Overview of R Graphics and **ggplot2**

Data Visualization Workshop @ Scientific Computing Day

Matthew D. Turner
Georgia State University
mturner46@gsu.edu

Preliminaries

Setup

- I am assuming that:
  - You have used R at least a little bit before today
  - You have R installed
  - You are using Rstudio as your interface to R
- You should also have followed the instructions in the email:

```
install.packages("tidyverse", dependencies = TRUE)
install.packages("gcookbook")
```

  - These commands need to be entered in the command window for R inside of RStudio
  - These commands only need to be run once (assuming they work!)
  - They must be run while your computer is connected to the internet

Setup

- I am assuming that:
  - You have used R at least a lit
  - You have R installed
  - You are using Rstudio as you erface to R
- You should also have follow ne instruction

IMPORTANT NOTE!!

Due to a webpage formatting issue, the double quote marks in the commands were formatted with quotes that are **incompatible** with R. If these commands failed for you, try **typing them in directly**. Then they should work!

```
install.packages("tidyverse", dependencies = TRUE)
install.packages("gcookbook")
```

- These commands need to be entered in the command window for R inside of RStudio
- These commands only need to be run once (assuming they work!)
- They must be run while your computer is connected to the internet

Setup

- You should also download the materials for the workshop today:
  - Github: fork the repo, then clone your copy locally

https://github.com/theEducationalCollective/2018-Scientific-Computing-Day-Visualization-Workshop

http://bit.ly/scd2018git

  - Or as a zip file:

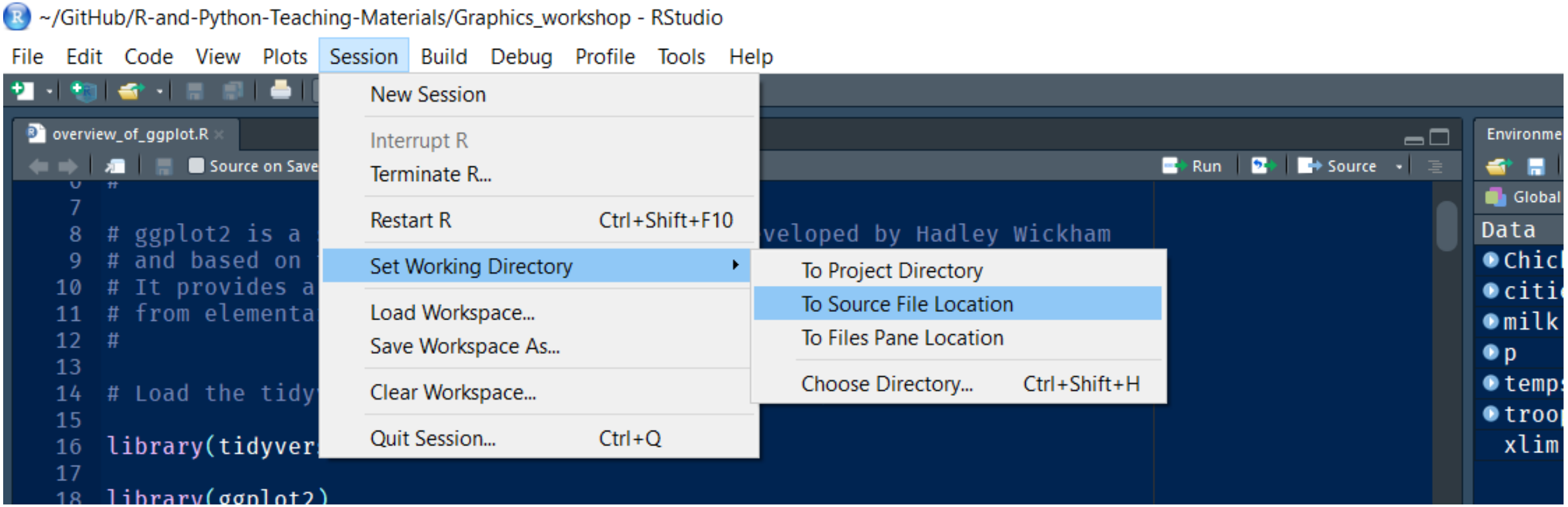
http://bit.ly/scd2018zip

Setup: Two Important Things

- Rstudio allows you to change its working directory to the location of the file you are using from the RStudio menu (at top)
  - You should do this for each of the files from the workshop today when you load a new one
  - Failure to do this may break the file, if that happens, just go to the menu change the working directory and try again

Setup: Two Important Things

- Rstudio allows you to change its working directory to the location of the file you are using from the RStudio menu (at top)



Setup: Two Important Things

- Rstudio allows you to change its working directory to the location of the file you are using from the RStudio menu (at top)
  - You should do this for each of the files from the workshop today when you load a new one
  - Failure to do this may break the file, if that happens, just go to the menu change the working directory and try again
- You need to load the ggplot library before using it
  - This can be done in two ways. Use one of the following commands:
    - **`library(ggplot2)`**
    - **`library(tidyverse)`**
- This will be done in the examples files, but you have to **run the files in order** if you skip around you may need to remember this!

