zeppelin universität

zwischen
Wirtschaft Kultur Politik

Data Analytics and Visualisation

III Data Visualisation — The Appearance of Data Science

David Zimmermann

2017-04-23

Zeppelin University

Outline

Outline

1. Recap
2. Grammer of Graphics Intro
3. `ggplot2`
4. Geometrics
5. Scales
6. Colors
7. Labels
8. Themes
9. Facetting
10. Functions
11. Maps

Recap

Recap

What have we done yesterday?

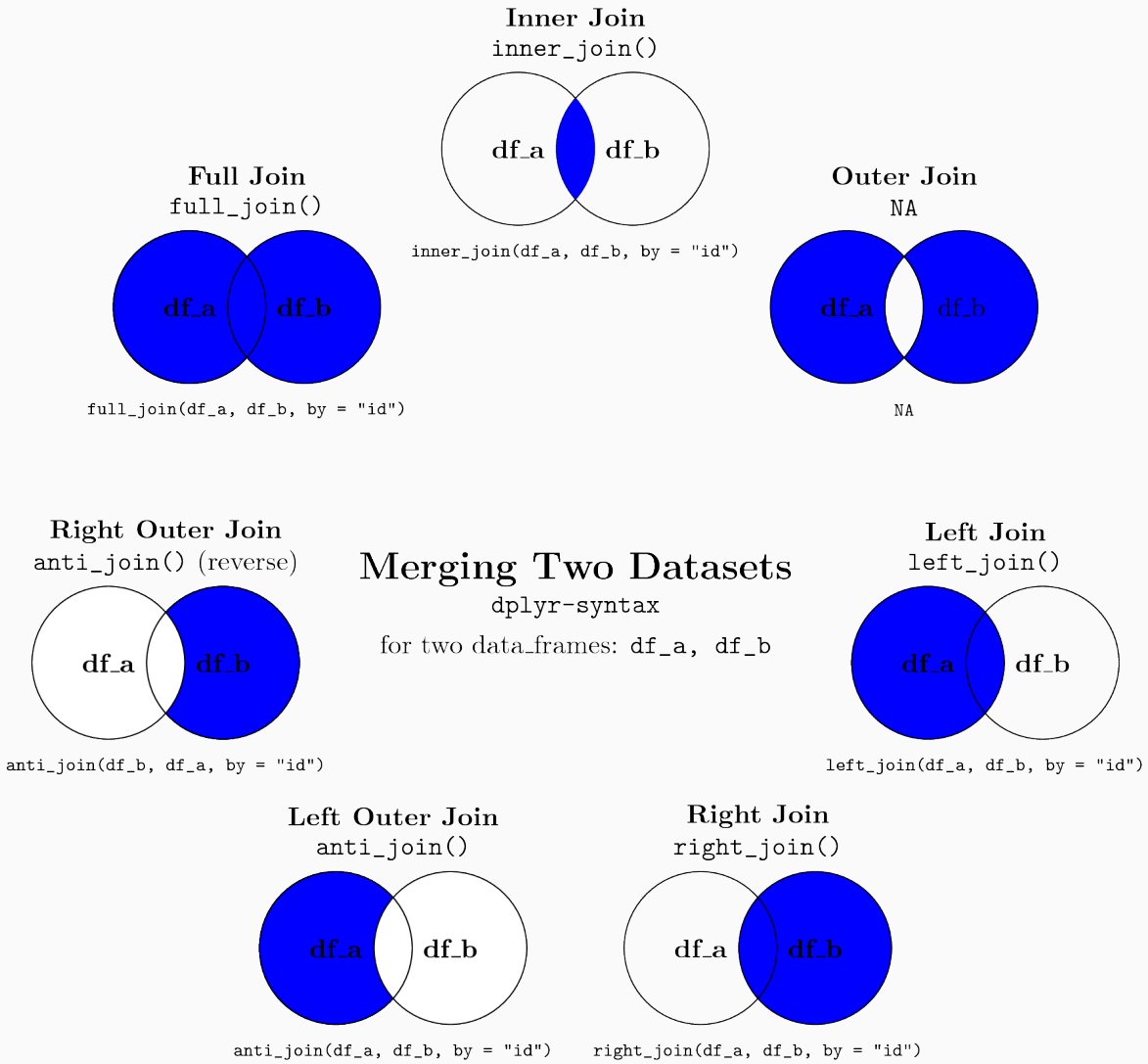
```
library(dplyr); library(readr)
library(tidyr); library(lubridate)
```

- Pipes: `f(y, x)` becomes `x %>% f(y, .)`
- Tibbles: Use `data_frame()` or `tibble()` to create a new data frame
- Hadleyverse: ...
- Data I/O: `readr` and `read_csv` or `write_csv`
- Tidy Data: `tidyr` using `gather/spread` and `separate/unite`
- Time and Dates: `lubridate` saves your day

Recap: Data Manipulation using `dplyr`

- `filter` for filtering rows/observations
- `select` for selecting columns/variables
- `arrange` for arranging rows
- `distinct` for selecting distinct observations
- `mutate` for creating new variables
- `group_by` for grouping operations
- `summarise` for summarising the data

Recap: Merging Data using Joins



Grammer of Graphics

Grammer of Graphics

"Grammer of Graphics" (Wilkinson 2005)

A redefinition of what a graph is and should be

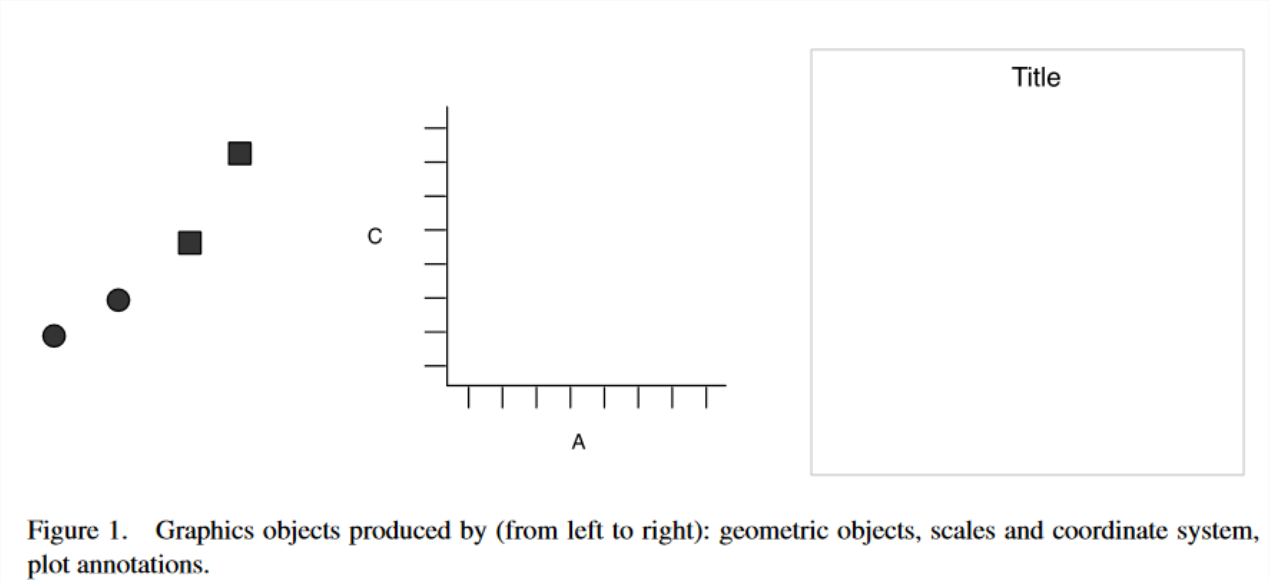


Figure 1. Graphics objects produced by (from left to right): geometric objects, scales and coordinate system, plot annotations.

Source: Hadley (2010) - A Layered Grammer of Graphics

Components of a Graph

- Data
- Aesthetic Mappings
- Geometrics
- Scales
- Coordinate System
- Annotations

A Basic ggplot

```
ggplot(data = df, aes(x = carat, y = price)) + geom_point()
```

Data

carat	price
1.05	4504
1.00	7056
2.33	18119
...	...

+

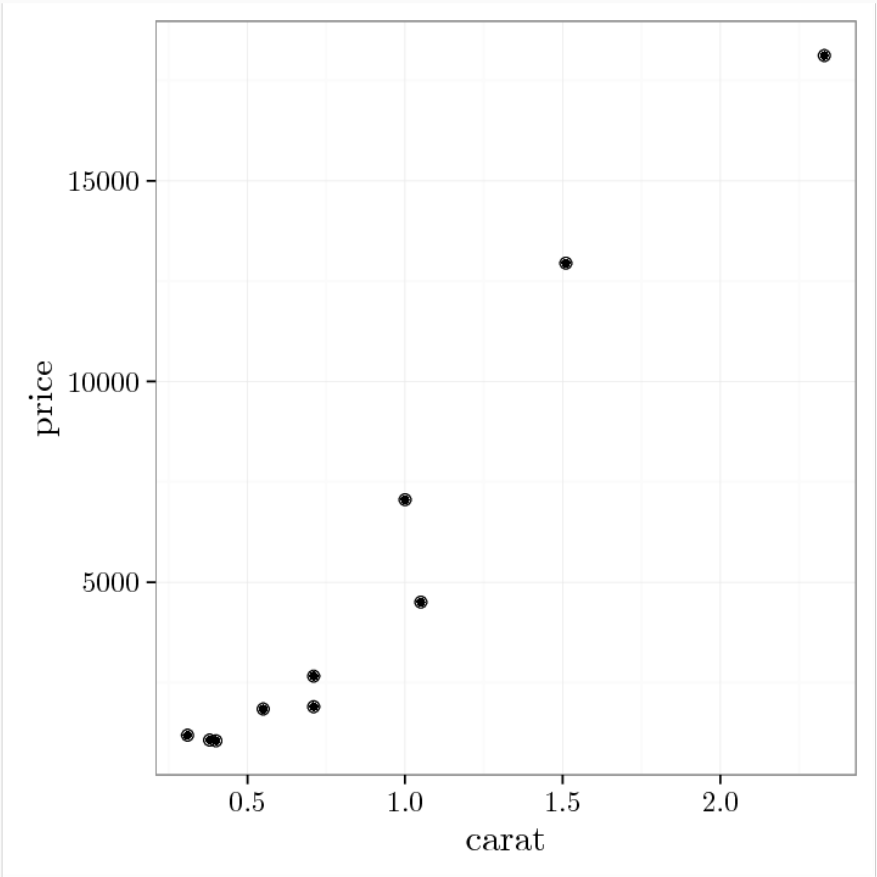
Aesthetics

Variable	Mapping
x	carat
y	price

+

Geometric

`geom_point`



Adding Color as an Aesthetic

```
ggplot(df, aes(x = carat, y = price, color = cut)) +
  geom_point()
```

Data

carat	price	cut
1.05	4504	Ideal
1.00	7056	Premium
2.33	18119	Good
...	...	...

