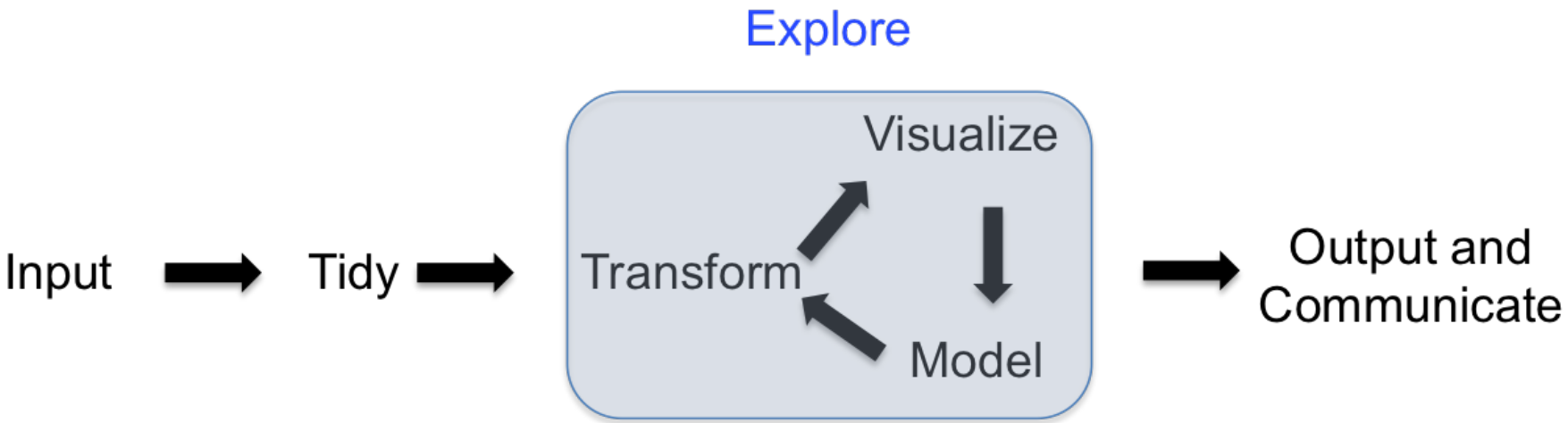
Lecture 10: Plotting in R

UNIVERSITY OF
SAN FRANCISCO

Abbie M Popa

BSDS 100 - Intro to Data Science with R

Onwards to Exploration!



Onwards to Exploration!

- Exploratory data analysis (EDA) is “an approach to analyzing data sets to summarize their main characteristics, often with visual methods... primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task." - Wikipedia

- Visualization is our primary means to develop hypotheses about a data set. It acts as a precursor to model building, discovery, and prediction

- Typically, we will transform and tidy our data to get it ready for visualization. But I think it will be easier for you to understand why we’re restructuring our data if we first learn how our plotting functions work.

Contents

- Conceptually, some different plot types
- Three different ways to plot in `R`
  - Base `R` (`plot()`, `boxplot()`, `hist()`)
  - Prettier plots with `ggplot2`
    - Input using `qplot()`
    - Input using `geoms`

So many plotting options!

You will (completely understandably!) ask me: "Why are we learning three ways to plot?"

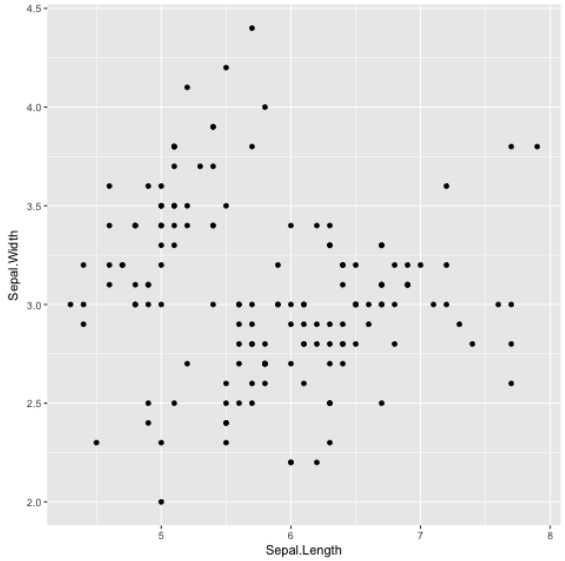
- When you join a lab or a team, you will rarely be coding from scratch, so you need to understand how the "legacy" code works.
- There are times you may need to use base `R` (company or collaborator requirements), but then `ggplot2` makes prettier graphs.

Part 1: Plots, Conceptually

Which is the plot for me?

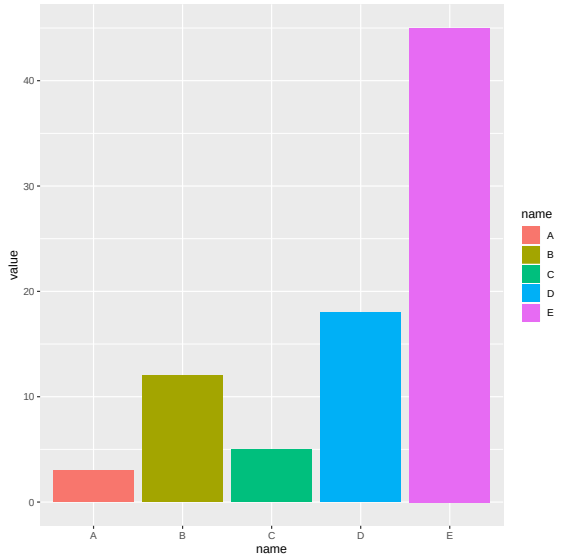
- Scatterplots, bar plots, histograms... there are many shapes of plot to choose from!
- We will discuss a few of the most common plots at a conceptual level to help you understand their advantages and disadvantages.
- Overall, the type of plot you need will depend on the nature of your data, and the question you are asking.

Scatter Plots



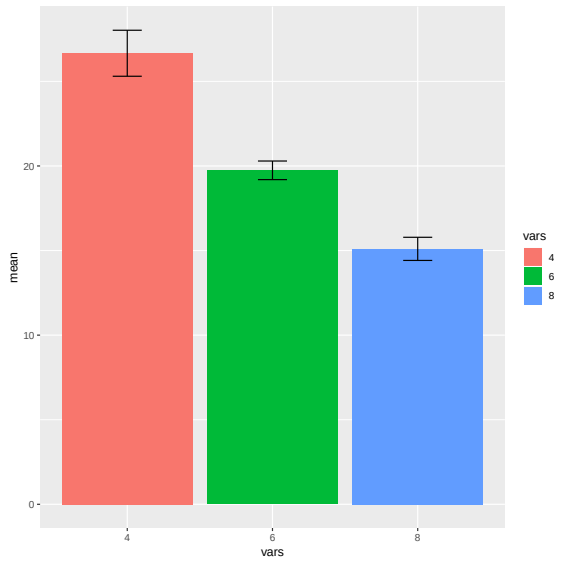
- Scatter plot: a graph in which the values of two variables are plotted along two axes against one another. These are used to explore whether or not there is correlation present between quantitative variables.
- Type of data: two continuous (e.g., numeric) variables
- Type of question: are these two variables related?
- More Info: https://www.r-graph-gallery.com/272-basic-scatterplot-with-ggplot2/

Bar Plots



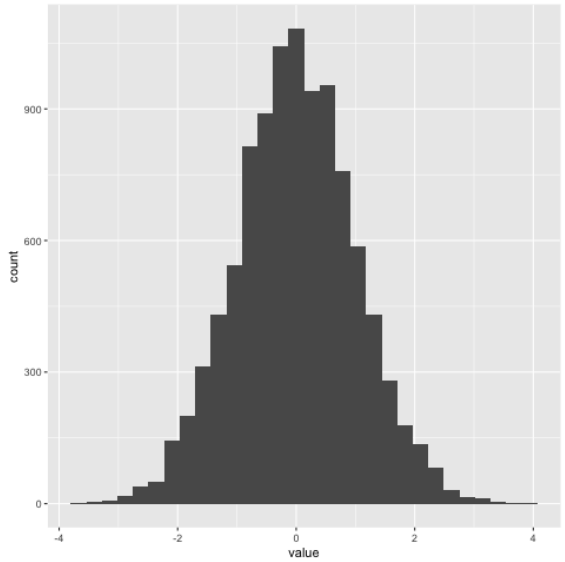
- Bar chart: a graph that presents grouped data with rectangular bars with lengths proportional to the values that they represent. Bars can be plotted vertically or horizontally.
- Type of data: one variable (in this case, the x-axis) is categorical, like group, the other is continuous
- Type of question: do groups differ on the continuous measure?

Bar Plots



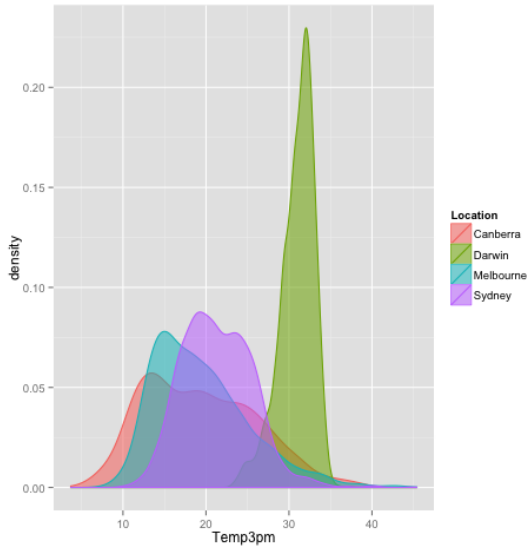
- Bar plots may be the most contentious type of plot!
- (Some) scientists love these, because it's easy to show if groups differ on a test of statistical significance
- But others (google #barbarplots) argue they hide too much information by merging all individuals in a group

Histograms



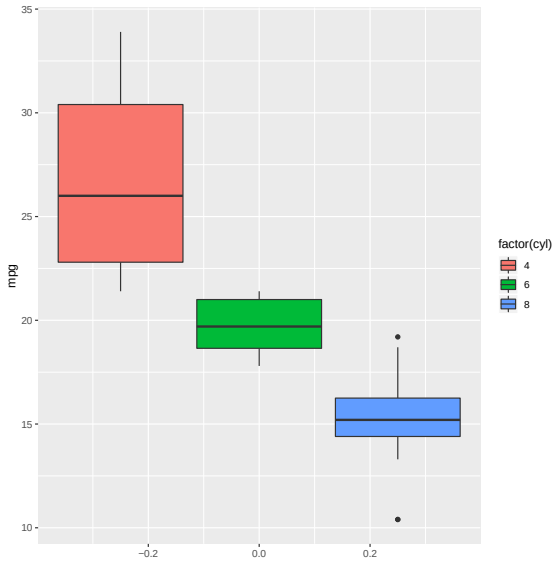
- Histogram: a diagram consisting of rectangles, the location (in this case, on the x axis) indicates the "bin" and the size (in this picture, height) indicates how many observations fall in that bin.
- Looks similar to bar plot, but it's different because it is only for one group!
- Type of data: One continuous variable
- Type of question: What is the distribution of this variable?
- More information https://www.r-graph-gallery.com/220-basic-ggplot2-histogram/

Density Plot



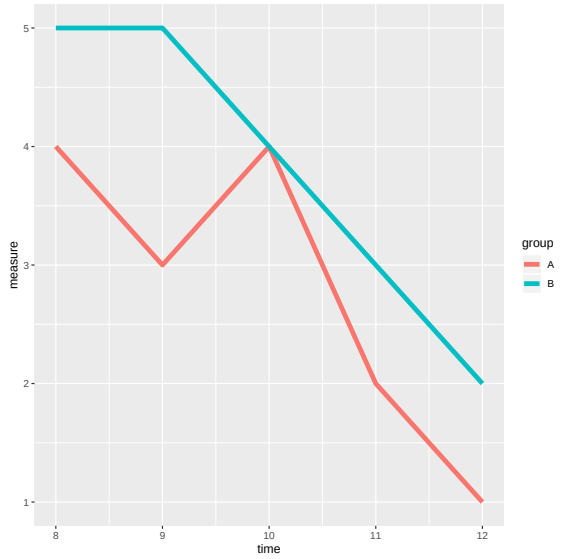
- Density curve: a “smoothed” version of a histogram
- Useful if you want to look at distributions of a few groups at the same time
- Since it shows probability that an observation falls in a bin your plot won’t be overwhelmed by groups that have more observations
- More information: http://www.r-graph-gallery.com/21-distribution-plot-using-ggplot2

