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

```
install.packages('ggplot2')
```

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

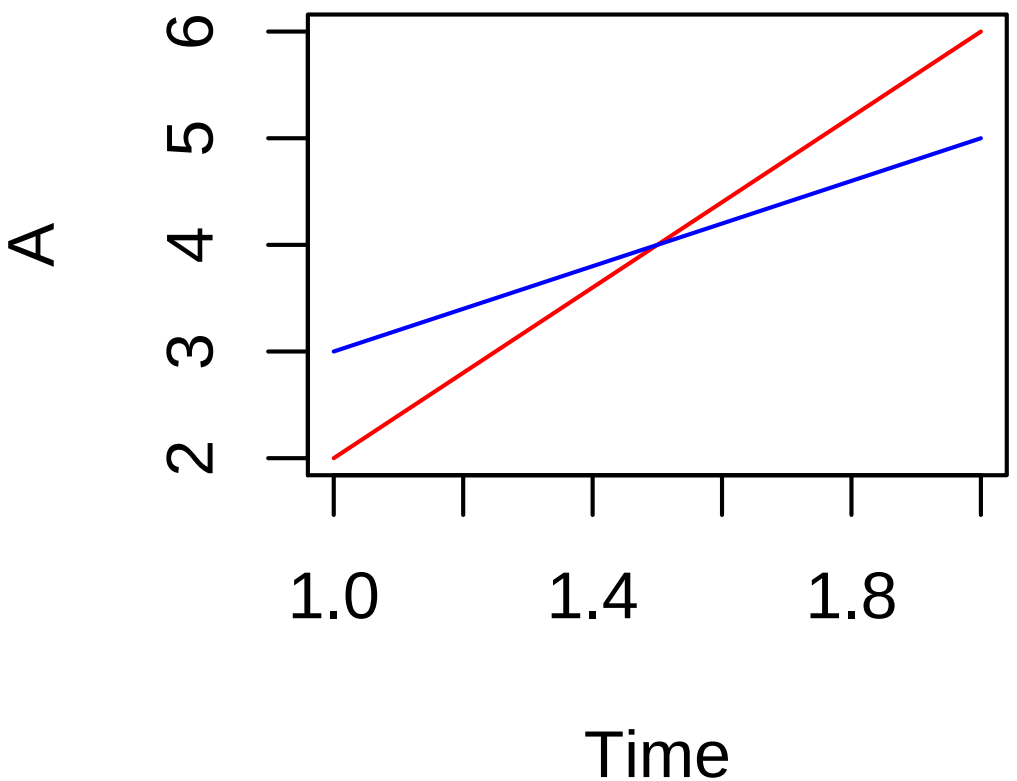
- We wish to plot the scores of each group, i.e., A and B on the vertical axis, with respect to *Time* on the horizontal axis, with different colors for each group
- First, create the data

```
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

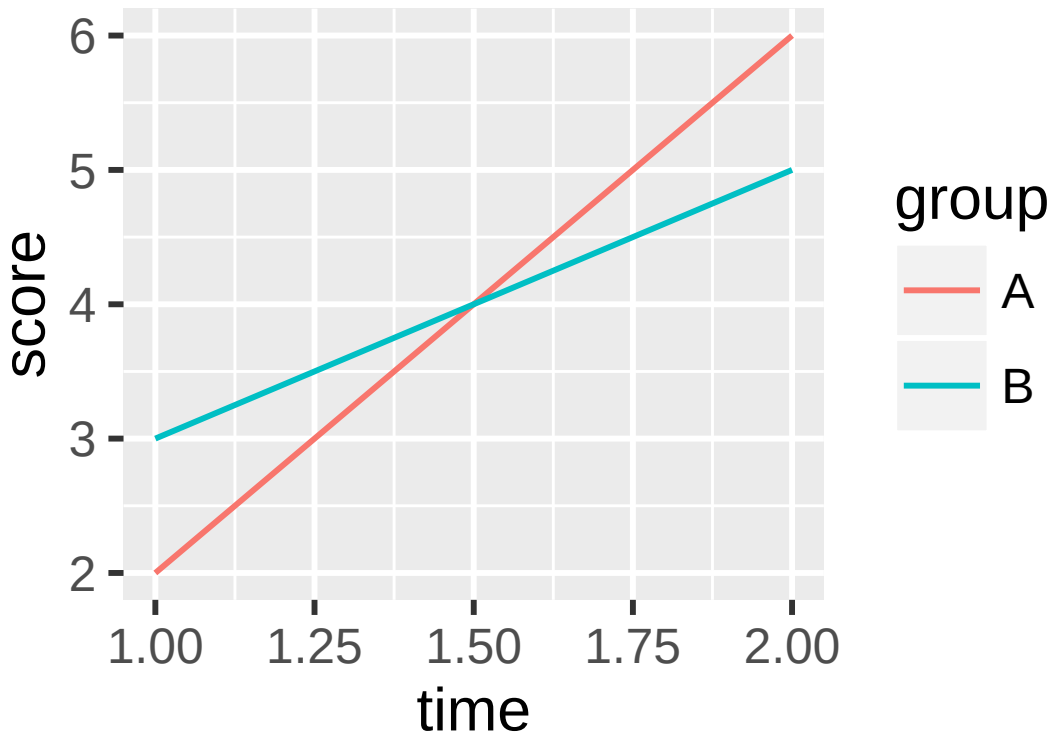
- Create data frame from the vectors, and *tidy* into *long* format (Note that the variables of interest are `time`, `score`, and `group`)