+

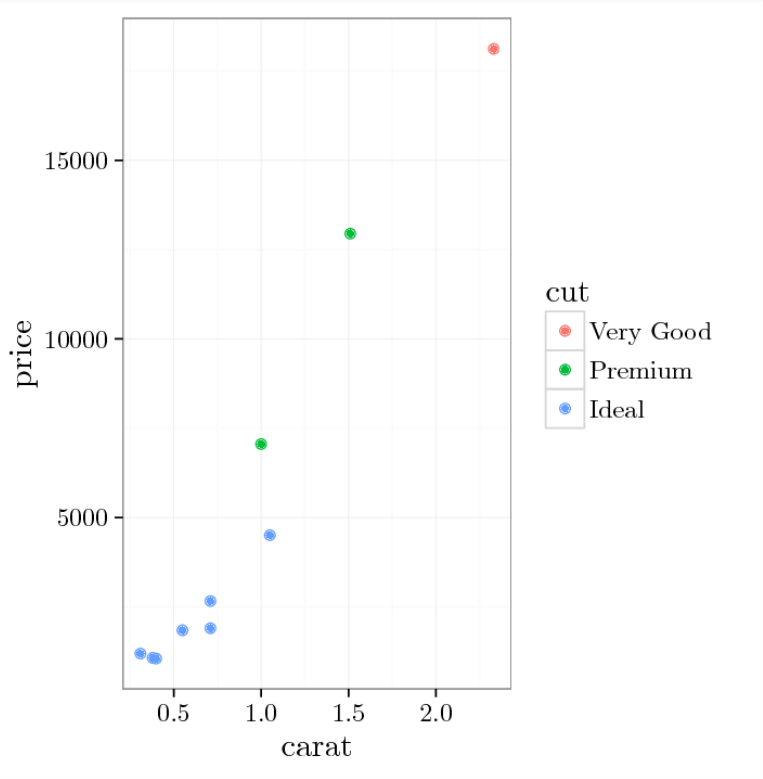
Aesthetics

Mapping	Variable
x	carat
y	price
color	cut

+

Geometric

`geom_point`



Adding Shape as an Aesthetic

```
ggplot(df, aes(x = carat, y = price, shape = cut)) +
  geom_point()
```

Data

carat	price	cut
1.05	4504	Ideal
1.00	7056	Premium
2.33	18119	Good
...	...	...

+

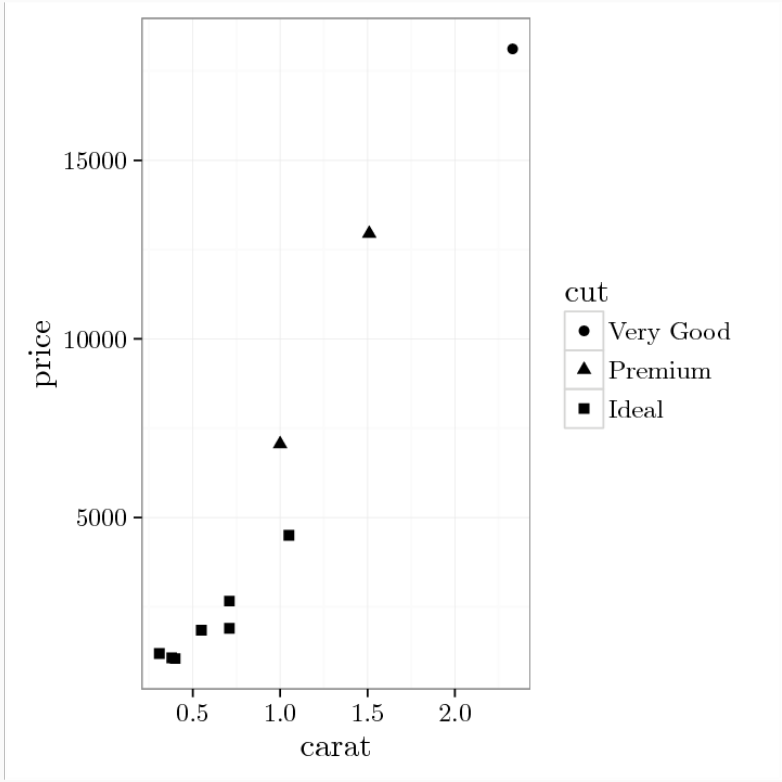
Aesthetics

Mapping	Variable
x	carat
y	price
shape	cut

+

Geometric

`geom_point`



Combining Color and Shape

```
ggplot(df, aes(x = carat, y = price, color = cut,
               shape = cut)) + geom_point()
```

Data

carat	price	cut
1.05	4504	Ideal
1.00	7056	Premium
2.33	18119	Good
...	...	...

+

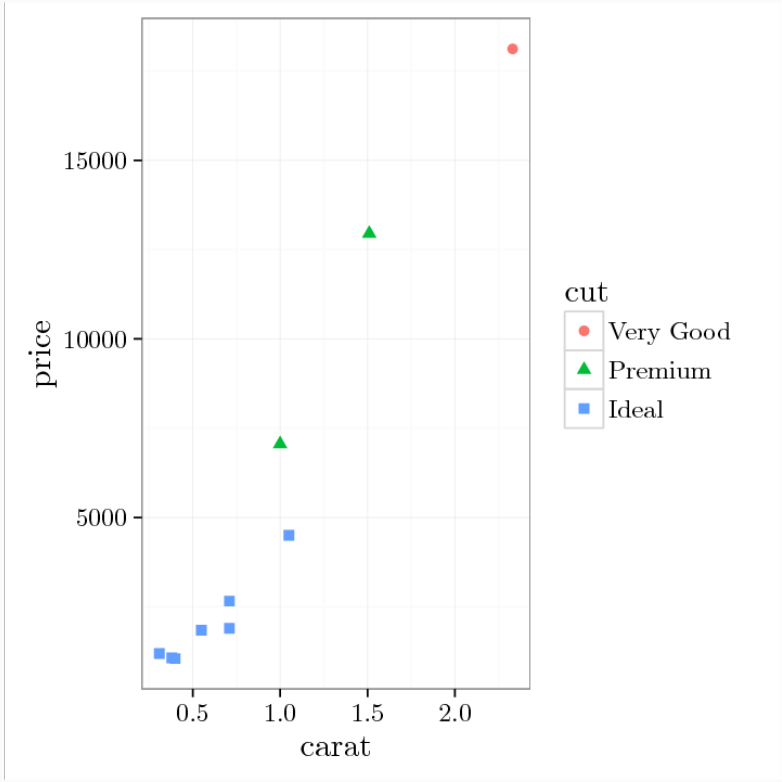
Aesthetics

Mapping	Variable
x	carat
y	price
color	cut
shape	cut

+

Geometric

`geom_point`



Adding Size as an Aesthetic

```
ggplot(df, aes(x = carat, y = price, color = cut, shape = cut,
               size = depth)) + geom_point()
```

Data

carat	price	cut	depth
1.05	4504	Ideal	61.9
1.00	7056	Premium	62.7
2.33	18119	Good	60.7
...	...	...	...

+

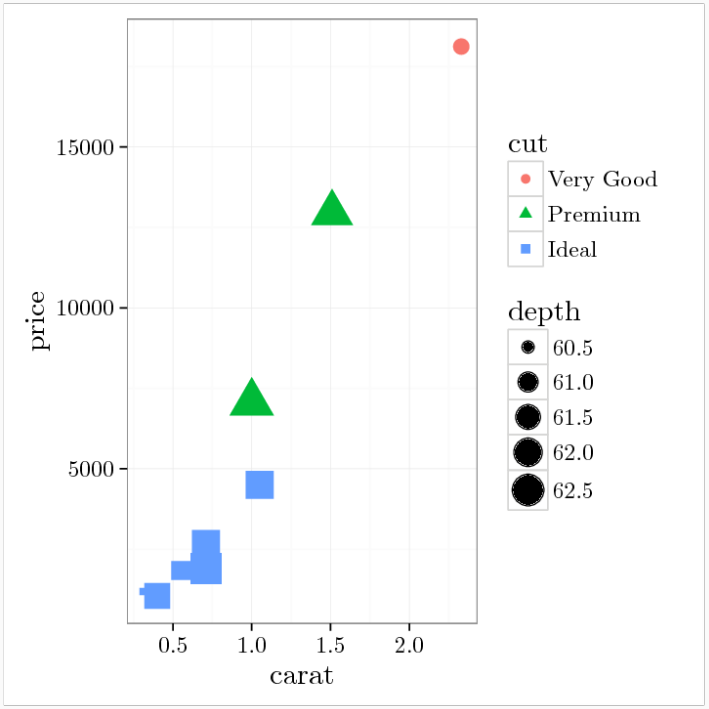
Aesthetics

Mapping	Variable
x	carat
y	price
shape	cut
color	cut
size	depth

+

Geometric

`geom_point`



Creating a Line Plot

```
ggplot(df2, aes(x = date, y = price, color = stock)) +
  geom_line()
```

Data

stock	date	price
A	1	98.18
A	2	98.81
...	...	...
B	1	99.46
B	2	99.15
...	...	...

+

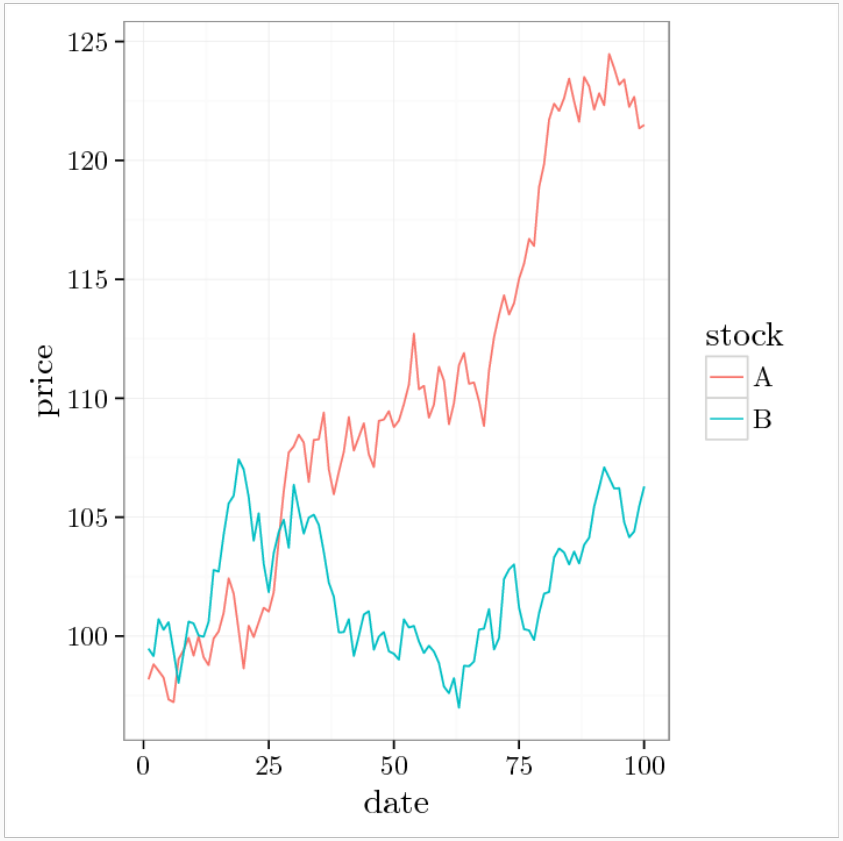
Aesthetics

Mapping	Variable
x	date
y	price
color	stock

+

Geometric

`geom_line`



Creating a Bar Plot

```
ggplot(df3, aes(x = name, fill = name)) + geom_bar()
```

Data

name
Alice
Bob
Alice
Charlie
...

