Session 3 - Visualization

Jongbin Jung

January 9-10, 2016

Dependencies

- Latest version ($\geq$ `3.1.2`) of `R` (*free* from `https://www.r-project.org/`)
- Latest version of `Rstudio` (also *free* from `https://www.rstudio.com/`)
- A bunch of *free* packages

```
# for general plotting
install.packages('ggplot2')
# spacial visualization on maps
install.packages('ggmap')
# for data pre-processing  and formatting
install.packages('dplyr')
install.packages('tidyr')
```

- Basic knowledge of data manipulation (as covered in Session 2)

Visualization: Introduction

- There is more than one framework for thinking about data visualization, e.g.,
  1. Mapping of vectors to 2D/3D surfaces
  2. Function of **inputs** given as varaibles of a data set, **geometries** and **aesthetics** that describe visual markings, and a **coordinate** system that defines the location of each marking
- The first approach is widely used in scientific visualization (e.g., `MATLAB`, classical plotting function in `R`), but doesn't scale well with data
- The second approach, implemented in `R` with the `ggplot2` package, is prefered when working with large scale data, but requires the data frame to be formatted an a specific manner (i.e., in the *long* format)

Quick Comparison: An Example

- We're given the following data as a result of some experiment

Time	Group A Score	Group B Score
1	2	3
2	6	5

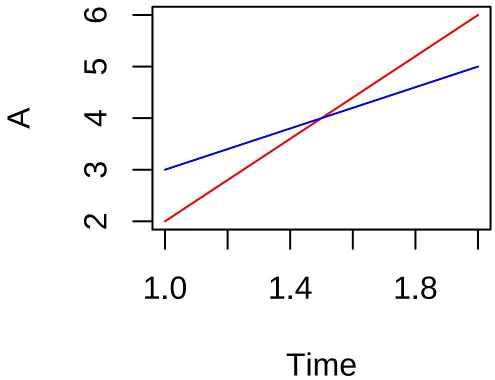
- We wish to plot the scores of each group, i.e., A and B on the vertical axis, with respect to *Time* on the horizontal axis, with different colors for each group
- First, create the data

```
Time <- c(1, 2)
A <- c(2, 6)
B <- c(3, 5)
```

Quick Comparison: The “Classic” Way

- Plot the coordinates of each vector `A` and `B` (no need to understand the code)

```
plot(Time, A, type='l', col='red')
lines(B, col='blue')
```



Quick Comparison: The `ggplot2` Way

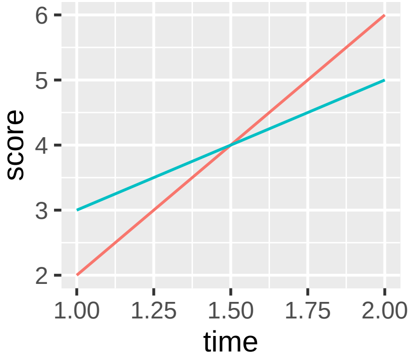
- Create data frame from the vectors, and *tidy* into *long* format (Note that the variables of interest are `time`, `score`, and `group`)

```
df <- data.frame(time=Time, A=A, B=B )
df.tidy <- gather(df, key=group, value=score, A:B)
```

- What does `df.tidy` look like?
- Then, use `ggplot2` to *visualize* the data frame (this is what we'll cover in this session, so you're not supposed to understand the following code)

(the ggplot2 code and plot)

```
p <- ggplot(df.tidy, aes(x=time, y=score))
p <- p + geom_line(aes(color=group))
p
```



