Data Visualization with `ggplot2`

Allison Theobold

April 03, 2019

"Visualization gives you answers to questions you didn't know you had." - Ben Schneiderman

"The simple graph has brought more information to the data analyst's mind than any other device." -John Tukey

Workshop Learning Objectives

- Create scatterplots, line graphs, boxplots, and bar charts using `ggplot`.
- Define global plotting settings for `ggplot` plots.
- Understand the purpose of and apply faceting in `ggplot`.
- Change aesthetics of a `ggplot` plot (axis labels, color, text size).

Data for Workshop

The data are gathered from end of semester student evaluations for a large sample of professors from the University of Texas at Austin. These data are merged with descriptors of the professors and the classes. In addition, six students rated the professors' physical appearance. The result is a data frame where each row contains a different course and each column has information on either the course or the professor. Data are available through the Open Intro Statitics course webpage.

This is a real dataset that has been used in a publication, *Looking Good on Course Evaluations*, published in 2013 in CHANCE volume 26 issue 2.

Load the data into your work space and inspect it to answer the following questions:

- How many observations (rows) do the data have?
- How many variables (columns) do the data have?
- What class are the variables in the data frame (e.g. numeric, double, character, factor)?

```
# To Download the data (already downloaded)
# download.file("http://www.openintro.org/stat/data/evals.RData", destfile = "evals.RData")

# Load in the evals data (evals.RData)
evals <- read.csv("data/evals.csv")

# Inspect the data here!
```

Data Visualization using `ggplot`

It great to have the knowledge to make boxplots, histograms, scatter plots, and line graphs all in base `R`, but don't you wish that there was less to memorize? Does it seem like each of these plots really is just a slight

modification of the other? Enter **ggplot2**. **ggplot2** is a package which makes data visualization simpler and more concise.

ggplot2 is one of the many packages available in R, developed and maintained by Hadley Wickham. Some of the functions that you have seen in previous workshops, or are potentially familiar with (e.g. `str`, `summary`) come built into R. In order to use the elements of the **ggplot2** package, you have to first install the package (only once) and then load the package (every time). The **ggplot2**, **dplyr**, and **tidyr** packages are included in the **tidyverse** package. This is an "umbrella-package" that installs several packages useful for data visualization and analysis which work together well, such as **purrr**, **readr**, and **tibble**.

First download the **tidyverse** package to your work space, using the Packages tab at the bottom-right corner. Once the package is downloaded, load the **tidyverse** package by typing:

```
library(tidyverse)    ## load the tidyverse packages, incl. dplyr
```

Notice that when loading in the **tidyverse** package it tells you which packages were added to your work space (**ggplot**, **purrr**, **tibble**, **dplyr**, **tidyr**, **stringr**, **readr**, **forcats**). The loading also tells you what functions have been masked (overwritten) from the loading of **tidyverse** (`filter` and `lag`). Lastly, it will tell you what version of R the package was built under. Running `R.version` will give you the version of R that you are currently working in, and will give you a good idea if you should update R!

What is ggplot2?

ggplot2 is a data visualization package that is more versatile in creating create complex plots than all of the functions you've been remembering in base R. **ggplot2** utilizes the *grammar of graphics* in a programmatic interface, which describes and builds graphs in an intuitive manner. With **ggplot2** you will learn **one** system for plotting and have the ability to apply it to many different situations. These minimal changes from plot to plot are the key advantages of **ggplot2**!

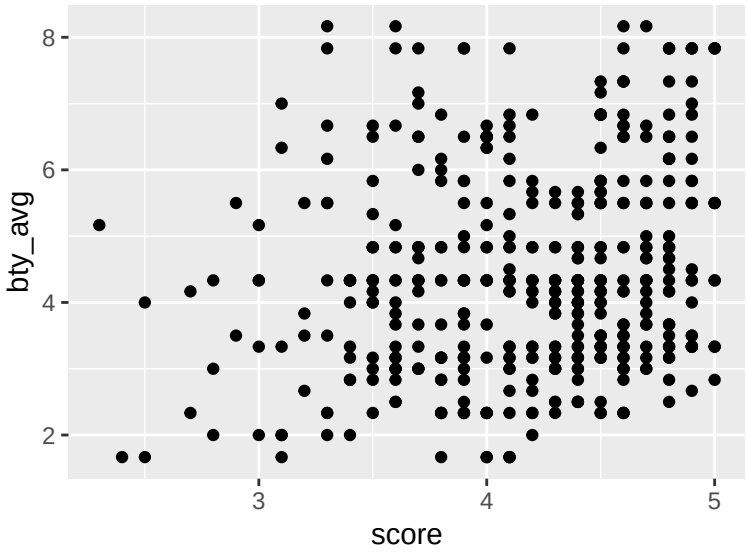
`ggplot` prefers for the data to be in "long" format, where every variable has it's own column and every row is an observation. Correctly formatted data will save you lots of time when you are creating plots with `ggplot`. For assistance in formatting your data, see the Data Wrangling workshop (Theobold, 2019) on the Montana State Library, tutorial video (link) and RStudio Cloud materials (link).

First Steps

Let's use our dataset to answer the question that these authors were interested in: do instructors with higher beauty ratings receive higher course evaluation scores? You may have an idea of what answer you might expect, but let's try and make our ideas more precise. Do you expect for the relationship to be positive or negative? Linear or non-linear?

To plot `bty_avg` on the y-axis and `score` on the x-axis run the following code:

```
ggplot(data = evals) +
  geom_point(mapping = aes(y = bty_avg, x = score))
```



This plot shows a positive relationship between the average course evaluation and the average beauty score, but the relationship is a bit messy.

`ggplot` Template

With `ggplot` you begin with a coordinate system that you add layers to. The first argument to `ggplot` is the data frame that you wish to use to create the plot. The statement `ggplot(data = evals)` creates an empty graph. Give it a try!

Next, you add layers to your graph. The `geom_point()` function adds a layer of points to the plot, using the `+` operator, creating the scatterplot. `ggplot2` comes with many possibly `geom` functions, each of which adds a different layer to your plot. We will discuss the `geom_line`, `geom_smooth`, `geom_boxplot`, and `geom_bar` functions today.