+

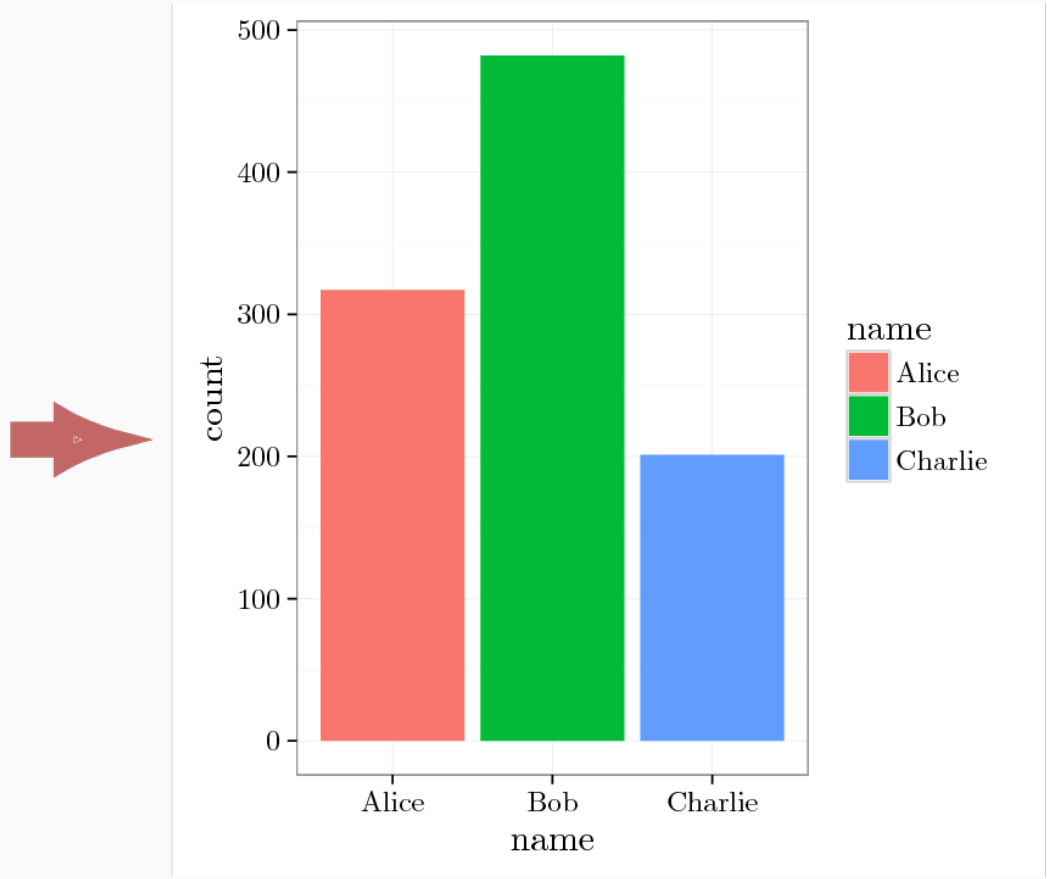
Aesthetics

Mapping	Variable
x	name
fill	name

+

Geometric

`geom_bar`



ggplot2

ggplot2

Mostly an art of knowing google and expanding basic plots...

ggplot2 is very fast at creating basic (goodish looking) plots, for really good plots some additional formatting is needed.

Helpful links include

- Documentation:
  - http://docs.ggplot2.org/current/
- R Cookbook:
  - http://www.cookbook-r.com/Graphs/
- ggplot cheatsheet:
  - https://www.rstudio.com/wp-content/uploads/2015/12/ggplot2-cheatsheet-2.0.pdf
- Stackoverflow:
  - https://www.stackoverflow.com/tags/ggplot2

Basic ggplot

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

source: R for Data Science

Usually wirtten as

```
ggplot(<DATA>, aes(<MAPPINGS>)) +
  <GEOM_FUNCTION>()
```

Decomposing a plot

- Data
  - a tidy `tibble`, `data_frame`, `data.frame`, etc.
- Aesthetic Mappings
  - What variable is mapped to what aesthetic (what "goes" on the x, y-axis, what determines color, fill, shape, size)?
- Geometrics
  - `0d`: point or text, ...
  - `1d`: path, line (ordered line), ...
  - `2d`: polygon, interval, ribbons, heatmaps, maps...

Plus additionals (automatic):

- Scales for aesthetics
  - `x` and `y`: `continuous`, `reverse`, `date`, `log`, `sqrt`
  - `color`: `continuous`, `gradient`, `discrete`, ...
  - ...
- Coordinate System
  - `cartesian`, `flip`, `fixed`, `polar`, `map`, ...

Data

Preparing some Data

```
library(ggplot2)
library(dplyr)
set.seed(12345)
df <- diamonds %>% sample_n(10) %>%
  select(carat, depth, cut, price)
df %>% head
```

```
## # A tibble: 6 × 4
##   carat depth     cut price
##   <dbl> <dbl>   <ord> <int>
## 1  0.40  61.6   Ideal  1050
## 2  0.55  60.9   Ideal  1841
## 3  0.31  60.4   Ideal  1188
## 4  0.71  62.4   Ideal  1899
## 5  1.51  62.4 Premium 12948
## 6  1.05  61.9   Ideal  4504
```

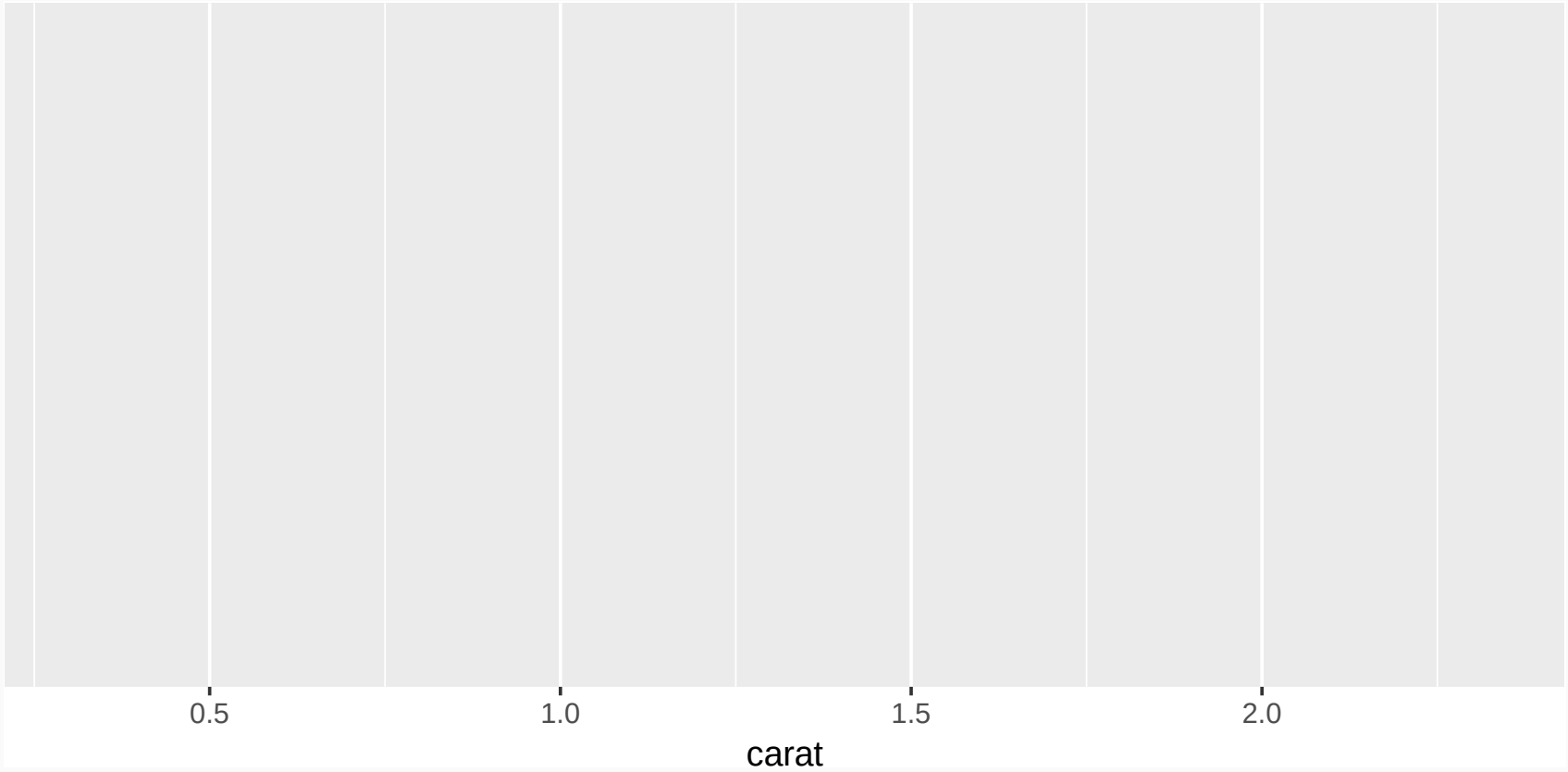
Building a ggplot

An empty ggplot

```
ggplot(df) # same as ggplot(data = df)
```

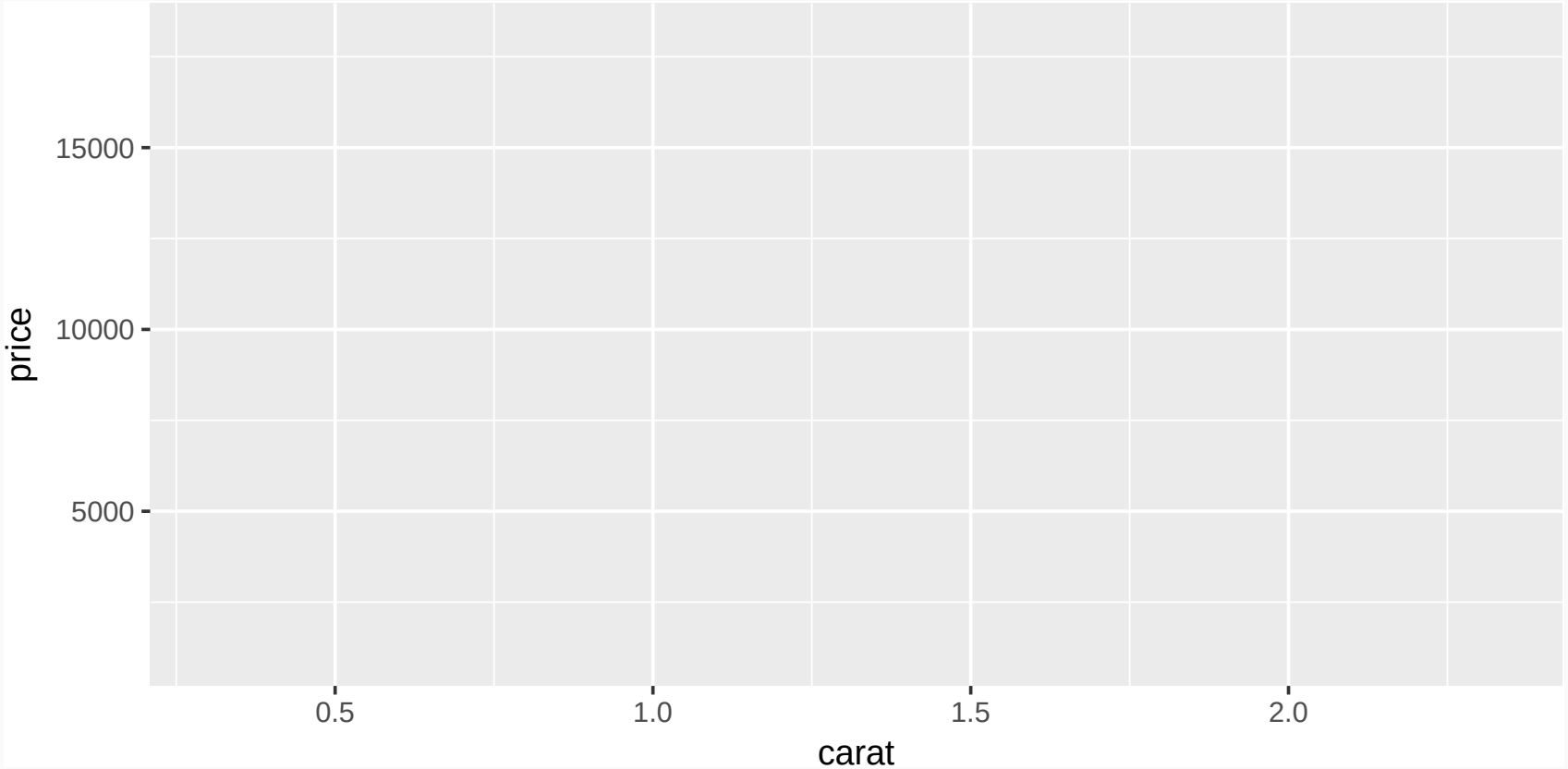
Building a ggplot cont'd

```
ggplot(df, aes(x = carat))
```