Some Common Visualization Tasks

- Most visualization tasks of a data scientist will fall into some combination of the following
  - Explore the distribution of some data with histograms/density plots
  - Plot points on a grid, lines in a plane with meaningful shape/linetype/size/colors
  - Transform coordinates (e.g., log-transform)
  - Make axis labels, tick-marks, etc. concise and meaningful
  - Plot geographic locations on a map
- The goal of this session is to become familiar with the basic concepts and building blocks, such that
  1. you can complete most of the required tasks by yourself
  2. when you need help, you know what to Google (and how to make sense of whatever it is you find)

ggplot2 Basics

Install and Load ggplot2

- Install and load the ggplot2 package like you would any other R package

```
# Install, if you haven't already.
# Only need to do this once on a single machine.
install.packages('ggplot2')
# load package into workspace
library('ggplot2')
```

Datasets

- For this session, we'll mainly use the `quakes` and `economics` datasets that are included with your `R` installation
- The `quakes` dataset contains the location (long/lat), depth (Km), Richter Magnitude, and ID of reporting station for 1,000 seismic events near Fiji since 1964
- The `economics` dataset contains monthly US economic time series data with variables date, personal savings rate (`psavert`), personal consumption expenditures (`pce`), number of unemployed (`unemploy`), median duration of unemployment (`uempmed`), and total population (`pop`)
- Take a look at each data set with

```
quakes
economics
```

The `ggplot` Object

- The basic concept of `ggplot2` is that you define a `ggplot` object, to which you can *add* various elements (e.g., data, visual markings, labels) as layers
- First, you start by defining an empty `ggplot` object with the initializing function `ggplot(data)`

```
p <- ggplot(data=quakes)
```

- Note that
  - The `ggplot` object is assigned to a variable (in this case `p`). The object exists in the workspace, and the *plot* is only generated when you *call* the object itself (i.e., if you type `p` in this case).
  - An initial `ggplot` object is blank, equivalent to a brand new canvas.

aesthetic Mappings

- A key concept that follows the `ggplot` object is aesthetic (`aes`) mappings
- `aes` mappings tell the `ggplot` object where to find the inputs for certain elements of the plot (e.g., x-axis coordinates, colors)
- For example, from the `quakes` data set, if we want to have the `depth` on the x-axis and `mag` on the y-axis, we could initialize our `ggplot` object as

```
p <- ggplot(quakes, aes(x=depth, y=mag) )
```

- Note that
  - `aes()` itself is a function that returns a mapping object, which is used as an argument in the `ggplot()` intialization
  - arguments within the `aes()` call can be column (variable) names
  - the `ggplot` object `p` is still blank: we haven't specified how we want x and y to be visualized

Adding geometries (and other elements)

- The building blocks of visual elements in `ggplot2` are `geometries`
- `geometries` define markings (e.g., points, lines) to be made on the *canvas*
- Elements such as `geometries` are (literally) **added** to existing `ggplot` objects
- For example

```
p <- ggplot(quakes, aes(x=depth, y=mag))
p <- p + geom_point()  # add 'point' geometry to p
```

- We'll explore different `geometries` and visual markings that can be **addedd** to `ggplot` objects in the following sections

Saving Plots

- You can save any plot from RStudio with Export > Save As ... or something like that
- That method of saving plots doesn't scale well, for obvious reasons
- Use `ggsave()` to save plots to files

```
ggsave('my_plot.png', width=5, height=5, plot=p)
```

- `ggsave()` is smart enough to determine the filetype from the extension of the filename that you specify (`png` in the above example)
- While many formats are supported, `png` and `pdf` are most commonly used
- Read the docs to harness the full power of `ggsave()`

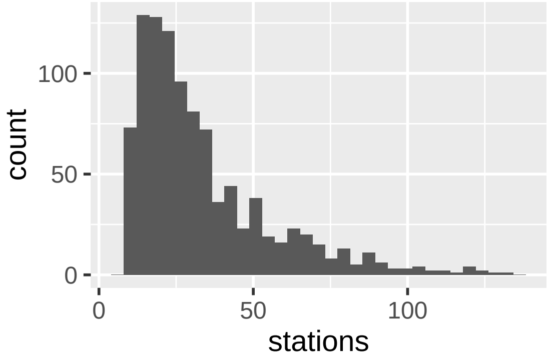
```
?ggsave
```

Single-variable Plots (usually distributions)

