# Exploring data with R

IMT 573A - Data Science 1 - Theoretical Foundations

6-Oct-2020 (Week 2, Day 3)

Our Zoom class sessions will be recorded.

# Today's topic

- Review - checklist of exploratory data analysis
- Markdown and Rmarkdown
- Data visualization for exploring data
- ggplot basics
- Lab2

DUE

- Lab2: upload by tomorrow (Oct 9) 9am PT
- PS1: deadline next Tuesday (Oct 13) 9am PT

# Exploring Data: A checklist

1. Interpret your data
2. Formulate your question
3. Read in your data
4. Examine your data, look at the top and bottom of your data, look at structure
5. Tidy data (Data cleaning)
6. Try the easy solution first
7. Challenge your solution, validate with external data
8. Follow up with new interesting questions/directions

**Think about your data! Think about your question!**

Peng, Roger D. Exploratory Data Analysis with R

# RMarkdown

## Writing in Markdwon

Markdown is a lightweight markup language that is used to format and structure text. It is a kind of “code” that you write in order to *annotate* plain text

```
This is a paragraph in which we'll add
**bold text**, _italicized text_, and `code`
into the middle of a sentence
```

This is a paragraph in which we'll add **bold text**, *italicized text*, and `code` into the middle of a sentence

| Syntax | Formatting |
|---|---|
| `_text_` | *italicized* using underscores ( `_` ) |
| `**text**` | **bolded** using two asterisks ( `*` ) |
| `` `text` `` | inline `code` with backticks ( `` ` `` ) \| \| ~~text~~ \| `~~strike-through~~` using tildes ( ~`) |

# Markdown

## Writing in Markdown

### Text Blocks

```
# Top Level header
## Second Level Header

Here is a normal paragraph

- List item 1
- List item 2
- List item 3

```
block of code
across multiple lines
```

> Here is a block quote
```

# Top Level header

## Second Level Header

Here is a normal paragraph

- List item 1
- List item 2
- List item 3

```
block of code
across multiple lines
```

> Here is a block quote

| Syntax | Formatting |
|---|---|
| `#` | Header (use `##` for 2nd-level, `###` for 3rd, etc.) |
| ` ``` ` | Code section (3 back ticks) that encapsulate the code |
| `-` | Bulleted/unordered lists (hyphens) |
| `>` | Block quote |

```
```{r import titanic data}
# Let's write some R code here
```
```

**Markdown can even make tables, create hyperlinks, and include images!**

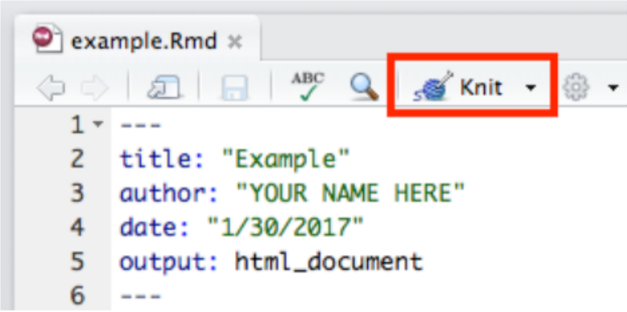
# Rmarkdown (review)

## `.Rmd` Content

### Header information

```
---
title: "Example"
author: "YOUR NAME HERE"
date: "1/30/2017"
output: html_document
---
```

### RStudio knitting




RStudio's Knit button

### R code chunks

```
```{r}
# Execute R code in here
x <- 201
```
```

### Markdown lists

```
```{r echo=FALSE}
markdown.list <- "
- Lions
- Tigers
- Bears
"
```

`r markdown.list`
```

Would output a list that looks like:

- Lions
- Tigers
- Bears

### Rendering Tables

```
### Including Tables
```{r echo=FALSE}
library(knitr)  # make sure you load this library (once per doc)

# make a data frame
letters <- c("a", "b", "c")
numbers <- 1:3
df <- data.frame(letters = letters, numbers = numbers)

# render the table
kable(df)
```
```

```
|letters | numbers|
|:-------|-------:|
|a       |       1|
|b       |       2|
|c       |       3|
```

https://rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf

# R Markdown Cheat Sheet

learn more at rmarkdown.rstudio.com

RStudio

### .Rmd files

An R Markdown (.Rmd) file is a record of your research. It contains the code that a scientist needs to reproduce your work along with the narration that a reader needs to understand your work.

### Reproducible Research

At the click of a button, or the type of a command, you can rerun the code in an R Markdown file to reproduce your work and export the results as a finished report.

### Dynamic Documents

You can choose to export the finished report as a html, pdf, MS Word, ODT, RTF, or markdown document; or as a html or pdf based slide show.

## Workflow

**1 Open a new .Rmd file** at File ▸ New File ▸ R Markdown. Use the wizard that opens to pre-populate the file with a template

**2 Write document** by editing template

**3 Knit document to create report** Use knit button or **render()** to knit

**4 Preview Output** in IDE window

**5 Publish** (optional) to web or server

Synch publish button to accounts at
- rpubs.com,
- shinyapps.io
- RStudio Connect

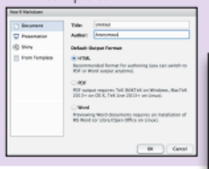
Reload document

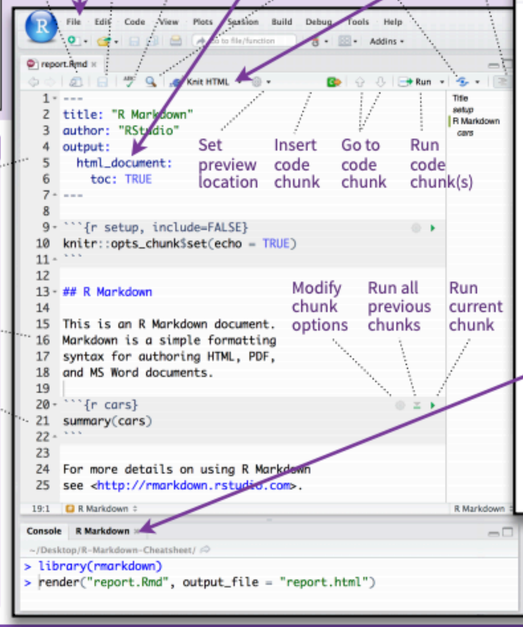
Find in document

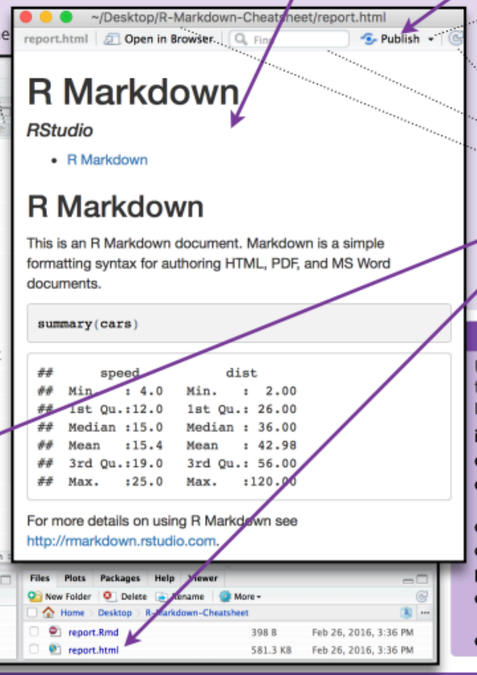
File path to output document

**6 Examine build log** in R Markdown console

**7 Use output file** that is saved alongside .Rmd






### .Rmd structure

**YAML Header**
Optional section of render (e.g. pandoc) options written as key:value pairs (YAML).
- At start of file
- Between lines of ---

**Text**
Narration formatted with markdown, mixed with:

**Code chunks**
Chunks of embedded code. Each chunk:
- Begins with ```{r}
- ends with ```

R Markdown will run the code and append the results to the doc.

It will use the location of the .Rmd file as the **working directory**

### render()

Use rmarkdown::**render()** to render/knit at cmd line. Important args:

**input** - file to render
**output_format**
**output_options** - List of render options (as in YAML)
**output_file**
**output_dir**
**params** - list of params to use
**envir** - environment to evaluate code chunks in
**encoding** - of input file

## Interactive Documents

Turn your report into an interactive Shiny document in 4 steps

1. Add **runtime: shiny** to the YAML header.
2. Call Shiny **input** functions to embed input objects.
3. Call Shiny **render** functions to embed reactive output.
4. Render with rmarkdown::**run** or click **Run Document** in RStudio IDE

```
---
output: html_document
runtime: shiny
---

```{r, echo = FALSE}
numericInput("n",
  "How many cars?", 5)

renderTable({
  head(cars, input$n)
})
```
```

How many cars? 5

| | speed | dist |
|---|---|---|
| 1 | 4.00 | 2.00 |
| 2 | 4.00 | 10.00 |
| 3 | 7.00 | 4.00 |
| 4 | 7.00 | 22.00 |
| 5 | 8.00 | 16.00 |

Embed a complete app into your document with shiny::**shinyAppDir()**

** Your report will rendered as a Shiny app, which means you must choose an html output format, like **html_document**, and serve it with an active R Session.*

## Embed code with knitr syntax

### Inline code

Insert with `` `r <code>` ``. Results appear as text without code.

```
Built with
`r getRversion()`
```
→ Built with 3.2.3

### Code chunks

One or more lines surrounded with ```{r} and ```. Place chunk options within curly braces, after **r**. Insert with

```{r echo=TRUE}
getRversion()
```
→
```
getRversion()
## [1] '3.2.3'
```

### Global options

Set with knitr::**opts_chunk$set()**, e.g.

```{r include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

### Important chunk options

**cache** - cache results for future knits (default = FALSE)

**cache.path** - directory to save cached results in (default = "cache/")

**child** - file(s) to knit and then include

**dependson** - chunk dependencies for caching (default = NULL)

**echo** - Display code in output document (default = TRUE)

**engine** - code language used in chunk

**fig.align** - 'left', 'right', or 'center' (default = 'default')

**fig.cap** - figure caption as character string (default = NULL)

**fig.height, fig.width** - Dimensions of

**message** - display code messages in document (default = TRUE)

**results** (default = 'markup')
'asis' - passthrough results
'hide' - do not display results

## Parameters

Parameterize your documents to reuse with different inputs (e.g., data sets, values, etc.)

**1 Add parameters** Create and set parameters in the header as sub-values of **params**

```
---
params:
  n: 100
  d: !r Sys.Date()
---
```
Indent 2 spaces

**2 Call parameters** Call parameter values in code as **params$<name>**

```
Today's date
is `r params$d`
```

## Pandoc's Markdown

Write with syntax on the left to create effect on right (after render)

```
Plain text

End a line with two spaces
to start a new paragraph.

*italics* and **bold**
`verbatim code`
sub/superscript^2^~2~
~~strikethrough~~
escaped: \* \_ \\
endash: --, emdash: ---
equation: $A = \pi*r^{2}$
equation block:

$$E = mc^{2}$$

> block quote

# Header1  {#anchor}

## Header 2  {#css_id}

### Header 3  {.css_class}

#### Header 4

##### Header 5

###### Header 6

<!--Text comment-->

\textbf{Tex ignored in HTML}
<em>HTML ignored in pdfs</em>

<http://www.rstudio.com>
[link](www.rstudio.com)
Jump to [Header 1](#anchor)
image:

![Caption](smallorb.png)

* unordered list
    + sub-item 1
    + sub-item 2
        - sub-sub-item 1

* item 2

    Continued (indent 4 spaces)

1. ordered list
2. item 2
        i) sub-item 1
            A.  sub-sub-item 1

(@)  A list whose numbering
continues after

(@)  an interruption

Term 1

:   Definition 1

| Right | Left | Default | Center |
|------:|:-----|---------|:------:|
|    12 |   12 |      12 |     12 |
|   123 |  123 |     123 |    123 |
|     1 |    1 |       1 |      1 |

- slide bullet 1
- slide bullet 2

(>- to have bullets appear on click)

horizontal rule/slide break:

***

A footnote [^1]

