Multipanel Figures & Adding To Your Plot

Last Time

So much customization!

- Color palettes
- Themes
- Manually changing things

This time

Multipanel Plots

- Faceting
- Combining separates into 1

Adding Stuff

- Lines
- Text

Faceting

Faceting lets you break up your plot into multiple sub-plots. There are two main types:

1. `facet_grid`
2. `facet_wrap`

For both of these, we will read the tilde (`~`) as the word "by"...

facet_grid

This is especially great when you have multiple factors to separate your graph on.

Like the name implies, `facet_grid` is going to make a grid. Just like a matrix, the left is the rows and right is the columns (i.e., 2x3 matrix = 2 rows, 3 columns). You can put your factor on either side, but that will change the layout of your grid! For example:

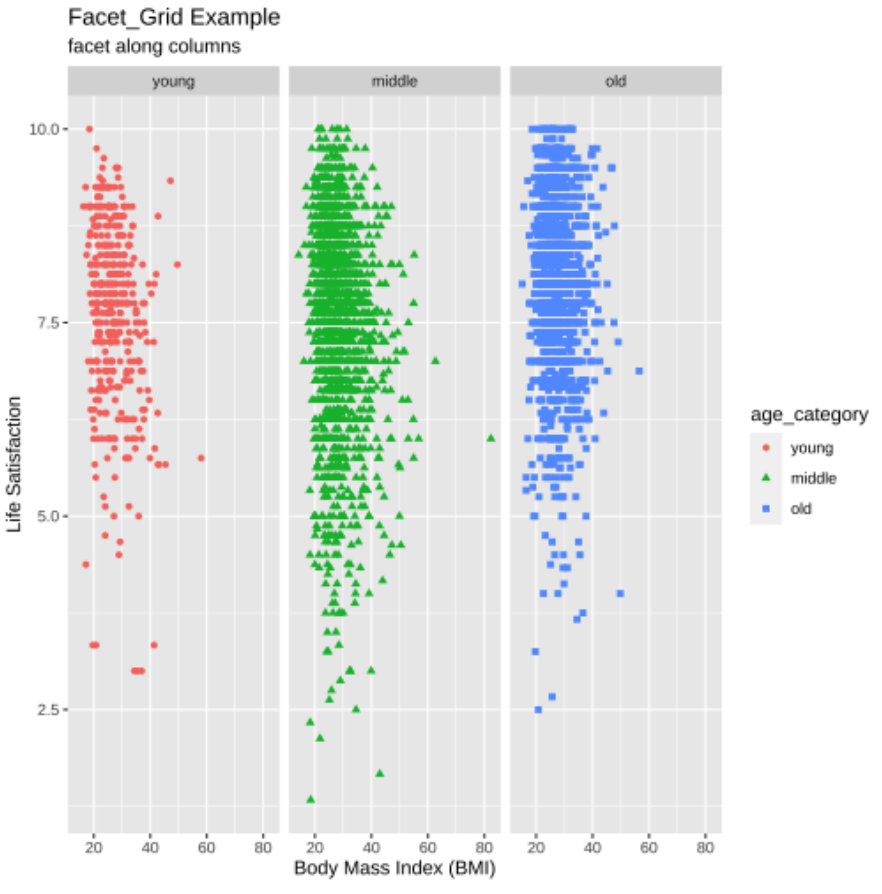
- `~ age_category` is read as *"by age category"*, and is in the column position. There are 3 levels of the `age_category` factor. Therefore, the result of `~ age_category` is a 1x3 grid.
- `age_category ~` is read as *"age category by"*. This doesn't work! You need something else to finish your "sentence". To indicate that you do not want to facet by any additional factor, use a period (`.`). So the correct syntax for faceting `age_category` in the *row* position is `age_category ~ .`. The result will be a 3x1 grid.

Let's see this in action...

facet_grid

Facet **by** age_category

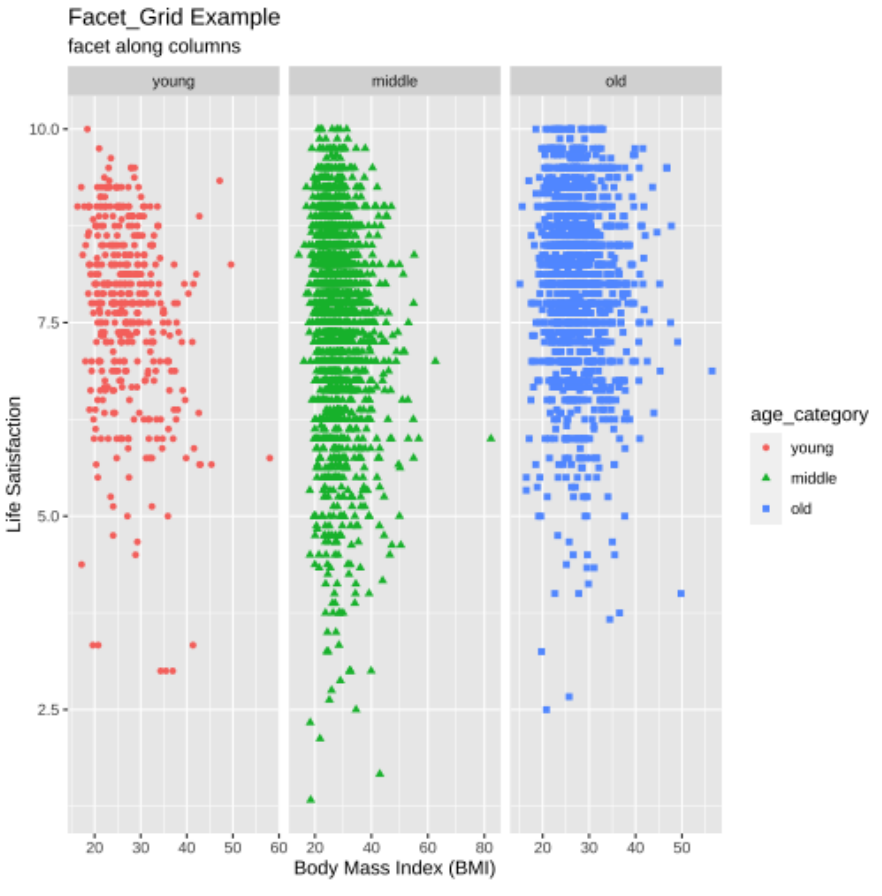
```
ggplot(data = midus,
       aes(x = BMI,
           y = life_satisfaction)) +
  geom_point(aes(color = age_category,
                 shape = age_category)) +
  labs(x = "Body Mass Index (BMI)",
       y = "Life Satisfaction",
       title = "Facet_Grid Example",
       subtitle = "facet along columns") +
  facet_grid(~ age_category)
```



facet_grid

Notice how the x-axis was the same for each of the 3 facets? What if we let them be specific to that particular facet?

