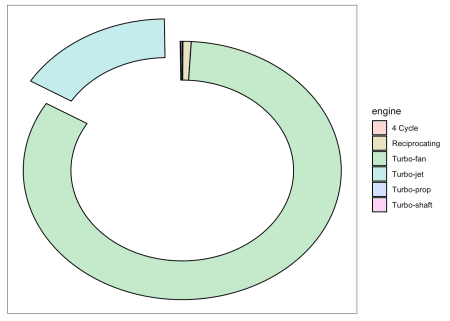
count(engine) %>%

mutate(focus = ifelse(engine == "Turbo-jet", 0.2, 0)) %>%

ggplot() +

geom\_arc\_bar(aes(x0 = 0, y0 = 0, r0 = 0.7, r = 1, amount = n, fill = engine, explode = focus), alpha = 0.3, stat = "pie") +

theme\_no\_axes()



This section is titled “Danger Zone”, because hanging the r0 in geom\_arc\_bar() may change the look of the plot to one that has fallen out of favor. That plot type happens to be the same name of the stat that we are using.

**Closing remarks**

ggforce is a great package that does a lot more than what I covered today. My hope is to have shared one or two things that will encourage you to try ggforce in your everyday work.

Special thanks to [Thomas Pedersen](https://github.com/thomasp85/), the author of the package and a co-worker of mine. His contributions to the R community also include the tidygraph and ggraph packages, which I [wrote about in this blog post](https://rviews.rstudio.com/2019/03/06/intro-to-graph-analysis/) a few months back.