[^1]: Here is the footnote.
```

Plain text
End a line with two spaces to start a new paragraph.
*italics* and **bold**
`verbatim code`
sub/superscript$^2$$_2$
~~strikethrough~~
escaped: * _ \
endash: –, emdash: —
equation: $A = \pi * r^2$
equation block:

$$E = mc^2$$

> block quote

# Header1

## Header 2

### Header 3

#### Header 4

##### Header 5

###### Header 6

*HTML ignored in pdfs*

http://www.rstudio.com
link
Jump to Header 1
image:

Caption

- unordered list
  - sub-item 1
  - sub-item 2
    - sub-sub-item 1
- item 2

  Continued (indent 4 spaces)

1. ordered list
2. item 2
   i. sub-item 1
      A. sub-sub-item 1

1. A list whose numbering continues after
2. an interruption

Term 1

Definition 1

| Right | Left | Default | Center |
|---|---|---|---|
| 12 | 12 | 12 | 12 |
| 123 | 123 | 123 | 123 |
| 1 | 1 | 1 | 1 |

- slide bullet 1
- slide bullet 2

(>- to have bullets appear on click)

horizontal rule/slide break:

A footnote [1]

1. Here is the footnote. ↩

# Data visualization basics (needed for exploring data)

Human brain is exceptionally good at visual pattern

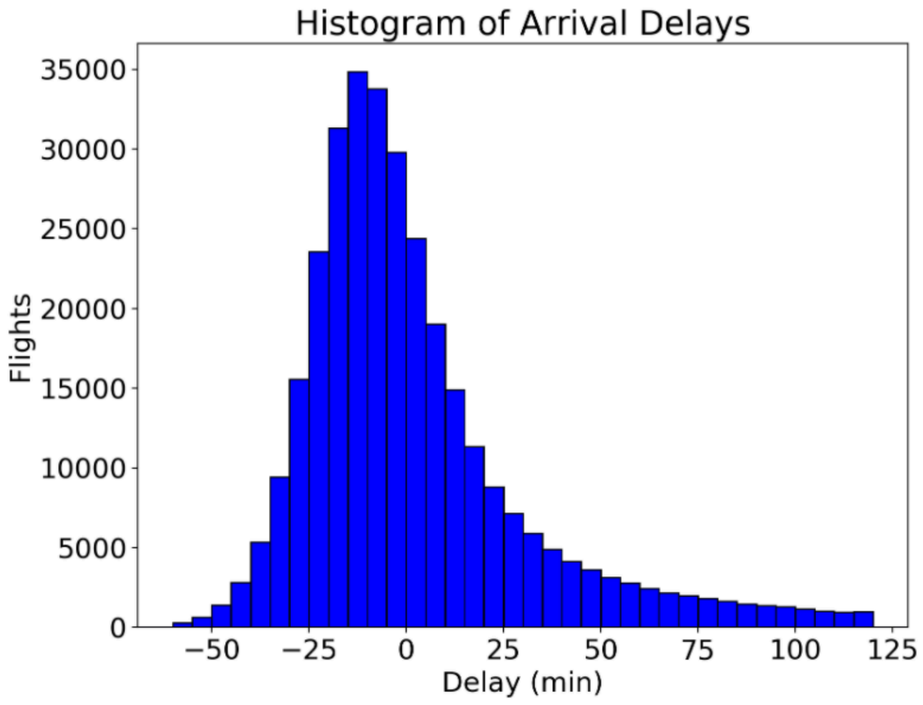
Map dimensions of data to visual characteristics

- Location
- Size/Length
- Color
- Shape

# Common visualizations for low dimensional data (1D/2D)

**Univariate** - single variable, 1D data

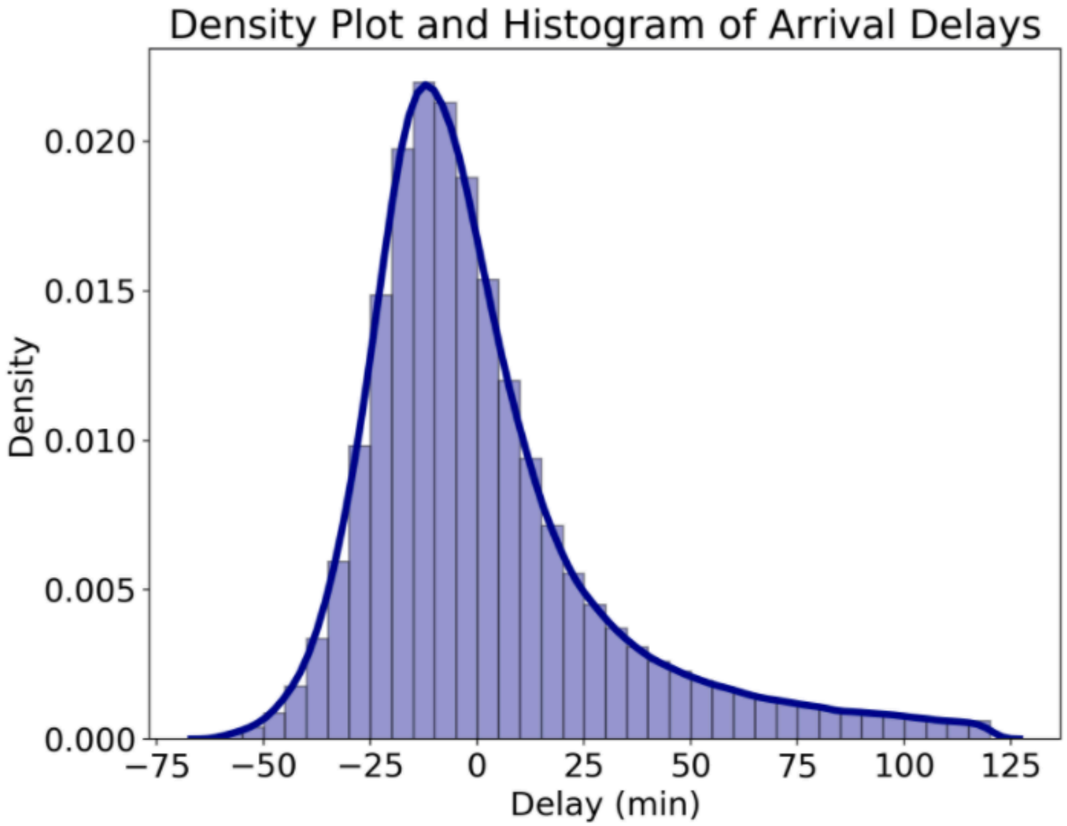
Visualization — Histograms/KDE



# Density plot (KDE)

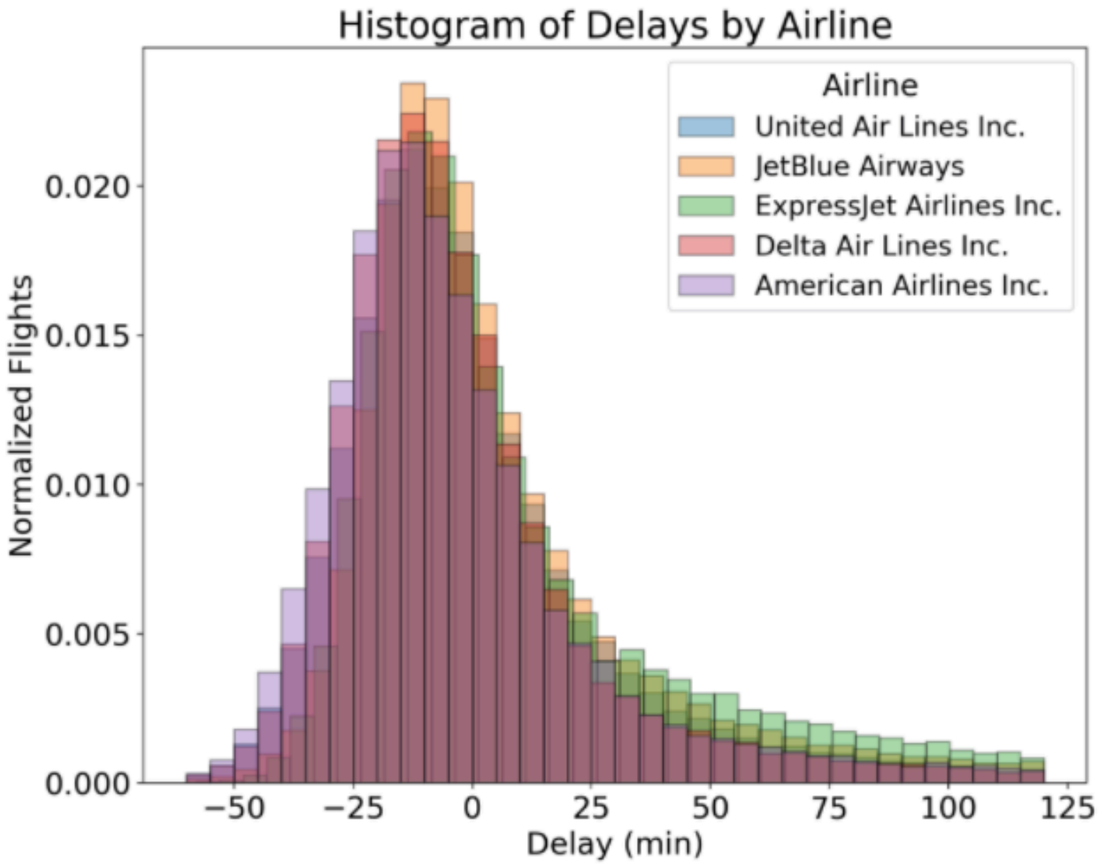
A smoothed, continuous version of a histogram estimated from data.

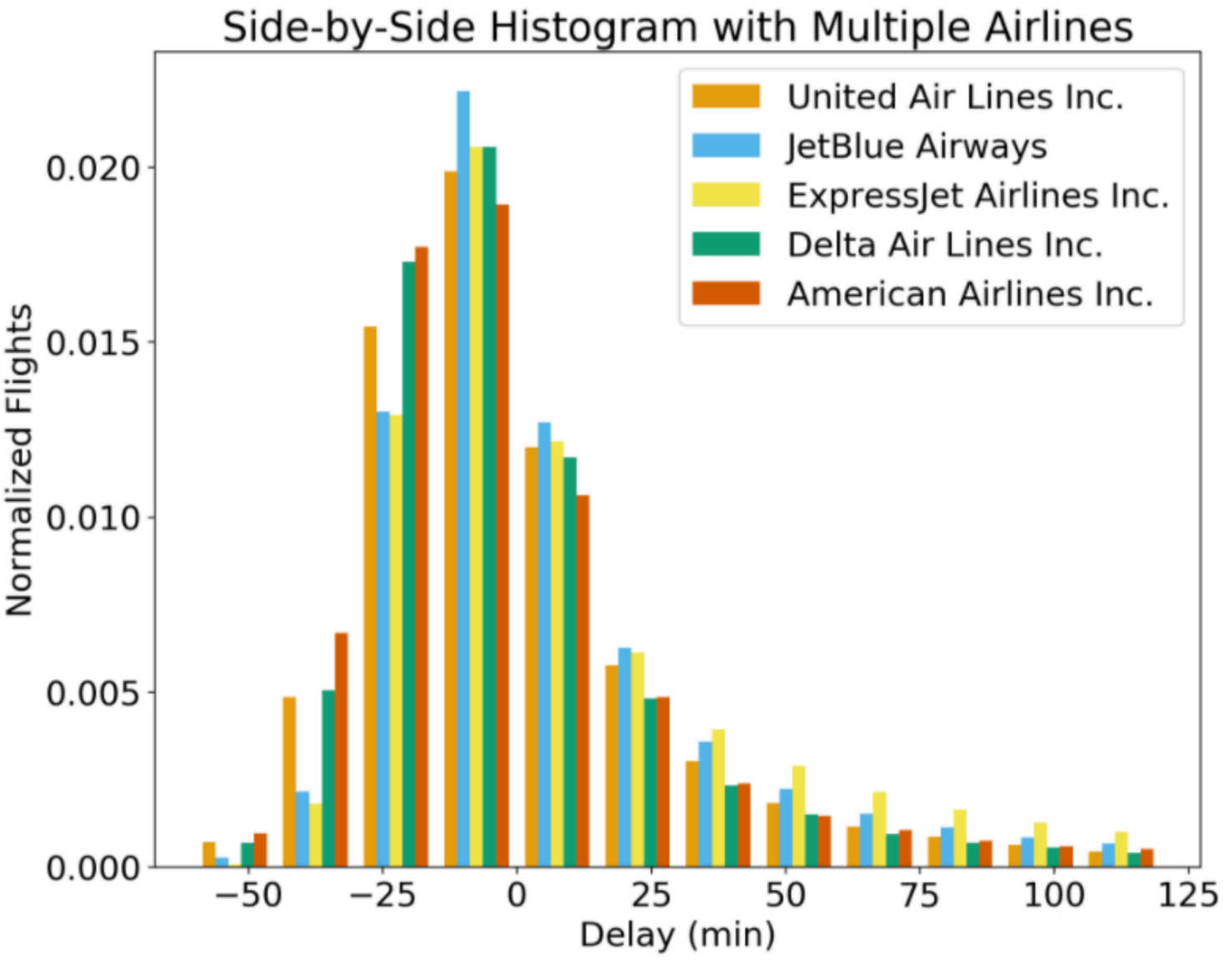