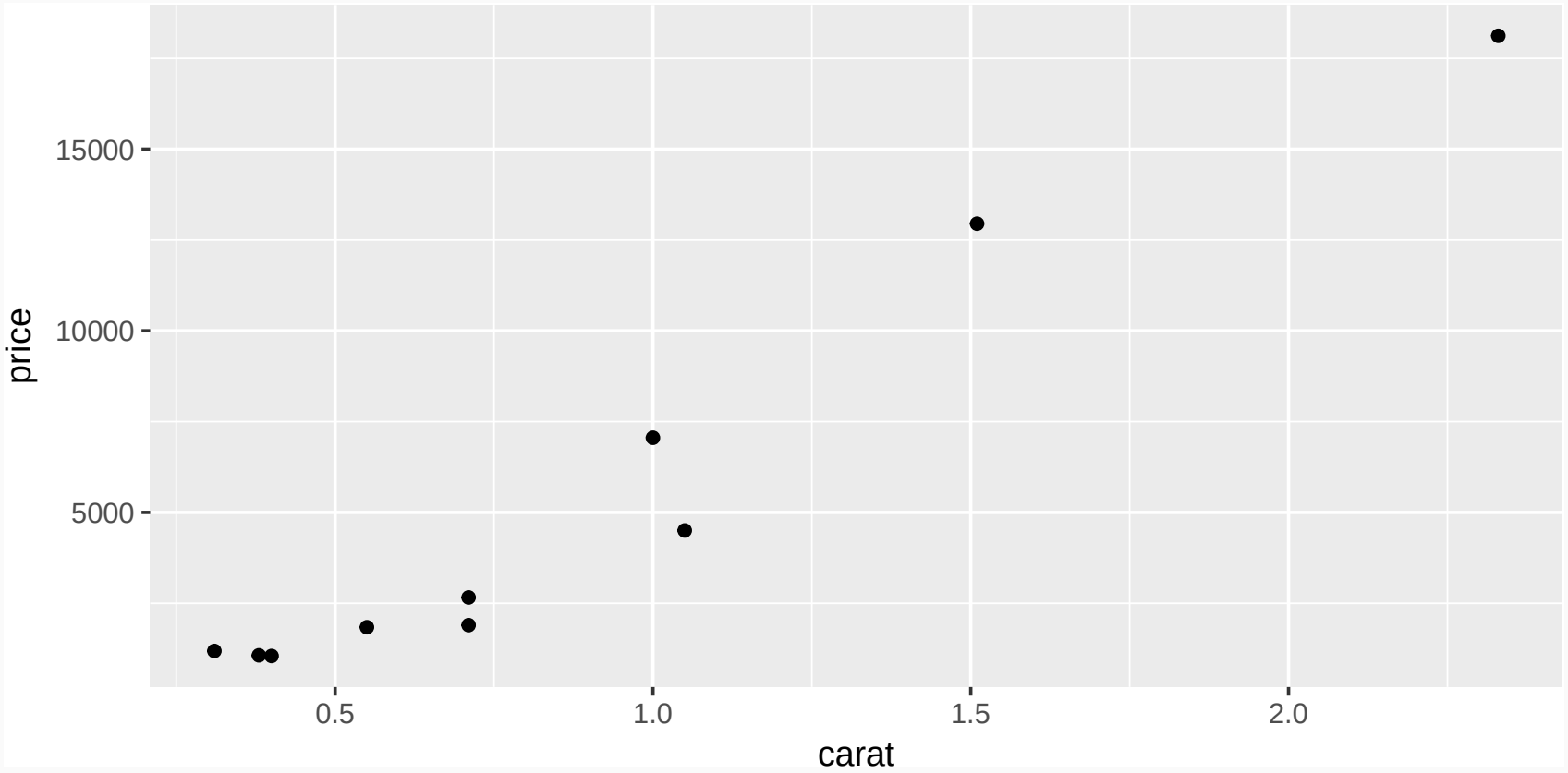
Building a ggplot cont'd

```
ggplot(df, aes(x = carat, y = price))
```



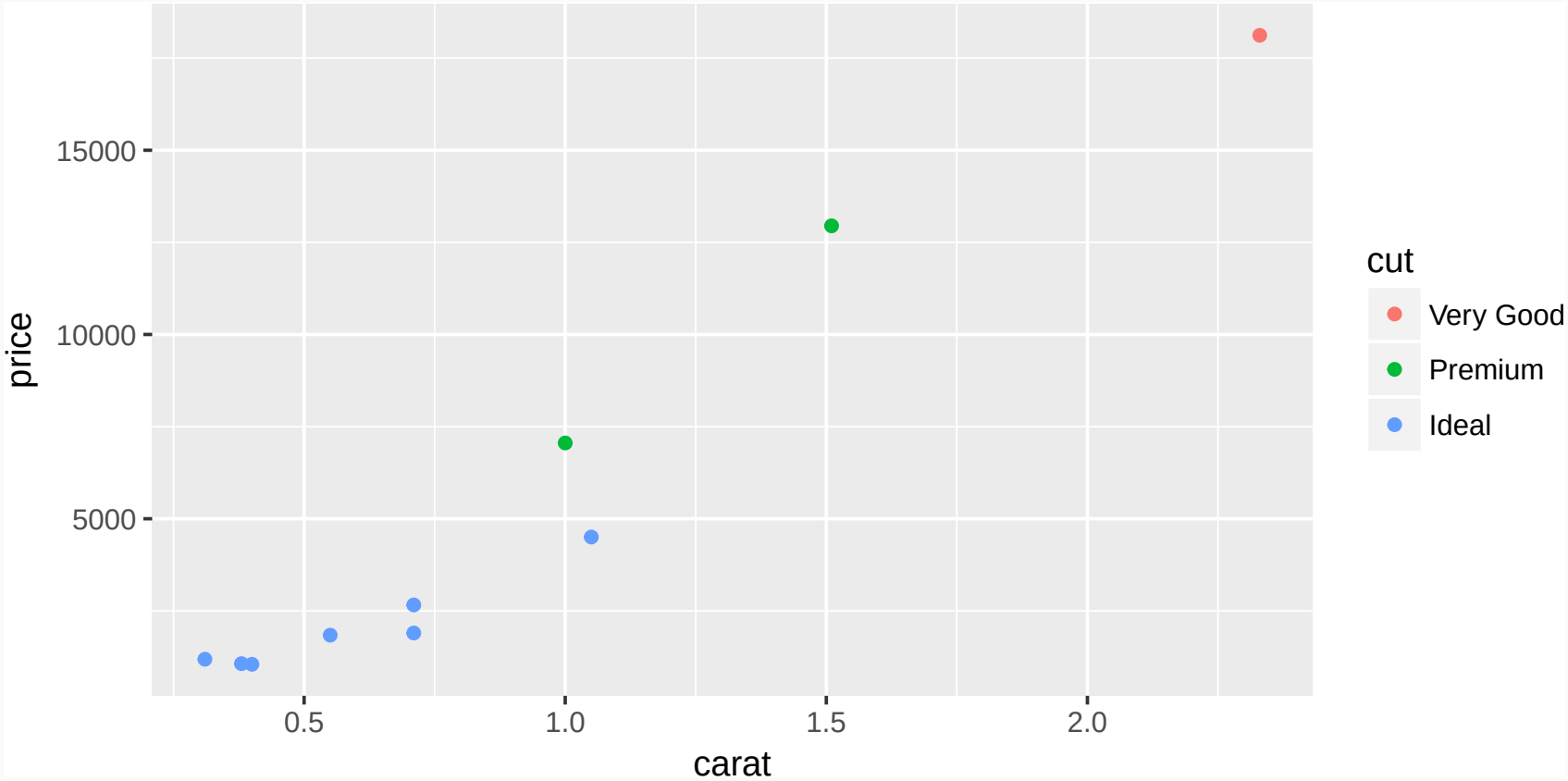
Building a ggplot cont'd

```
ggplot(df, aes(x = carat, y = price)) +
  geom_point()
```



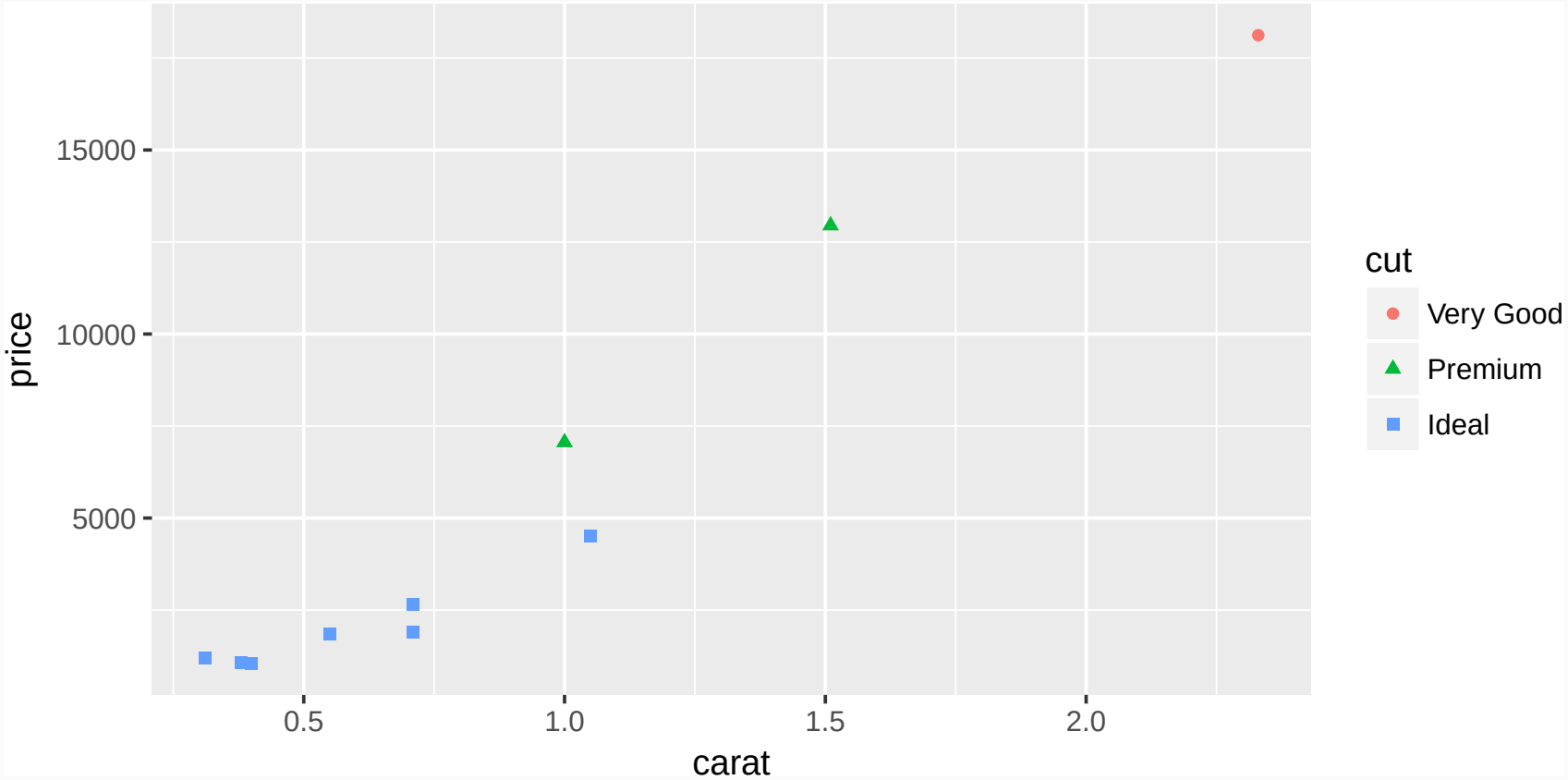
Building a ggplot cont'd

```
ggplot(df, aes(x = carat, y = price, color = cut)) +
  geom_point()
```