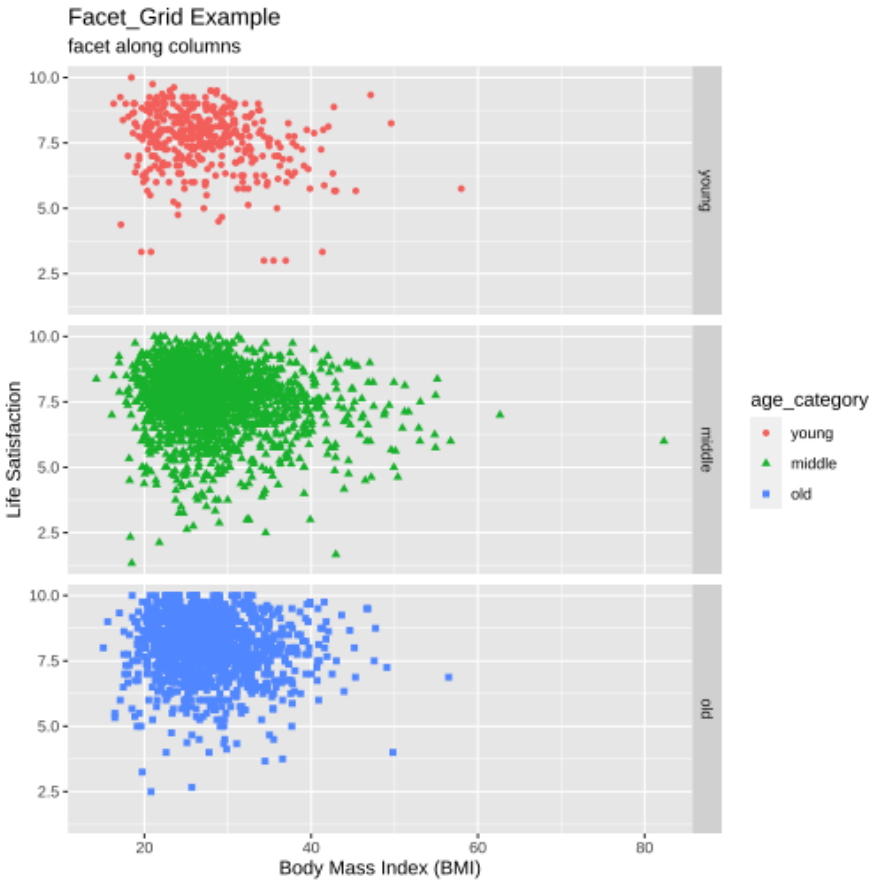
```
ggplot(data = midus,
       aes(x = BMI,
           y = life_satisfaction)) +
  geom_point(aes(color = age_category,
                 shape = age_category)) +
  labs(x = "Body Mass Index (BMI)",
       y = "Life Satisfaction",
       title = "Facet_Grid Example",
       subtitle = "facet along columns") +
  facet_grid(~ age_category,
             scales = "free_x")
```



facet_grid

Facet `age_category` **by** `.`

```
ggplot(data = midus,
       aes(x = BMI,
           y = life_satisfaction)) +
  geom_point(aes(color = age_category,
                 shape = age_category)) +
  labs(x = "Body Mass Index (BMI)",
       y = "Life Satisfaction",
       title = "Facet_Grid Example",
       subtitle = "facet along columns") +
  facet_grid(age_category ~ .)
```



Faceting based on 2 variables

But first... Side track

What happens when a categorical variable is treated continuously, and not as a factor?

The following creates a new variable called `session` and is scored as 1 and 2. We will make sure to treat this as a numeric variable

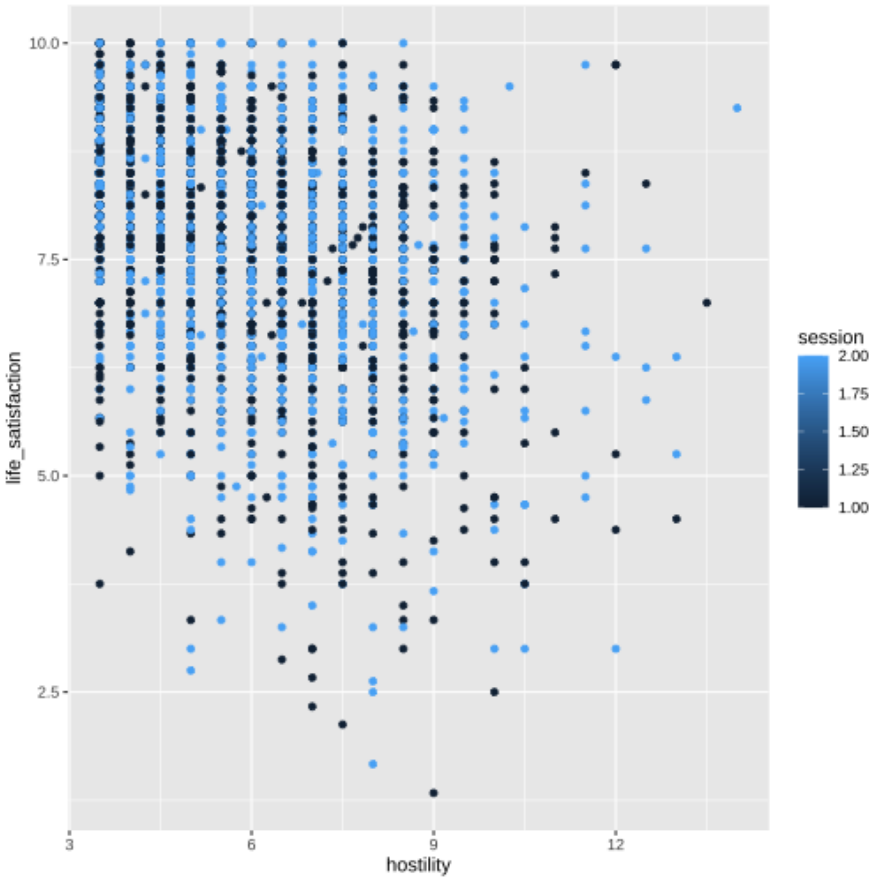
```
session <- c(rep(c(1, 2), times = 935))
midus["session"] <- rep(session, times = 2)
midus$session <- as.numeric(midus$session)

head(midus$session)
```

```
## [1] 1 2 1 2 1 2
```

But first... Side track

If we plot this, check out the legend and colorings...

```
ggplot(data = midus,
       aes(x = hostility,
           y = life_satisfaction)) +
  geom_point(aes(color = session))
```



But first... Side track

If you are working with categorical variables, it is strongly recommended that you tell R to treat it as a factor -- don't skimp on this step!

It is easier to keep categorical variables as words, rather than numeric codes. So instead of `1` and `2`, let's change the scores to be `session1` and `session2` for our new `session` variable.

