Section 5

Exploratory Data Analysis in R

summary()

- summary() is a generic function used to produce result summaries of the results of various model fitting functions; the function invokes particular methods which depend on the class of the first argument

```
> summary(mtcars[1:3])
      mpg             cyl              disp
 Min.   :10.40   Min.   :4.000   Min.   : 71.1
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8
 Median :19.20   Median :6.000   Median :196.3
 Mean   :20.09   Mean   :6.188   Mean   :230.7
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0
 Max.   :33.90   Max.   :8.000   Max.   :472.0
```

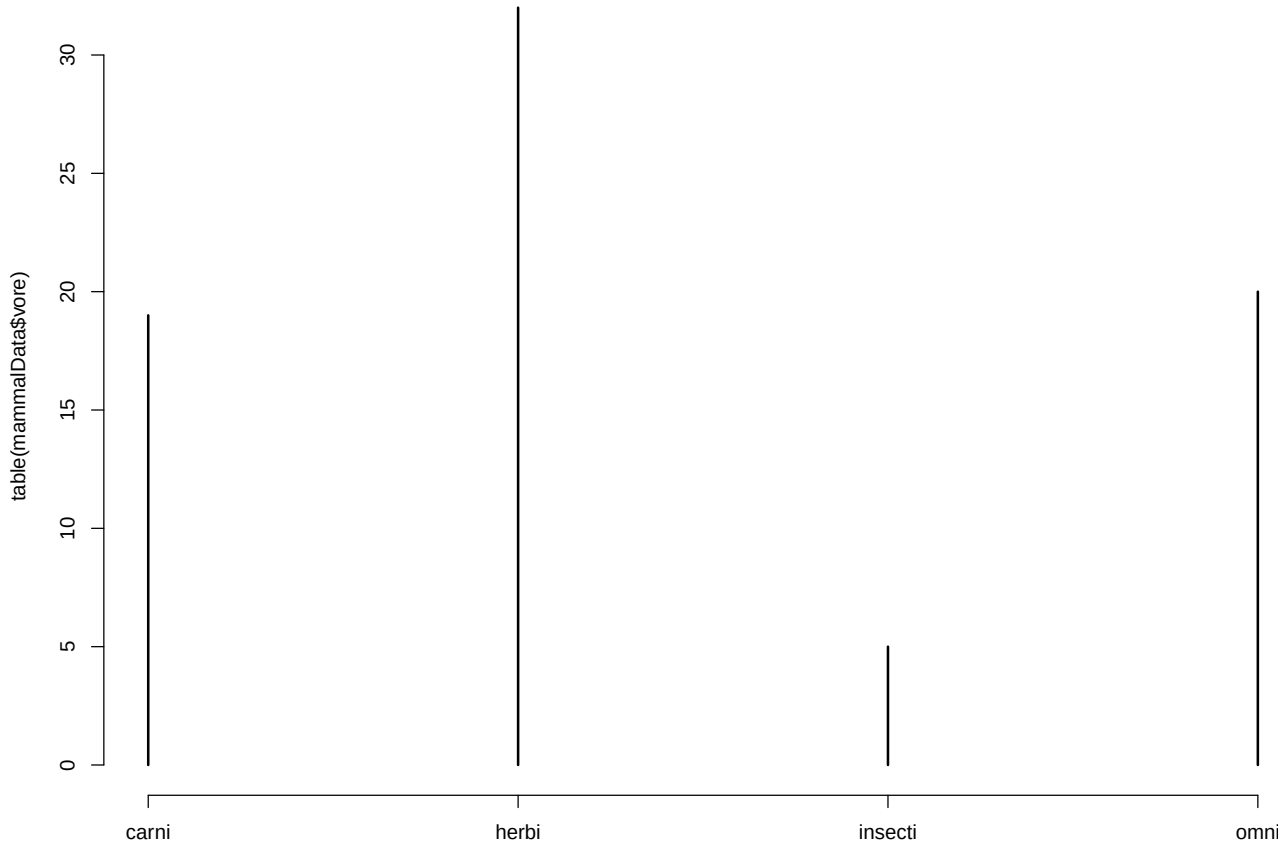
- summary() of an lm object will be different

Graphics in Base R

- The base R graphics package, while being quick to code and functional, is not particularly aesthetically pleasing, is syntactically cumbersome, and is far 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

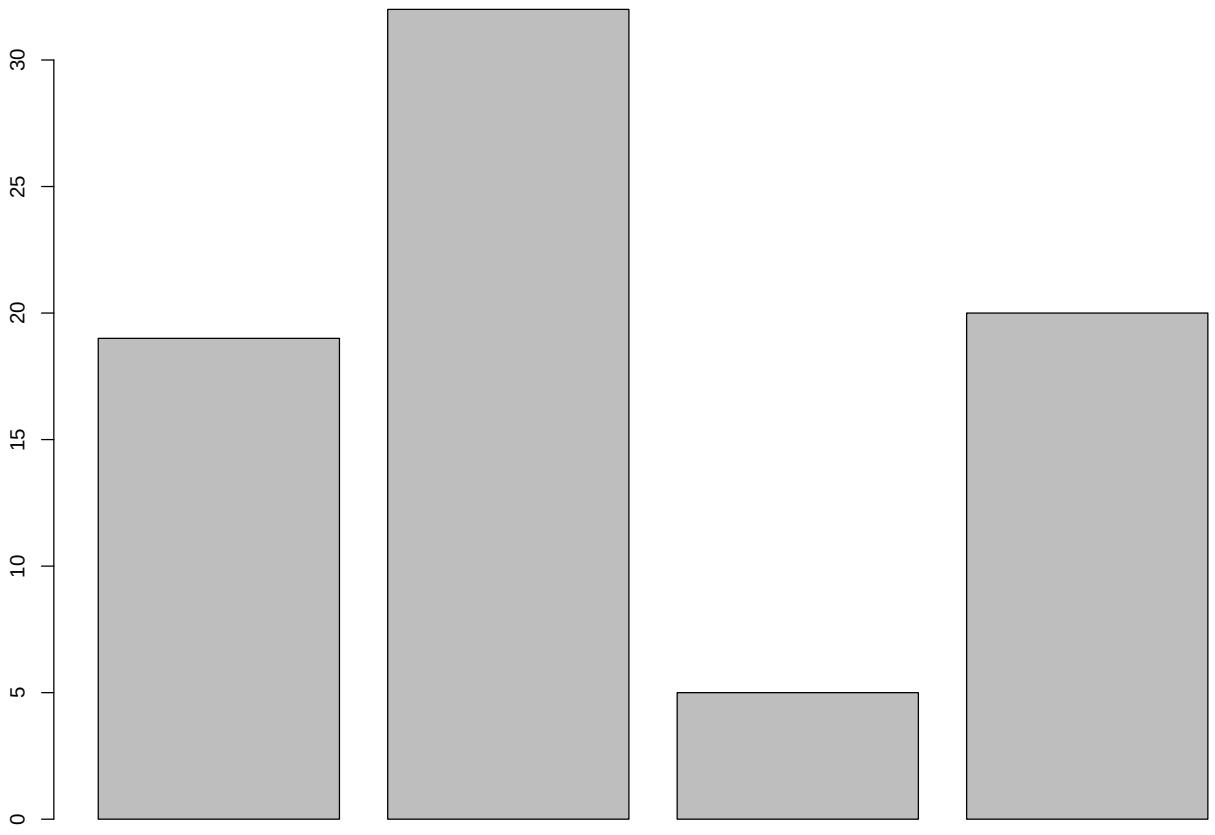
plot() [DYNAMIC]

```
> plot(table(mammalData$vore)) ## mammalSleep.csv
```



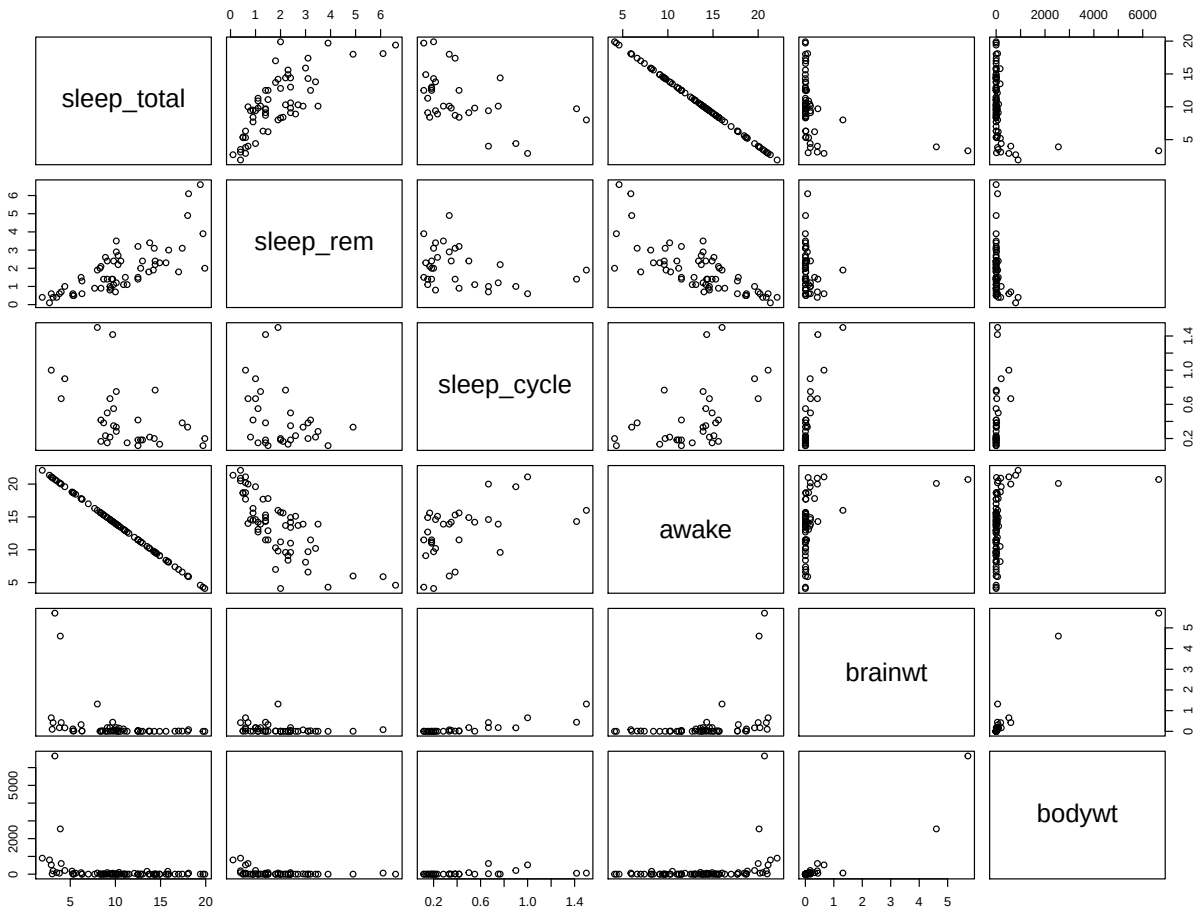
barplot() [BAR GRAPH]

```
> barplot(table(mammalData$vore)) ## mammalSleep.csv
```



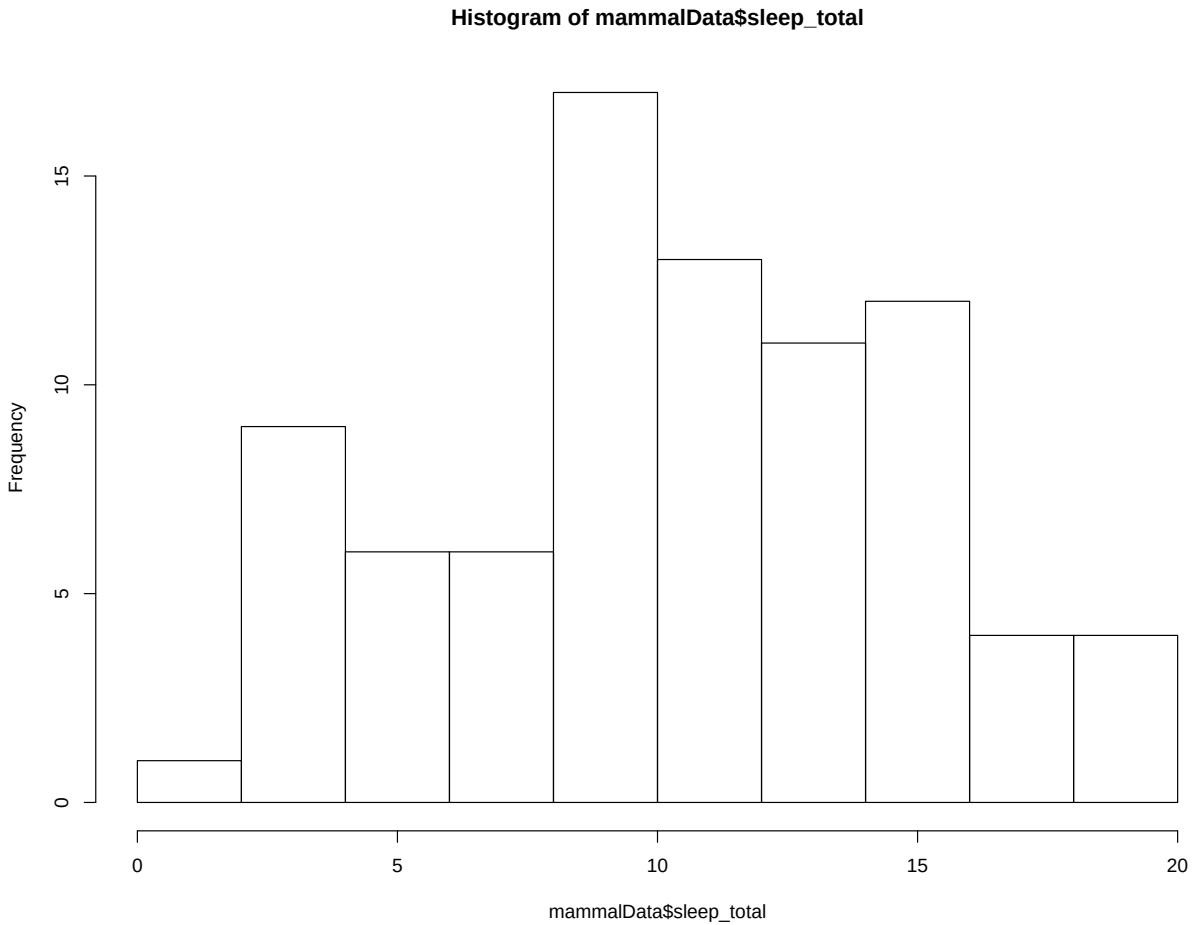
plot() [SCATTER PLOT MATRIX]

```
> mammalData %>% select_if(is.numeric) %>% plot()
```