```
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 folloing 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
  - Visualize comparisons with a bar plot
  - 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')
```

- For this session, we'll mainly use the quakes data set that's included with your R installation
- The data set contains the location (long/lat), depth (Km), Richter Magnitude, and ID of reporting station for 1,000 seismic events near Fiji since 1964
- Take a look at it with

```
quakes
```

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 black, equivalent to a brand new canvas.

aesthetic Mappings

- A key concept that folows 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 visual 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 elements that can be **addedd** to `ggplot` objects in the following sections

Saving Plots

- You can save any plot from `R` with right-click > 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()`

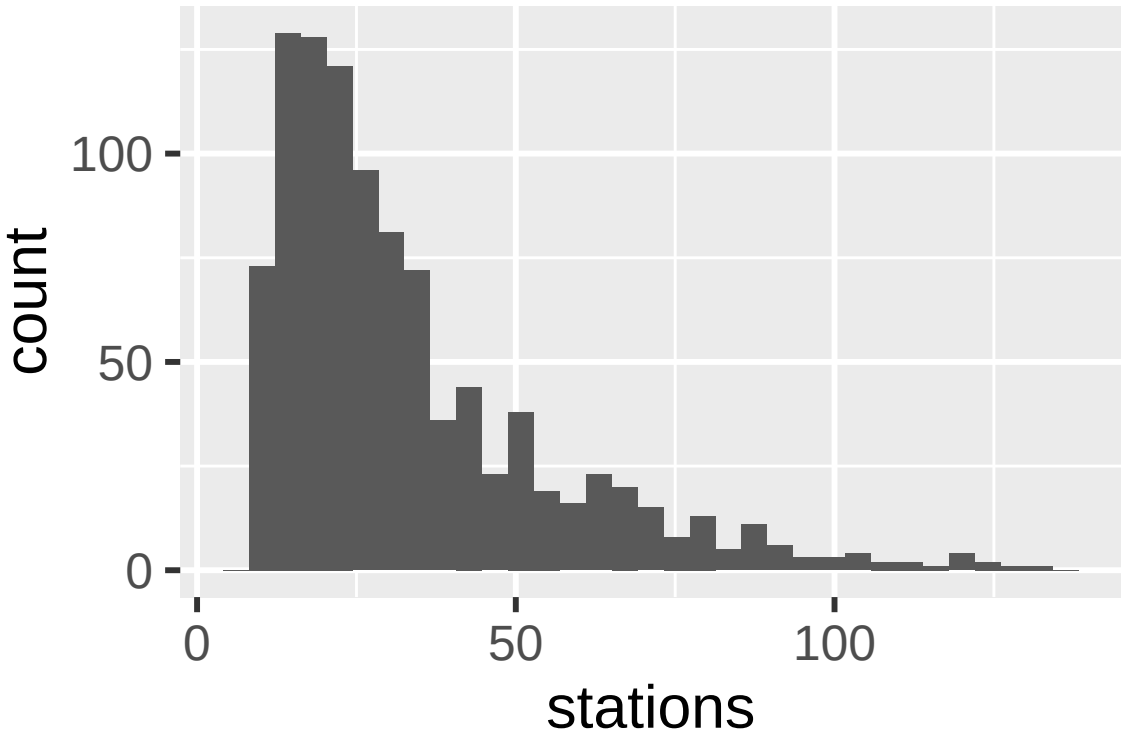
```
?ggsave
```

Single-variable Distributions

Histograms

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

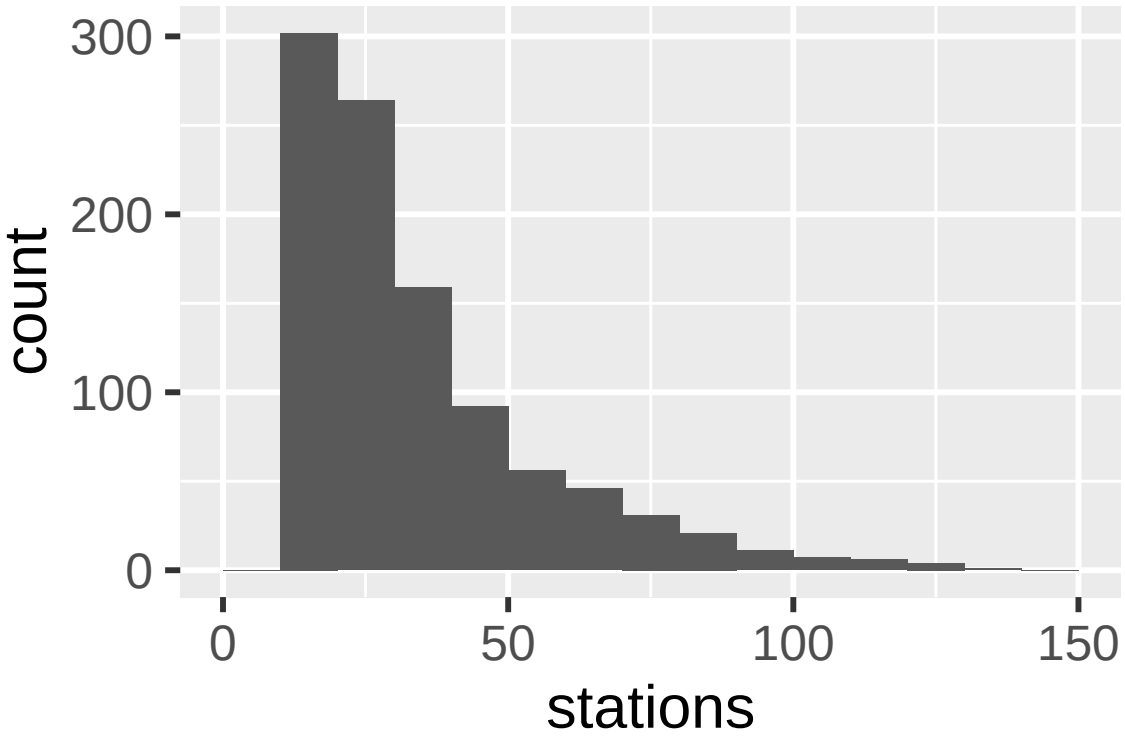
```
p <- ggplot(quakes, aes(x=stations))
p <- p + geom_histogram()
p
```



Histograms (cont'd)

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

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



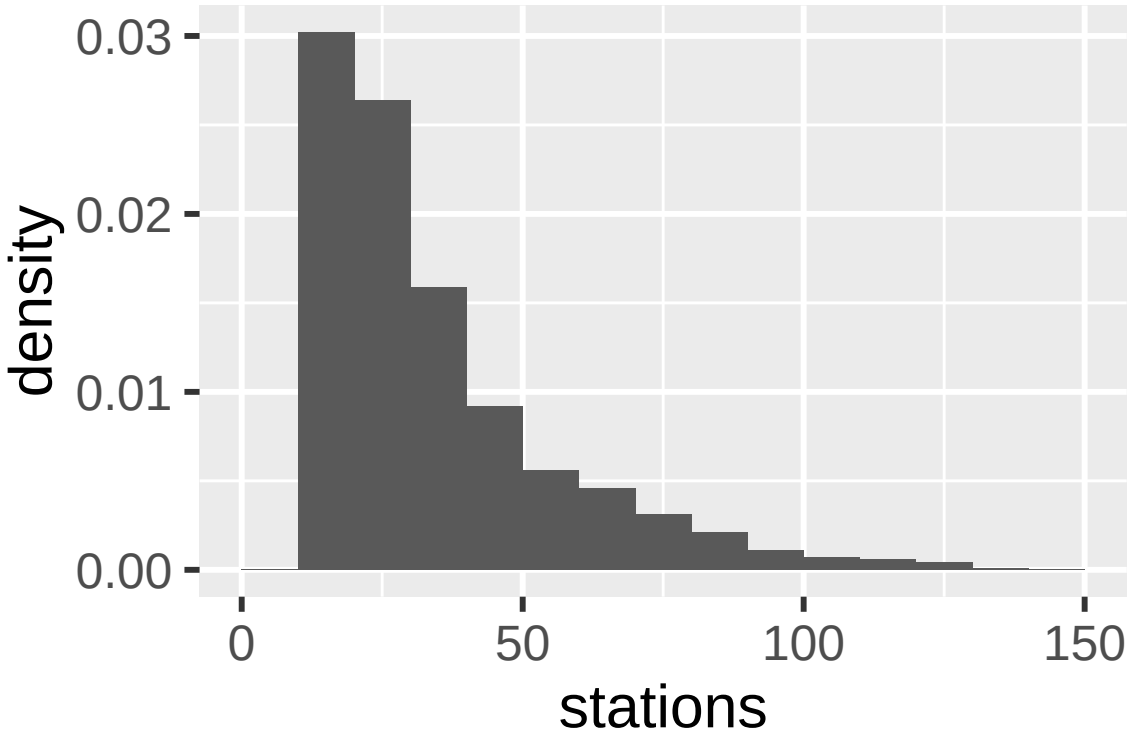
Histograms (cont'd)

- 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
```