**KDE** - most common form of estimation
a continuous curve (the kernel) is drawn at individual data point and all of these curves are then added together to make a single smooth density estimation.



# Histograms & KDE

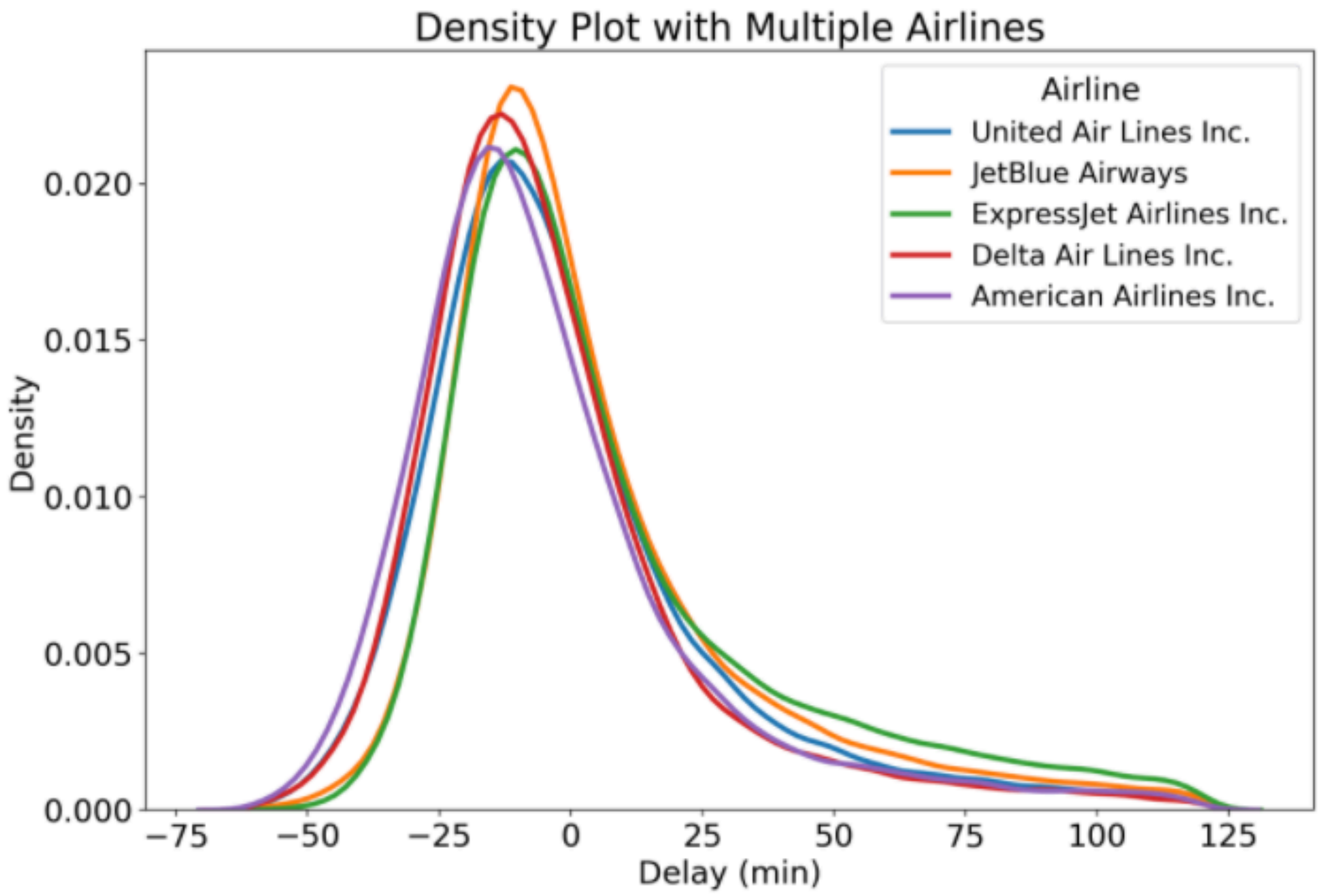
**When histograms fail** — Histograms are great for exploring single variable from one category, but not great for a single variable across multiple categories.




Side-by-Side Histogram with Multiple Airlines
United Air Lines Inc.
JetBlue Airways
ExpressJet Airlines Inc.
Delta Air Lines Inc.
American Airlines Inc.
Normalized Flights
0.000
0.005
0.010
0.015
0.020
−50
−25
0
25
50
75
100
125
Delay (min)

# Density plot

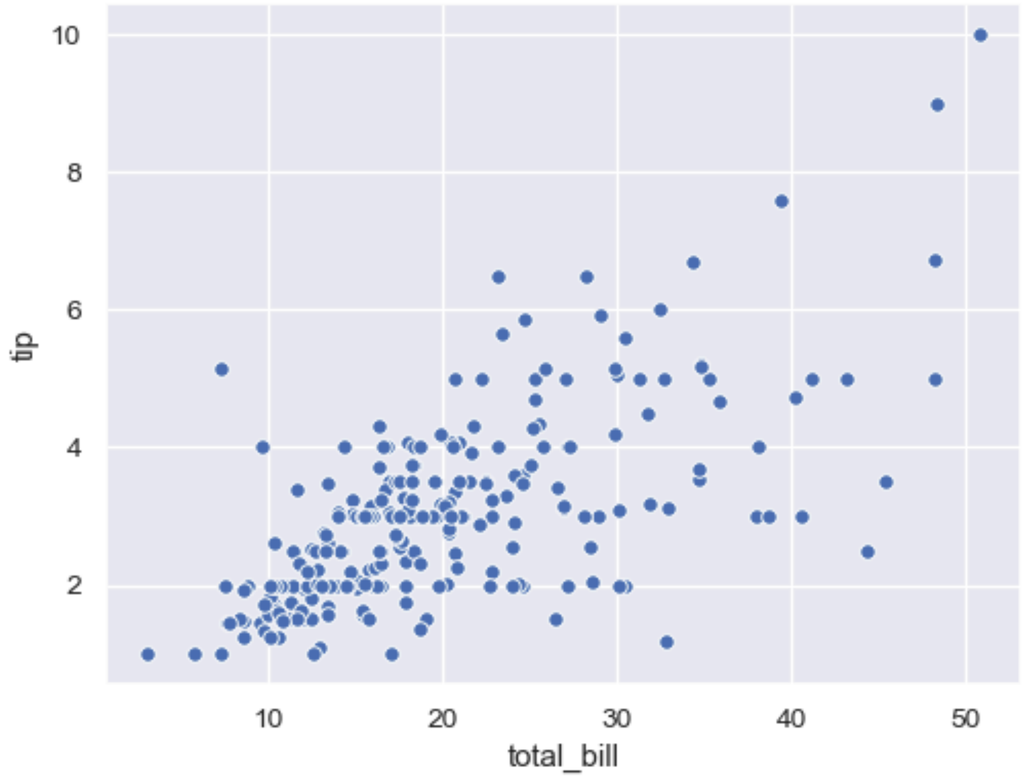
A smoothed, continuous version of a histogram estimated from the data.



# Common visualizations for low dimensional data (1D/2D)

**Bivariate** - 2 variables.

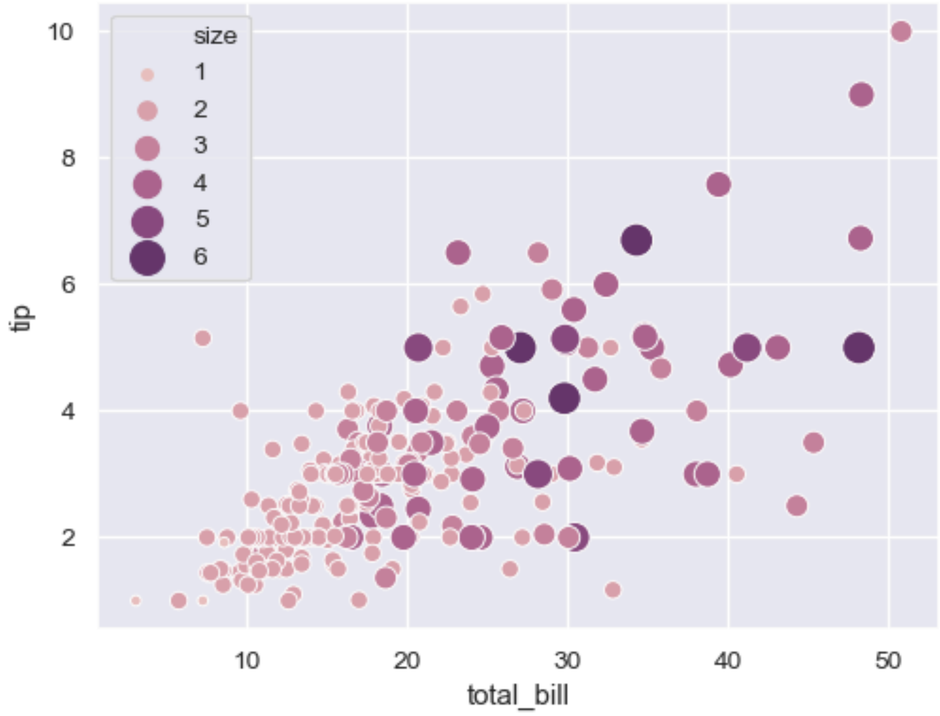
**Scatterplot** - Qualitative (continuous) x Quant (continuous), correlation



# Common visualizations for low dimensional data (1D/2D)

**Bivariate** - 2 variables.

**Scatterplot** - Qualitative (continuous) x Quant (continuous), correlation

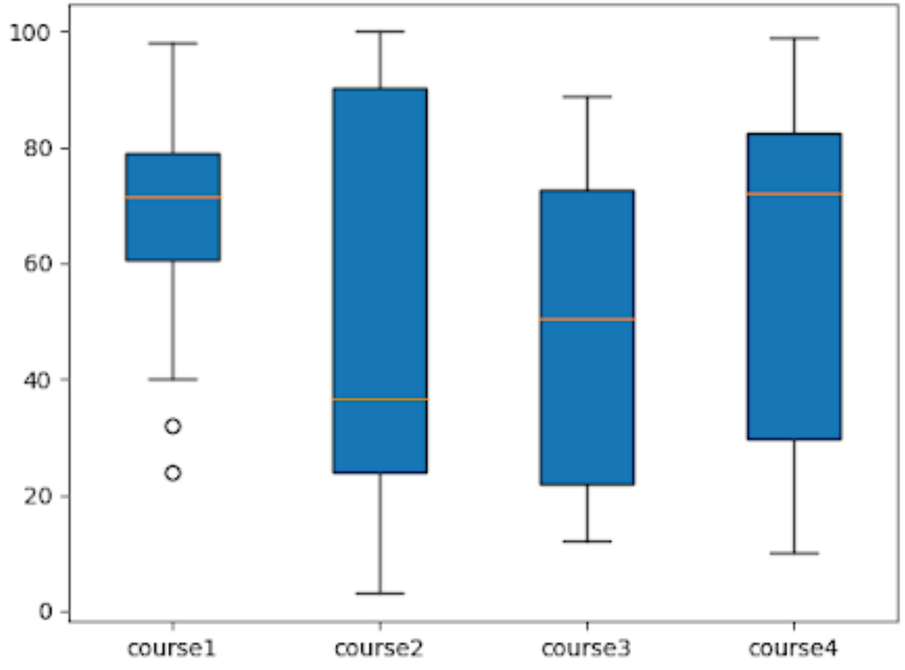


# Common visualizations for low dimensional data (1D/2D)

**Bivariate** - 2 variables.

**Boxplot -** Continuous  x Categorical

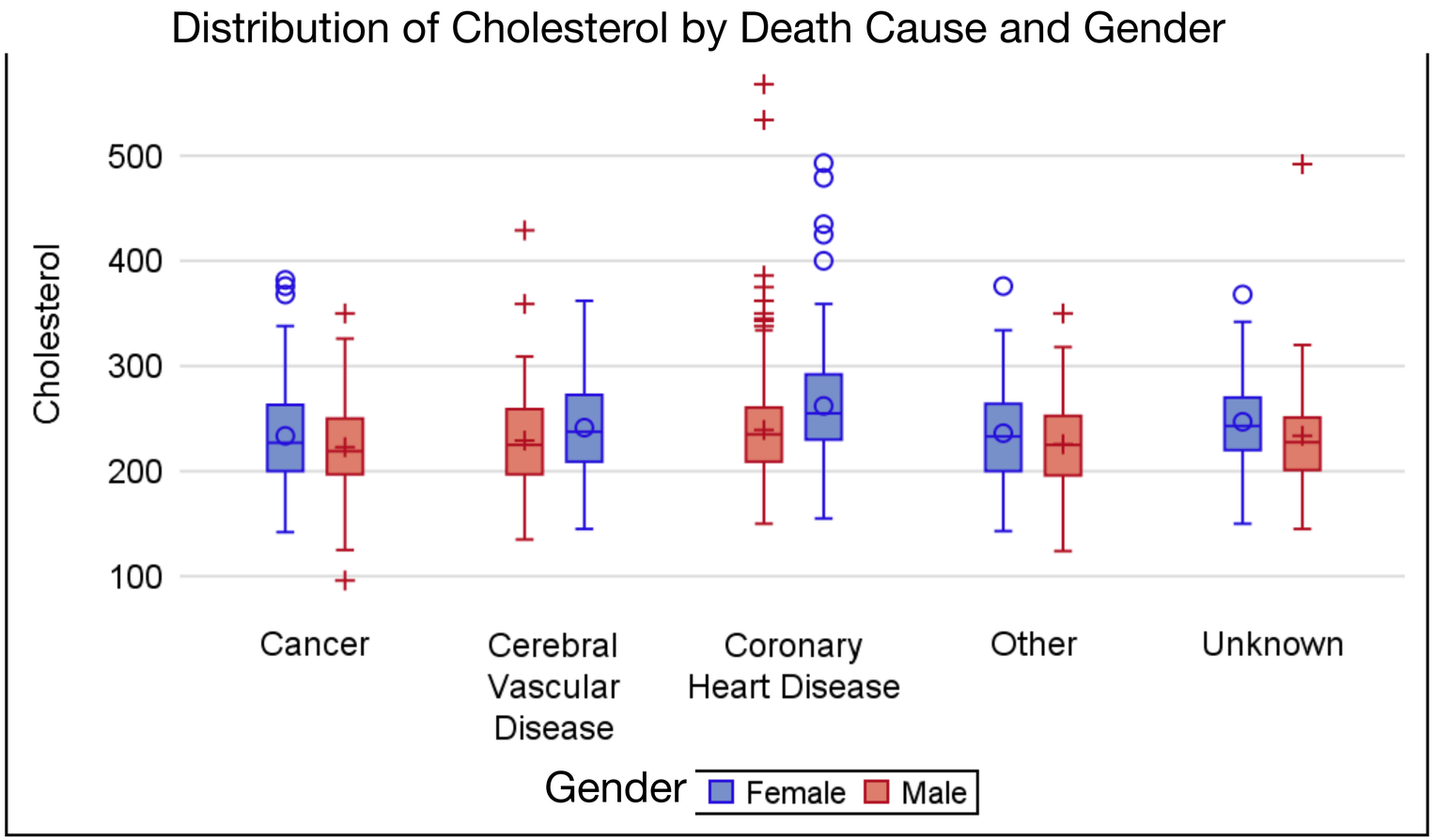
**Grade** Distribution by Course



# Common visualizations for low dimensional data (1D/2D)

**Bivariate** - 2 variables.

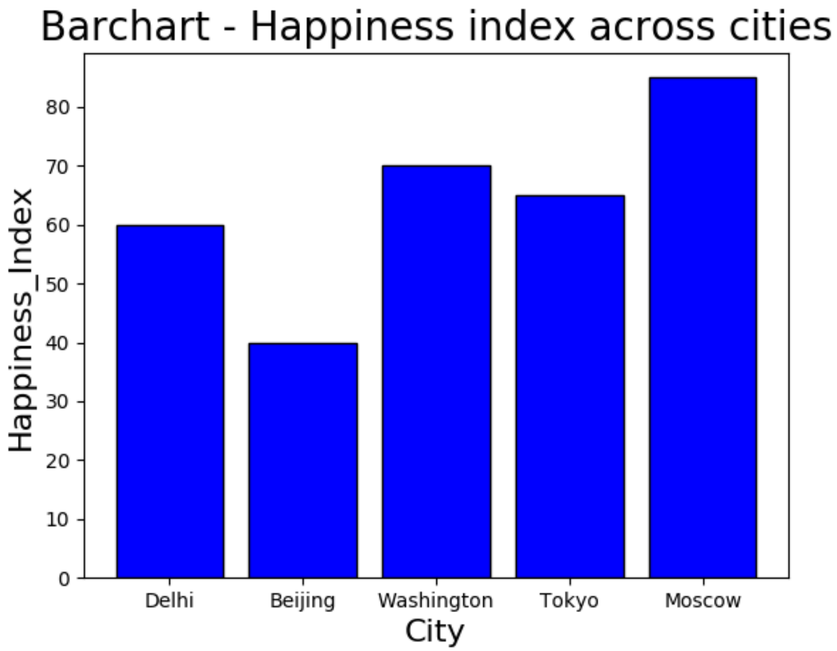
**Boxplot -** Continuous x Categorical

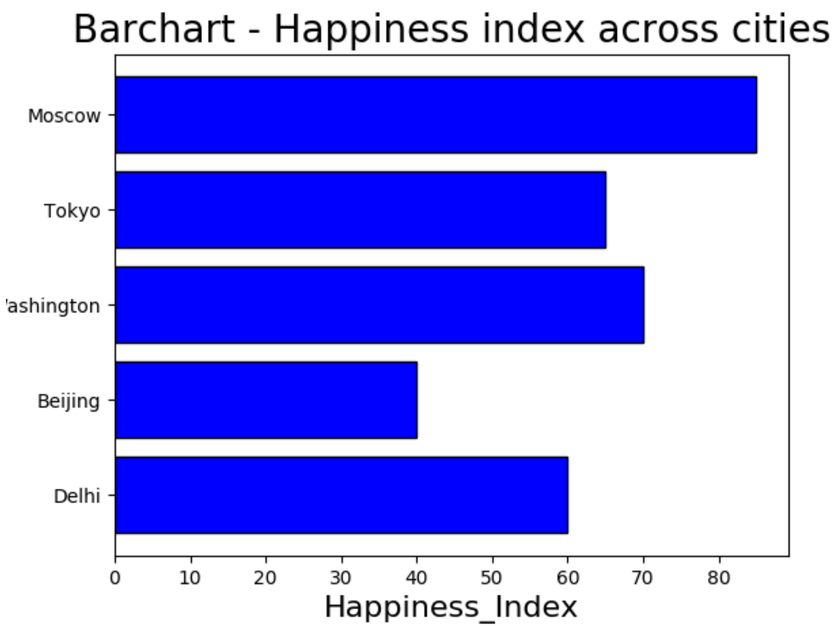