Histograms

- Plot a simple histogram by specifying the x-axis variable, and adding the histogram geometry with `geom_histogram()`

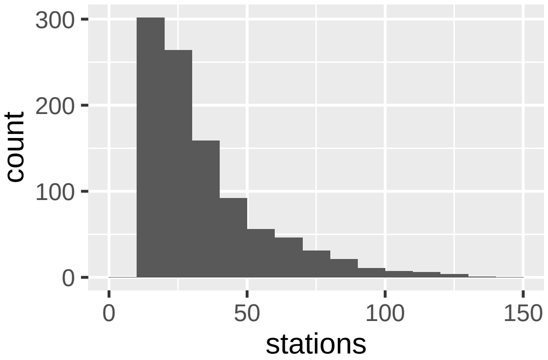
```
p <- ggplot(quakes, aes(x=stations))
p <- p + geom_histogram()
p
```



Histograms (cont'd)

- Specify the size of each bin in the histogram with the `binwidth` argument in `geom_histogram()`

```
p <- ggplot(quakes, aes(x=stations))
p <- p + geom_histogram(binwidth=10)
p
```



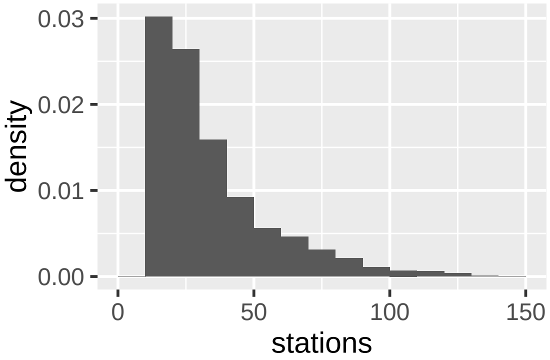
Histograms (cont'd)

- Notice that the default y-axis is `count`, i.e., the observation count of each bin
- This can be changed by specifying the `aes()` mapping of `y`
- For example, to generate a density histogram such that the points of each bin integrates to 1, set `aes(y=..density..)`
- For more options, see

```
?geom_histogram
```

Histogram with aes(y=..density..)

```
p <- ggplot(quakes, aes(x=stations))
p <- p + geom_histogram(binwidth=10,
                        aes(y=..density..))
p
```



Exercise

1. Plot a density histogram of 1,000 random samples from a standard normal distribution using binwidth 0.5 (hint: use `rnorm()`)
2. Plot the (smooth) density of the population (`pop`) variable from the `economics` data

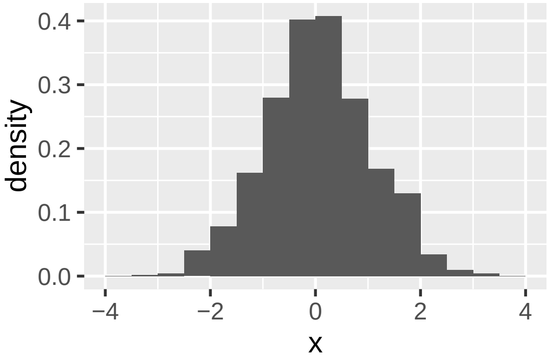
Exercise Solution

WARNING

- Solutions to the exercise are presented in the next slide
- Try the exercise before proceeding!

