Appendix to Visualizing Complex Data with Embedded Plots

Garrett Grolemund*
RStudio
and
Hadley Wickham†
Department of Statistics, Rice University

December 23, 2013

A ADVANCED IMPLEMENTATION

The layered grammar of graphics includes more than just geoms and aesthetic mappings. It also uses stats, parameters, position adjustments, and coordinate axes to describe and build graphs (Wickham, 2010). This appendix discusses how each of these can be used to create a useful interface for building embedded plots. We illustrate our ideas with code from the `ggsubplot` package.

A.1 Geoms

Software that creates embedded plots should provide a quick way to build glyph-like geoms. Glyph-like geoms can be built in current implementations of the grammar of graphics by combining traditional geoms with polar coordinates. For example, an analyst can build a star geom by combining a line geom with polar coordinates, Figure 1.a, or a frequency polygon geom with polar coordinates, Figure 1.b. He or she can build a coxcomb geom by combining a bar geom with polar coordinates, Figure 1.c.

*Garrett Grolemund is Statistician, RStudio, Boston, Massachusetts 02210 (email: grolemund@rstudio.com)
†Hadley Wickham is Adjunct Professor, Rice University, Houston, TX 77005 (email: hadley@rice.com)

```
library(ggplot2)
library(ggsubplot)
one_nasa <- nasa[nasa$id == "1-1", ]

# Figure A1.a, a star geom
ggplot(one_nasa) +
  geom_line(aes(x = date, y = fahrenheit)) +
  coord_polar()

# Figure A1.b, a star geom
ggplot(one_nasa) +
  geom_freqpoly(aes(x = ozone)) +
  coord_polar()

# Figure A1.c, a coxcomb geom
ggplot(diamonds) +
  geom_bar(aes(x = color, fill = color)) +
  coord_polar()
```

This two step process is fine for traditional plots, where glyph-like geoms are rarely used. Embedded plots, however, use glyph-like subplots frequently. These subplots will be easier to create if glyph-like objects are already available as pre-made geoms. For example, an analyst can use `ggsubplot` to draw a star, Figure 1.d; freqstar, Figure 1.e; or coxcomb geom, Figure 1.f. Each is called with a standard geom syntax like below and is analogous to the polar version.

```
# Figure A1.d, star geom
ggplot(one_nasa) +
  geom_star(aes(angle = date, r = fahrenheit,
    fill = mean(fahrenheit)), r.zero = FALSE)

# Figure A1.e, freqstar geom
ggplot(one_nasa) +
```

```
  geom_freqstar(aes(angle = ozone, fill = mean(ozone)))

# Figure A1.f, coxcomb geom
ggplot(diamonds) +
  geom_coxcomb(aes(angle = color, fill = color))
```