# Common visualizations for low dimensional data (1D/2D)

**Bivariate** - 2 variables.

**bar chart -** Continuous  x Categorical

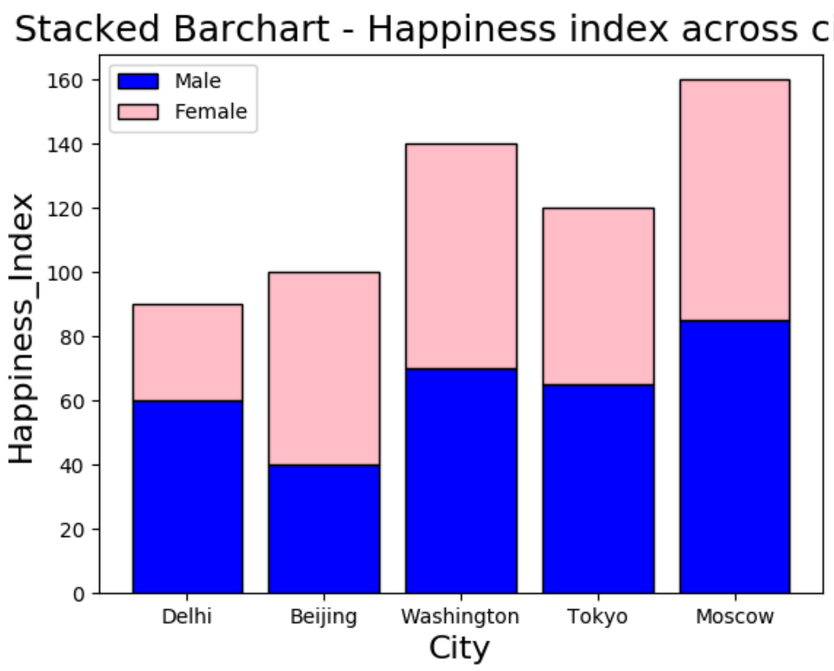


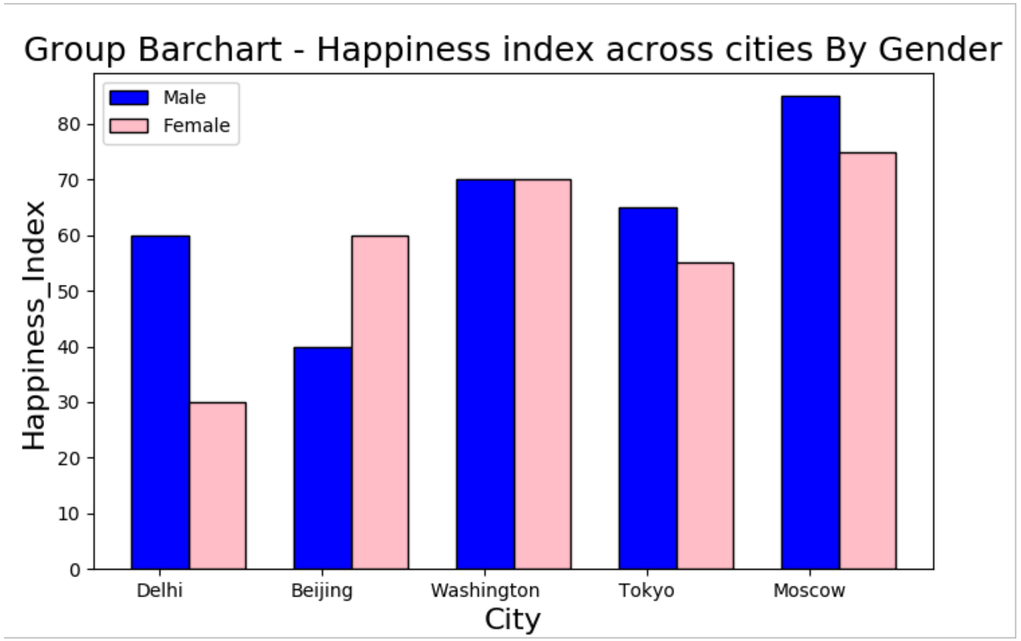



# Common visualizations for low dimensional data (1D/2D)

**Bivariate** - 2 variables.

**bar chart -** Continuous x Categorical



# Data visualization basics (needed for exploring data)

Human brain is exceptionally good at visual pattern

- Time-series: A single variable is captured over a period of time - line chart
- Ranking: Categorical subdivisions are ranked in ascending or descending order - bar chart
- Part-to-whole: Categorical subdivisions are measured as a ratio to the whole (i.e., a percentage out of 100%) - pie chart
- Deviation: Categorical subdivisions are compared again a reference - bar chart
- Frequency distribution: Shows the number of observations of a particular variable for given interval - histogram
- Correlation: Comparison between observations represented by two variables (X,Y) to determine if they tend to move in the same or opposite directions - scatter plot
- Nominal comparison: Comparing categorical subdivisions in no particular order - bar chart
- Geographic or geospatial: Comparison of a variable across a map - cartogram

# Code to visual - ggplot

# Basic plotting

ggplot basics

ggplot2 is now over 10 years old and is used by hundreds of thousands of people to make millions of plots!

It is a system for creating graphics, based on The Grammar of Graphics.

## Installation

