Code

STAT 209 Lab 2: Introduction to Visualization With `ggplot2`

RMarkdown

In this lab we will use a type of document called an **RMarkdown** file.

This format provides several advantages over using an R script for code and a word processor for our writeup:

1. Everything is in one place, with text, code, and plots interleaved
2. Changes to data or code automatically get reflected in the document (no worry about copy-pasting and getting out of sync)
3. Your work is stored in plain text: More portable (don't need MS Office, e.g.)
    1. Works with version control systems

Creating a Markdown document

Let's create a new Markdown document together. But first! Make a new project, and put it in a sensible place within your home directory.

1. Once you've done that, go to File > New File > RMarkdown.
2. Fill in a Title (e.g., `STAT 209: Lab 2` and an Author (your name).
3. Before doing anything else, save the file, with the extension `.Rmd`.

We'll get into some nuances of the RMarkdown format later; for now the main things to know about are:

1. Create headings and subheadings with #, ##, ###, etc. One # for a main heading, two for a subheading, three for a sub-subheading, etc.
2. Bold text with two *s on either side
3. Italicized text with _ on either side
4. Create blocks of R code with Insert > R (or use a keyboard shortcut). Chunks begin and end with a line containing three backticks (on the tilde key on most keyboards).
5. Lines between the ``` lines are interpreted as code
6. Can run individual chunks, or all chunks prior to a chunk, using the buttons in the top right of the chunk region
7. “Compile” the document to HTML with “Knit > Knit to HTML”
8. The opening line of a code chunk has curly braces with some chunk options separated by commas.
    1. Use `message = FALSE` in the braces to suppress info from R as when loading packages
    2. Use `warning = FALSE` if you are getting a warning that you’re convinced isn’t a problem, and you don’t want it to be displayed (but be sure first!)
    3. Use `echo = FALSE` if you don’t want the code to show up in the output
    4. Use `eval = FALSE` if you want the code to be displayed but not run
    5. Use `results = 'hide'` if you want the code to be run but don’t want the results to be displayed
    6. Use `cache = TRUE` for time-consuming chunks so they don’t re-run every time

Something to note with Markdown documents: When Knitting the entire document (but not when running individual chunks), the process does *not* have access to objects in your environment: *only* objects defined within the document are available.

This is useful, since if you left something undefined you’ll usually get an error. But you will not always want to Knit the entire document, since this can take time; be careful when running one chunk at a time, since this *can* use objects in your environment.

Graphics with `ggplot2`

Introduction

Goal: By the end of this lab, you will be able to use `ggplot2` to build some basic data graphics

Setting up

First create a Markdown document (or use the one you have open already).

Remember Before we can use a library like `ggplot2`, we have to load it. In this case, we load the `tidyverse` package, which automatically loads `ggplot2` for us (since it depends on it).

Exercise 1 Make a new code chunk containing the command to load the `tidyverse` package. Adjust the chunk options to suppress unwanted output. Knit the document.

Note: Remember, you shouldn’t copy and paste code directly from the web. Type it out yourself so that you slow yourself down a bit to process what you’re reading, and to develop your muscle memory.

(Click the “Code” button on the right to show the code)

Hide

```
library(tidyverse)
```

Why `ggplot2`?

Advantages of `ggplot2`:

1. Consistent underlying **grammar of graphics** (Wilkinson, 2005)
2. Is a mature and complete graphics system
3. Plot specification is at a high level of abstraction
4. Flexible
5. Has a `theme` system to polish plot appearance (more on this later)
6. Used by many, many people

What is *The Grammar Of Graphics?*

The big idea: independenly specify plot building blocks and combine them to create just about any kind of graphical display you want. Building blocks of a graph include:

1. data (`data=`)
2. aesthetic mappings (`aes()`)
3. geometric objects (`geom_*()`)
4. statistical transformations
5. scales
6. coordinate systems
7. position adjustments
8. faceting

Using `ggplot2`, we can specify different parts of the plot, and combine them together using the `+` operator. [Note that the `+` operator is similar to the `%>%` pipe operator, but is **not** interchangeable]

Example: Housing prices

Let’s start by looking at some data on housing prices (click the “Code” button to see the code; include a chunk with this code in your document):