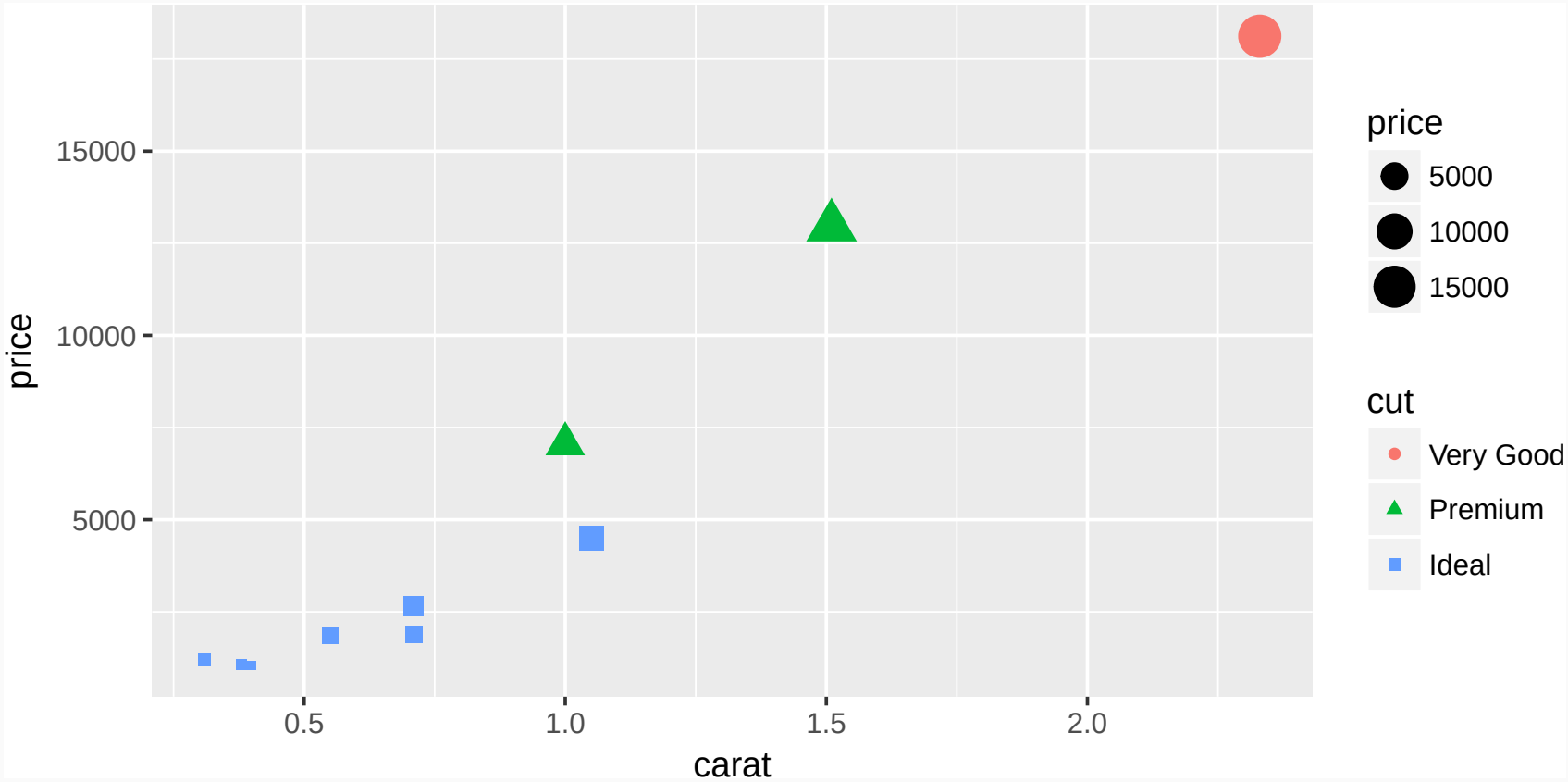
Building a ggplot cont'd

```
ggplot(df, aes(x = carat, y = price, color = cut,
               shape = cut)) +
  geom_point()
```



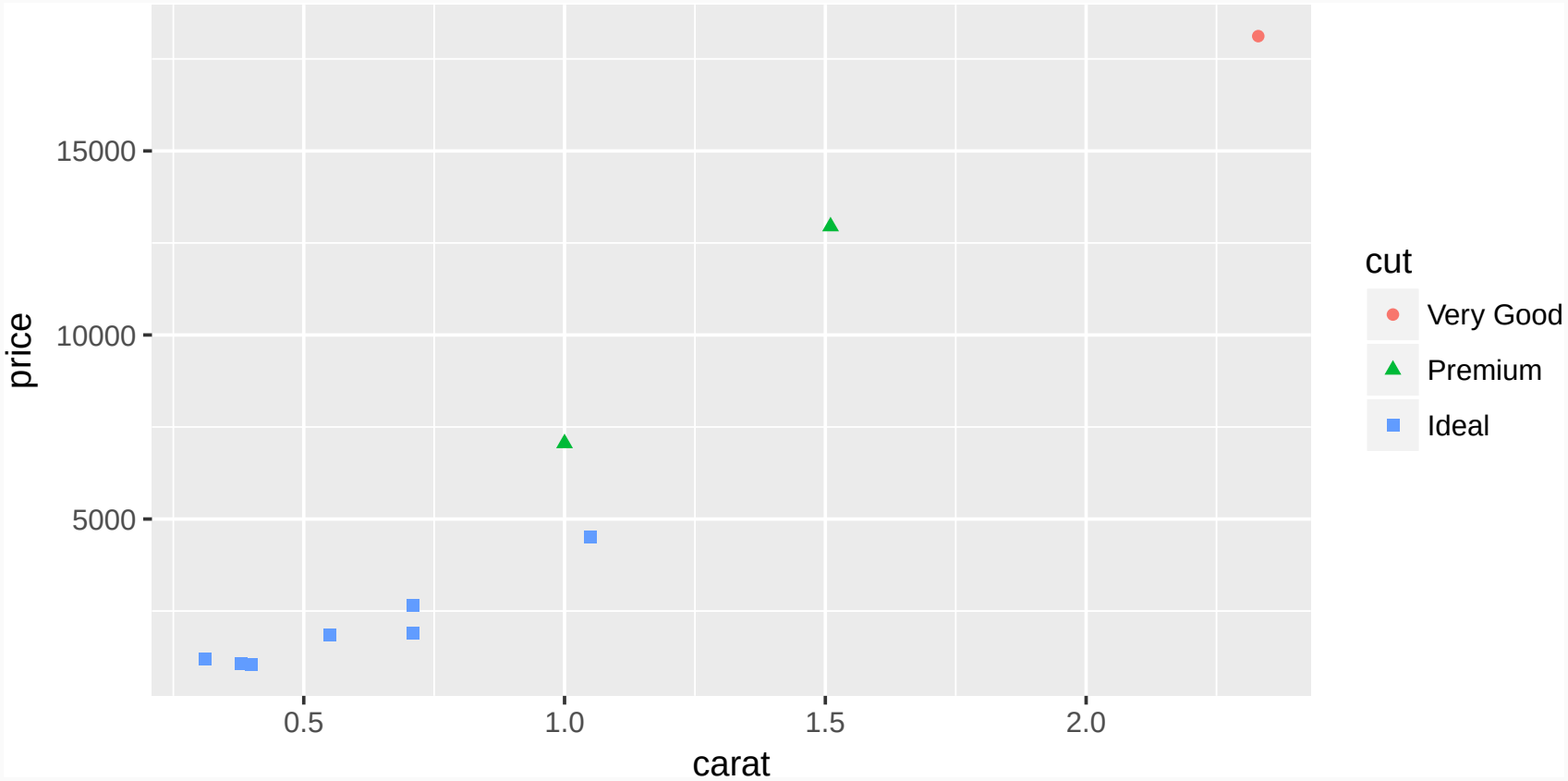
Building a ggplot cont'd

```
ggplot(df, aes(x = carat, y = price, color = cut,
               shape = cut, size = price)) +
  geom_point()
```



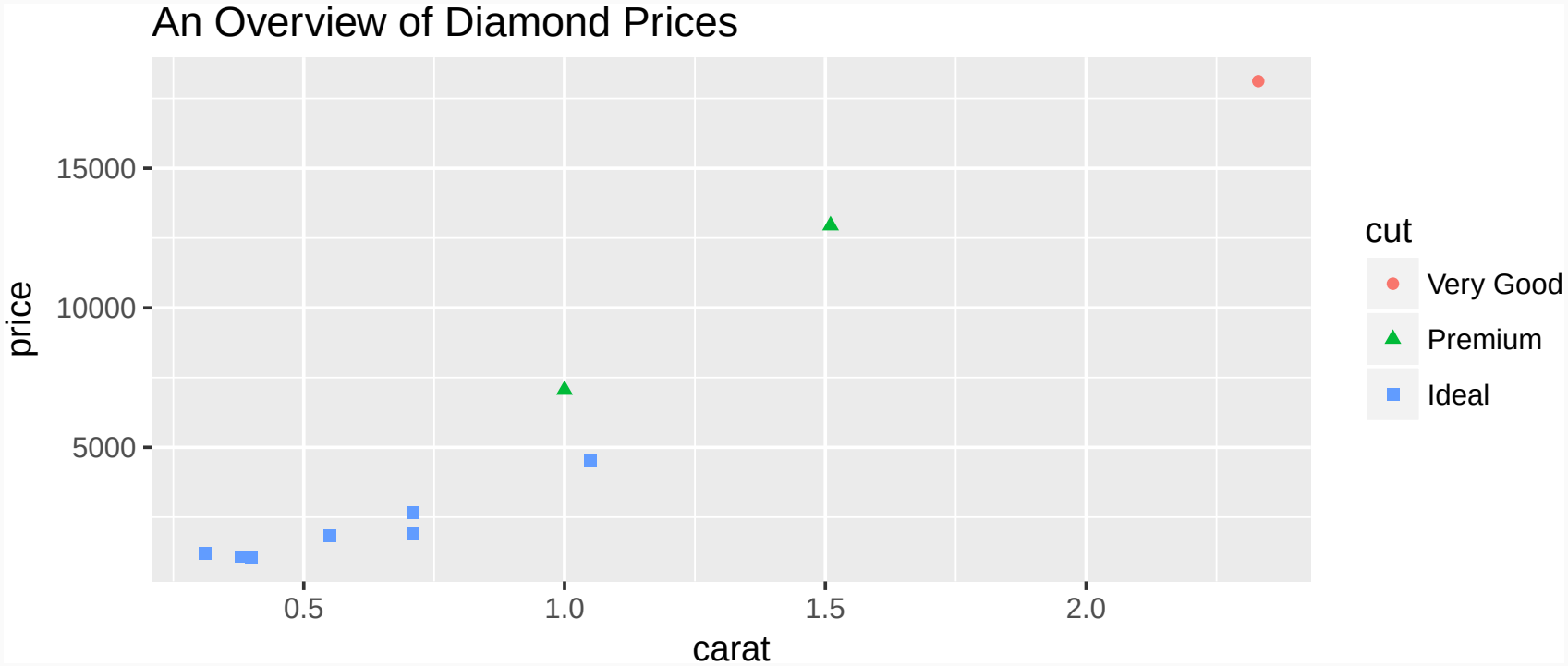
Assigning a Plot

```
plot1 <- ggplot(df, aes(x = carat, y = price, color = cut,
                        shape = cut))
plot1 + geom_point()
```