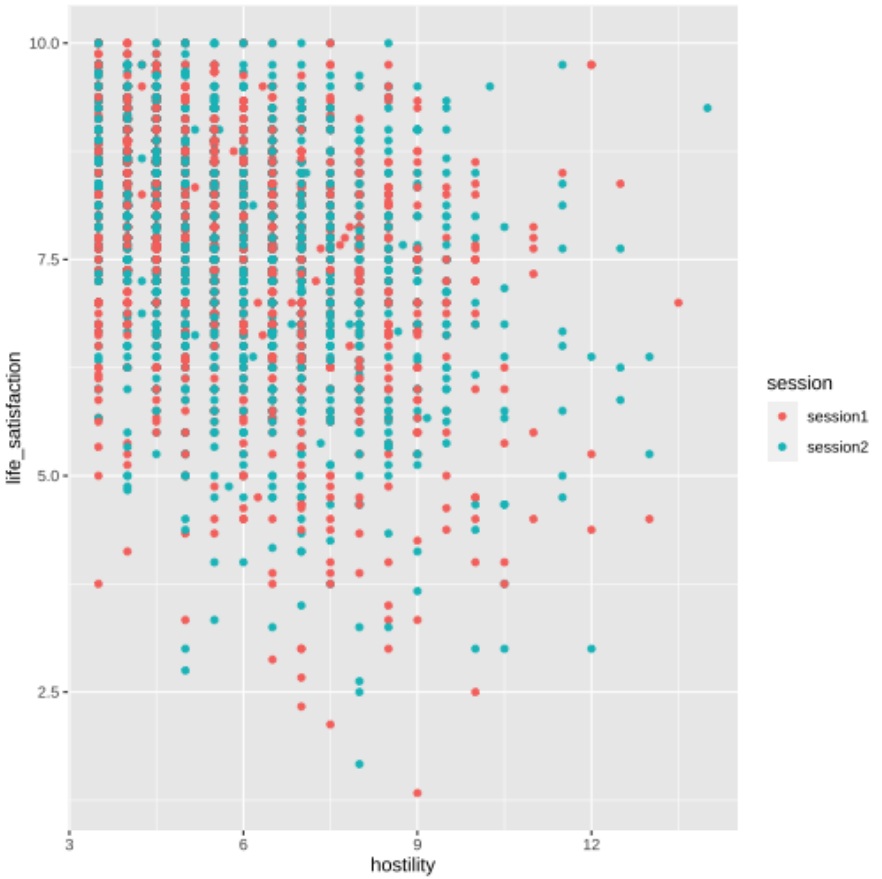
```
midus$session <- factor(midus$session,
                        labels = c("session1", "session2"))
class(midus$session)
```

```
## [1] "factor"
```

But first... Side track

Much better!

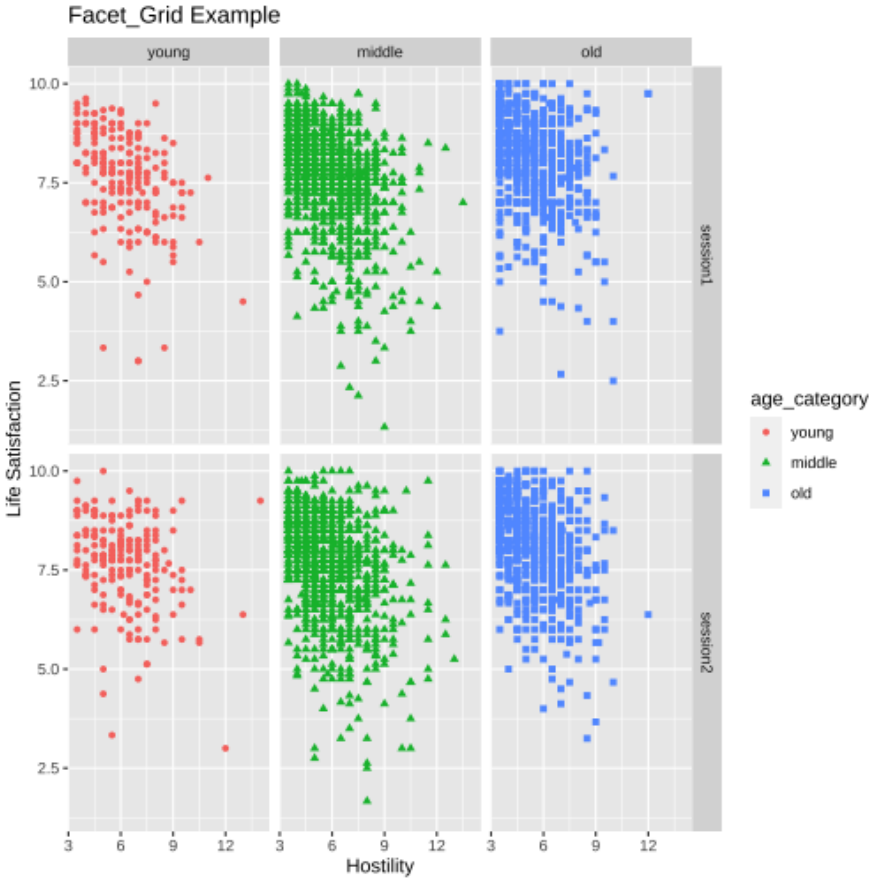
```
ggplot(data = midus,
       aes(x = hostility,
           y = life_satisfaction)) +
  geom_point(aes(color = session))
```



Faceting based on 2 variables

Session x Age Category:

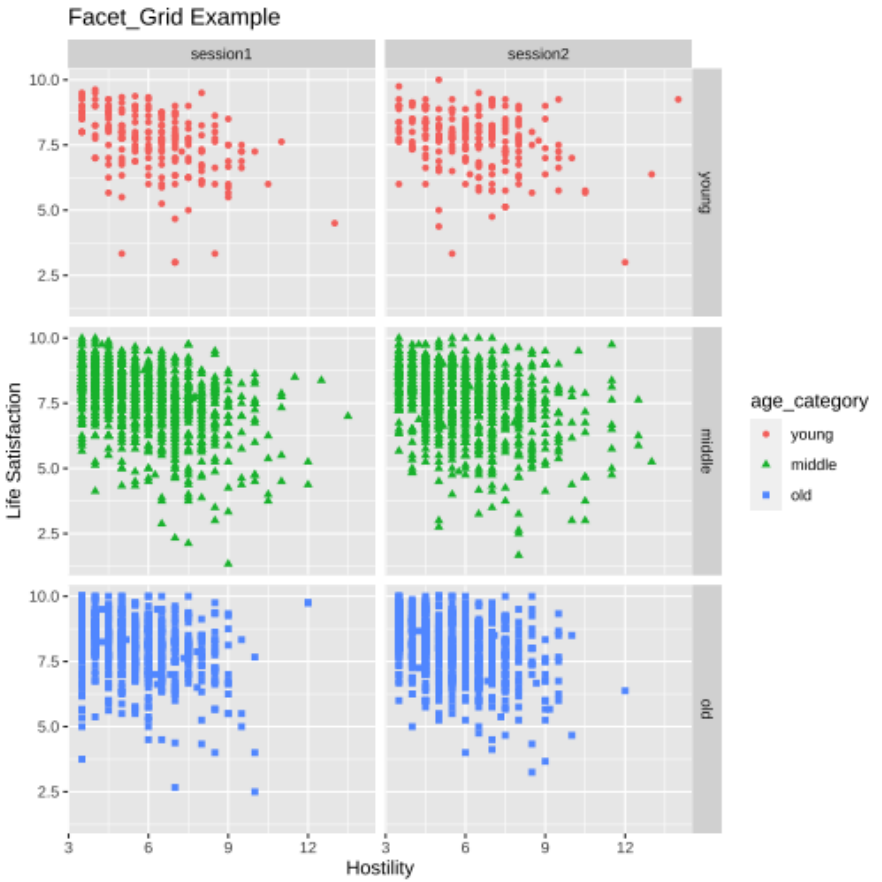
```
ggplot(data = midus,
       aes(x = hostility,
           y = life_satisfaction)) +
  geom_point(aes(color = age_category,
                 shape = age_category)) +
  labs(x = "Hostility",
       y = "Life Satisfaction",
       title = "Facet_Grid Example") +
  facet_grid(session ~ age_category)
```



Faceting based on 2 variables

Age Category x Session:

```
ggplot(data = midus,
       aes(x = hostility,
           y = life_satisfaction)) +
  geom_point(aes(color = age_category,
                 shape = age_category)) +
  labs(x = "Hostility",
       y = "Life Satisfaction",
       title = "Facet_Grid Example") +
  facet_grid(age_category ~ session)
```



facet_wrap

This basically creates a ribbon that will just continue on to the next row when ready.

This is useful for when you have categorical variables, but they don't necessarily need to be in a grid or matrix format.

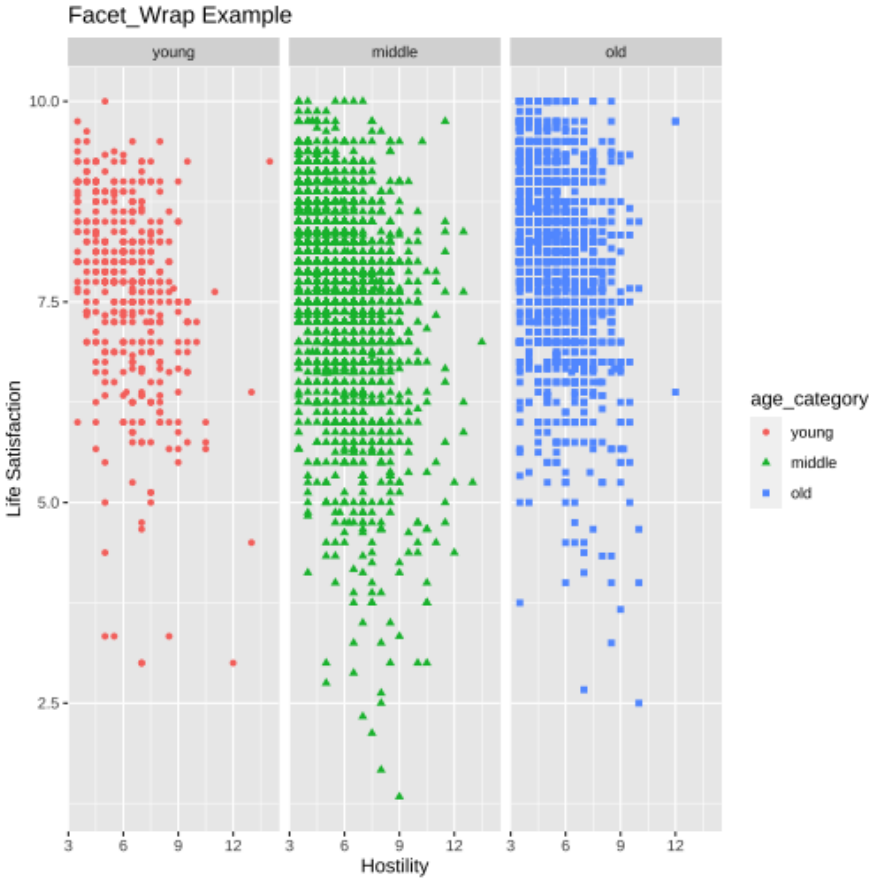
For instance, if one of the `facet_grid` cells would be empty, you probably don't want to show an empty plot (a plot with no points/shapes – you'd rather it be just blank space).

This does not follow the rows by columns syntax – it will always just go to the next row.

facet_wrap

Since there are only 3 levels, by default it will look just like `facet_grid`

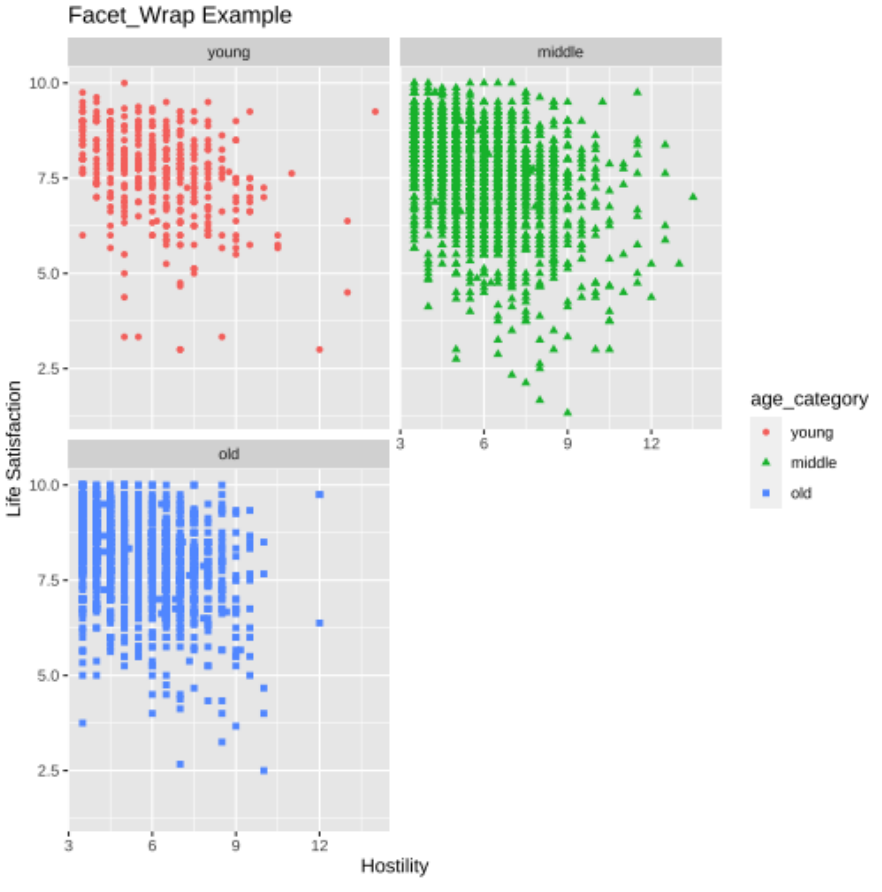
```
ggplot(data = midus,
       aes(x = hostility,
           y = life_satisfaction)) +
  geom_point(aes(color = age_category,
                 shape = age_category)) +
  labs(x = "Hostility",
       y = "Life Satisfaction",
       title = "Facet_Wrap Example") +
  facet_wrap(~ age_category)
```



