Lecture 7: Plotting in R

BSDS 100 - Intro to Data Science with R

Exploration and Visualization

- 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
- One of the more creative and artistic aspects of DS.

A few important plots to consider

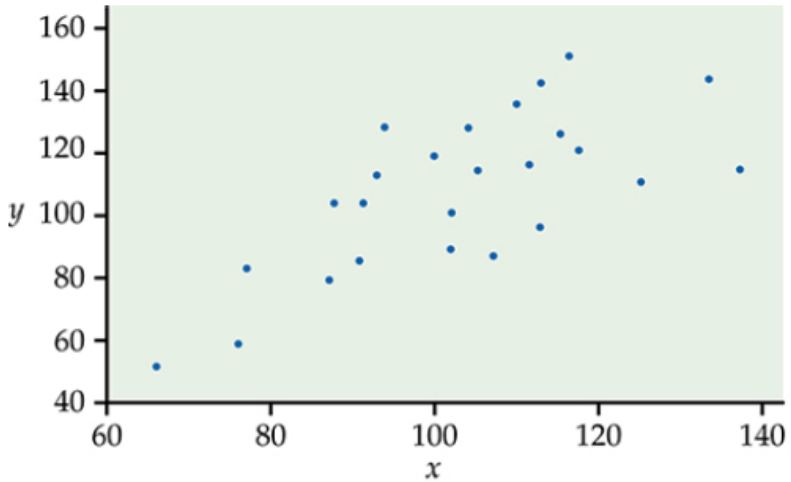
- Scatter plot: a graph in which the values of two variables are plotted along two axes against one another. These are used to explore whether or not there is correlation present between quantitative variables.
- Bar chart: a graph that presents grouped data with rectangular bars with lengths proportional to the values that they represent. Bars can be plotted vertically or horizontally.
- Histogram: a diagram consisting of rectangles whose area is proportional to the frequency of a variable whose width is equal to the class interval.

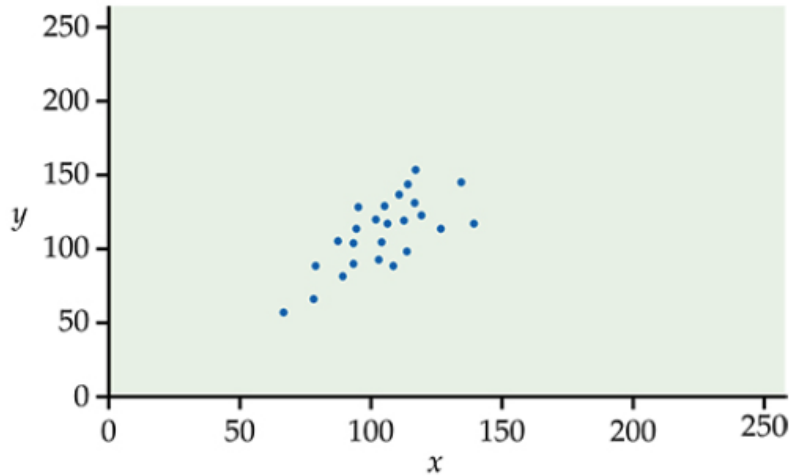
A few important plots to consider

- Density curve: a “smoothed” version of a histogram
- Box plot: plot that displays the five number summary of a variable: the minimum, first quartile, median, third quartile, and maximum
- 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.
  Generally I reccomend not to use these!! People cannot distinguish areas of a circle very well. 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!





Graphics in Base R

- Base R graphics package:
  - quick to code and functional
  - not particularly aesthetically pleasing
  - cumbersome
  - outshone by `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

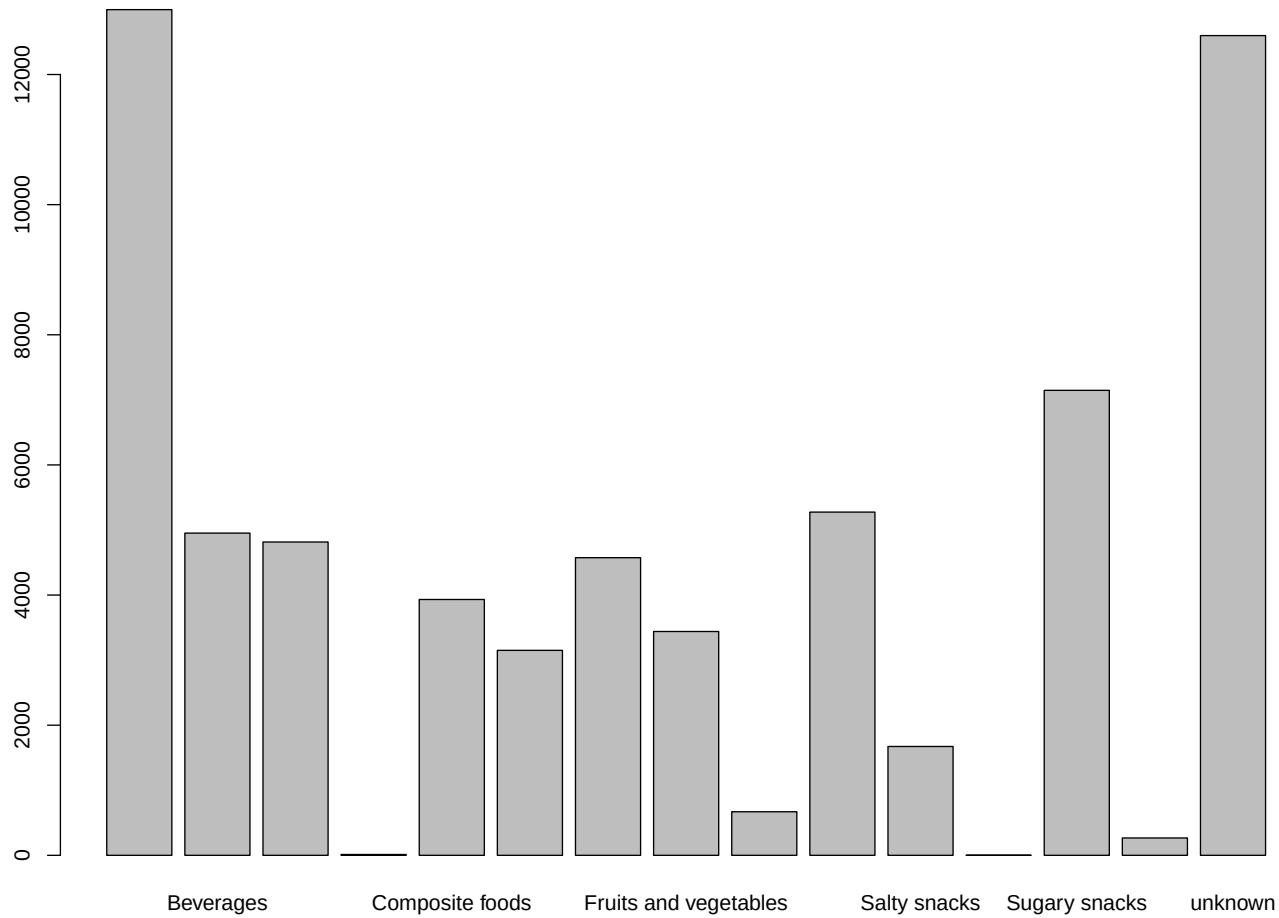
- Scatterplots and bar charts: `plot()`
- Look at documentation for learning how exactly to plot what you want.
- There are *a lot* of arguments that can be provided to make detailed plots, including changing the x- and y- axes, tick marks, labels, the size, color, and type of lines in each plot, etc.
- 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 in R, but we will at least need one other dataset, *Food.csv* from out github.

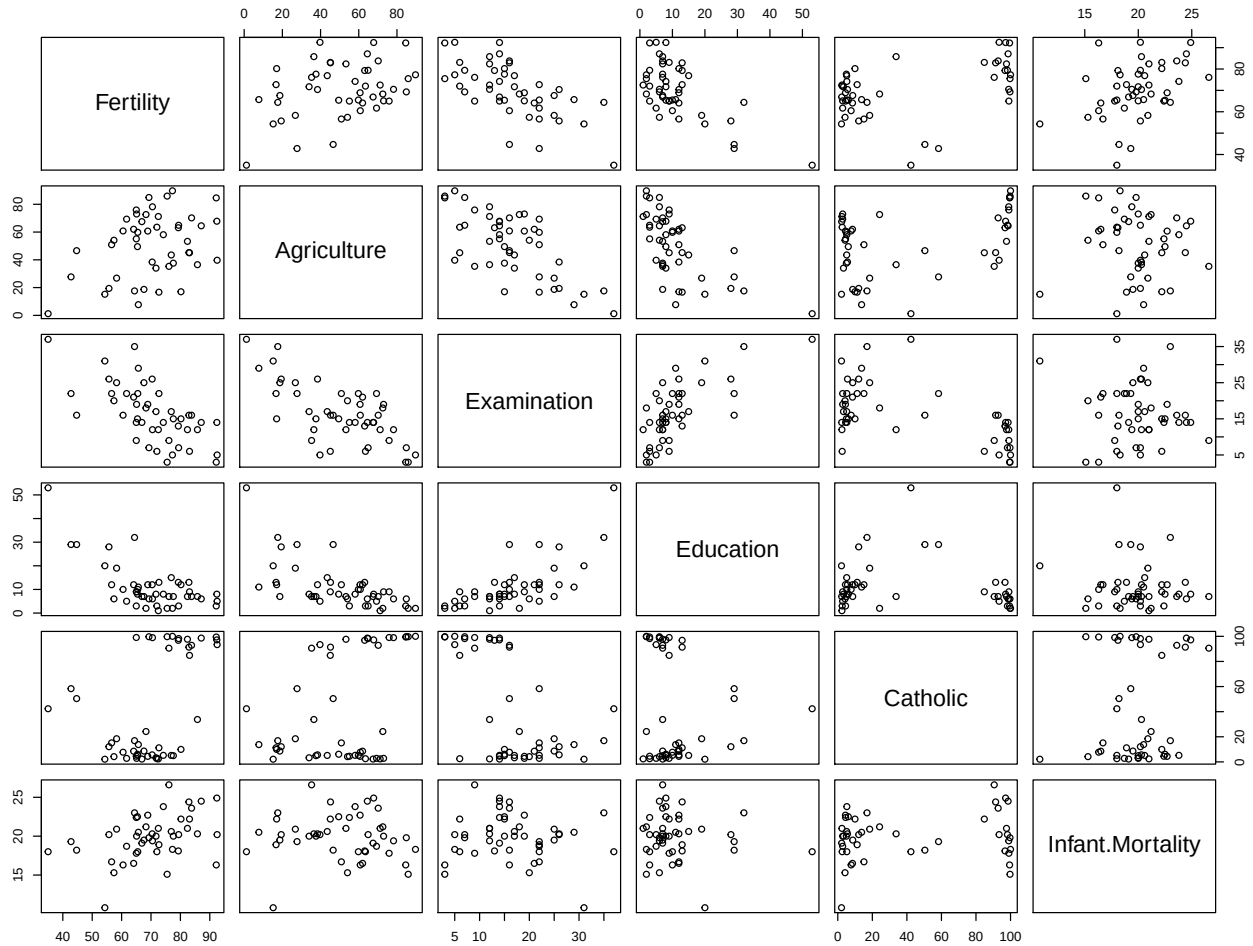
`plot()` a bar chart for categorical (factor) variables

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



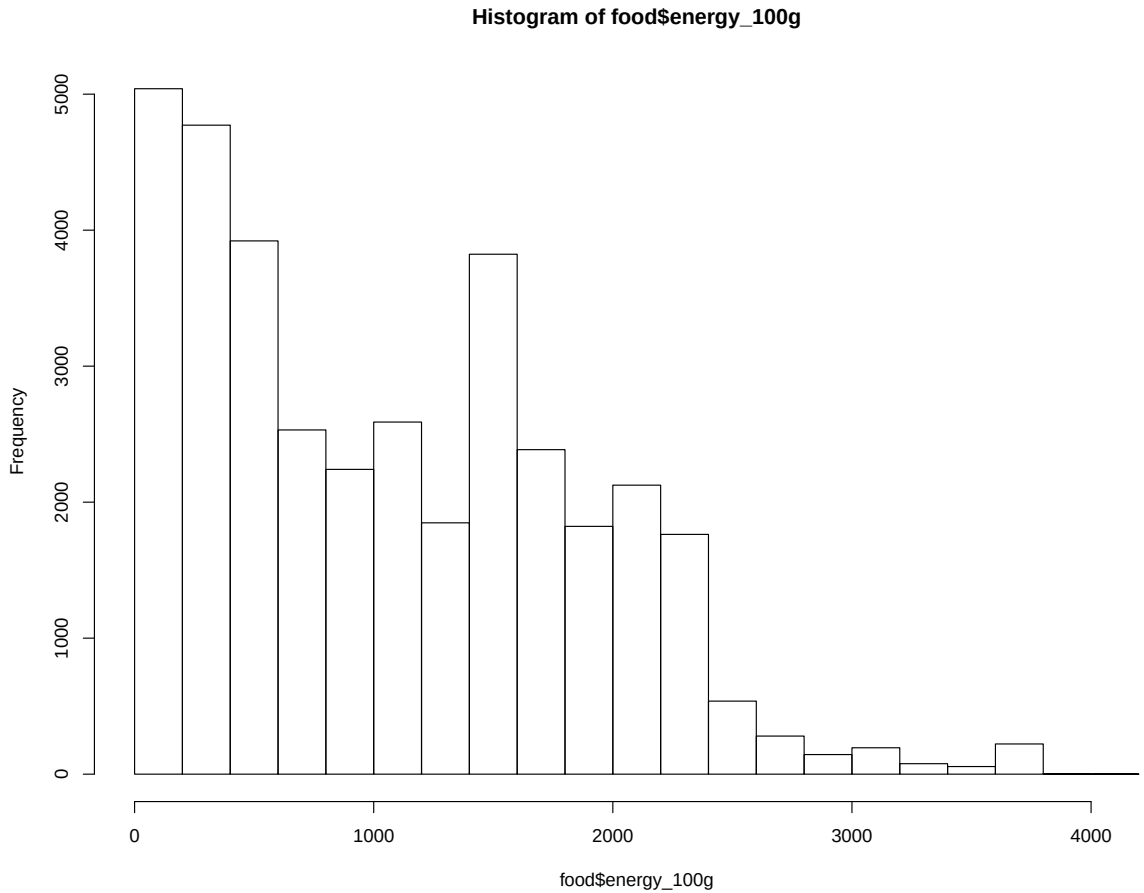
`plot()` a pairwise scatter plot matrix for data frame

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



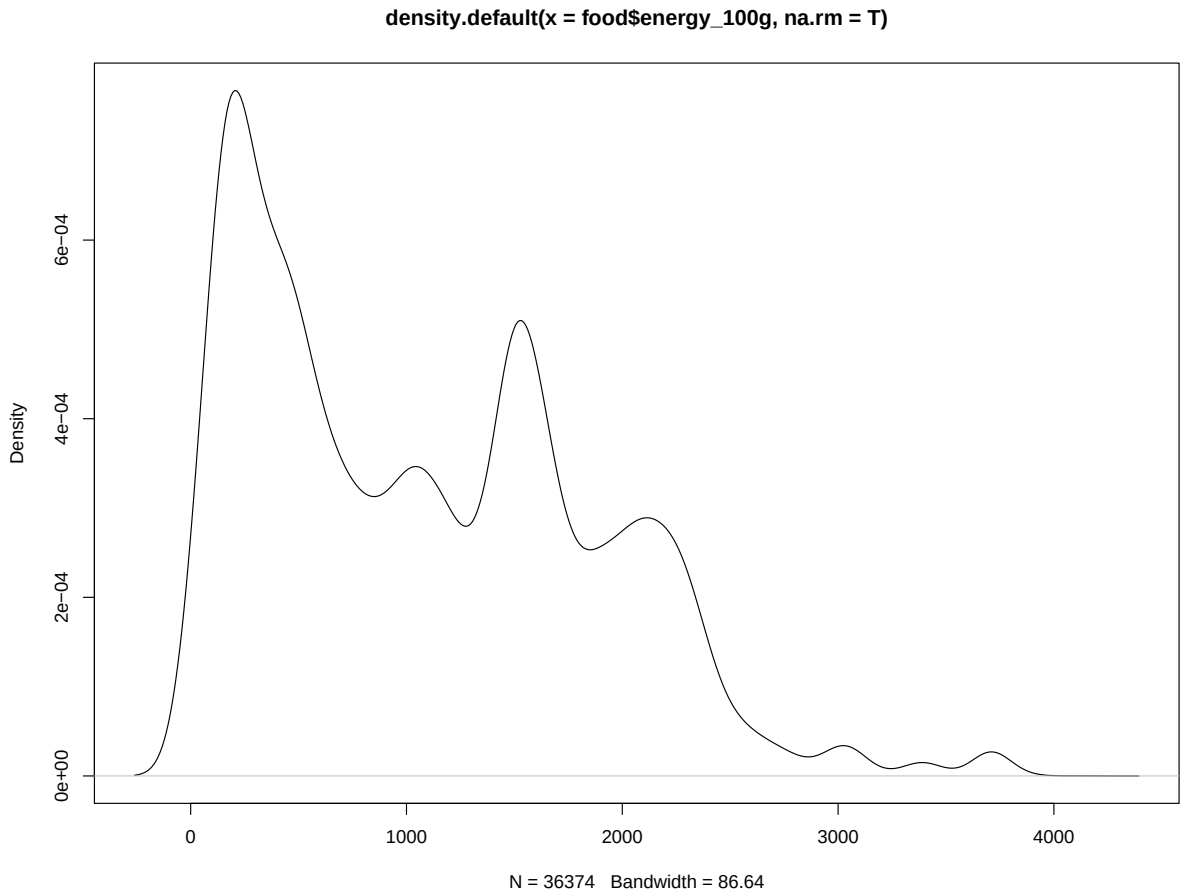
Make a histogram with `hist()` for continuous (numeric) variables

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



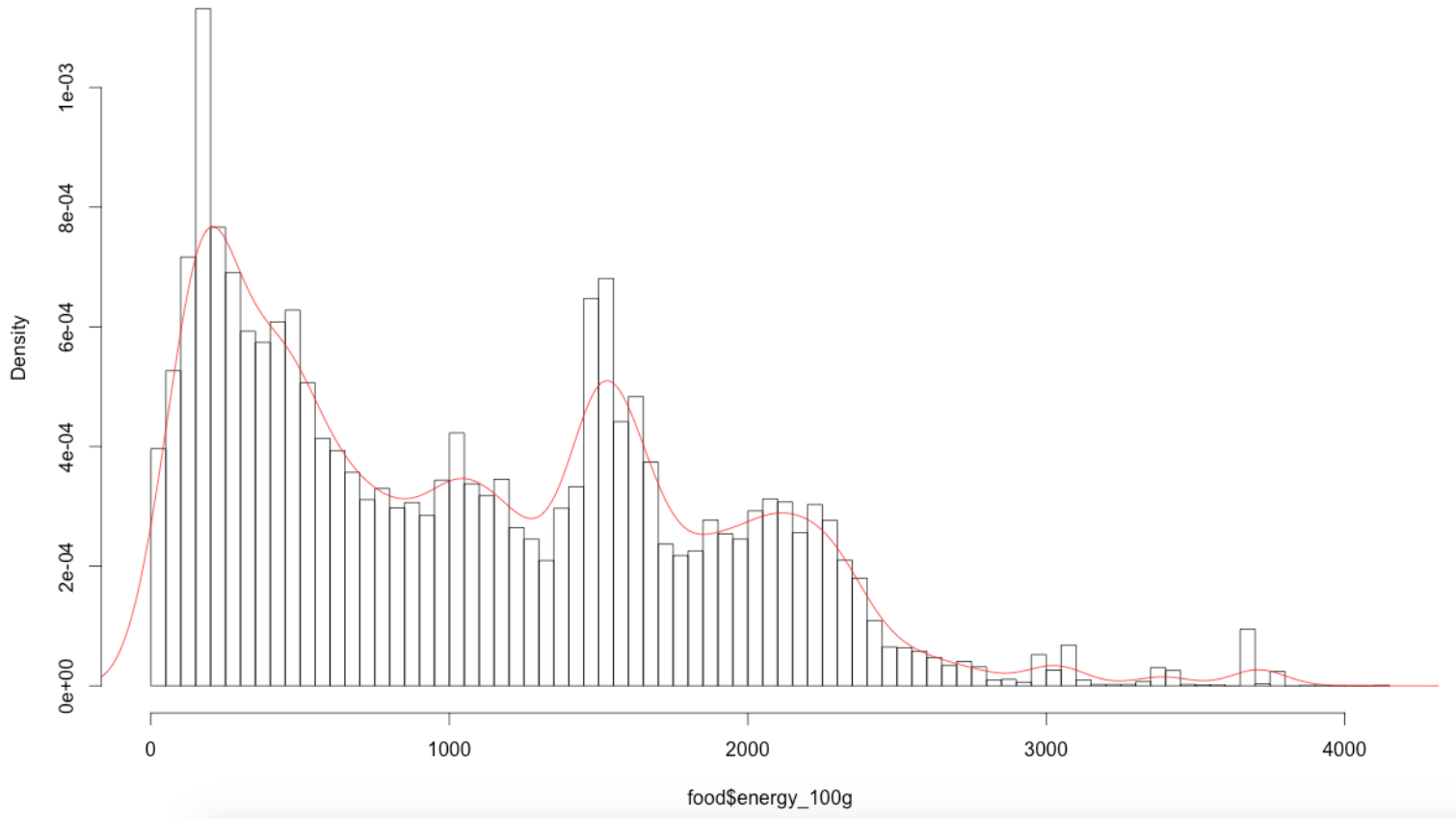
`plot()` the kernel density for numeric variables

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



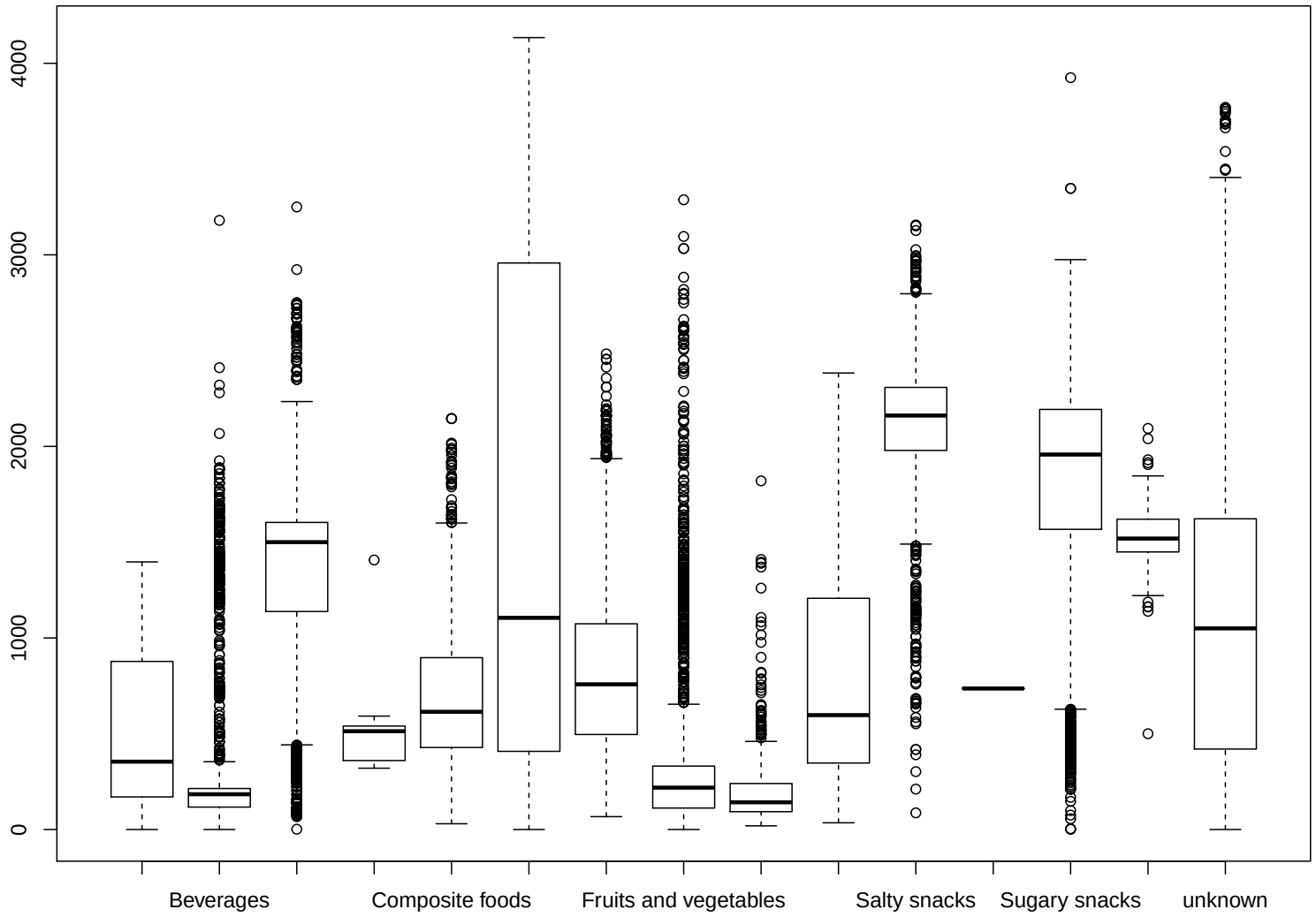
Overlay a kernel density on a probability histogram

```
#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


```
> with(food, boxplot(energy_100g ~ pnns_groups_1))
```

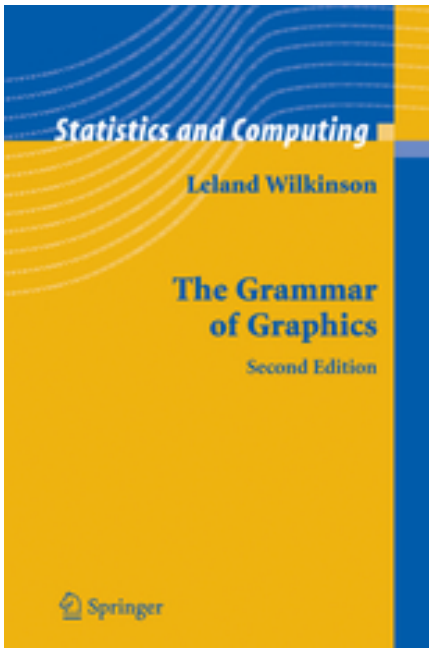


Useful arguments for `plot()` - esque functions

- 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`.

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

Example plot

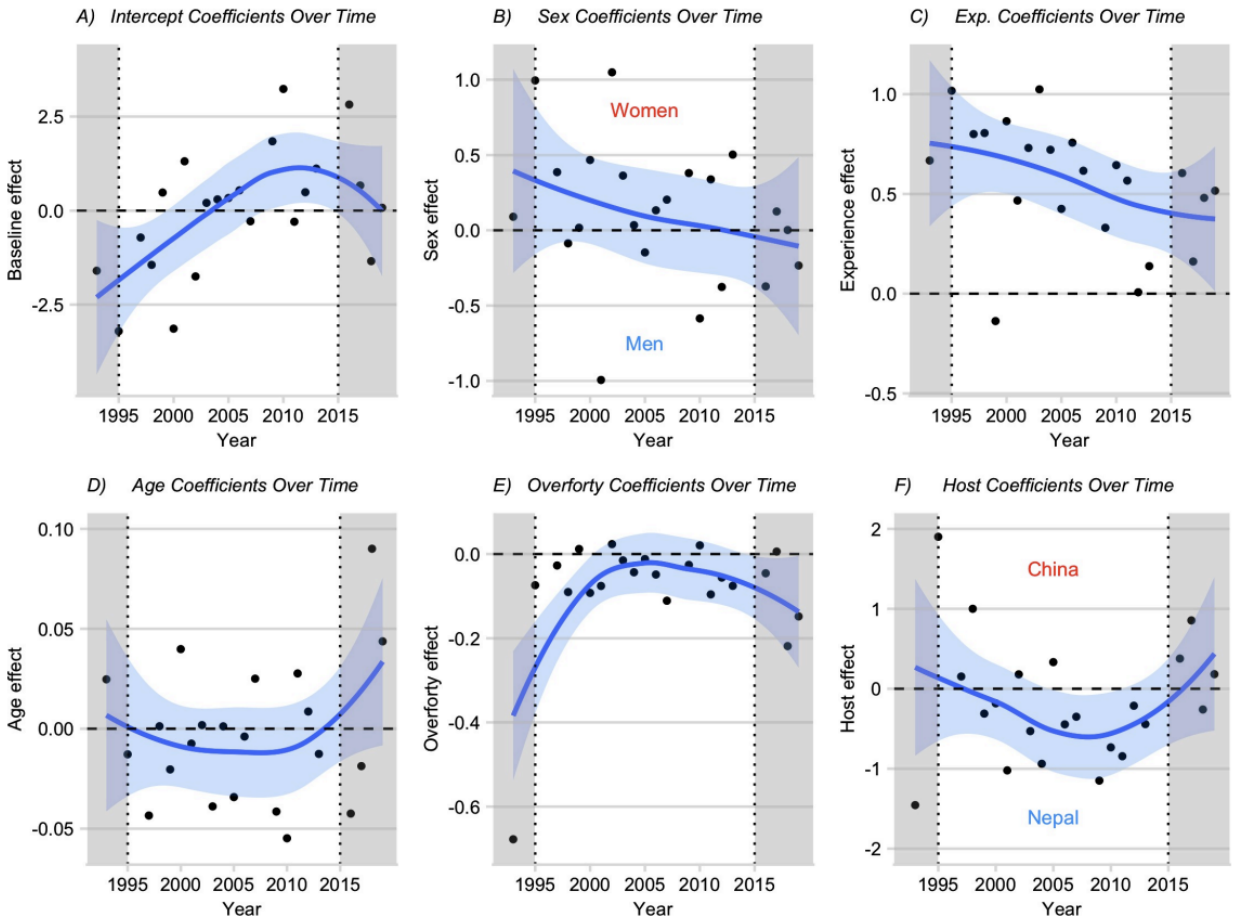


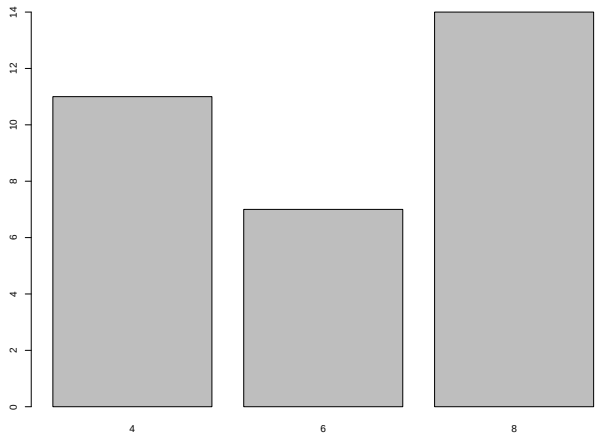
Fig 3. Effects of covariates (DCM) on summiting rates over time. Blue lines are obtained via local linear smoothing. Trends are reliable only in the central region (1995–2015, see Methods). The blue shaded areas represent 95% pointwise confidence regions for the mean effects as a function of year. For factor covariates (e.g., sex), a point on the "Women" side of the horizontal line indicates that women were more likely than men to summit in that year, and vice versa for a point on the "Men" side. Grayed zones indicate years which are unreliable because of boundary effects (**Methods**).

https://doi.org/10.1371/journal.pone.0236919.g003

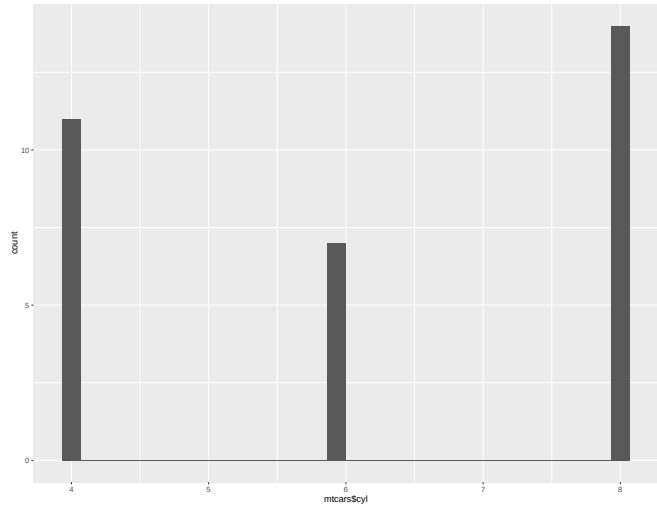
qplot() versus plot()

- 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)
```

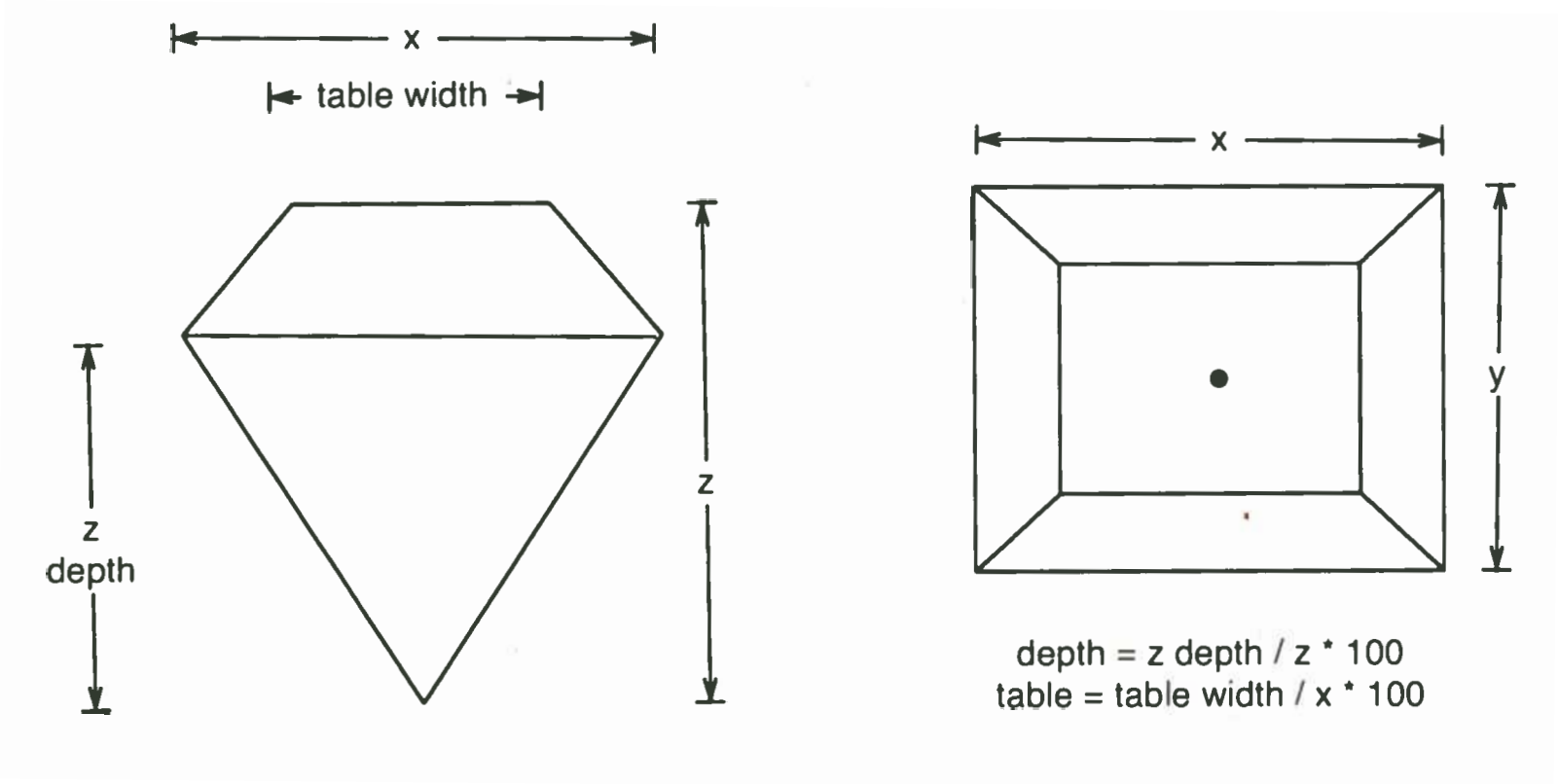


Case Study: The `diamonds` dataset

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

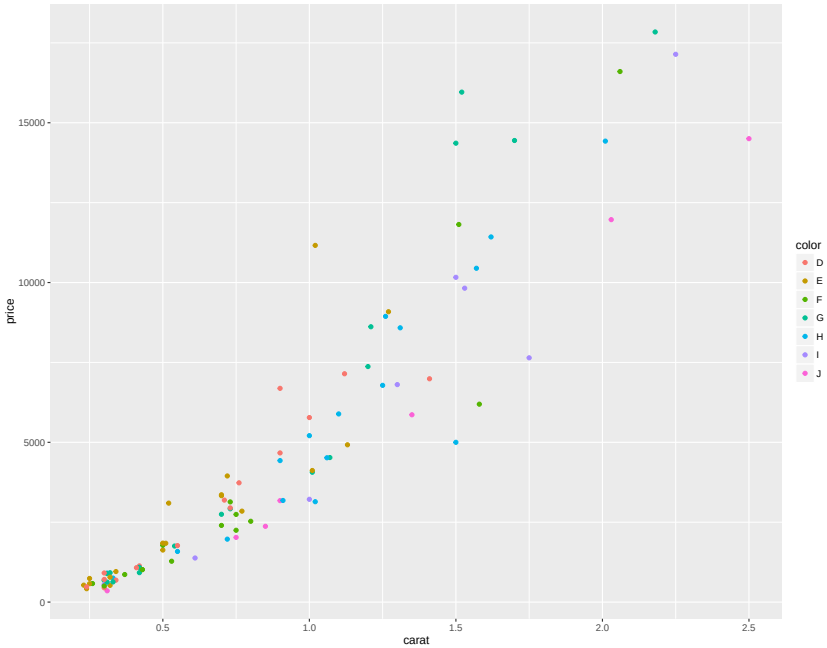


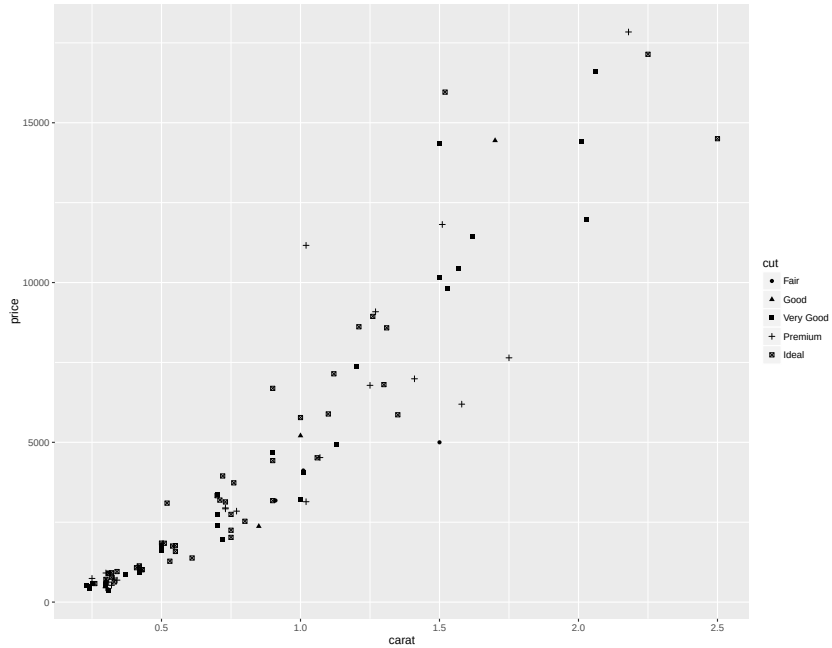
Aesthetic Attributes

- The first major difference when using `qplot()` instead of base graphing in `R` is the when assigning 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
- Other common aesthetics include `alpha` to control for opacity

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)
```



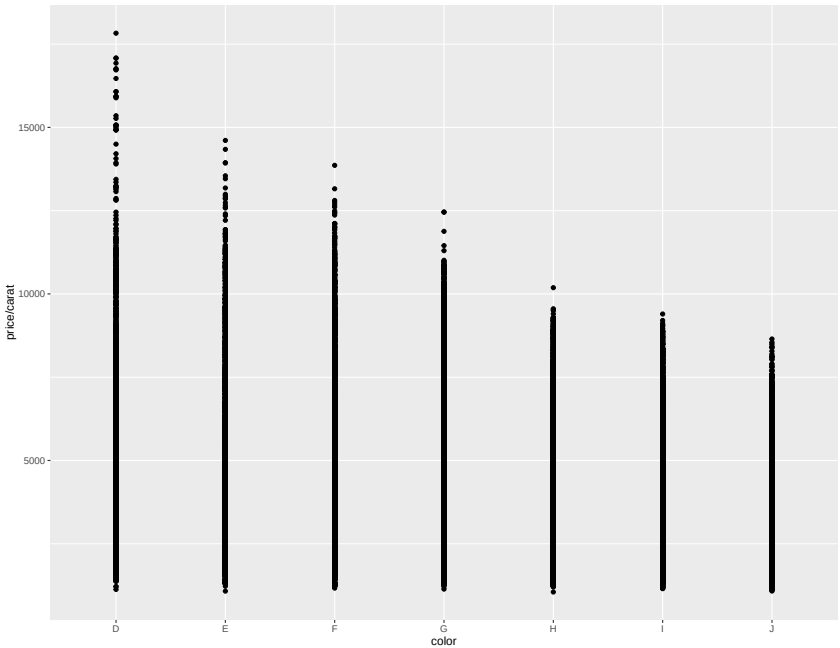


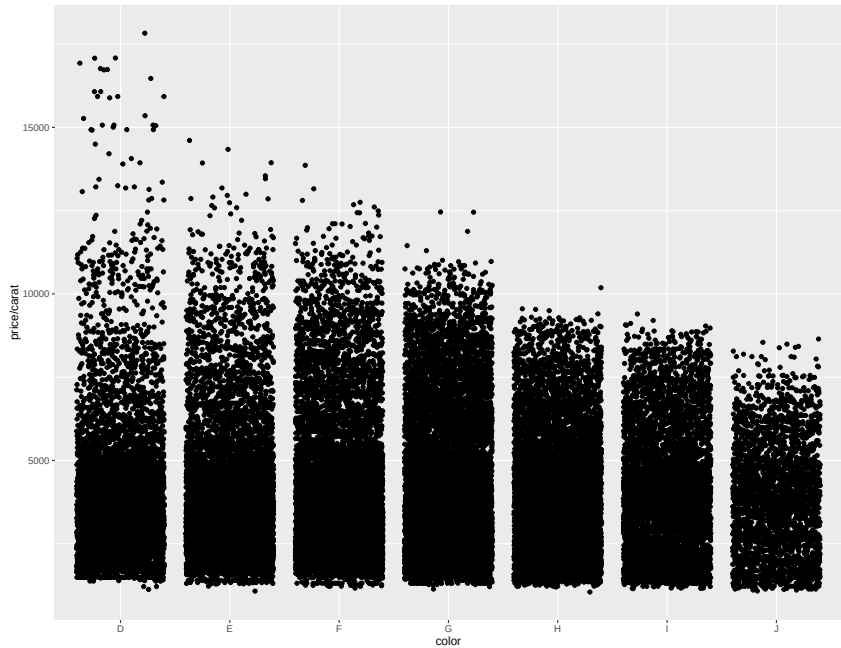
Geometric Objects

- Geometric objects, or `geom` 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 = "path"` (line in any direction)
  4. `geom = "line"` (line from left to right)
  5. `geom = "smooth"` (smoothed line with standard error)

`geom = "jitter"`

- 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



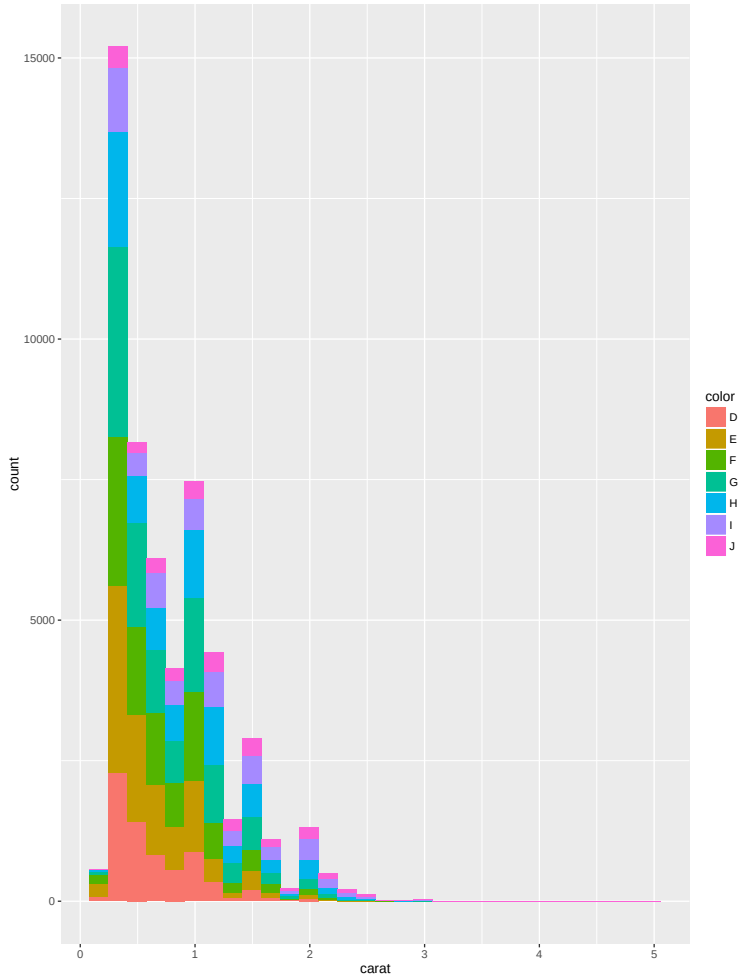


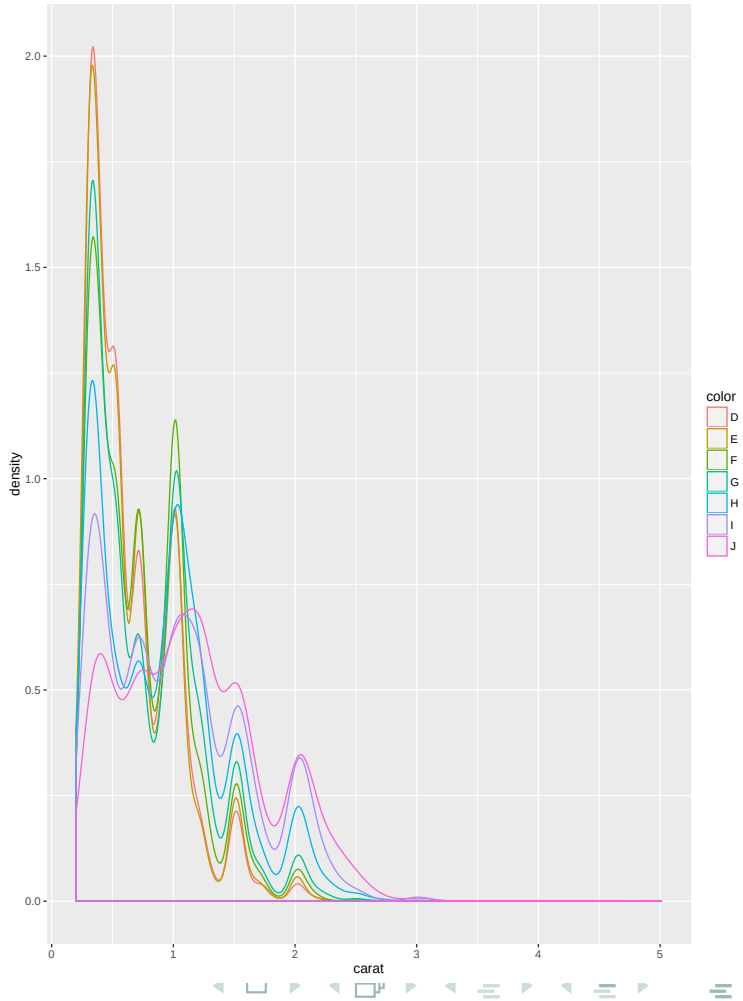
`geom = "histogram"` & `geom = "density"`

- With histograms, play around with the `binwidth` argument to select the bin size 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 = "histogram" & geom = "density"

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

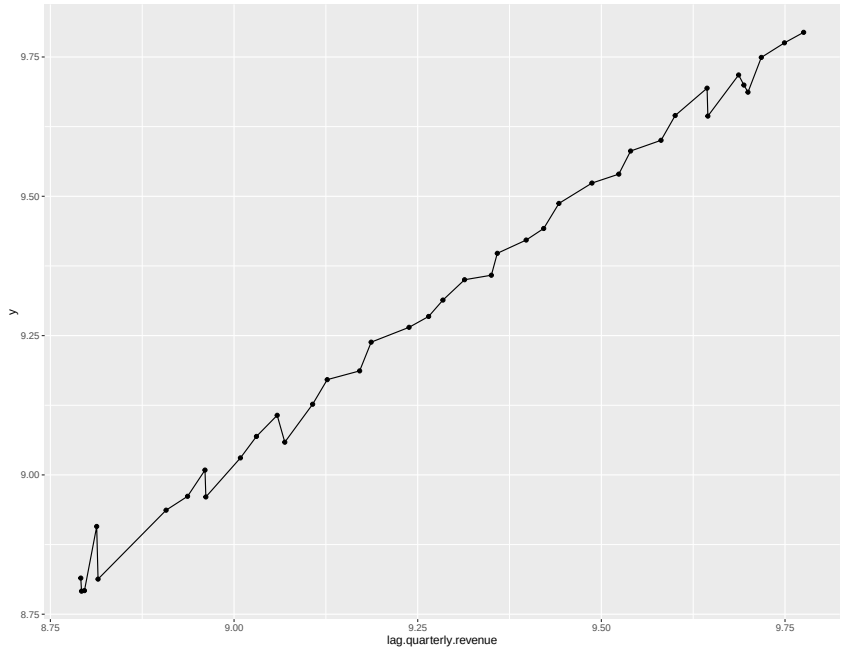




Using Multiple `geom`

- 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"))
```

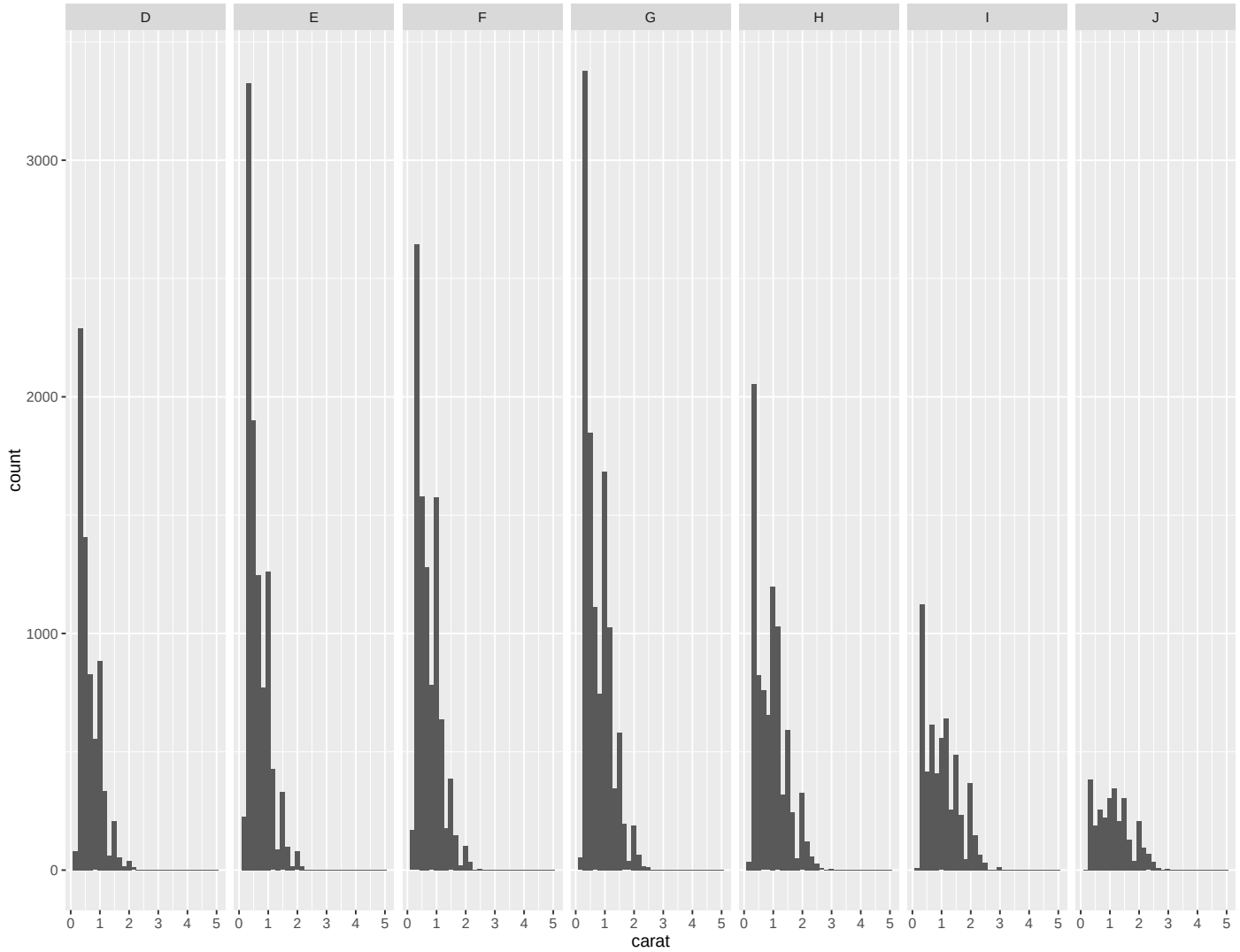


Faceting

- 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")
```



`qplot()` versus `ggplot()`

- Plots can be created in two ways:
  1. all at one 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

- 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

> qplot(bodywt, brainwt, data = msleep, log= "xy") + 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**
- Do not reference data that is not passed to `ggplot()` as the default data frame, as this makes it impossible to encapsulate all of the data needed for plotting in a single object

```
> ggplot(mtcars, aes(x = cyl, y = diamonds$cut[1:32]))
+ geom_point()
```

- You could make it work (as the above does), but it defeats the purpose, strength and flexibility of `ggplot()`

aes()

- 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 = weight, y = height, colour = age)` maps `weight` to `x`, `height` to `y` and `colour` to `age`
- **Note**: Any variables in an `aes()` specification **must** be contained inside the plot or layer data

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())
```

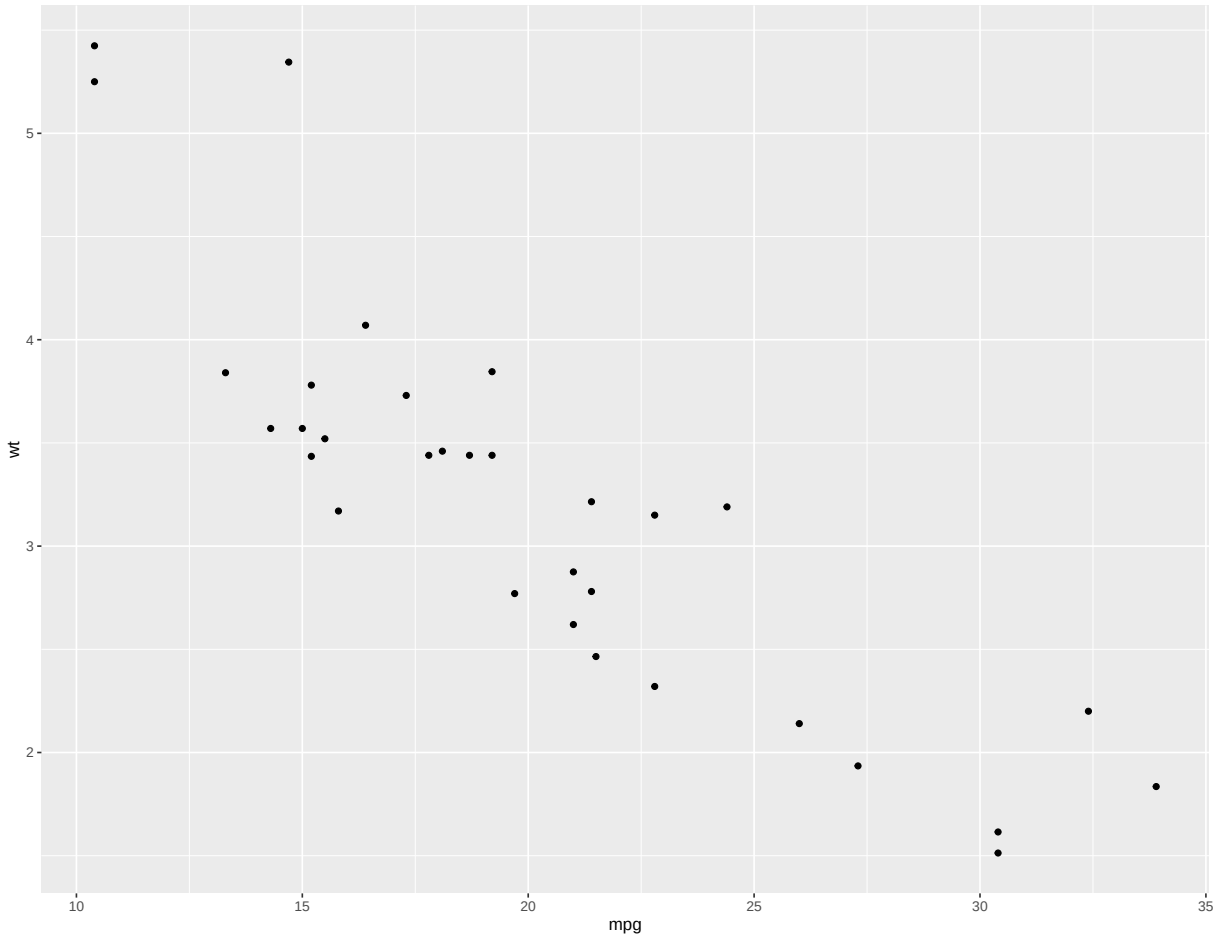
```
> myPlot + geom_point(aes(colour = factor(cyl)))
```

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

- Aesthetic mappings specified in a layer affect only that layer
- Axis labels and legend titles will be based on plot defaults (see examples in following slides)

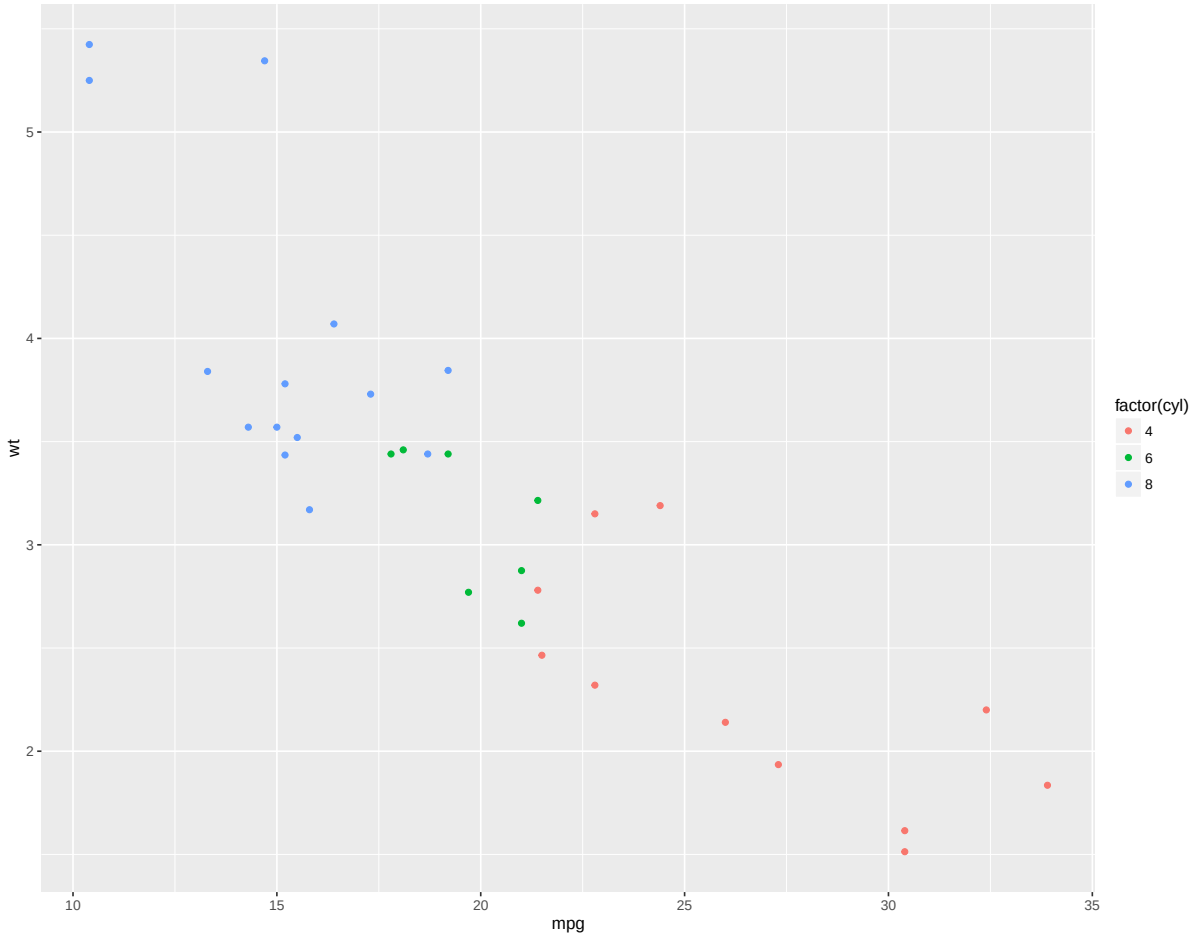
Updating Aesthetics on Layers

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



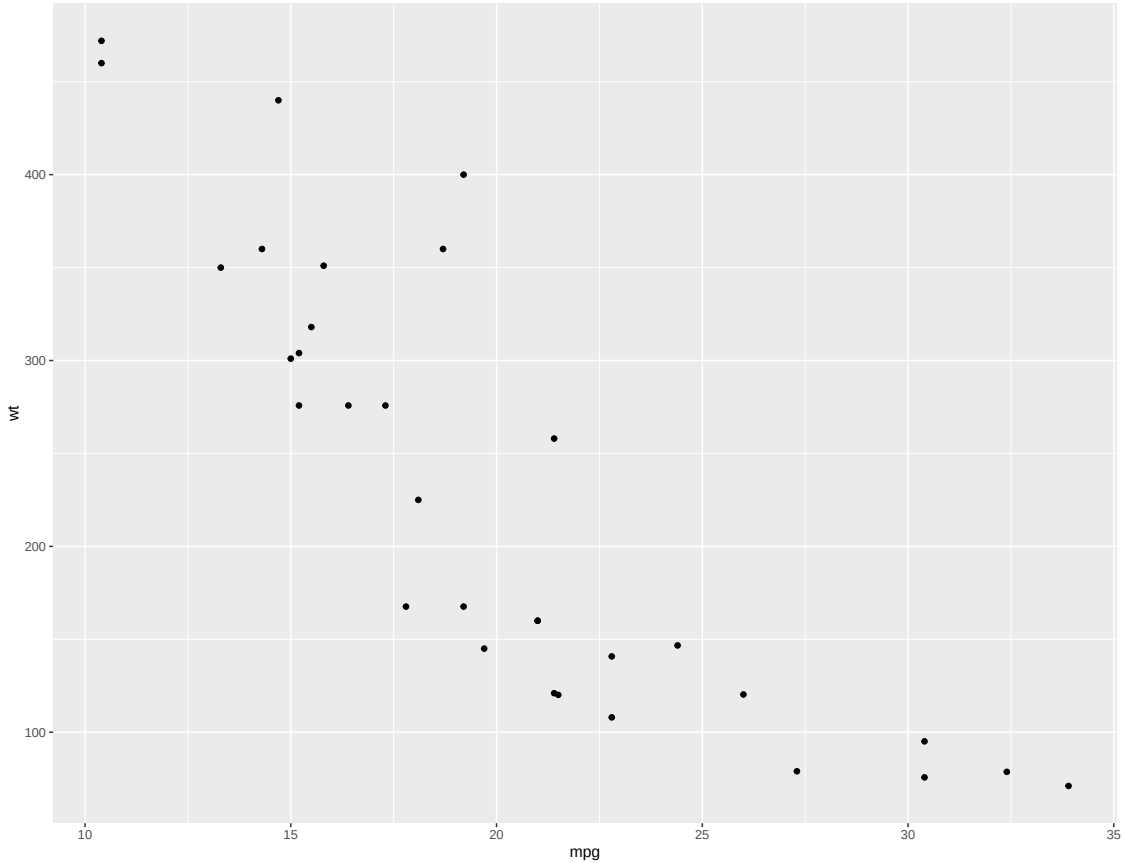
Updating Aesthetics on Layers

```
> 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., polygons
- The `group` aesthetic controls which observations go into which individual graphical element

Grouping

- By default, the `group` is set to the interaction of all discrete variables in the plot, which typically generates the expected results
- In the event it does not (or when no discrete variables are used in the plot), the `group` can be mapped to a variable that has a different value for each group
- `interaction()` is useful if a single pre-existing variable doesn't cleanly separate groups, but a combination does

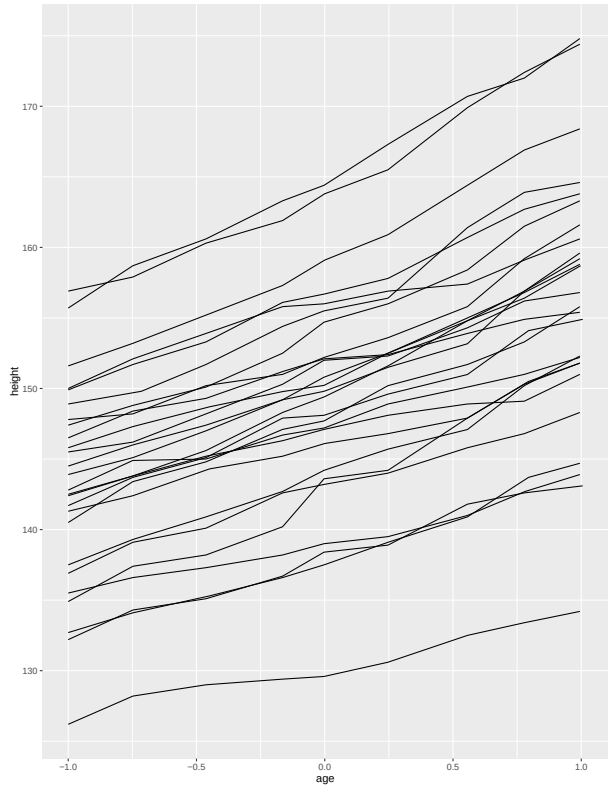
Grouping Example

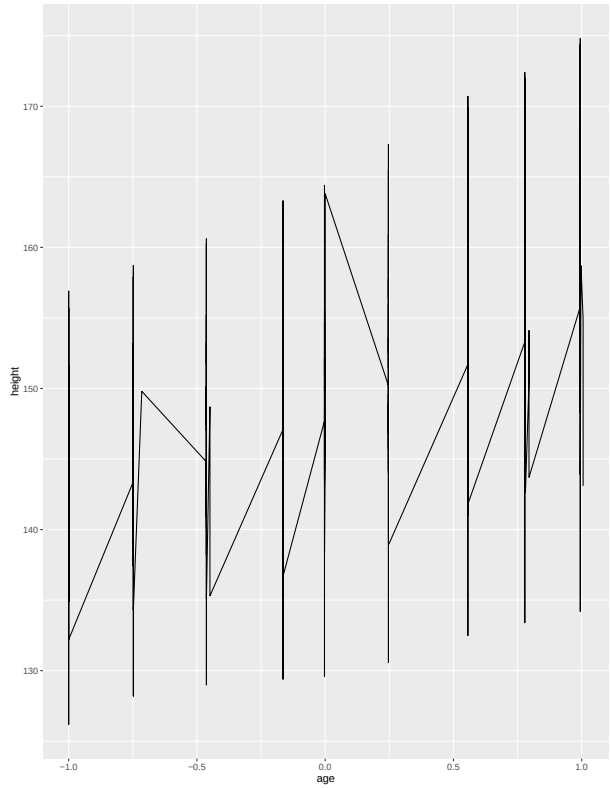
- Grouping examples on the following slides will use longitudinal data from `nmle` package called `Oxboys`, which records `height`, `age` and `subject` (26 boys) measured at nine `occasion`s
- **Objective**: separate the data into groups to distinguish between individual subjects, but render all of them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

Multiple Groups, One Aesthetic

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

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



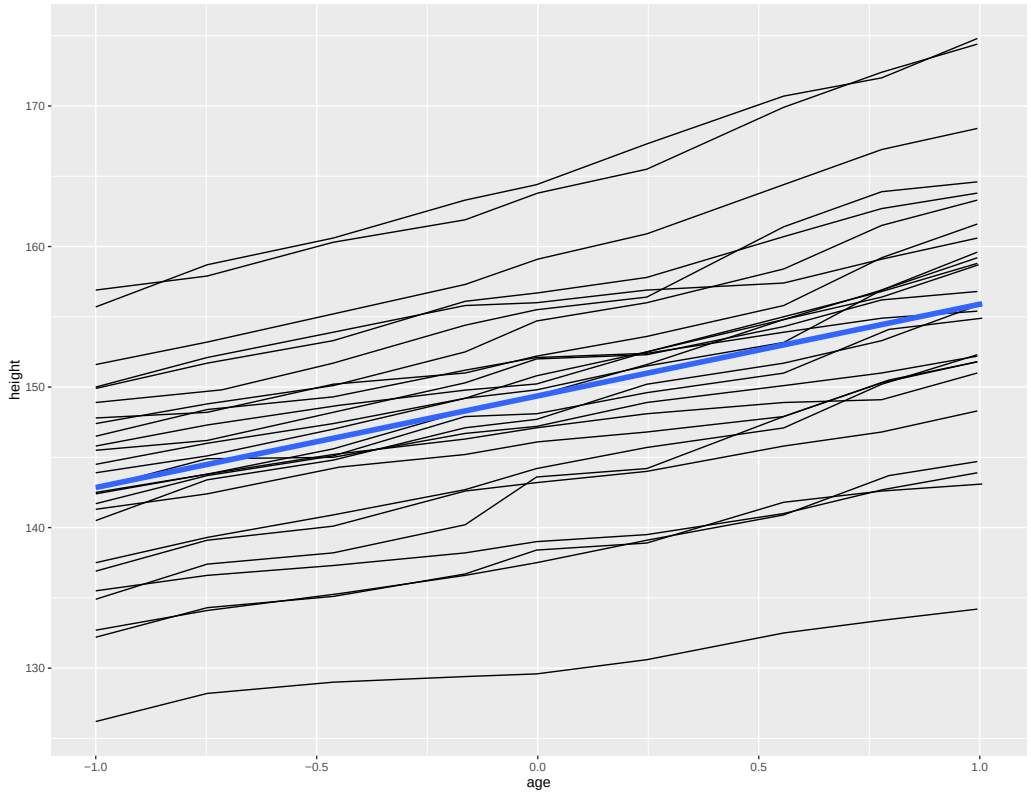


Different Groups on Different Layers

- **Objective**: plot data based on different layers of aggregation
- In this example, a second layer in create whereby a line of best-fit is overlayed using **all** data, hence `group = 1`

Different Groups on Different Layers

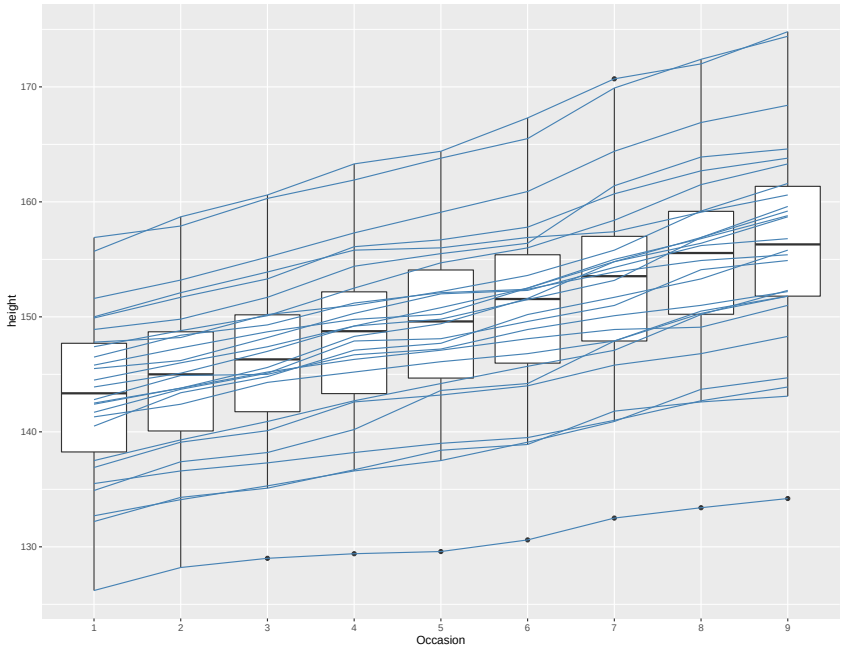

```
myPlot = ggplot(Oxboys, aes(age, height, group = Subject)) +
geom_line()
myPlot + geom_smooth(aes(group = 1), method = "lm", size = 2, se = F)
```



Overriding Default Groupings

- **Objective**: a plot has a discrete scale, but there is a desire to draw lines that connect across groups (think interaction plots)

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



More on `geom`

- `geom` performs the actual rendering of the layer
- Each `geom` has a set of aesthetics it *can* determine without direction, and a set that are **required** for drawing:
  - `geom_point()` **requires** both an `x` and `y` coordinate, but will automatically process color, size and shape aesthetics
  - `geom_bar()` **requires** a height `ymax`, but will automatically process width, border color and fill color
- Every `geom` has a default statistic and vice versa

E.g. The `bin` statistic automatically defaults to using the bar `geom` to create a histogram

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

stats in ggplot2 [Wickham, 2009]

Name	Description
bin	Bin data
boxplot	Calculate components of box-and-whisker plot
contour	Contours of 3d data
density	Density estimation, 1d
density_2d	Density estimation, 2d
function	Superimpose a function
identity	Don't transform data
qq	Calculation for quantile-quantile plot
quantile	Continuous quantiles
smooth	Add a smoother
spoke	Convert angle and radius to xend and yend
step	Create stair steps
sum	Sum unique values. Useful for overplotting on scatterplots
summary	Summarise y values at every unique x
unique	Remove duplicates

Position Adjustments

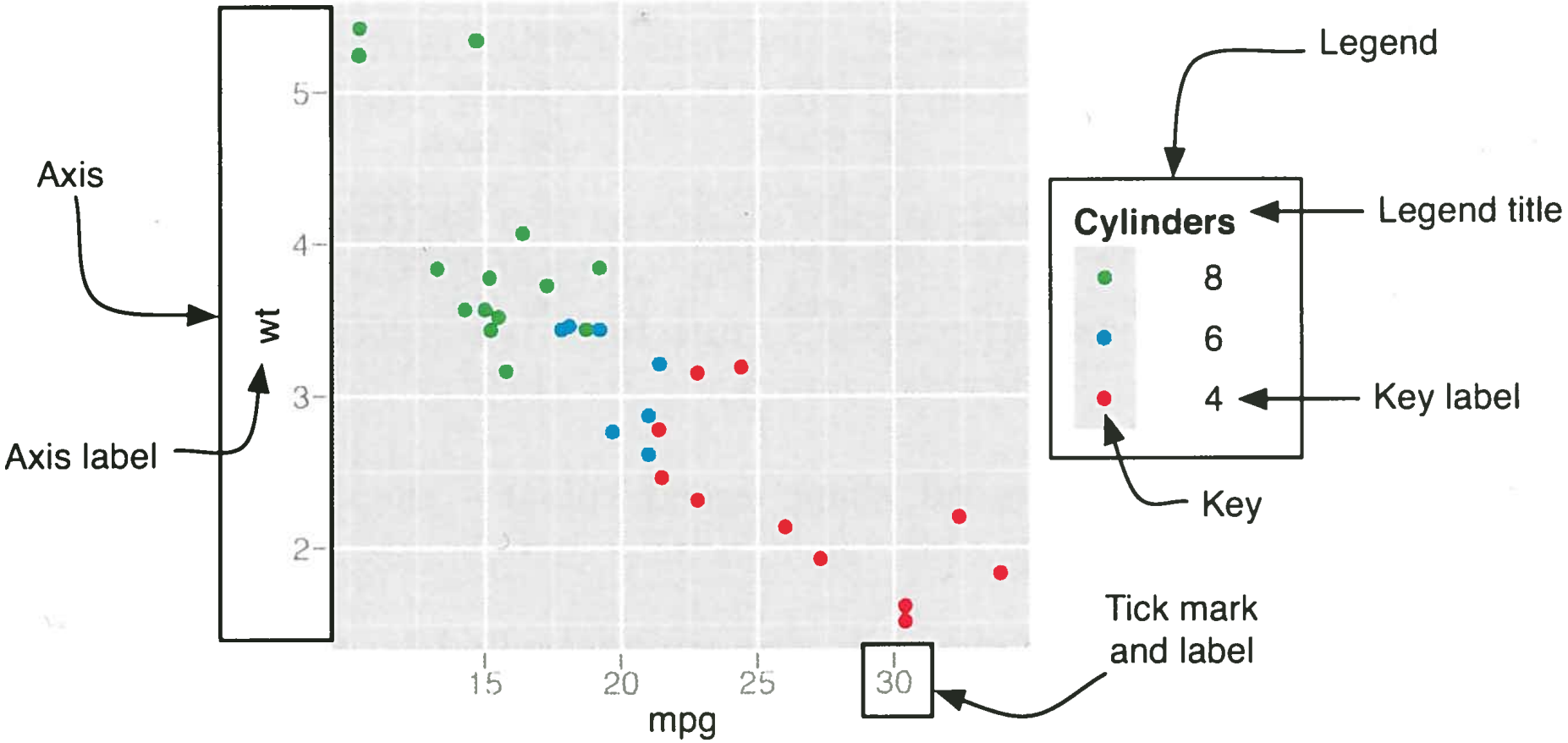
- Position adjustments apply minor tweaks to the position of elements within a layer
- Position adjustments are usually used with discrete data
- Position can be changed with the argument `position=`

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

Annotating Plots

- `ggplot2` makes it very easy to annotate a plot with
  1. Vertical bars
  2. Horizontal bars
  3. Shaded regions
  4. Text
  5. Mathematical symbols and equations
  6. Arrows

Components of the Axes and Legend



Axes

- Be sure to set a font size that is **legible** on your graphs, particularly when dealing with axis labels and axis ticks
- When dealing with large orders of magnitude on an axis, either reduce the order of magnitude manually **or** use the `scales` package and include commas
  1. If you have $15,000,000,000 as an axis tick, it might be more succinct and digestible for the reader to simply have $15, and in the axis title include a parenthetical reference ($B)
  2. If you have $2500000 on an axis tick, you force the reader to parse the text to figure out the order of magnitude; in lieu add commas using the `scales` package, resulting in $2,500,000

For More Information about Plotting