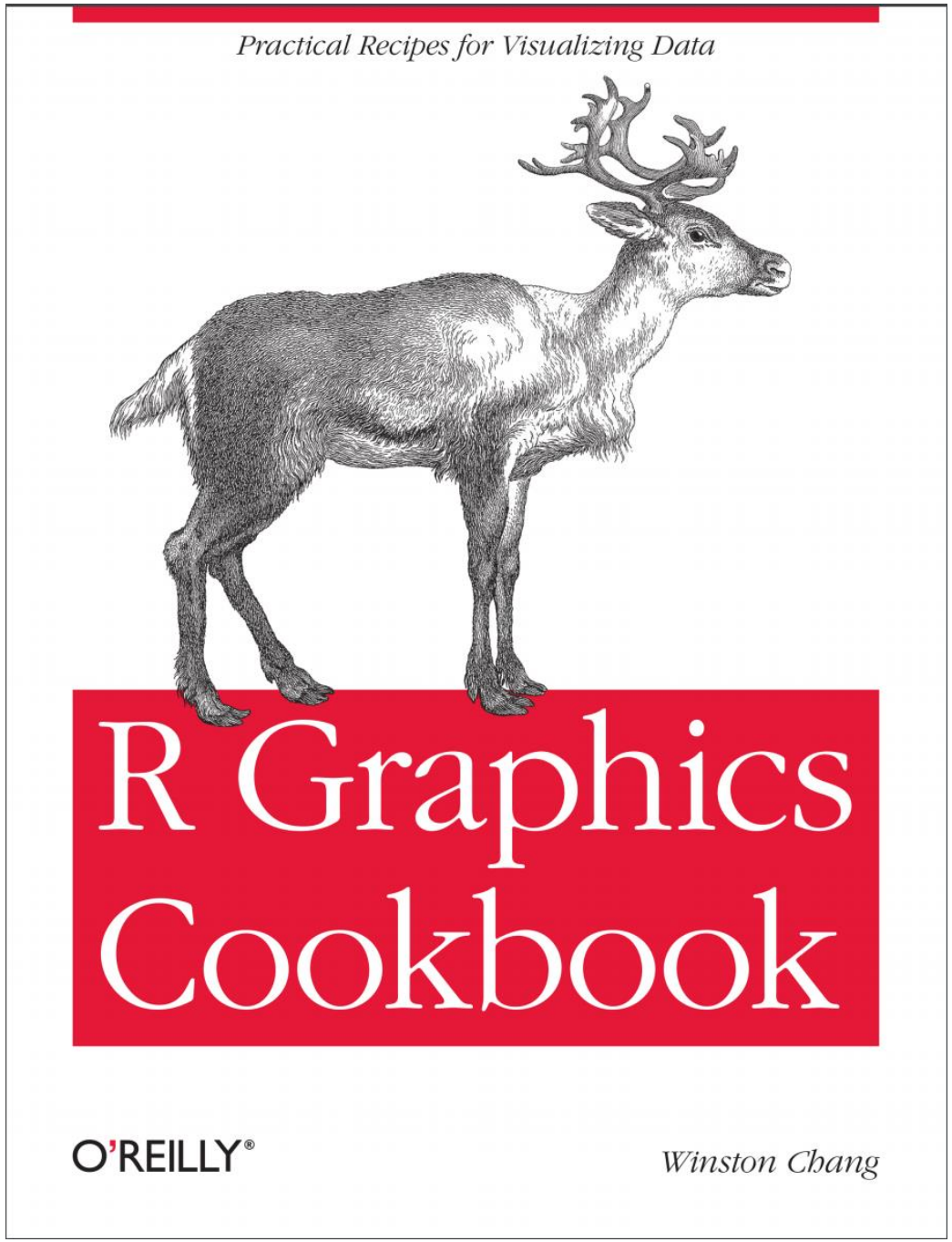
On to ggplot

R Plotting Systems

- R has three plotting systems:
  1. Base R Graphics
  2. Lattice Graphics
  3. **ggplot2**
- Most people start with the base R graphics system
  - I will assume you have used this a little bit
- The ggplot2 system is built on top of it
- We'll focus on the ggplot2 system today because it is the easiest way to get good quality figures easily for a wide variety of types of data

Note

- Some of the examples I will be using today come from a set of online notes from a course at IDRE at UCLA
  - The course has slides at: stats.idre.ucla.edu/stat/data/intro_ggplot2/ggplot2_intro_slidy.html
  - The full set of materials for the UCLA workshop is at: stats.idre.ucla.edu/r/seminars/ggplot2_intro/
- The presentation here will be somewhat simpler and assume less experience with R
- The UCLA workshop is a good follow-up for more



Another good resource for ggplot and R graphics generally is Winston Chang's **R Graphics Cookbook** now available in a **2nd** edition.

This book provides hundreds of examples of a wide variety of plots as well as some introduction to general use of ggplot, and a chapter on **data munging** (data cleaning/re-arrangement).

History

- Leland Wilkinson developed a system for building graphic displays called the “grammar of graphics”
  - The idea was to develop a general language for describing plots of data
  - It allows more arbitrary mappings of data to the aesthetic aspects of pictures like size, color, position, etc.

- Hadley Wickham substantially extended and modified this system and implemented it in the R language
  - **`ggplot1`** was a prototype version with very different syntax
  - **`ggplot2`** is essentially the first (and only) version ever available

- This system has been continuously developed by many people

Syntax: Overview

- **`ggplot2`** builds graphics in **layers** that are specified in *ascending* order
  - You build the plots from items in the **background** to the **foreground**

- Each layer specifies **aesthetic mappings**
  - More in a moment!
  - If these are not given, then they are inherited from the previous layer

- Finally, the physical parts of the plot are specified by "**geoms**"

Syntax: Aesthetics

- Aesthetics are aspects of the plot that can be used to convey information
- Basically it is a list of physical aspects of the plot that could carry information

- These can be mapped to variables of interest

Some example aesthetics:

- `x`: positioning along x-axis
- `y`: positioning along y-axis
- `color`: color of objects; for 2-d objects, the color of the object's outline (compare to fill below)
- `fill`: fill color of objects
- `alpha`: transparency of objects (value between 0, transparent, and 1, opaque – inverse of how many stacked objects it will take to be opaque)
- `linetype`: how lines should be drawn (solid, dashed, dotted, etc.)
- `shape`: shape of markers in scatter plots
- `size`: how large objects appear

From: https://stats.idre.ucla.edu/r/seminars/ggplot2_intro/

Syntax: Geoms

- Geoms are standard plot elements that we are used to using for displaying data
- Each geom function accepts a subset of the aesthetics available

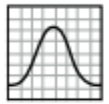
Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

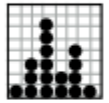
Continuous

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

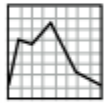
a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")



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

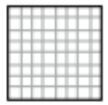
c <- ggplot(map, aes(long, lat))

c + geom_polygon(aes(group = group)**)**

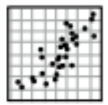
Two Variables

Continuous X, Continuous Y

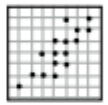
f <- ggplot(mpg, aes(cty, hwy))



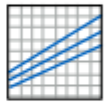
f + geom_blank()



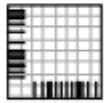
f + geom_jitter()
x, y, alpha, color, fill, shape, size



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

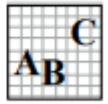


f + geom_quantile()
x, y, alpha, color, linetype, size, weight



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

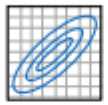
g <- ggplot(mpg, aes(class, hwy))

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

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



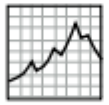
i + geom_density2d()
x, y, alpha, colour, linetype, size

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

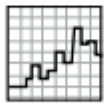
Continuous Function

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

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



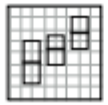
j + geom_line()
x, y, alpha, color, linetype, size



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()

From the ggplot2 cheat sheet (included with workshop materials). From: https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

Notice what physical features of a plot geom_point() makes available for data…

m's aesthetic properties to represent variables. Each function returns a layer.