Box Plot



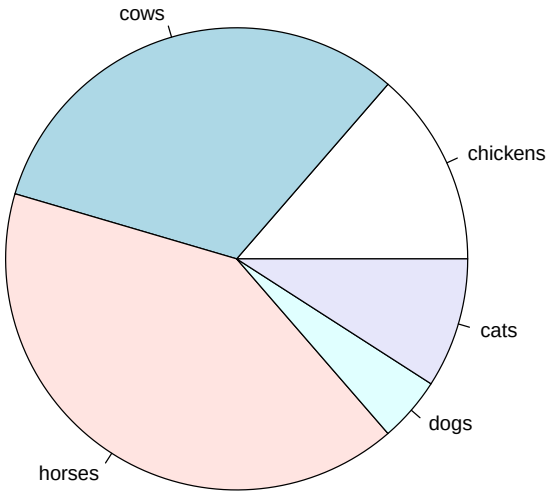
- Box plot: plot that displays the five number summary of a variable: the minimum, first quartile, median, third quartile, and maximum
- Type of data: One grouping (categorical) variable, and one continuous (e.g., numeric) variable
- Type of question: Do the groups differ on a variable
- Advantage over bar plot: Gives a more nuanced view of how individuals differ in each group

Line Plot



- Line plot: Plot shows the value (or perhaps the mean value for a group) of one variable at different values of another variable
- This is generally most useful when the value on the x axis is time, for example hour or day of a the y axis measure
- You may sometimes see this used in lieu of a bar plot, though this can be misleading when both values are not continuous

Pie Chart

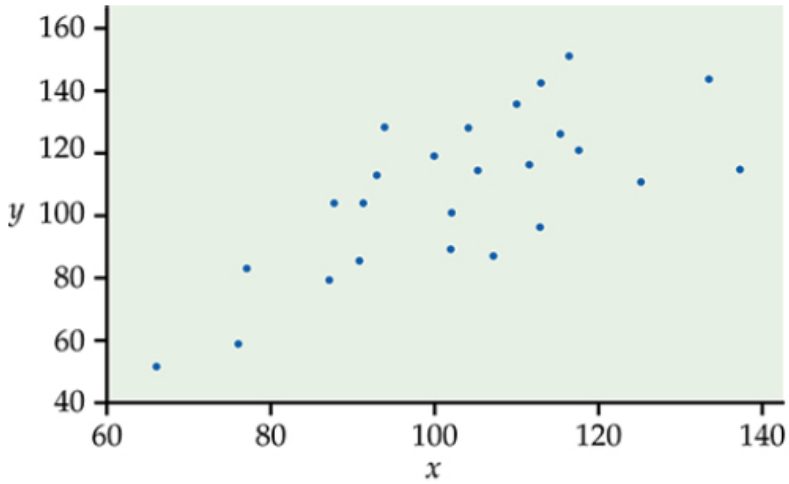


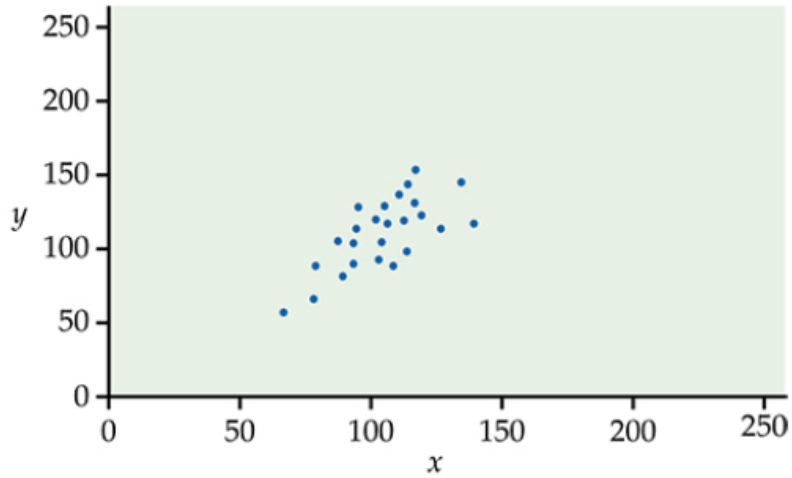
- Pie chart: graph that represents proportions of a categorical variable by dividing a circle into corresponding segments where each segment takes up the proportional amount of area. Do not use these!! People cannot compare areas of a circle. You can always use a bar chart instead.

Cautions

- Plots are **exploratory** only! They **do not** prove anything, but should be used for **intuition**.

- Plots can be misleading. Always consider the scale, variables involved, axes limits, etc. **Example:** These are the same scatter plots just with different scales!





Part 2: Making Plots in R

Graphics in Base R

- The base R graphics package, while being quick to code and functional, is not particularly aesthetically pleasing, cumbersome, and is outshone by what can be achieved using the `ggplot2` package
- While I would not recommend generating graphical output for external consumption using the base R graphics package, it does offer a quick and dirty way to graphically examine data

Plotting in Base R

- Scatterplots and bar charts can both be drawn using the `plot()`
- Look at the documentation for this function to get a better grasp of how exactly to plot what you want.
- There are *a lot* of arguments that can be provided to make detailed plots, including changing the x- and y- axes, tick marks, labels, the size, color, and type of lines in each plot, etc.
- I am only going to mention a few in this lecture, as we'll focus on the (prettier) way of plotting using the package `ggplot2`.

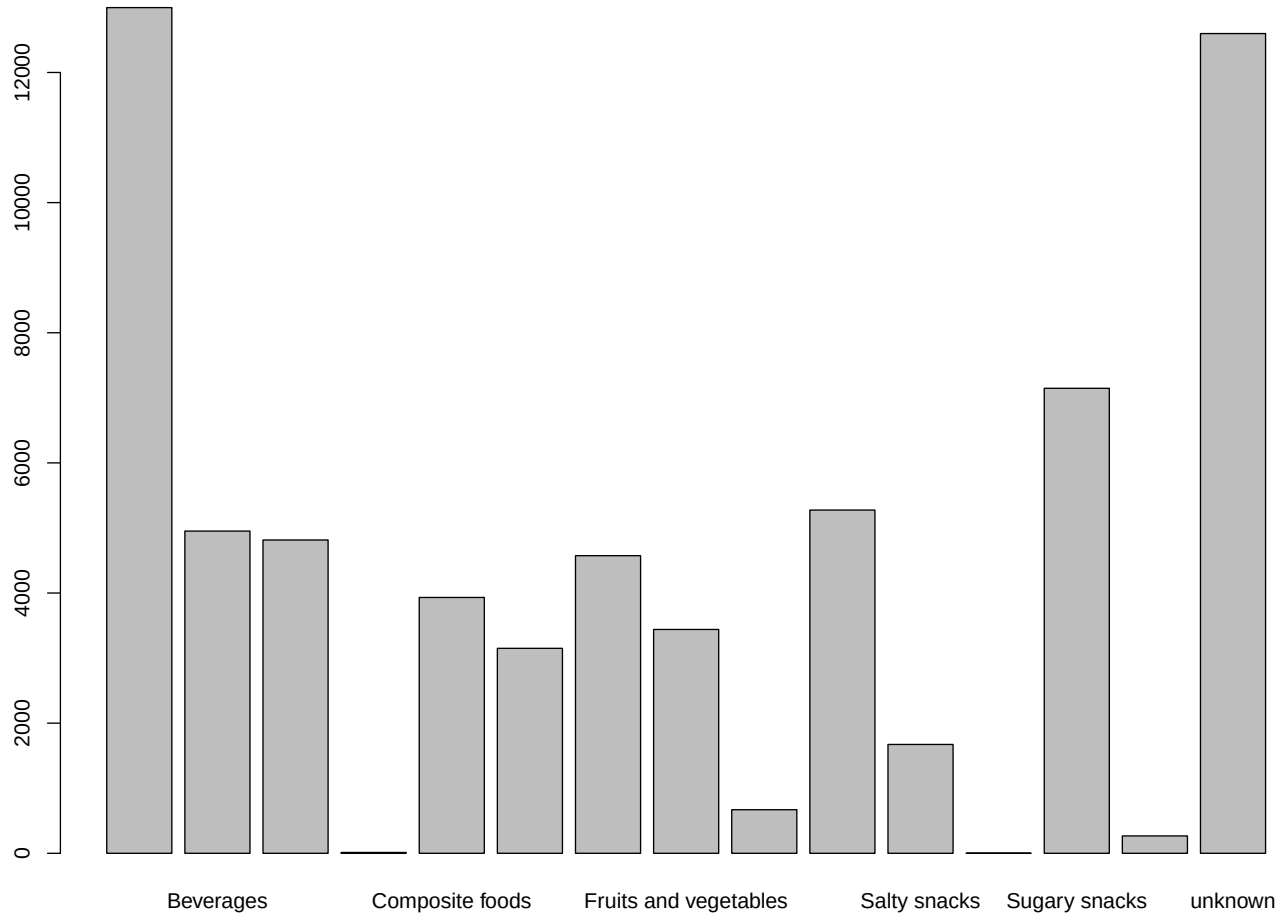
Plotting in Base R

- Let's look at a few examples...
- We will use several datasets in the following slides. Many are pre-loaded data sets in R, but we will at least need one other dataset, *Food.csv* from the GitHub site.
- Load the data into your R workspace using the following command:

```
> food <- read.csv(file = "https://raw.githubusercontent.com/abbiepopa/
bsds100/master/Data/Food.csv", header = TRUE, sep = ",")
```

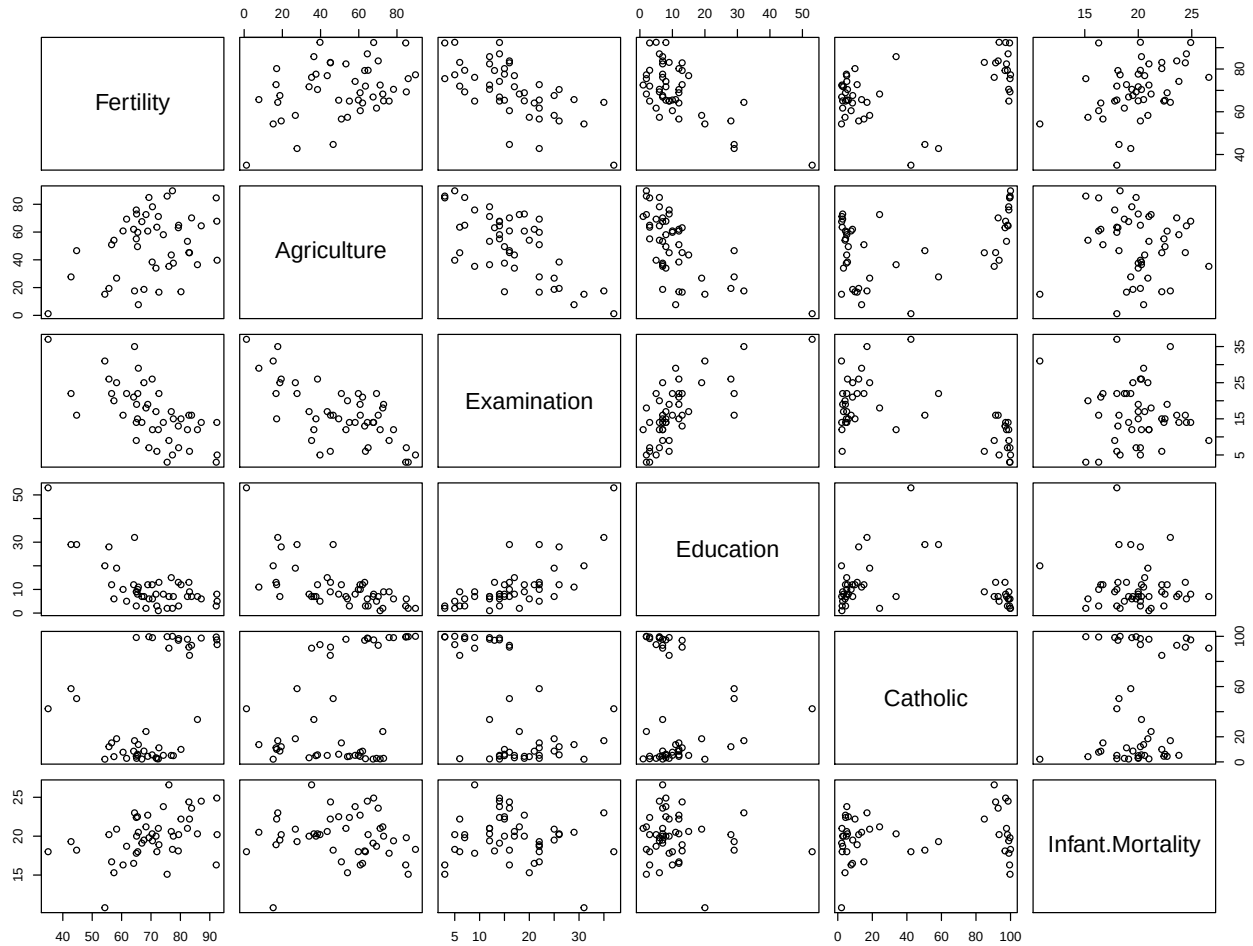
`plot()` a bar chart for categorical (factor) variables

```
> plot(food$pnns_groups_1) ## World Wide Food Facts Data (Kaggle)
```



plot() a pairwise scatter plot matrix for data frame

```
> plot(swiss) ## Base R Data
```

