Week 12 Module 2: ggplot2 Basics

CSCI E-5a: Programming in R

Module Overview

In this module, we will begin our exploration of the ggplot2 graphics system.

In my opinion, `ggplot2` has a steep learning curve, because it has a very unusual approach to generating graphics images.

We can't cover everything in the package, so my goal in this lecture is just to get you started, and hopefully to clarify some aspects of this approach that can by mystifying for beginners.

In this module, I want to provide an overview of the main framework of how ggplot2 works.

Section 1: The Pen-And-Paper Model

So far in CSCI E-5a, we've been using the base R graphics system to create all our various images.

The base R graphics system uses what's known as a "pen-and-paper" approach to creating images.

In this analogy, when we create a graph, we start with a blank sheet of paper, and then make marks on the paper with a pen.

Once we've drawn something on the graph, it's there, and can't be modified, although we can then superimpose other shapes on top of the pre-existing image.

If we don't like the image, you can of course make adjustments to the code for the graph, and run this again.

But for any particular image that's being created, once you've made a mark on the page, it's there.

Similarly, once you've determined things like the range of the x- and y-axes, by using the `xlim` and `ylim` arguments, then they are fixed, and if you want to adjust those you have to modify the code and re-run it.

This approach is hardly unique to the base R graphics system.

In fact, just about every graphics system that I'm aware of uses this method.

Even graphics design packages such as Adobe Illustrator or Affinity Designer use this model, although they allow for a more sophisticated approach to managing layers.

In this model, the base R graphics functions behave very differently from most other kinds of functions.

With this approach, graphics functions such as `stripchart` operate by drawing on the graphics display, and they don't return any value.

These graphics functions are very unusual "functions" from a theoretical standpoint.

When I originally defined the concept of a "function", I had to be careful to indicate that functions in R could be different from the kinds of functions that we know from mathemetics.

In mathematics, a function takes some collection of objects as input argumens, and then returns another objects (often a number).

Usually when we call a function and then assign the output value to a variable, R does not produce any display:

```
x <- exp( 2 )
```

In this case, the function `exp()` calculates the value of `exp(2)`, returns this value as its output, and then stores this return value in a variable.

Neither the function call nor the assignment operation produce any sort of display itself.

If we now directly display the variable `x` R will evaluate this variable and then produce a display:

```
x
```

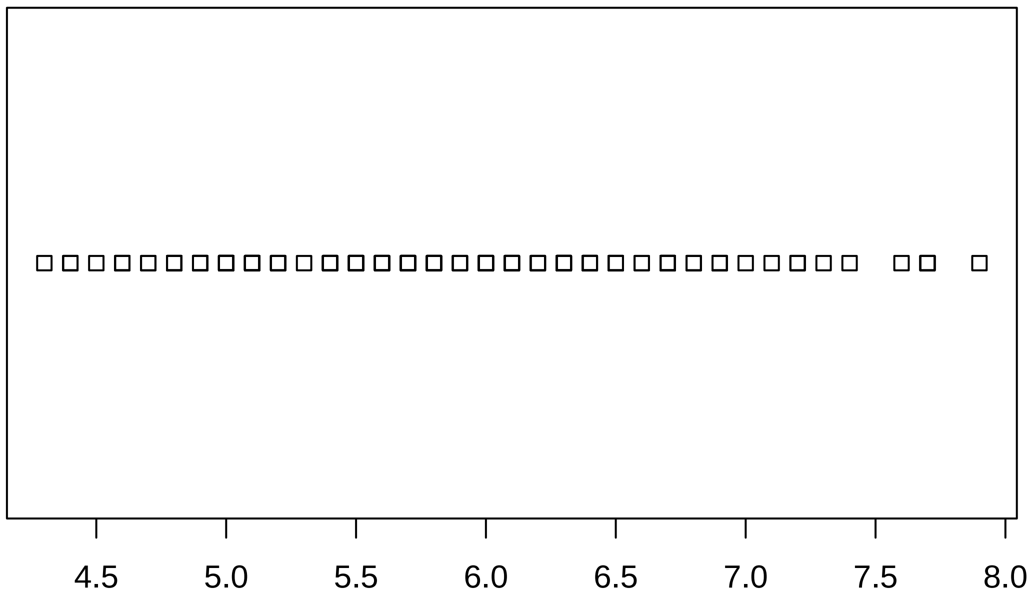
```
## [1] 7.389056
```

This is all quite standard, and we've seen this in action hundreds of times throughout this course.

By contrast, the base R graphics functions don't "return" an object – instead, they simply perform an action.

To see this, let's call the `stripchart()` function and try to store the return value in a variable:

```
stripchart.return.value <-
    stripchart( iris$Sepal.Length )
```



Notice that in this case a display is produced, even though we are performing an assignment.