Exercies

1. Plot a density histogram of 1,000 random samples from a Uniform (1, 5) distribution using binwidth 0.5 (hint: use `runif()`)
2. Plot the (smooth) density of the `mag` variable from the `quakes` data

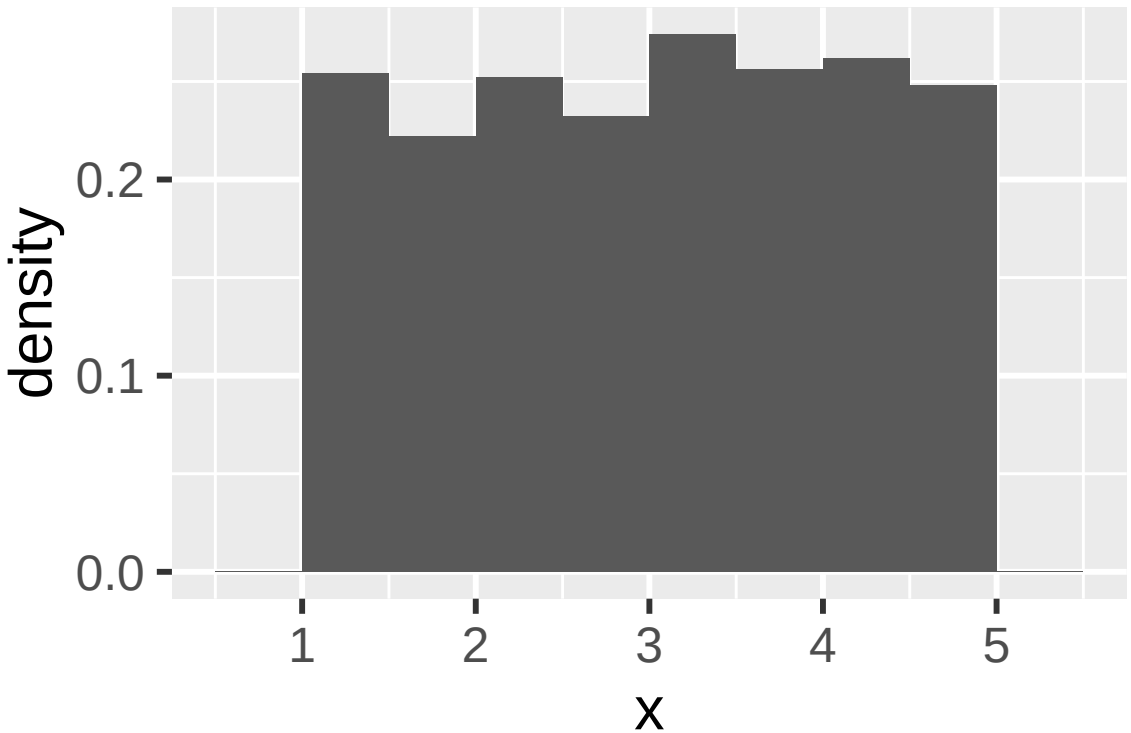
Exercise Solution

WARNING

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

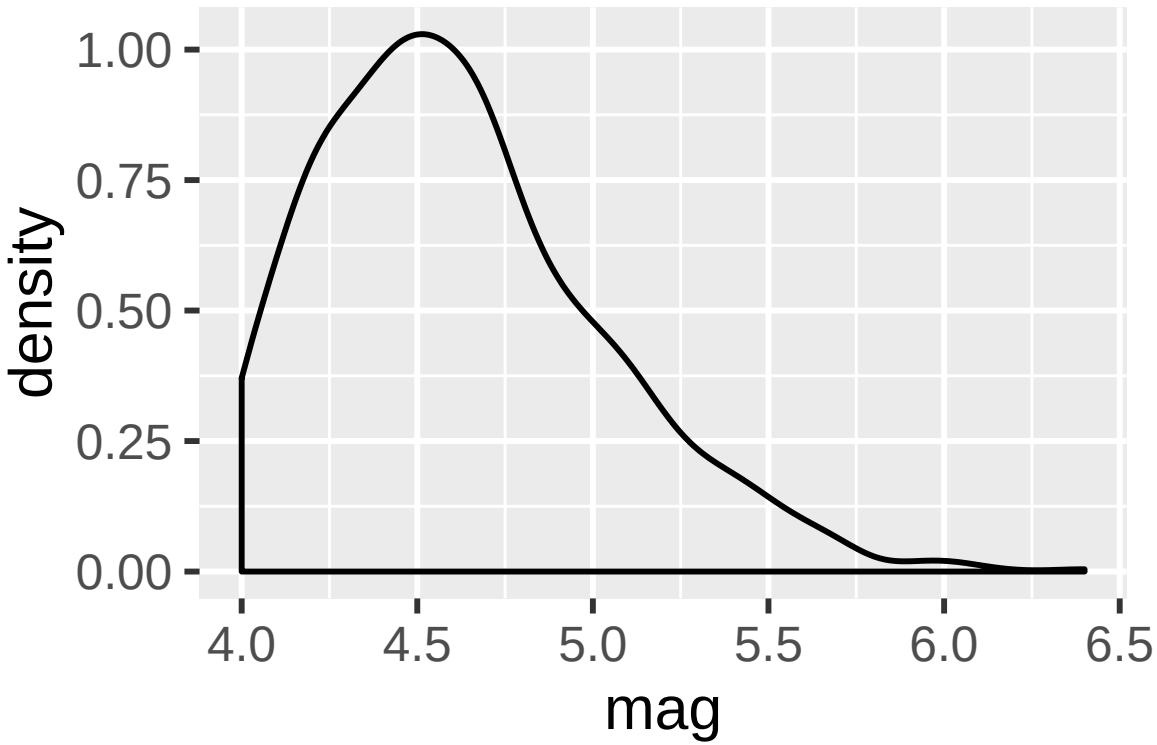
Solution 1

```
X <- data.frame(x=runif(1000, 1, 5))
p <- ggplot(data=X, aes(x=x))
p <- p + geom_histogram(binwidth=0.5, aes(y=..density..))
p
```



Solution 2

```
p <- ggplot(data=quakes, aes(x=mag))
p <- p + geom_density()
p
```



Two-variable Plots (points, lines, and bars)

Scales, Coordinates, Labels, and More

Maps

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`