```
# The easiest way to get ggplot2 is to install the whole tidyverse:
install.packages("tidyverse")

# Alternatively, install just ggplot2:
install.packages("ggplot2")
```

http://r-statistics.co/ggplot2-Tutorial-With-R.html#1.%20The%20Setup

# Basic plotting

ggplot basics

ggplot2 is now over 10 years old and is used by hundreds of thousands of people to make millions of plots!

**Setup:** First tell ggplot what data to use.

## Installation

```
# The easiest way to get ggplot2 is to install the whole tidyverse:
install.packages("tidyverse")

# Alternatively, install just ggplot2:
install.packages("ggplot2")
```

http://r-statistics.co/ggplot2-Tutorial-With-R.html#1.%20The%20Setup

# Basic plotting ggplot basics

**Setup:**

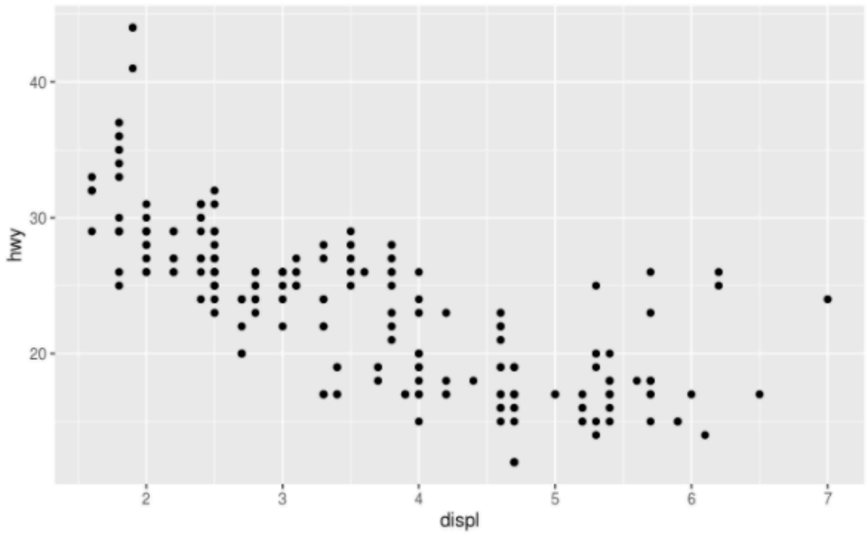
- First tell ggplot what data to use.

- Add whatever aesthetics you want to apply to your ggplot (inside `aes()` argument) - such as X and Y axis by specifying the respective variables from the dataset.

- The variable based on which the color, size, shape and stroke should change can also be specified

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```

displ: displacement — represents engine size
hwy: highway miles per gallon — represents Fuel efficiency




What does the plot tell you?
Can you figure out how X and Y are related?

http://r-statistics.co/ggplot2-Tutorial-With-R.html#1.%20The%20Setup

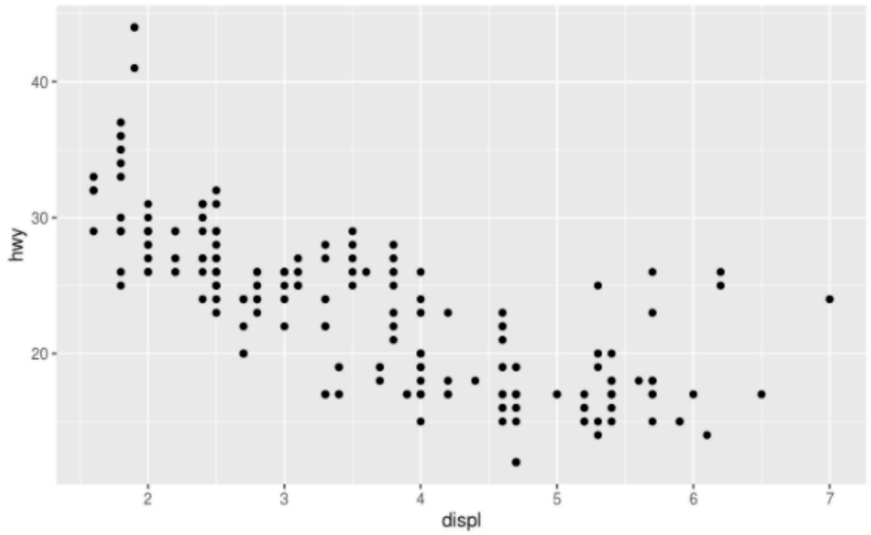
# A graphing template

R for Data Science: https://r4ds.had.co.nz/data-visualisation.html

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Understanding the `<MAPPINGS>` component.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```




A **`geom`** is the geometrical object that a plot uses to represent data. Here we are using **`point`** geom. (bar charts use **bar** geoms, line charts use **line** geoms, **boxplots** use boxplot geoms)

Each geom function takes a **`mapping`** argument. This defines how variables in your dataset are mapped to visual properties.

The **`mapping`** argument is always paired with `aes()`, and the `x` and `y` arguments of `aes()` specify which variables to map to the x and y axes.

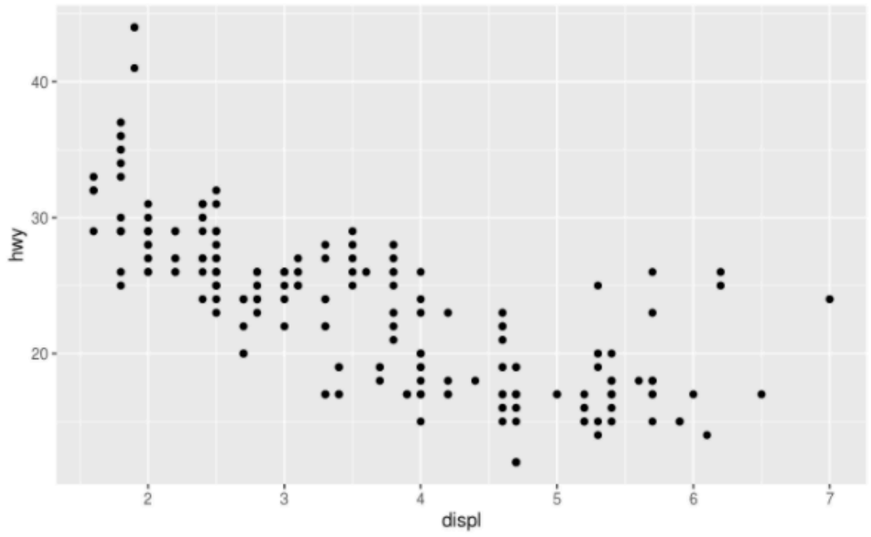
# A graphing template

R for Data Science: https://r4ds.had.co.nz/data-visualisation.html

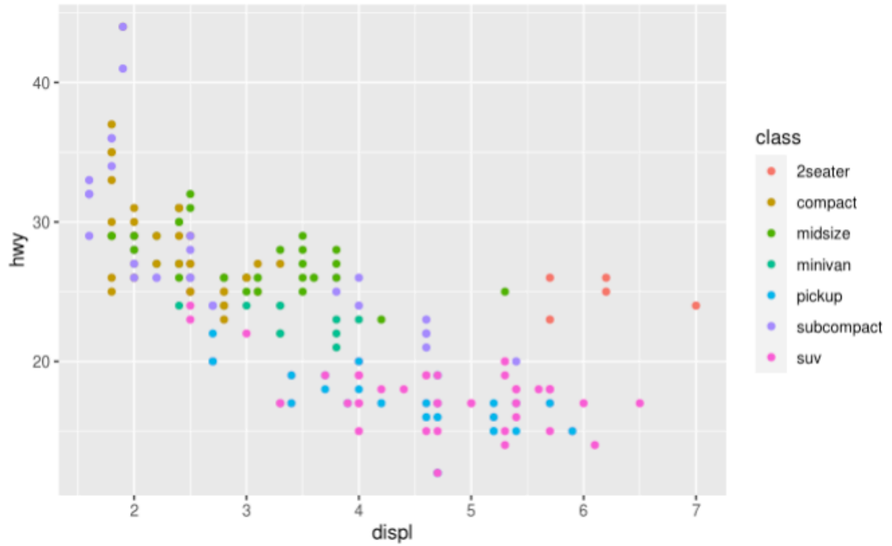
```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Understanding the `<MAPPINGS>` component.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```




```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



# A graphing template

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

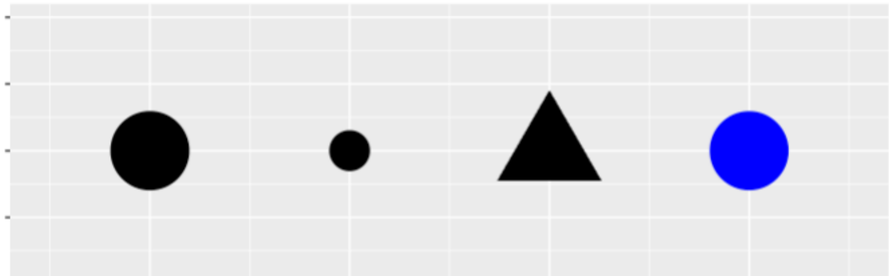
Understanding the `<MAPPINGS>` component.

**Aesthetic mappings**

A visual property of the objects in your plot.
Aesthetics include:

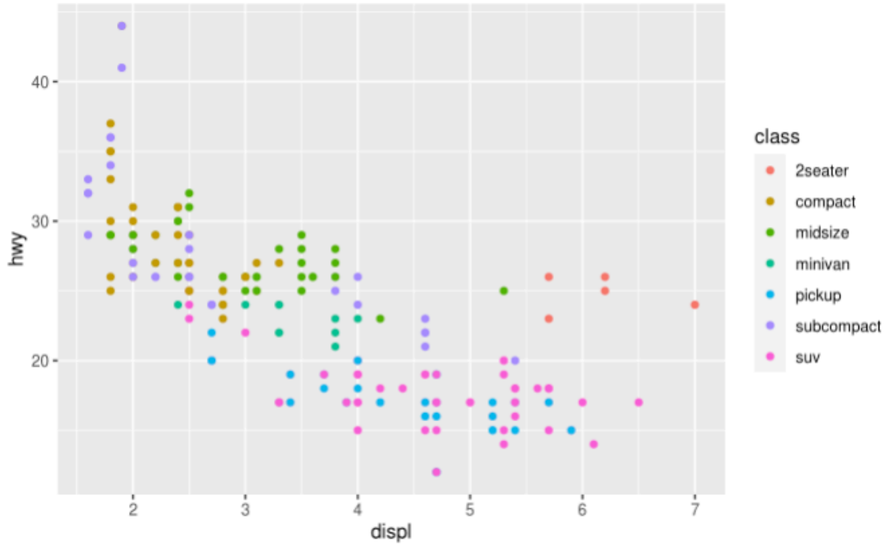
- size,
- shape, or
- color of your points.

Display a point in different ways by changing the values of its aesthetic properties.



Mapping a variable to the **color** aesthetic

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



# A graphing template

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

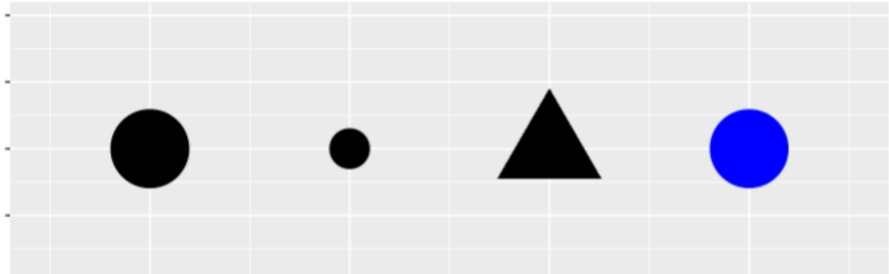
Understanding the `<MAPPINGS>` component.

**Aesthetic mappings**

A visual property of the objects in your plot.
Aesthetics include:

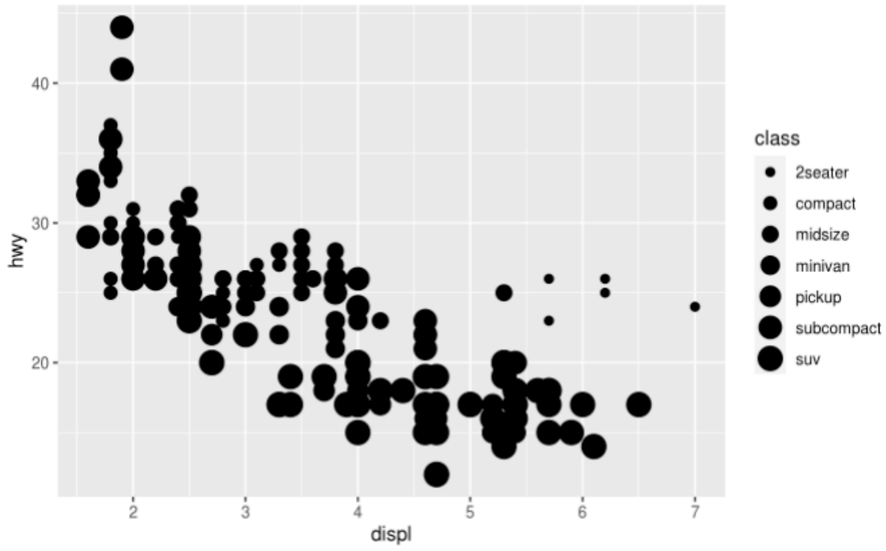
- size,
- shape, or
- color of your points.

Display a point in different ways by changing the values of its aesthetic properties.



Mapping a variable to the **size** aesthetic

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, size = class))
#> Warning: Using size for a discrete variable is not advised.
```



# A graphing template

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Understanding the `<MAPPINGS>` component.

**Aesthetic mappings**

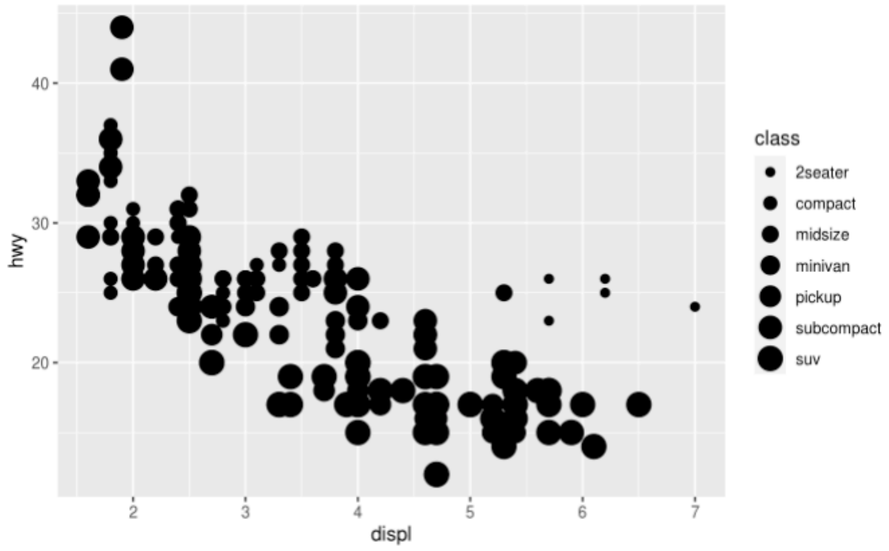
All three give the same result:

```
ggplot(mpg) +
geom_point(aes(x=displ,y=hwy))
```

```
ggplot(mpg, aes(x=displ,y=hwy)) +
geom_point()
```

Mapping a variable to the **size** aesthetic

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, size = class))
#> Warning: Using size for a discrete variable is not advised.
```



# A graphing template

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

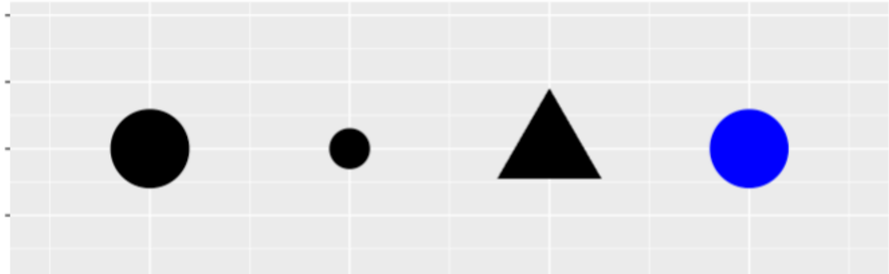
Understanding the `<MAPPINGS>` component.

**Aesthetic mappings**

A visual property of the objects in your plot. Aesthetics include:

- size,
- shape, or
- color of your points.

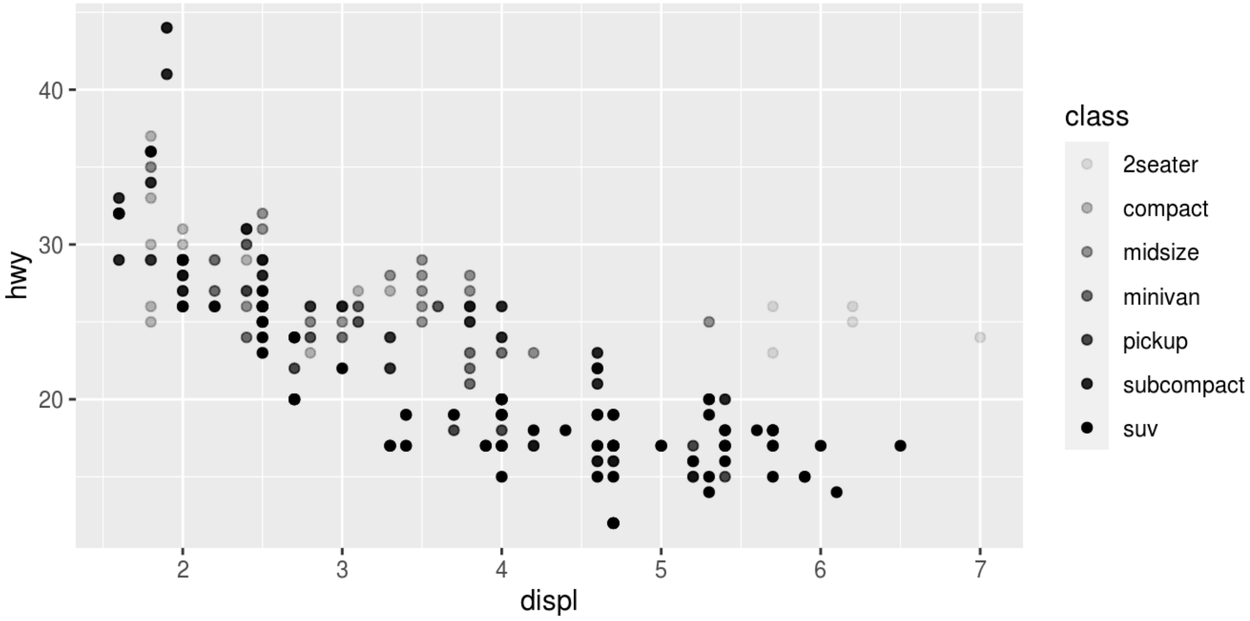
Display a point in different ways by changing the values of its aesthetic properties.



Mapping a variable to the **alpha** aesthetic

*Controls the transparency of the points*

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, alpha = class))
```



# A graphing template

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

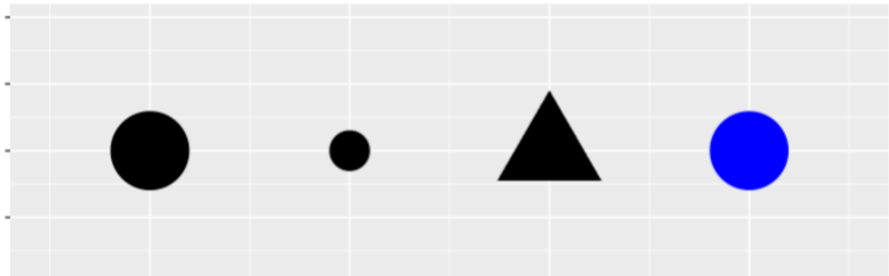
Understanding the `<MAPPINGS>` component.