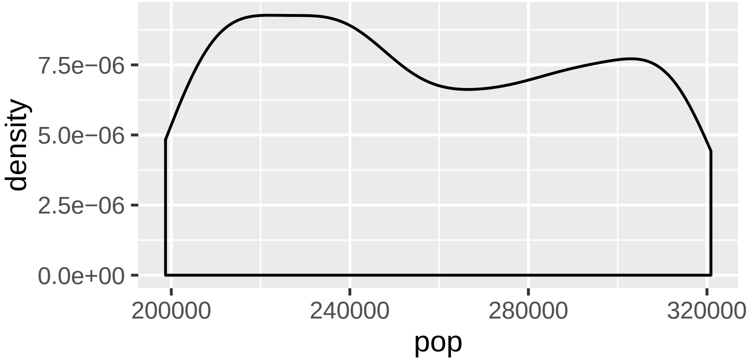
Solution 1

```
X <- data.frame(x=rnorm(1000))
p <- ggplot(data=X, aes(x=x))
p <- p + geom_histogram(binwidth=0.5,
                        aes(y=..density..))
p
```



Solution 2

```
p <- ggplot(data=economics, aes(x=pop))
p <- p + geom_density()
p
```

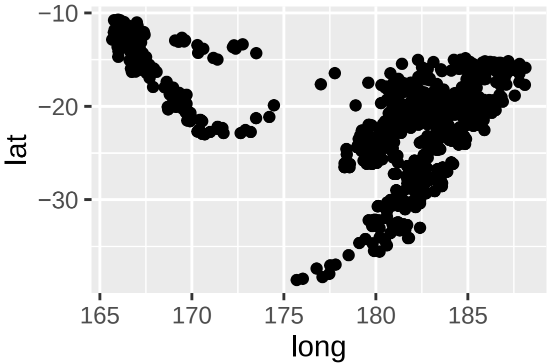


Two-variable Plots (points and lines)

Points with geom_point()

- Plot points on a 2D plane by specifying variables corresponding to the x and y-axis, and adding the point geometry with geom_point()

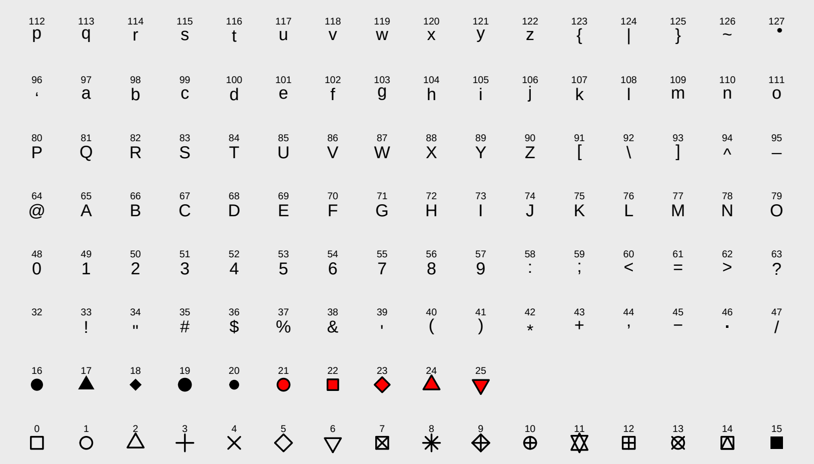
```
p <- ggplot(quakes, aes(x=long, y=lat))
p <- p + geom_point()
p
```



aesthetics for geom_point()

- Popular aesthetics for geom_point() are
  - alpha: point visibility; 0 = invisible, 1 = opaque
  - color: color of the points (try colors() to see a list of some pre-defined colors)
  - shape: shape of the points (predefined, see next slide for reference)
  - size: size of the points
  - fill: color used to fill-in the points (only applies to certain shapes, i.e., shape numbers 21 to 25)