Two Variables

Continuous X, Continuous Y

lot(mpg, aes(cty, hwy))

_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

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))

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

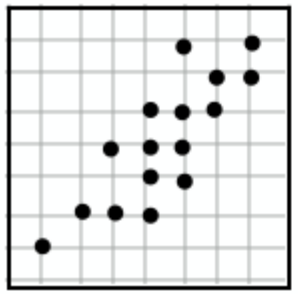
a + geom_freqpoly()
x, y, alpha, color, linetype, si
b + geom_freqpoly(aes(y = ..

a + geom_histogram(binw
x, y, alpha, color, fill, linetype
b + geom_histogram(aes(y =

Discrete

b <- ggplot(mpg, aes(

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



f + geom_point()

x, y, alpha, color, fill, shape, size

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))

Discrete X, Continuous Y

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

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

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

k + geom_errorbar()

Graphical Primitives

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

c + geom_polygon(aes(group = group))

From the ggplot2 cheat sheet (included with workshop materials). From: https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

Syntax

- All of your data needs to be in a single data frame for simple plots

- You may need to restructure your data a bit to make things work
  - Much of the **`tidyverse`** collection of R packages were made to make this easier
  - We only have time to just touch on the tidyverse today, if you are interested there is a lot of information available online (it is very powerful)

Simple Examples

```
library(tidyverse)
```

```
> library(tidyverse)
-- Attaching packages ------------------------------------- tidyverse 1.2.1 --
v ggplot2 2.2.1     v purrr   0.2.4
v tibble  1.4.2     v dplyr   0.7.4
v tidyr   0.8.0     v stringr 1.3.0
v readr   1.1.1     v forcats 0.3.0
-- Conflicts ---------------------------------------- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

Loading the **`tidyverse`** basically configures R to use the "tidy" data format:

- Rows are data "points" (subjects, observations, units, etc.)
- Columns are variables observed on units
- The tidy format is a "tall" data format: if you have multiple observations on a unit, each observation has its own row!

The **`tidyverse`** us a collection of tools to make this format easy to achieve and maintain as you process your data set.

```
library(tidyverse)
```

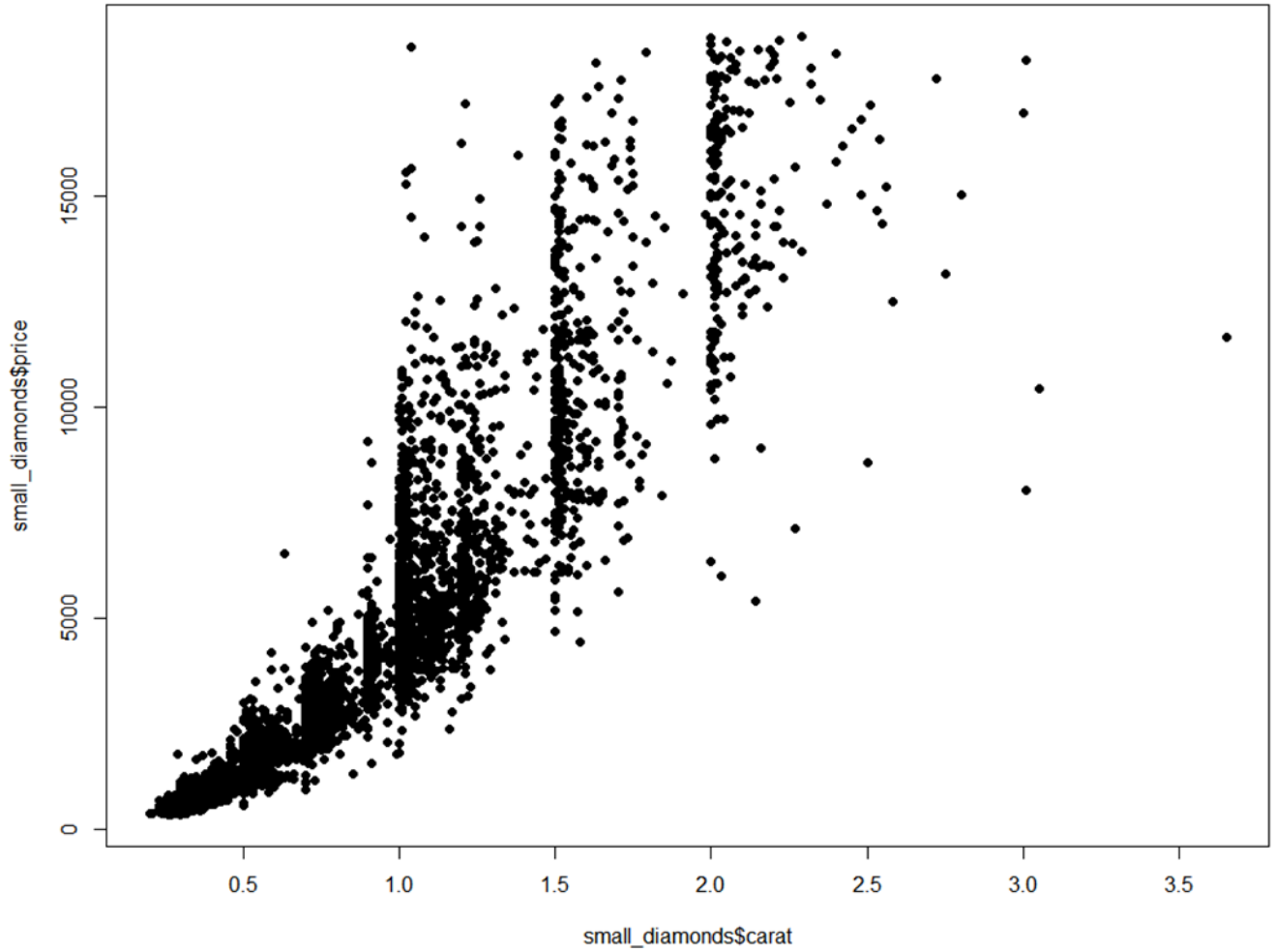
```
> library(tidyverse)
-- Attaching packages ------------------------------------ tidyverse 1.2.1 --
v ggplot2 2.2.1     v purrr   0.2.4
v tibble  1.4.2     v dplyr   0.7.4
v tidyr   0.8.0     v stringr 1.3.0
v readr   1.1.1     v forcats 0.3.0
-- Conflicts --------------------------------------- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

```
data("diamonds")
head(diamonds)
dim(diamonds)
```

```
> head(diamonds)
# A tibble: 6 x 10
  carat cut       color clarity depth table price     x     y     z
  <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1 0.230 Ideal     E     SI2      61.5   55.   326  3.95  3.98  2.43
2 0.210 Premium   E     SI1      59.8   61.   326  3.89  3.84  2.31
3 0.230 Good      E     VS1      56.9   65.   327  4.05  4.07  2.31
4 0.290 Premium   I     VS2      62.4   58.   334  4.20  4.23  2.63
5 0.310 Good      J     SI2      63.3   58.   335  4.34  4.35  2.75
6 0.240 Very Good J     VVS2     62.8   57.   336  3.94  3.96  2.48
> dim(diamonds)
[1] 53940    10
```

```
small_diamonds <- sample_n(diamonds, size = 5000)

plot(small_diamonds$carat, small_diamonds$price, pch = 16)
```

This plot command is a base R graphics command and the sort that most people should be familiar with using. The output is fairly simple but we could add more nice features.