Adding to a Plot

```
plot1 <- ggplot(df, aes(x = carat, y = price, color = cut,
                        shape = cut))
plot1 + geom_point() +
  ggtitle("An Overview of Diamond Prices")
```



Saving a Plot

Save a plot-object using ggsave

```
plot1 <- ggplot(df, aes(x = carat, y = price, color = cut,
                        shape = cut))

plot1 <- plot1 + geom_point()

ggsave("plots/plot1.pdf", plot1, width = 7, height = 5)
ggsave("plots/plot1.png", plot1, width = 7, height = 5,
       dpi = 600)
```

Exercises: `geom_point()`

Geometrics

Mapping Aesthetic

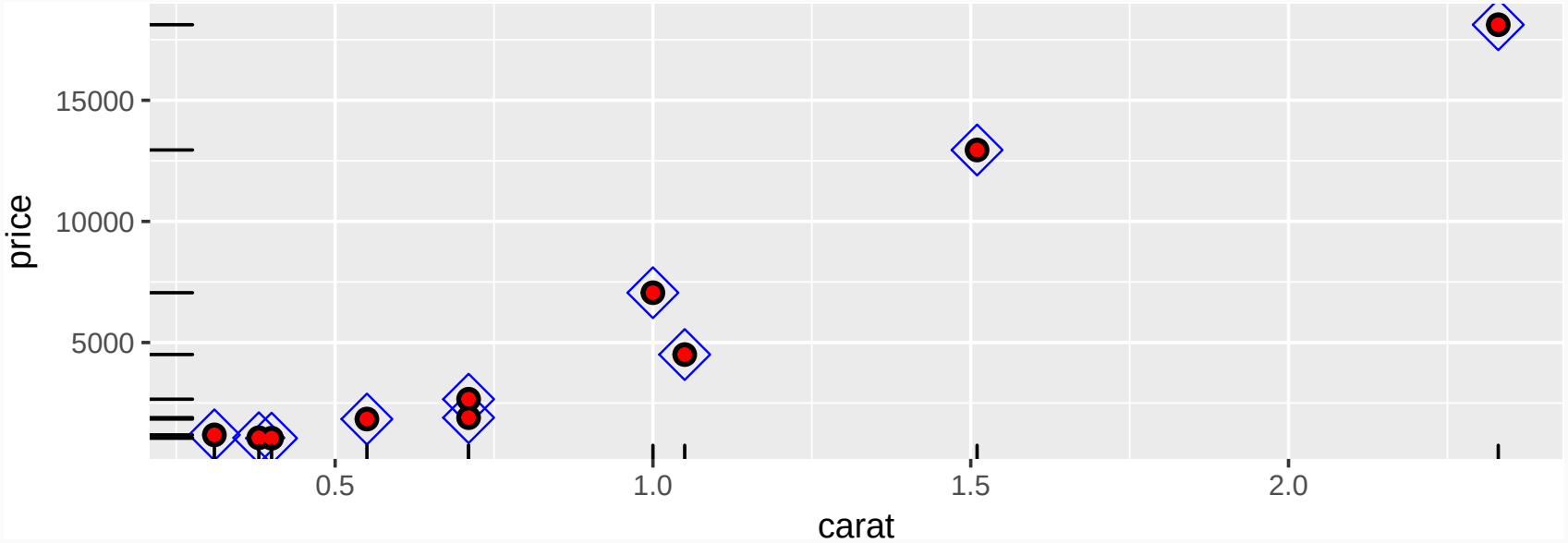
Aesthetics go into the `aes()`-part of a `ggplot` if mapped to a variable, otherwise into the geometric.

- `x`
- `y`
- `color`
- `shape`
- `linetype`
- `fill`
- `size`
- `alpha`
- `...`

Multiple Geometrics

Combining geometrics is possible

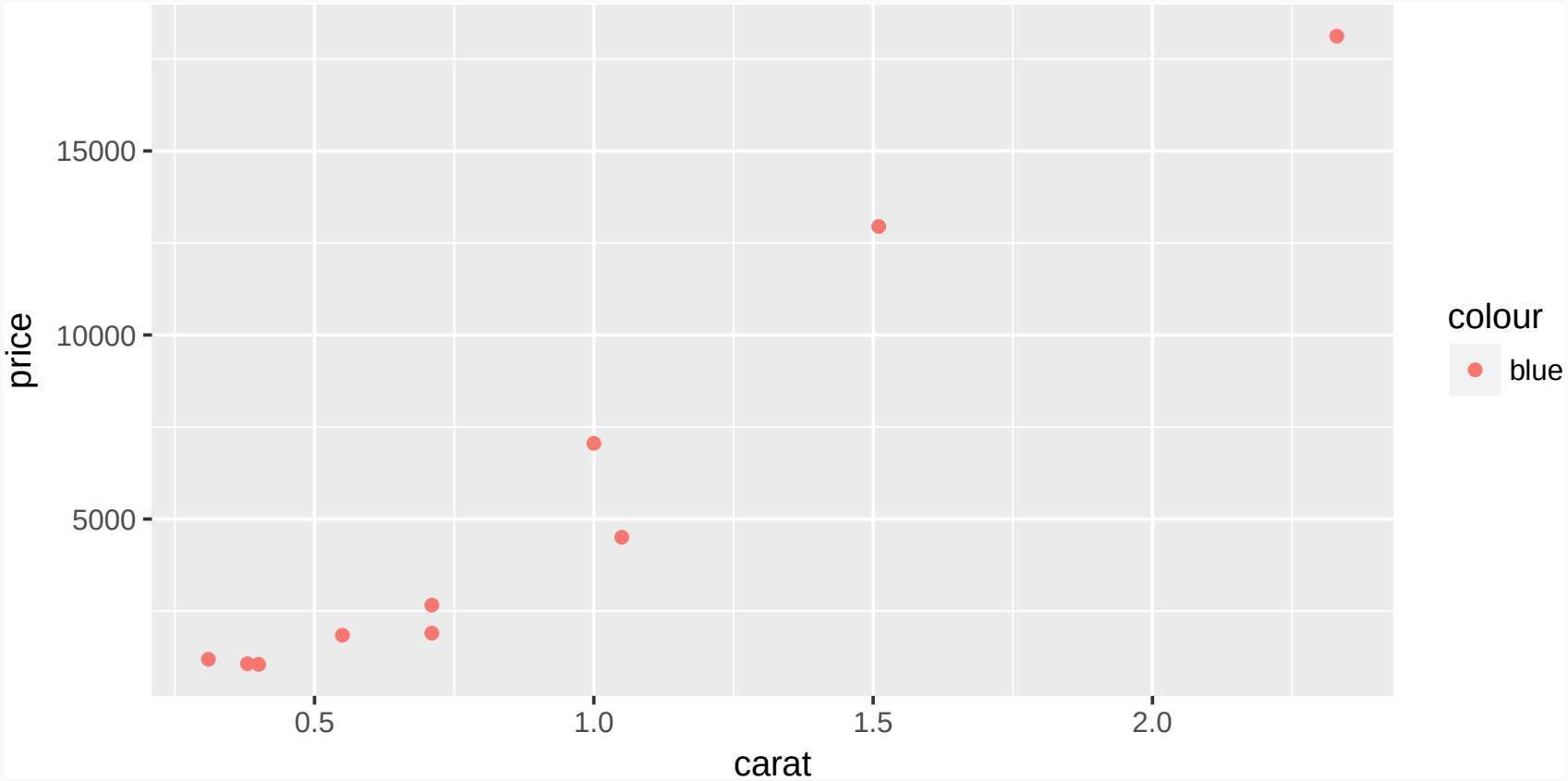
```
ggplot(df, aes(x = carat, y = price)) +
  geom_point(size = 5, color = "blue", shape = 5) +
  geom_point(size = 3, color = "black") +
  geom_point(size = 1.5, color = "red") +
  geom_rug()
```



Aesthetics cont'd

What happens if we want to have a specific color

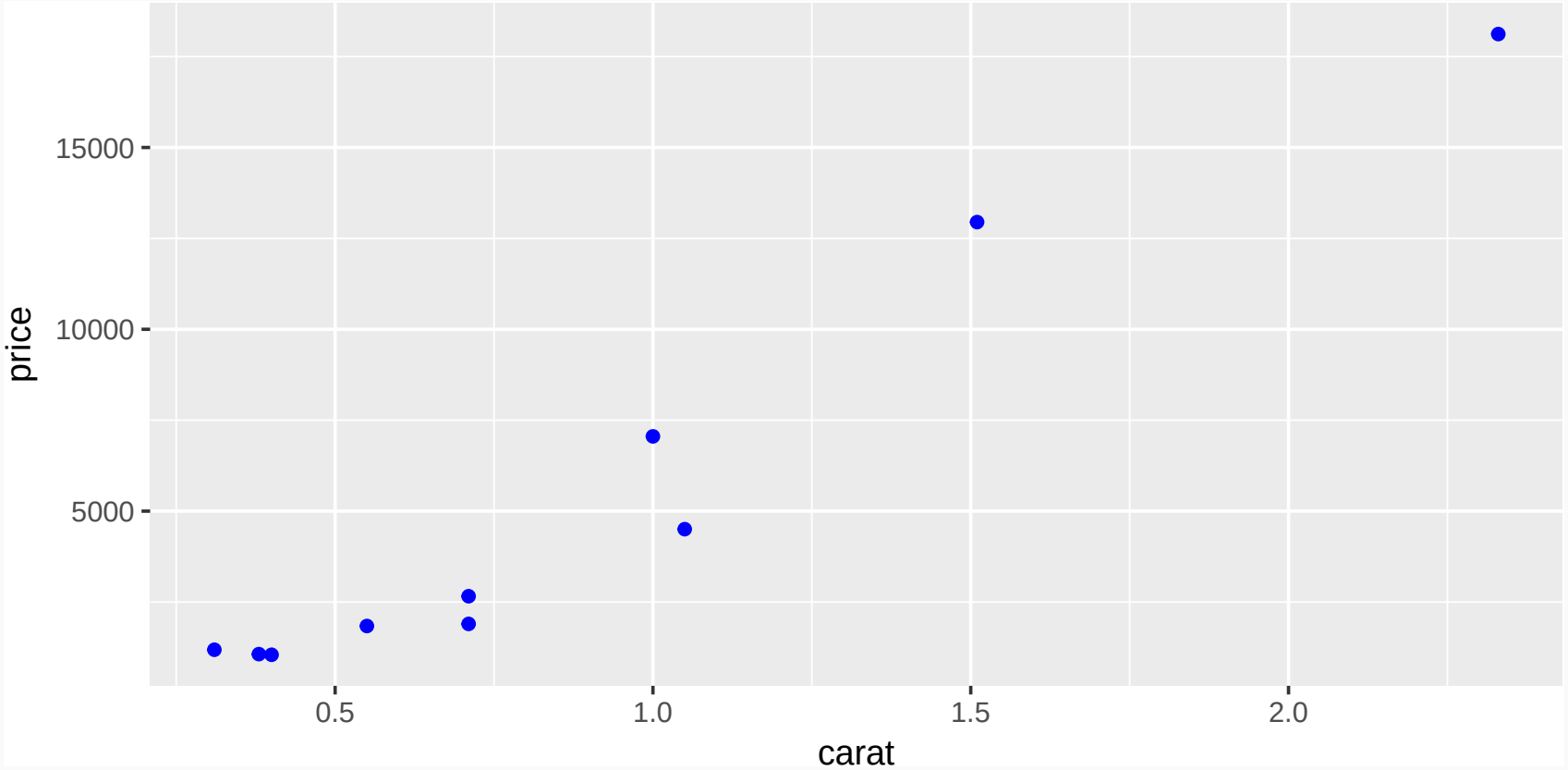
```
ggplot(df, aes(x = carat, y = price, color = "blue")) +
  geom_point()
```