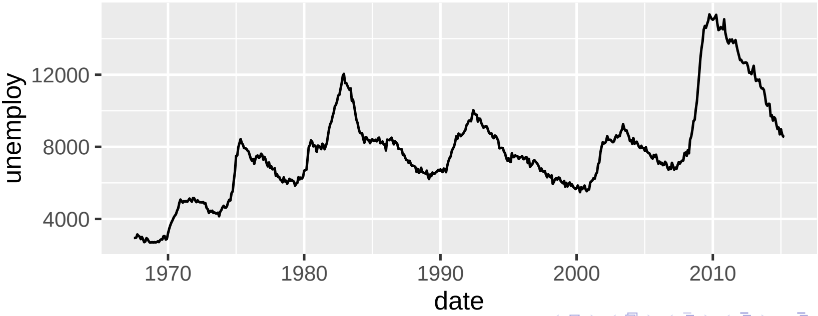
Reference: Shapes



Lines with geom_line()

- Similarly, plot lines on a 2D plane by specifying variables corresponding to the x and y-axis, and adding the line geometry with geom_line()

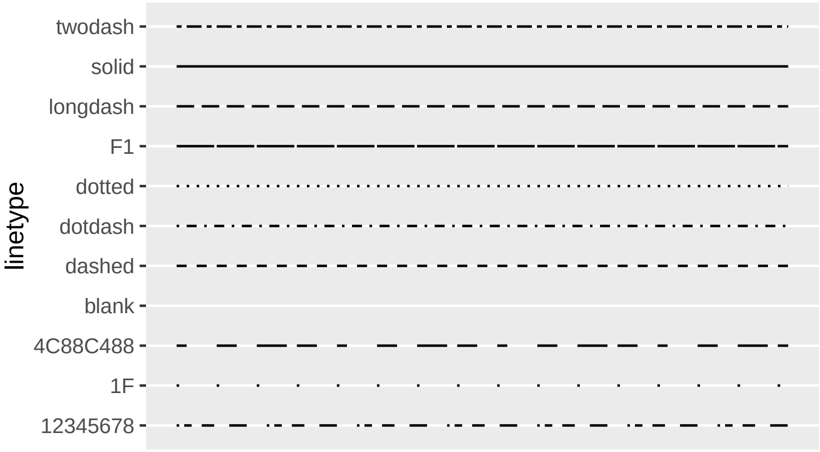
```
p <- ggplot(economics, aes(x=date, y=unemploy))
p <- p + geom_line()
p
```



aesthetics for geom_line()

- Popular aesthetics for geom_point() are
  - alpha: line visibility; 0 = invisible, 1 = opaque
  - color: color of the lines
  - linetype: shape of lines (predefined, see next slide for reference)
  - size: size (thickness) of the lines

Reference: Linetypes



A Note on `data` and `aes()` Arguments

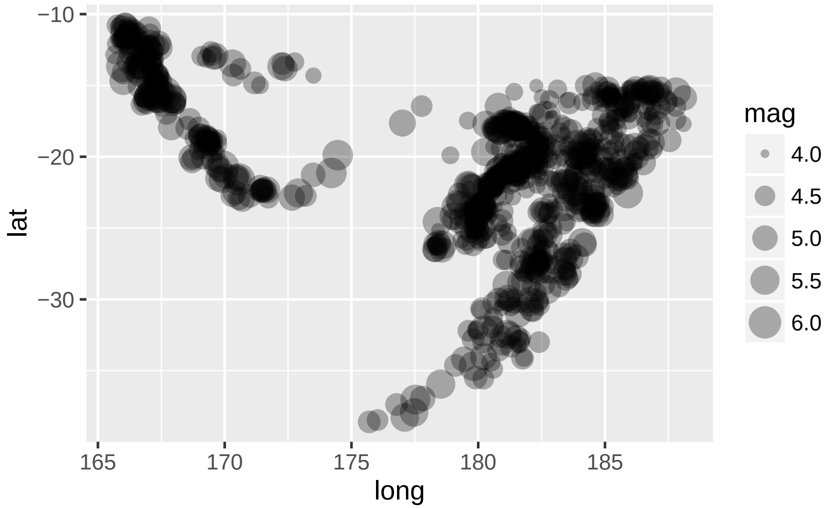
- The `data` and `aes()` arguments, can be declared globally in the `ggplot()` function, or locally in each geometry function
- Also, aesthetics can be either
  - mapped to a variable *globally*, i.e., in `ggplot(aes())`
  - mapped to a variable *locally*, i.e., in `geom_*(aes())`, or
  - defined explicitly for a local `geom_*()`, outside of `aes()`