How would this go in **`ggplot2`**?

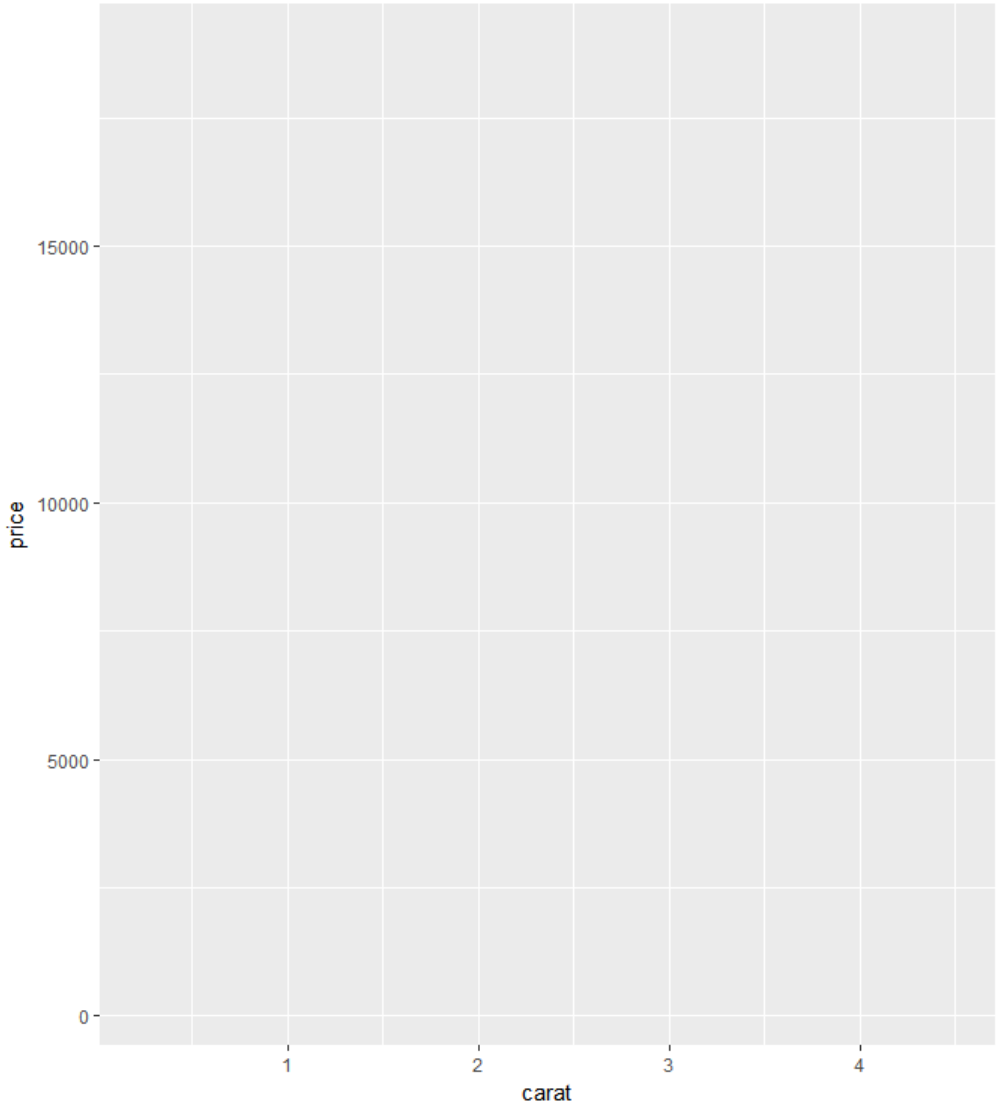
- We begin by calling the ggplot command (no 2!) and setting our data source and baseline aesthetic

```
pvc <- ggplot(aes(x = carat, y = price), data = small_diamonds)

pvc
```

"aes" for aesthetic

In R, there are two ways to give variables to functions:

1. List them in order (like with the previous plot command)
2. Specify them by name (as done here)



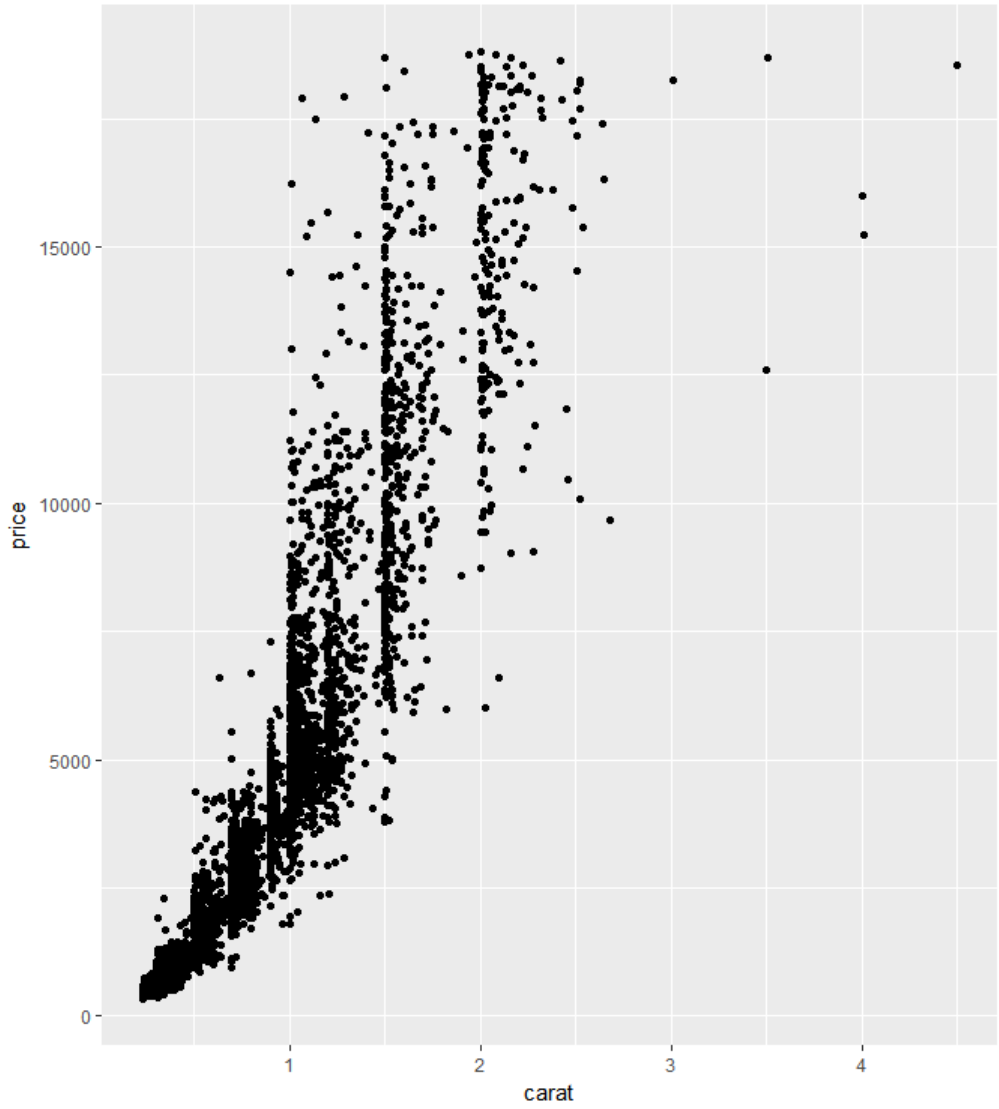
How would this go in **ggplot2**?

- We begin by calling the ggplot command (no 2!) and setting our data source and baseline aesthetic
- Notice:
  - No plot is made, but the axes are computed based on the data set
  - The first line sets the aesthetics and data
  - The second line shows the plot (just **pvc** – the name of the plot!)
- To actually make a figure, we need to add a “geom” (geometry or geometric object)
  - ggplot builds plots by adding together different plot objects

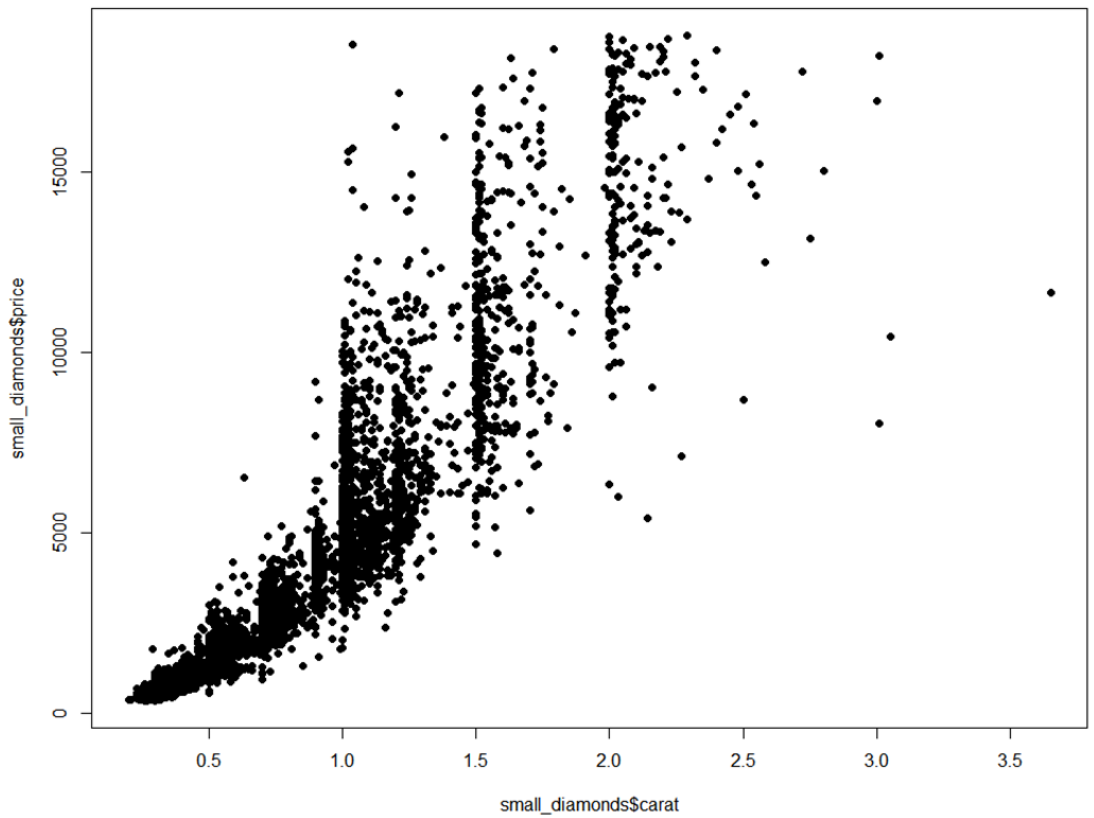
```
pvc <- ggplot(aes(x = carat, y = price), data = small_diamonds) +
       geom_point()

pvc
```

So for a really simple figure, ggplot is a little more complicated to start, but we get some nice features for free:

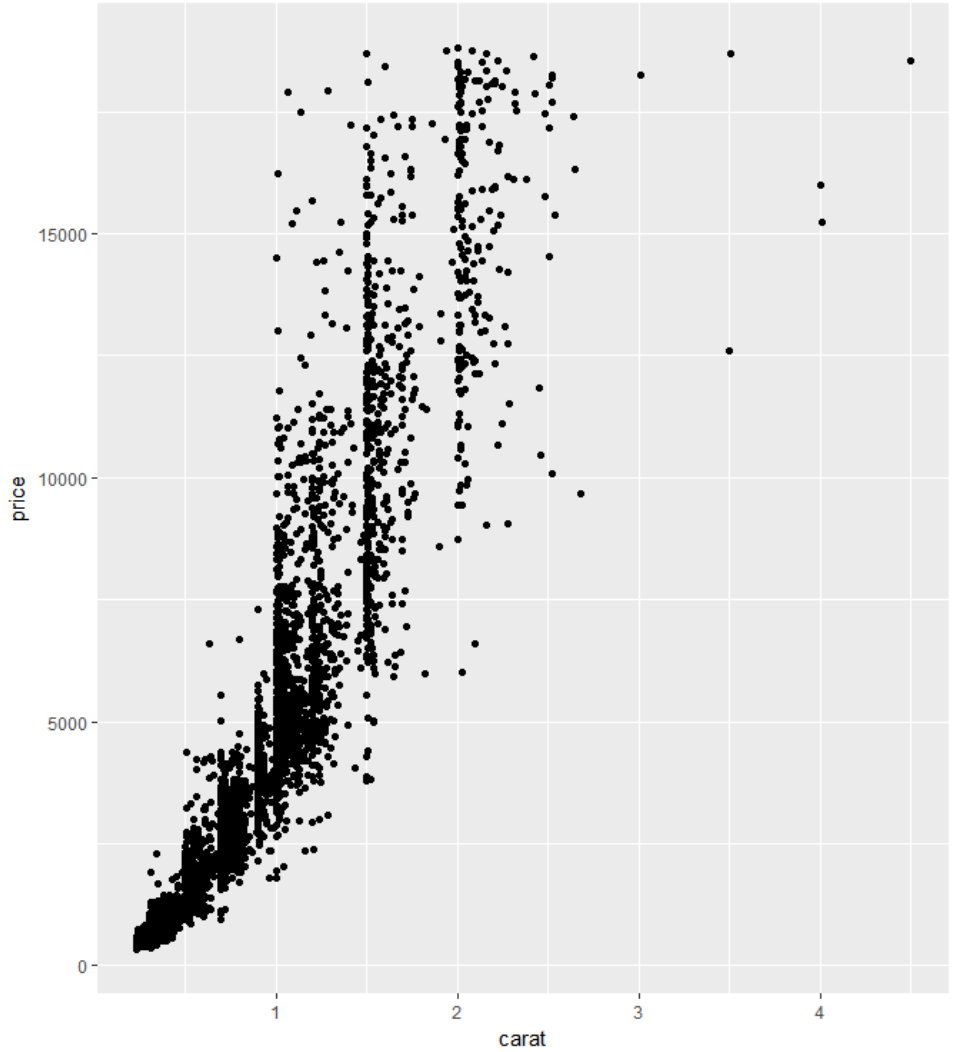
- Axis names are cleaner
- The grid background is visually helpful



It is worth noting that there is really nothing that you can do with ggplot that you cannot do with base R graphics, but doing those things can take a lot of code.



Base R Graphics

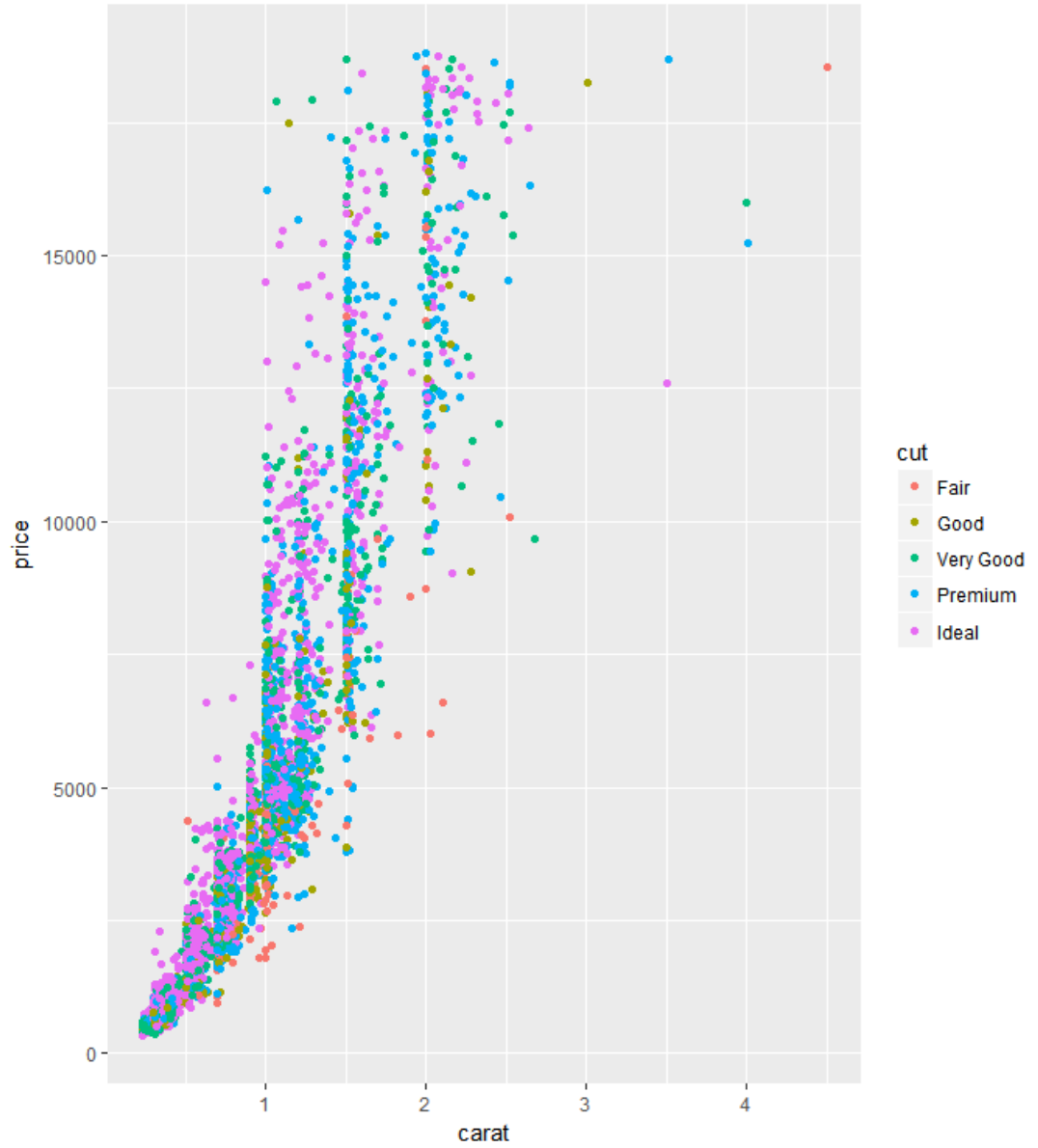


ggplot2

- The critical benefit of ggplot appears when you go beyond a basic plot

- For this example, let’s add the cut of the diamond to the plot
  - In ggplot this is very easy…

```
pvc <- ggplot(aes(x = carat, y = price), data = small_diamonds) +
        geom_point(aes(color = cut))
pvc
```