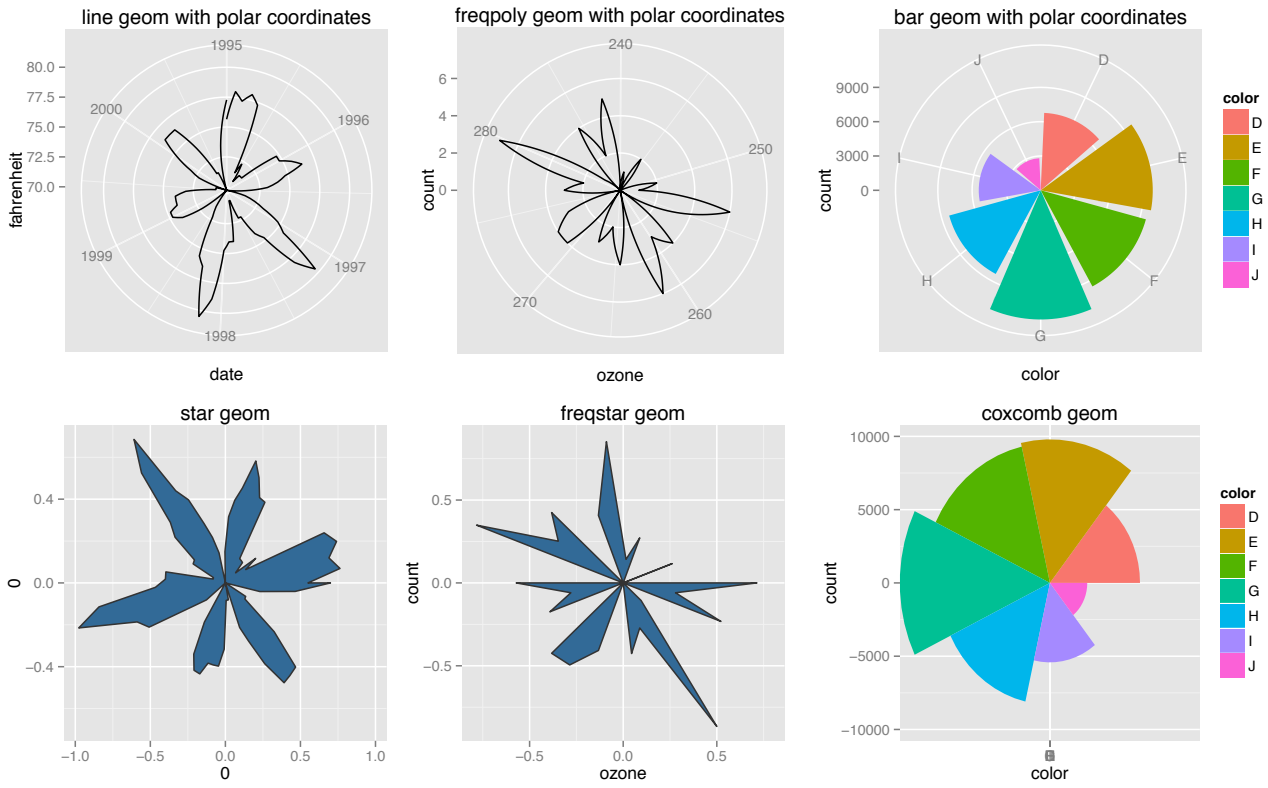

Figure 1: In the layered grammar of graphics, glyph-like objects can be built by combining common geoms with polar coordinates (**A**, **B**, **C** - *top row*). However, this arrangement can be cumbersome when building subplots. In that case, it is more convenient to create glyph-like objects with their own geoms (**D**, **E**, **F** - *bottom row*)

A.2 Mapping and Stats

Aesthetic mappings for subplots must behave differently than aesthetic mappings for simple geoms. When you place a simple geom, like a point, in the coordinate plane, you map a single data value

to a single x coordinate, and a single data value to a single y coordinate, Figure 2.a. Each subplot, however, describes a group of points. To place a subplot in the x and y coordinate field, you must map a *group* of data values to a single x value and a *group* of data values to a single y value, Figure 2.b. This can easily be done by taking the max, min, mean, median, mode, etc. of the group of values.

