Telling Stories With Data: Comparing Program Outcomes with ggplot2

Andy Grogan-Kaylor

2019-11-27

Contents

1 Background

`ggplot2` is a powerful graphing library that can make beautiful graphs.
`ggplot2` can also help us to understand ideas of an underlying *"grammar of graphics"*.

However, ggplot can be difficult to learn. I am thinking that one way to better understand `ggplot2` might be to see how this graphing library could be applied to a concrete example of comparing program outcomes.

In this example, **program** is a *factor* and **outcome** is *numeric*.

2 Load the Simulated Social Service Agency Data

```
load("social_service_agency.RData") # simulated data
```

Table 1: Table continues below

ID	age	gender	program	mental_health_T1
4746	26.79	Male	Program B	97.53
3471	24.86	Male	Program B	82.72
4343	24.47	Male	Program C	101.2
3566	23.53	Female	Program C	92.74
2082	18.71	Male	Program C	87.08
3963	29.95	Other Identity	Program C	97.98

mental_health_T2	latitude	longitude
107.2	42.13	-83.67
103.9	42.05	-83.8
94.14	42.25	-83.63
103.4	42.11	-83.75
96.56	42.1	-83.62
92.21	42.34	-83.82

3 Load the Libraries

```
library(ggplot2) # beautiful graphs

library(ggthemes) # beautiful themes
```

4   First Approach (x is program; y is mental health)

There is *a lot of code* below. This is where we are setting up the *grammatical logic* of the graphing approach.

Devoting some time to setting up the initial logic of the plot will pay dividends in terms of exploring multiple geometries later on.

Note that I am adding optional `scale_...` and `theme...` arguments just to make the graphs look a little nicer, but these are not an essential part of the code.

```
myplot1 <- ggplot(clients, # the data I am using
                aes(x = program, # x is program
                    y = mental_health_T2, # y is mental health
                    color = program, # color is also program
                    fill = program)) + # fill is also program
  labs(y = "mental health at time 2") + # labels
  scale_color_viridis_d() + # beautiful colors
  scale_fill_viridis_d() + # beautiful fills
  theme_minimal() + # minimal theme
  theme(axis.text.x = element_text(size = rel(.5))) # smaller labels
```

5   Add Geometries That Show The Average

Now that we have devoted *a lot of code* to setting up the *grammar* of the graph, it is a relatively simple matter to try out different geometries. The geometries show the *average* value.

5.1   Bar Chart

```
myplot1 +
  stat_summary(fun.y = "mean", # summarize at mean
              geom = "bar") # bar geometry
```
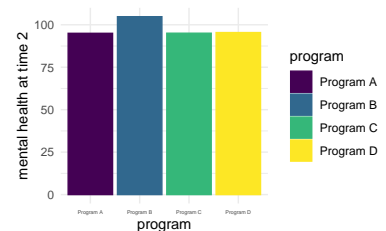

Figure 1: Bar Chart

5.2   Horizontal Bar Chart

```
myplot1 +
  stat_summary(fun.y = "mean", # summarize at mean
              geom = "bar") + # bar geometry
  coord_flip() # flip coordinates
```
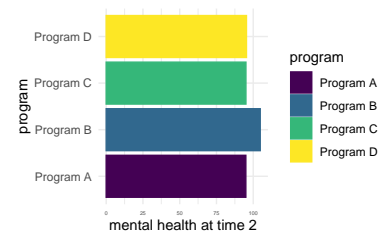

Figure 2: Horizontal Bar Chart

5.3   Point Chart

```
myplot1 +
  stat_summary(fun.y = "mean", # summarize at mean
               geom = "point", size = 5) # point geometry
```
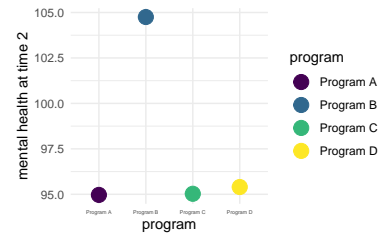
5.4   "Lollipop" Chart

The segments connecting the x axis with the points, require their own `geometry` that has its own `aesthetic`.

```
myplot1 +
  stat_summary(fun.y = "mean",
               geom = "point",
               size = 5) +
  geom_segment(aes(x = program, # x starts at
                   xend = program, # x ends at
                   y = 0, # y starts at
                   yend = mean(mental_health_T2))) # y ends at
```
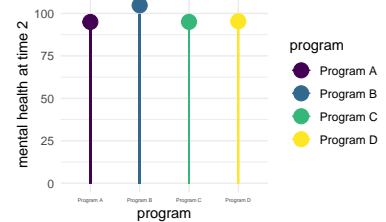
5.5   Line Chart

An extra element of the `aesthetic` is required for lines.

```
myplot1 +
  stat_summary(aes(group = 1), # line geom needs group aesthetic
               color = "black", # consistent color
               fun.y = "mean",
               geom = "line")
```
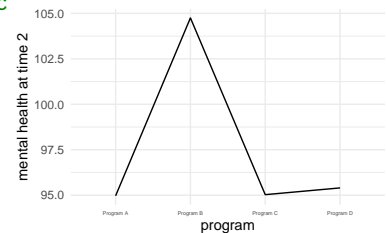
6   Add Geometries That Show the Distribution

Now that we have devoted *a lot of code* to setting up the *grammar* of the graph, it is a relatively simple matter to try out different `geometries`. The `geometries` show the *distribution* of all values.

6.1   Boxplot

```
myplot1 + geom_boxplot(fill="white") # boxplot geometry
```
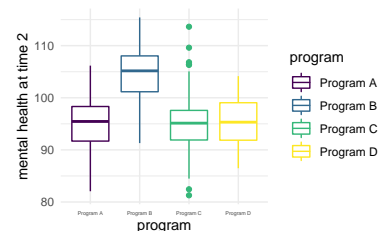
6.2   Violin Plot

```
myplot1 + geom_violin() # violinplot geometry
```


Figure 3: Point Chart

Figure 4: Lollipop Chart

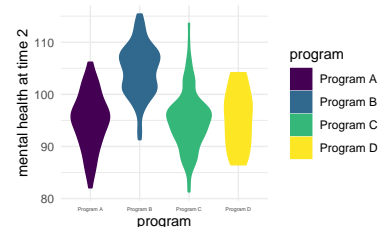

Figure 5: Line Chart

Figure 6: Boxplot

Figure 7: Violin Plot

6.3   Points

```
myplot1 + geom_point() # point geometry
```

6.4   Jittered Points

```
myplot1 + geom_jitter() # jittered point geometry
```

6.5   Beeswarm Plot

```
library(ggbeeswarm) # beeswarm geometry
```

```
myplot1 + geom_beeswarm() # beeswarm geometry
```

7   Second Approach (x is mental health; facet wrap on program)

Again, there is a *lot of code* below. This is where we are setting up the *grammatical logic* of the graphing approach.

```
myplot2 <- ggplot(clients, # the data I am using
                aes(x = mental_health_T2, # x is mental health
                    fill = program)) + # fill is program
  facet_wrap(~program) + # facet on this variable
  labs(x = "mental health at time 2") + # labels
  scale_color_viridis_d() + # beautiful colors
  scale_fill_viridis_d() + # beautiful fills
  theme_bw() # bw theme makes facets more clear
```

8   Add Geometries

However, now that we have devoted *a lot of code* to setting up the *grammar* of the graph, it is again a relatively simple matter to try out different geometries.

8.1   Histogram

```
myplot2 + geom_histogram() # histogram geometry
```

8.2   Density

```
myplot2 + geom_density() # density geometry
```
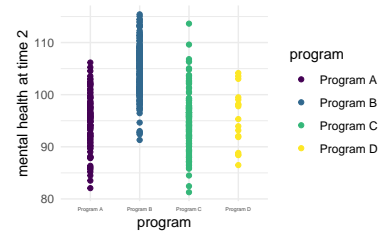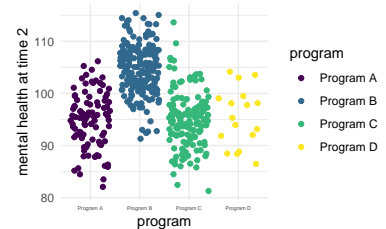

Figure 8: Points

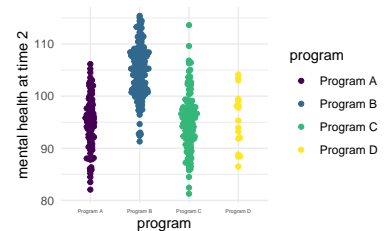Figure 9: Jittered Points
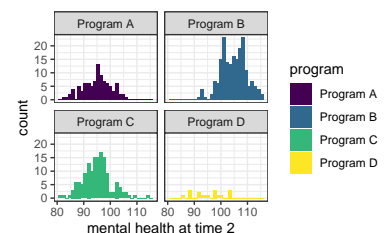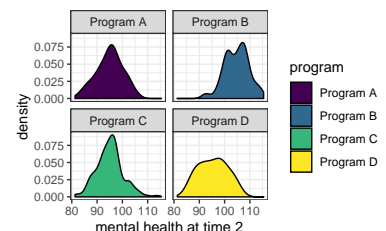

Figure 10: Beeswarm Plot

Figure 11: Histogram

Figure 12: Density

9   Third Approach (x is mental health; transparent geometries)

One last time, there is a *lot of code* below. This is where we are setting up the *grammatical logic* of the graphing approach.

```
myplot3 <- ggplot(clients, # the data I am using
                  aes(x = mental_health_T2, # x is mental health
                      fill = program)) + # fill is program
  labs(x = "mental health at time 2") + # labels
  scale_color_viridis_d() + # beautiful colors
  scale_fill_viridis_d() + # beautiful fills
  theme_minimal() # minimal theme
```

10   Add Geometries

And again, now that we have devoted a lot of code to setting up the *grammar* of the graph, it is again a relatively simple matter to try out different geometries.[1]

[1] It is important to use (alpha = ...) to create transparency with these geoms.

10.1   Histogram

```
myplot3 +
  geom_histogram(alpha = .5) # histogram geometry (transparent)
```


Figure 13: Histogram

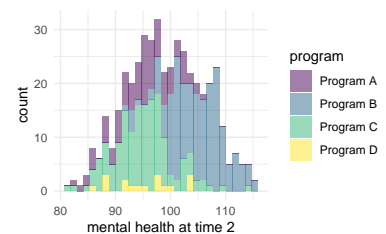### 10.2   Density

```
myplot3 +
  geom_density(alpha = .5) # density geometry (transparent)
```
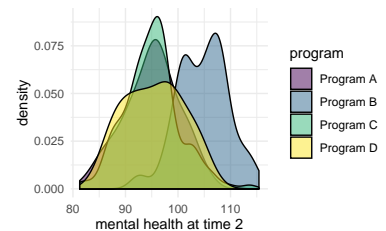

Figure 14: Density