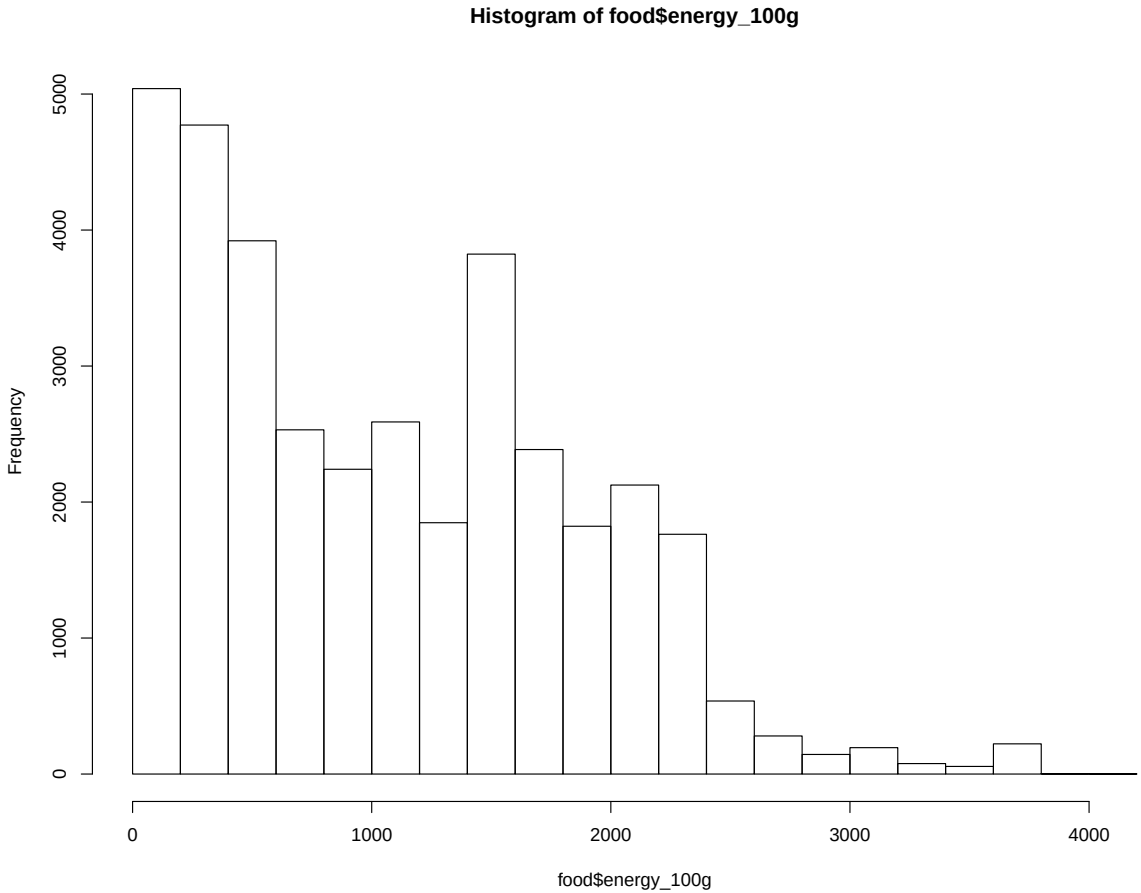


`plot()` made a bar chart one time, and a scatter plot a different time?

- `plot()` makes it's best guess based on the type of data you give it
- This is another advantage of some of `ggplot2`, we will get more control

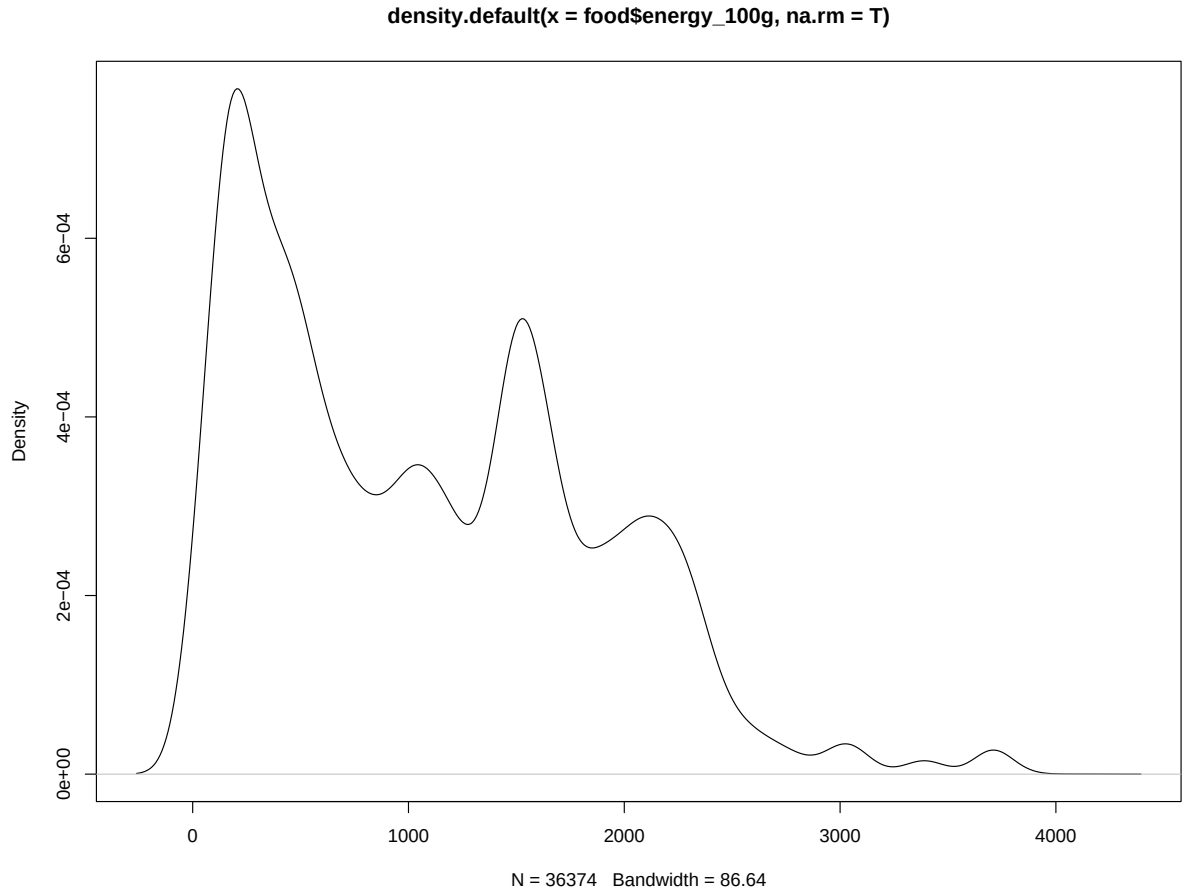
Make a histogram with `hist()` for continuous (numeric) variables

```
> hist(food$energy_100g) ## World Wide Food Facts Data (Kaggle)
```



`plot()` the kernel density for numeric variables

```
> plot(density(food$energy_100g, na.rm = T))
```



Interlude...

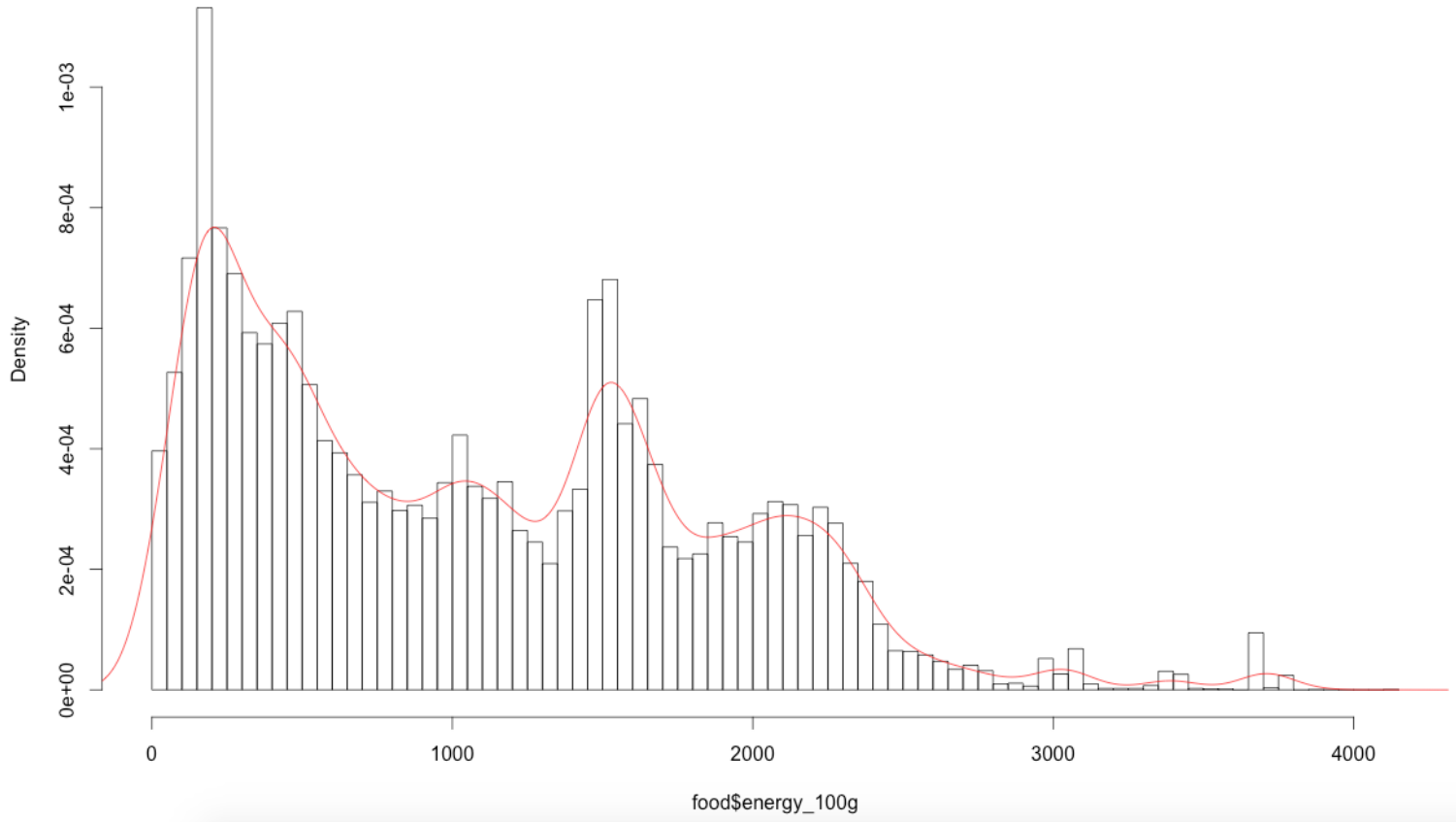
Q: How do you remember all these commands?!

A: I don't! I remember the ones I use the most often, the rest I look up when I need them.

- We can also layer multiple plots on top of each other, some functions like `plot()` and `hist()` produce a base layer, while others like `lines()` and `points()` go on top of base layers
- We can also modify what our plots look like, for example, their color