facet_wrap

But now, let's say there should only be 2 columns. What happens to our third subplot?

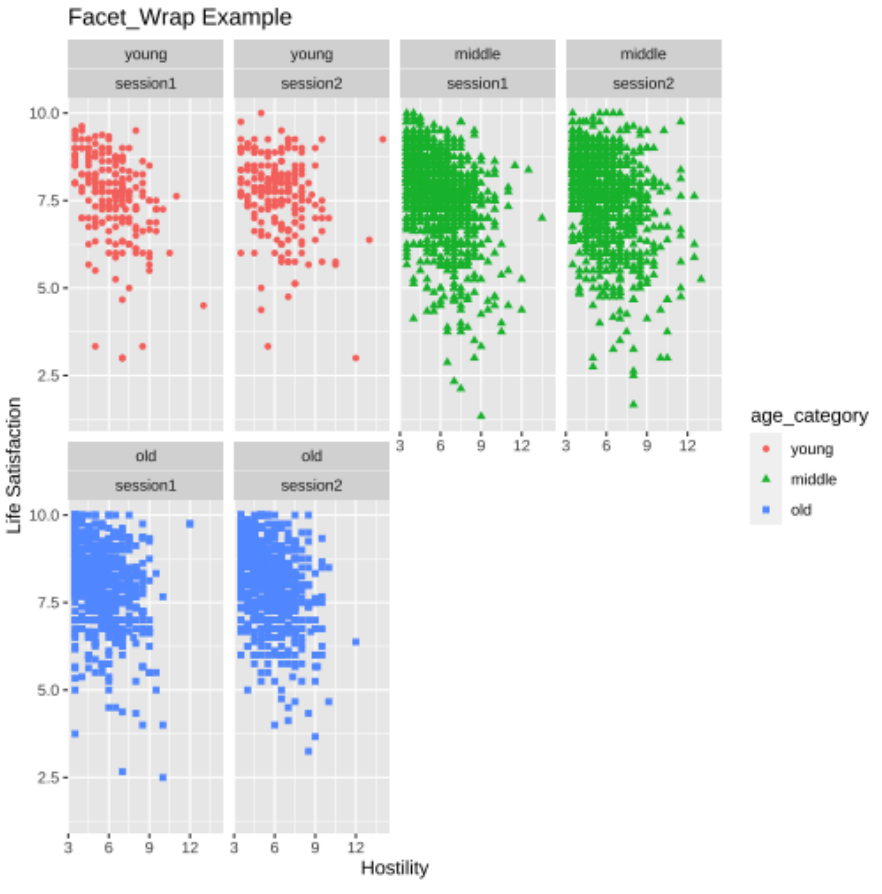
```
ggplot(data = midus,
       aes(x = hostility,
           y = life_satisfaction)) +
  geom_point(aes(color = age_category,
                 shape = age_category)) +
  labs(x = "Hostility",
       y = "Life Satisfaction",
       title = "Facet_Wrap Example") +
  facet_wrap(~ age_category, ncol = 2)
```



facet_wrap

We can facet on 2 variables with `facet_wrap`, but now we're adding rather than making a true grid.

```
ggplot(data = midus,
       aes(x = hostility,
           y = life_satisfaction)) +
  geom_point(aes(color = age_category,
                 shape = age_category)) +
  labs(x = "Hostility",
       y = "Life Satisfaction",
       title = "Facet_Wrap Example") +
  facet_wrap(~ age_category + session, ncol
```



Arranging Multiple Plots

Faceting is great because you're making the same plot but carved up based on some criteria.

Now let's say you have 3 completely independent plots (not subplots!) that you want to arrange into a cohesive figure. You're arranging a **grid of plots**. To do this, we will use the `ggarrange()` function from the `ggpubr` package. If you do not already have this package installed, please do so now.

Importantly, you need to STORE these plots into your environment first (which means the plot won't immediately appear when you run the code). Then you can arrange the plots based on the names you assigned them.

Arranging Multiple Plots

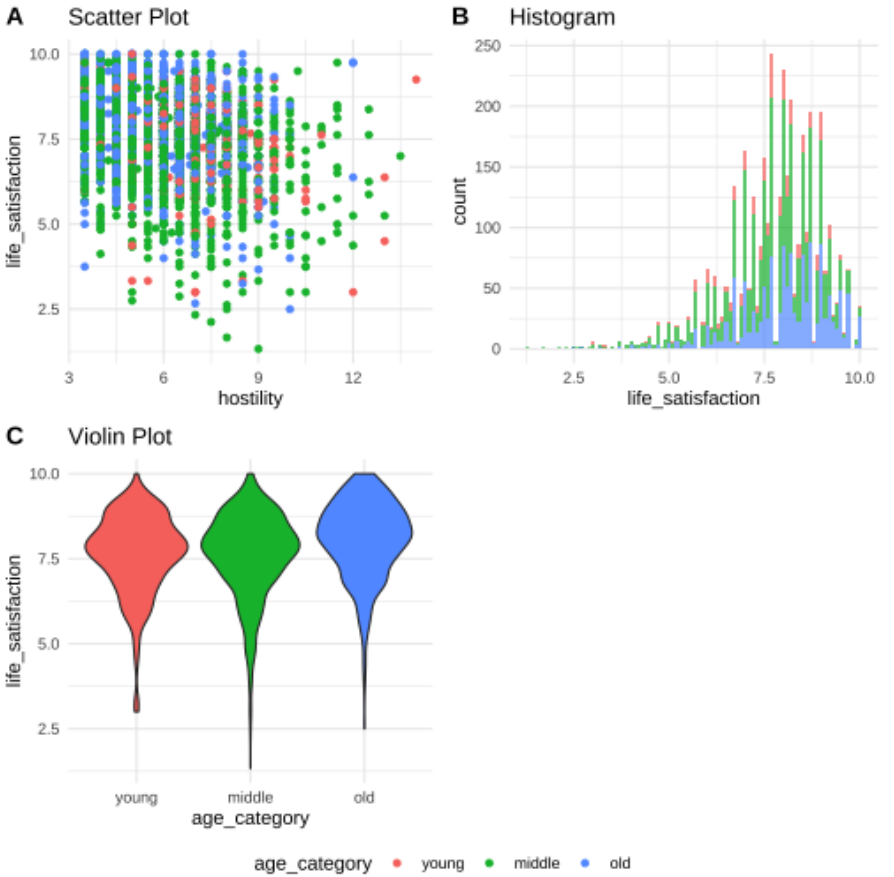
Let's first create our plots. If we want a single legend for every plot (e.g., the colors are the same for all plots), then make sure you have coded this accordingly.

```
library(ggpubr)

# plot A - a scatter plot
plotA <- ggplot(data = midus, aes(x = hostility, y = life_satisfaction)
  geom_point(aes(color = age_category)) +
  theme_minimal() +
  labs(title = "Scatter Plot")

# plot B - histogram
plotB <- ggplot(data = midus, aes(x = life_satisfaction)) +
  geom_histogram(binwidth = .1, aes(fill = age_category), alpha = .7) +
  theme_minimal() +
  labs(title = "Histogram")

# plot C - violin plot
plotC <- ggplot(data = midus, aes(x = age_category, y = life_satisfactio
  geom_violin(aes(fill = age_category)) +
  theme_minimal() +
  labs(title = "Violin Plot")
```