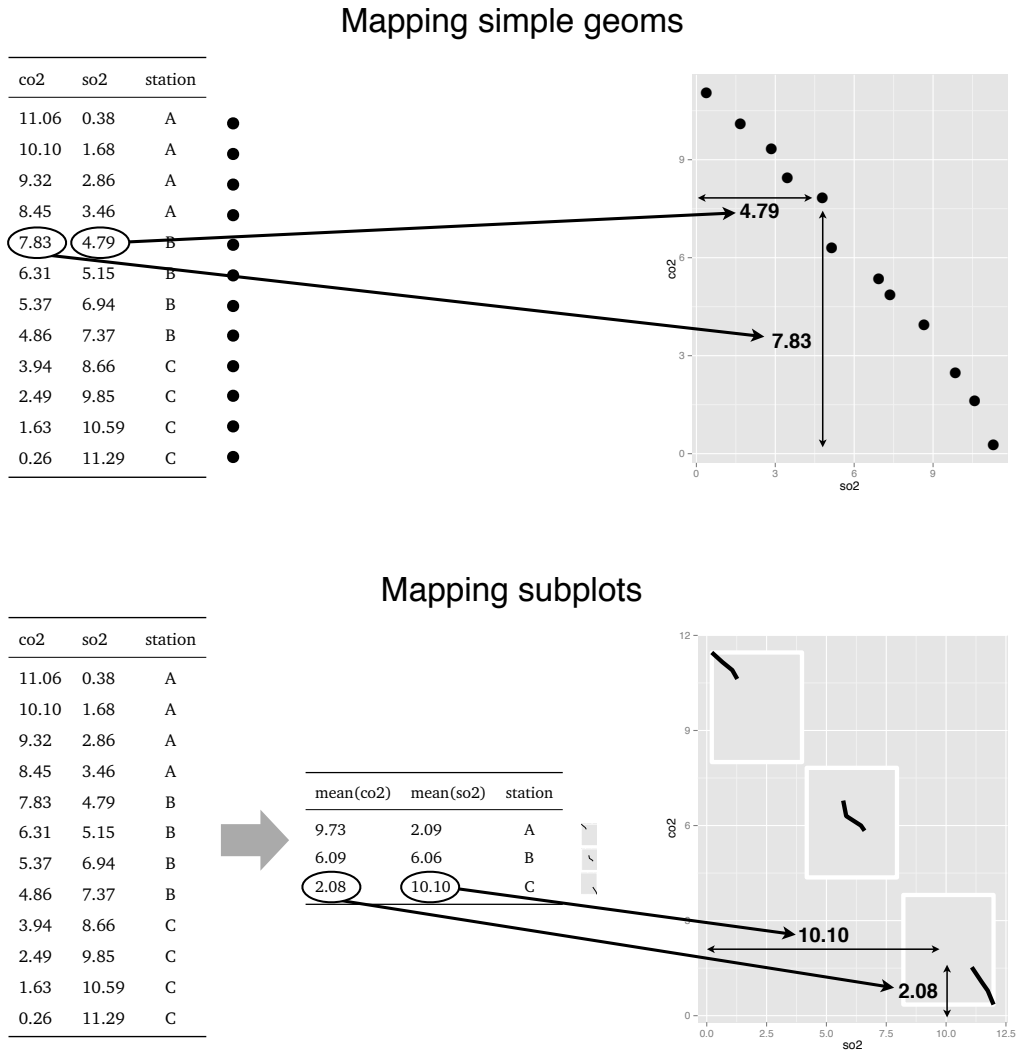

Figure 2: Simple geoms map one point to one object. A subplot maps a group of points to one subplot. To place a subplot, a user must map a group of values to a single x or y point. One way to do this intermediate transformation is with a stat.

In the layered grammar of graphics, such group-wise summaries are done with a *stat*. A stat

is a specific transformation that transforms a user's data set into a new data set that can be used to draw geoms. For example, in `ggplot2`, the boxplot geom uses a stat. Boxplots behave like subplots, they describe multiple data points with a single geom object. To place a boxplot, `ggplot2` first uses a stat to transform the group of values to a set of five numbers (a median, a 25^{th} percentile, a 75^{th} percentile, etc.). These numbers are then used to draw the boxplot. The histogram geom does something similar. It uses a stat to transform a group of values into a set of bins and counts. `ggplot2` then uses these bins and counts to draw the histogram. In `ggplot2` stats come pre-packaged with names like `stat_boxplot`, `stat_bin`, and `stat_quantile`. Every geom is associated with a default stat, so users usually do not need to worry about transforming their data with a stat to build their graph; `ggplot2` takes care of it automatically.

A programmer can add a stat that positions subplots to their software. However, a user would then be limited to positioning subplots with the transformations provided by the stat. In `ggsubplot`, we demonstrate an alternative approach. Mappings for subplots are automatically calculated on a group-wise basis. If a user passes a mapping such as `x = mean(surftemp)`, `ggsubplot` does not use the entire data set to calculate `mean(surftemp)` (which would result in a single value). `ggsubplot` first divides the data set into groups according to the subplot geom's group mapping. Then `ggsubplot` calculates `mean(surftemp)` once for each group, which results in a separate value for each subplot. This procedure is analogous to the split-apply-combine strategy described by Wickham (2011).

