Telling Stories With Data: Comparing Program Outcomes with ggplot2

Andy Grogan-Kaylor

2022-03-08

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 **mental health at time 2** is *numeric*.

2 Load the Simulated Social Service Agency Data

```
load("social-service-agency.RData") # simulated data
```

> The *mental health* variables are scaled to have an average of 100. Lower numbers indicate lower mental health, while higher numbers indicate higher mental health.

Table 1: Table continues below

ID	age	gender	race_ethnicity	family_income	program
2892	23	Male	African American	42359	Program B
1971	39	Female	Asian American	66500	Program C
4728	26	Female	Asian American	52726	Program C
1020	24	Male	Latinx	52911	Program D
4429	36	Female	Asian American	50287	Program C
3136	33	Male	African American	45570	Program C

mental_health_T1	mental_health_T2	latitude	longitude
95.25	106.8	42.16	-83.6
82.64	96.3	42.29	-83.88
80.49	98.72	42.14	-83.78
93.82	91.67	42.24	-83.68
83.37	99.69	42.18	-83.64
75.28	92.9	42.21	-83.7

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(.75))) # 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 `geom`etries. The `geom`etries show the *average* value.

5.1 Bar Chart

```
myplot1 +
  stat_summary(fun = "mean", # summarize at mean
               geom = "bar") + # bar geometry
  labs(title = "Bar Chart")
```

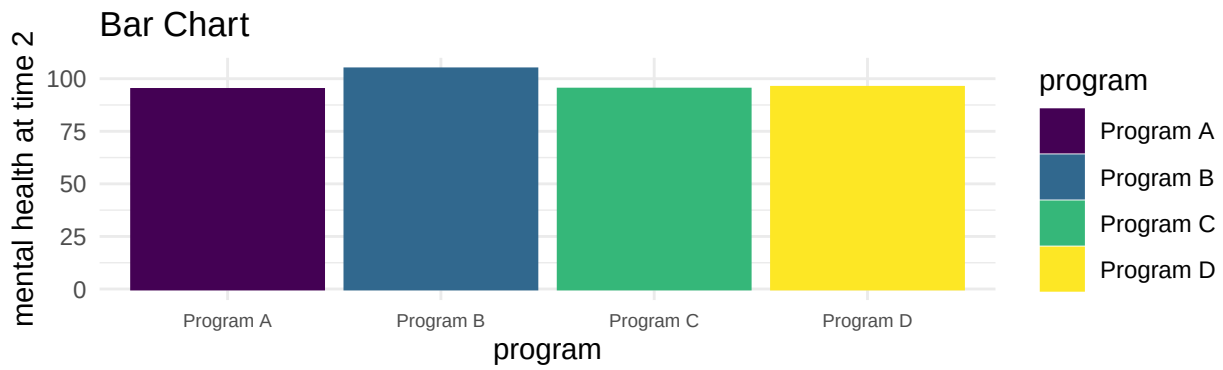

Figure 1: Bar Chart

5.2 Horizontal Bar Chart

```
myplot1 +
  stat_summary(fun = "mean", # summarize at mean
               geom = "bar") + # bar geometry
  coord_flip() + # flip coordinates
  labs(title = "Horizontal Bar Chart")
```