Aesthetics cont'd

What happens if we assign these values in the geom

```
ggplot(df, aes(x = carat, y = price)) +
  geom_point(color = "blue")
```



Geoms

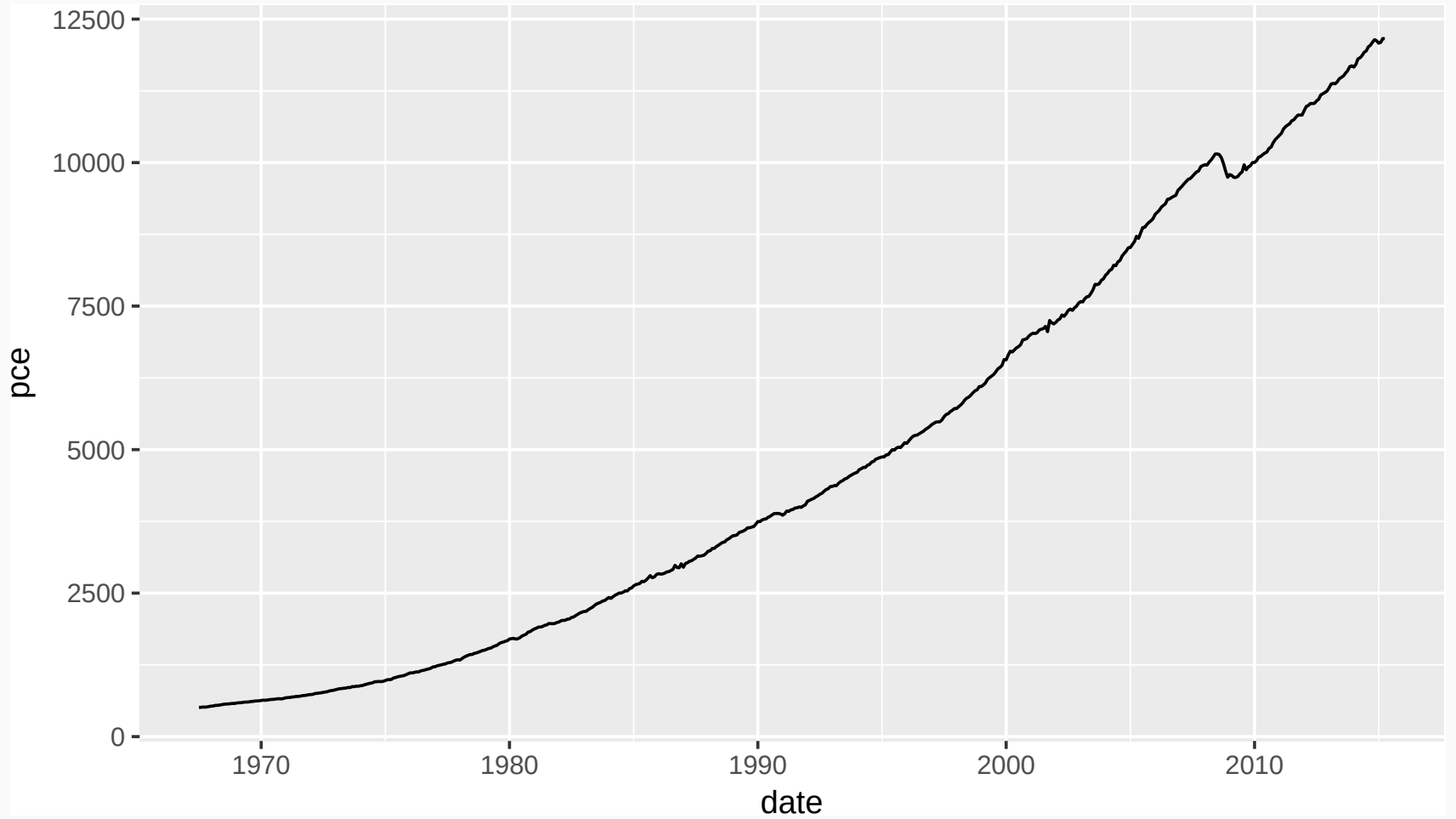
Geoms are added using the +-operator at the end

- `geom_point`
- `geom_line`
- `geom_area`
- `geom_bar`
- `geom_histogram`
- `geom_density`
- `geom_boxplot`
- `geom_smooth`

Detailed description: http://docs.ggplot2.org/current/

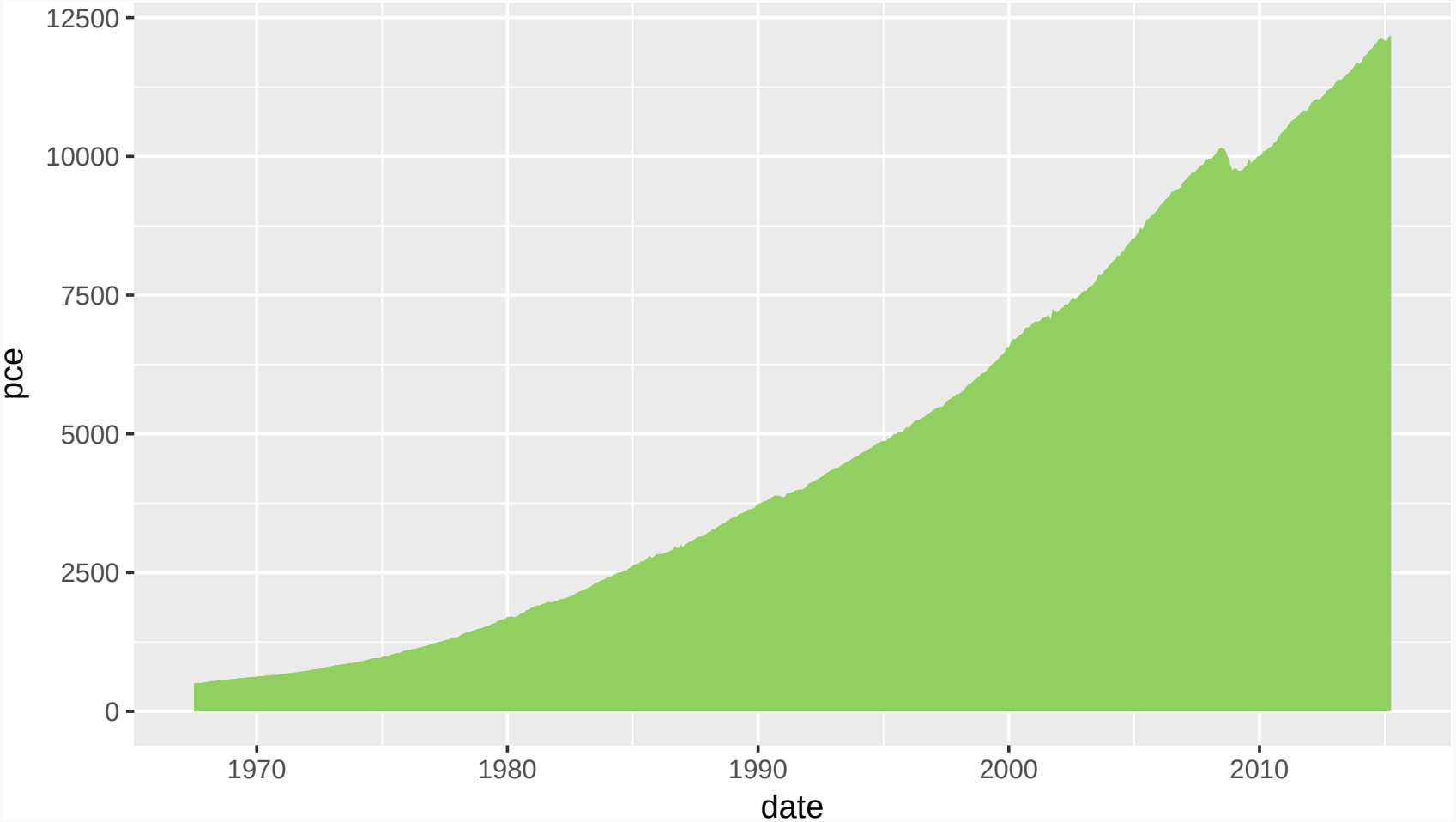
Line Chart

```
ggplot(economics, aes(x = date, y = pce)) + geom_line()
```



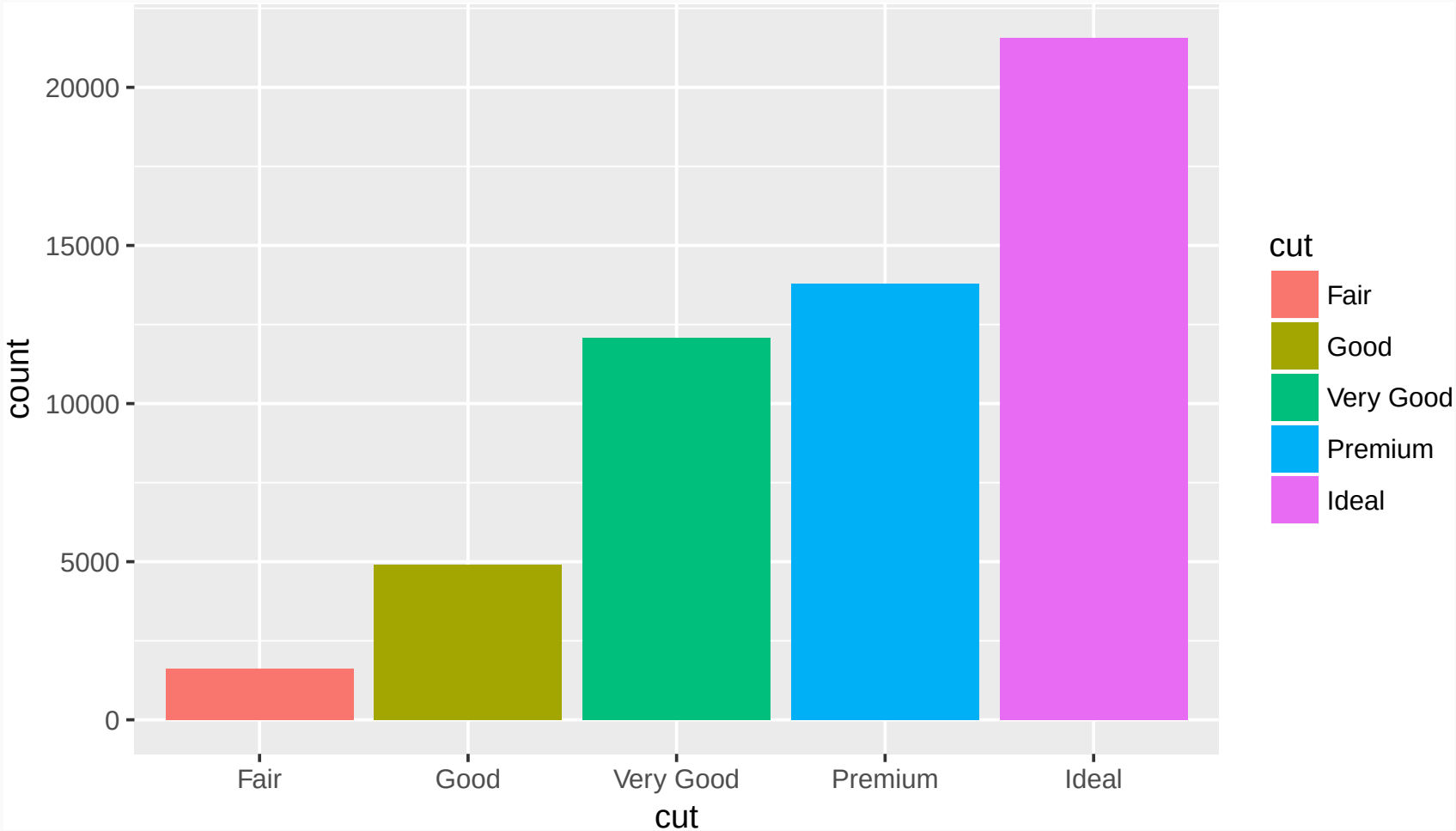
Area Chart

```
ggplot(economics, aes(x = date, y = pce)) +
  geom_area(fill = "#91cf60")
```