Example: Global aes() mapping

```
p <- ggplot(quakes, aes(x=long, y=lat, size=mag))
p <- p + geom_point(alpha=.3)
p
```

- the `data` and `aesthetic mappings` for `x`, `y`, and `size` are defined globally in `ggplot()`
- this means any `geom_*` added to this `ggplot` will have the specified `x`, `y`, and `size` aesthetic *mappings*, unless assigned otherwise within their own `geom_*()` function
- the `alpha` aesthetic for `geom_point`, on the other hand, is defined **explicitly** (i.e., it is **set** to 0.3, and not mapped to a variable)

Example: Global aes() mapping (figure)

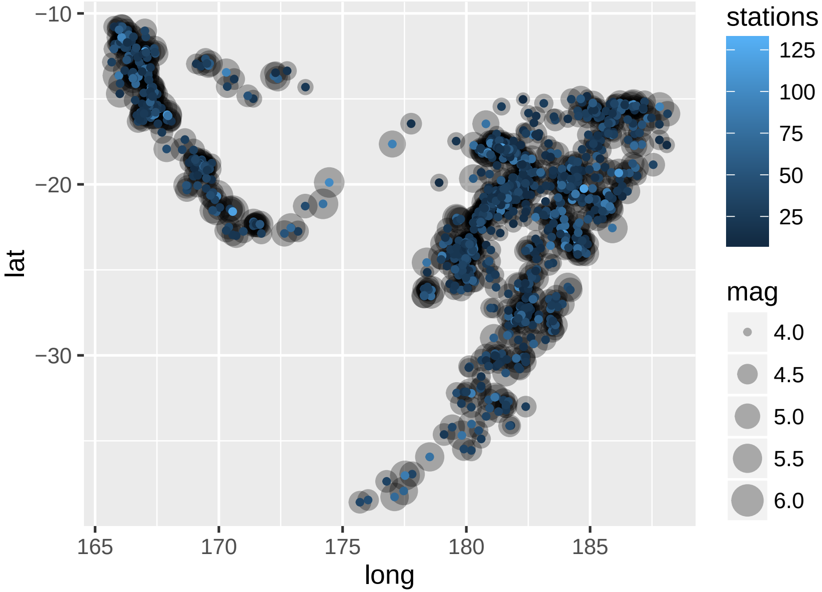


Example: Local aes() mapping

```
p <- ggplot(quakes, aes(x=long, y=lat))
p <- p + geom_point(alpha=.3, aes(size=mag))
p <- p + geom_point(size=1, aes(color=stations))
p
```

- Here, the data and aesthetic mappings for x and y are defined globally in ggplot()
- But the aesthetic mapping/value for size is defined locally for each specific geom_point()
- The first geom_point() maps size to the mag variable, which means the size of the points will depend on the corresponding value of mag
- The second geom_point() explicitly assigns size to the fixed value 1, but maps the color aesthetic to the stations variable
- What do you think the plot will look like?

Example: Local aes() mapping (figure)



Exercise

1. Using the `economics` dataset, plot lines for the values of `unemploy` and `pop` with different linetypes, against `date` as the horizontal axis. (hint: you'll need to select the variables you need, and tidy the data into long format)
2. With the `quakes` dataset, generate a scatter plot of the mean `depth` for seismic events reported by each of the 102 `stations`, with the `stations` on the horizontal (x) axis. Let the colors of each point represent each station, the size represent the ratio `min(mag)/max(mag)` within the seismic events reported from each station, and set `alpha=.6`. (hint: group and summarize the data with `dplyr` first)