Arranging Multiple Plots

Now that we created the 3 plots, let's arrange them with `ggarrange()`

```
ggarrange(plotA, plotB, plotC,
          nrow = 2, ncol = 2,
          common.legend = TRUE,
          labels = c("A", "B", "C"),
          legend = "bottom")
```



Arranging Multiple Plots

This looks fine, but it's kind of smushed.

What you actually want to see is the bottom biolin plot taking up the entire 2 columns (e.g., spanning the entire width of this newly created plot).

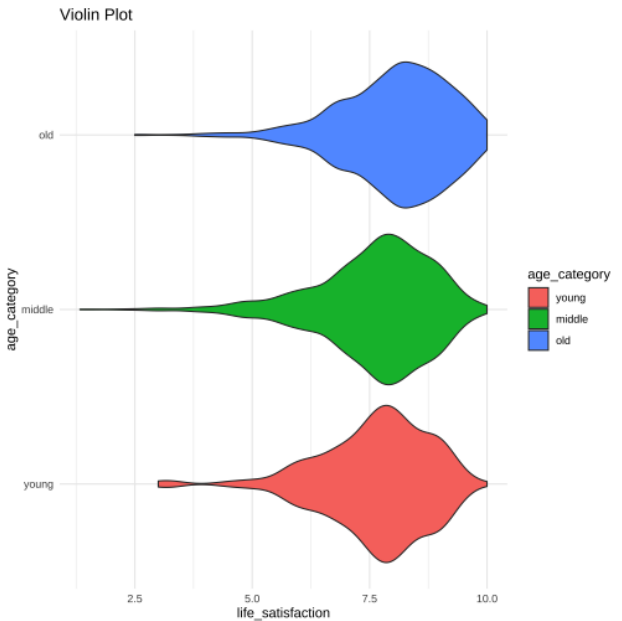
It would be even better if the violin plots had the distributions stacked vertically, rather than horizontally.

Let's do it!

Arranging Multiple Plots

To change the violin plot, all we need to do is flip the coordinates.

```
plotCNew <- ggplot(data = midus, aes(x = age_category, y = life_satisfaction)) +
  geom_violin(aes(fill = age_category)) +
  theme_minimal() +
  labs(title = "Violin Plot") +
  coord_flip()

plotCNew
```



Arranging Multiple Plots

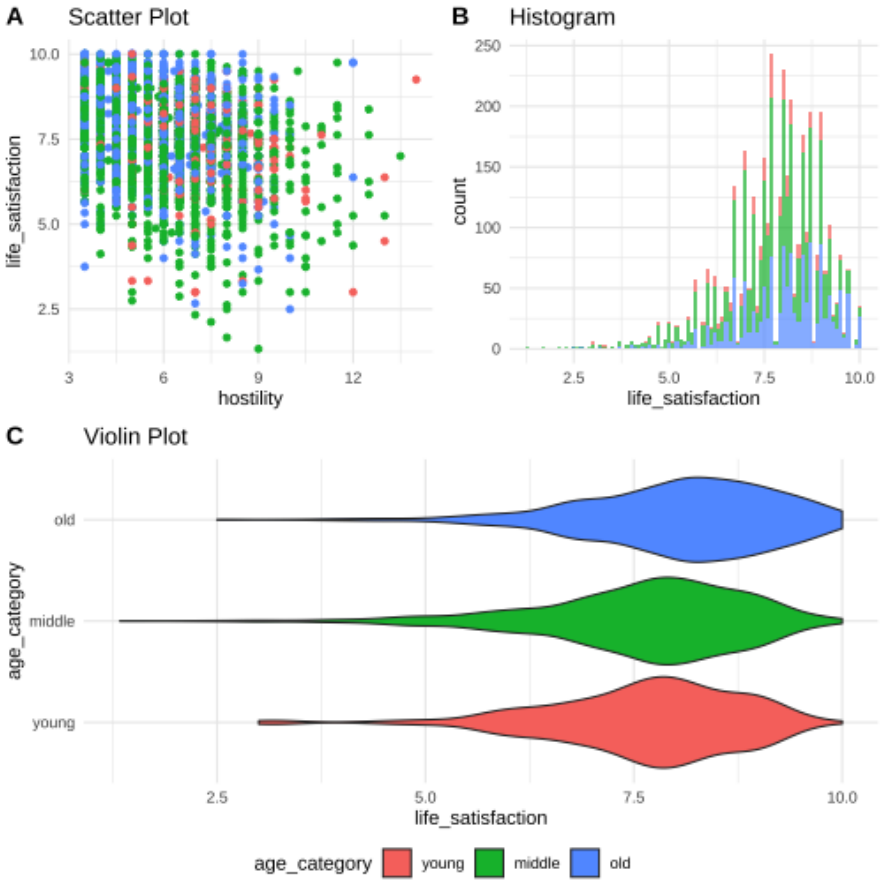
Ok, that looks good.