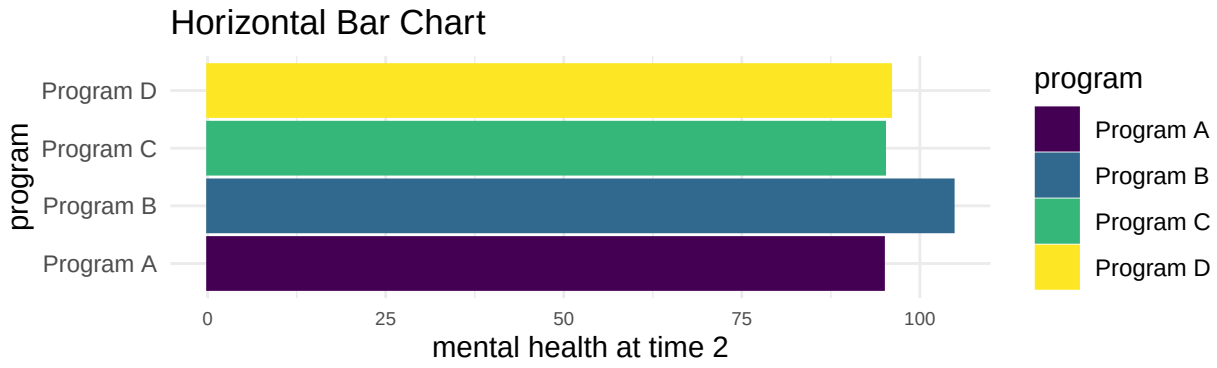

Figure 2: Horizontal Bar Chart

5.3 Point Chart

```
myplot1 +
  stat_summary(fun = "mean", # summarize at mean
               geom = "point", size = 5) + # point geometry
  labs(title = "Point Chart") +
  ylim(90, 105) # manually adjust y limits
```

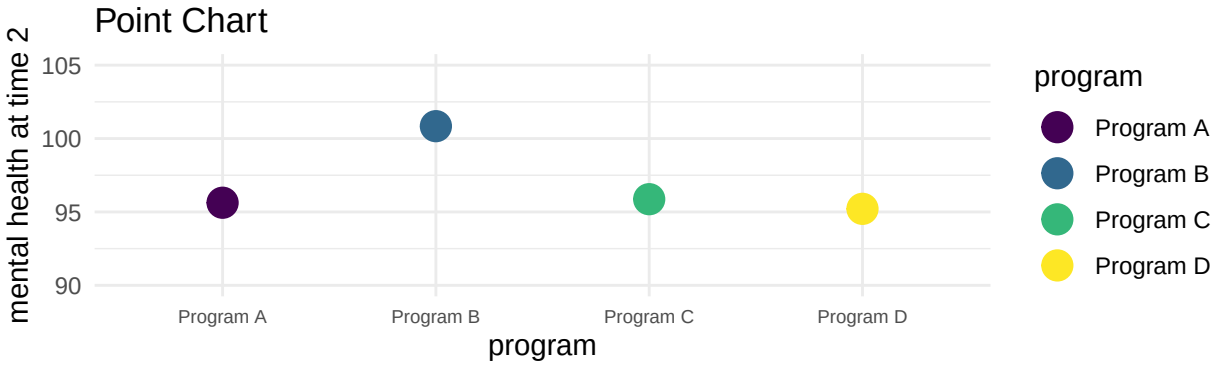

Figure 3: Point Chart

5.4 "Lollipop" Chart

The segments connecting the x axis with the points, require their own `geom`etry that has its own `aes`thetic.

```
myplot1 +
  stat_summary(fun = "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
  labs(title = "Lollipop Chart") +
  ylim(0, 105) # manually adjust y limits
```