This arrangement offers two advantages over supplying the user with a stat. First, the user can use any function they wish to position subplots; the user is not limited to transformations that exist as a stat. The only constraint is that the user must choose a function that takes a group of values and returns a single value. Second, the method can be abstracted to use with non-embedded plots. For example `ggsubplot` provides the `ply_aes` function, which takes a `ggplot2` layer object and modifies it so the layer's mappings are calculated by group according to the layer's group aesthetic. `ply_aes` enforces summarization by subsetting the output of each mapping to just its first value. A warning message is given if the mapping would have otherwise returned multiple values. `geom_subplot` automatically uses `ply_aes`. Figure 3 shows how this technique can remarkably reduce overplotting to reveal structure.

```
# Figure A3.a overplotted ozone vs. temperature
```

```
ggplot(nasa) +
  geom_point(aes(x = atmos.fahrenheit, y = ozone, color = lat))

# Figure A3.b. Combine like points, plot their means
ggplot(nasa) +
  ply_aes(geom_point(aes(x = mean(atmos.fahrenheit), y = mean(ozone),
    color = lat[1], group = interaction(long, lat))))
```

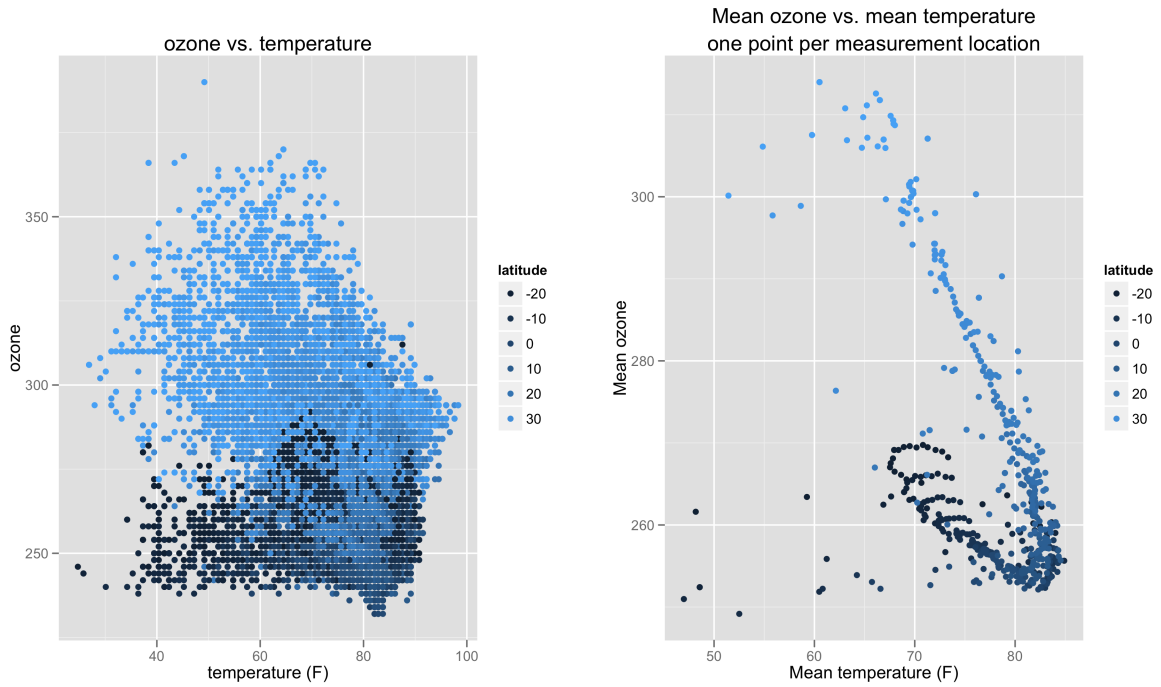

Figure 3: `ply_aes` offers a new strategy for overplotted graphs, like the one on the left. Groups of geoms are combined into single geoms that display summary information. This approach reveals that $mean(ozone)$ has a different linear relationship with temperature in the southern hemisphere than it does in the northern hemisphere (*right*). Each collection of points that share the same latitude and longitude on the left is represented by a single point on the right.

A.3 Parameters

The layered grammar of graphics allows users to control the visual appearance of geoms with parameters. Both parameters and aesthetic mappings can influence visual aspects of a geom, such

as size, shape, and color. However, parameters and mappings influence the geom in different ways. An aesthetic mapping will use values in the data set to choose the specific size, shape, and color of a specific geom. As a result, variations in the size, shape, and color will mirror variations in the values of the data set. In contrast, parameters set the size, shape, and colors of *every* geom in a layer to the same user-supplied value. In other words, parameters let the user manually tweak the appearance of geoms in a plot.

The two most useful parts of a subplot to tweak are the width and height of the subplot. `ggsubplot` allows users to manually adjust these dimensions with `width` and `height` parameters, Figure 4. Each is measured in the units of the respective major x (or y) axis.

```
# Default width and height
ggplot(nasa) +
  geom_subplot(aes(long, lat, group = id,
    subplot = geom_star(aes(r = fahrenheit, angle = date,
      fill = mean(fahrenheit)), r.zero = FALSE)))

# Decreased width and height
ggplot(nasa) +
  geom_subplot(aes(long, lat, group = id,
    subplot = geom_star(aes(r = fahrenheit, angle = date,
      fill = mean(fahrenheit)), r.zero = FALSE)),
    width = 1, height = 1)
```

`ggsubplot` also provides a relational system that can simplify setting width and height. `ggsubplot` identifies the smallest distance between any two subplots on the x axis and uses this as the default width. `ggsubplot` identifies the smallest distance between any two subplots on the y axis and uses this as the default height. A user can set the height or width to a proportion of these defaults with the `rel` function, Figure 4.

```
# Relative widths and heights
ggplot(nasa) +
  geom_subplot(aes(long, lat, group = id,
```

```
subplot = geom_star(aes(r = fahrenheit, angle = date,
  fill = mean(fahrenheit)), r.zero = FALSE)),
width = rel(1.5), height = rel(1.5))
```

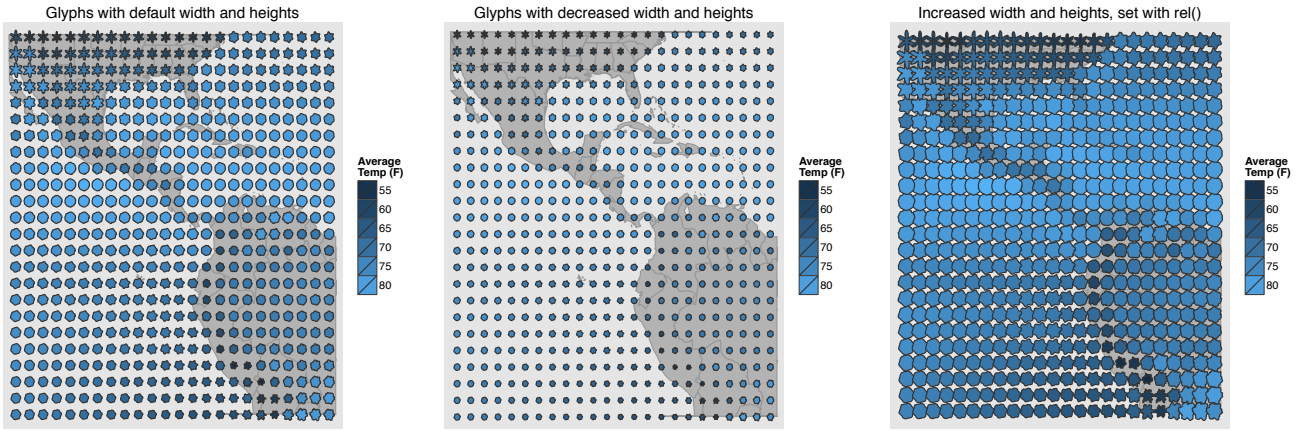

Figure 4: A user can control the width and height of subplots with the width and height parameters. Width is measured in the units of the major x axis, height in the units of the major y axis. Alternatively, a user can use the `rel` function to set width and height to proportions of the default width and height. The plot on the right uses a width and height equal to 1.5 times the default width and height. This is accomplished by setting `width = rel(1.5), height = rel(1,5)`.