**Aesthetic mappings**

A visual property of the objects in your plot. Aesthetics include:

- size,
- shape, or
- color of your points.

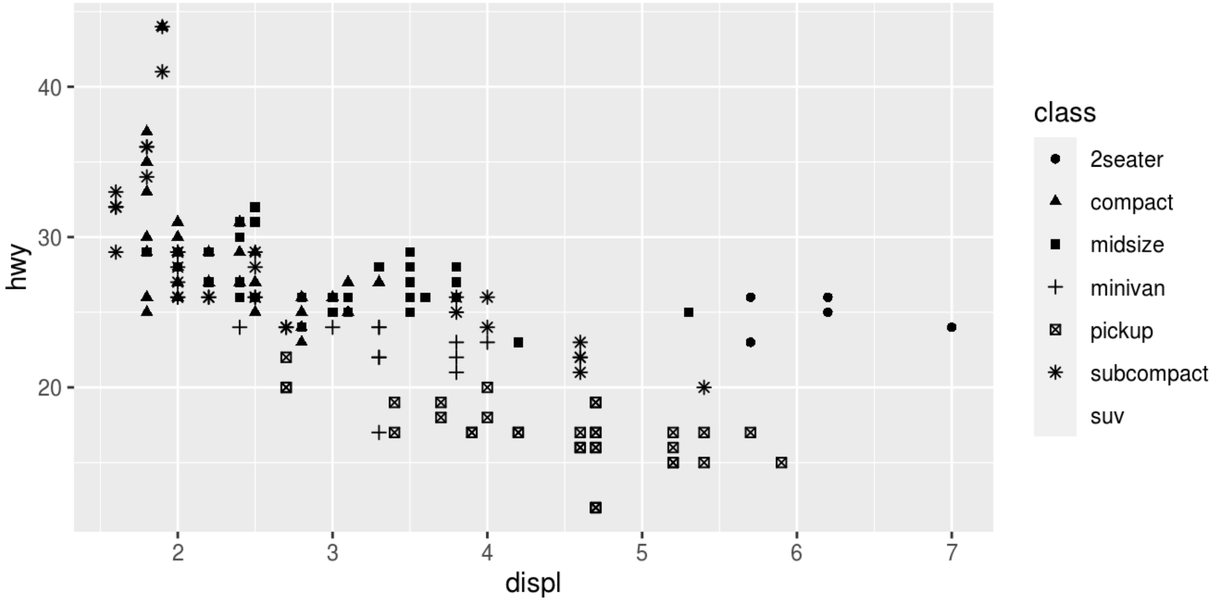
Display a point in different ways by changing the values of its aesthetic properties.



Mapping a variable to the **shape** aesthetic

*Controls the transparency of the points*

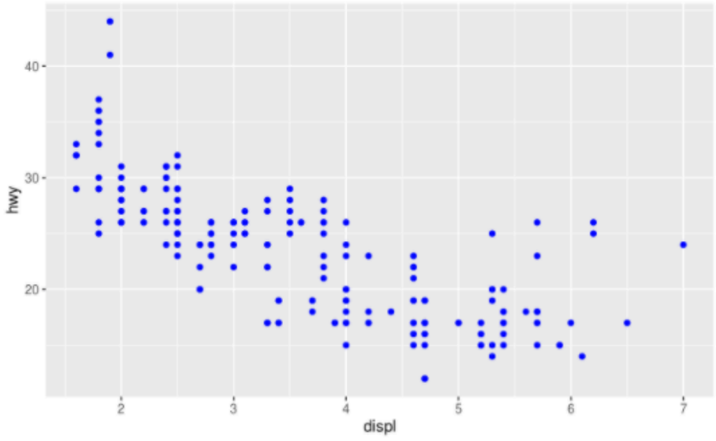
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, shape = class))
```



# Setting aesthetic manually

To set an aesthetic manually, set the aesthetic by name as an argument of your geom function; i.e. it goes *outside* of `aes()`.

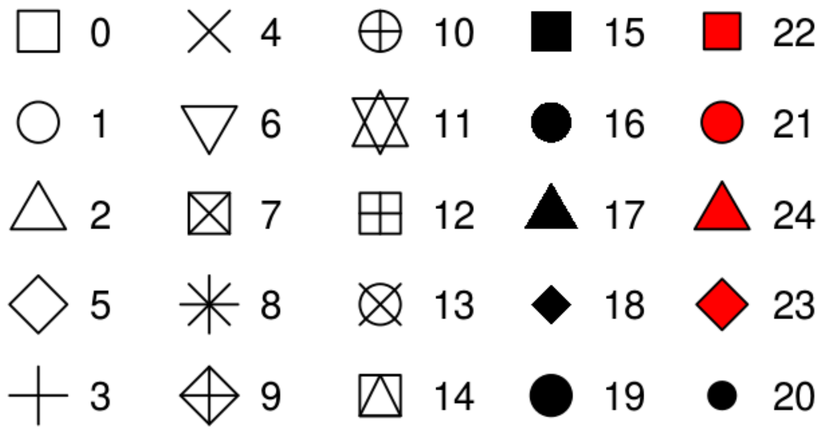
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue")
```




Here color doesn't convey information about a variable, but only changes the appearance of the plot.

# Setting aesthetic manually

To set an aesthetic manually, set the aesthetic by name as an argument of your geom function; i.e. it goes *outside* of `aes()`.




R has 25 built in shapes that are identified by numbers.

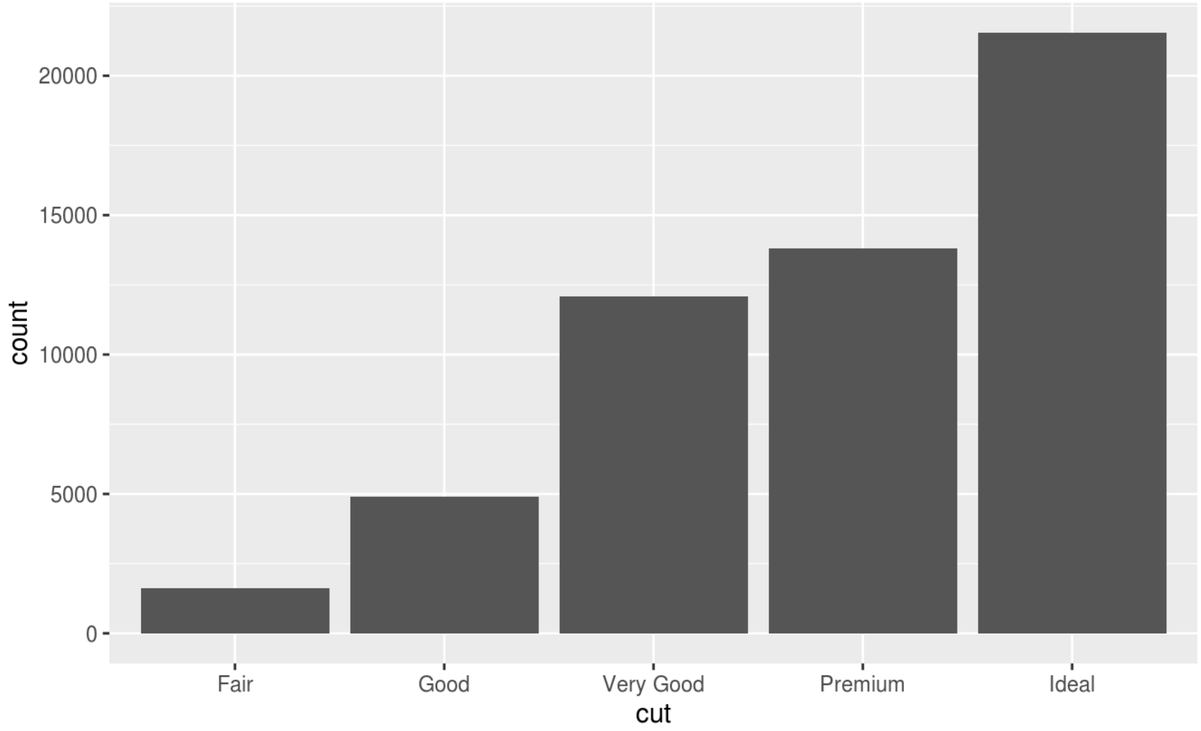
You'll need to pick a level that makes sense for that aesthetic:

- The name of a color as a character string.
- The size of a point in mm.
- The shape of a point as a number, as shown

# Changing the geom in your plot (statistical transformations of the data)

Change the geom function

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))
```



# Statistical transformations

1. **`geom_bar()`** begins with the **`diamonds`** data set

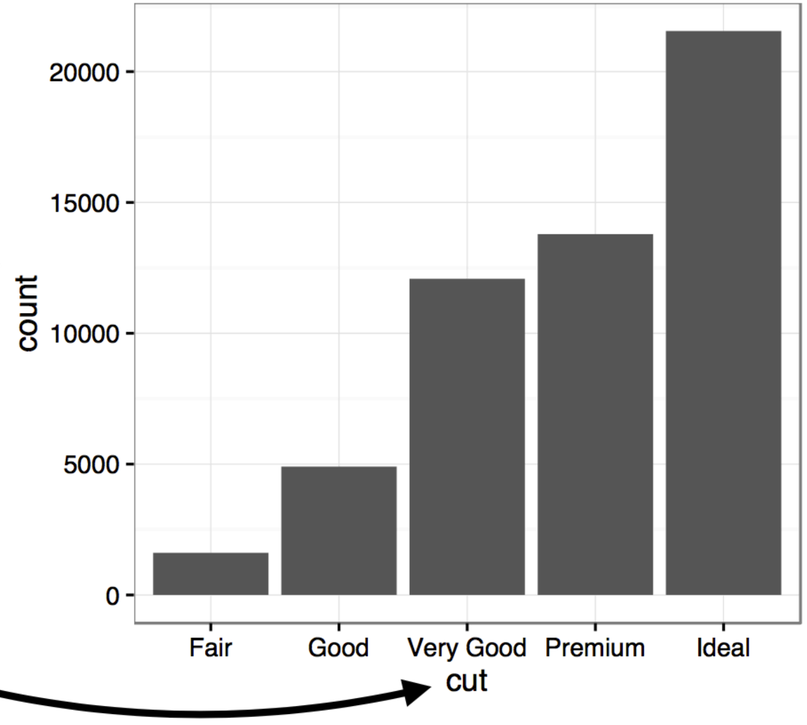
2. **`geom_bar()`** transforms the data with the "count" stat, which returns a data set of cut values and counts.

3. **`geom_bar()`** uses the transformed data to build the plot. cut is mapped to the x axis, count is mapped to the y axis.

| carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|
| 0.23 | Ideal | E | SI2 | 61.5 | 55 | 326 | 3.95 | 3.98 | 2.43 |
| 0.21 | Premium | E | SI1 | 59.8 | 61 | 326 | 3.89 | 3.84 | 2.31 |
| 0.23 | Good | E | VS1 | 56.9 | 65 | 327 | 4.05 | 4.07 | 2.31 |
| 0.29 | Premium | I | VS2 | 62.4 | 58 | 334 | 4.20 | 4.23 | 2.63 |
| 0.31 | Good | J | SI2 | 63.3 | 58 | 335 | 4.34 | 4.35 | 2.75 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

stat_count()

| cut | count | prop |
|---|---|---|
| Fair | 1610 | 1 |
| Good | 4906 | 1 |
| Very Good | 12082 | 1 |
| Premium | 13791 | 1 |
| Ideal | 21551 | 1 |

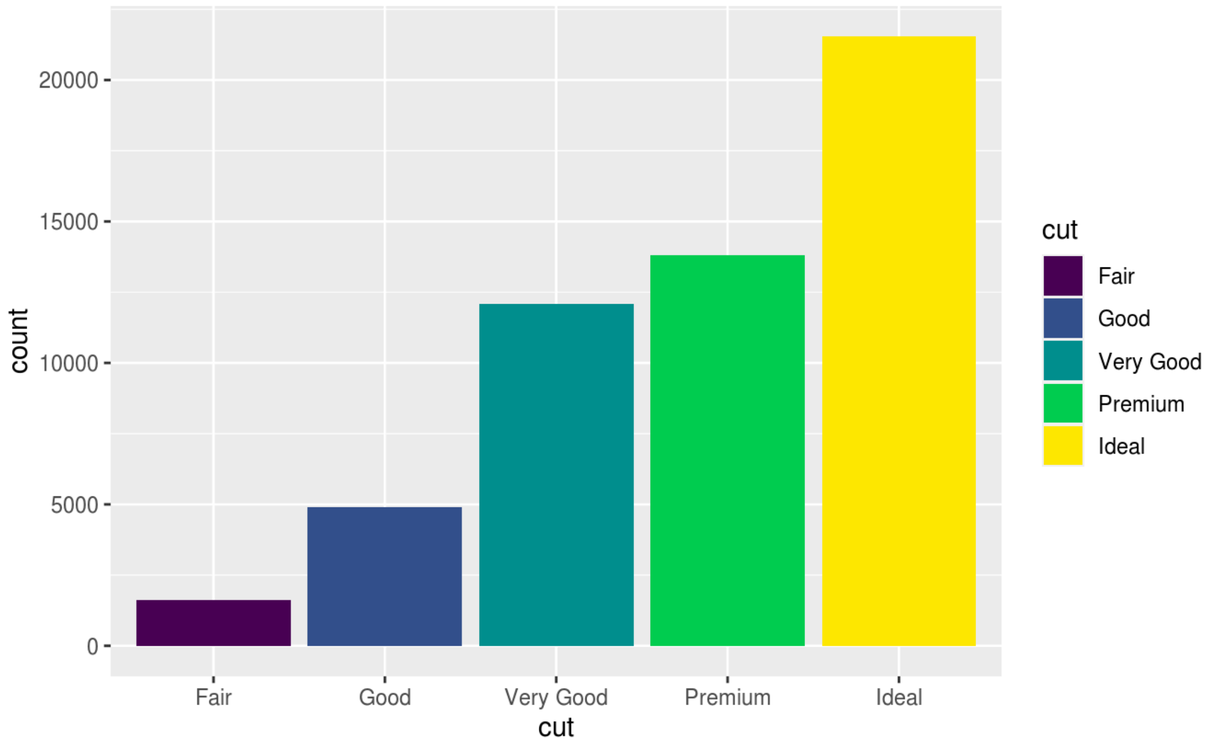


# geom_bar

https://r4ds.had.co.nz/data-visualisation.html

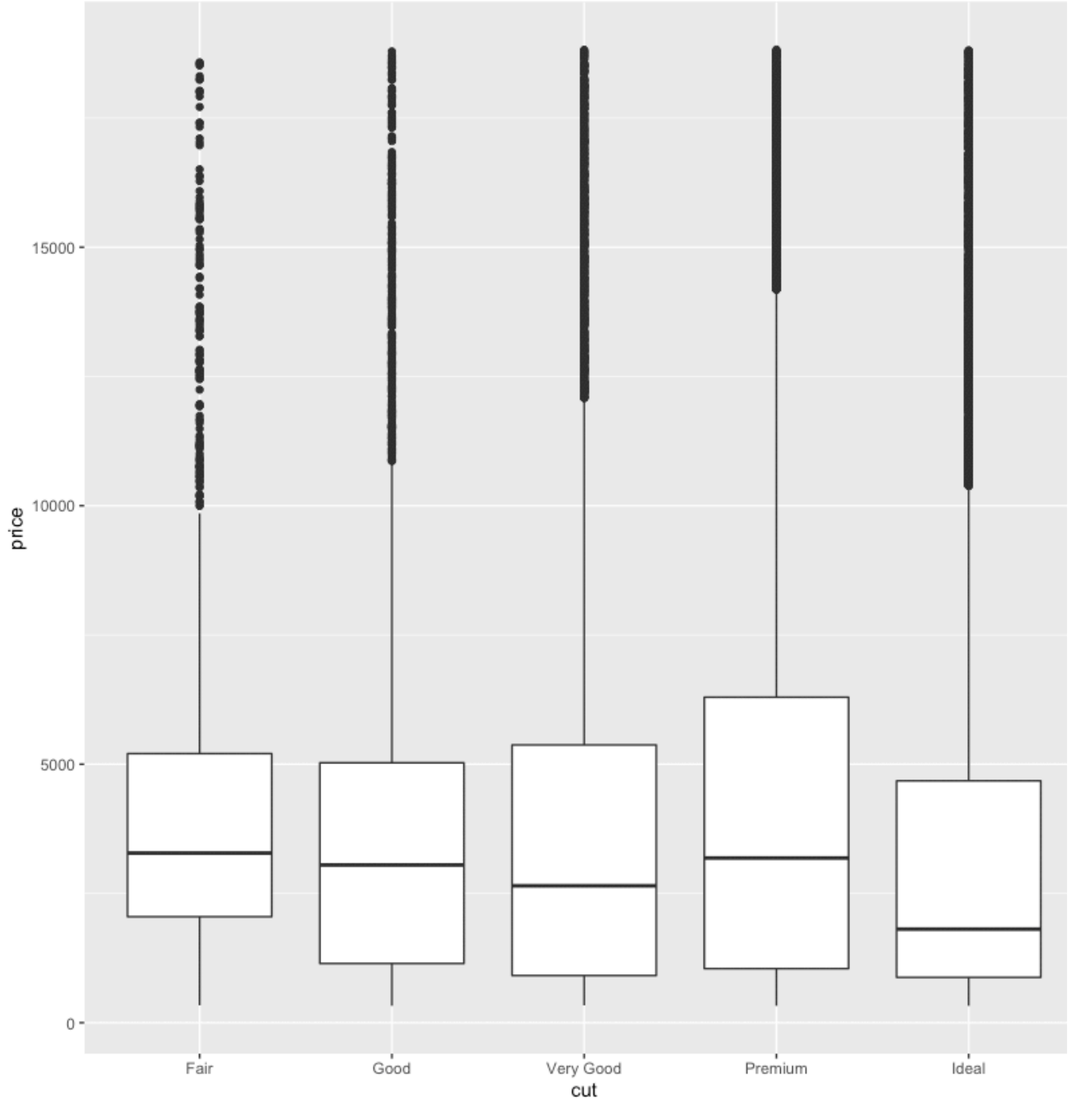
You can color a bar chart using either the **color** aesthetic, or, more usefully, **fill**:

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = cut))
```



# geom_boxplot

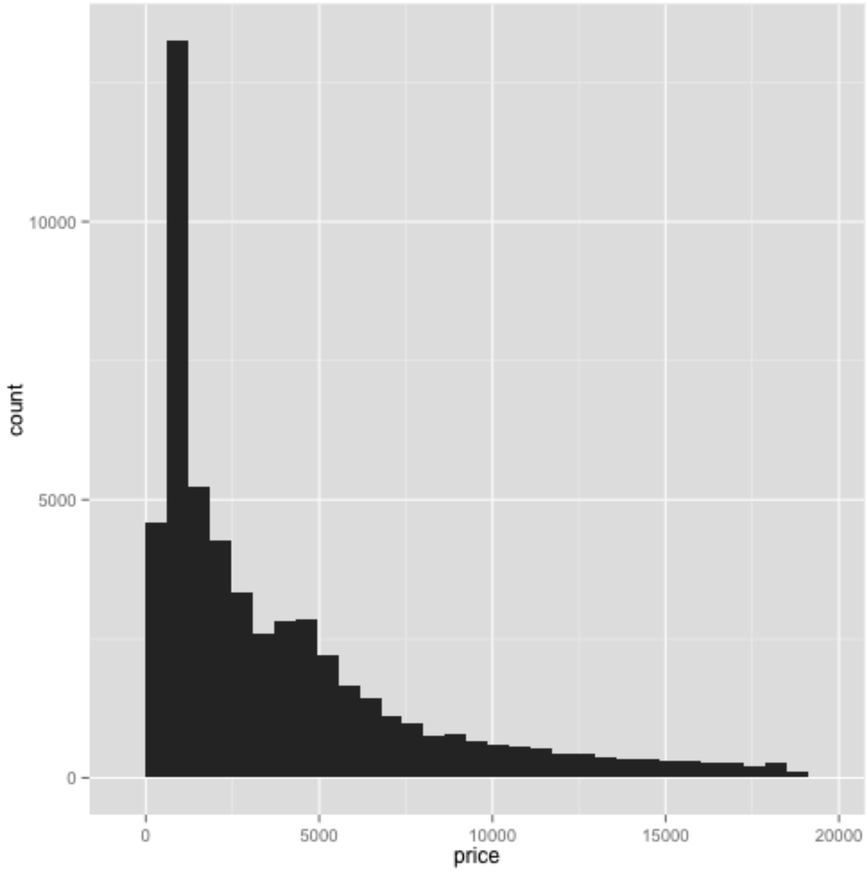
```
ggplot(data = diamonds) +
  geom_boxplot(aes(x=cut, y=price))
```



# geom_histogram

```
ggplot(diamonds, aes(x=price)) + geom_histogram()
```

visualization of the density of the distribution of price.

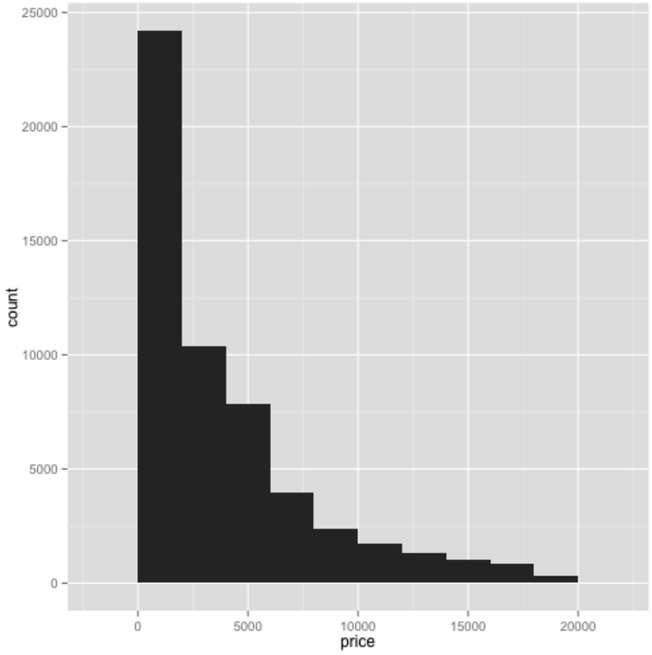


# geom_histogram

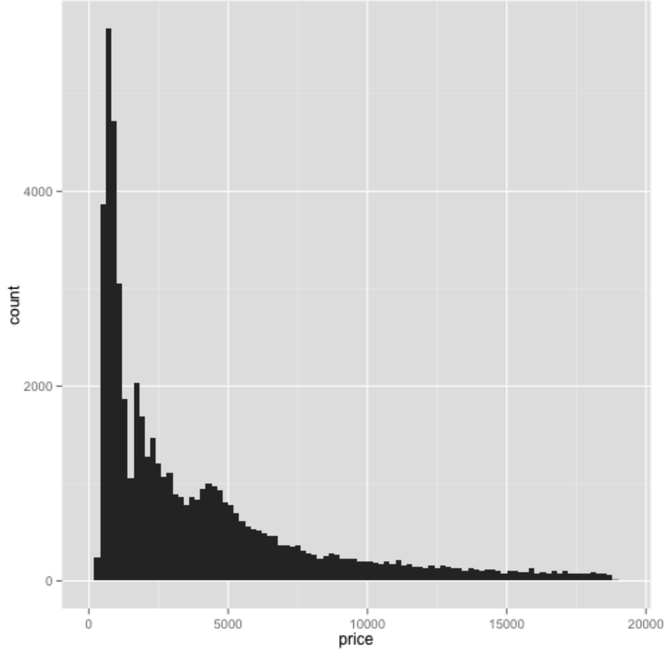
visualization of the density of the distribution of price.

Changing width of each bin to make them wider or thinner

```
ggplot(diamonds, aes(x=price)) +
geom_histogram(binwidth=2000)
```



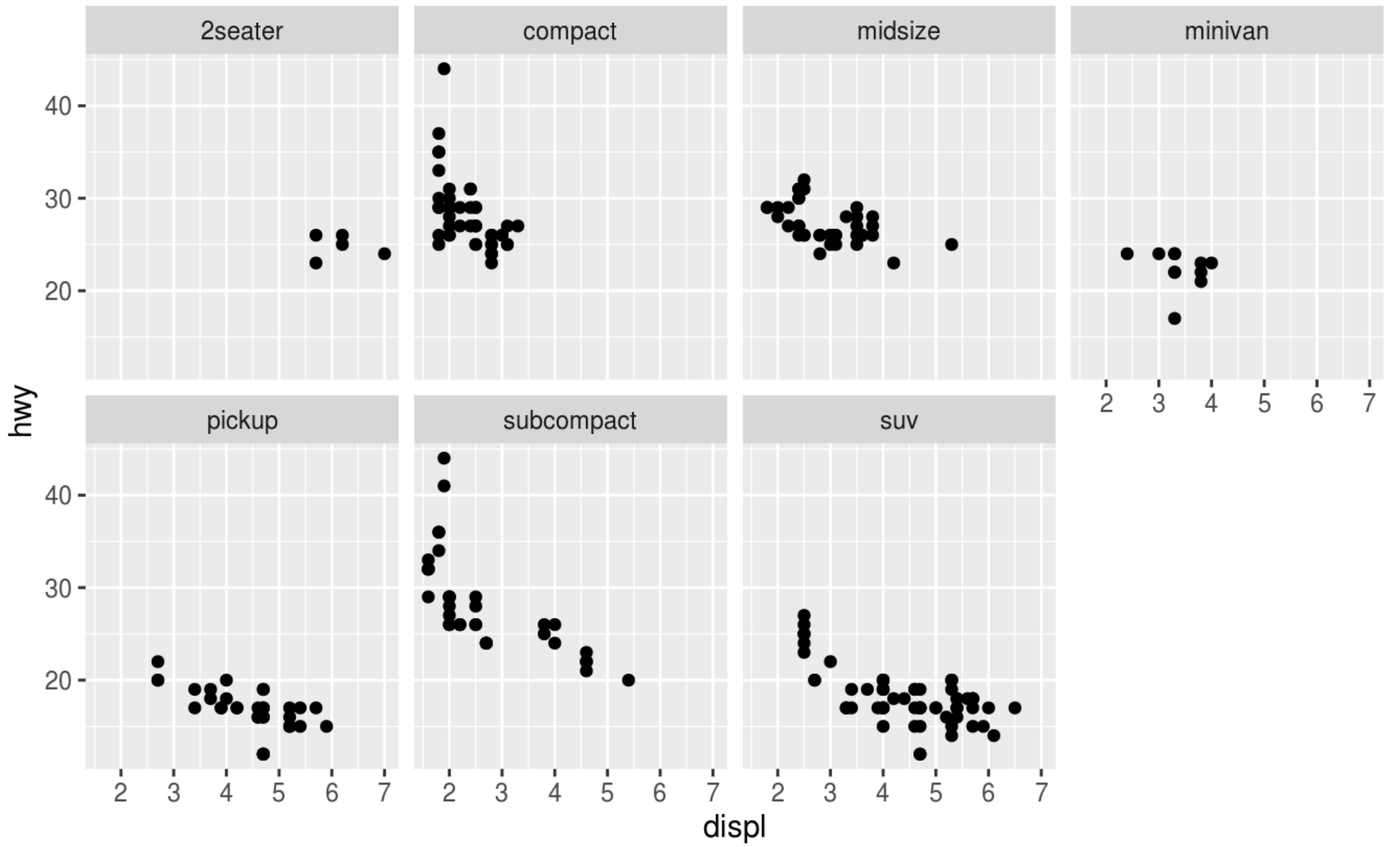

```
ggplot(diamonds, aes(x=price)) +
geom_histogram(binwidth=200)
```