Exercise Solution

WARNING

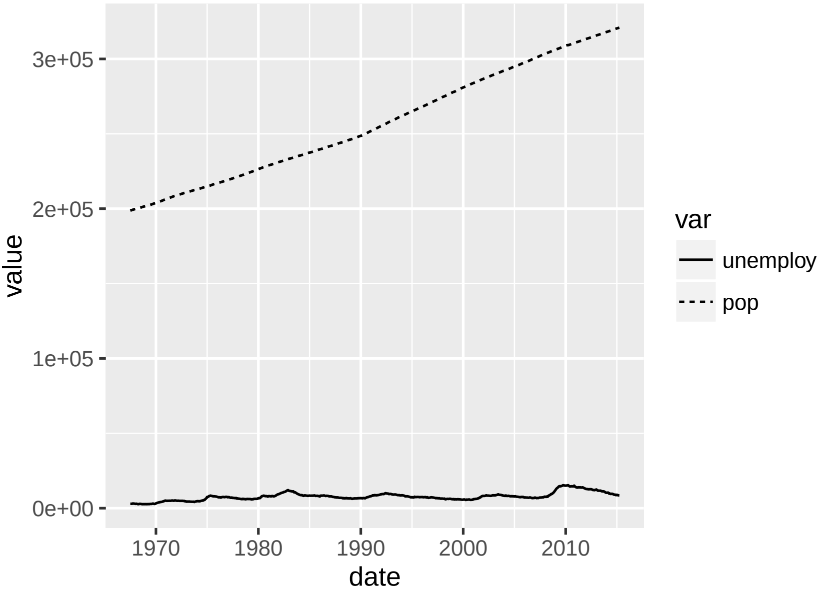
- Solutions to the exercise are presented in the next slide
- Try the exercise before proceeding!

Solution 1

```
# first, get the data into the right format
econ.tidy <- economics %>%
    select(date, unemploy, pop) %>%
    gather(var, value, unemploy:pop)

# generate the plot
p <- ggplot(econ.tidy, aes(x=date, y=value))
p <- p + geom_line(aes(linetype=var))
p
```

Solution 1 (figure)

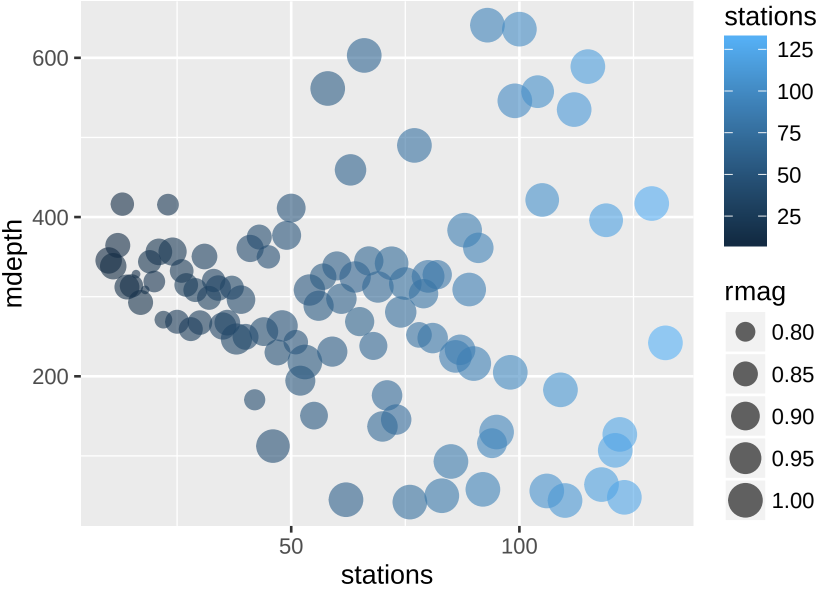


Solution 2

```
# summarize the data
quakes_by_stations <- quakes %>%
    group_by(stations) %>%
    summarize(mdepth=mean(depth),
              rmag=min(mag)/max(mag))

# generate plot
p <- ggplot(quakes_by_stations,
            aes(x=stations, y=mdepth))
p <- p + geom_point(alpha=.6,
                    aes(size=rmag, color=stations))
p
```