Every `geom` function accepts a `mapping` argument, which defines how the variables in the data frame will be plotted. The `mapping` argument **always** comes with the `aes()` function, specifying what variables to map to which plot aesthetics.

A template for making plots with `ggplot` is:

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))

# Note: the + must go at the END of each line, NOT a the beginning of the next line
```

Exercise 1: Make a scatterplot of age verses course evaluation score.

```
# Exercise 1 code here!
```

Exercise 2: Make a scatterplot of rank and gender. Why is the plot not useful?

```
# Exercise 2 code here!
```

Aestheics

It appears that there are some professors with low average evaluation scores, but relatively high average beauty scores. If we are interested in investigating what these professors look like (pun intended!), we could add a third variable (e.g. `gender`, `minority`, `language`) to our two-dimensional scatterplot, by mapping it to an aesthetic.

Aesthetics are visual properties of your plot, they include things like the color, shape, and size of points. You can change the display of a point by changing it's aesthetic properties. Typically, these aesthetics will take on discrete values, so it is more natural to refer to the "levels" of aesthetics.

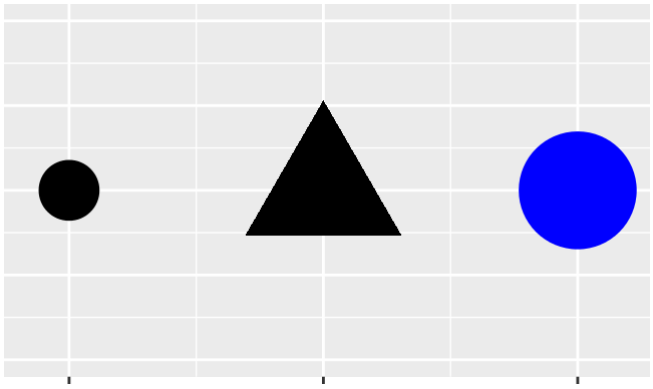
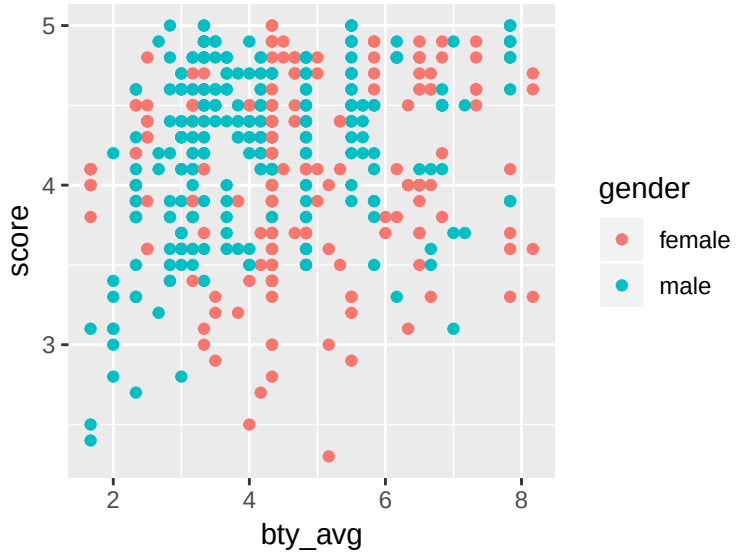


Figure 1: Wickham, H. & Grolemund, G. (2017) *R for Data Science*. Sebastopol, California: O'Reilly.

You can include information about the data by mapping variables in your dataset to aesthetics in your plot. For example, you can map the colors of your points to the `gender` variable, to reveal the sex each professor identified as. In the code below I've made two changes:

1. I've moved the `mapping` argument to the `ggplot` statement – this declares a mapping for **every** geom the `ggplot`S uses.
2. I've added a new `mapping` argument specific to the `geom_point` statement – this declares that the points should be colored by the levels of the `gender` variable.

```
ggplot(data = evals, mapping = aes(y = score, x = bty_avg)) +
  geom_point(mapping = aes(color = gender))
```




```
# can use color or colour in the aes mapping
```

When you map a variable to the name of an aesthetic, `ggplot` will automatically assign a unique level to each value of that variable. Here, `gender` (really sex) takes on binary (two) values, male and female. This is the same process that is enacted when you convert a variable to a factor (using `as.factor` or `factor`). The levels of this variable are then assigned to a different color and plotted. Additionally, `ggplot` will automatically add a legend that explains what color goes with which level of a variable.

We could have instead mapped the variable `gender` to the size of a point by including the `size = gender` argument to the aesthetic mapping. What happens if you do this? Does this plot do a better or worse job of conveying information about how sex relates to beauty and course eval scores?

```
# Change the above code to map gender to a point's size
```

There are other aesthetic mappings we could have considered, such as `alpha` (point transparency) and `shape`. For every aesthetic you include, the `aes()` function will associate that aesthetic with a variable and use it to create that layer of the plot. Once you map an aesthetic, `ggplot` will handle the rest, choosing what scale to use and creating a legend for the mapping.

Exercise 3: Map the `shape` aesthetic to the variable `rank`. What happens if you map `rank` to a second aesthetic variable?

```
# Exercise 3 code here!
```

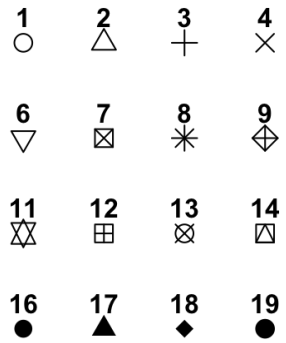
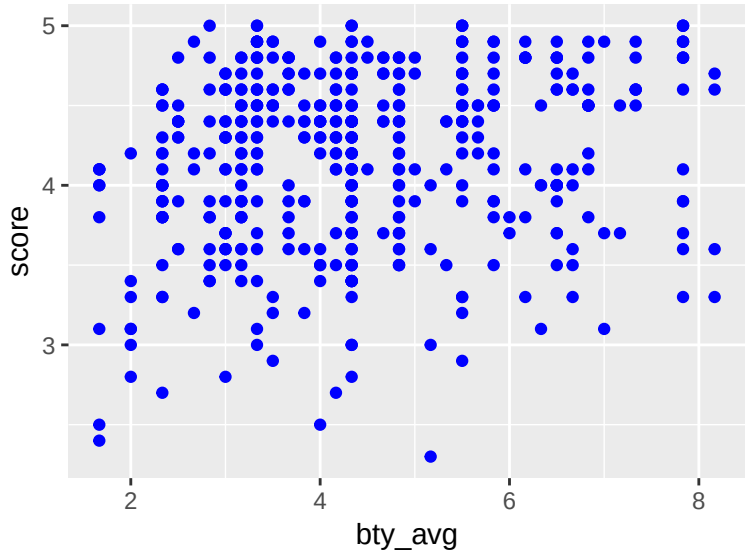


Figure 2: R has 20 built in plotting characters, identified by number.

We could also set the aesthetics of our plot manually, by specifying them *outside* of the `aes()` function. For example, if I wanted all of the points in my scatterplot to be blue, I would use the following code:

```
ggplot(data = evals, mapping = aes(y = score, x = bty_avg)) +
  geom_point(color = "blue")
```



This is a way to change characteristics of the plot, but it does not convey any additional information about the data. If you wish to specify these outside of the `aes()` function you need to specify :

- color: a character string of the color you wish to use (e.g. “dark green”)
- shape: the number of plotting character you wish to use (see Figure 2)
- size: the size of the point (in mm)

Exercise 4: Map a continuous variable to `color`, `size`, and `shape` and see what happens. How do these aesthetic mappings differ for continuous variables as compared to categorical variables?

```
# Excercise 4 code here!
```

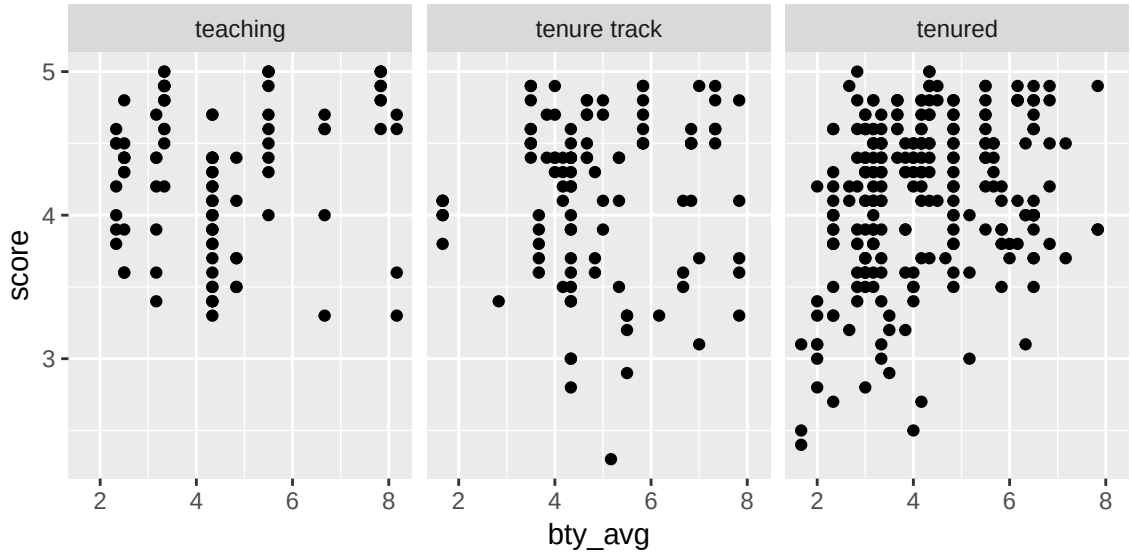
Exercise 5: Use what you have learned to create a scatterplot of `score` over `rank` with `minority` showing in different colors. Is this a good display of this type of data?

```
# Excercise 5 code here!
```

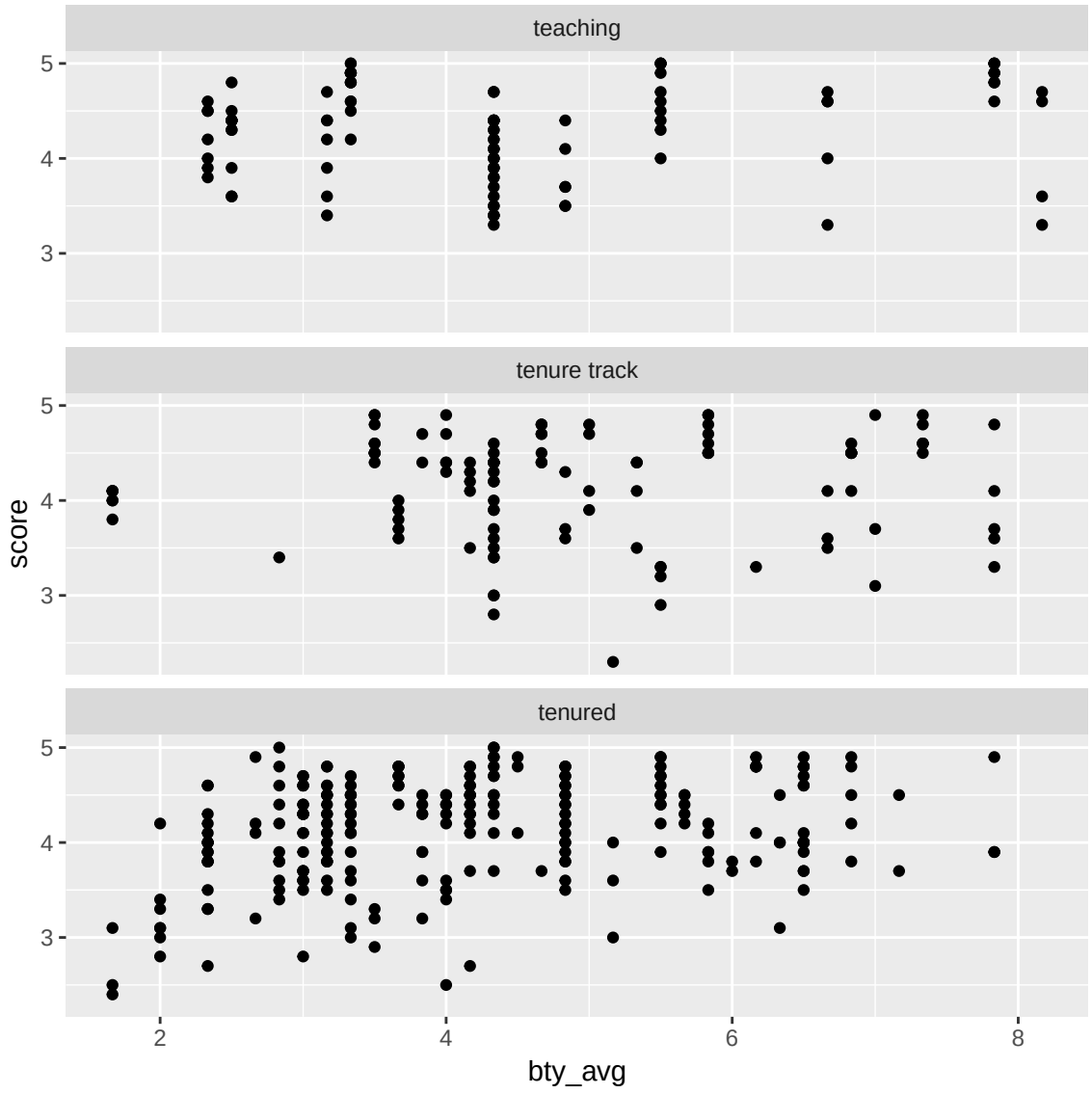
Facets

Another way to add information to your plots is to divide the plot into different *facets*, or subplots, for each subset of the data. For example, if we wanted to compare the relationship between average beauty score and average evaluation score for professors of different tenure rankings, we could have a simpler comparison by splitting `rank` apart and plotting this relationship for each different value of rank.