Hide

```
housing <- read_csv("http://colindawson.net/data/landdata-states.csv")
glimpse(housing)
```

```
## Observations: 7,803
## Variables: 11
## $ State            <chr> "AK", "AK", "AK", "AK", "AK", "AK", "AK", "AK",…
## $ region           <chr> "West", "West", "West", "West", "West", "West",…
## $ Date             <dbl> 2010.25, 2010.50, 2009.75, 2010.00, 2008.00, 20…
## $ Home.Value       <dbl> 224952, 225511, 225820, 224994, 234590, 233714,…
## $ Structure.Cost   <dbl> 160599, 160252, 163791, 161787, 155400, 157458,…
## $ Land.Value       <dbl> 64352, 65259, 62029, 63207, 79190, 76256, 72906…
## $ Land.Share..Pct. <dbl> 28.6, 28.9, 27.5, 28.1, 33.8, 32.6, 31.3, 29.9,…
## $ Home.Price.Index <dbl> 1.481, 1.484, 1.486, 1.481, 1.544, 1.538, 1.534…
## $ Land.Price.Index <dbl> 1.552, 1.576, 1.494, 1.524, 1.885, 1.817, 1.740…
## $ Year             <dbl> 2010, 2010, 2009, 2009, 2007, 2008, 2008, 2008,…
## $ Qrtr             <dbl> 1, 2, 3, 4, 4, 1, 2, 3, 4, 1, 2, 2, 3, 4, 1, 2,…
```

(Data originally from https://www.lincolninst.edu/subcenters/land-values/land-prices-by-state.asp (https://www.lincolninst.edu/subcenters/land-values/land-prices-by-state.asp), via Jordan Crouser at Smith College)

Geometric Objects and Aesthetics

Geometric Objects (`geom`)

Geometric objects or `geoms` are the actual marks we put on a plot. Examples include:

1. points (`geom_point()` , for scatter plots, dot plots, etc.)
2. lines (`geom_line()` , for time series, trend lines, etc.)
3. boxplot (`geom_boxplot()` , for, um…)

among others

A plot must have at least one `geom` , but you can combine multiple `geom` s in a single plot. Remember that you can add elements to an existing plot using the `+` operator (elements can be chained together in a single command, or intermediate plots can be assigned to a variable and added to later).

You can see a list of the `geom_*()` functions in `ggplot2` using the following command:

Hide

```
help.search("geom_", package = "ggplot2")
```

In RStudio, if you simply type `geom_` and then press the tab key, you will see a dropdown list of possible ways to complete the text. This is a useful trick generally, to save repetitive typing. Once you have completed a function name and typed the open paren `(` , tab will also show you a list of valid argument names for that function.

Aesthetic Mappings (`aes`)

In `ggplot2` , *aesthetic* means “something you can see”. Each aesthetic is a mapping between a visual cue and a variable. For example, we can map variables to the following cues:

1. position (i.e., on the x and y axes)

2. color (the “outside” color of a geometric object)
3. fill (the “inside” color of a geometric object)
4. shape (of points)
5. line type
6. size

Each type of `geom` accepts only a subset of all aesthetics — refer to the help pages of individual `geom_()` functions to see what mappings each `geom` accepts. Aesthetic mappings are set with the `aes()` function.

Points

Now that we know about geometric objects and aesthetic mapping, we’re ready to make out first `ggplot` : a scatterplot. We’ll use `geom_point` to do this, which requires `aes` mappings for `x` and `y` . Other mappings (such as color) are optional.

Example

Hide

```
## Get a subset of the data (more on filter() later)
hp2013Q1 <- housing %>% filter(Date == 2013.25)

ggplot(hp2013Q1, aes(y = Structure.Cost, x = Land.Value)) +
  geom_point()
```

Exercise 2 Create a scatterplot of the value of each home in the first quarter of 2013 as a function of the value of the land.

Possible Solution

Hide

```
ggplot(hp2013Q1, aes(y = Home.Value, x = Land.Value)) +
  geom_point()
```

Plot objects

The output of the `ggplot()` function is an object. Since we want to modify the plot that we created above, it’s helpful to store the plot object in a named variable.

Hide

```
base_plot <- ggplot(hp2013Q1, aes(y = Structure.Cost, x = Land.Value))
```

To actually show the plot, we just print it, as we would print the value of a numeric value or a data frame.

Hide

```
base_plot
```

Notice that although the axes are set up and labeled, there's no data being depicted. That's because we haven't specified any `geom`s – in other words, we haven't told R what we actually want it to draw. However, the aesthetic mapping is defined, and if we take this base plot and add `geom`s to it, the resulting plots will use the mapping that we defined in `base_plot`.