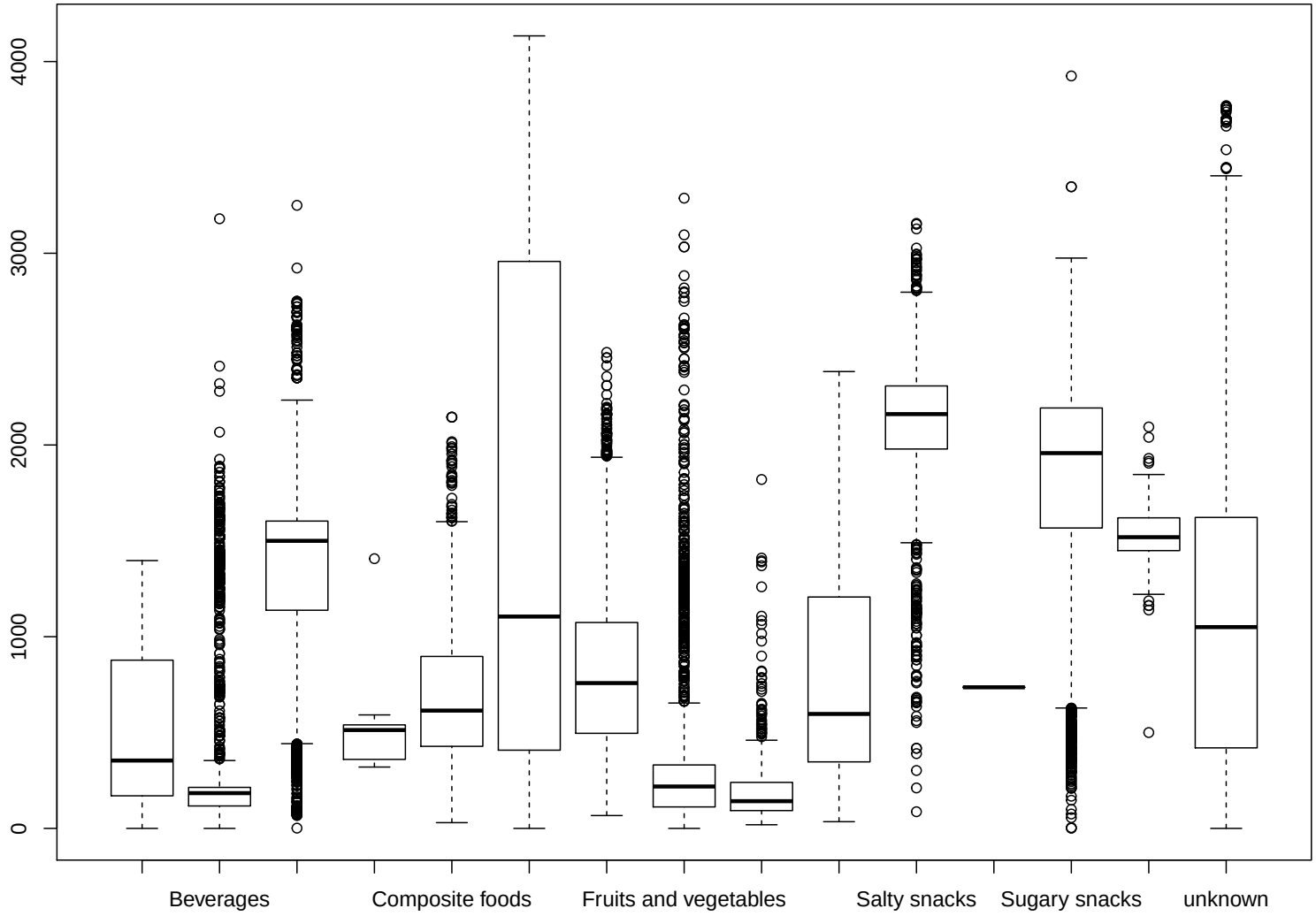
Overlay a kernel density on a probability histogram

```
#100 bins; probability density
>  hist(food$energy_100g, n = 100, probability = TRUE)
#red density curve
>  lines(density(food$energy_100g, na.rm = T), col = "red")
```



Make a `boxplot()` for numeric variables

```
> boxplot(food$energy_100g ~ food$pnns_groups_1)
```



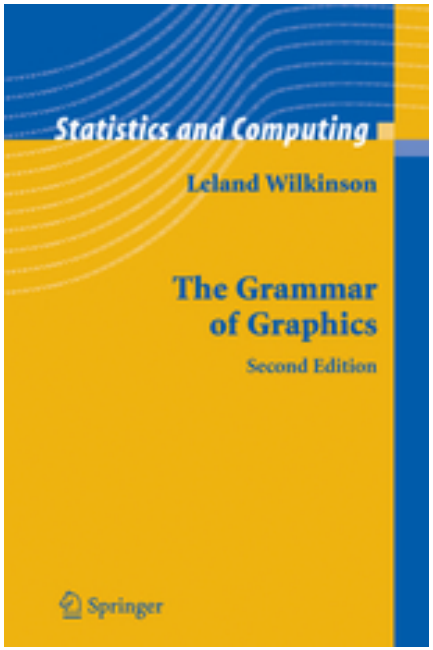
Useful arguments for `plot()` - esque functions

- There are *many* arguments for the `plot` function!
- Changing size, shape, color, line type, point type, text size, labels, axes, etc. etc.
- Too many to list here, but see this website as a resource: http://www.statmethods.net/advgraphs/parameters.html
- For now, we will focus on a more state of the art way of visualizing data using the package `ggplot2`.

Part 3: `ggplot2`

What is ggplot2?

- ggplot2 is an R package for producing graphics
- ggplot2 differentiates itself from other packages as it is based on a deep underlying grammar, adopted from Wilkinson's *The Grammar of Graphics*



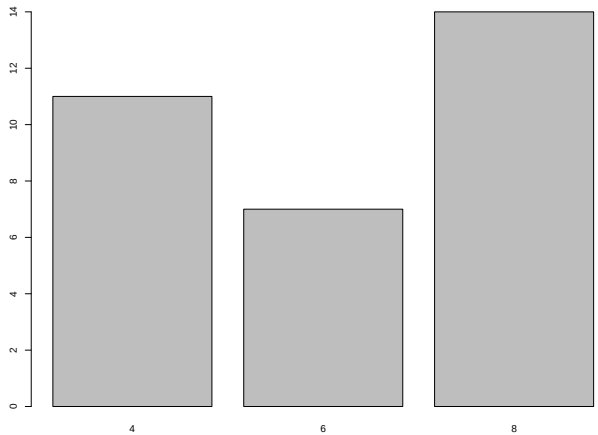
What is `ggplot2`?

- This grammar of graphics is composed of a set on independent components that can be combined in many different ways
- You are not limited to a set of pre-specified graphics, but you can create new graphics that are precisely tailored to your problem
- Publication-quality graphs can be coded in seconds, with many of the extraneous details (e.g, legends) taken care of by `ggplot2` default behavior

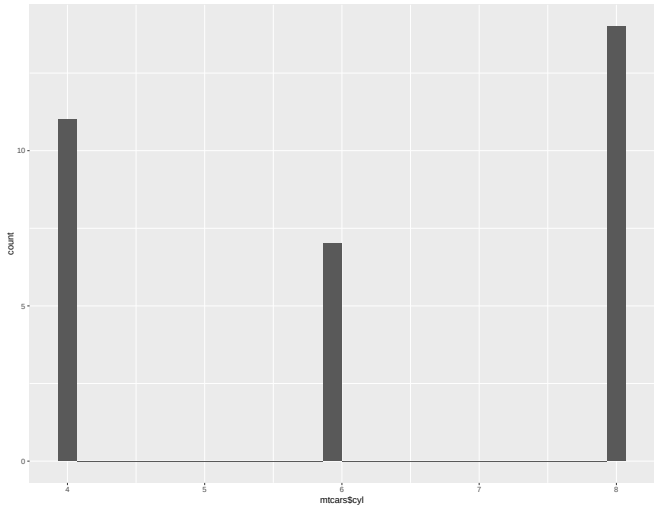
qplot() versus plot()

- plot() is the base R graphics plotting package
- qplot() is a function from the ggplot2 package meant to mimic and improve upon the simplicity and speed of plotting in base R

```
plot(as.factor(mtcars$cyl))
```




```
qplot(mtcars$cyl)
```

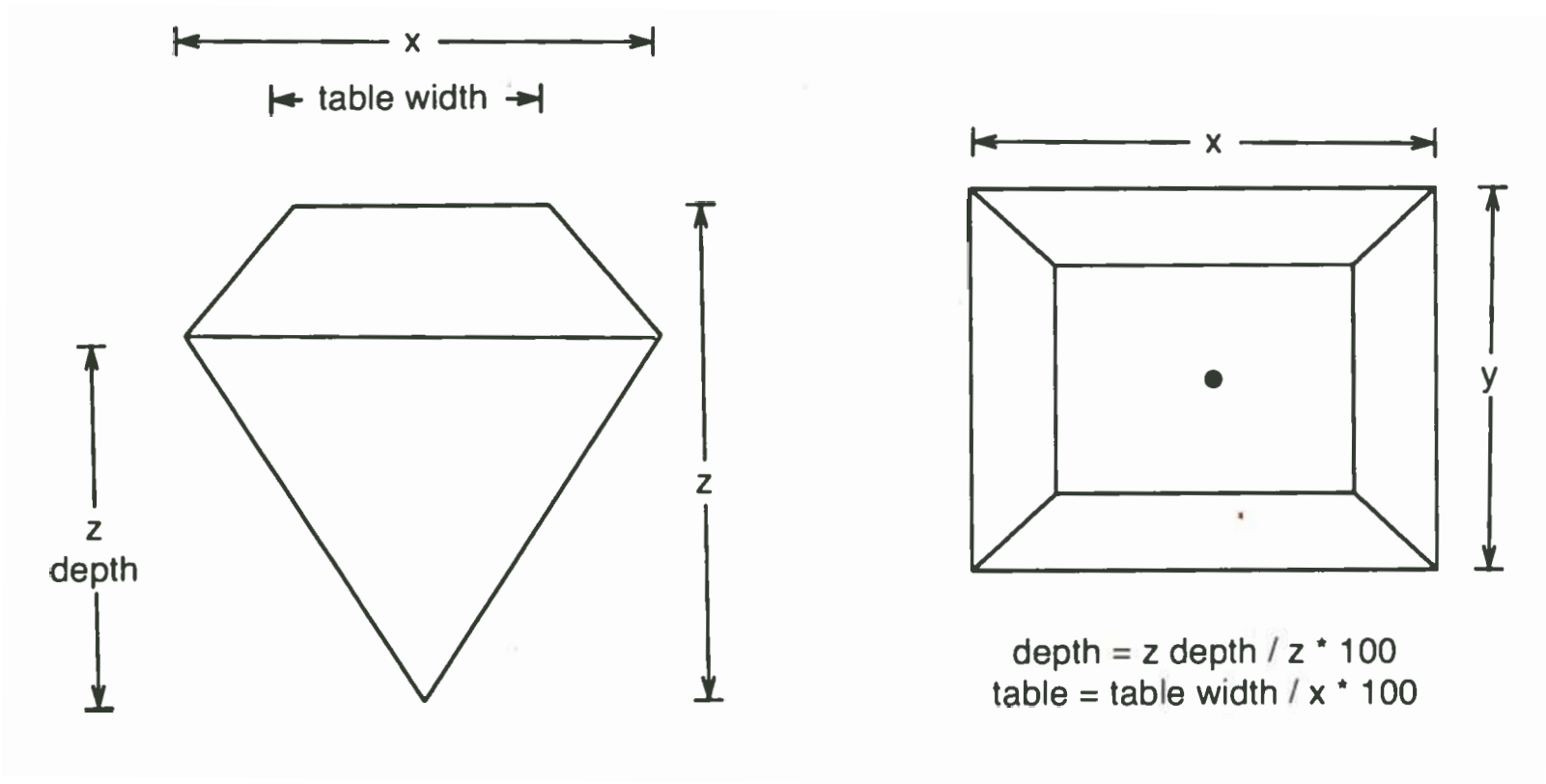


Case Study: The `diamonds` dataset

A data frame with 53940 rows and 10 variables:

- price: price in US dollars (\$326–\$18,823)
- carat: weight of the diamond (0.2–5.01)
- cut: quality of the cut (Fair, Good, Very Good, Premium, Ideal)
- color: diamond colour, from J (worst) to D (best)
- clarity: a measurement of how clear the diamond is (I1 (worst), SI1, SI2, VS1, VS2, VVS1, VVS2, IF (best))
- x: length in mm (0–10.74)
- y: width in mm (0–58.9)
- z: depth in mm (0–31.8)
- depth: total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)
- table: width of top of diamond relative to widest point (43–95)

Case Study: The `diamonds` dataset



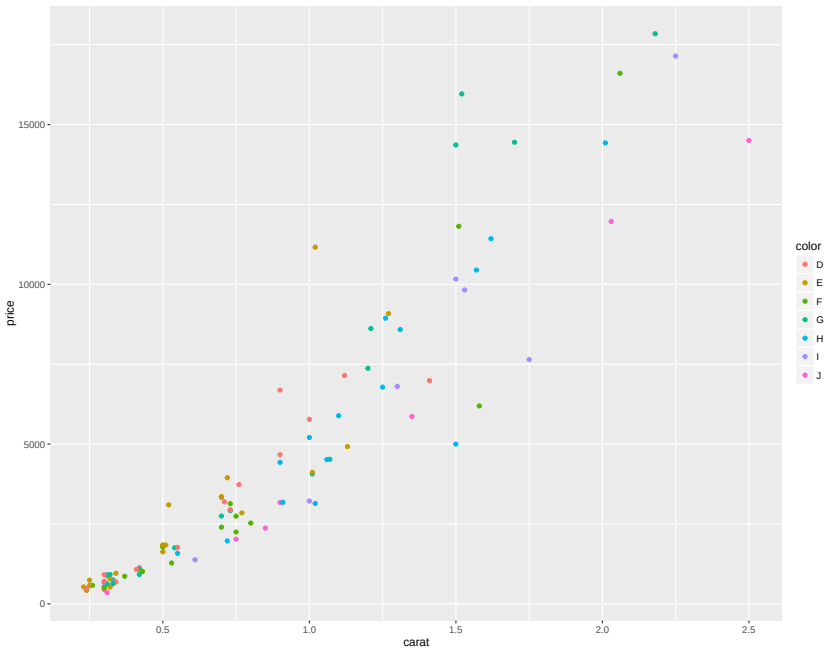
- When you load `ggplot2` you gain access to the `diamonds` data set!