```
ggplot(data = evals, mapping = aes(y = score, x = bty_avg)) +
  geom_point() +
  facet_wrap(. ~ rank, nrow = 1) # plots different ranks in columns
```



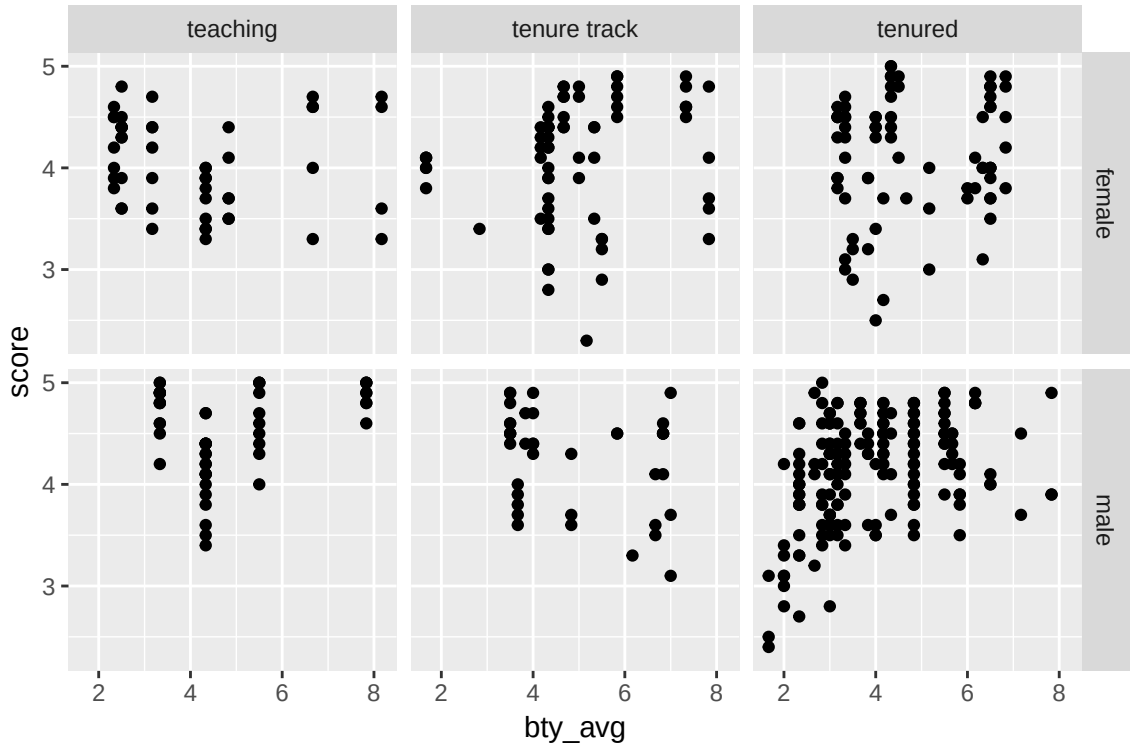
```r
ggplot(data = evals, mapping = aes(y = score, x = bty_avg)) +
  geom_point() +
  facet_wrap(rank ~ ., ncol = 1) # plots different ranks in rows
```



If you want to facet by a single variable, then you use the `facet_wrap()` function. The first argument is a formula of the form `<ROW VARIABLE> ~ <COLUMN VARIABLE>`. If you want your variable to appear on the rows, then you pass `<VARIABLE> ~ .` to `facet_wrap`. If you want your variable to appear in the columns, then you pass `. ~ <VARIABLE>` to `facet_wrap`. The variable that you pass to `facet_wrap` should be discrete, with preferably not a terrible number of levels.

If you would like to facet your plot by two variables, then you use the `facet_grid()` function. The first argument to `facet_grid` is identical to `facet_wrap`, where the first variable will be plotted as rows and the second variable will be plotted as columns (`<ROW VARIABLE> ~ <COLUMN VARIABLE>`).