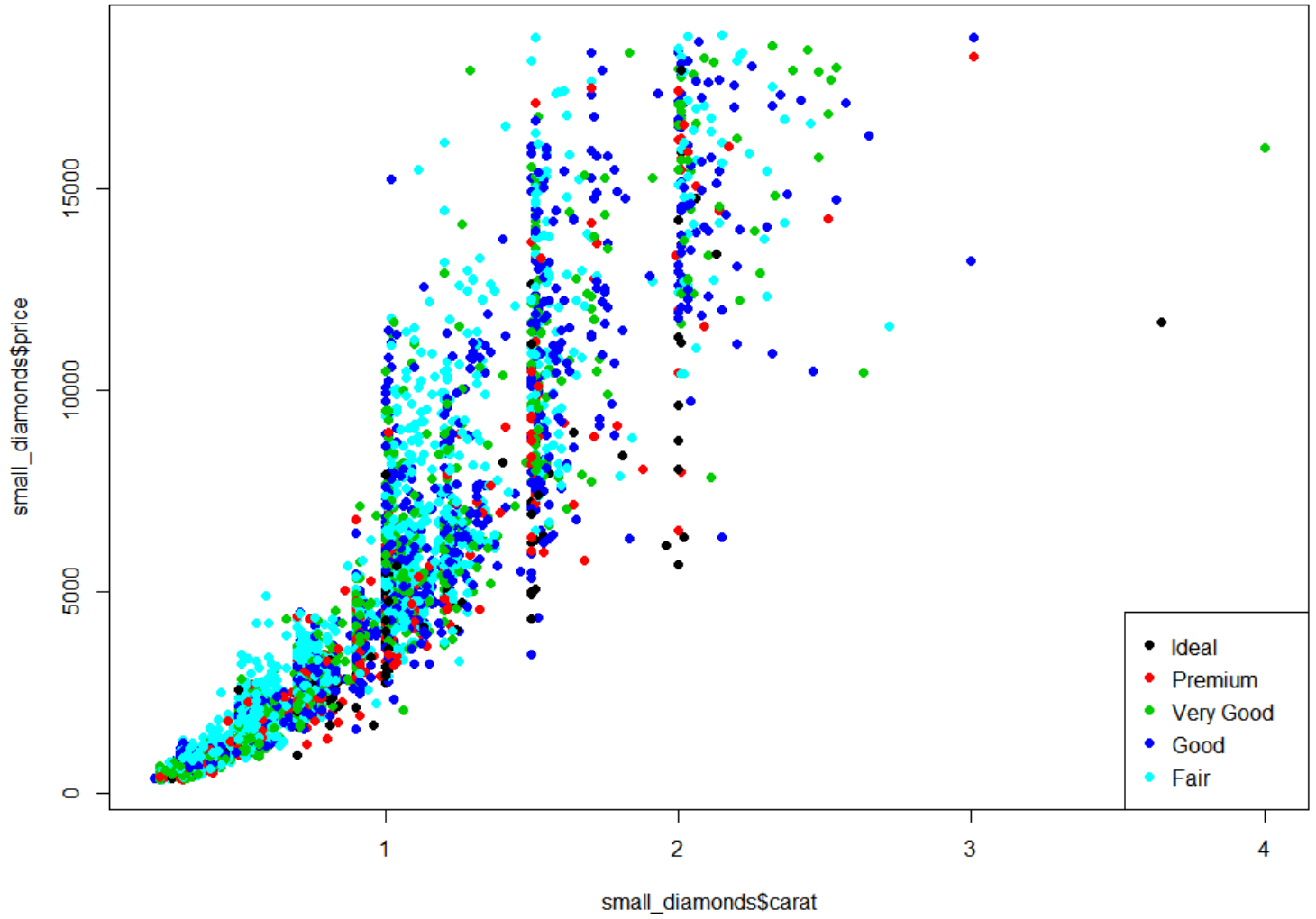
The only change here is to add a new aesthetic relating the color of the points to the variable “cut.”

Notice that the plot region is shaped a little differently due to the legend being added for the colors.

We can change this by adding a size specification to the region.

```
plot(small_diamonds$carat, small_diamonds$price,
     col = small_diamonds$cut, pch = 16)
legend("bottomright", legend = unique(small_diamonds$cut),
       col = 1:length(small_diamonds$cut), pch = 16)
```

Base R Version



Specifying and Modifying Aesthetics

- Notice that these two commands make the same plot:

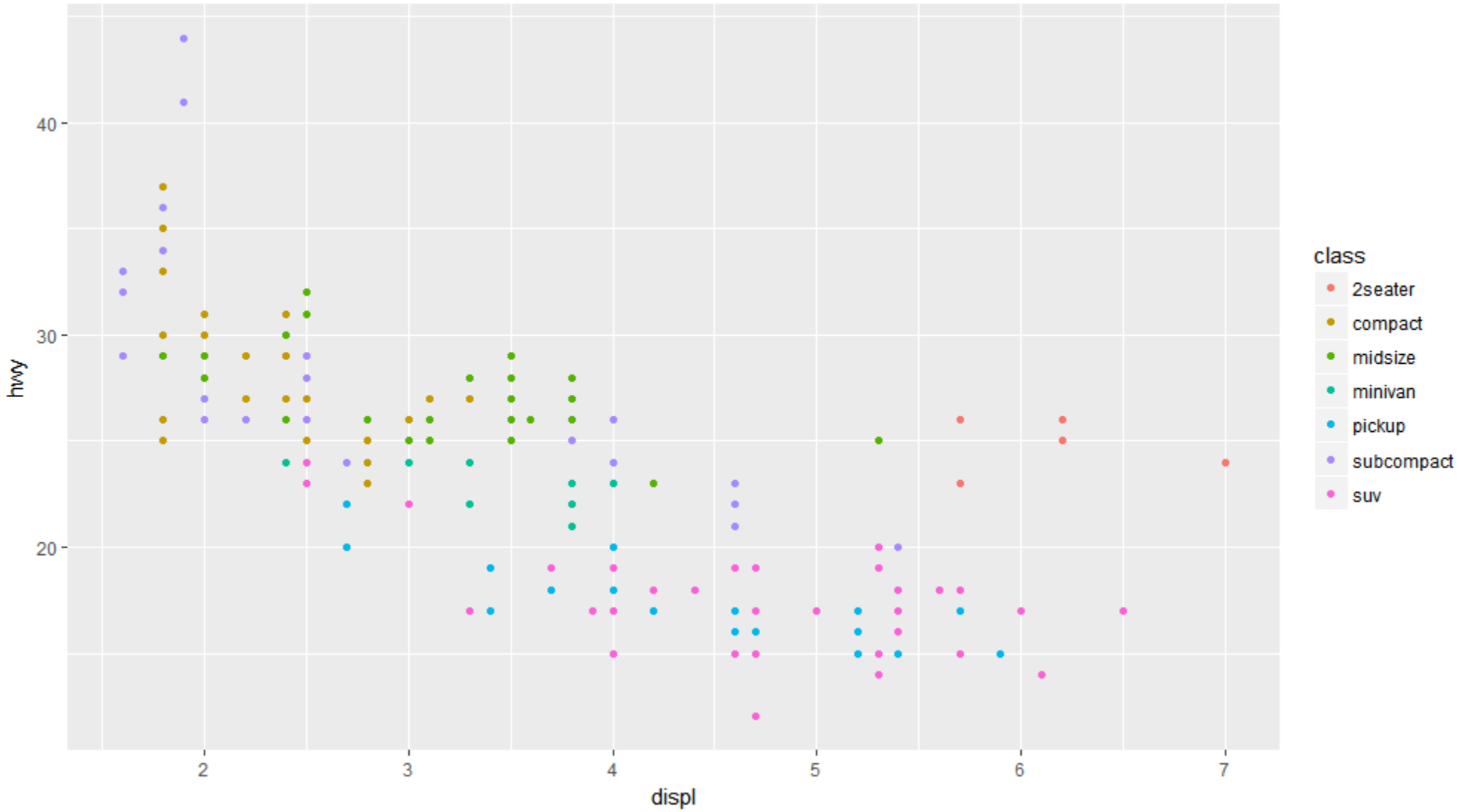
```
pvc <- ggplot(aes(x = carat, y = price), data = small_diamonds) +
        geom_point(aes(color = cut))

pvc <- ggplot(aes(x = carat, y = price, color = cut), data = small_diamonds) +
        geom_point()
```