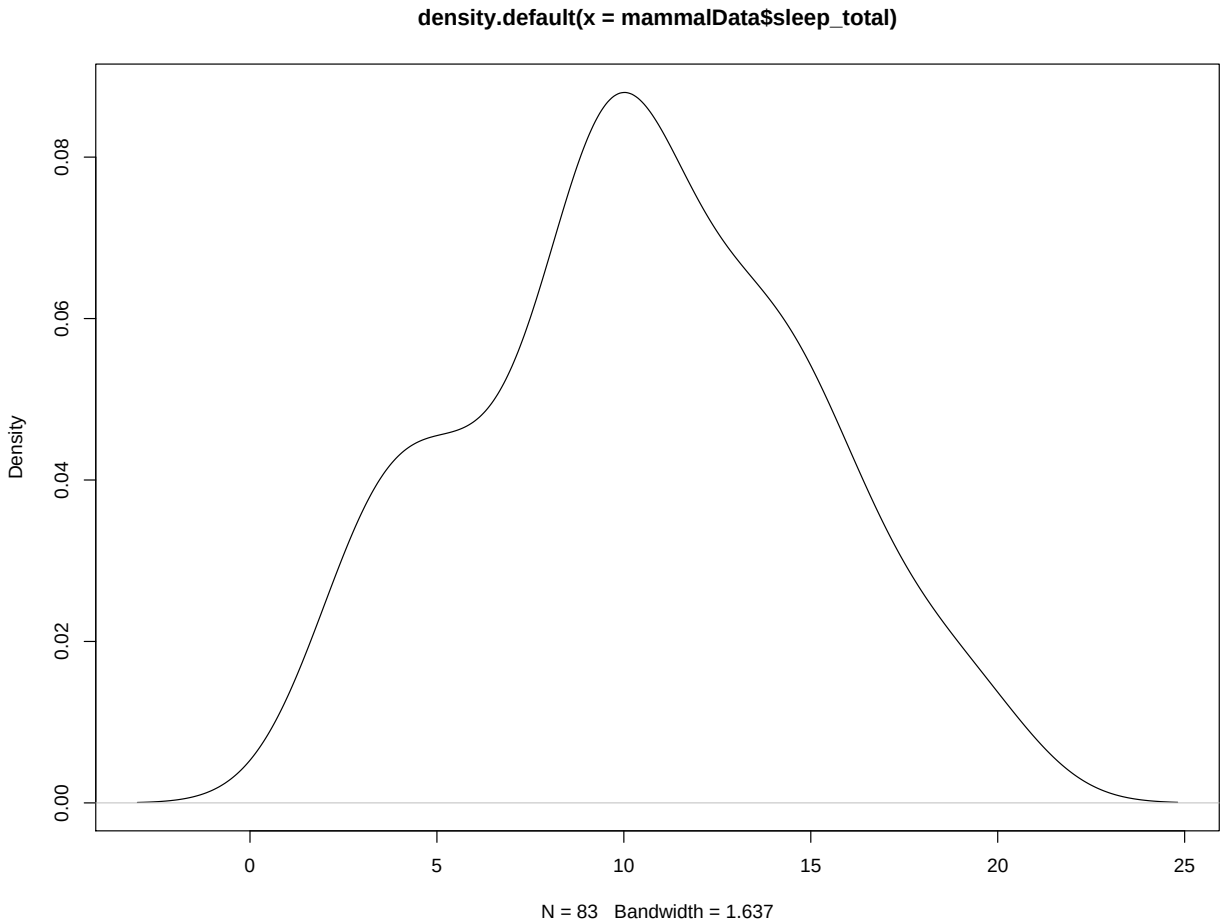
hist() [HISTOGRAM]

```
> hist(mammalData$sleep_total) ## mammalSleep.csv
```



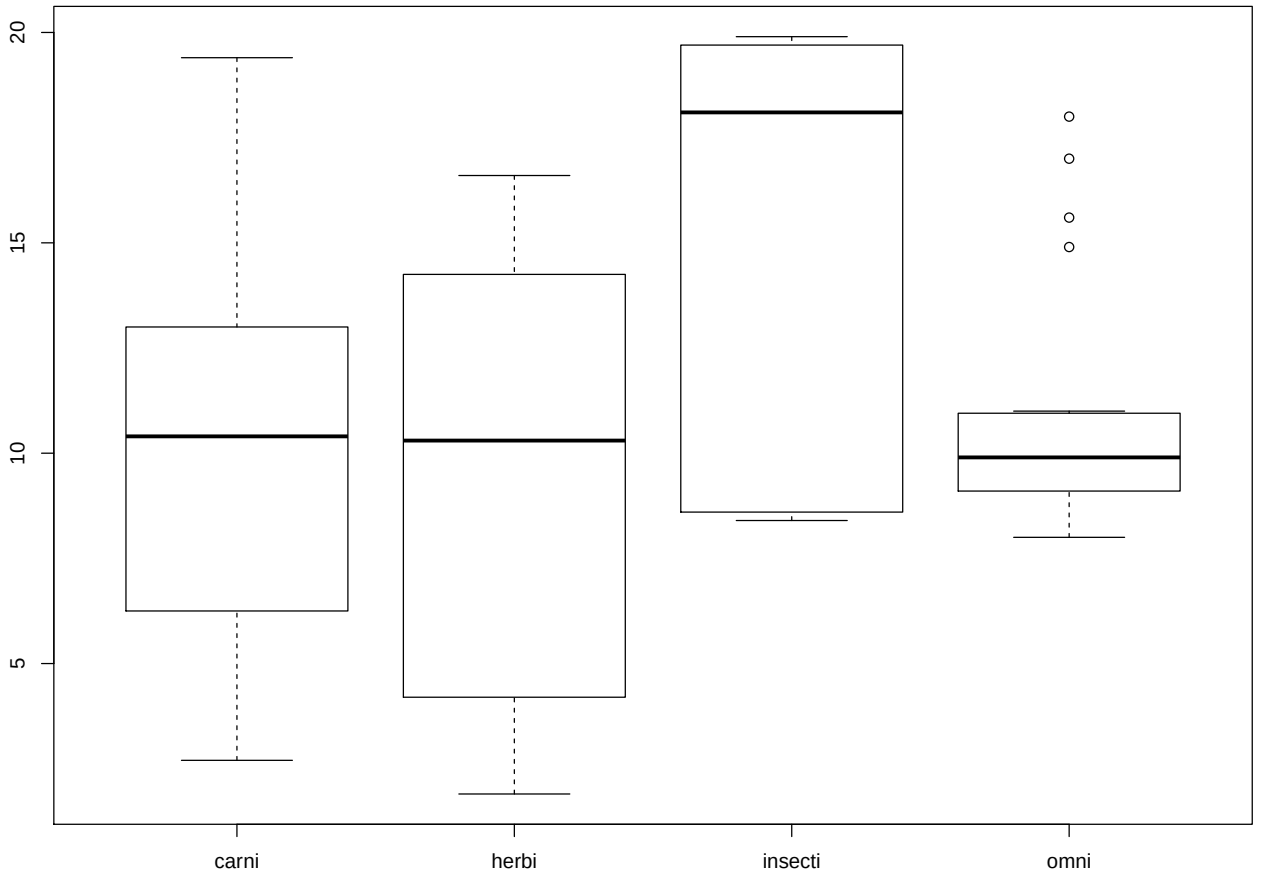
plot() [KERNEL DENSITY PLOT]

```
> plot(density(mammalData$sleep_total)) ## mammalSleep.csv
```



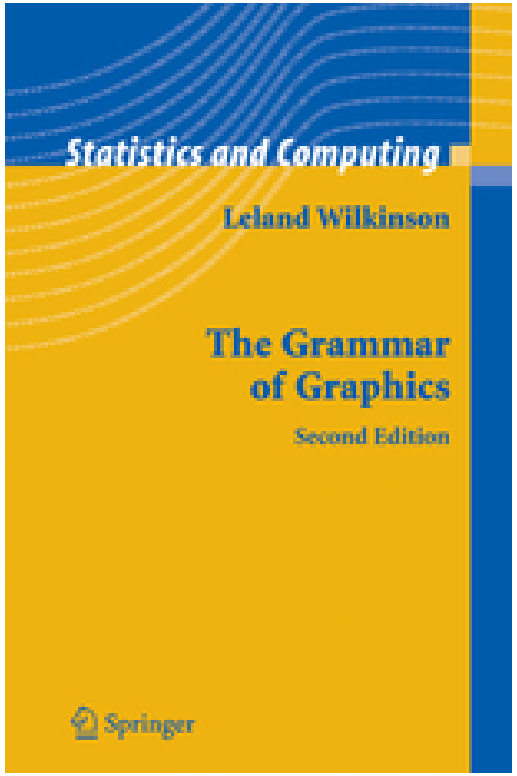
boxplot() [BOX PLOT]

```
> boxplot(sleep_total ~ vore, data = mammalData) ## mammalSleep.csv
```



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*



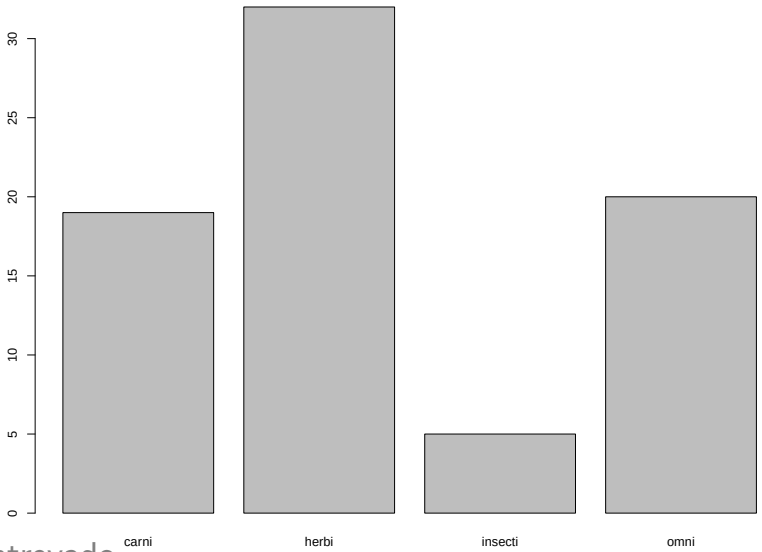
What is ggplot2 [CONT'D]

- This grammar of graphics is composed of a set on independent components that can be composed in many different ways
- This makes What is ggplot2 very powerful, because you are not limited to a set of pre-specified graphics, but you can create new graphics that are precisely tailored to your problem
- Do not be fooled: even though ggplot2 has a high level of flexibility and complexity, publication-quality graphs can be coded in seconds, with many of the extraneous details (e.g, legends) taken care of by ggplot2 default behavior

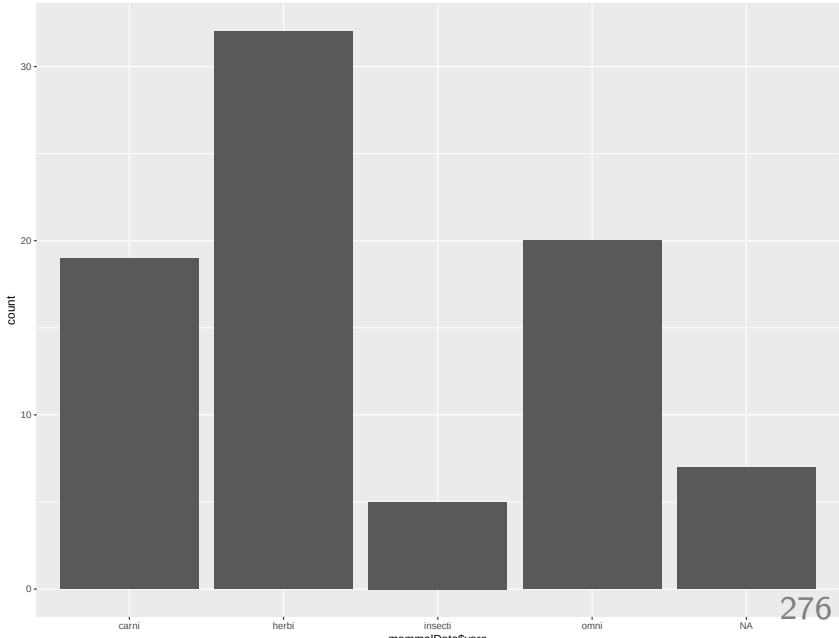
qplot() versus plot()

- 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

```
> barplot(table(mammalData$vore))
```




```
> qplot(mammalData$vore)
```

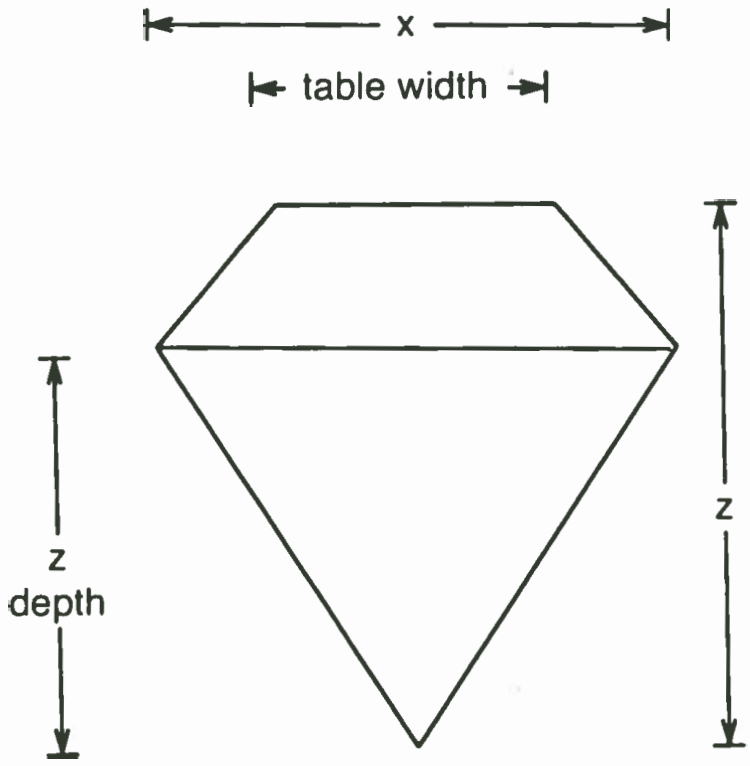


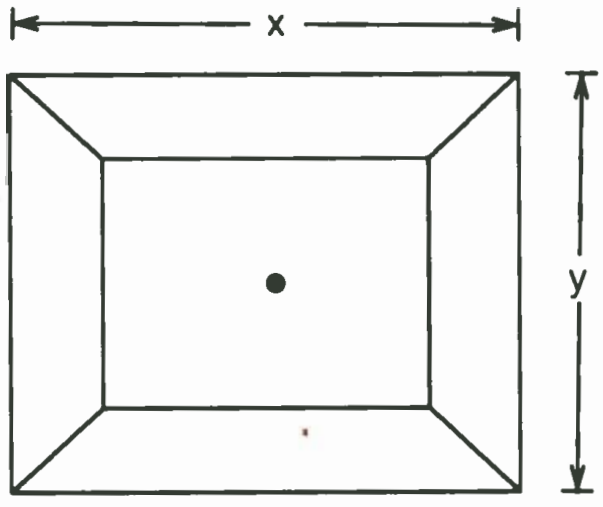
diamonds dataset

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)

diamonds dataset [CONT'D]





depth = z depth / z * 100
table = table width / x * 100

ggplot: Syntax

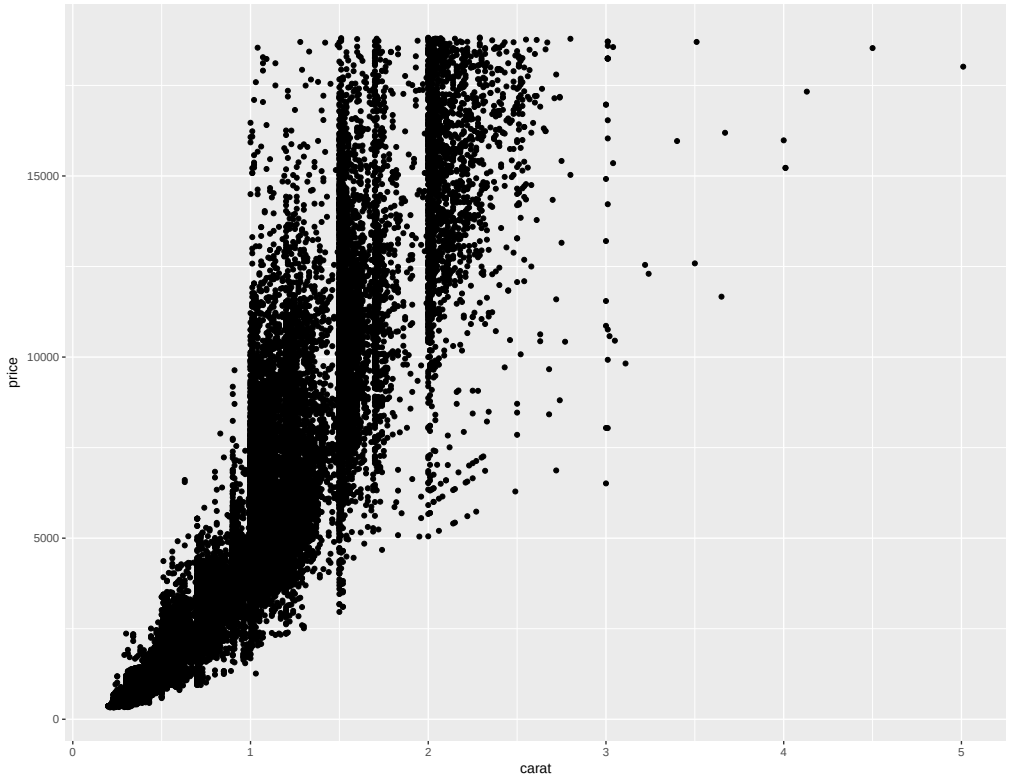
```
> ggplot(data = <myData>) + <geom_*>(mapping = aes(<myPreferredMapping>))

...OR, with piping operator...

> <myData> %>%
     ggplot() +
     <geom_*>(mapping = aes(<myPreferredMapping>))
```

E.g.,

```
> library(tidyverse)
> library(magrittr)

> diamonds %>%
     ggplot(aes(x = carat, y = price))

# the above line of code does not generate a graph, why?
```

ggplot: Syntax [CONT'D]

```
> library(tidyverse)
> library(magrittr)

> diamonds %>%
    ggplot(aes(x = carat, y = price)) +
    geom_point()
```



geoms

- Geometric objects, or geoms as they are commonly known and used, describe the type of object used to display the data
- Common one-dimensional geoms include
  1. geom_histogram
  2. geom_freqpoly
  3. geom_density
  4. geom_bar
- Common two-dimensional geoms 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)

ggplot Geometric Objects

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

One Variable

Continuous

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

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

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

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

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

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

Discrete

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

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

Graphical Primitives

c <- ggplot(map, aes(long, lat))

c + geom_polygon(aes(group = group))
x, y, alpha, color, fill, linetype, size

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

d + geom_path(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, linetype, size

d + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900))
x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

e + geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat))
x, xend, y, yend, alpha, color, linetype, size

e + geom_rect(aes(xmin = long, ymin = lat, xmax= long + delta_long, ymax = lat + delta_lat))
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y

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

f + geom_blank()

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

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

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

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

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

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

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight

g + geom_boxplot()
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + geom_dotplot(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill

g + geom_violin(scale = "area")
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

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

Continuous Bivariate Distribution

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

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

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

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

Continuous Function

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

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

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

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

Visualizing error

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

k + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + geom_errorbar()
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)

k + geom_linerange()
x, ymin, ymax, alpha, color, linetype, size

k + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

l + geom_map(aes(map_id = state), map = map) + expand_limits(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

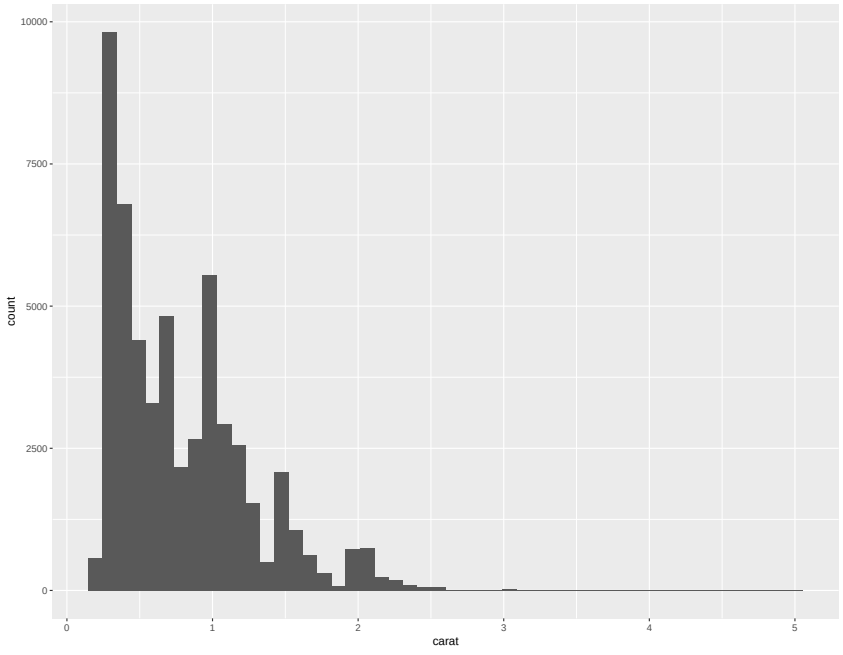
m + geom_contour(aes(z = z))
x, y, z, alpha, colour, linetype, size, weight

m + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

m + geom_tile(aes(fill = z))
x, y, alpha, color, fill, linetype, size

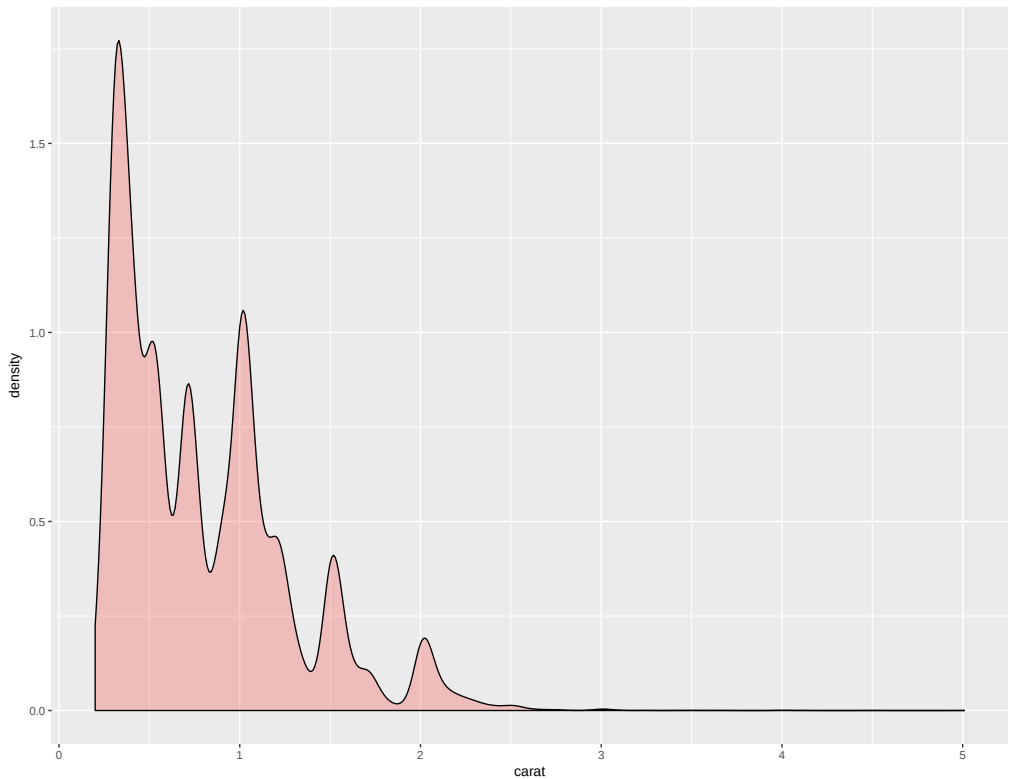
geoms: geom_histogram

```
> diamonds %>%
    ggplot(aes(x = carat)) +  # note only x, why?
    geom_histogram()

    # ...OR...

> diamonds %>%
    ggplot() +
    geom_histogram(aes(x = carat), bins = 50) # default bins = 30
```



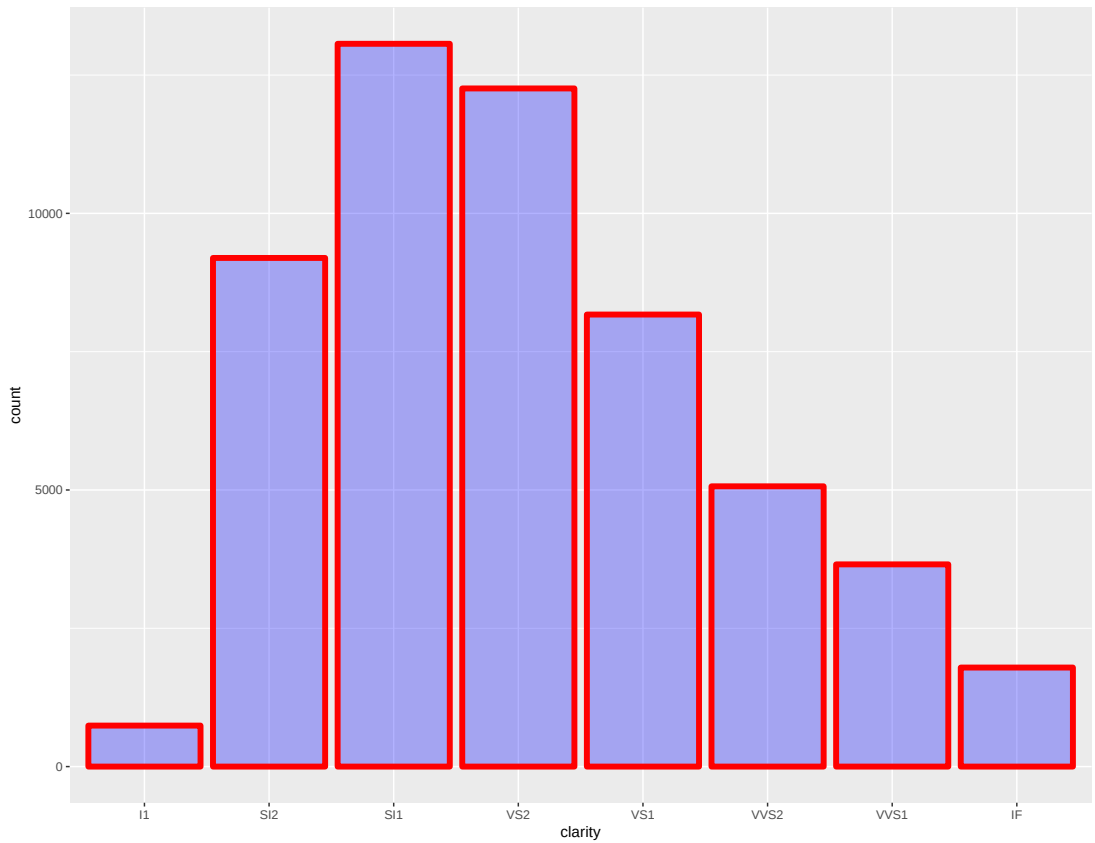
geoms: geom_density

```
> diamonds %>%
    ggplot() +
    geom_density(aes(carat), fill = "red", alpha = 0.2)
    # fill and alpha are optional
```



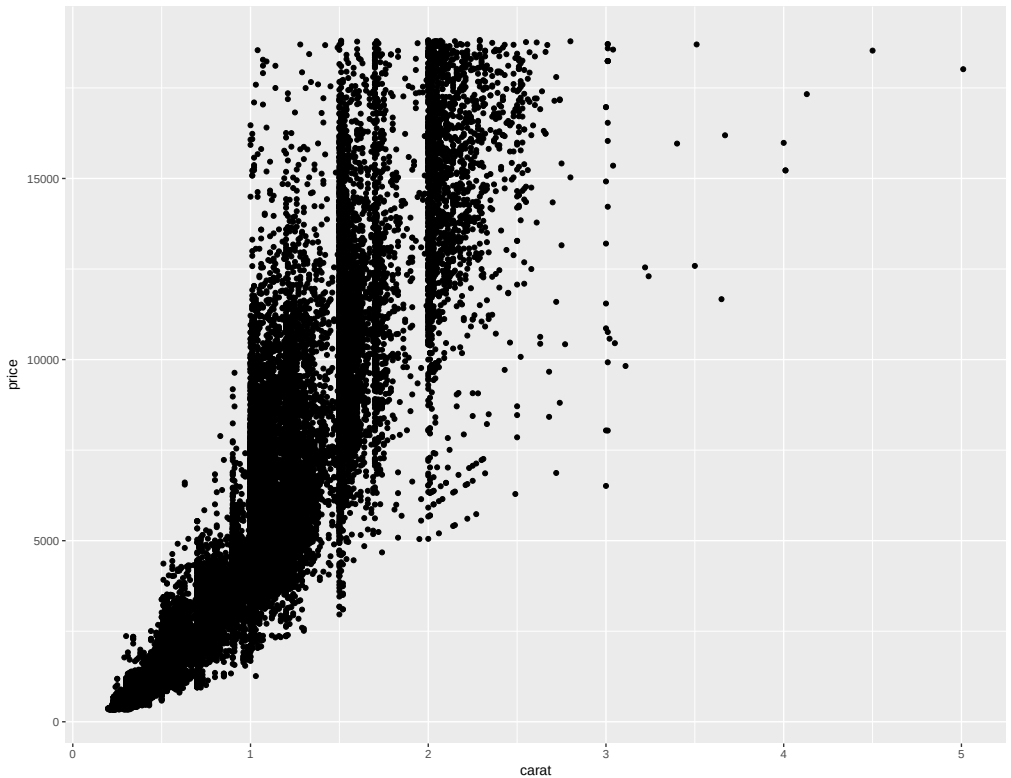
geoms: geom_bar

```
> diamonds %>%
    ggplot() +
    geom_bar(aes(x = clarity), colour = "red", fill = "blue", size = 2, alpha = 0.3)
    # fill, colour, size and alpha are optional
```



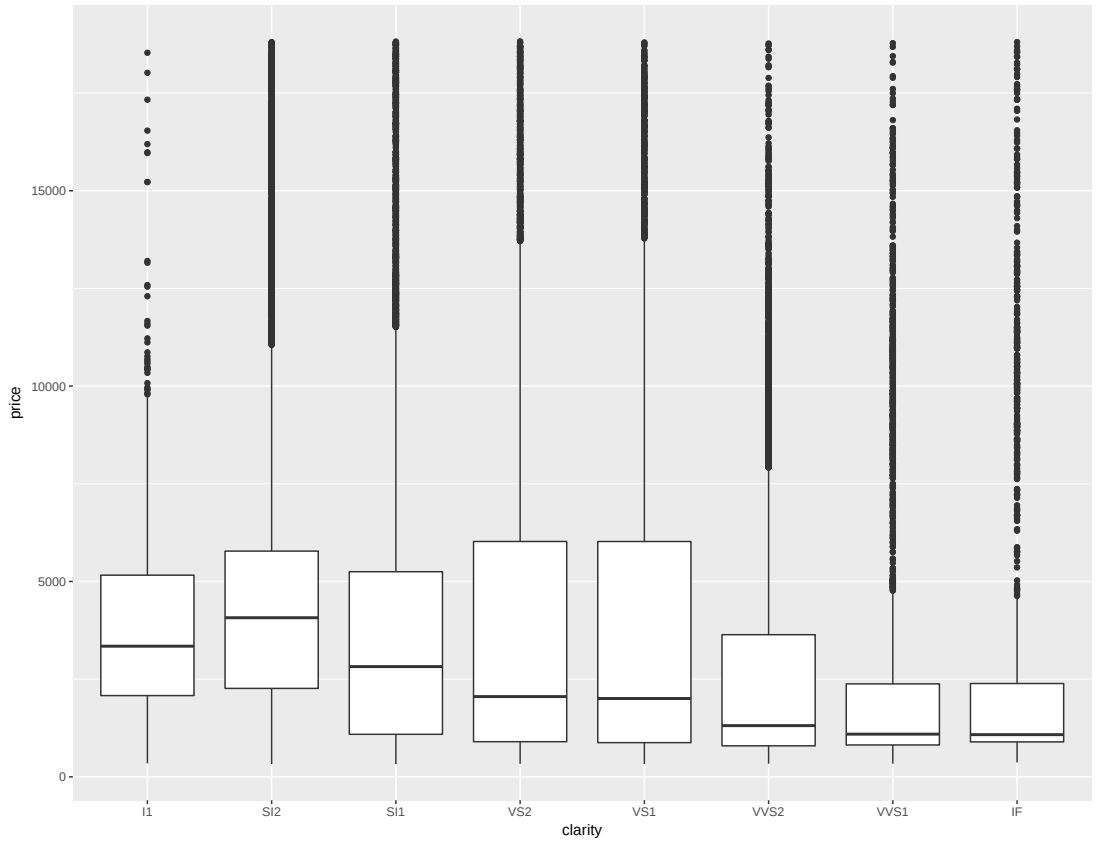
geoms: geom_point

```
> diamonds %>%
    ggplot() +
    geom_point(aes(x = carat, y = price))
```



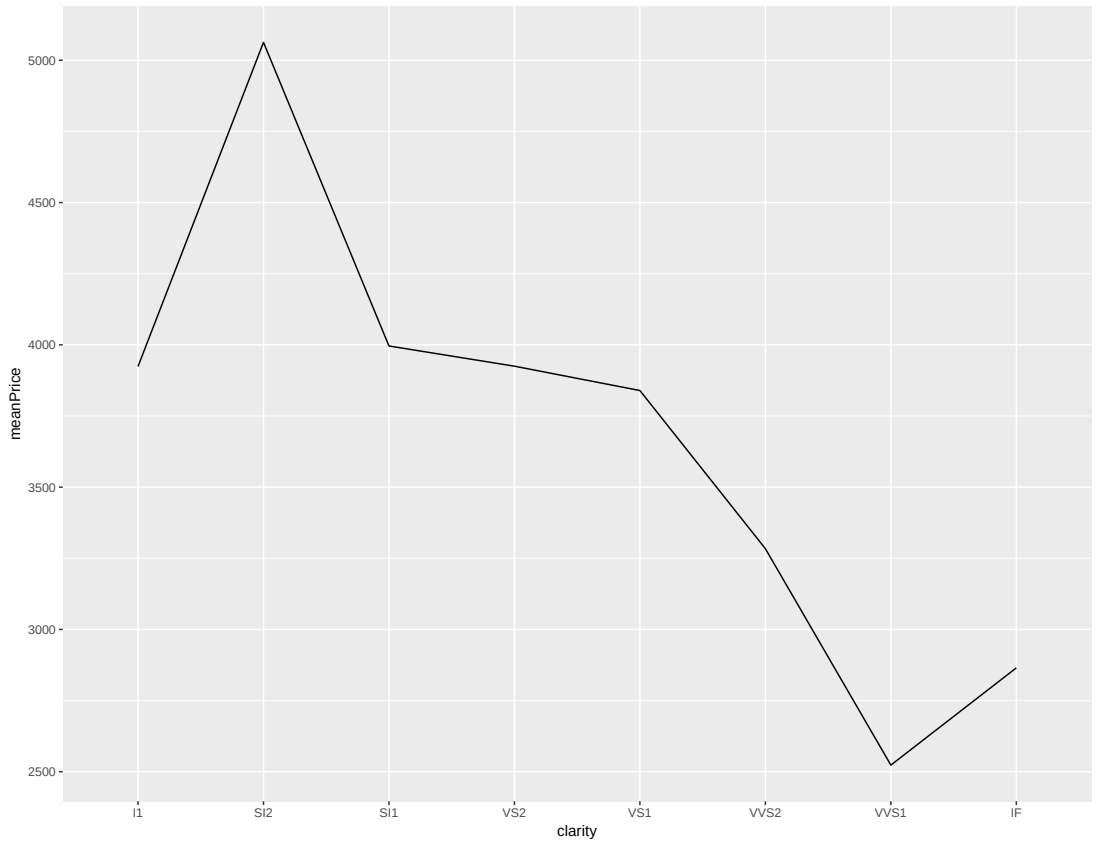
geoms: geom_boxplot

```
> diamonds %>%
    ggplot() +
    geom_boxplot(aes(x = clarity, y = price))
```



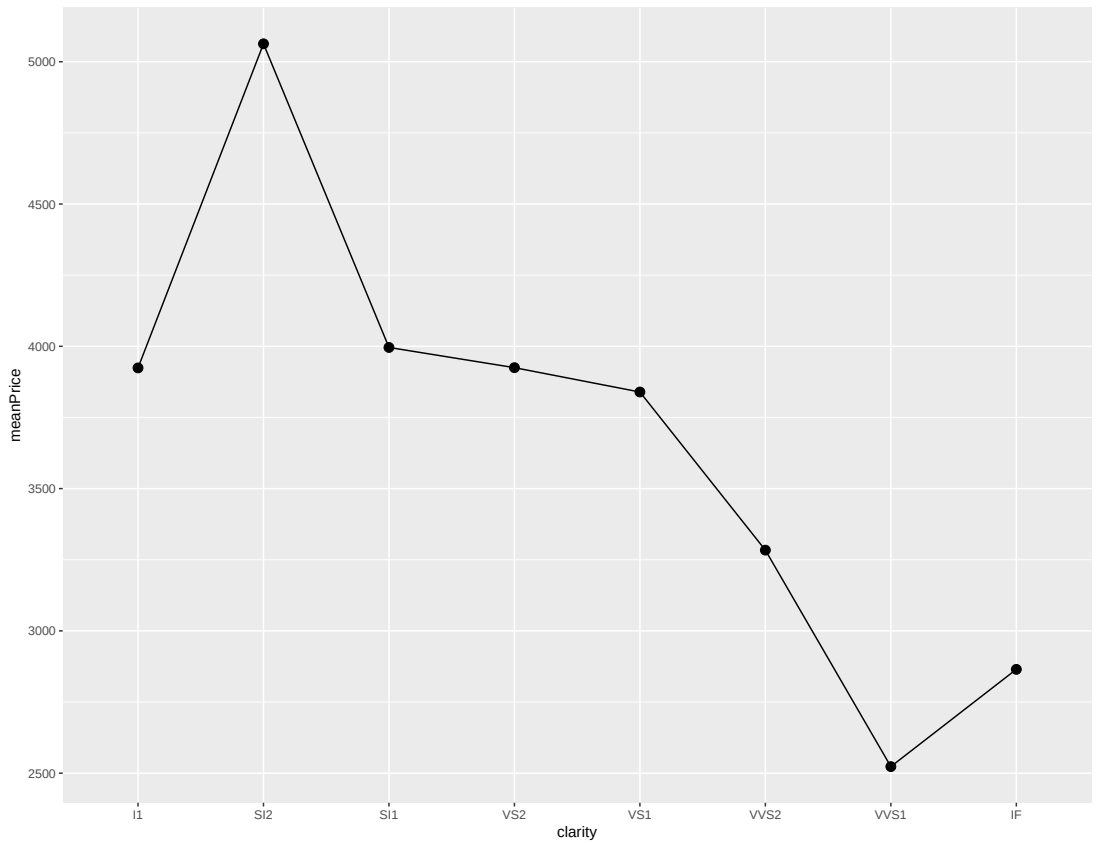
geoms: geom_line

```
> diamonds %>% group_by(clarity) %>% summarise(meanPrice = mean(price)) %>%
      ggplot() +
      geom_line(aes(x = clarity, y = meanPrice), group = 1)
```



Layering geoms

```
> diamonds %>% group_by(clarity) %>% summarise(meanPrice = mean(price)) %>%
      ggplot() +
      geom_line(aes(x = clarity, y = meanPrice), group = 1) +
      geom_point(aes(x = clarity, y = meanPrice), size = 3) # size optional
```



Data

- 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

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

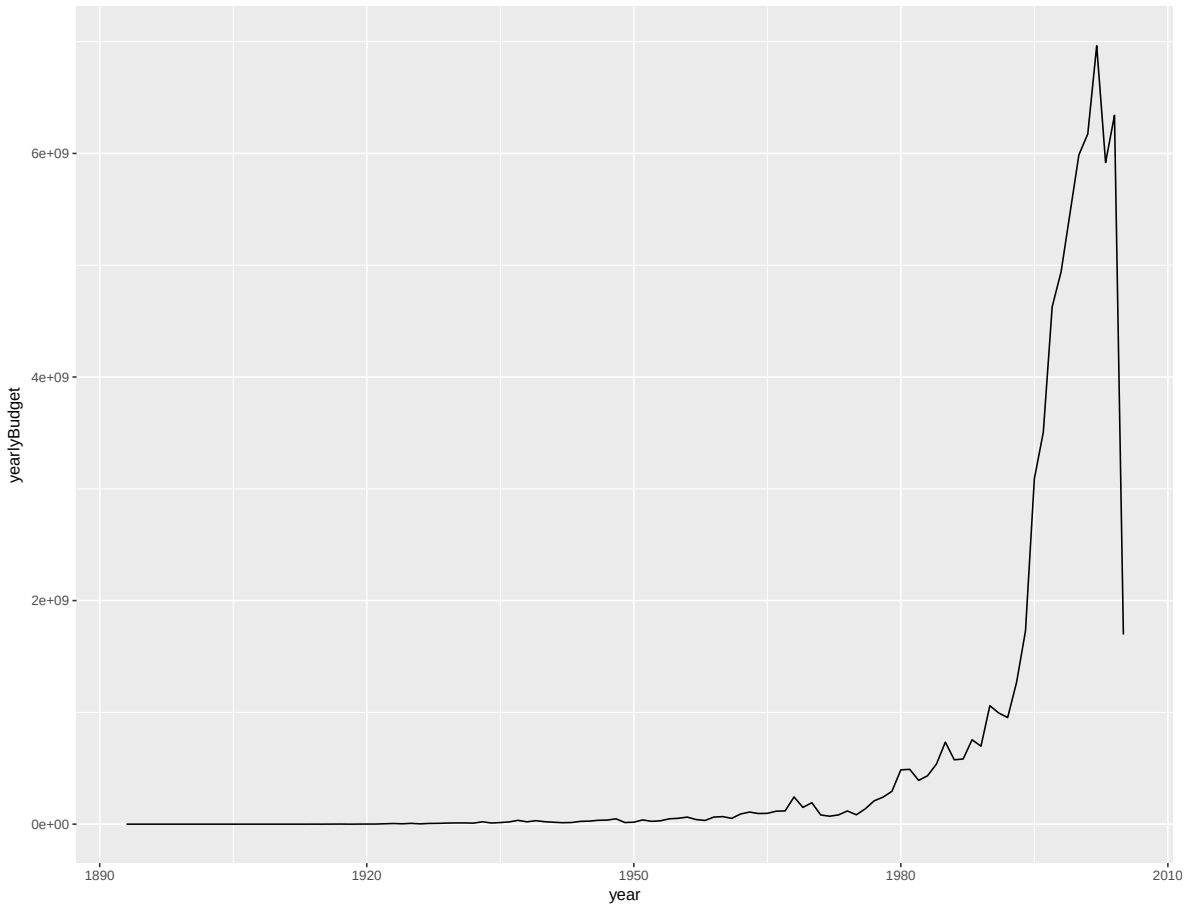
- You could make it work (as the above does), but it is bad form and, moreover, it defeats the purpose, strength and flexibility of `ggplot()`

aes(): Aesthetic Mappings

- To map variables to different parts of the plot, we use the `aes()` function, which takes a list of aesthetic-variable pairs
- For a one-variable graph, e.g., a histogram, the minimal information required to be supplied to `aes(x = <myXdata>)`
- In a two variable graph, aesthetic information required include `aes(x = <myXdata>, y = <myYdata>)`
- Aeshtetics can be mapped at the `ggplot()` level or at the `geom_*()`

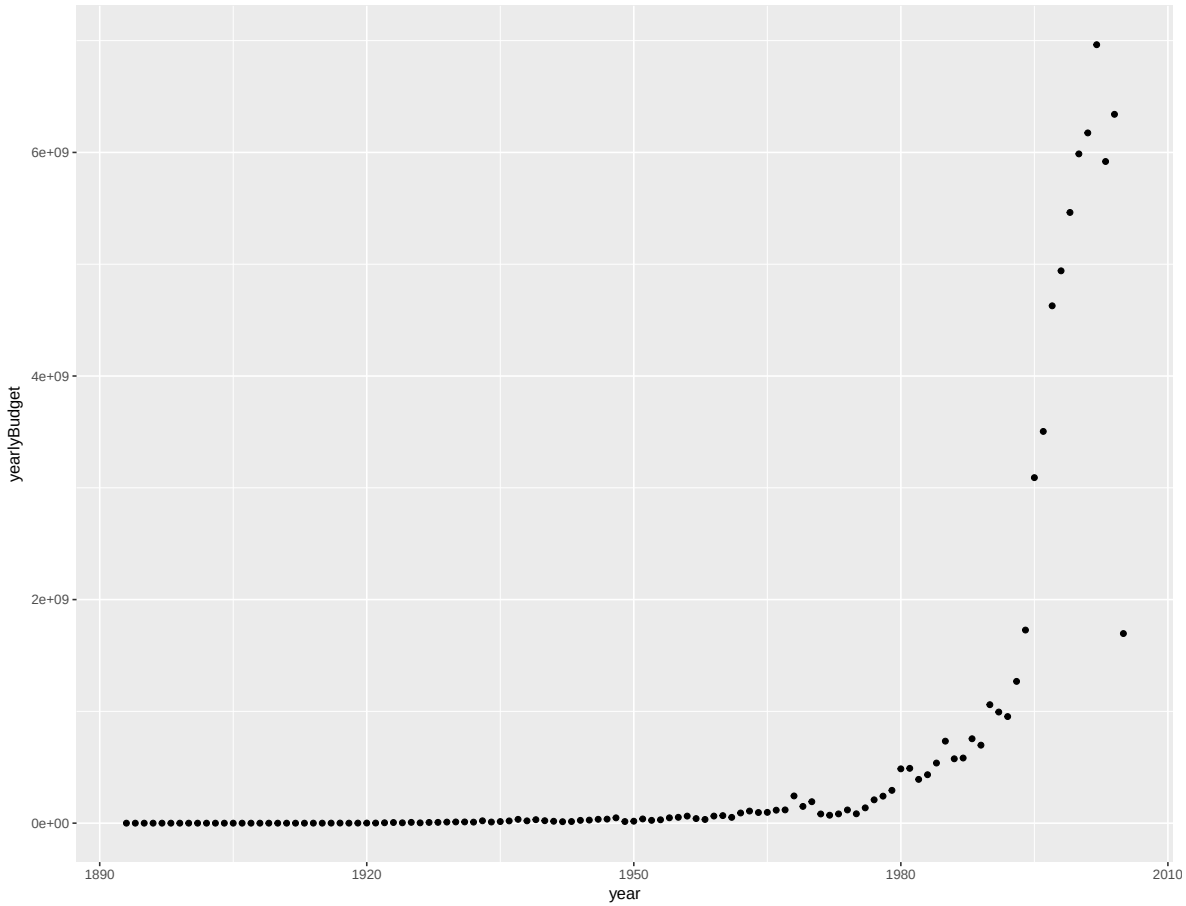
aes(): Aesthetic Mappings [EXAMPLE 1/2]

```
> ggplot2movies::movies %>% group_by(year) %>%
    summarize(yearlyBudget = sum(as.numeric(budget), na.rm = T)) %>%
      ggplot() + geom_line(aes(x = year, y = yearlyBudget))
```



aes(): Aesthetic Mappings [EXAMPLE 2/2]

```
> ggplot2movies::movies %>% group_by(year) %>%
    summarize(yearlyBudget = sum(as.numeric(budget), na.rm = T)) %>%
      ggplot() + geom_point(aes(x = year, y = yearlyBudget))
```

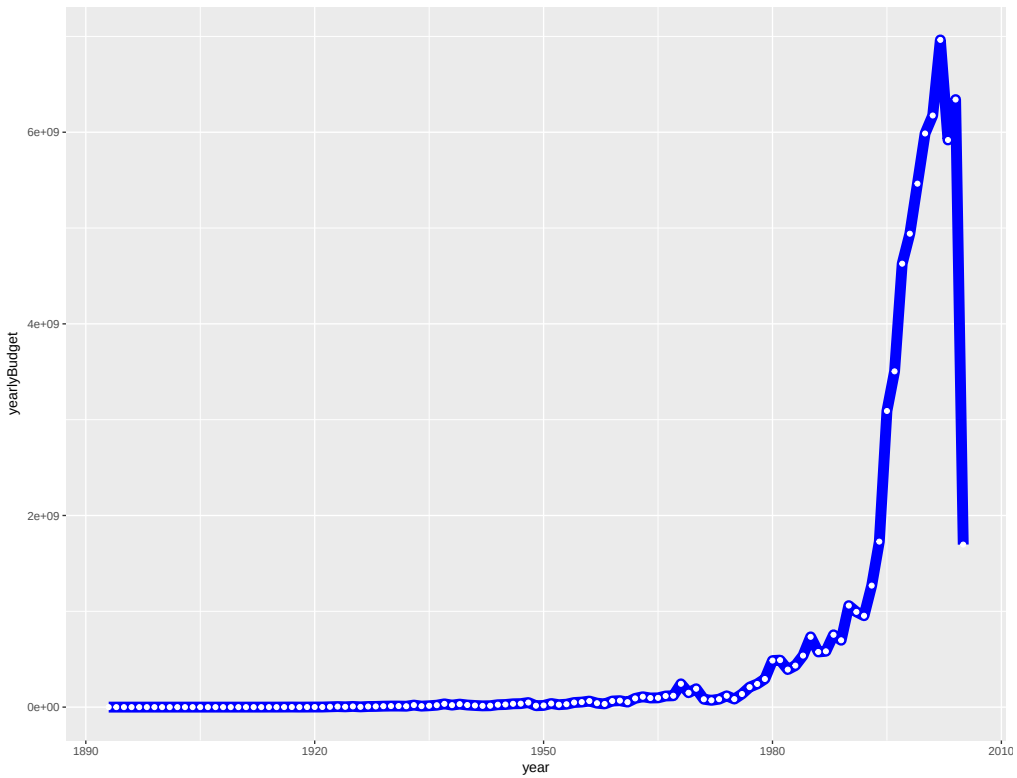


Layers

- Perhaps the most powerful aspect of `ggplot()` is the ability to construct graphics from layers
- Creating the `ggplot()` object creates a base graphical object
- Creating a `geom` or layering multiple `geom`s provides the user an empty canvas with which to create highly-stylized graphics to convey information
- The order in which `geom`s are layered will affect the output

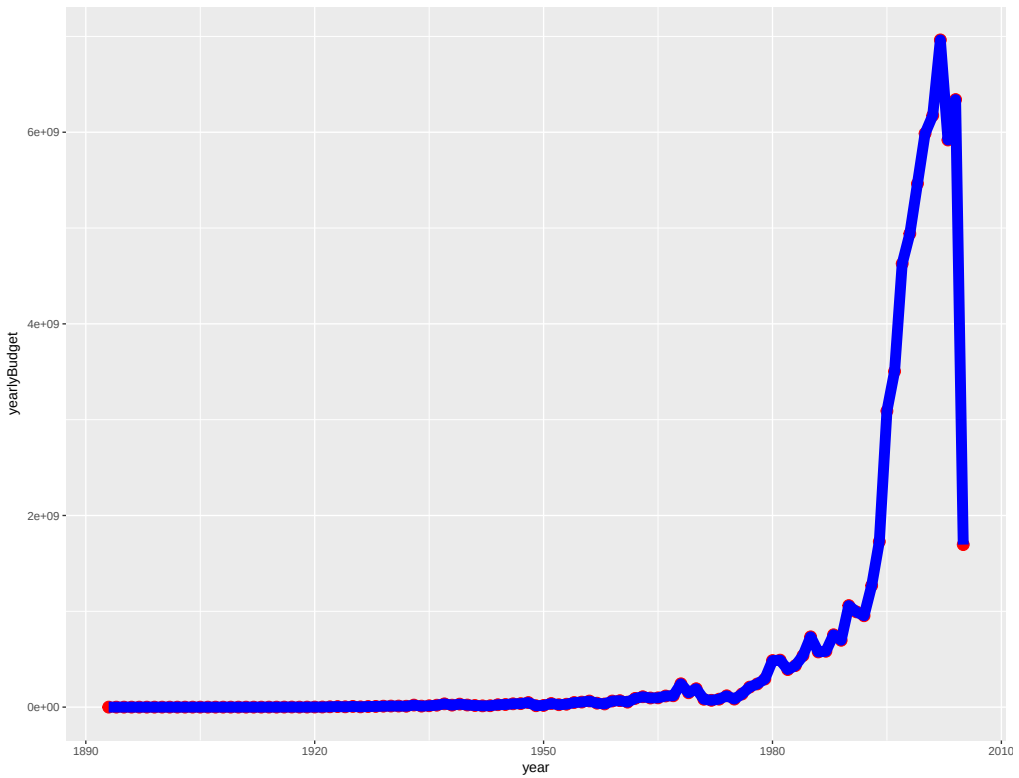
Layering geoms [EXAMPLE 1/3]

```
> ggplot2movies::movies %>%
    group_by(year) %>% summarize(yearlyBudget = sum(as.numeric(budget),
    na.rm = T)) %>% ggplot() +
        geom_line(aes(x = year, y = yearlyBudget), color = "blue", size = 4) +
        geom_point(aes(x = year, y = yearlyBudget))
```



Layering geoms [EXAMPLE 2/3]

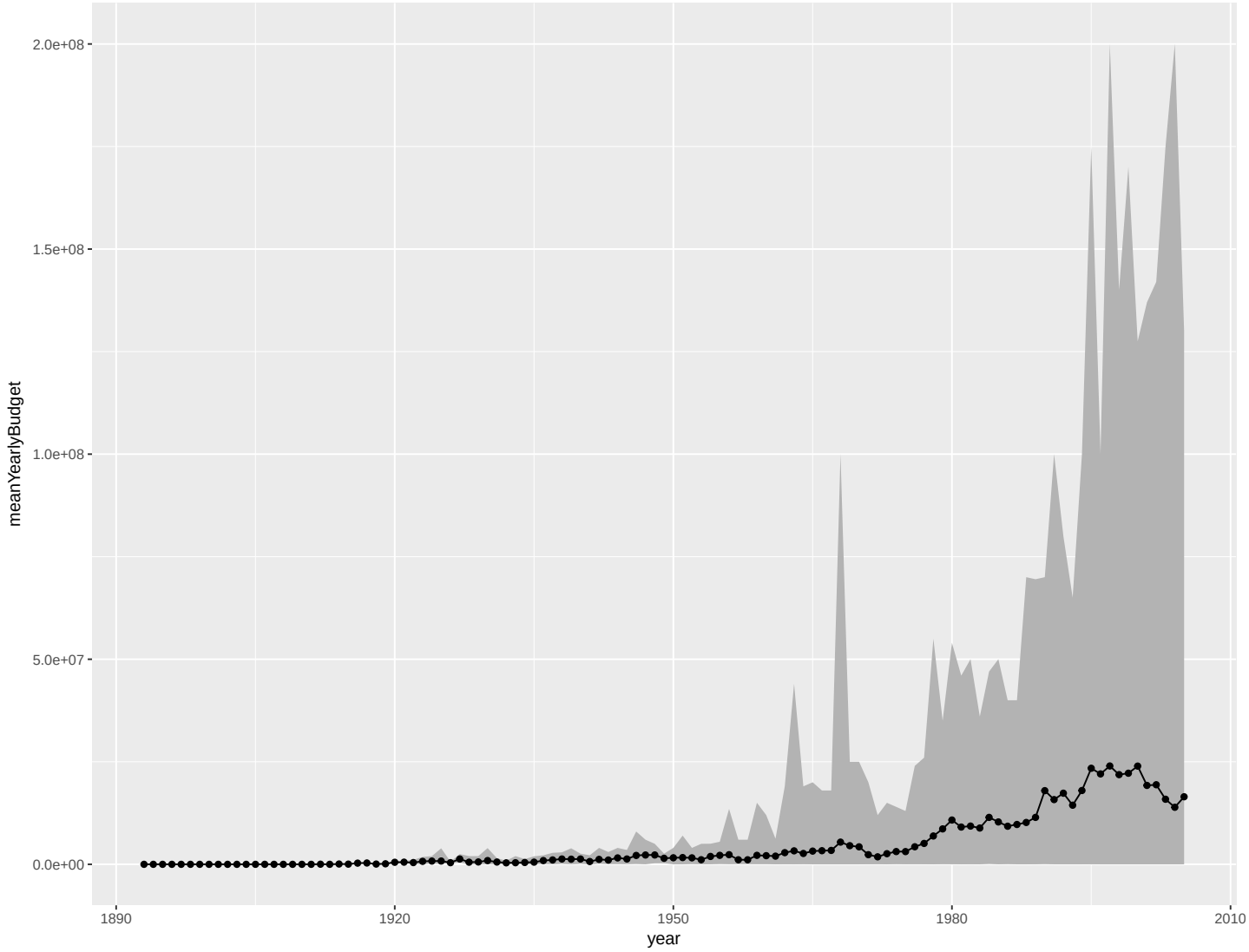
```
> ggplot2movies::movies %>%
    group_by(year) %>% summarize(yearlyBudget = sum(as.numeric(budget),
    na.rm = T)) %>% ggplot() +
        geom_point(aes(x = year, y = yearlyBudget), color = "red", size = 4) +
        geom_line(aes(x = year, y = yearlyBudget), color = "blue", size = 4)
```



Layering geoms [EXAMPLE 3/3]

```
# create DF of mean yearly budgets
meanYearlyBudget <- ggplot2movies::movies %>%
  group_by(year) %>% summarize(meanYearlyBudget = mean(as.numeric(budget),
    na.rm = T))

# create DF of min/max for each year
summaryStats <- ggplot2movies::movies %>%
  group_by(year) %>% summarize(minBudget = min(as.numeric(budget), na.rm = T),
    maxBudget = max(as.numeric(budget), na.rm = T))

# remove NaNs from DF
meanYearlyBudget$meanYearlyBudget[is.nan(meanYearlyBudget$meanYearlyBudget)] <- 0

# remove Inf/-Inf from DF
summaryStats[summaryStats == Inf | summaryStats == -Inf] <- 0

# plot mean line (line + point) with underlying ribbon for min/max
 # ordering of layers is important here
ggplot() +
  geom_ribbon(data = summaryStats, aes(x = year, ymin = minBudget,
    ymax = maxBudget), fill = "grey70") +
  geom_point(data = meanYearlyBudget, aes(x = year, y = meanYearlyBudget)) +
  geom_line(data = meanYearlyBudget, aes(x = year, y = meanYearlyBudget))
```

Layering geoms [EXAMPLE 3/3] [CONT'D]

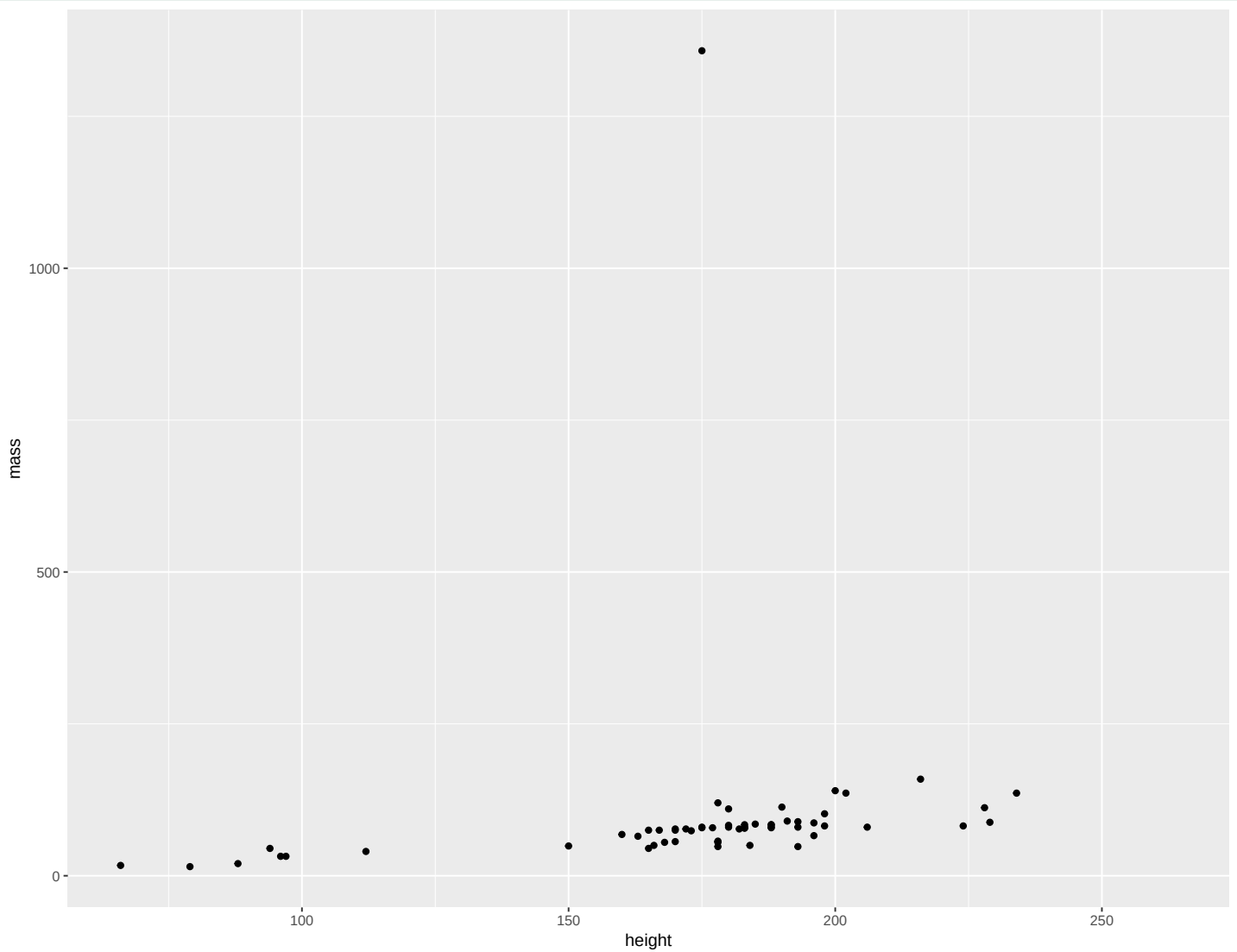


TRY THIS

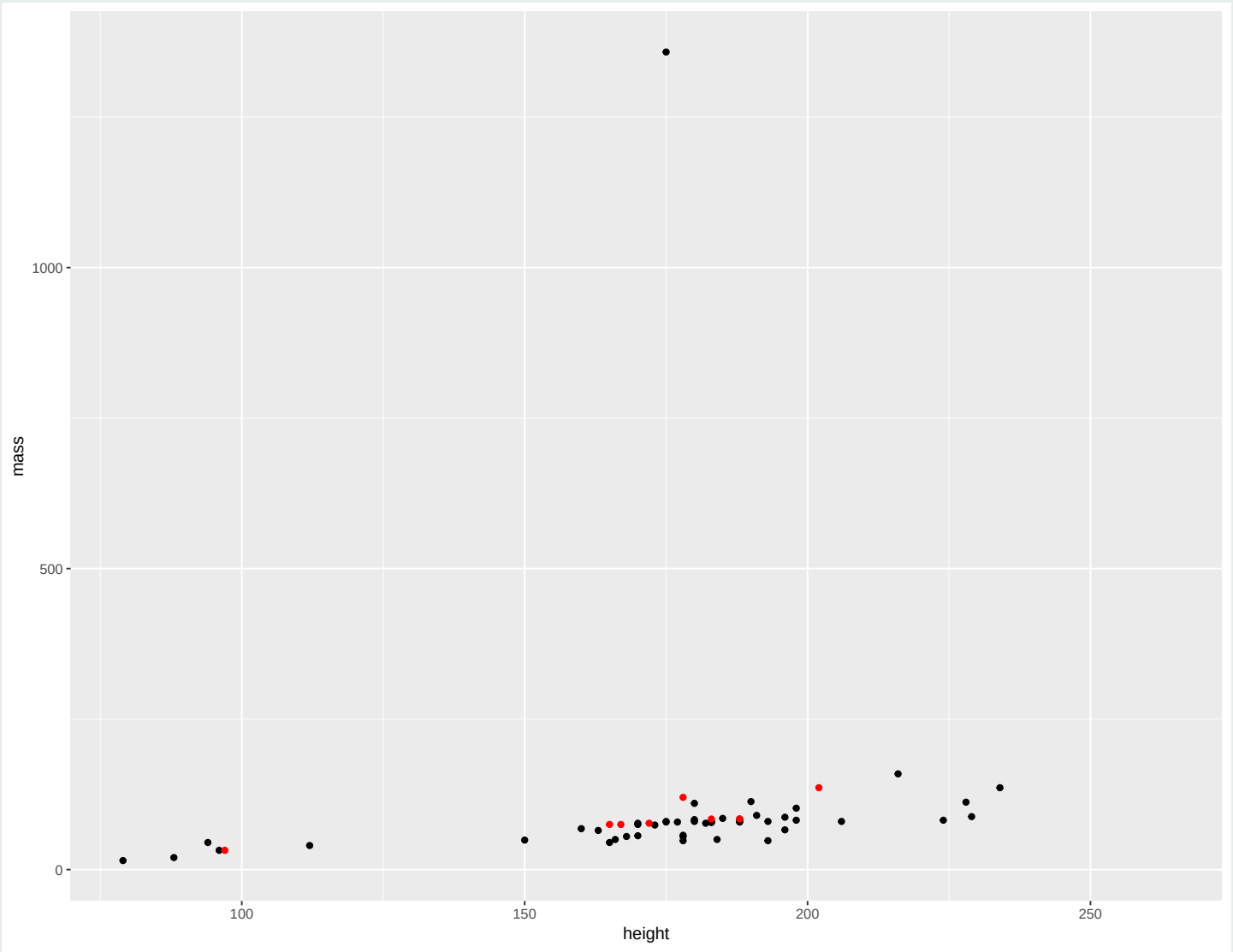
Is there a relationship between `height` and `mass` in the star wars universe?

1. Using `dplyr::starwars`, create a scatter plot in `ggplot` with $x =$ `height` , $y =$ `mass`
2. Create the same graph as but on this one, highlight all of the characters from Tatooine in red

FINAL OUTPUT 1



FINAL OUTPUT 2



SOLUTION 1

```
> dplyr::starwars %>%
    ggplot() +
     geom_point(aes(x = height, y = mass))
```

SOLUTION 2

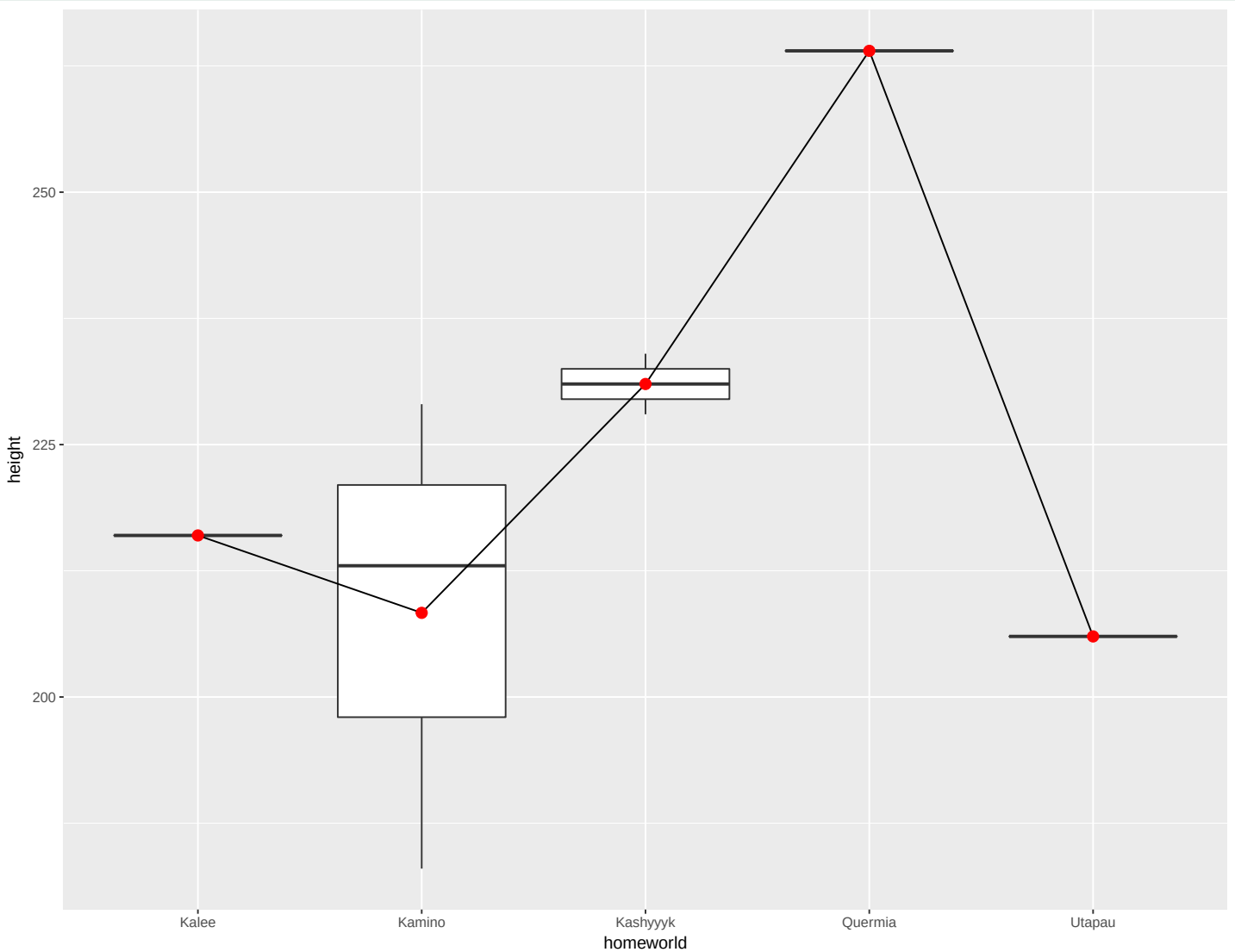
```
> starSub <- starwars %>% filter(homeworld == "Tatooine")

> ggplot() +
     geom_point(aes(x = height, y = mass), data = starwars) +
     geom_point(aes(x = height, y = mass), data = starSub, color = "red")
```

TRY THIS

Using `dplyr::starwars`, create a graph in `ggplot` with

1. Find the planets with the top five mean heights
2. Create a box plot of the distributions of those heights
3. Overlay a line connecting the mean heights of those five plannets (*hint: use `group = 1`*)

FINAL OUTPUT



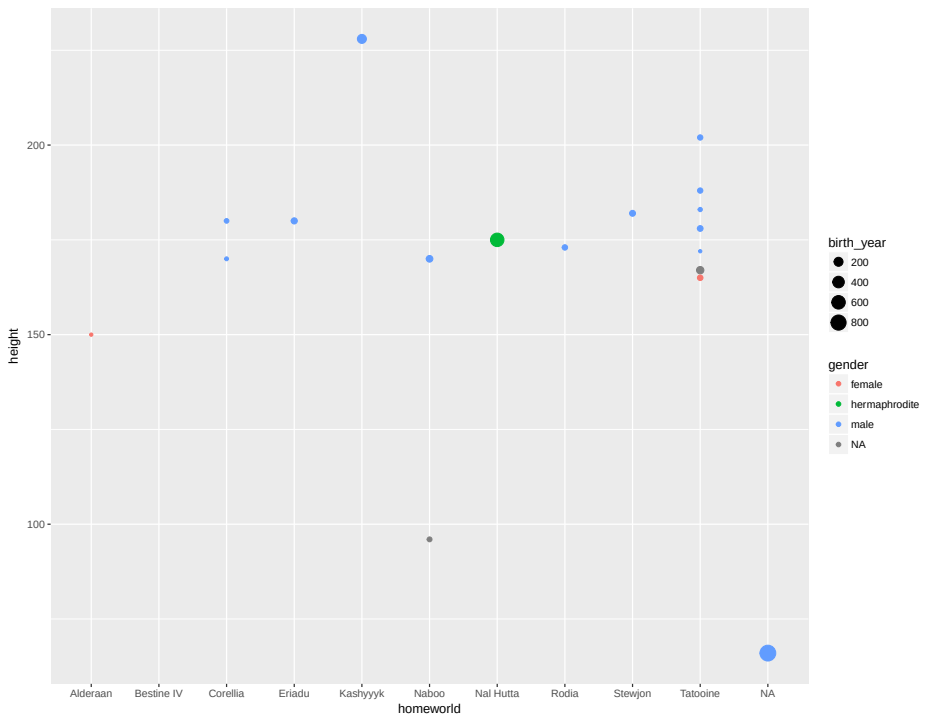
SOLUTION

```
df1 <- dplyr::starwars %>%
    select(height, homeworld) %>%
      group_by(homeworld) %>%
        summarise(meanHeight = mean(height, na.rm = T)) %>%
          arrange(desc(meanHeight)) %>%
            ungroup() %>%
              slice(1:5)

df2 <- dplyr::starwars %>%
  select(height, homeworld) %>%
    filter(homeworld %in% x[["homeworld"]])

ggplot() +
    geom_boxplot(data = df2, aes(x = homeworld, y = height)) +
    geom_line(data = df1, aes(x = homeworld, y = meanHeight), group = 1) +
    geom_point(data = df1, aes(x = homeworld, y = meanHeight),
      size = 3, color = "red")
```

2D Graphic **BUT** *n*D Information

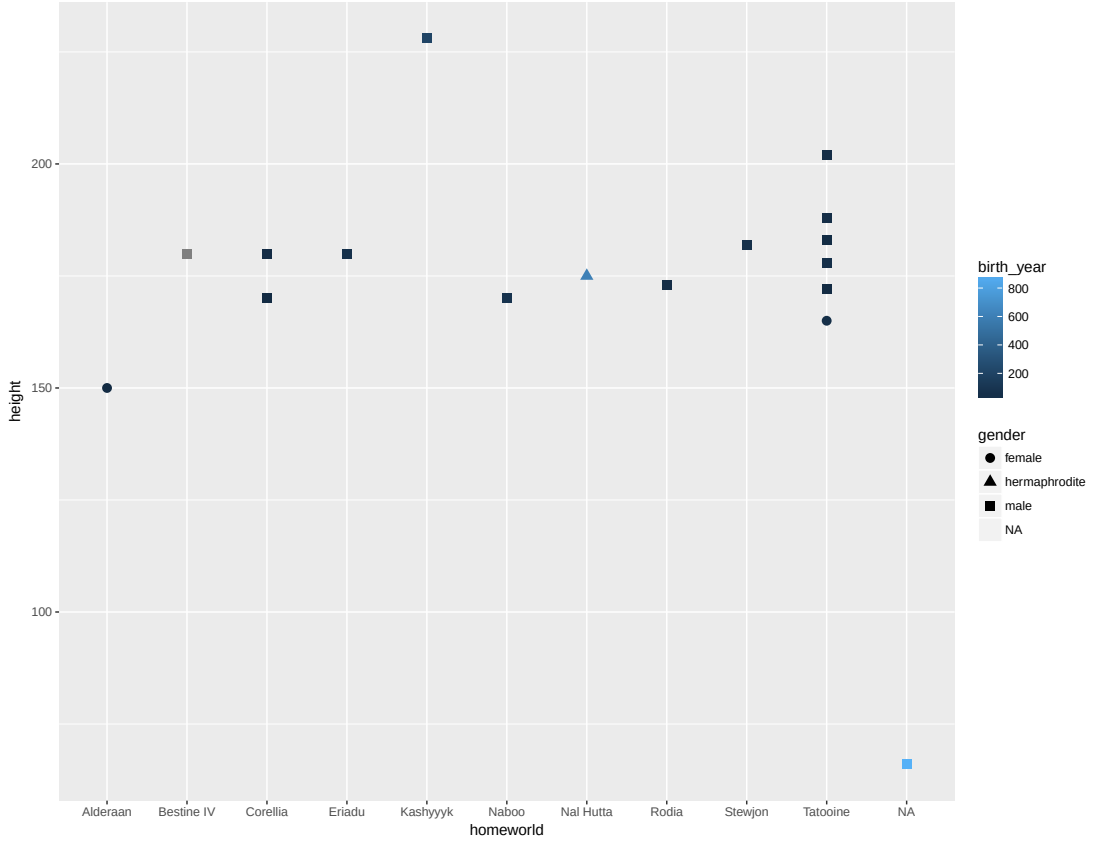
- A third variable can be added to a plot by mapping it to an aesthetic, such as color, shape, size

```
dplyr::starwars[1:20, ] %>% ggplot() +
    geom_point(aes(x = homeworld, y = height, color = gender, size = birth_year))
```



2D Graphic **BUT** *n*D Information [CONT'D]

```
dplyr::starwars[1:20, ] %>% ggplot() +
  geom_point(aes(x = homeworld, y = height,
             shape = gender, color = birth_year), size = 5)
```



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 =`

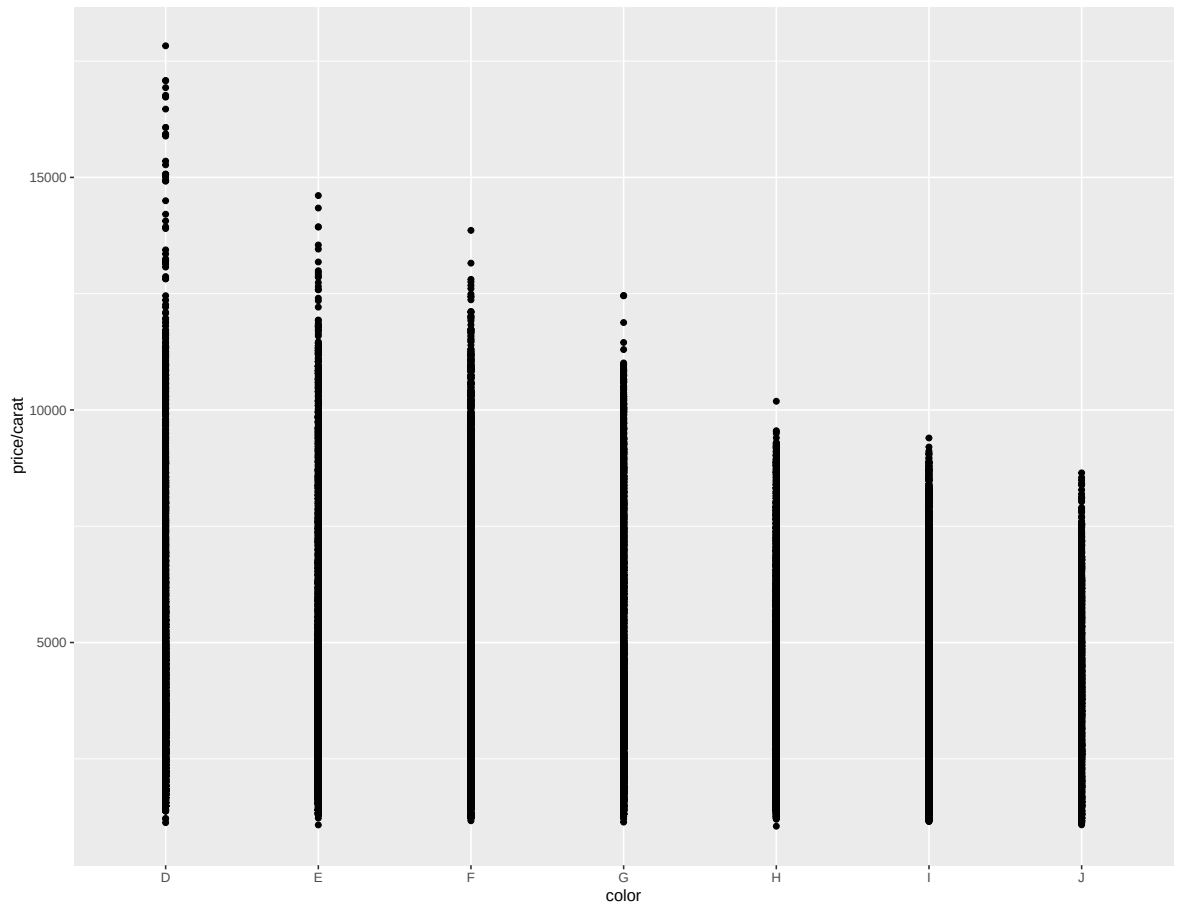
Adjustment	Description
dodge	Adjust position by dodging overlaps to the side
fill	Stack overlapping objects and standardise have equal height
identity	Don't adjust position
jitter	Jitter points to avoid overplotting
stack	Stack overlapping objects on top of one another

`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 points in an effort get a better sense of how many points exist at any given response level

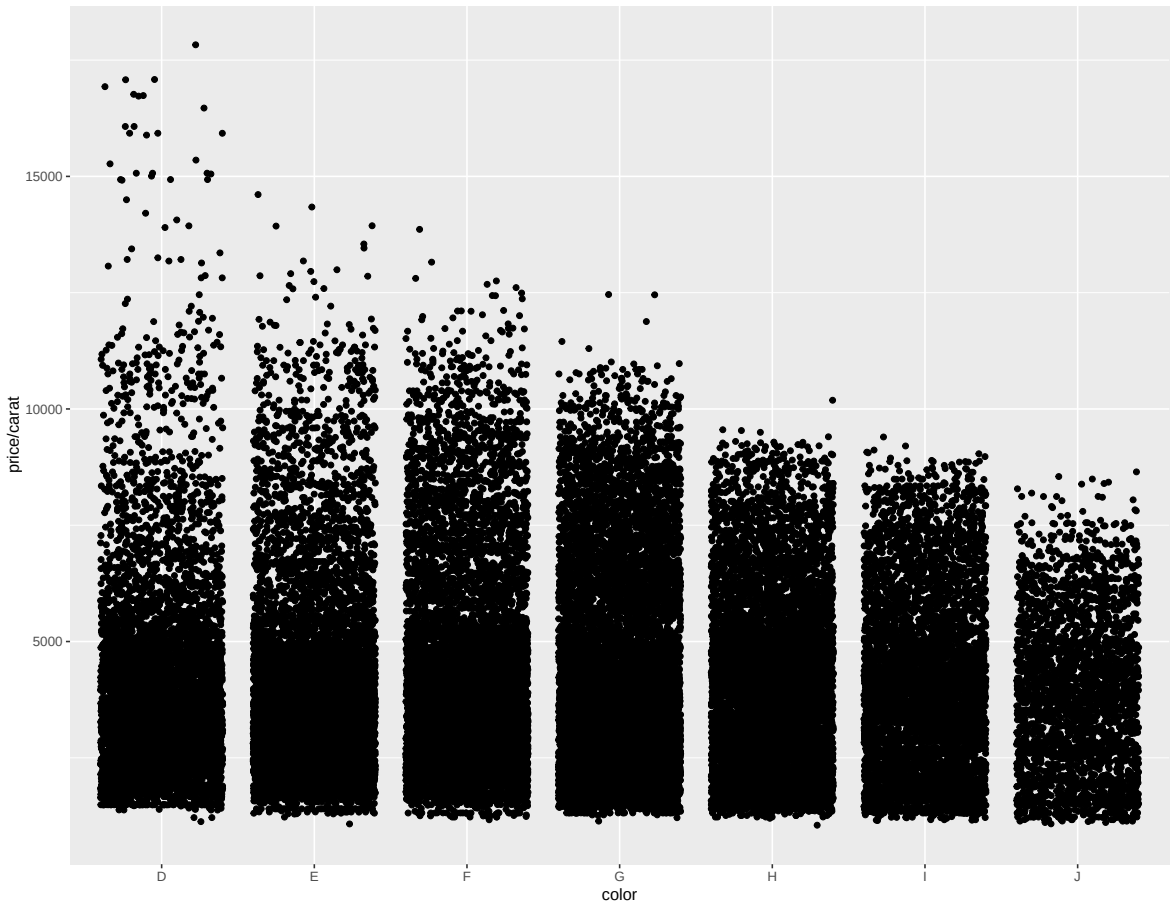
geom = "jitter" [EXAMPLE 1/2]

```
diamonds %<>% mutate(pricePerCarat = price / carat)
myDiamonds %>% ggplot(aes(x = color, y = pricePerCarat)) +
               geom_point() ### no jitter here
```



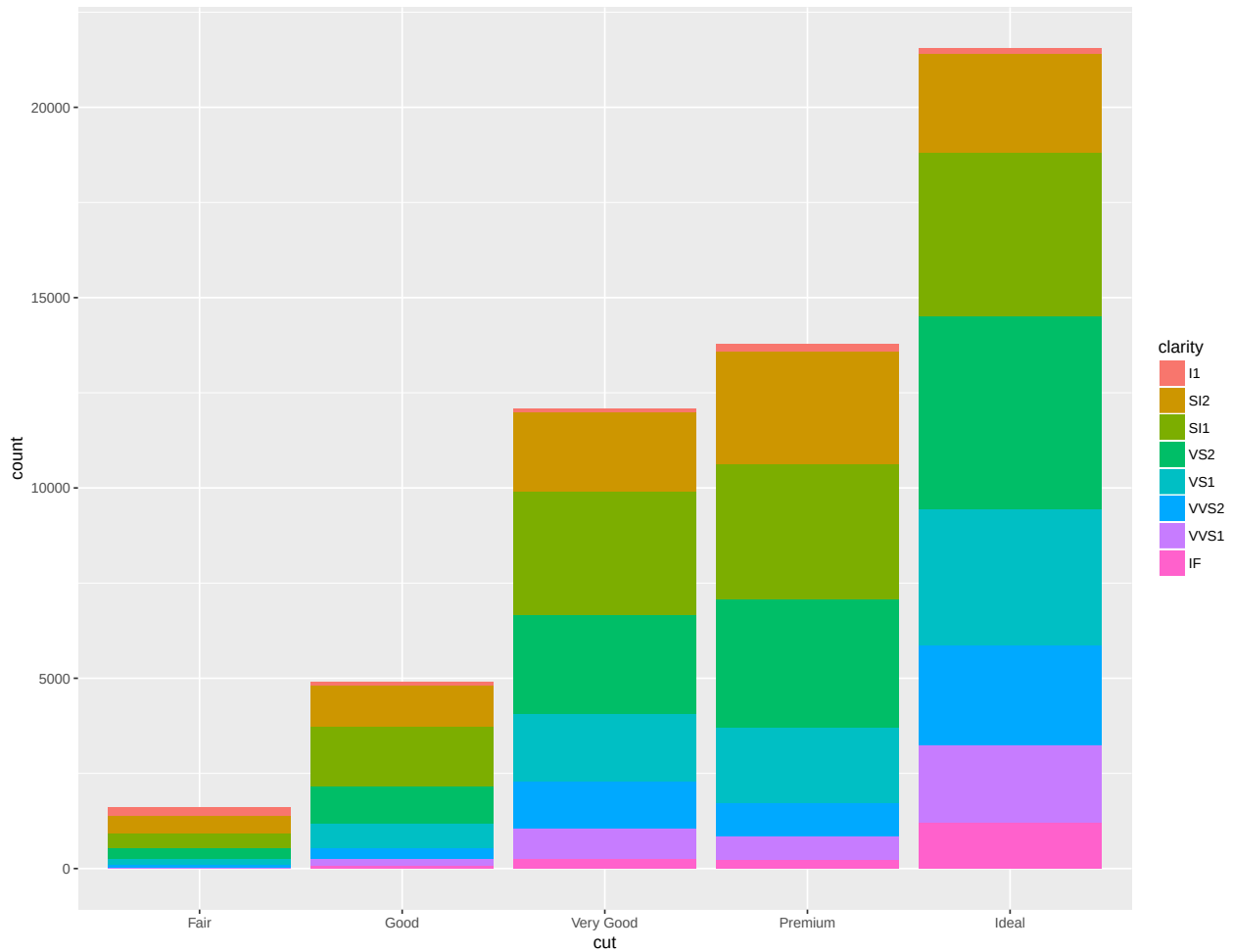
geom = "jitter" [EXAMPLE 2/2]

```
diamonds %<>% mutate(pricePerCarat = price / carat)
myDiamonds %>% ggplot(aes(x = color, y = pricePerCarat)) +
                geom_point(position = "jitter")
```



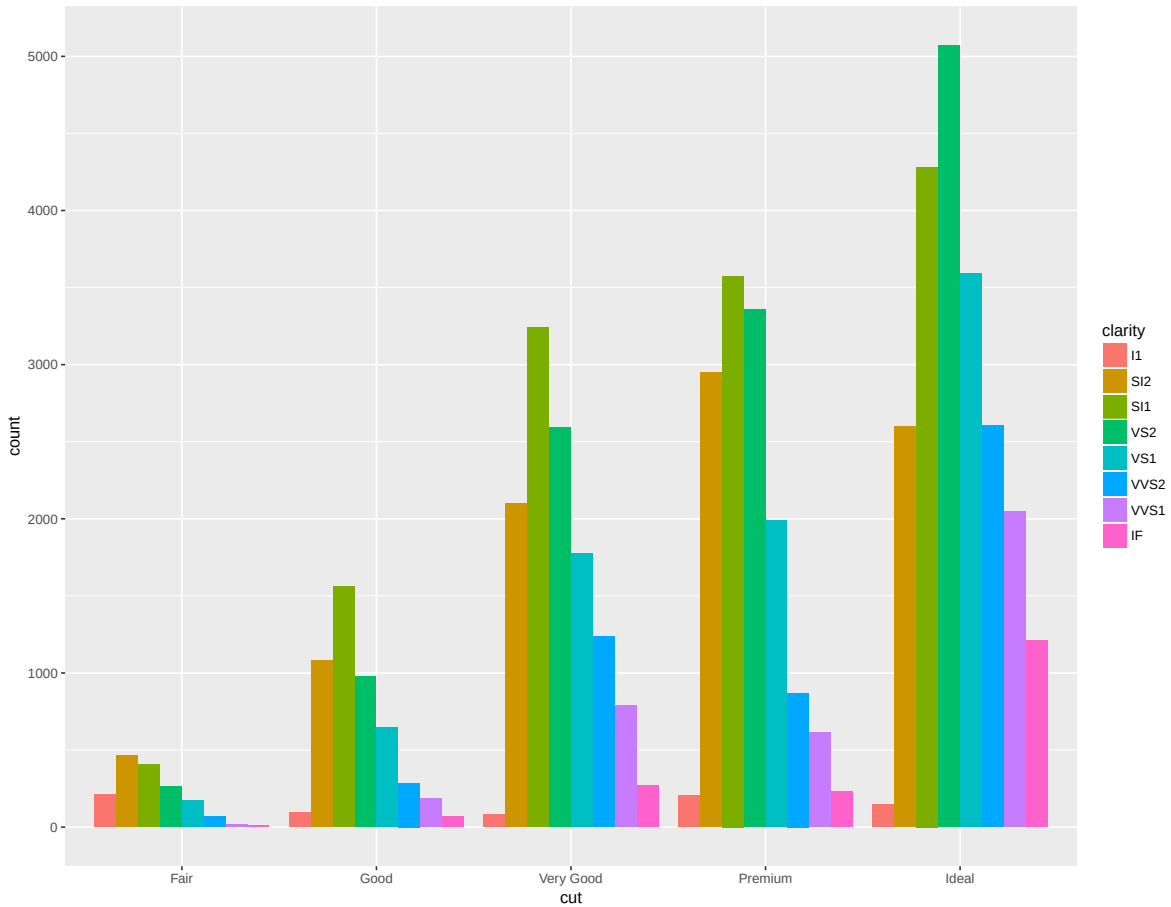
Useful Plots [1/2]

```
ggplot2::diamonds %>% ggplot() + geom_bar(aes(x = cut, fill = clarity))
```



Useful Plots [2/2]

```
ggplot2::diamonds %>% ggplot() + geom_bar(aes(x = cut, fill = clarity),
                                          position = "dodge")
```

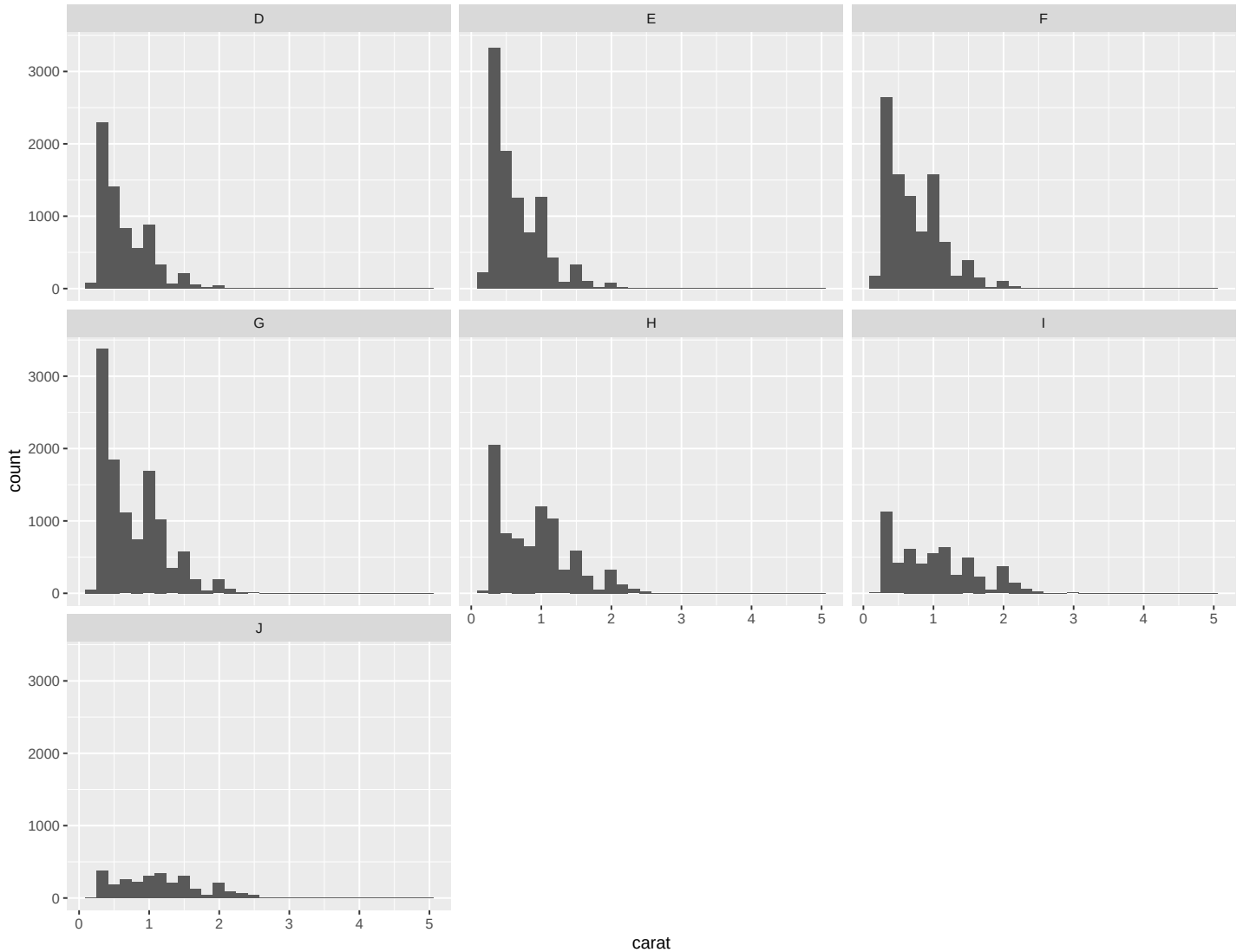


Faceting

- 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
- The following code generates the graph on the proceeding slide

```
> diamonds %>% ggplot() + geom_histogram(aes(carat)) + facet_wrap(~ color)
```

Faceting [CONT'D]



Grouping

- geoms can be roughly divided into two groups: individual and collective
- An individual geom has a distinctive graphical object for each observations in a data frame, e.g., `geom_point()` has single point for each observation
- Collective geoms 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

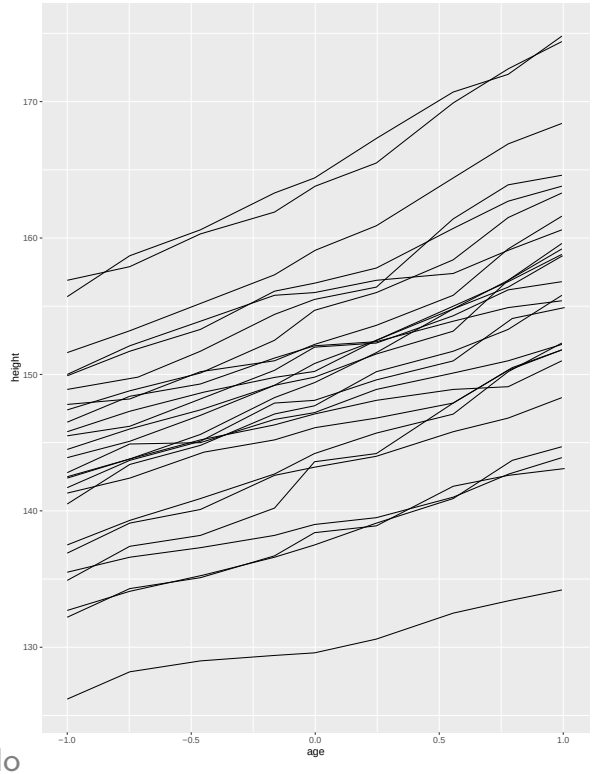
Grouping [CONT'D]

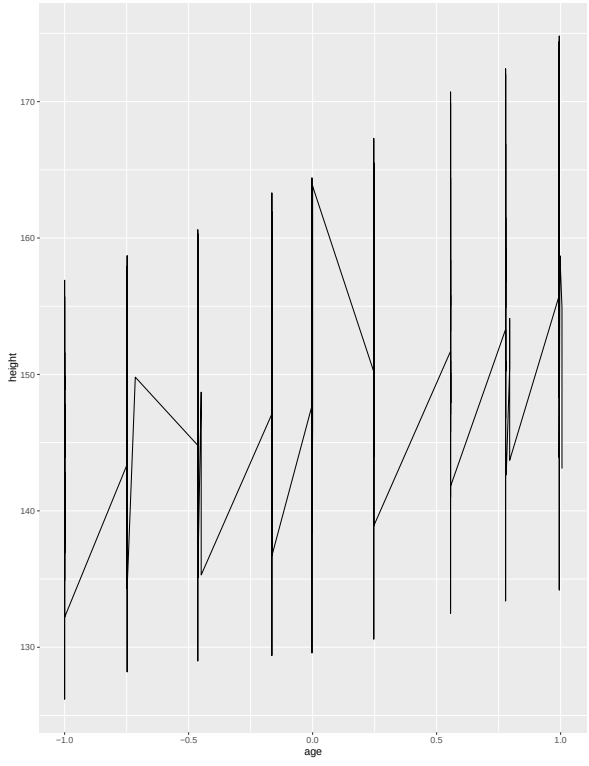
- Be default, the `group` is set to the interaction of all discrete variables in the plot, which typically generates the expected results
- In the event it does not (or when no discrete variables are used in the plot), the `group` can be mapped to a variable that has a different value for each group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

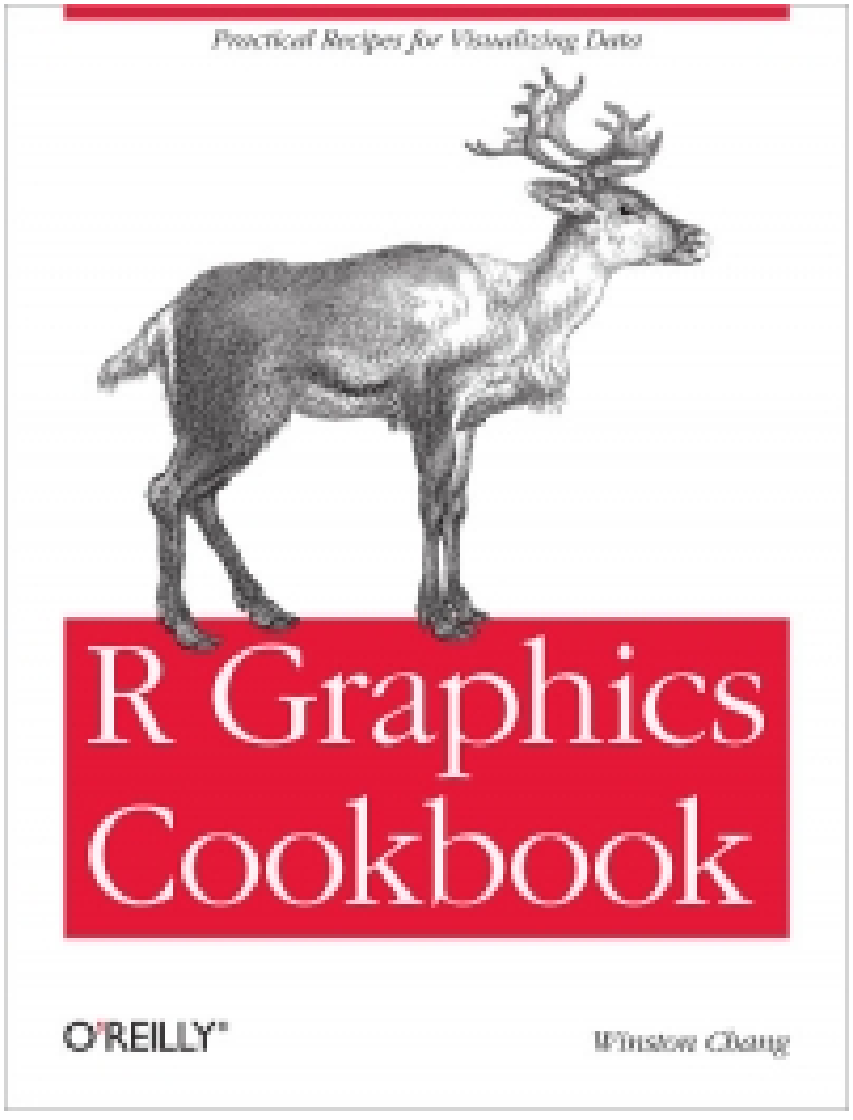
Multiple Groups / One Aesthetic

- Objective: separate the data into groups in an effort to distinguish between individual subjects, but render all of them in the same way

LEFT
```
myPlot <- ggplot(nlme::Oxboys, aes(age, height, group = Subject)) + geom_line()
```
RIGHT
```
myPlot <- ggplot(nlme::Oxboys, aes(age, height)) + geom_line()
```



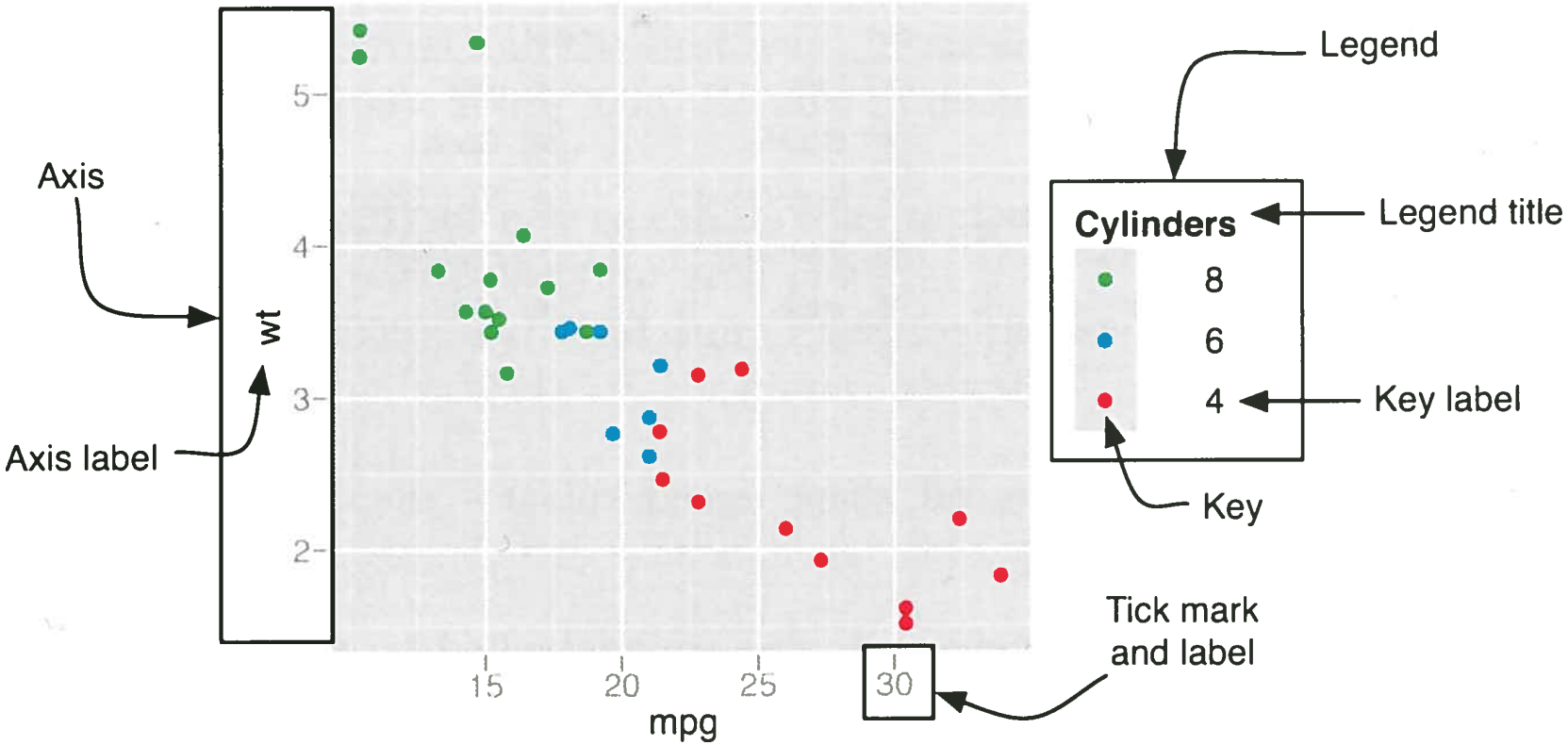



Practical Recipes for Visualizing Data
R Graphics Cookbook
O'REILLY
Winston Chang

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



Notes

- 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 $15,000,0000,000 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