# Facets subplots that each display one subset of the data

To facet your plot by a single variable, use `facet_wrap()`. The first argument of `facet_wrap()` should be a formula, which you create with `~` followed by a variable name

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```

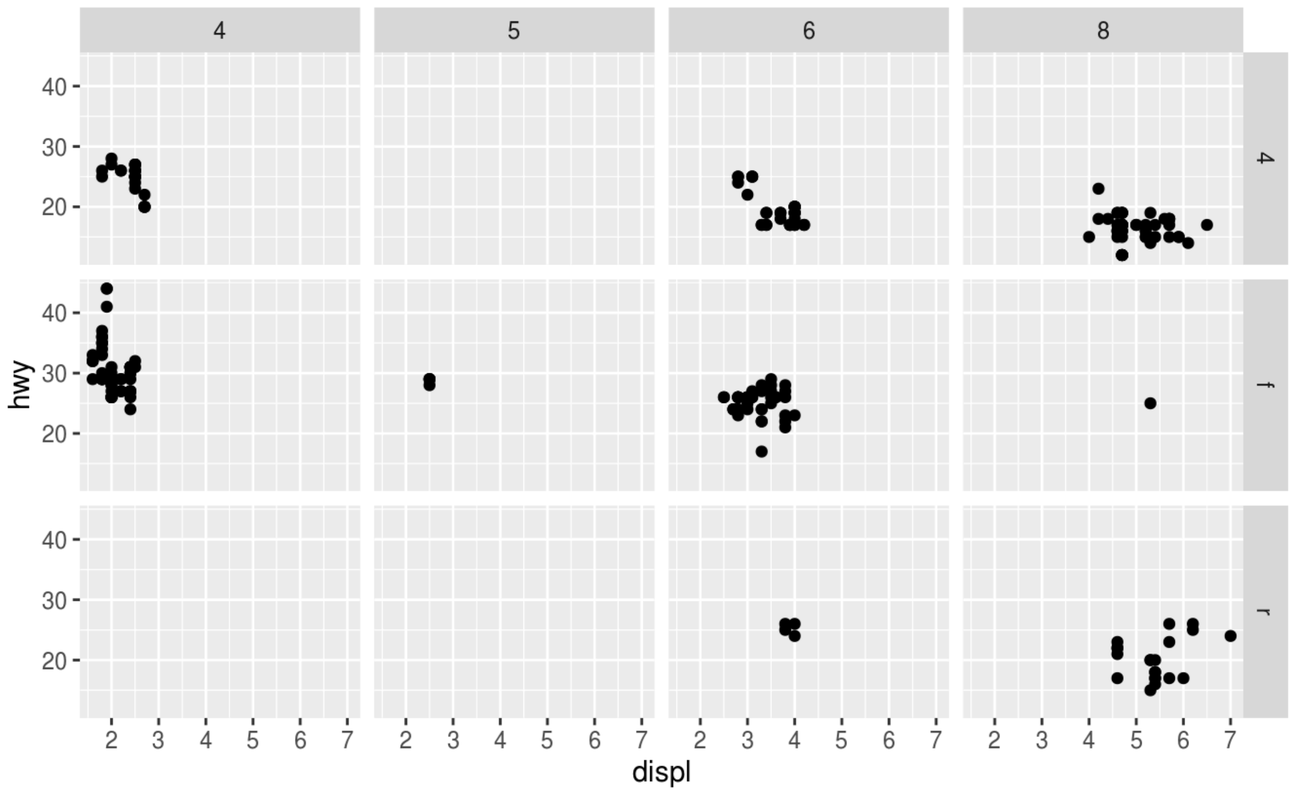


# Facets

subplots that each display one subset of the data

To facet your plot on a combination of 2 variables, use `facet_grid()`. The first argument of `facet_wrap()` is also a formula, which should be 2 variable names separated with `~`

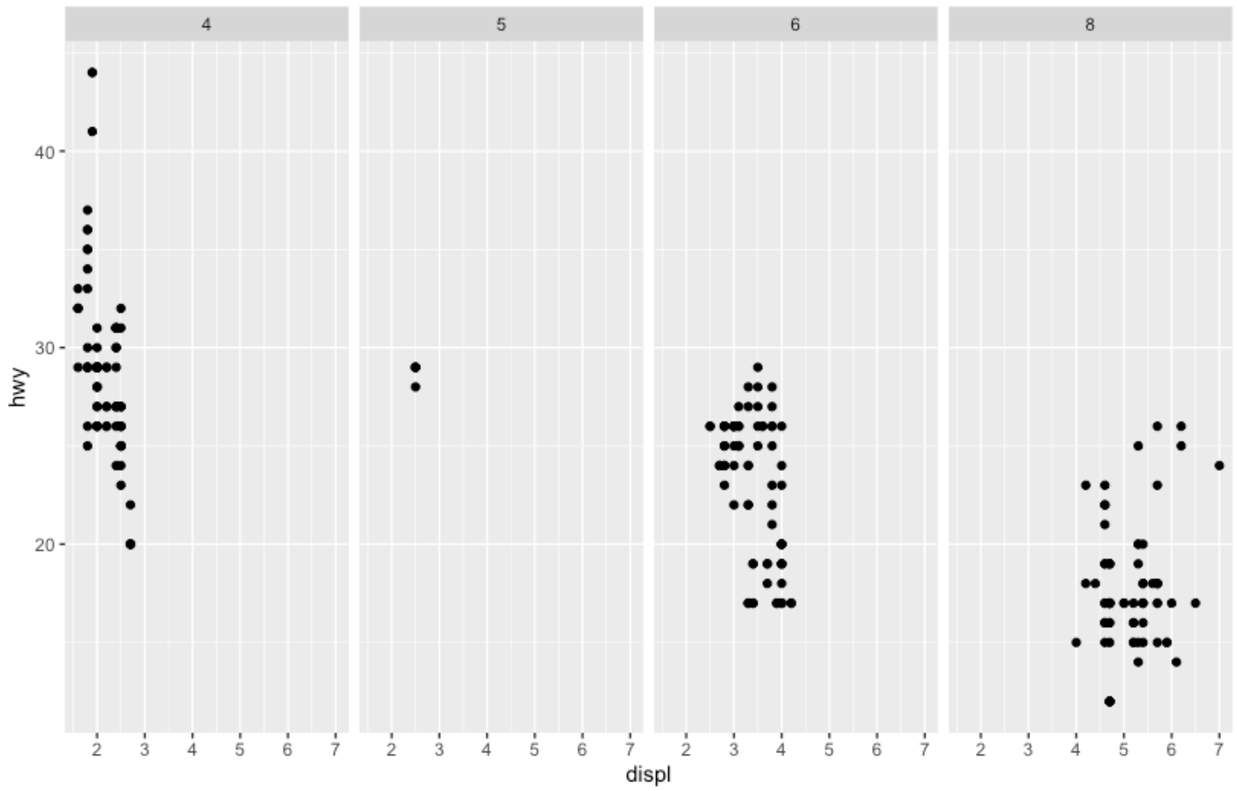
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(drv ~ cyl)
```



# Facets subplots that each display one subset of the data

If you prefer to not facet in the rows or columns dimension, use a `.` instead of a variable name, e.g. `+ facet_grid(. ~ cyl)`.

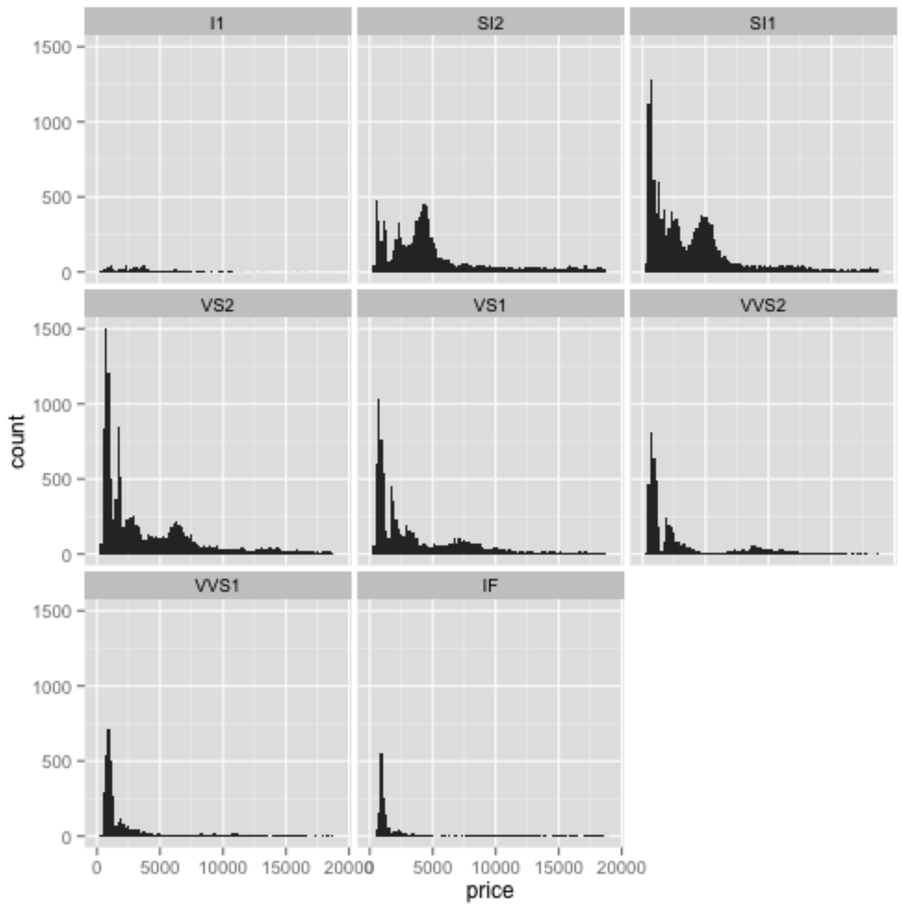
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(. ~ cyl)
```



# Facets subplots that each display one subset of the data

Another example

```
ggplot(diamonds, aes(x=price)) +
geom_histogram(binwidth=200) +
facet_wrap(~ clarity)
```




http://varianceexplained.org/RData/code/code_lesson2/

# Common Problems

As you start to run R code, you're likely to run into problems. Don't worry — it happens to everyone.

- Start by carefully comparing the code that you're running to the code in the book.
- R is extremely picky, and a misplaced character can make all the difference.
- Make sure that every ( is matched with a ) and every " is paired with another ".
- Sometimes you'll run the code and nothing happens. Check the left-hand of your console: if it's a +, it means that R doesn't think you've typed a complete expression and it's waiting for you to finish it. In this case, it's usually easy to start from scratch again by pressing ESCAPE to abort processing the current command.
- One common problem when creating ggplot2 graphics is to put the + in the wrong place: it has to come at the end of the line, not the start. In other words, make sure you haven't accidentally written code like this:

```
ggplot(data = mpg)
+ geom_point(mapping = aes(x = displ, y = hwy))
```

# LAB: Exploring the titanic data

# LAB: Exploring the titanic data

Sample solution: https://bio304-class.github.io/bio304-fall2017/data-story-titanic.html

Other useful references to learn plotting:

Data Visualization with ggplot2

**Lesson 2: Visualizing Data Using ggplot2**

**https://bookdown.org/mikemahoney218/IDEAR/visualizing-your-data.html**

**https://r4ds.had.co.nz/data-visualisation.html**

# Next class

dplyr

**TUE, OCT 13**

Data Manipulation

*DUE: Problem Set 1 - Exploring Data*

**Required Reading:**

★ *Wickham and Grolemund* (2017). R for Data Science. Chapters 9-14. OReilly Media, Inc.

*Lab: Working with Data Part I*

DUE

- Lab2: upload by tomorrow (Oct 9) 9am PT
- PS1: deadline next Tuesday (Oct 13) 9am PT