A.4 Position adjustments

The geoms in a plot will often overlap, or "collide", with each other. In this case, a user may wish to alter the positions of the geoms so that they do not interfere with each other. In the layered grammar of graphics, each layer of a graphic contains a position adjustment that determines how to plot graphical elements that interfere with one another. Wilkinson and Wills (2005) refer to this concept as a collision modifier. The `ggplot2` implementation of the grammar of graphics contains four possible position adjustments. Overlapping elements can be adjusted to appear above each other (stacking), next to each other (dodging), in random nearby locations (jittering), or left as they are to overlap (identity).

A user may feel that these arrangements do not work well for embedded plots, because subplots often use position in a special way. The location of a subplot signals which data points have been

included in the subplot. Adjusting the location of the subplot would disrupt this relationship.

A programmer can attempt to handle overplotting without moving subplots by merging overlapping subplots into a single subplot. However, this tactic presents difficult programming challenges: software must identify which subplots overlap, decide how many subplots should be used to display a cluster of overlapping subplots, and determine the size and the location of merged subplots. `ggsubplot` provides the merge position adjustment, which implements this approach with limited success, Figure 5.b.

```
ggplot(seasons) +
  geom_subplot(aes(lon, lat, group = stn,
    subplot = geom_line(aes(x = time, y = pred))),
    height = 1, width = 2, position = "merge")
```

The same problem can be solved in a far simpler manner, by first binning the data into a 2D grid, and then drawing a separate subplot for each bin in the grid, Figure 5.c. `ggsubpot` provides a specific geom, `geom_subplot2d`, to do just this.

```
ggplot(seasons) +
  geom_subplot2d(aes(lon, lat,
    subplot = geom_line(aes(x = time, y = pred))),
    binwidth = c(2, 1))
```

A.5 Reference objects

Graphs usually come with a set of coordinate axes that act as a reference object for the plot. Users can judge values by scanning the axes, and users can use the axes to make comparisons across multiple plots. However, axes are difficult to read at the small scales used in subplots. Boxes and lines can also allow comparison and scale better to the smaller sizes of subplots. `ggsubplot` creates these objects with a reference parameter in the subplot layer, see Figure 6.

These reference objects allow viewers to judge the position of geoms inside the subplot and to make comparisons against the position of geoms in other subplots. The dimensions of reference objects do not vary across subplots, which facilitates accurate comparisons. However, other features