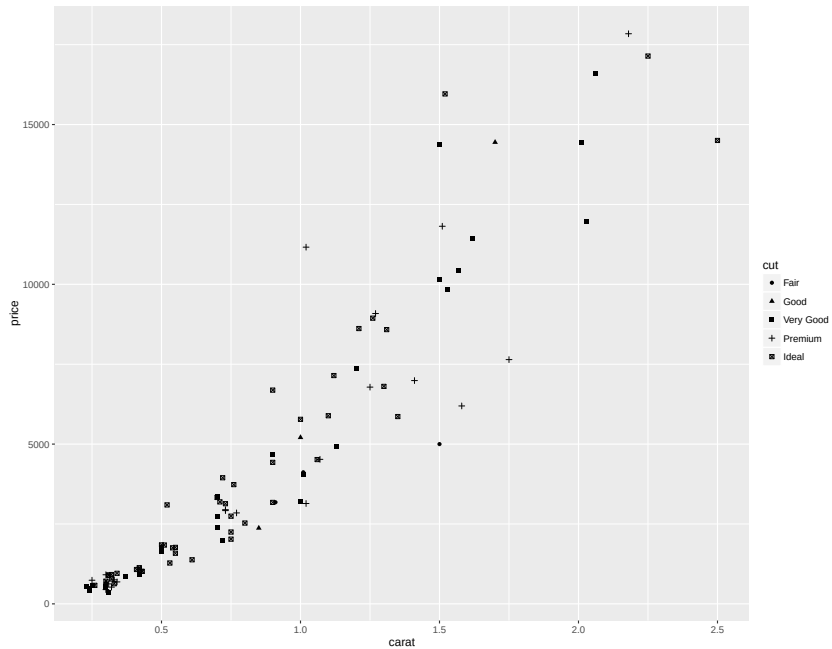
Aesthetic Attributes

- When using `qplot()` instead of base graphing in R we assign aesthetics: colors, sizes, shapes, etc.
- Color, size and shape are all examples of aesthetic attributes, visual properties that affect the way observations are displayed
- For every aesthetic attribute, there is a function, called a **scale**, which maps data values to valid values for that aesthetic
- It is the scale that controls the appearance of the points and associated legend

Example: Diamond Carat against Price


```
> set.seed(1410)
> diamonds_subset_01 <- diamonds[sample(nrow(diamonds), 100), ]
> qplot(carat, price, data = diamonds_subset_01, colour = color)
> qplot(carat, price, data = diamonds_subset_01, shape = cut)
```



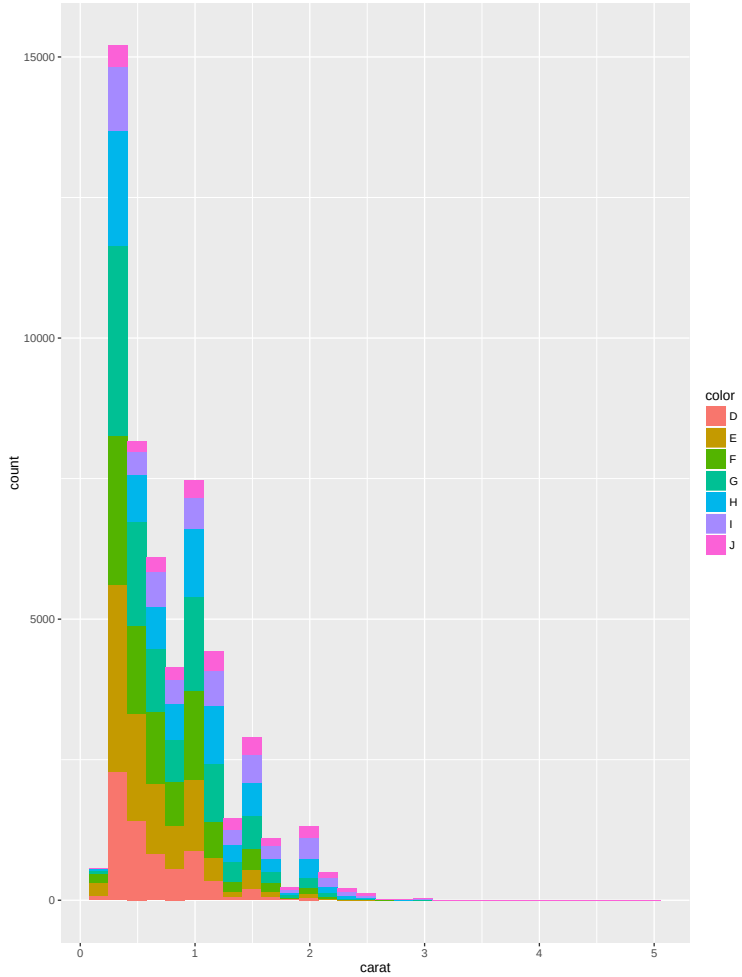


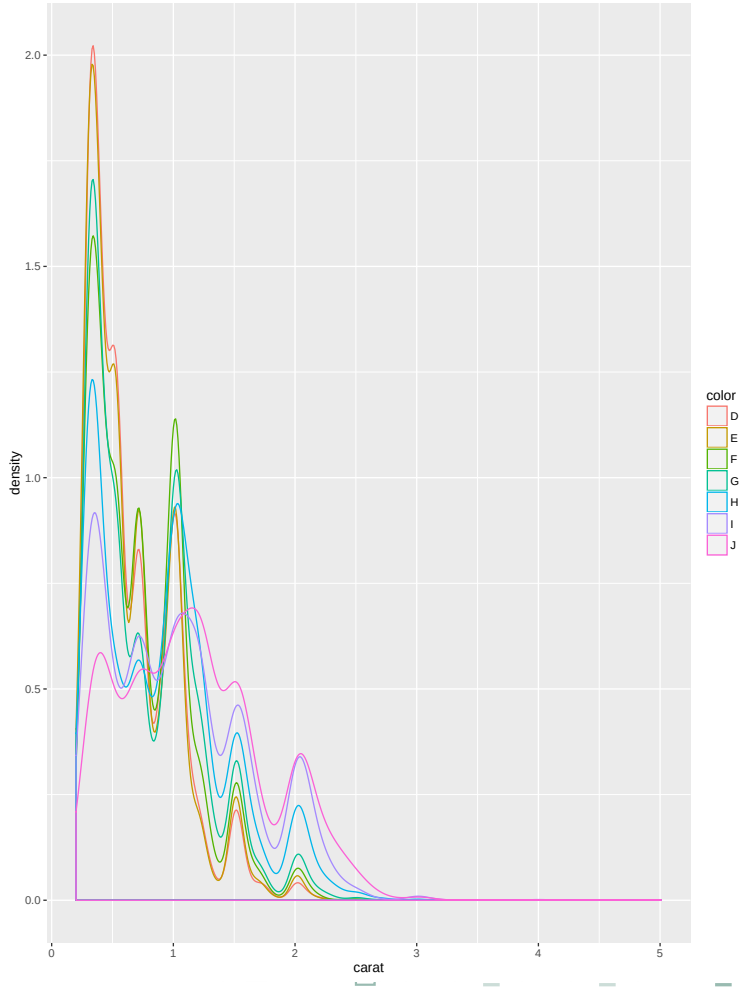
Geometric Objects

- So we managed to choose the type of plot earlier by using the silly `as.factor()` wrapper, but `ggplot2` provides a better way
- Geometric objects, or `geom`s as they are commonly known and used, describe the type of object used to display the data
- Common one-dimensional `geom` include
  1. `geom = "histogram"`
  2. `geom = "density"`
  3. `geom = "bar"`
- Common two-dimensional `geom` include
  1. `geom = "point"`
  2. `geom = "boxplot"`
  3. `geom = "line"` (line from left to right)
  4. `geom = "smooth"` (smoothed line with standard error)

geom = "histogram" & geom = "density"

```
> qplot(carat, data = diamonds, geom = "histogram", fill = color)
> qplot(carat, data = diamonds, geom = "density", colour = color)
```



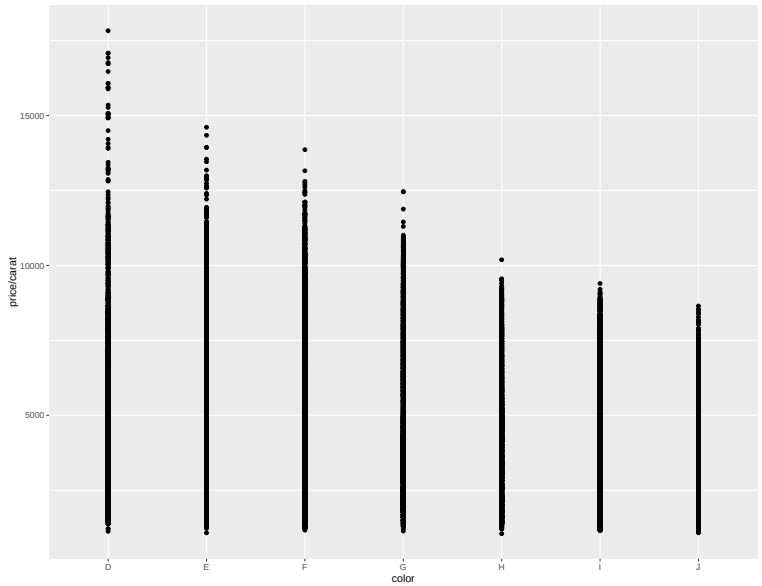


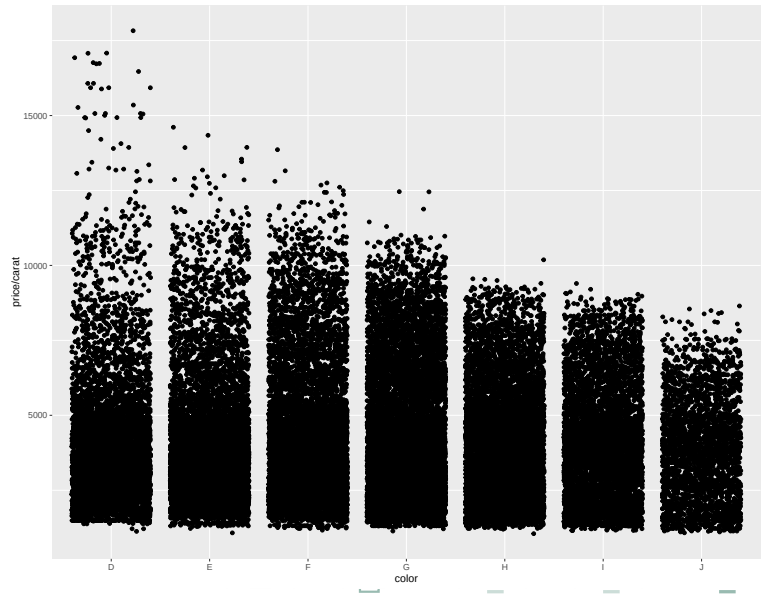
`geom = "histogram"` & `geom = "density"`

- With histograms, play around with the `bins` argument to select the number of bins that best communicates the structure of the data
- Similarly, iterate through various values of `adjust` for density plots to find the right smoothing factor (larger values of `adjust` generate smoother lines)

geom = "jitter"

- geoms can also be used to make adjustments to your graph
- For example, when plotting a significant number of points with a continuous response variable (y) and a categorical predictor (x), plots can often suffer from overplotting, i.e., multiple superimposed points
- geom = "jitter" spreads (jitters) points in an effort get a better sense of how many points exist at any given response level

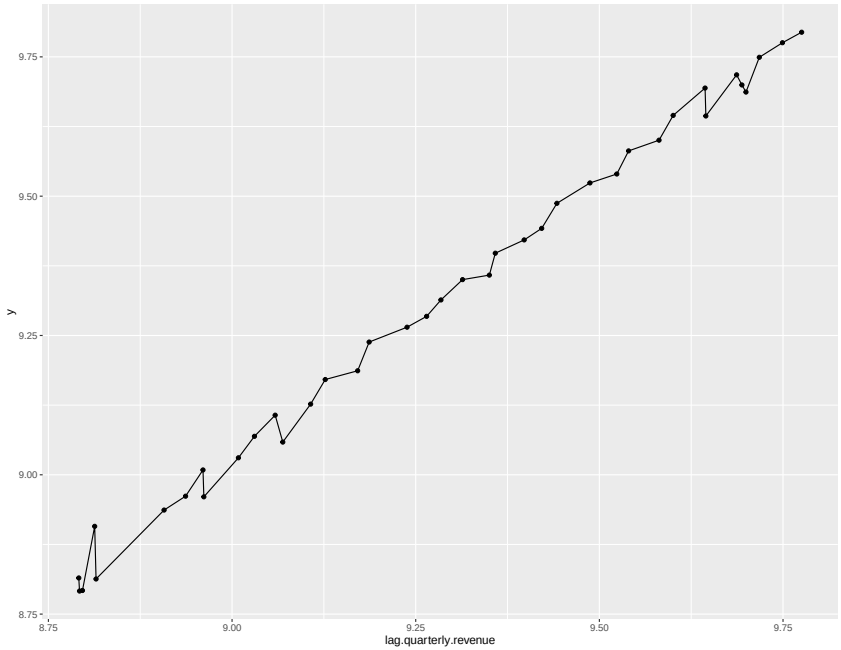




Using Multiple `geom`

- Time series graphs are often nicer not just with a line connecting adjacent data, but with points representing the data as well
- Multiple geom can be employed in a single `qplot()` by creating a vector of `geom`, e.g., `geom = c("line", "point")`

```
qplot(lag.quarterly.revenue, y, data = freeny, geom = c("line", "point"))
```

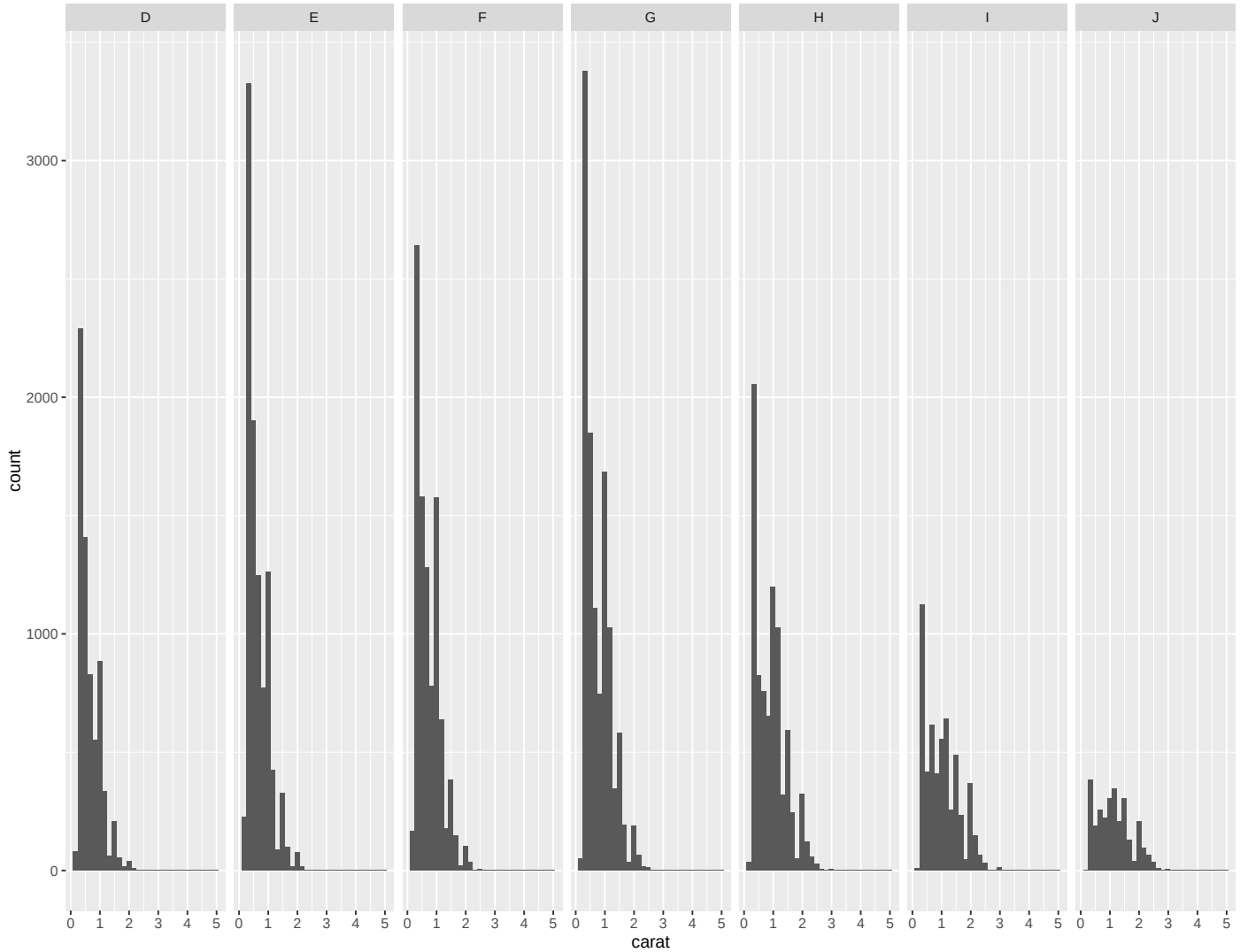


Faceting

- Although aesthetics may be employed to compare subgroups, all groups are drawn on the same plot
- Faceting takes an alternative approach, creating tables of graphics by splitting data into subsets, and displaying sub-graphs (typically) in a grid arrangement
- Use the `facet = <rowVar> ~ <colVar>` option—where both `rowVar` and `colVar` should be categorical variables
- Use `.` notation as an optional placeholder

Faceting Example

```
> qplot(carat, data = diamonds, facets = . ~ color, geom = "histogram")
```



Some notes on coding style

- What makes "good" coding style?
- Code that is succinct and has clearly named variables is easier to read than code that has many unnecessary lines and unclear variable names
- When you first start coding it is hard to have good style, but a few tips will bring you a long way

Unnecessary Variables

- Let's say we want to get the even numbers from 2 to 10, each number repeated 2 times, in order
- I would do this by writing the functions inside each other, like so:

```
sort(rep(seq(2, 10, 2), 2))
```

- Note, the operations happen from the inside out, first it makes a sequence of the even numbers from 2 to 10, then it repeats this twice, then it sorts it
- This code is "good" because it is succinct and doesn't make any unnecessary variables, but most of you will use transition variables for each step.

Good Variable Names

- Many (most?) of you will use transition variables while you are learning, this is okay! But I encourage you to give your variables clear names for readability. For example:

```
v1 <- seq(2, 10, 2)
v2 <- rep(v1)
v3 <- sort(v2)
```

Can be rewritten as:

```
even_nums <- seq(2, 10, 2)
rep_even_nums <- rep(even_nums)
sorted_rep_even_nums <- sort(rep_even_nums)
```

- I also encourage you at the end of this to remove the transition variables with `rm(list = c("even_nums", "rep_even_nums"))`

One more option

- The library `magrittr` will allow you to write functions from left to right instead of inside each other
- After loading the library with `library(magrittr)` place `%>%` between each step
- For example, `sort(rep(seq(2, 10, 2), 2))` becomes

```
seq(2, 10, 2) %>% rep(, times = 2) %>% sort()
```

- Or, when saving the object:

```
sort_rep_even_nums <- seq(2, 10, 2) %>% rep(,
times = 2) %>% sort()
```

- Please feel free to use this **if** it is easier for you, it is not required

Two important things...

- When coding in R you **must** save anything you do to modify a character, for example:

```
dog_cat <- factor(c("dog", "dog", "cat", "dog"))
as.character(dog_cat)
```

  does nothing! the `dog_cat` vector is still a factor

- To change it, remember to use the assignment symbol:

```
dog_cat <- factor(c("dog", "dog", "cat", "dog"))
dog_cat <- as.character(dog_cat)
```

Two important things...

- Subsetting will never go away, for a thorough review please review the slides, notes, and answer keys related to data frames
- The most common way you will subset a data frame is to select the data frame, a conditional subset of the rows, and a column (or two!), or as a template:

```
df_name[row_conditional, column]
```

- For example, to get all the "versicolor" iris "Sepal.Length"s:

```
iris[iris$Species == "versicolor", "Sepal.Length"]
```

- The column you select does **not** need to be the same as the column used for row selection, for example, here we select rows based on the "Species" but output the "Sepal.Length" column
- Also be careful with quote and $ placement

In sum...

- Don’t panic, you’re all doing fine, but I hope these tips will help you with the homework and next case study
- Programming is inherently cumulative, don’t forget what you learned early on!

New Day Warm-Up

- The `iris` data set contains five variables (columns), `Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`, and `Species`.
- Fill in the blanks in `plot(_____ , _____)` to produce a scatter plot with `Sepal.Length` on the x-axis and `Sepal.Width` on the y-axis:

(a) `iris$Sepal.Length`
(b) `iris`
(c) `Sepal.Width`
(d) `iris$Sepal.Width`
(e) `Sepal.Length`

- Fill in the blanks in `qplot(_____, _____, data = _____, _____ = _____)` to produce a scatter plot with `Sepal.Length` on the x-axis, `Sepal.Width` on the y-axis, and points colored by `Species`

(a) `iris$Sepal.Length`
(b) `iris`
(c) `Sepal.Width`
(d) `iris$Sepal.Width`
(e) `Sepal.Length`
(f) `iris$Species`
(g) `Species`
(h) `fill`
(i) `colour`

`qplot()` versus `ggplot()`

- Plots can be created in two ways:
  1. all at once with `qplot()`
  2. piece-by-piece using `ggplot()` and layer functions, which provide far more control and complexity over a graph
- `ggplot()` allows us to build a plot *layer by layer*, where each layer can come from a different data sets and have a different aesthetic mappings
- This advanced behavior differs from `qplot()` which permits only a single data set and a single set of aesthetic mappings

From `qplot()` to `ggplot()`

```
> qplot(carat, price, data = diamonds, colour = cut)
```

- With `ggplot()`, this is a little more complicated
- Aesthetics in `ggplot()` are wrapped in `aes()`
- `ggplot()` code generates a *plot object*—**not** the plot itself:

```
> ggplot(diamonds, aes(carat, price, colour = cut))
```

Note: This generates an *empty* plot because there are no `geom`, i.e., layers

Layers

- A minimal layer may do nothing more than specify a `geom`
- By adding a `geom`, we fill the empty graph with data

```
> ggplot(diamonds, aes(carat, price, colour = cut)) + geom_point()

# note that a ggplot plot object can be stored in a variable
# the following code generate identical results

> myPlot <- ggplot(diamonds, aes(carat, price, colour = cut))

> (myPlot <- myPlot + geom_point())
```

Layers

- We can also store any layers (not just the base) as variables
- Layers allow for writing clean, concise and reusable code.
- **Example**: to include a thick, blue line of best fit, one can create the generic layer, and then add it to any plot:

```
> myBestFitLine <- geom_smooth(method = "lm", se = F,
                               colour = alpha("steelblue", 0.5), size = 2)

> qplot(sleep_rem, sleep_total, data = msleep) + myBestFitLine

> qplot(awake, brainwt, data = msleep) + myBestFitLine
```

The Plot Object as a Variable

- You can inspect the structure of the plot without actually plotting it

```
> myPlot <- ggplot(diamonds, aes(carat, price, colour = cut)) +
              geom_point()

> summary(myPlot)
data: carat, cut, color, clarity, depth, table, price, x, y, z
  [53940x10]
mapping:  x = carat, y = price, colour = cut
faceting: facet_null()
-----------------------------------
geom_point: na.rm = FALSE
stat_identity: na.rm = FALSE
position_identity
```

Preparing data for `ggplot`

- Input data for `ggplot()` **must be a data frame** (can be a tibble data frame!)
- To map variables to different parts of the plot, we use the `aes()` function, which takes a list of aesthetic-variable pairs
- Example: `aes(x = carat, y = price, colour = color)` maps `carat` to `x`, `price` to `y` and `colour` to `color`

aes()

Default aesthetic mappings can either be set when the plot is initialized or modified at a later time

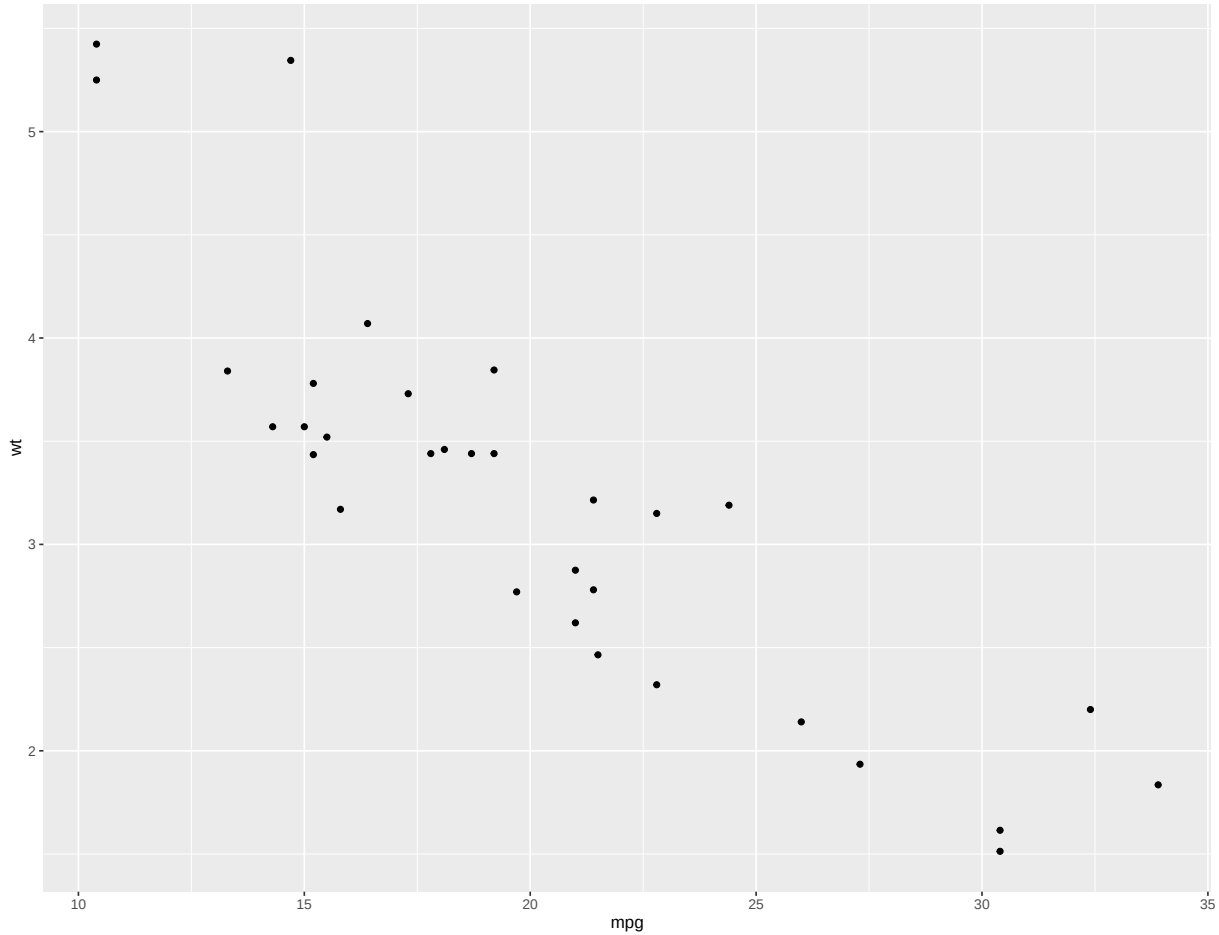
```
> (myPlot <- ggplot(mtcars, aes(x = mpg, y = wt)) + geom_point())
```

```
> myPlot + geom_point(aes(colour = factor(cyl)))
```

- Aesthetic mappings specified in a layer affect only that layer
- Axis labels and legend titles will be based on what you put in `ggplot()` (see examples in following slides)

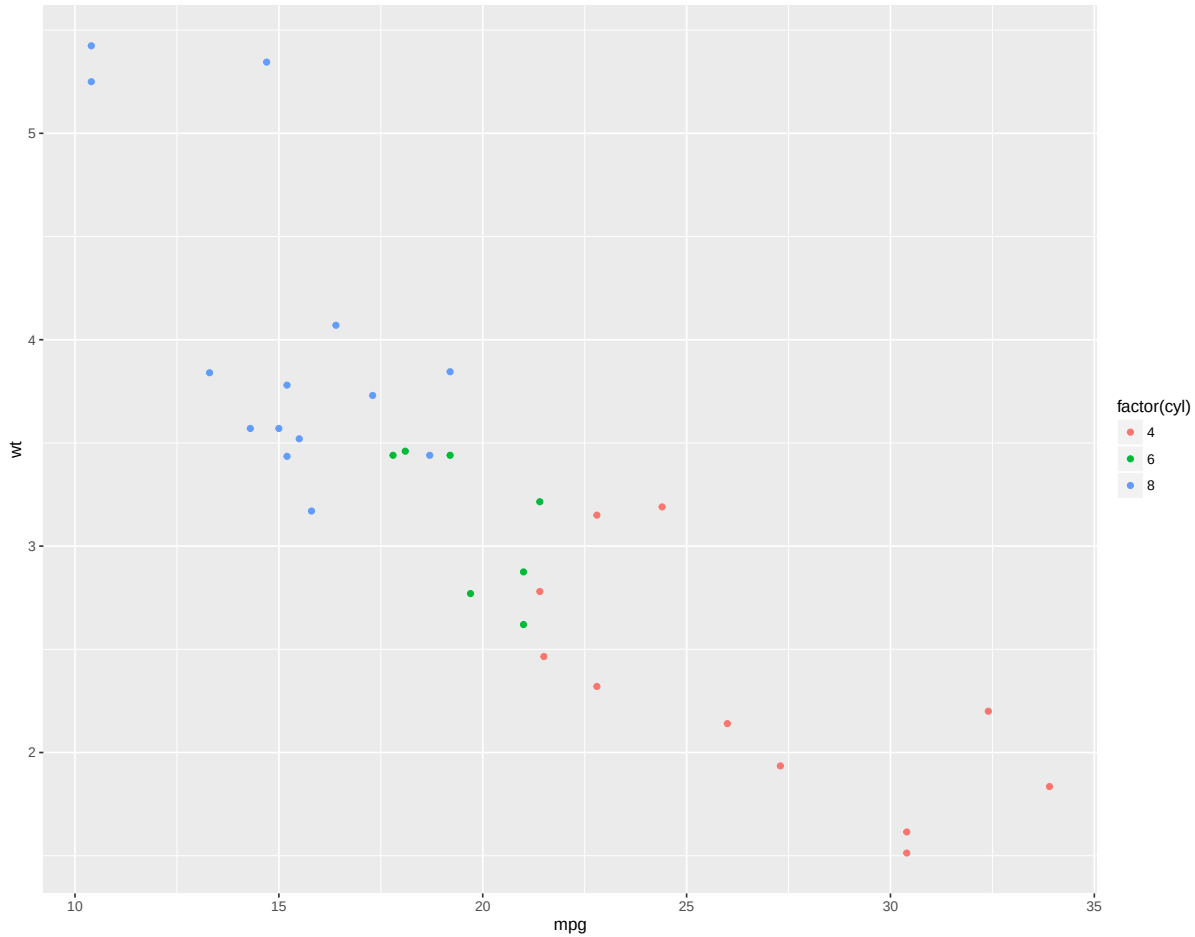
Updating Aesthetics on Layers

```
> myPlot <- ggplot(mtcars, aes(x = mpg, y = wt))
> myPlot + geom_point()
```



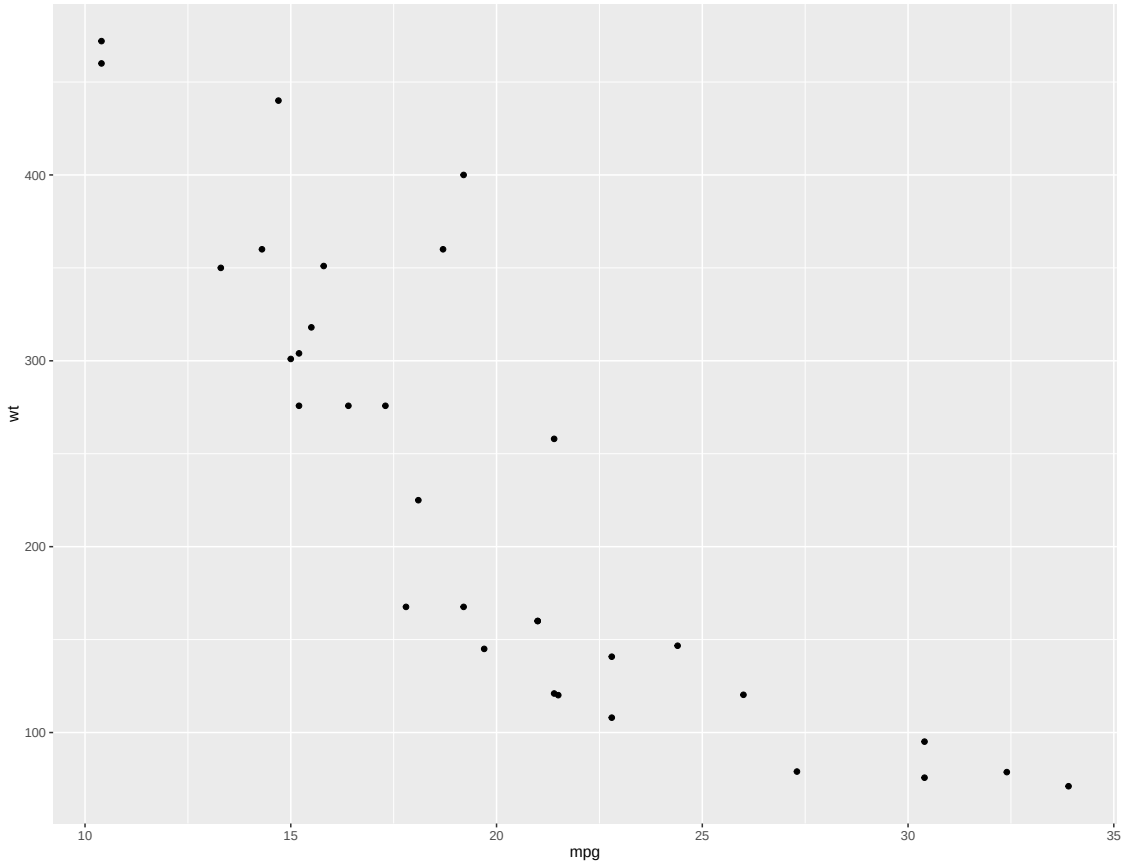
Updating Aesthetics on Layers

```
> myPlot + geom_point(aes(colour = factor(cyl)))
```



Updating Aesthetics on Layers

```
> myPlot + geom_point(aes(y = disp))
```



Grouping

- `geom` is divided into two groups: individual and collective
- An individual `geom` has a distinctive graphical object for each observation in a data frame, e.g., `geom_point()` has single point for each observation
- Collective `geom` represent multiple observations, the result of a statistical summary or just fundamental to the display of a particular `geom`, e.g., boxs or bars
- The `group` aesthetic controls which observations go into which individual graphical element (e.g, into a box, a bar, or a line)

Grouping

- **Usually** `ggplot()` can figure out how to group variables without us setting a group, but sometimes it will guess incorrectly
- We can manually tell `ggplot()` how to group variables using the `group =` setting
- `interaction()` is useful if a single pre-existing variable doesn't cleanly separate groups, but a combination does

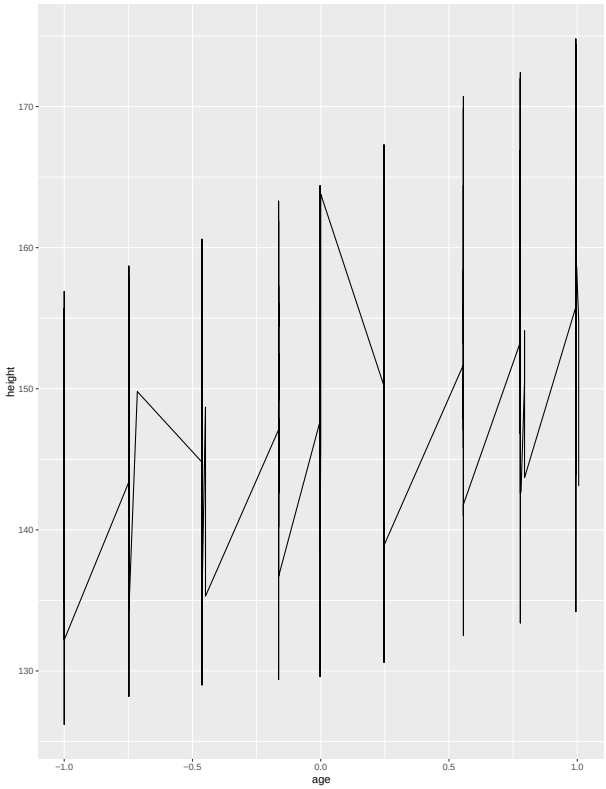
Grouping Example

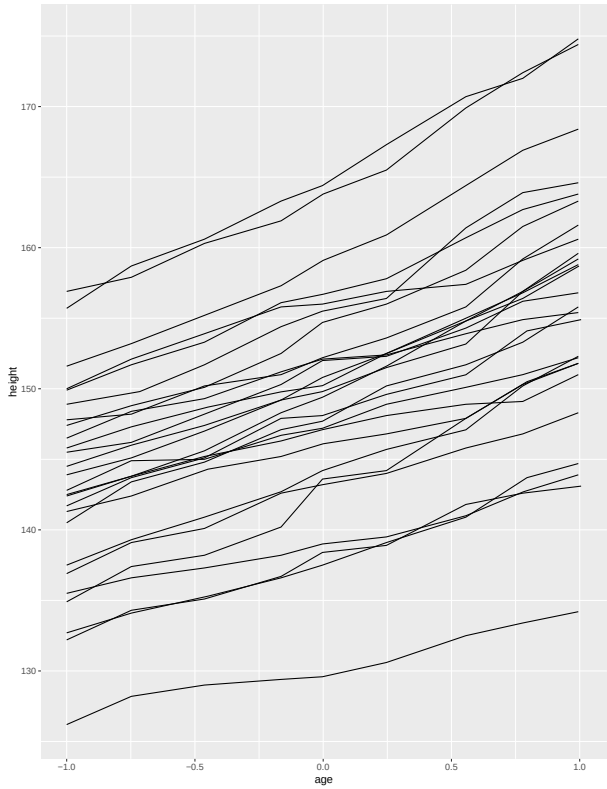
- Grouping examples on the following slides will use longitudinal data from `nmle` package called `Oxboys`, which records `height`, `age` and `subject` (26 boys) measured at nine `occasion`s
- **Objective**: separate the data into groups to distinguish between individual subjects, but render all of them in the same way

Multiple Groups, One Aesthetic

```
ggplot(Oxboys, aes(age, height)) + geom_line()
```

```
ggplot(Oxboys, aes(age, height, group = Subject)) + geom_line()
```



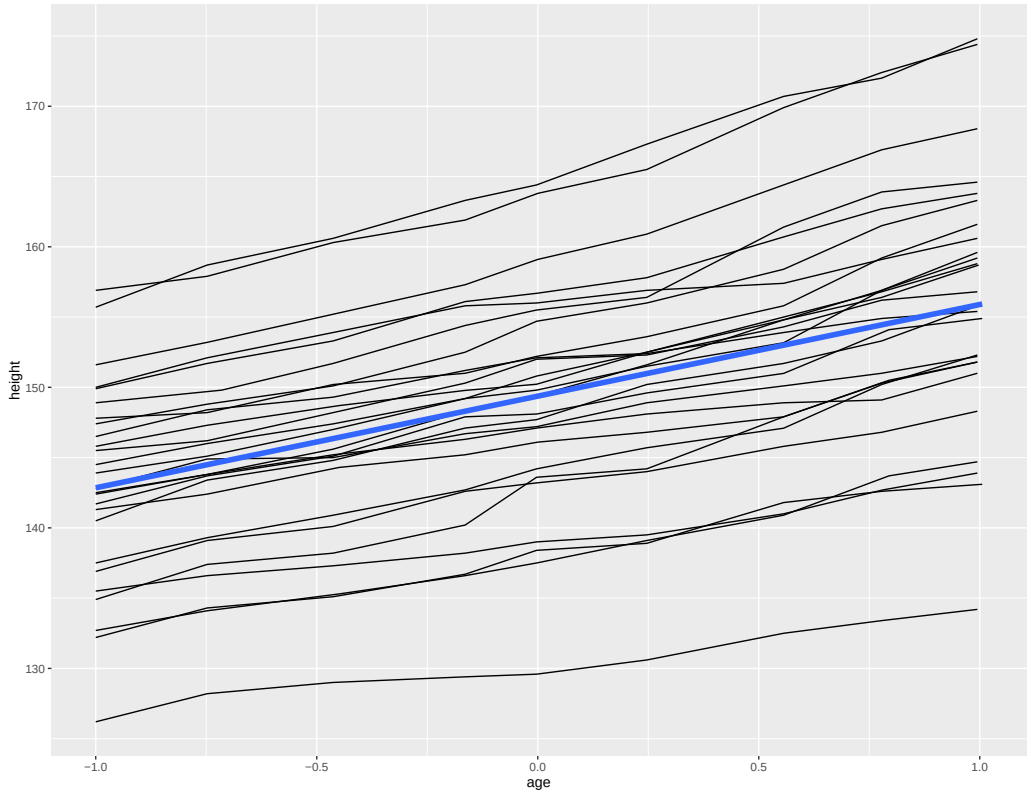


Different Groups on Different Layers

- **Objective**: We want to add an **average** line to this plot
- But an average of each of our groups, when our groups are not one person each, doesn't make sense!
- We want to our average to use **all** data, a "trick" for this is to set `group = 1`

Different Groups on Different Layers

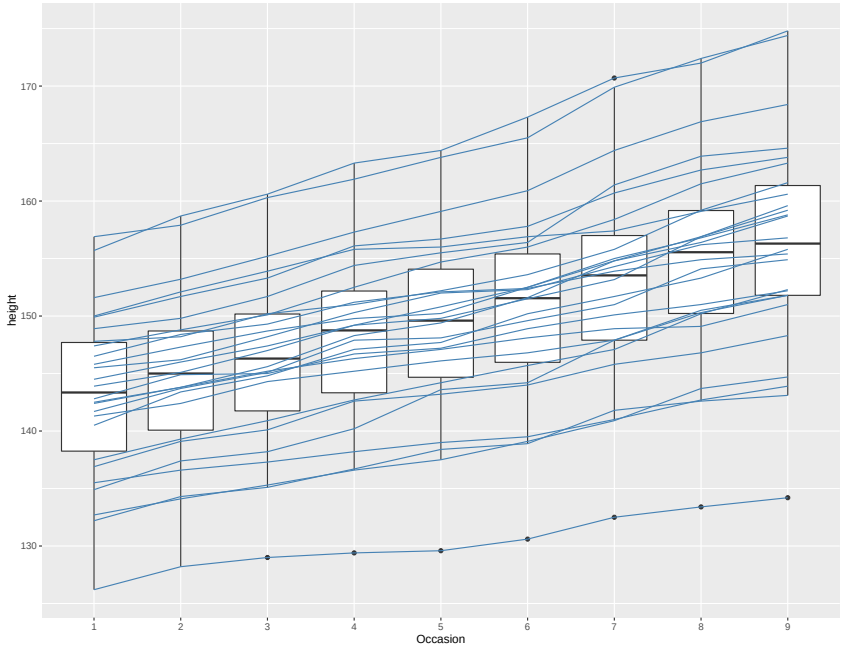
```
ggplot(Oxboys, aes(age, height, group = Subject)) + geom_line()
      + geom_smooth(group = 1)
```



Overriding Default Groupings

- We can also layer multiple groupings, in this case, the box's contain multiple boys, but the lines are one boy each

```
ggplot(Oxboys, aes(Occasion, height)) + geom_boxplot()
      + geom_line(aes(group = Subject), colour = "steelblue")
```



More on `geom`

- Depending on the `geom` you select, it will have different aesthetics it requires you provide
- For example `geom_point()` requires both `x` and `y` be provided
- On the other hand `geom_bar()` only requires a y

ggplot2 geom Options [Wickham, 2009]

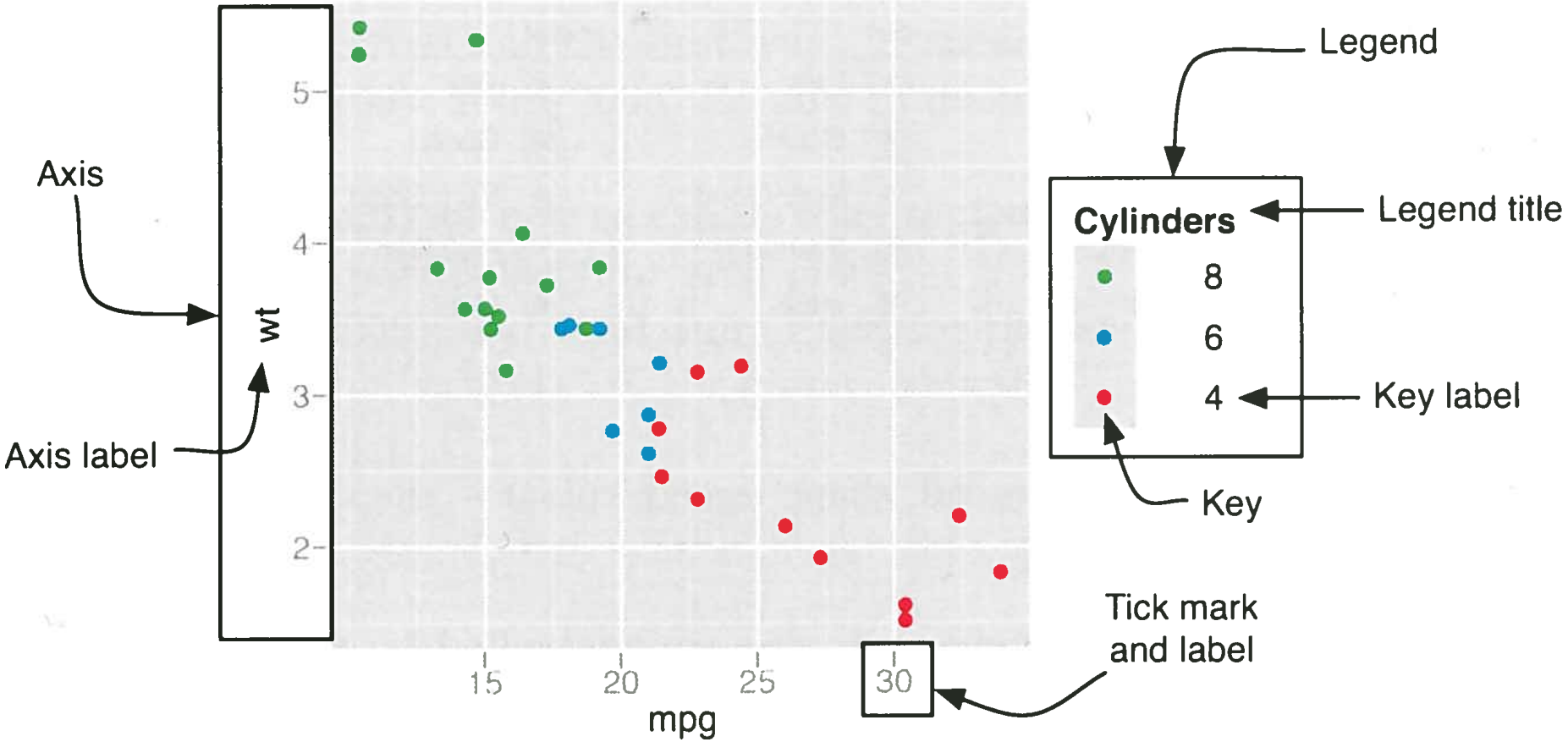
Name	Description
abline	Line, specified by slope and intercept
area	Area plots
bar	Bars, rectangles with bases on y-axis
blank	Blank, draws nothing
boxplot	Box-and-whisker plot
contour	Display contours of a 3d surface in 2d
crossbar	Hollow bar with middle indicated by horizontal line
density	Display a smooth density estimate
density_2d	Contours from a 2d density estimate
errorbar	Error bars
histogram	Histogram
hline	Line, horizontal
interval	Base for all interval (range) geoms
jitter	Points, jittered to reduce overplotting
line	Connect observations, in order of x value
linerange	An interval represented by a vertical line
path	Connect observations, in original order
point	Points, as for a scatterplot
pointrange	An interval represented by a vertical line, with a point in the middle
polygon	Polygon, a filled path
quantile	Add quantile lines from a quantile regression
ribbon	Ribbons, y range with continuous x values
rug	Marginal rug plots
segment	Single line segments
smooth	Add a smoothed condition mean
step	Connect observations by stairs
text	Textual annotations
tile	Tile plot as densely as possible, assuming that every tile is the same size
vline	Line, vertical

Position Adjustments

- Position adjustments apply minor tweaks to the position of elements within a layer
- Position can be changed with the argument `position=`

Adjustment	Description
dodge	Adjust position by dodging overlaps to the side
fill	Stack overlapping objects and standardise have equal height
identity	Don't adjust position
jitter	Jitter points to avoid overplotting
stack	Stack overlapping objects on top of one another

Components of the Axes and Legend



Axes

- `ggplot2` will automatically set x and y axes labels and legend
- You may wish to adjust fontsize by adding

  `+ theme_grey(base_size = )`
- It is important to make your font size readable!
- You can set custom axes labels by adding `+ xlab("Some Label")` and `+ ylab("Some Label")`
- We can also limit how long our axes are using `xlim()` and `ylim()`

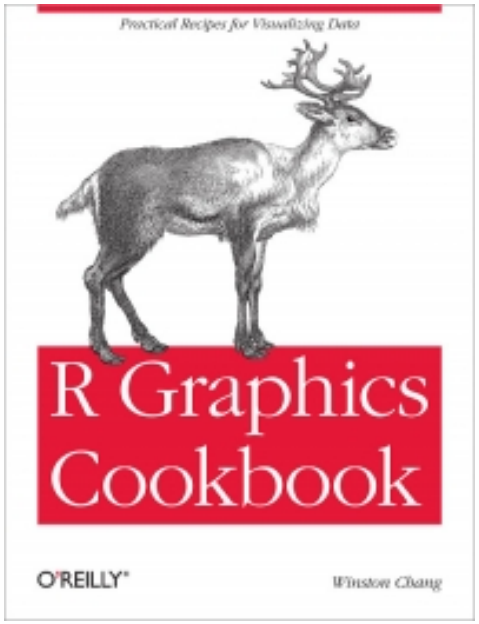
Plot Title

- You can also add a title to your plot with `ggtitle("Some Title")`
- I like to center my title by adding `+ theme(plot.title = element_text(hjust = 0.5))`
- You can also change size, color, etc by adding additional arguments to `element_text`

Themes and Colors

- You can change the overall look of your graph by adding a theme (e.g., `+ theme_bw()`)
- You can pick a new color palette for your filled objects (e.g., bars) using `+ scale_fill_brewer(palette = "Some Palette")`
- You can pick a new color palette for your lines and points using `+ scale_colour_brewer(palette = "Some Palette")`

For More Information about Plotting



- https://www.r-graph-gallery.com/
- http://www.cookbook-r.com/Graphs/

End of Class

- No class or office hours on October 16! (Fall Break)
- Begin work on the `ggplot` lab located on the github site https://github.com/abbiepopa/bsds100
- You do not need to work on the lab at home, you will have time to work on it in class on October 18 before submitting