Now let's directly display the return value:

```
stripchart.return.value
```

```
## NULL
```

This `NULL` value means that the `stripchart()` function didn't return anything.

Instead, it was in the process of actually evaluating the function that the graph was produced.

This is called a "side effect" of the function – it's something that happens other than producing some sort of return value.

Mathematical functions such as `exp()` don't have side effects, and all they do is to calculate a return value.

Base R graphics functions on the other hand *do* produce side effects, and we're generally not interested in the return value generated by the function call.

Section 2: The Object-Oriented Model

As I've pointed out, pretty much every graphics package uses some form of the pen-and-paper model.

This model is so straightforward that it's difficult to imagine any alternative.

In fact, the approach in `ggplot2` is quite different.

In the `ggplot2` framework, we construct special graphical *objects*.

These graphical objects are complex data structures, and as we build them they do not automatically produce displays.

Instead, it's only when the object is directly evaluated that a graphical image is rendered.

This is how many R objects work.

For instance, let's create a vector, and store it in a variable:

```
first.vector <- 1:10
```

Notice that no output was produced here, because the assignment operation does not display anything.

Now we'll create a new vector:

```
second.vector <- 2 * first.vector
```

Again, nothing was displayed.

Let's do one more operation:

```
third.vector <- second.vector + 5
```

You get the idea: nothing is displayed.

Now we will evaluate `third.vector`:

```
third.vector
```

```
##  [1]  7  9 11 13 15 17 19 21 23 25
```

Now we *do* see a display, because we are directly evaluating the `third.vector` object.

This is the basic idea of `ggplot2`:

- We start by creating a `ggplot2` object.
- We then modify this `ggplot2` object and store the result in a variable.
- When we're done with our modifications of the object, we can then generate a graphical display by directly evaluating the object.

Thus, the functions in the `ggplot2` package *do* operate in a manner similar to how standard mathematical functions work.

The functions in `ggplot2` do not produce a graphics image as a side effect, but instead return a `ggplot2` graphics object, which can be further modified.

When we finally want to see the image, we can just directly display the object, and only then does R render a graph.

We'll see lots of examples of this process in the next module, but for right now I want you to simply understand this framework in a very general way.

What's the advantage to this strange way of creating graphics images?

The idea is that we can create a basic graphics object and save this, then apply different modifications to it to obtain different types of visualizations.

This allows to quickly explore many different visualizations of our data.

With the conventional approach, we would have to construct each different visualization from scratch, but with `ggplot2` we can simply modify a pre-existing graphical object.

Section 3: The Grammar of Graphics

The `ggplot2` framework views graphs as constructed from layers.

There is a complicated theory for these layers, and this is called the "Grammar of Graphics".

You don't have to understand the theory of the grammar of graphics in detail in order to make `ggplot2` graphics, but you should have some basic familiarity with the concept.

There are seven layers:

- The first layer is called "Data", and consists of the actual data that is being graphed.
- The second layer is called "Aesthetics", and consists of mappings of data values to visual components of the graph.
- The third layer is called "Geometries", and consists of various geometric objects such as points and lines.
- The fourth layer is called "Statistics", and consists of statistical functions of the data.
- The fifth layer is called "Facets", and allows for multi-panel displays.
- The sixth layer is called "Coordinates", and this controls the range of the display.
- The seventh and final layer is called "Theme", and this consists of visual elements that are not strictly related to the data e.g. the color and size of the title.

There is an enormous amount of material here, and we can't expect to cover it in the course of a 2 to 3 hour lecture.

For this lecture, I'm going to focus on the "Aesthetics" and "Geometries" layers.

Much of the challenge of learning the `ggplot2` framework is in understanding how these two layers work.

If you can understand these two layers, then everything else becomes much clearer.

Section 4: The Aesthetics Layer

For me, the most difficult concept to grasp was the notion of "Aesthetics", but it's also a fundamental part of the framework, so you have to master this if you want to use this system.

Before we go on, let me say that I think this is an incredibly unfortunate choice of terminology.

The word "Aesthetics" in English has strong associations with the concept of beauty.

In philosophy, this word means the theory of art or beauty.

If we say that something is "aesthetically appealing", that means that it looks really nice.

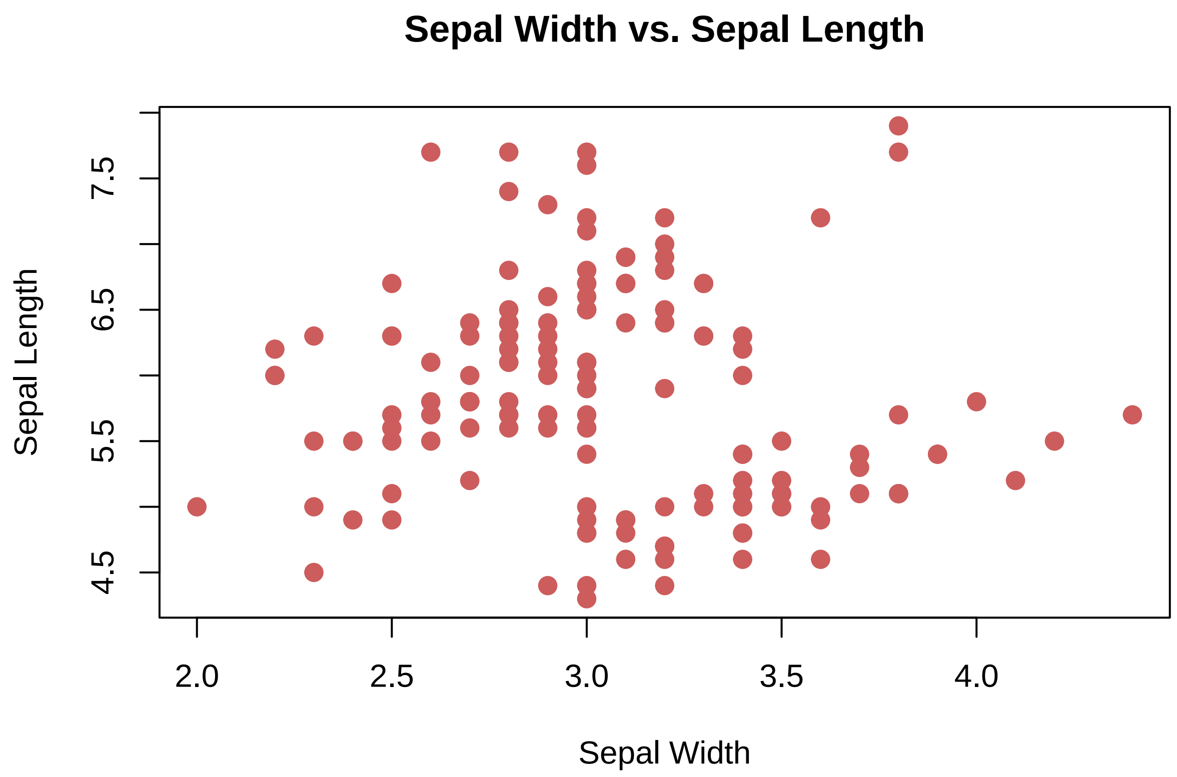
This is **NOT** what the word means in the `ggplot2` framework.

In `ggplot2`, the term "Aesthetics" is in no way supposed to suggest beauty or visual appeal, although the graphs can be quite stunning.

Instead, the concept of an "Aesthetic" is an association or mapping of some aspect of the data to a component of the visual display.

For instance, suppose we create a scatterplot of `Sepal.Width` vs. `Sepal.Length` in the `iris` dataset:

```r
plot(
    x = iris$Sepal.Width,
    y = iris$Sepal.Length,
    main = "Sepal Width vs. Sepal Length",
    xlab = "Sepal Width",
    ylab = "Sepal Length",
    pch = 19,
    cex = 1.2,
    col = "indianred"
)
```



Here, the data for `Sepal.Width` has been mapped or associated with the range of x values i.e. the horizontal axis.

Likewise, the data for `Sepal.Length` has been mapped or associated with the range of y values i.e. the vertical axis.

This is the sense of the term "Aesthetics" – it just means that components of the data are mapped or associated with components of the visual display.

You can see from the scatterplot that we've been doing this sort of mapping throughout the course, although in an informal way.

We can map other aspects of the data to other components of the display, such as the size or color of the points.

This is why I think the choice of the word "Aesthetics" is so unfortunate, because it obscures what's really going on.

Section 5: The `Geom` Layer

The "Aesthetics" layer establishes the mappings of data to visual components.

Once we've done that, we then need to select a particular form of the visual display.

For instance, we could display data as either points or lines.

For a one dimensional range, we could display either a histogram or a boxplot.

These different forms of visual display are called `geoms` in `ggplot2`.

Once we've created a `ggplot2` object with "Aesthetic" mappings of data to visual components, we can then create different graphs by adding `geoms` to the graphics object.

We can quickly generate many alternative visualizations of data by creating one graphics object, saving it in a variable, and then applying different `geoms` to it.

This is a large part of the appeal of `ggplot2`: that it allows for rapid exploration of a dataset.

We'll see specific examples of this in the next module, but for right now I just want you to understand the overall approach.

Section 6: Some Remarks About Learning `ggplot2`

Before we get started with the actual code, let me make a few personal observations.

There's no doubt that `ggplot2` is fashionable, and there seems to be a consensus that if you want to do advanced graphics in R this is really the best approach.

Given this consensus, then I want to be clear: if you are trying to market yourself as an R expert, someone whose professional identity is that they are highly skilled in working with R, then you must be competent with `ggplot2`.

If you are trying to sell yourself as an R programmer and you can't use `ggplot2`, then you just look weird.

So – if you want to be perceived as someone whose primary professinal competency is R programming, you have to know `ggplot2`.

But what about someone who *isn't* trying to market him or herself as an R professional, but instead wants to be able to use R in the course of their other work:

- A biologist who wants to analyze experimental data.
- A financial analyst who needs to work with financial data.
- An administrator who wants to explore data in order to make better decisions.

The list goes on and on, but the basic idea is that many people have an interest in using R in their work but are not primarily R expert programmers.

For these people, the decision to use `ggplot2` is more nuanced.

The problem here is that the `ggplot2` framework has a steep learning curve, and the marginal benefits of the package don't appear until you've developed a high degree of fluency.

I've seen many people announce proudly that they are using `ggplot2` for their graphics, and then produce mediocre images.

Just because you load the package into R doesn't mean that your graphs will automatically look better.

You really have to know the system and be comfortable with the framework to achieve good results.

As I've mentioned, there is a consensus among many people that `ggplot2` is necessary in order to produce high-end graphics.

In my experience, much of this consensus is just uncritical groupthink and recycled conventional wisdom, and many of the people making this claim do not actually know enough about base R graphics or `ggplot2` graphics to give a detailed comparison.

In fact, it's not clear to me that `ggplot2` adds much value to conventional one- and two-dimensional plots such as histograms, stripcharts, scatterplots, and pie charts.

For these sorts of graphs the object-oriented approach introduces a lot of unnecessary overhead and clutter to the code, without any compensating benefit.

In contrast, base R offers a compact syntax that allows us to easily specify many of the details of these graphs using a simple and concise system of notation.

For these sorts of graphs, you'll make better graphs faster and with less effort if you use the base R graphics system.

In fact, it's not clear to me that there are any visualizations that you can do in `ggplot2` that you can't do in base R graphics, although some visualizations might be easier in one system rather than another.

The point here is not to discourage you from using `ggplot2`, but rather to suggest that you should think carefully about whether the additional power and flexibility of the system will justify the cost of ascending that learning curve.

In any case, no matter what graphics system you decide to use, you should commit to mastering it.

In all the graphs that we've made in CSCI 5a, we've improved the images substantially by paying attention to details.

So you're better off developing fine control of your images rather than struggling with an advanced package that you can't handle properly.

One other consideration that you should bear in mind is that much of the added value of `ggplot2` is in its ability to rapidly generate exploratory multidimensional visualizations of complex datasets.

But often in the real world we don't need to do this.

Instead, we *know* what sort of image we want to create.

For instance, if we want to show the annual profits for the four WiDgT locations, then a barplot is good way to do this, and there's not much exploration involved.

Bear in mind that complex, layered, multidimensional plots can be extremely difficult to interpret, especially for an audience that isn't experienced with this sort of graphical image.

In addition, there are always practical constraints to consider.

If you work in a group where everyone is using `ggplot2` to make graphs, then it's in your interest to learn the system.

Likewise, if there is some particular specialty graphics package that you want to use for your work, and it's built on the `ggplot2` framework, then that's a strong argument for learning the package.

Conversely, if you're collaborating with people who don't have any familiarity with the package, then using `ggplot2` might cause problems.

Finally, you might just like the look of the `ggplot2` graphs.

This really is a situation where the term "aesthetics" is appropriate!

If you find the `ggplot2` graphics visually appealing, then maybe that's the system for you.

Let's summarize these thoughts, and then I'll shut up:

- If you want to be an R professional, then you have to learn the package.
- If you're not trying to be an R professional, but instead just use R as part of your work, then you should think more carefully about the costs and benefits of the package.

In any case, I really want to emphasize that having strong skills in creating visualizations is very important, and no matter what system you decide to use you should intentionally commit to achieving mastery.

Section 7: Cheatsheets

A “cheatsheet” is a piece of paper that is extensively covered with notes.

In America, the term originally referred to something that a student would smuggle into an exam in order to cheat.

It’s come to mean a one- or two-page summary of a subject.

There are many cheatsheets available at the RStudio website.

You should download the “Data Visualization” cheatsheet, which features an extremely compressed summary of the `ggplot2` package.

Really, you should block out some time and download all of the cheatsheets.

The Data Visualization Cheatsheet is an extremely valuable resource, although it is so densely packed with information that it can actually be difficult to navigate.

One of the goals of this lecture is to get you started with the cheatsheet.