Now we are going to nest 2 `ggarrange()` functions:

- The 1st (inner most) `ggarrange` will combine plots A & B into a single figure. Here, we want 1 row, 2 columns.
- The 2nd (outer most) `ggarrange` will take the one from above, and combine it with our newly created flipped violin plot. We will keep this as 1 column, and 2 rows.
    - This means that C has to take up the full width

Arranging Multiple Plots

```
ggarrange(ggarrange(plotA, plotB,
                    ncol = 2,
                    labels = c("A", "B"),
                    legend = "none"),
          plotCNew,
          nrow = 2,
          common.legend = TRUE,
          legend = "bottom",
          labels = c("", "C"))
```



Arranging Plots

A final note about `ggarrange()`:

- It does not like it when either the `ncol =` or `nrow =` parameters are set to equal `1`.
- If you want a plot with 1 column and 3 rows, do *NOT* specify `ncol = 1`. Instead, use `nrow = 3`.

Adding Stuff

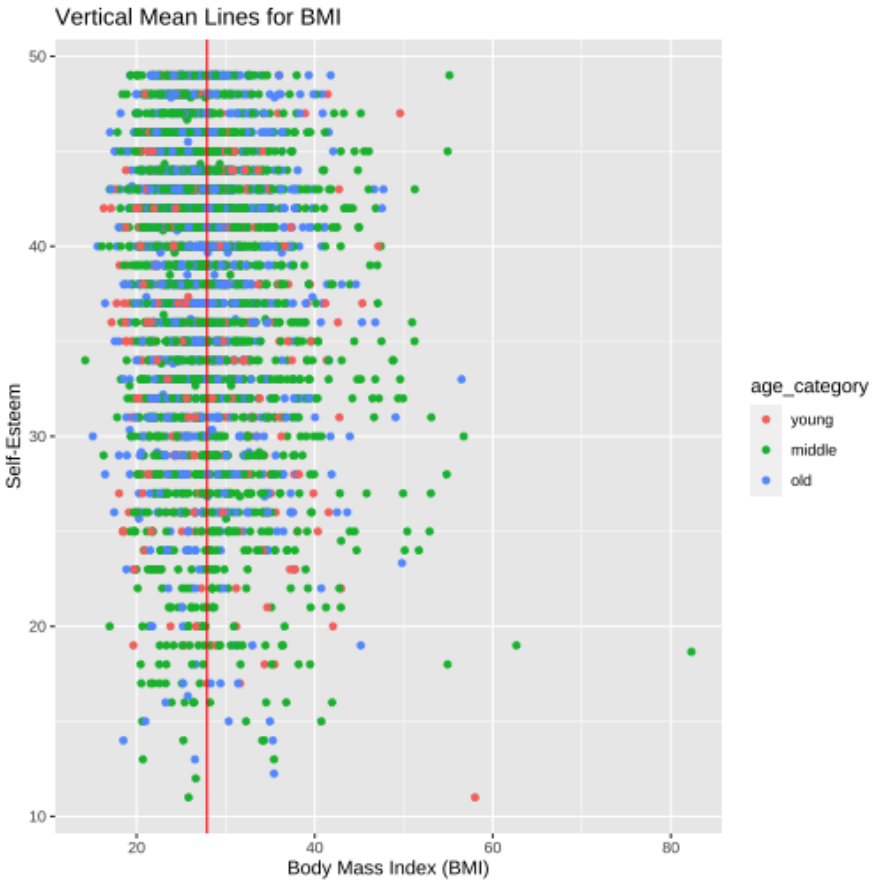
- Lines
- Text

Horizontal and Vertical Lines

- Use `geom_vline` or `geom_hline`
- You'll need to specify an x or y intercept, respectively
  - This intercept is based on the actual scales on the graph!

Horizontal and Vertical Lines

```
ggplot(data = midus,
       aes(x = BMI,
           y = self_esteem)) +
  geom_point(aes(color = age_category)) +
  labs(x = "Body Mass Index (BMI)",
       y = "Self-Esteem",
       title = "Vertical Mean Lines for BMI
    geom_vline(xintercept = mean(midus$BMI)
               color = "red")
```



By groups?

What if we wanted to make a vertical line that reflects the mean of some variable *per level of a categorical variable*?

We need to create a data.frame that contains the means for each level of the factor, store this information in an object, and then call that object from within the plot.

This is where `tidyverse` is extremely useful!

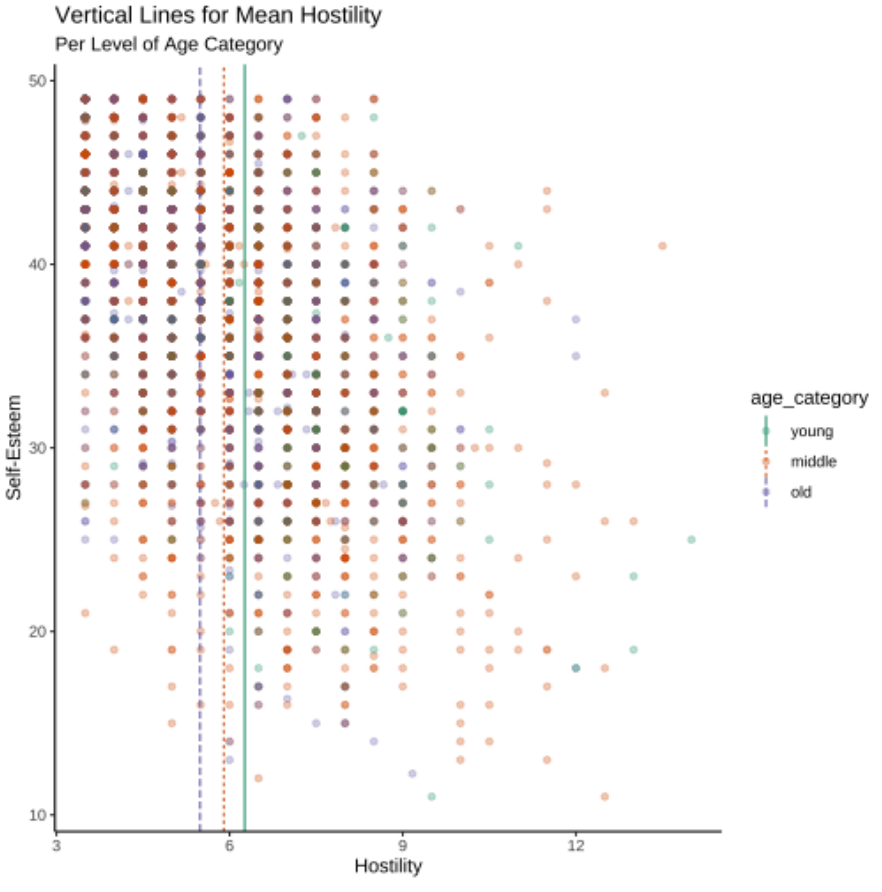
By groups

```
meansHostility <- midus %>%
  group_by(age_category) %>%
  summarize(xint = mean(hostility))

meansHostility
```

```
## # A tibble: 3 x 2
##   age_category  xint
##   <fct>        <dbl>
## 1 young         6.26
## 2 middle        5.90
## 3 old           5.49
```