Solution 2 (figure)



Scales, Labels, and More

Maps

The map layer with ggmap

- With data that involve spacial coordinates (i.e., long/lat), you might want to use the corresponding geographic map as a 'canvas'
- This is easily achieved in `ggplot` with the ggmap package
- Install and load the ggmap package like you would any other `R` package

```
# Install, if you haven't already.
# Only need to do this once on a single machine.
install.packages('ggmap')
# load package into workspace
library('ggmap')
```

Initializing a map layer

- Use the qmap function to initialize a ggplot object with a map layer
- The main arguments for qmap are
  - location: an address, vector of longitude/latitude pair (in that order), or vector of left/bottom/right/top bounding box
  - zoom: the zoom level in integer values from 3 (continent) to 21 (building)
  - color: either 'color' or 'bw' (for grayscale)
  - maptype: character string providing map theme, e.g., 'terrain', 'satellite', 'roadmap', 'hybrid', etc.
  - legend: position for the legend, e.g., 'top', 'right', 'topleft', 'bottomright', 'none', etc.
- qmap loads maps from Google maps by default, but can be changed to other sources
- see ?qmap, ?ggmap, and ?get_map for more details

A Map Layer Example: Location Search

```r
# Map of stanford by search
p <- qmap(location='Stanford, CA', zoom=15,
          maptype='satellite')
p
```

A Map Layer Example

```
# Bounding box around North America
# (I Googled the numbers)
p <- qmap(location=c(left=-52.233,
          bottom=5.4996, right=-167.2764,
          top=83.1621), zoom=3)
p
```

Exercise

- Plot the quakes data as a scatter plot on a map, using a bounding box with an appropriate zoom level. Let the size of points represent the mag variable, and color represent depth. Set alpha to 0.6. (hint: use the max/min of long/lat variables to define the bounding box)

Exercise Solution

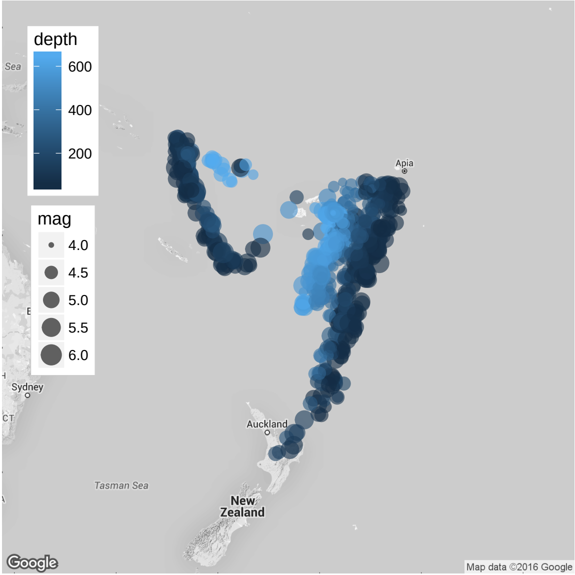
WARNING

- Solutions to the exercise are presented in the next slide
- Try the exercise before proceeding!

Solution

```
maxmag <- filter(quakes, mag==max(quakes$mag))
p <- qmap(location=c(left=min(quakes$long),
                     bottom=min(quakes$lat),
                     right=max(quakes$long),
                     top=max(quakes$lat)),
          zoom=4, color='bw', legend='topleft')
p <- p + geom_point(data=quakes, alpha=0.6,
                    aes(x=long, y=lat, size=mag,
                        color=depth))
p
```

Solution (figure)



Reference

- A great "cheat sheet" for data visualization with `ggplot2` is available for free at `https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf`