Data Visualization

http://datascience.tntlab.org

Module 7

Today’s Agenda

- A Brief Reminder to Update your Software

- A walkthrough of *ggplot2*
  - Big picture
  - New cheatsheet, with some familiar caveats

  - Geometric Objects
  - Aesthetics
  - Statistics
  - Coordinates
  - Facets
  - Themes

Before We Begin

- We've now been using R for many weeks
- New versions of R have been released since we started

- Any time you install a new library that throws a version warning, update.

- Go back to the R website and re-download
- After installing R, redownload R Studio
- After installing R Studio, update your packages using **update.packages(ask=F)**
- If you find packages disappear/stop working, trace through error messages and reinstall using **install.packages()**

Big Picture on Visualization with *ggplot2*

- Visualization is arguably the single most important step in data analysis
  - Allows you to check assumptions quickly and intuitively
  - Quickly communicate to other people on your project what you've learned
  - Does the same for stakeholders

- *ggplot2*, like the rest of *tidyverse*, tries to guess what you are trying to do and glosses over some of the details
  - Base-R does no glossing, which is why we're focusing on *ggplot2*
  - *ggplot2* syntax allows you to very quickly try out multiple visualizations quickly, and keeps a record of precisely where those visualizations came from
  - Remember the *data pipeline* in R scripting

What Does ggplot2 Get Us?

- ggplot2 handles a lot of the "under the hood" mathematics that you would normally need to do in R

- Remember that to run base-R's **barplot**(), we had to use the **table**() command first to create a table of frequencies
  - Try to create a basic barplot with both **ggplot**() and **barplot**()

- Default visualization settings with *ggplot2* are much nicer looking than base-R, and it's much easier to modify those settings

- It also gets us to think like data scientists in terms of visualization, an area where social science is typically *horrible*

ggplot2's Grammar of Graphics

- Each ggplot is a base ggplot object, defined with *ggplot*(). That object contains its own aesthetics plus geometric objects (geoms) that also have their own aesthetics, by default *inherited* from higher parent objects.
- If you want a geom to have *varying properties,* this needs to be specified in a parent object. If you want all instances of a geom to be the same, specify it within the geom. More specific always overrides less specific.

- Step 1: Create a ggplot object, specify **data,** and define its fundamental **aesthetics** (e.g., which axis is which)
- Step 2: Add geometric objects you want (**geoms**) and their own **aesthetics**

- Use + to combine features. For multiline coding, end with +
- Call a ggplot object directly to display it.

Getting Data into a ggplot2-friendly Format

- Although just about any structure of your data may be *functional*, you typically want one variable of values for each DV plus grouping factors

- For univariate statistics
  - Isolate your outcome values as a single column
    - If your data are longitudinal, restructure so that all DV measurements are in a single variable
  - Isolate your grouping variables as unique columns
    - If your data are split by condition, convert raw condition codes into their main effect representations (e.g., in a between-subjects 2x2, conditions 1 2 3 4 should be converted into two new factor variables with descriptive values).

- For multivariate statistics
  - Isolate each DV as a column
  - Still isolate your grouping variables as unique columns

- Always convert to correct types (numeric, factor, etc) at before starting to code a **ggplot**()

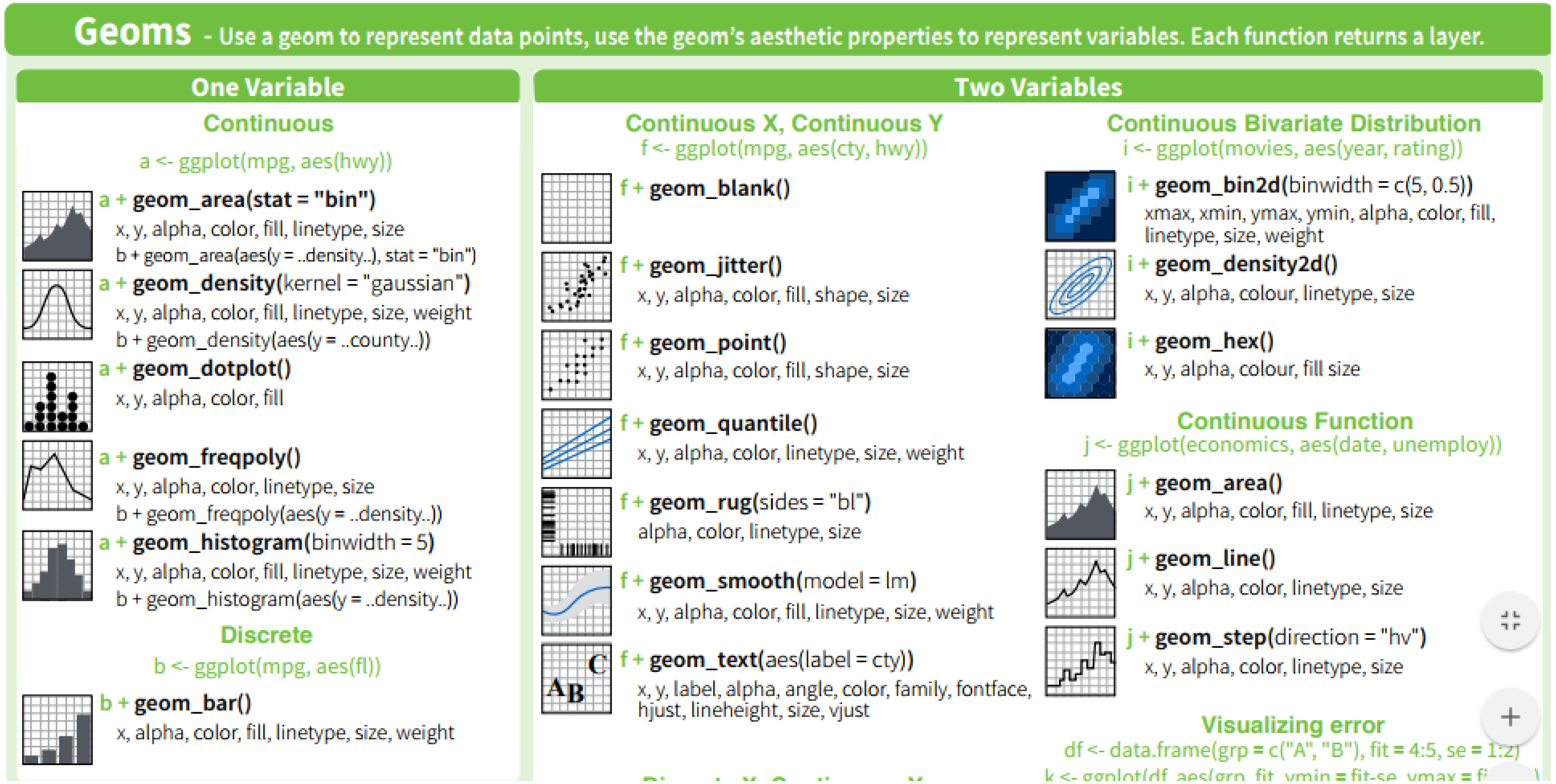
General Aesthetics of a *ggplot*

- Common Aesthetics (but there are more)
  - **x** and **y**, your axes
  - **color**, which is line color
  - **fill**, which is fill color
  - **size**, which is marker size
  - **shape**, which is marker type
  - **weight**, which is line thickness
  - **alpha**, which refers to "alpha channel", i.e., transparency
  - **linetype**, which is line pattern
  - **labels**, which refers to any text

- Different geoms inherit different aesthetics, so decide on your geoms first
  - Notice that this is the opposite order taken when writing code
  - Can be helpful to sketch it on paper first
- Within a geom, aesthetics may be attributes instead (no aes() function)

New Cheatsheet!

- https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf



Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + **geom_area(stat = "bin")**
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + **geom_density**(kernel = "gaussian")
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

a + **geom_dotplot()**
x, y, alpha, color, fill

a + **geom_freqpoly()**
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

a + **geom_histogram**(binwidth = 5)
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

b + **geom_bar()**
x, alpha, color, fill, linetype, size, weight

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

f + **geom_blank()**

f + **geom_jitter()**
x, y, alpha, color, fill, shape, size

f + **geom_point()**
x, y, alpha, color, fill, shape, size

f + **geom_quantile()**
x, y, alpha, color, linetype, size, weight

f + **geom_rug**(sides = "bl")
alpha, color, linetype, size

f + **geom_smooth**(model = lm)
x, y, alpha, color, fill, linetype, size, weight

f + **geom_text**(aes(label = cty))
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + **geom_bin2d**(binwidth = c(5, 0.5))
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + **geom_density2d()**
x, y, alpha, colour, linetype, size

i + **geom_hex()**
x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

j + **geom_area()**
x, y, alpha, color, fill, linetype, size

j + **geom_line()**
x, y, alpha, color, linetype, size

j + **geom_step**(direction = "hv")
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fi

Defining Specific Colors

- Web colors (the basis for many color systems) are traditionally coded in hexadecimal (i.e., base-16)
  - 0 1 2 3 4 5 6 7 8 9 A B C D E F
  - So, 00 = 0, 10 = 16, 20 = 32, FF = 255
  - Windows has a helpful calculator for this.

- Colors are represented as three channels of 8-bit **additive** color marked with # called a **hex triplet**:
  Red, Green, Blue (RGB)
  - #000000 = 0% R, 0% G, 0% B, so black
  - #FF0000 = 100% R, 0% G, 0% B, so red
  - #C0C0C0 = 75% R, 75% G, 75% B, so 25% grey
  - #FF00FF = 100% R, 0% G, 100% B, so fuchsia

- http://htmlcolorcodes.com/ can be helpful for finding specific colors
- But remember colors can also be referred to by name, as strings.

Scale Aesthetics

- **scale** functions change the appearance of the x or y scale, and are followed by **continuous, discrete, date, datetime, log10, time**, or **sqrt**
  - scale_x_
  - scale_y_
  - scale_color_
  - scale_fill_
  - scale_shape_
  - scale_linetype_

- Further scale modifiers, which all take vectors; but you usually want to work in the coordinates layer instead if you need these, since they literally slice apart the figure rather than just the part of the figure you see
  - **limits**
  - **breaks**
  - **expand**

Other Useful Aesthetics

- **jitter** is useful for exploratory plots but should generally not be used in published papers unless you are working with "big data" and indicate what you're doing very clearly

- **position** is useful to reposition pieces of data (as a group) within a figure, e.g., to switch from a simple bar to a stacked bar
  - **stack**, which is position by stacking
  - **dodge**, which is position by offset
  - **fill**, which rescales values as proportions

Reusing Aesthetics

- Don't do this
  - ggplot(my_df, aes(x=var1, y=var2)) + geom_point(col="#ABCDEF", size=4, shape=3)
  - ggplot(my_df, aes(x=var1, y=var3)) + geom_point(col="#ABCDEF", size=4, shape=3)
  - ggplot(my_df, aes(x=var1, y=var4)) + geom_point(col="#ABCDEF", size=4, shape=3)
  - ggplot(my_df, aes(x=var1, y=var5)) + geom_point(col="#ABCDEF", size=4, shape=3)
- Maybe do this
  - my_geom_point <- geom_point(col="#ABCDEF", size=4, shape=3)
  - ggplot(my_df, aes(x=var1, y=var2)) + my_geom_point
  - ggplot(my_df, aes(x=var1, y=var3)) + my_geom_point
  - ggplot(my_df, aes(x=var1, y=var4)) + my_geom_point
  - ggplot(my_df, aes(x=var1, y=var5)) + my_geom_point
- Better yet, restructure your data using **gather**() from *dplyr* and use **facet_grid**()

Statistics Layer

- Peeks under the hood a bit of *ggplot2*
- You will not often need to dip into the statistics layer, but you might need it for specific projects
  - Examples: adding regions of significance to a regression scatterplot or adding a specific type of error bar to a graph
- You can generally do things like this in two ways
  - Calculating the equivalent statistics yourself as new variables and plotting those variables
  - Adding a stat_ layer: **stat_summary**() for summarizing y data given x, **stat_function**() for predicting y from x, and **stat_qq**() for Q-Q plots

Deconstructing **stat_summary**() Calls

-
```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
    stat_summary(fun.y = mean, geom = "point") +
    stat_summary(fun.data = mean_sdl, fun.args = list(mult = 1),
          geom = "errorbar", width = 0.1)
```

- What is happening here?
  - **stat_summary**() summarizes observed y given x
  - Thus, these **stat_summary**() summarize Sepal.Length given values of Species
  - In the first, simple means are displayed as points for the mean of Species (y)
  - In the second, we need to provide a range of 3 values for **geom_errorbar**() defined as x, ymin and ymax (see ?geom_errorbar)
    - A function from the Hmisc package, **smean.sdl**() will do this for us, but they won't be named what we need (see difference between **smean.sdl**() and **mean_sdl**() output)
    - **mean_sdl**() essentially just renames the elements of the output list
    - **smean.sdl**() defaults to +/- 2 SDs but can be overridden with the mult argument
    - Because we're inside another function, the arguments must be passed as a list

Coordinates Layer

- Many different coordinate systems but you will rarely need anything other than a Cartesian coordinate system

- **coord_cartesian**(xlim = c(1,2), ylim = c(3,4))
  - Specifies limits for axes
  - Data science default ideology is to display all data with as much detail as possible, common to exploratory approaches
  - Very useful for us, e.g., to specify a 1 – 5 range of y for a Likert-type scale
  - Might be able to accomplish the same thing with **coord_equal**() and some attributes

Facets Layer

- Create multiple figures with the same aesthetics and geoms from a single base figure

- Uses formula notation, which we'll deal with in much more detail in the next few weeks
  - y ~ x1 + x2 == "univariate linear model of y with predictors x1 and x2"
  - y ~ . == "model of y with all other variables in the data frame as predictors"

- In the case of facets, formula notation specifies how the splitting is done
  - rowVar ~ .
  - . ~ columnVar
  - rowVar ~ columnVar   # same order as df/matrix

- So, for example:
  - ggplot(mydf, aes(x = condition, y = score)) + facet_grid(. ~ gender)

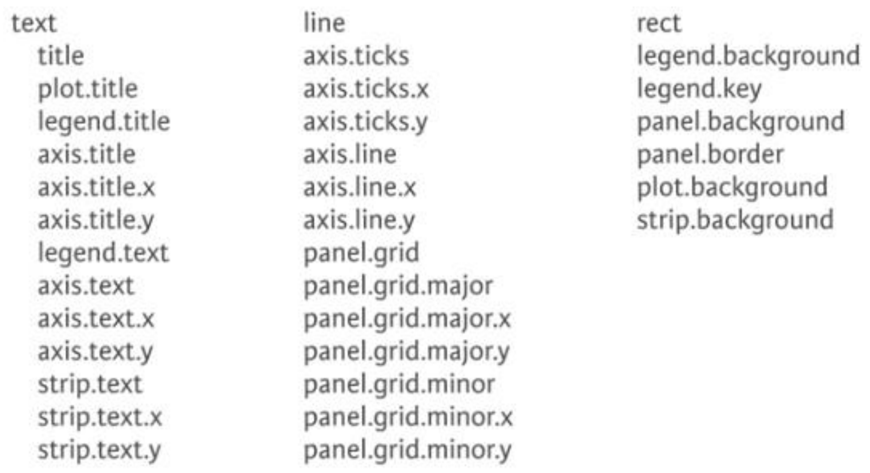
Themes Layer

- Use themes to control text and shape formatting after the figure is finalized; use aesthetics first if possible

- Change formatting of parts of a figure
  - **element_text**(), **element_line**(), **element_rect**(), and **element_blank**() (which is really transparent)

- Look at ?theme for the full list

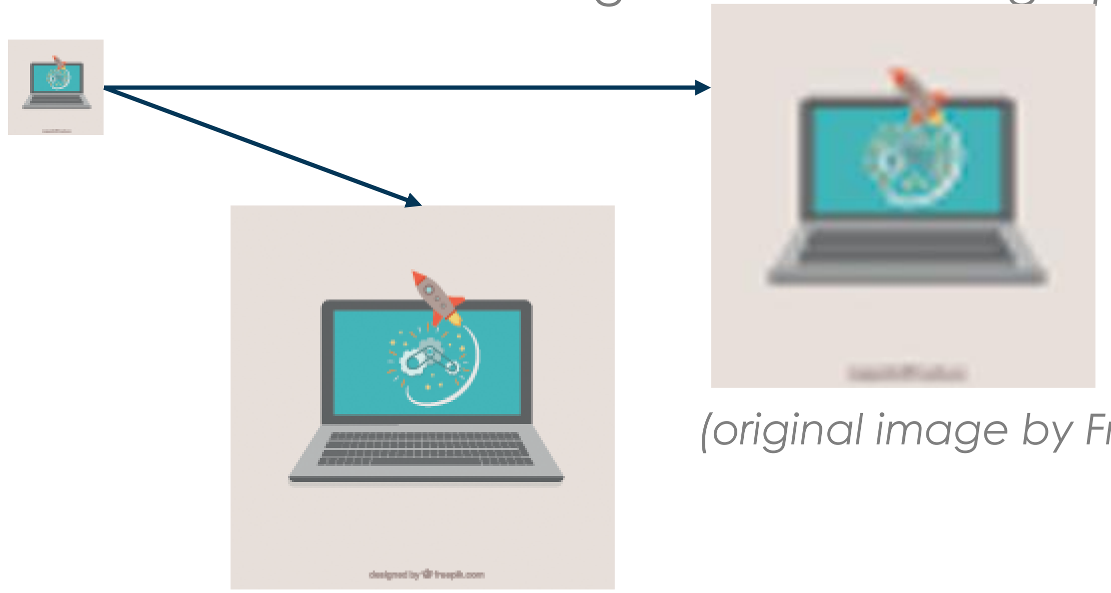
- Best practice is to define a theme separately and then apply it, e.g.,
  - myTheme <- theme(
    legend.background = element_rect(fill="#543215"),
    plot.title = element_text(size = 4))
  - ggplot(mydf, aes(x=condition, y=value)) + theme(myTheme)



```
text               line                   rect
  title            axis.ticks             legend.background
  plot.title       axis.ticks.x           legend.key
  legend.title     axis.ticks.y           panel.background
  axis.title       axis.line              panel.border
  axis.title.x     axis.line.x            plot.background
  axis.title.y     axis.line.y            strip.background
  legend.text      panel.grid
  axis.text        panel.grid.major
  axis.text.x      panel.grid.major.x
  axis.text.y      panel.grid.major.y
  strip.text       panel.grid.minor
  strip.text.x     panel.grid.minor.x
  strip.text.y     panel.grid.minor.y
```

Margins, Spacing and Positioning

- Margins, spacing and positioning are typically done by distance with either "units" or presets (like legend.position = "right")
- What a "unit" appears as visually depends on your coordinates layer; a ggplot unit is defined as "one axis length from origin", so c(.8, .8) == 80% up each axis
- Units can be arbitrarily created or converted via other functions (like **unit**())
- This is done because R figures are *vector graphics* instead of *raster graphics*



(original image by Freepik)

Quick Plots

- Uses *ggplot2* aesthetics but in a single command and with a lot of defaults set for you
  - You just need a "data=" somewhere to specify the dataset
- Not really needed and you probably shouldn't use it
- It was designed to ease the transition between base-R's **plot**() and the **ggplot**() grammar

- HOWEVER
  - **ggpairs**() is a quick and dirty visualization tool which you should definitely use from the *GGally* library
  - You may see references to a function called *plotmatrix* but this has been dropped from *ggplot2*