```
ggplot(data = evals, mapping = aes(y = score, x = bty_avg)) +
  geom_point() +
  facet_grid(gender ~ rank)
```



Inspecting `facet_wrap()` we see that there are `nrow` and `ncol` arguments we can specify. What other options do we have to control the layout of these plots? Why does `facet_grid` **not** have `nrow` and `ncol` arguments?

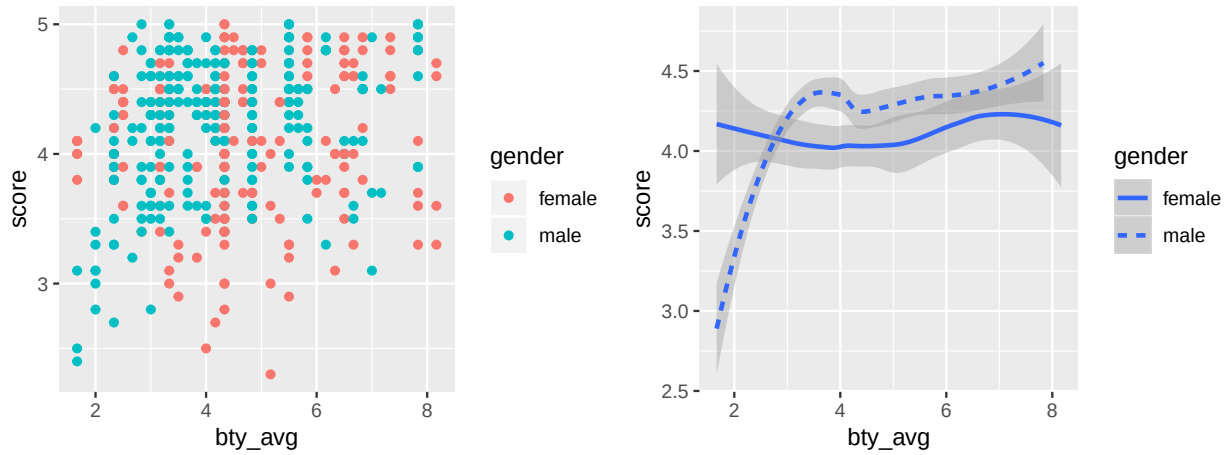
Exercise 6: Facet `bty_avg` and `score` by `language` and `rank`. What do the empty cells mean?

```
# Exercise 6 code here!
```

Exercise 7: Consider the first plot created in this section. Create the same comparison of variables, but plot the values of `rank` using different colors. What are the advantages of faceting over using a color aesthetic? Would your opinion change if you had a larger dataset?

```
# Exercise 7 code here!
```

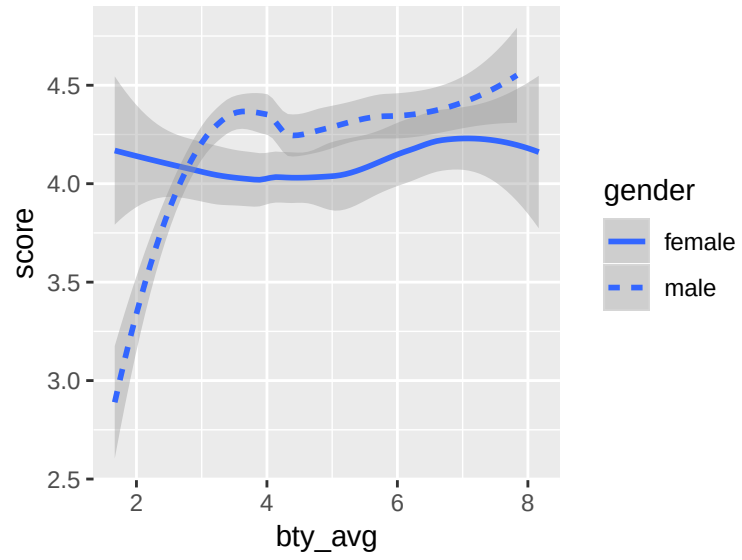
Different Geometric Objects



What is the difference between these two plots? They both plot the same data and have the same axes, but the plot on the right uses different *geoms* to represent the data. As we saw previously, to make a scatterplot we used a point geom. To create a different plot with the same data, we could have used a bar geom (for a bar chart), a line geom (for a line graph), or a boxplot geom (for a boxplot). To change the geom of your plot you change the geom function that `ggplot` is using.

For example, to make the plot on the right I used the following code:

```
ggplot(data = evals, mapping = aes(y = score, x = bty_avg)) +
  geom_smooth(mapping = aes(linetype = gender))
```

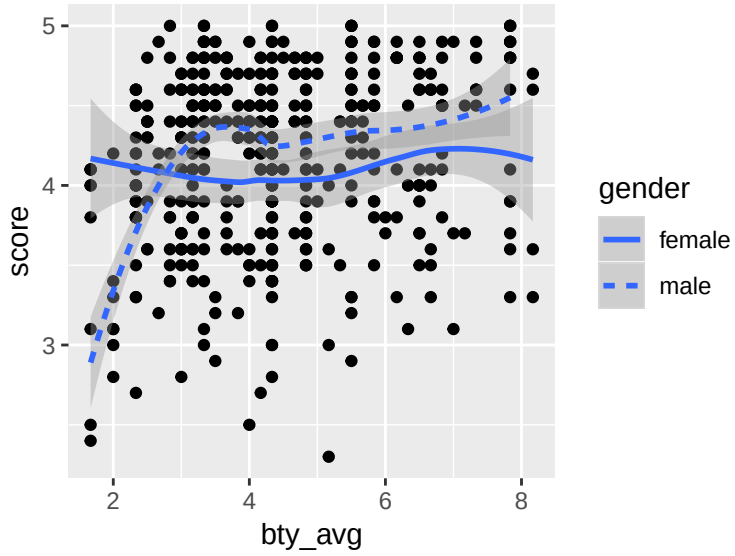


Every geom takes aesthetic values, but not every aesthetic works for every geom. For example, you can change the shape of a point, but you cannot change the shape of a line. Instead, with a line you change the linetype. **`ggplot2`** provides over 30 geoms, with more available through extension packages, each of which take different aesthetic arguments. For a comprehensive overview see the additional handout given to you or go here to access the **`ggplot2`** cheatsheet (link).

Multiple Geoms One Plot

Displaying multiple geoms in one plot is as simple as adding another `geom_` statement to your plotting code. If we wished to see both the raw data (points) and trend lines (smoothers) for each sex of professors we could use the following code:

```
ggplot(data = evals, mapping = aes(y = score, x = bty_avg)) +
  geom_point() +
  geom_smooth(mapping = aes(linetype = gender))
```



Notice here that the points are not differentiated by sex, only the lines. We can locally define mappings for each geom layer, so points and lines are differentiated.

Exercise 8: Run this code in your head and predict what the output will be.