Let's add some points!

Hide

```
base_plot + geom_point()
```

Exercise 3 We have a lovely scatterplot now, but we haven't stored it. Store the scatterplot you created in the previous exercise as an object called `home_value_plot`.

Possible Solution

Hide

```
home_value_plot <- ggplot(hp2013Q1, aes(y = Home.Value, x = Land.Value)) +
  geom_point()
```

Lines

A plot constructed with `ggplot` can have more than one `geom`. For example, we could connect all of the points using `geom_line()`. By default, the aesthetic mapping defined in the base plot is carried over to any new `geom`s that we add. Note that now we see both points **and** lines.

Hide

```
base_plot + geom_point() + geom_line()
```

Exercise 4 Does it make sense to connect the observations with `geom_line()` in this case? Do the lines help us understand the connections between the observations? What do the lines represent?

Smoothers

Not all geometric objects are simple shapes – `geom_smooth()` includes both a line and a ribbon, where the line is a "smoothed" moving average of the y variable, and the band is a 95% confidence band, represent our uncertainty about what the moving average actually would be if we had infinite data.

Hide

```
base_plot +
  geom_point() +
  geom_smooth()
```

Other smoothing methods and band definitions are available too. You can find out more about the various options by looking at the documentation page for `geom_smooth()`.

Text

Each `geom` accepts a particular set of aesthetics (i.e., mappings) – for example, `geom_text()` accepts a `labels` mapping. This mapping wasn't defined in the base plot, so we can add it here.

Note that in the following plot we are *not* using `geom_point()` or `geom_line()` – we have *only* `geom_text()` since we only want the state labels to be drawn, not points or lines.

Hide

```
base_plot +
  geom_text(aes(label = State), size = 3)
```

Aesthetic Mapping vs. Assignment

Note that the variables are mapped to aesthetics with the `aes()` function, while fixed visual cues are set outside the `aes()` call. This sometimes leads to confusion, as in this example:

Hide

```
base_plot +
  geom_point(aes(size = 2), # not what you want, because 2 is not a variable
             color = 'red') # this just turns all points red)
```

The `aes()` function can also be used outside of a call to a `geom`. Here we update the `base_plot` with an additional mapping, assigning the color cue to the home value variable.

Hide

```
base_plot <- base_plot +
  aes(color = Home.Value)
```

Exercise 5 In your `home_value_plot`, map color to the cost of the structure, and show your scatterplot.

Sample solution

Hide

```
home_value_plot +
  aes(color = Structure.Cost) +
  geom_point()
```

Mapping Variables to Other Aesthetics

Other aesthetics are mapped in the same way as `x` and `y` in the previous example.

Hide

```
base_plot +
  geom_point(aes(shape = region))
```

Scales: Controlling Aesthetic Mapping

Aesthetic mapping (i.e., with `aes()`) only says that a variable should be mapped to an aesthetic. It doesn't say *how* that should happen. For example, when mapping a variable (say, `z`) to *shape* with `aes(shape = z)`, you don't say *what* shapes should be used. Similarly, `aes(color = z)` doesn't say *what* colors should be used. Describing what colors/shapes/sizes, etc. to use is done by modifying the corresponding **scale**. In `ggplot2`, `scales` include:

1. `position`
2. `color`, `fill`, and `alpha` (these control the "outer" and "inner" colors and the opacity (from transparent at 0 to opaque at 1), respectively, of the geometric objects)
3. `size`
4. `shape`
5. `linetype`

Scales are modified with a series of functions using a `scale_<aesthetic>_<type>` naming template. Try typing `scale_` followed by the tab key to see a list of scale modification functions.

Common Scale Arguments

The following arguments are common to most `scales` in `ggplot2`:

1. `name`: the first argument specifies the axis or legend title
2. `limits`: the minimum and maximum of the `scale`
3. `breaks`: the points along the scale where labels should appear
4. `labels`: the text that appears at each break

Specific scale functions may have additional arguments; for example, the `scale_color_continuous()` function has arguments `low` and `high` for setting the colors at the low and high end of the scale.

Scale Modification Examples

Start by constructing a dot plot to show the distribution of home values by `Date` and `State`.