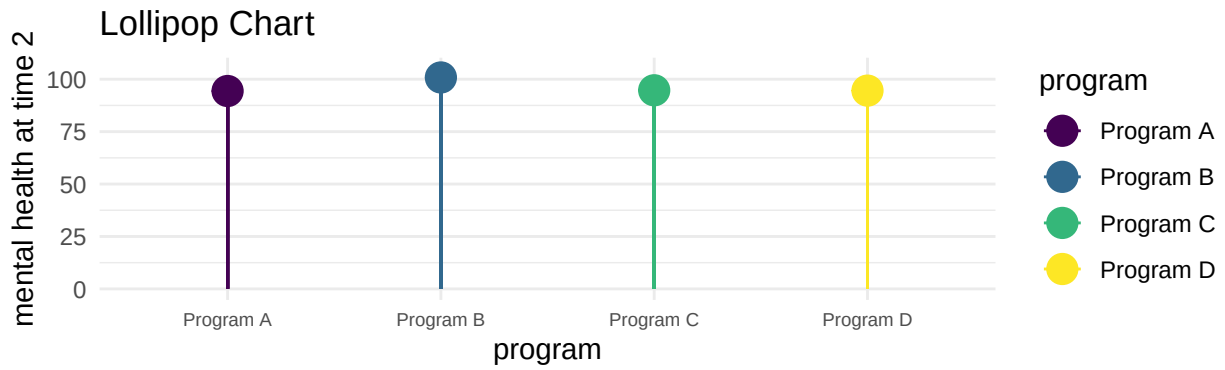

Figure 4: Lollipop Chart

5.5 Line Chart

An extra element of the `aes`thetic is required for lines.

```
myplot1 +
  stat_summary(aes(group = 1), # line geom needs group aesthetic
               color = "black", # consistent color
               fun = "mean",
               geom = "line") +
  labs(title = "Line Chart")
```

> A line chart is likely *not* an appropriate way to show these program outcomes as a line chart is more appropriate when the x axis represents some kind of *time trend*.

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 `geom`etries. The `geom`etries show the *distribution* of all values.

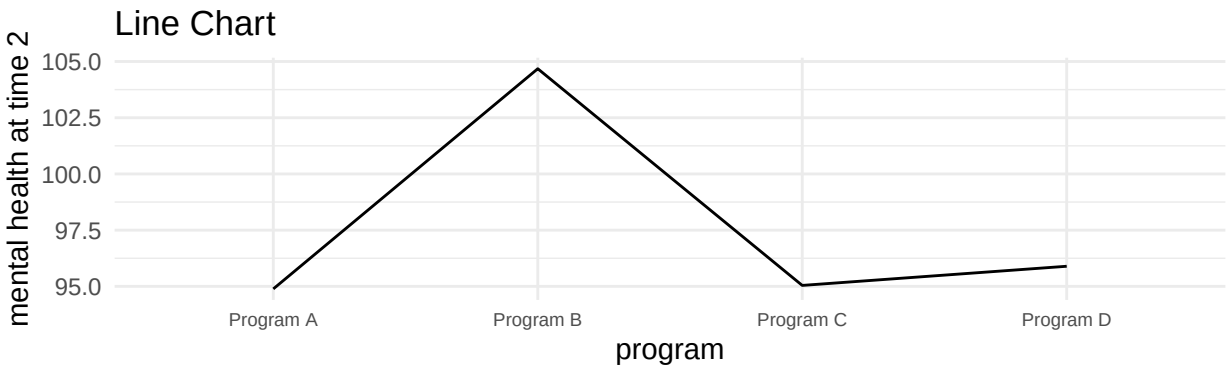

Figure 5: Line Chart

6.1 Boxplot

```
myplot1 +
  geom_boxplot(fill="white") + # boxplot geometry
  labs(title = "Boxplot")
```

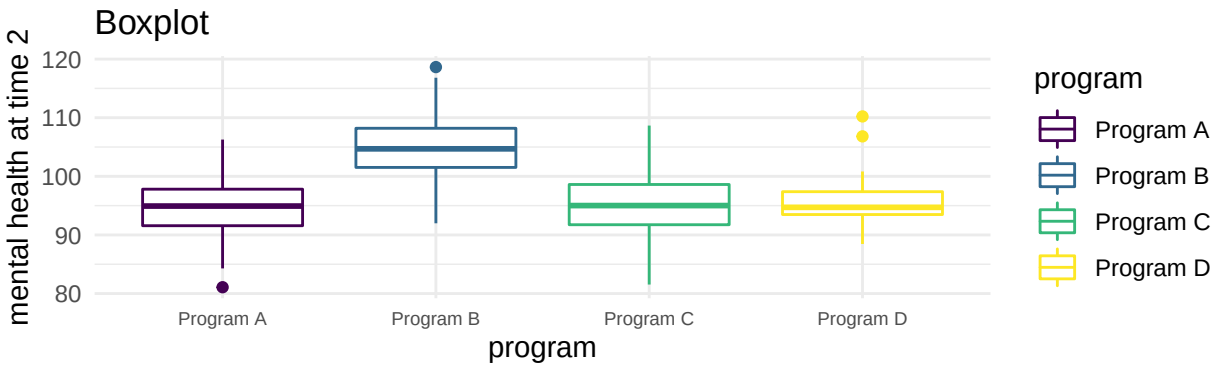

Figure 6: Boxplot

6.2 Violin Plot

```
myplot1 +
  geom_violin() + # violinplot geometry
  labs(title = "Violin Plot")
```

6.3 Points

```
myplot1 +
  geom_point() + # point geometry
  labs(title = "Points")
```

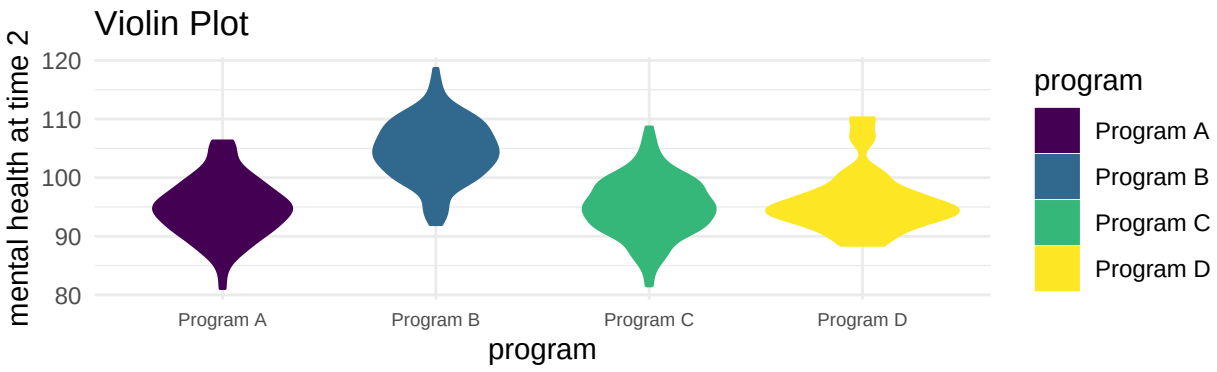

Figure 7: Violin Plot

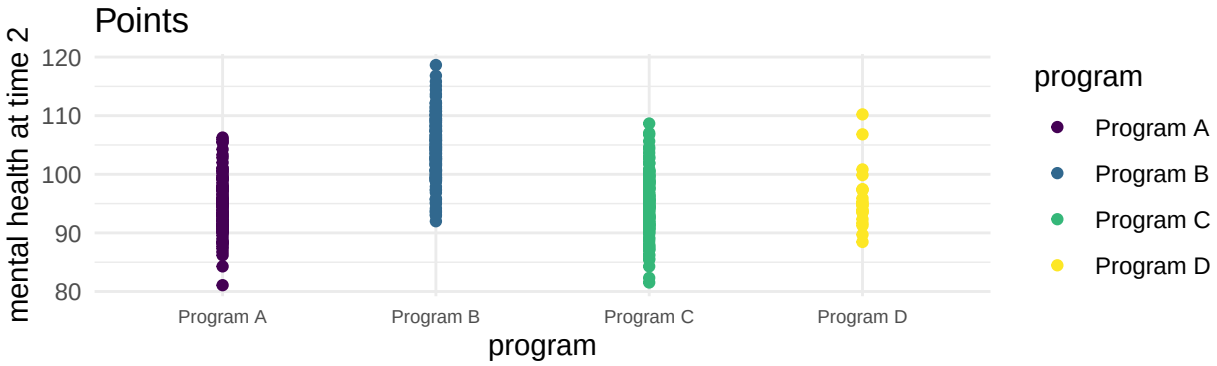

Figure 8: Points

6.4 Jittered Points

```
myplot1 +
  geom_jitter() + # jittered point geometry
  labs(title = "Jittered Points")
```

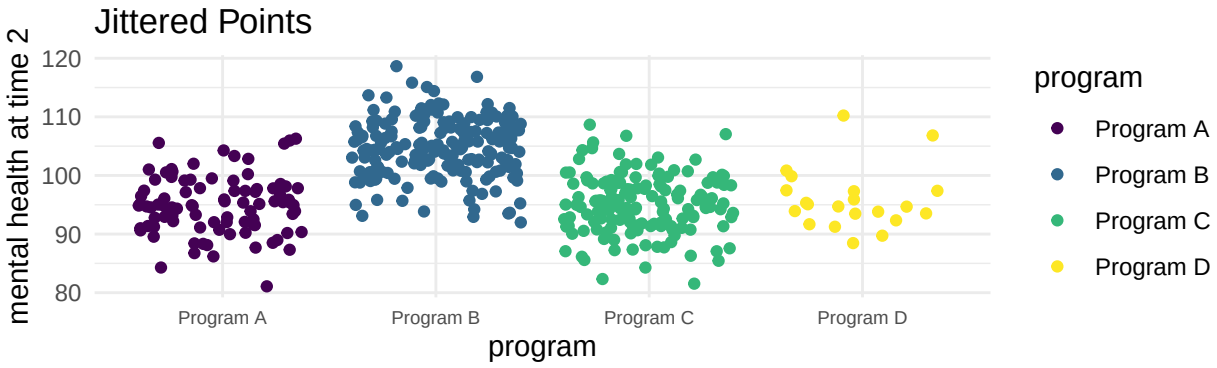

Figure 9: Jittered Points

6.5 Beeswarm Plot

```
library(ggbeeswarm) # beeswarm geometry

myplot1 +
  geom_beeswarm() + # beeswarm geometry
  labs(title = "Beeswarm Plot")
```

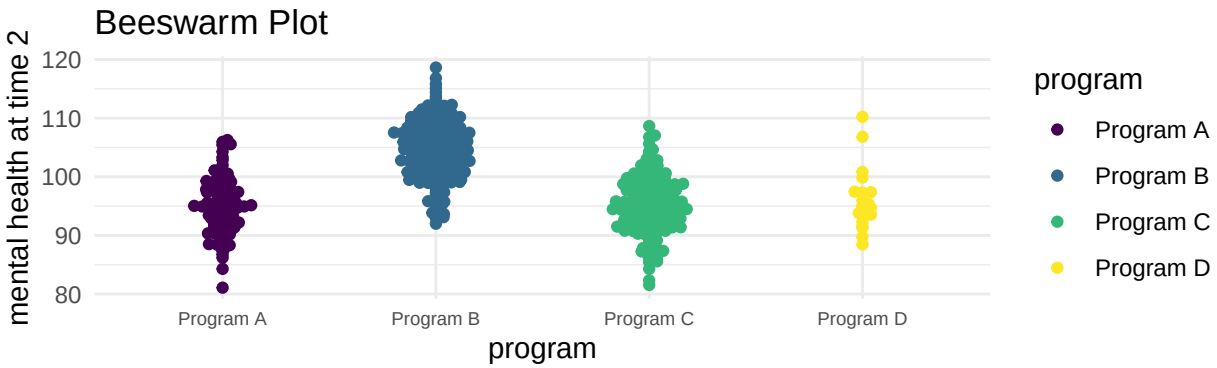

Figure 10: Beeswarm Plot

6.6 Dotplots

```
library(ggdist) # dotplot geometry

myplot1 +
```

```
stat_dots() + # dotplot geometry
labs(title = "Dotlot")
```

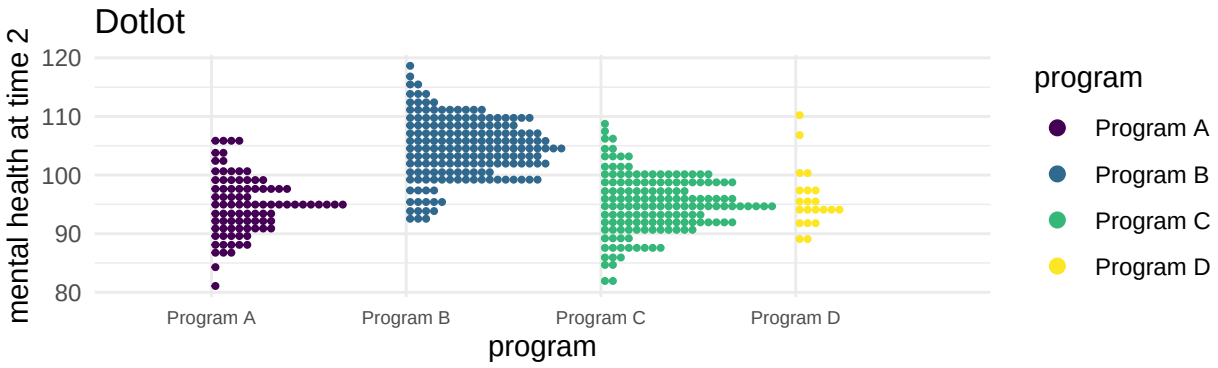

Figure 11: Dotplot

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 `geom`etries.

8.1 Histogram

```
myplot2 +
  geom_histogram() + # histogram geometry
  labs(title = "Histogram")
```

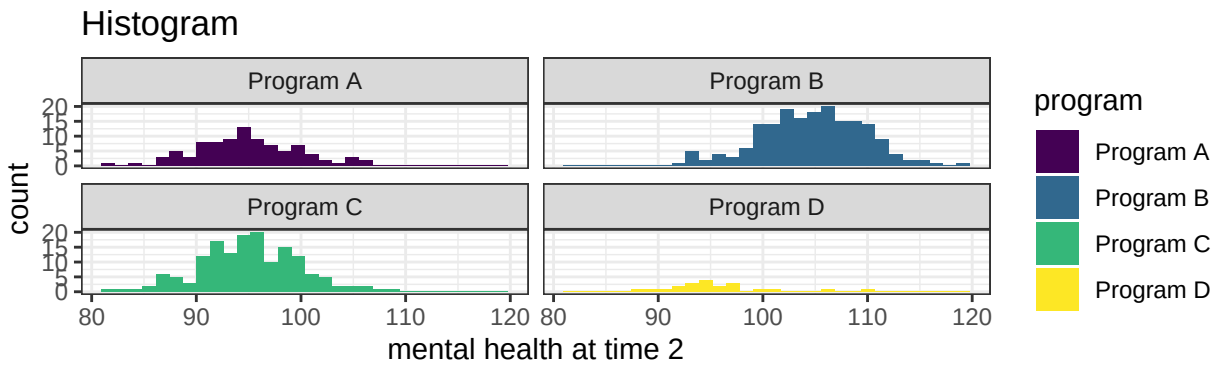

Figure 12: Histogram

8.2 Density

```
myplot2 +
  geom_density() + # density geometry
  labs(title = "Density")
```

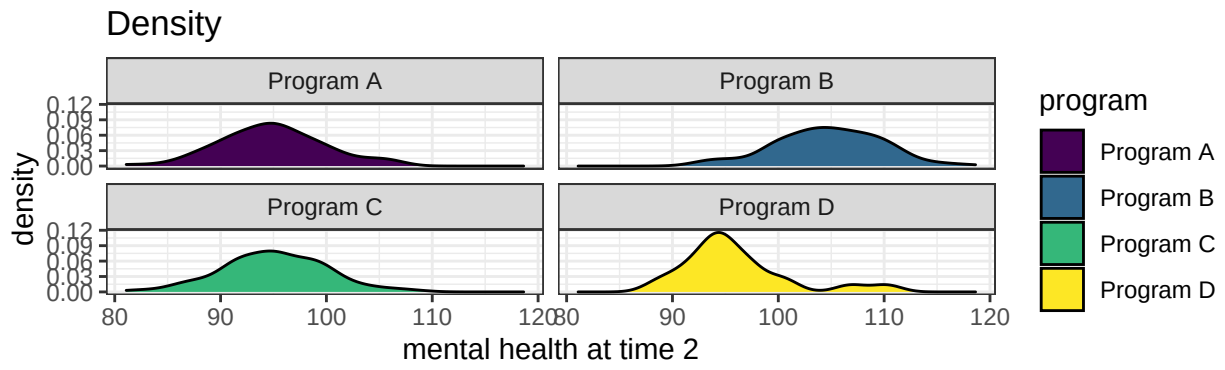

Figure 13: Density

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 `geom`etries.[1]

10.1 Histogram

```
myplot3 +
  geom_histogram(alpha = .5) + # histogram geometry (transparent)
  labs(title="Histogram")
```

10.2 Density

```
myplot3 +
  geom_density(alpha = .5) + # density geometry (transparent)
  labs(title = "Density")
```

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

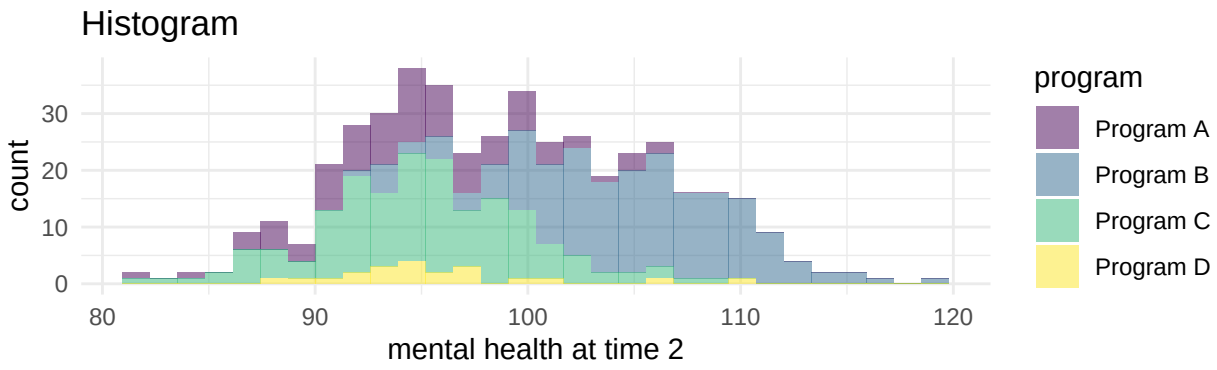

Figure 14: Histogram

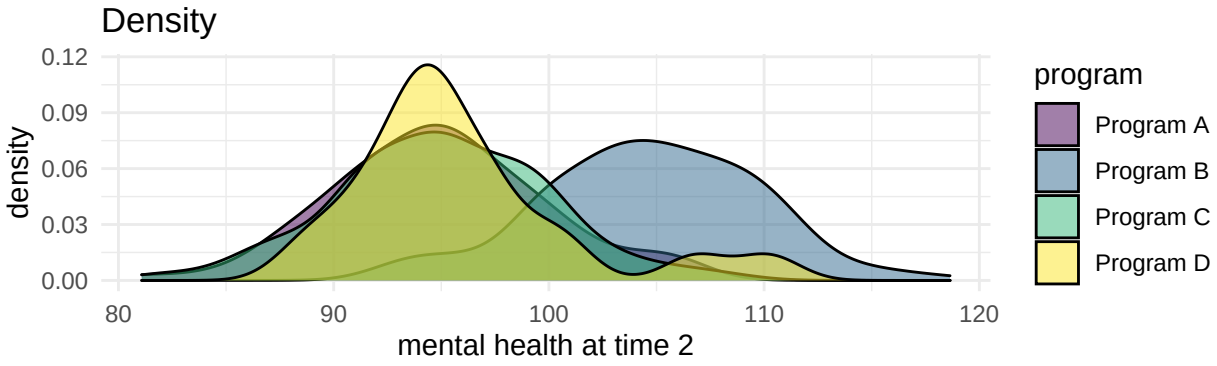

Figure 15: Density