```
ggplot(data = evals,
       mapping = aes(x = bty_avg, y = score, color = rank)) +
  geom_point() +
  geom_smooth(se = FALSE)
```

Exercise 9: Create a plot where all professor's scores are displayed, but each point is differentiated by what language was the professor's native language. Draw a trend line over the top of these points, where the trend line represents the trend between average beauty score and average course evaluation for **all** of these professors.

```
# Exercise 9 code here!
```

Adding Labels and Changing Themes

An important part of communicating with data visualizations is to use informative labels for a graphic. You can declare new label options inside of the `labs()` function. Some options you can use are:

`labs(`

- `x = "New x axis label"`
- `y = "New y axis label"`
- `title = "Add a plot title"`
- `caption = "Add a caption below plot"`
- `<aes used> = "New <aes> legend title" )`

```
ggplot(data = evals,
       mapping = aes(x = bty_avg, y = score, color = gender)) +
  geom_point() +
  labs(x = "Average Beauty Score",
       y = "Average Course Evaluation Score",
       title = "Course Evaluations and Beauty",
       color = "Sex")
```

Another aspect of creating effective data visualizations is making them easily digestible. Themes can aid in this adventure, allowing for you to change the background of a graphic. The standard `ggplot` them uses a grey background with a grid, which some people may find difficult to read. You can change this aspect to a `ggplot` by adding a theme statement. Some `theme` options are:

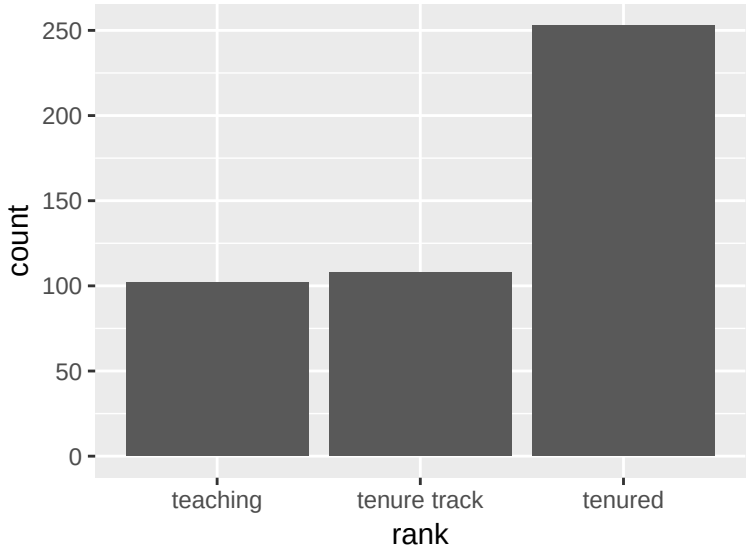
- `theme_bw()` – makes the background white
- `theme_classic()` – makes a "classic" base `R` appearance plot
- `theme(panel.grid = element_blank())` – removes the grid from the plot
- `theme(legend.position = "CHOICE")` – places legend at location of choice, "bottom", "top", "left", or "right"
- `theme(axis.text = element_text(size = 15))` – makes the text of the axis 15 pt font
- `theme(axis.text.x = element_text(color = <"COLOR">, size = <#pt>, angle = <#>, hjust = <#>, vjust = <#>))` – all options for x-axis text, where `angle` is angle of text with axis, `hjust` and `vjust` are options between 0 and 1 for where the text should be justified (0 = left, 1 = right)
- `theme(axis.text.y = element_blank())` – removes y-axis text
- `theme(axis.ticks.y = element_blank())` – removes y-axis tick marks
- `theme(panel.spacing.y = unit(0.25, 'lines'))` – declares how much space you want between facet panels and that they should have line borders

Transformations

Bar Plots

At first glimpse, you would think that a bar plot would be simple to create, but bar plots reveal a subtle nuance of the plots we have created thus far. The following bar chart displays the total number of professors in the `evals` dataset, grouped by their tenure ranking. The plot shows that there are about the same number of teaching and tenure-track faculty, and almost twice the amount of tenured faculty.

```
ggplot(data = evals) +
  geom_bar(mapping = aes(x = rank))
```



The x-axis displays the three levels of `rank`, a variable in the `evals` dataset. On the y-axis `count` is displayed, but `count` is **not** a variable in our dataset! Where did `count` come from? Graphs, such as the scatterplots, display the raw values of your data. Other graphs, like bar charts and boxplots, calculate new values (from your data) to plot.

- Bar charts and histograms bin your data and then plot the number of observations that fall in each bin.
- Boxplots find summaries of your data (min, max, quantiles, median) and plot those summaries in a tidy box, with "outliers" (data over 1.5*IQR from min/max) plotted as points.
- Smoothers (as used in `geom_smooth`) fit a model to your data (you can specify, but we used the `loess` default) and then plot the predictions from that model (with associated confidence intervals).

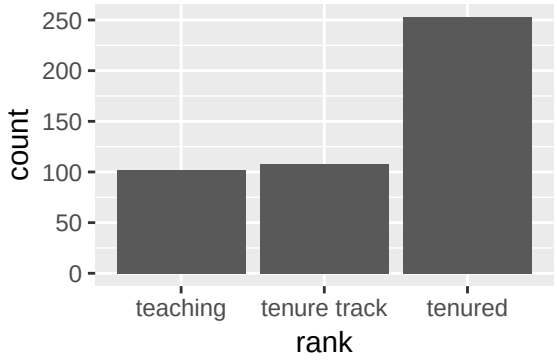
To calculate each of these summaries of the data, `R` uses a different statistical transformation, or *stat* for short. With a bar chart this looks like the following process:

1. `geom_bar` first looks at the entire data frame
2. `geom_bar` then transforms the data using the `count` statistic
3. the `count` statistic returns a data frame with the number of observations (rows) associated with each level of `rank`

4. `geom_bar` uses this summary data frame, to build the plot – levels of `rank` are plotted on the x-axis and `count` is plotted on the y-axis

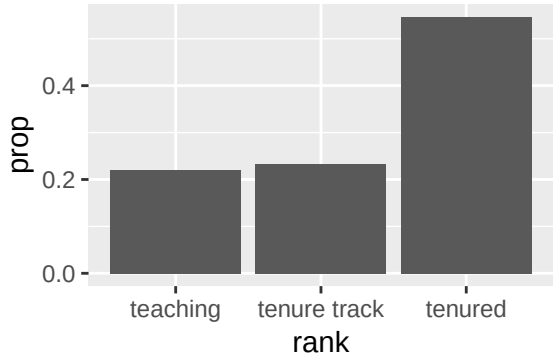
Generally, you can use geoms and stats interchangeably. This is because every geom has a default stat and visa versa. For example, the following code produces the same output as above:

```
ggplot(data = evals) +
  stat_count(mapping = aes(x = rank))
```



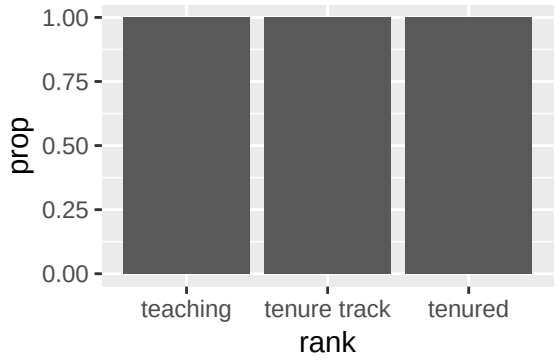
If you so wish you could override the default stat for that geom. For example, if you wanted to plot a bar chart of proportions you would use the following code to override the `count` stat:

```
ggplot(data = evals) +
  geom_bar(mapping = aes(x = rank, y = ..prop.., group = 1))
```



Exercise 10: Why do we need to set `group = 1` in the above proportion bar chart? In other words, what is wrong with the plot below?

```
ggplot(data = evals) +
  geom_bar(mapping = aes(x = rank, y = ..prop..))
```



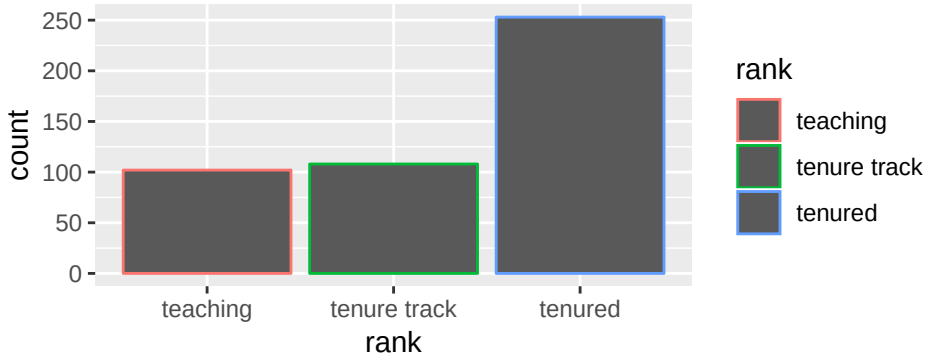
Exercise 11: What variables does `stat_smooth` compute? How can you control its behavior?

Colored and/or Stacked Bar Charts

Another piece of visual appeal to creating a bar chart is the ability to use colors to differentiate the different groups, or to plot two different variables in one bar chart (stacked bar chart). Let's start with adding color to our bar chart.

As we saw before, to add a color aesthetic to the plot we need to map it to a variable. However, if we use the `color` option that we used before we get a slightly unsatisfying result.

```
ggplot(data = evals) +
  geom_bar(mapping = aes(x = rank, color = rank))
```



We notice that the color only appears in the outline of the bars. For a bar chart, the aesthetic that we are interested in is the `fill` of the bars.

Exercise 12: Change the above code so that each bar is filled with a different color.

```
# Exercise 12 code here!
```

Now suppose you are interested in whether the number of women teaching differs by tenure ranking. This would require for you to create a bar plot with two categorical variables. You have two options:

1. each of the bars for gender could be stacked within a ranking *OR*
2. the bars for gender could be side-by-side within a ranking

Let's see how the two approaches differ. To stack bars of a second categorical variable we would instead use this second categorical variable as the `fill` of the bars. Run these two lines of code and see how they differ.

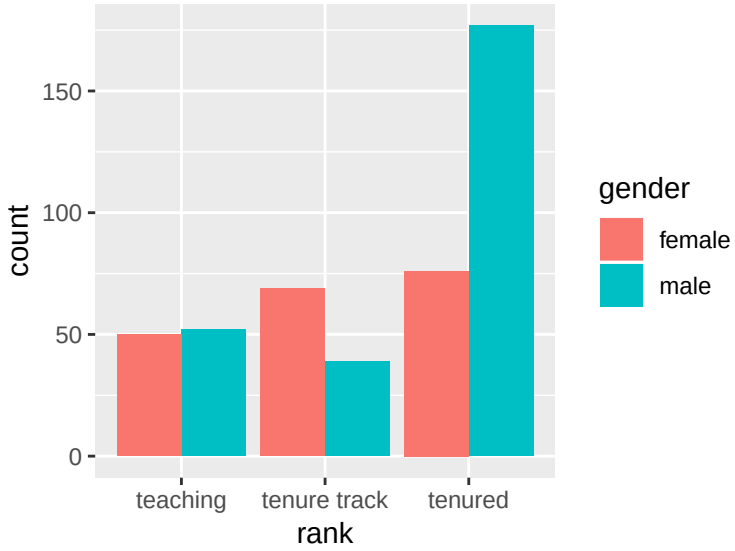
```
ggplot(data = evals) +
  geom_bar(mapping = aes(x = rank, fill = gender))

ggplot(data = evals) +
  geom_bar(mapping = aes(x = rank, fill = gender), position = "fill")
```

In the first plot, the position was chosen automatically, but in the second plot the `position` argument was made explicit. What changes did this make in the plots?

Another position option is `position = "dodge"` which places the previously overlapping objects directly next to each other. This position makes it easier to directly compare individual values.