By groups

```
ggplot(data = midus,
       aes(x = hostility,
           y = self_esteem)) +
  geom_point(aes(color = age_category),
             alpha = .3) +
  labs(x = "Hostility",
       y = "Self-Esteem",
       title = "Vertical Lines for Mean Hos
       subtitle = "Per Level of Age Categor
  theme_classic() +
  scale_color_brewer(palette = "Dark2") +
  geom_vline(data = meansHostility,
             aes(xintercept = xint,
                 color = age_category,
                 linetype = age_category))
```



Regression Lines

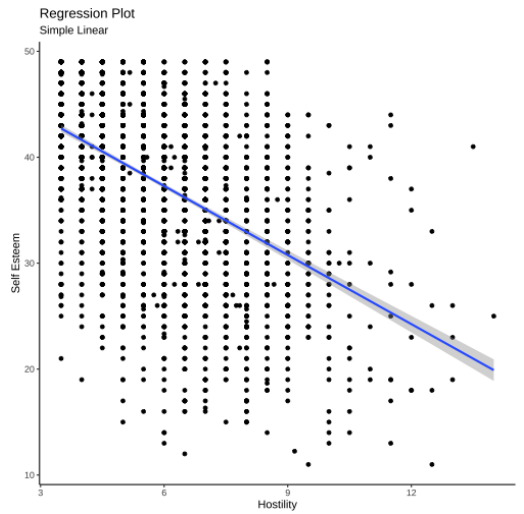
For simple linear regression, you only need to add `geom_smooth`

- By default, it will add a Loess line -- usually kinda curvy
- For the most part, you'll want to specify that it's a linear model using the `method = "lm"` argument

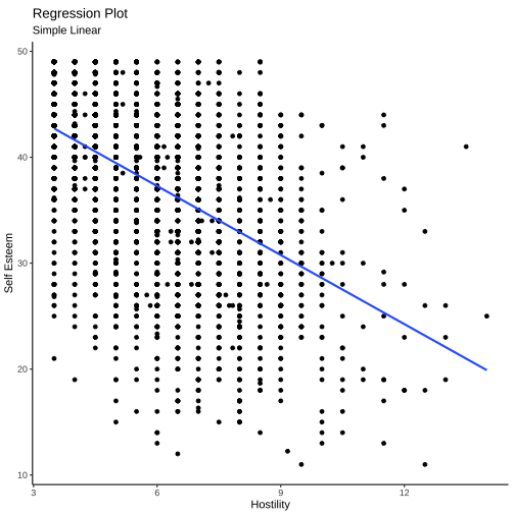
For complex interactions from multiple regressions, check out the `ggpredict()` function from the `ggeffects` package (beyond our scope, sadly).

Regression Lines

```
ggplot(data = midus,
       aes(x = hostility,
           y = self_esteem)) +
  geom_point() +
  theme_classic() +
  labs(title = "Regression Plot",
       subtitle = "Simple Linear",
       x = "Hostility",
       y = "Self Esteem") +
    geom_smooth(method = "lm")
```




```
ggplot(data = midus,
       aes(x = hostility,
           y = self_esteem)) +
  geom_point() +
  theme_classic() +
  labs(title = "Regression Plot",
       subtitle = "Simple Linear",
       x = "Hostility",
       y = "Self Esteem") +
    geom_smooth(method = "lm", se = FALSE)
```



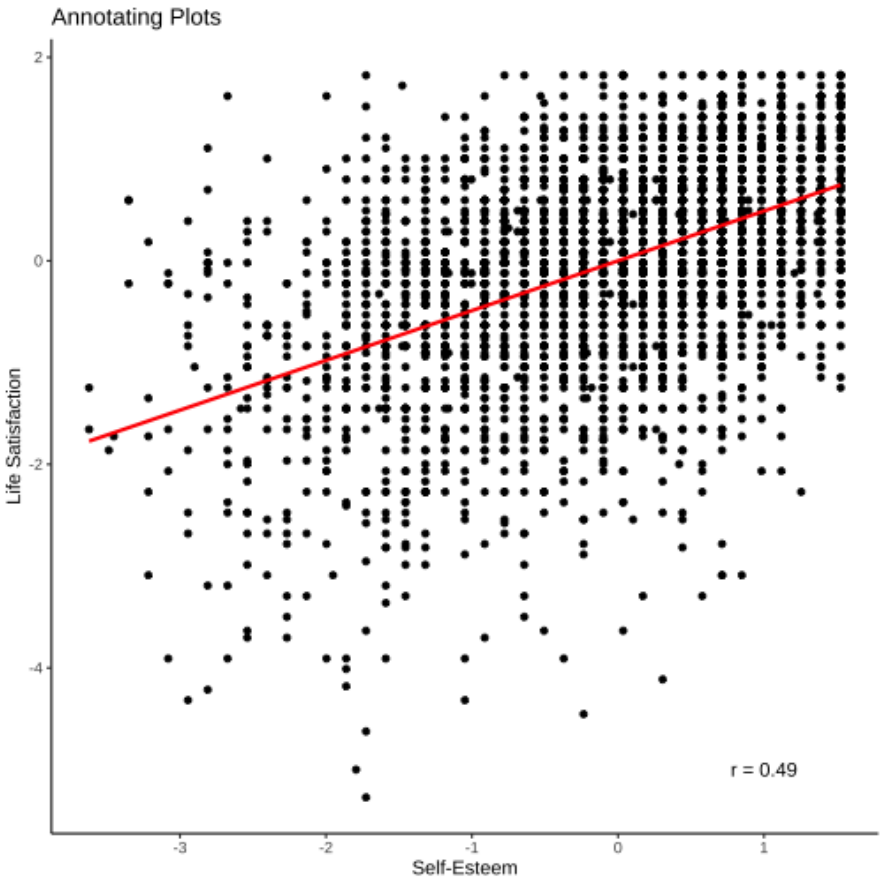
Adding text

Sometimes you might want to add something like an R^2 value or a correlation coefficient or some sort of text to your plot, usually in a corner. We can do that!

Let's use a very nerdy stats example. The coefficient of a simple linear regression of 2 standardized variables is equal to the correlation of those variables.

Adding text

```
# get correlation
corCoef <- round(x = cor(midus$self_esteem,
                         midus$life_satisfa
                 digits = 2)

ggplot(data = midus,
       aes(x = scale(self_esteem),
           y = scale(life_satisfaction))) +
  geom_point() +
  geom_smooth(method = "lm",
              se = FALSE,
              color = "red") +
  theme_classic() +
  labs(title = "Annotating Plots",
       x = "Self-Esteem",
       y = "Life Satisfaction") +
  annotate(geom = "text",
           x = 1,
           y = -5,
           label = paste0("r = ", corCoef))
```



Next up...

- Error bars
- Jittering
- Adding layers
- Debugging plots