Hide

```
home_plot <- ggplot(housing, aes(y = State, x = Home.Price.Index)) +
  geom_point(aes(color = Date),
             alpha = 0.3,
             size = 1.5,
             position = position_jitter(width = 0, height = 0.25))
```

First, let's change the label on the vertical axis.

Hide

```
home_plot <- home_plot +
  scale_y_discrete(name = "State Abbreviation")
```

Now let's modify the `breaks` and `labels` for the `x` axis and color scales:

Hide

```
home_plot +
  scale_color_continuous(breaks = c(1975.25, 1994.25, 2013.25),
                         labels = c(1971, 1994, 2013))
```

Now let's change the low and high values to `blue` and `red` for a plot that's a bit more dramatic:

Hide

```
home_plot <- home_plot +
  scale_color_continuous(
    breaks = c(1975.25, 1994.25, 2013.25),
    labels = c(1971, 1994, 2013),
    low = "blue", high = "red")
home_plot
```

Using different color scales

`ggplot2` has a wide variety of color `scales`; here is an example using `scale_color_gradient2()` to interpolate between three different colors:

Hide

```
home_plot +
  scale_color_gradient2(
    breaks = c(1975.25, 1994.25, 2013.25),
    labels = c(1971, 1994, 2013),
    low = "blue", high = "red", mid = "gray60",
    midpoint = 1994.25)
```

Exercise 6 Since a home price index of 1 is an important benchmark, it is worth highlighting as a contextual reference in our plot. Use `geom_vline()` to add a dotted, black, vertical line to the plot we created above.

Possible solution

Hide

```
home_plot +
  geom_vline(
    aes(xintercept = 1),
    linetype = 3,
    color = "black") +
  scale_color_gradient2(
    breaks = c(1975.25, 1994.25, 2013.25),
    labels = c(1971, 1994, 2013),
    low = "blue", high = "red", mid = "gray60",
    midpoint = 1994.25)
```

Exercise 7 Recall that layers in `ggplot2` are added sequentially. How would you put the dotted vertical line you created in the previous exercise *behind* the data values?

Available Scales

Note: In RStudio, you can type `scale_` followed by TAB to get the whole list of available scales.

Faceting

The idea behind faceting is to create separate graphs for subsets of the data, and tile those graphs in a manner that makes it easy to visually compare them.

`ggplot2` offers two functions for creating facets:

1. `facet_wrap()` : define subsets as the levels of a single grouping variable, and tile the resulting plots in one dimension, “wrapping around” as needed.
2. `facet_grid()` : define subsets as the crossing of two grouping variables

By splitting data into separate subplots it is possible to keep the amount of clutter in a single plot under control, while keeping all the information in easy visual proximity to facilitate comparison among plots.

Example: What is the trend in housing prices in each state?

Let’s start by using a technique we already know: map `State` to `color` :

Hide

```
state_plot <- ggplot(housing, aes(x = Date, y = Home.Value))

state_plot +
  geom_line(aes(color = State))
```

This plot is horrendous. There are two problems: the distinctions among colors are too fine-grained to be able to see them, and the lines obscure each other.

Faceting to the rescue?

We can fix the previous plot by faceting by `State` rather than mapping `State` to `color`:

Hide

```
state_plot +
  geom_line() +
  facet_wrap(~State, ncol = 10)
```

Notice the tilde (~) syntax before the variable name. This is a convention borrowed from the syntax R uses to define regression models, and which the `lattice` graphics package (as well as the `mosaic` package) use to define plots.

The `facet_grid()` function can be used to create facets that vary according to two grouping variables. Its syntax is

Hide

```
facet_grid(y ~ x)
```

where `y` and `x` are the names of grouping variables that define the rows (vertical) and columns (horizontal) of the faceted grid, respectively.

Exercise 8 Use a `facet_wrap` and/or `facet_grid` to create a data graphic of your choice that illustrates something interesting about home prices.

Getting credit

1. Save your .Rmd file to the path `~/stat113/turnin/lab2` on the RStudioPro server.
2. Post the command you used in Exercise 8 as a DM on Slack *to both me and the student grader, Christian Ikeokwu*. You can assume that I've read in the data as `housing`, but do not rely on any other stored variables. Tag your response with the hashtag `#lab2`, and also type the Honor Pledge, which indicates that you have completed the rest of the lab up to that point, that your code is your own and that you have (mostly) understood what you have written.
3. Export your plot as a file from the Plots tab in RStudio, download it to your computer, and then post the image file to the `#lab2` channel on Slack