- So there are various places to specify aesthetics
- You can also change aesthetics as you add layers

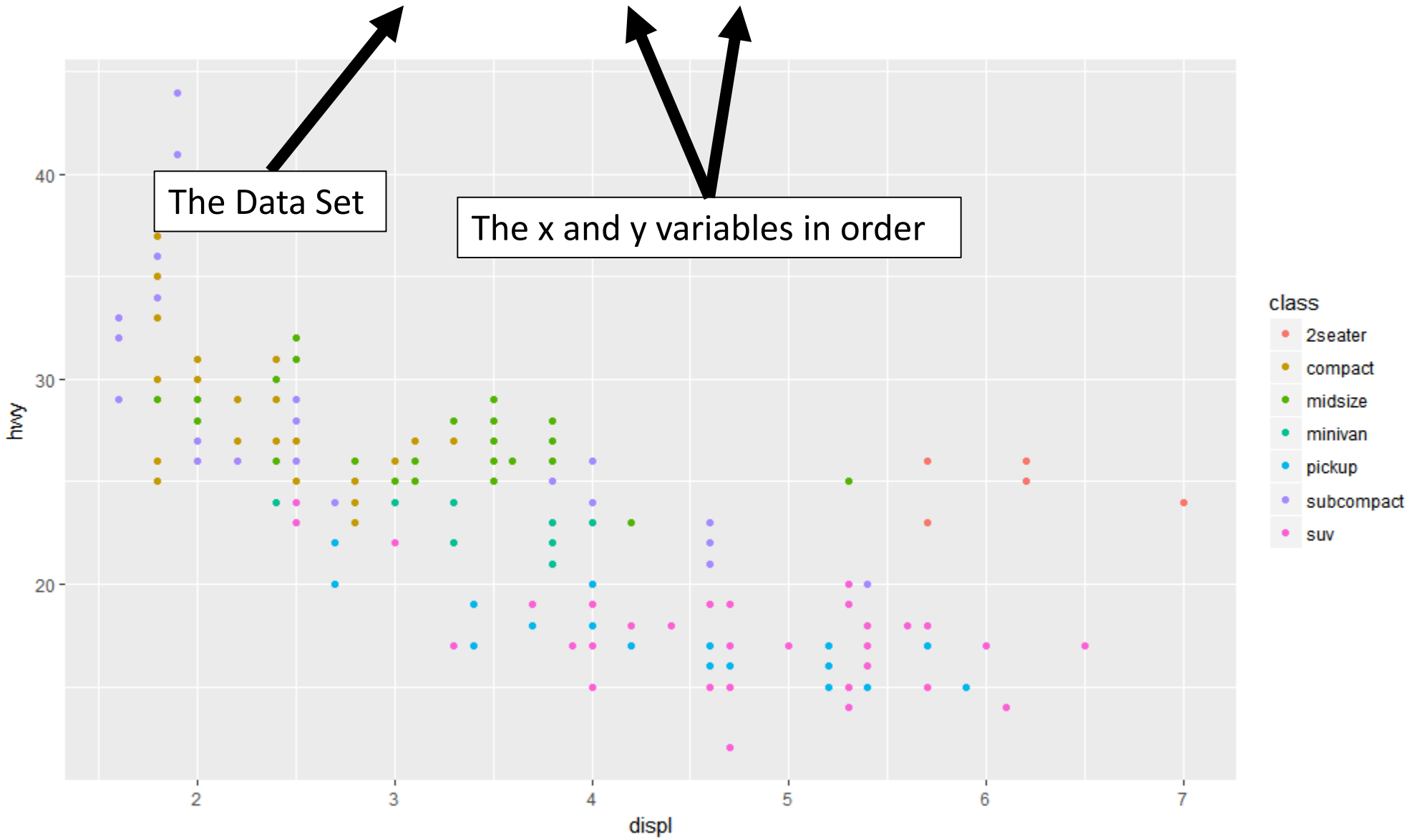
Alternative Specifications of **`aes()`**

- There are various equivalent ways of specifying the aesthetic mappings as shown on the following slides…

```
mpg_plot <- ggplot(mpg, aes(displ, hwy, color = class)) + geom_point()
```



```
mpg_plot <- ggplot(mpg, aes(displ, hwy, color = class)) + geom_point()
```



Specifying the aesthetics in the plot vs. in the layers

Aesthetic mappings can be supplied in the initial `ggplot()` call, in individual layers, or in some combination of both. All of these calls create the same plot specification:

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

Notice that ggplot2 accepts both British and American spellings of color.

https://rpubs.com/hadley/ggplot2-layers

R Example File

- We will now continue inside the R file: **`overview_of_ggplot2.R`**