```
ggplot(data = evals) +
  geom_bar(mapping = aes(x = rank, fill = gender), position = "dodge")
```



Exercise 13: There is another `position` adjustment that is not useful for bar charts but can be very helpful for scatterplots. The `position = "jitter"` argument can be added to a scatterplot to add random noise to each data point.

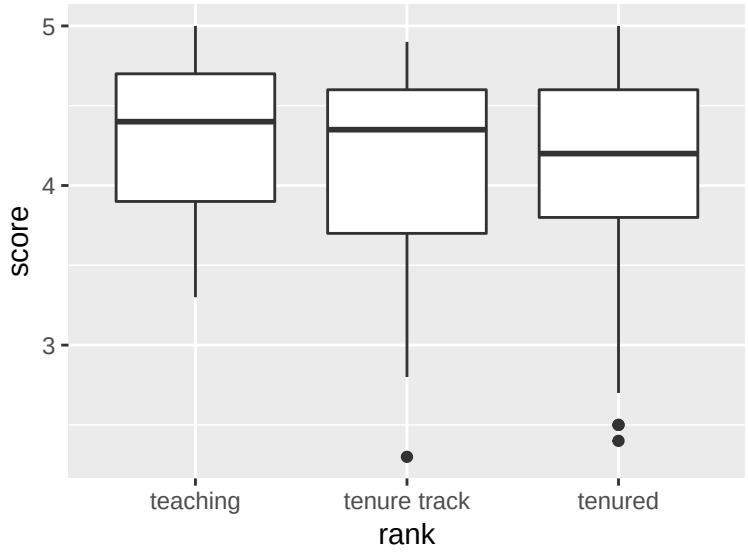
Add this argument to the very first scatterplot we made. Compare the difference it made on the plot. What are advantages of this? What are disadvantages?

```
# Exercise 13 code here!
```

Boxplots

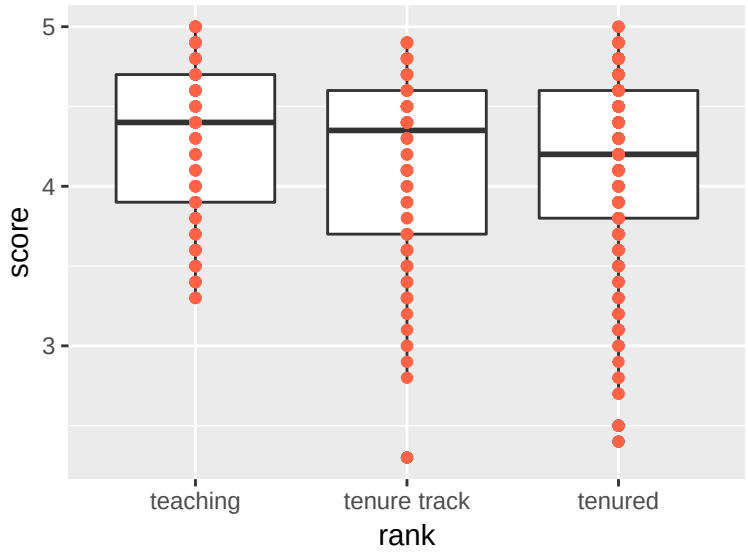
We could instead use boxplots to visualize the distribution of course evaluation scores within each rank of professor:

```
ggplot(data = evals, mapping = aes(x = rank, y = score)) +
  geom_boxplot()
```



An unfortunate aspect of a boxplot is that it only plots aggregates (summaries) of the data. By adding points to boxplot, we can have a better idea of what the data look like and their distribution:

```
ggplot(data = evals, mapping = aes(x = rank, y = score)) +
  geom_boxplot() +
  geom_point(color = "tomato")
```

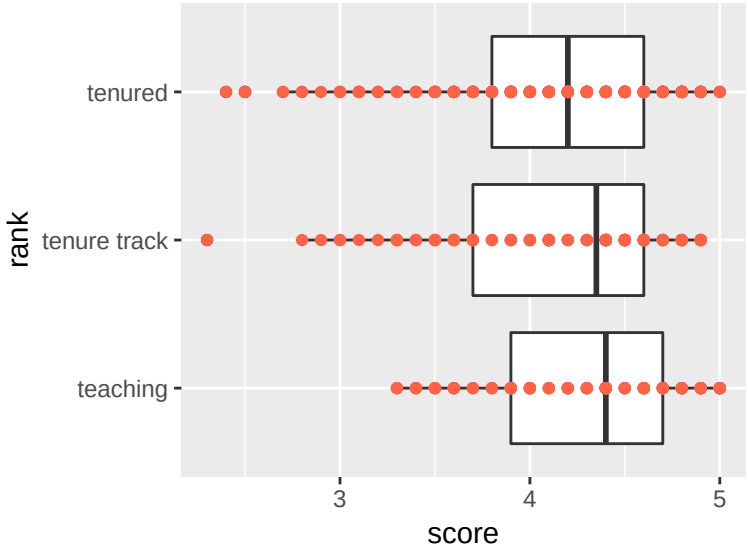


Note how the points are placed on top of the boxplot, this is because we are first plotting the boxplot layer and then plotting the point layer over it. For large datasets, the points could cover the boxplot enough that you are unable to see that layer.

Exercise 14: How would you change the above code if your dataset was very large, but you still wanted to visualize the distribution of points and the boxplot together?

If you would rather have your boxplot (or bar chart) oriented horizontally, you can add the option to flip the coordinates of your plot:

```
ggplot(data = evals, mapping = aes(x = rank, y = score)) +
  geom_boxplot() +
  geom_point(color = "tomato") +
  coord_flip()
```



Exercise 15: Boxplots are useful summaries, but hide the *shape* of the distribution. For example, a boxplot would not indicate to us if there is a bimodal distribution. An alternative a boxplot is a violin plot (sometimes known as a beanplot), where the shape (density) of the distribution of observations is drawn.

Replace the flipped boxplot above with a violin plot; see `geom_violin()`.

```
# Exercise 15 code here!
```

Workshop created on: 2019-04-03 11:10:38