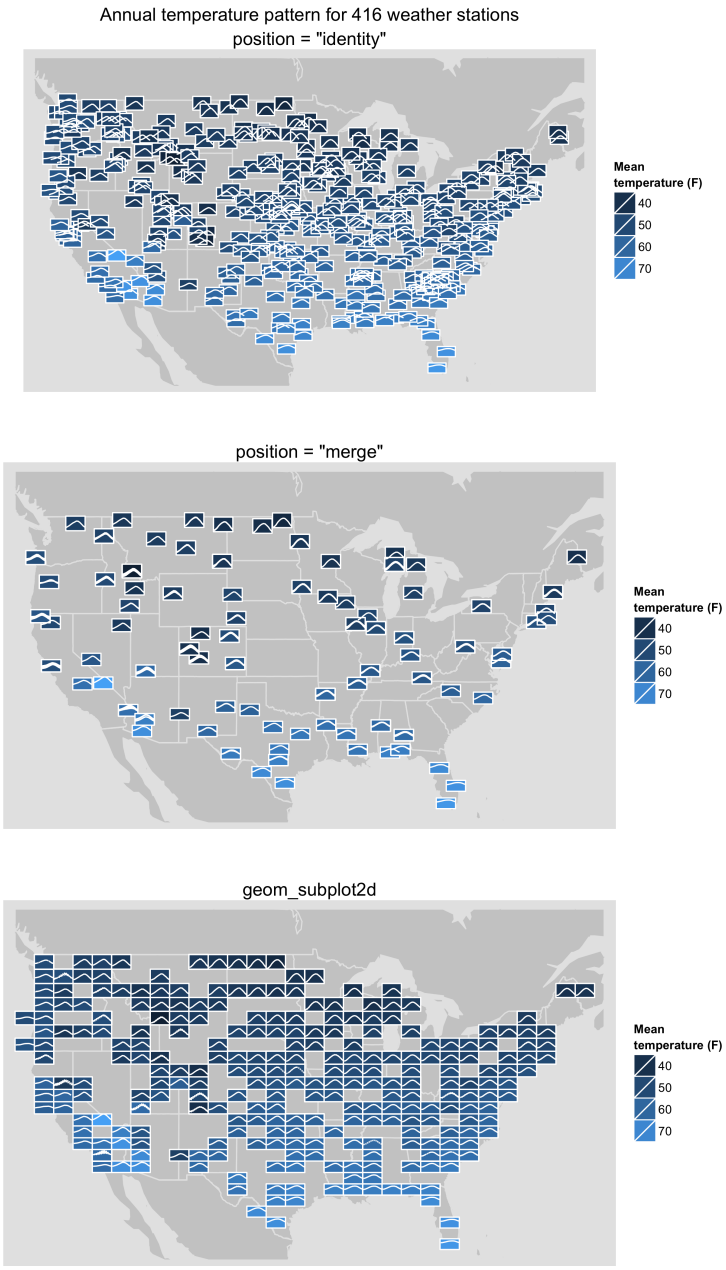

Figure 5: Temperature changes from 2000 to 2001 for multiple locations. The position of a subplot is often related to which points the subplot shows. Position = merge and `geom_subplot2d` provide two ways to avoid overlapping subplots without disrupting this relationship.

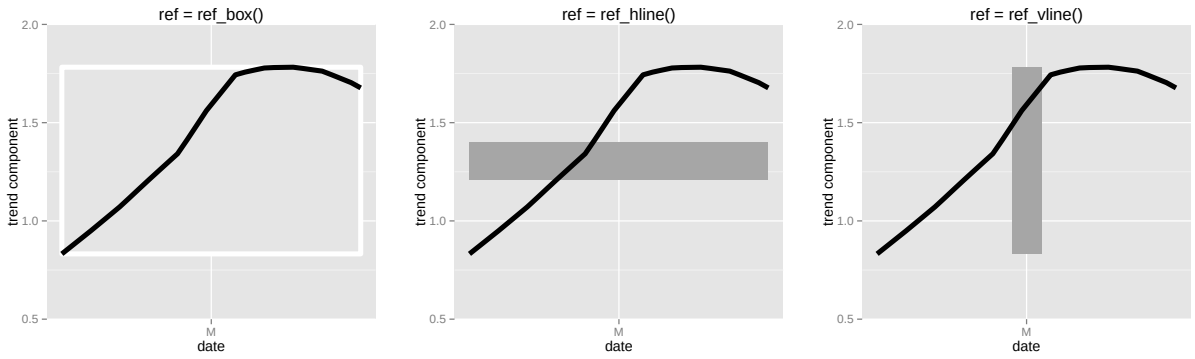

Figure 6: Reference objects allow comparison across subplots and can be more easily read at small scales than coordinate axes. In `ggsubplot`, users can add one of three types of reference objects to subplots by adding `ref = ref_box()`, `ref = ref_hline()`, or `ref = ref_vline()` to `geom_subplot` and `geom_subplot2d` calls.

of the reference object can vary to provide additional information about a subplot. For example, the fill, color, and transparency of a reference object can display group level information about the data in a subplot. The `ggsubplot` reference parameter allows users to set these aesthetics with the functions `ref_box`, `ref_vline` and `ref_hline`, see Figure 6. By default, `ref_box` displays with a grey background and white border. This matches the color scheme of `ggplot2`'s default background, while still delineating the dimensions of the subplot. Reference objects provide a quick way to compare across subplots. However, if users require a precise judgement they should still plot the subplot in its own graph with a pair of axes.

References

Wickham, H. (2010), "A layered grammar of graphics," *Journal of Computational and Graphical Statistics*, 19, 3–28.

— (2011), "The split-apply-combine strategy for data analysis," *Journal of Statistical Software*, 40, 1–29.

Wilkinson, L. and Wills, G. (2005), *The Grammar of Graphics*, Springer Verlag.