Bar Chart

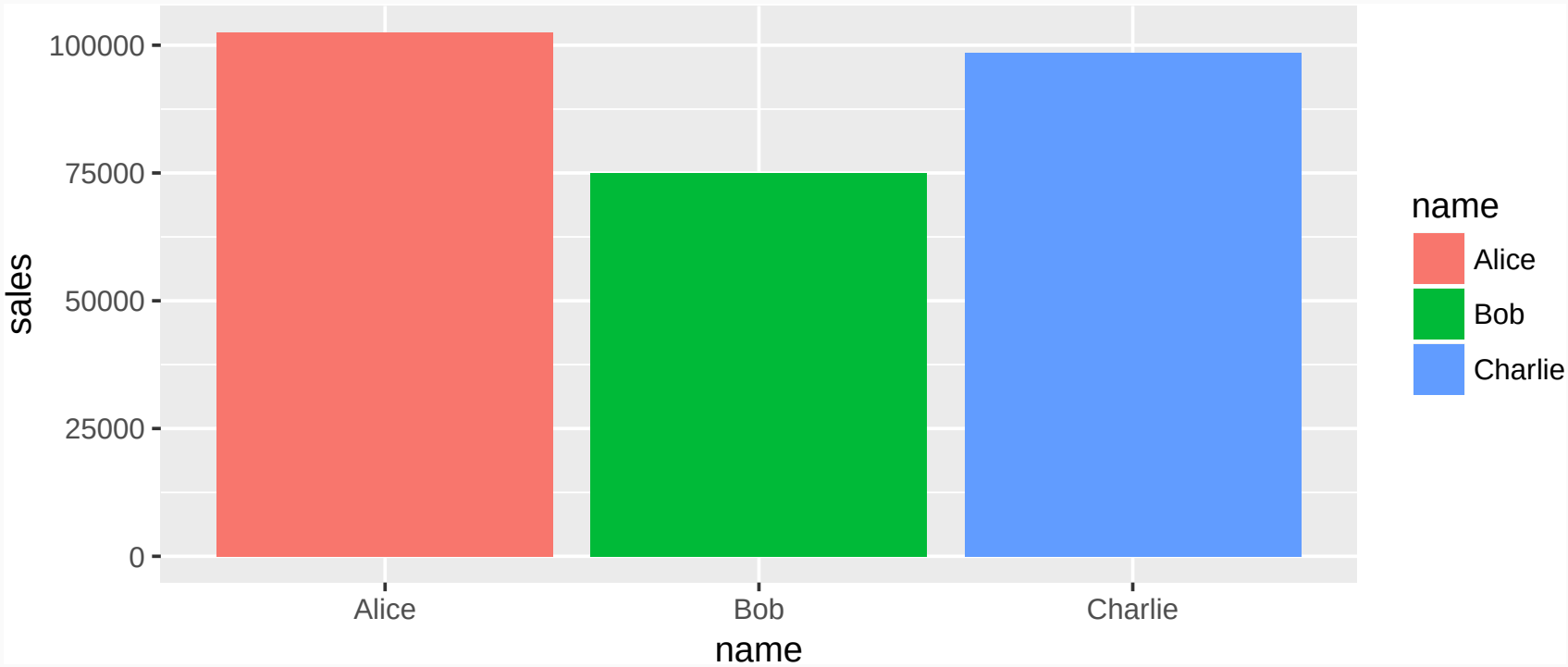
```
ggplot(diamonds, aes(x = cut, fill = cut)) + geom_bar()
```



Bar Chart 2 aka Col Chart

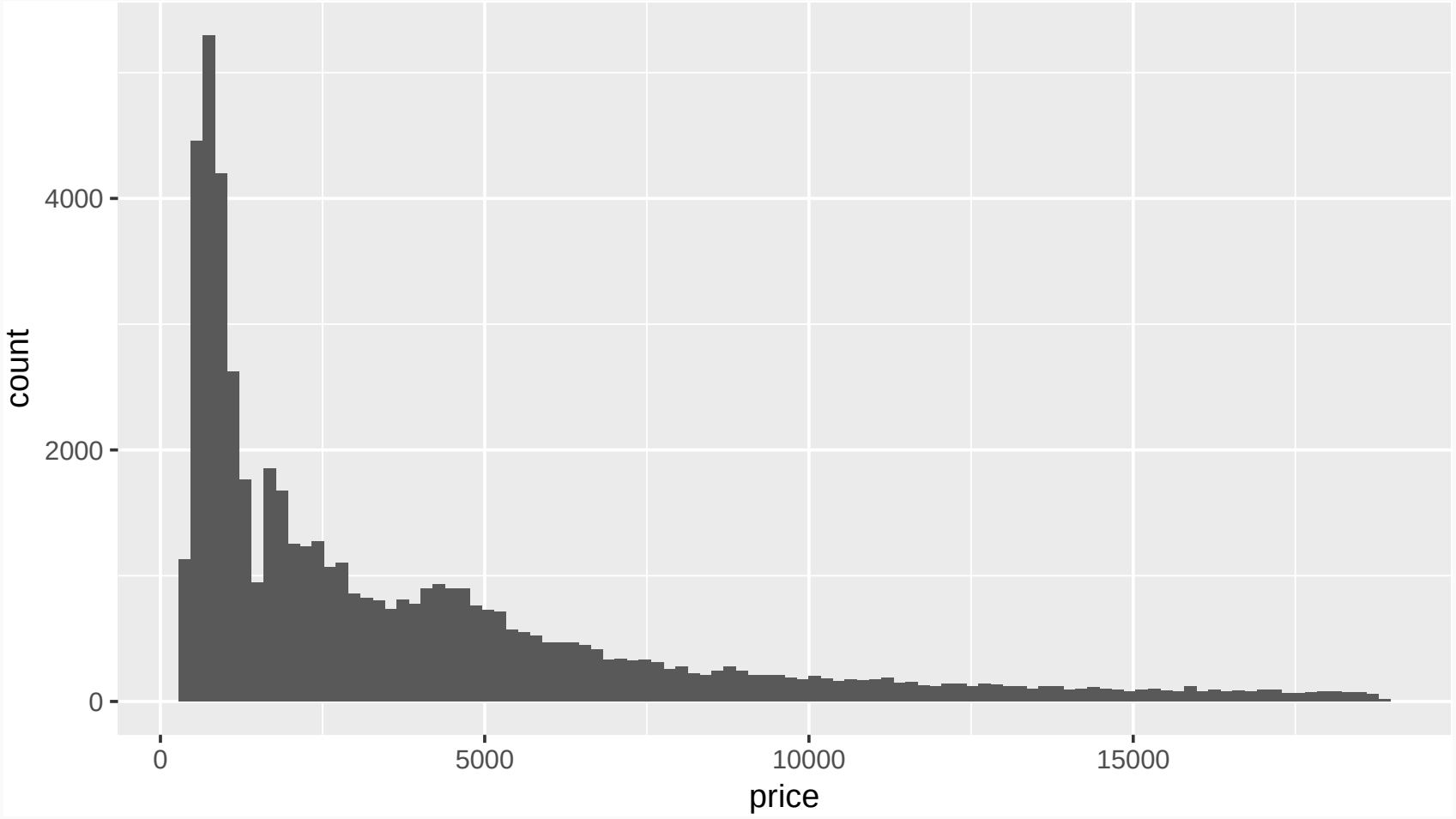
If we don't count the values, but assign a value to the y-axis, we use `geom_col()`

```
df <- data_frame(name = c("Alice", "Bob", "Charlie"),
                 sales = c(102547, 75014, 98541))
ggplot(df, aes(x = name, y = sales, fill = name)) + geom_col()
```



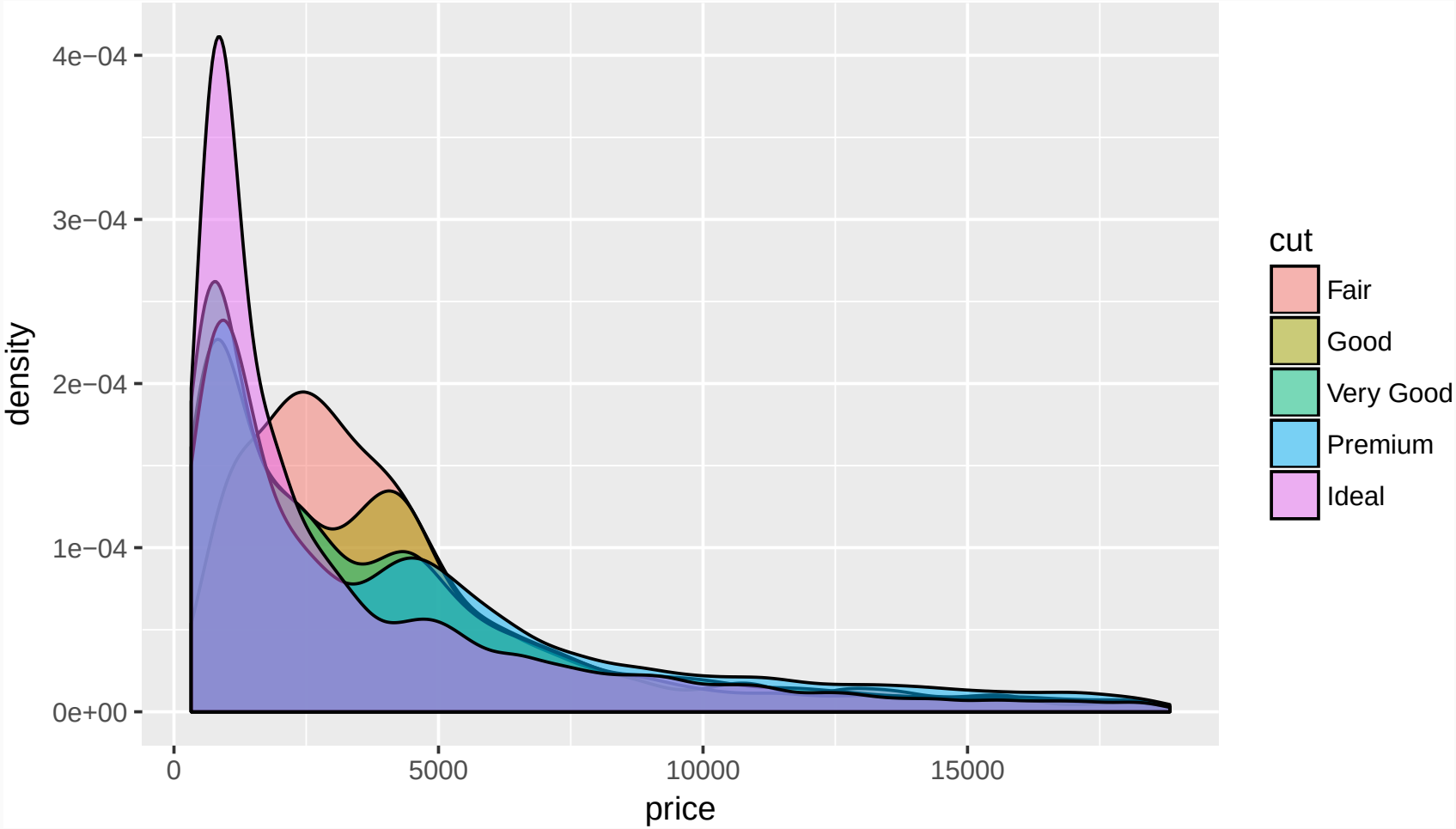
Histogram

```
ggplot(diamonds, aes(x = price)) + geom_histogram(bins = 100)
```



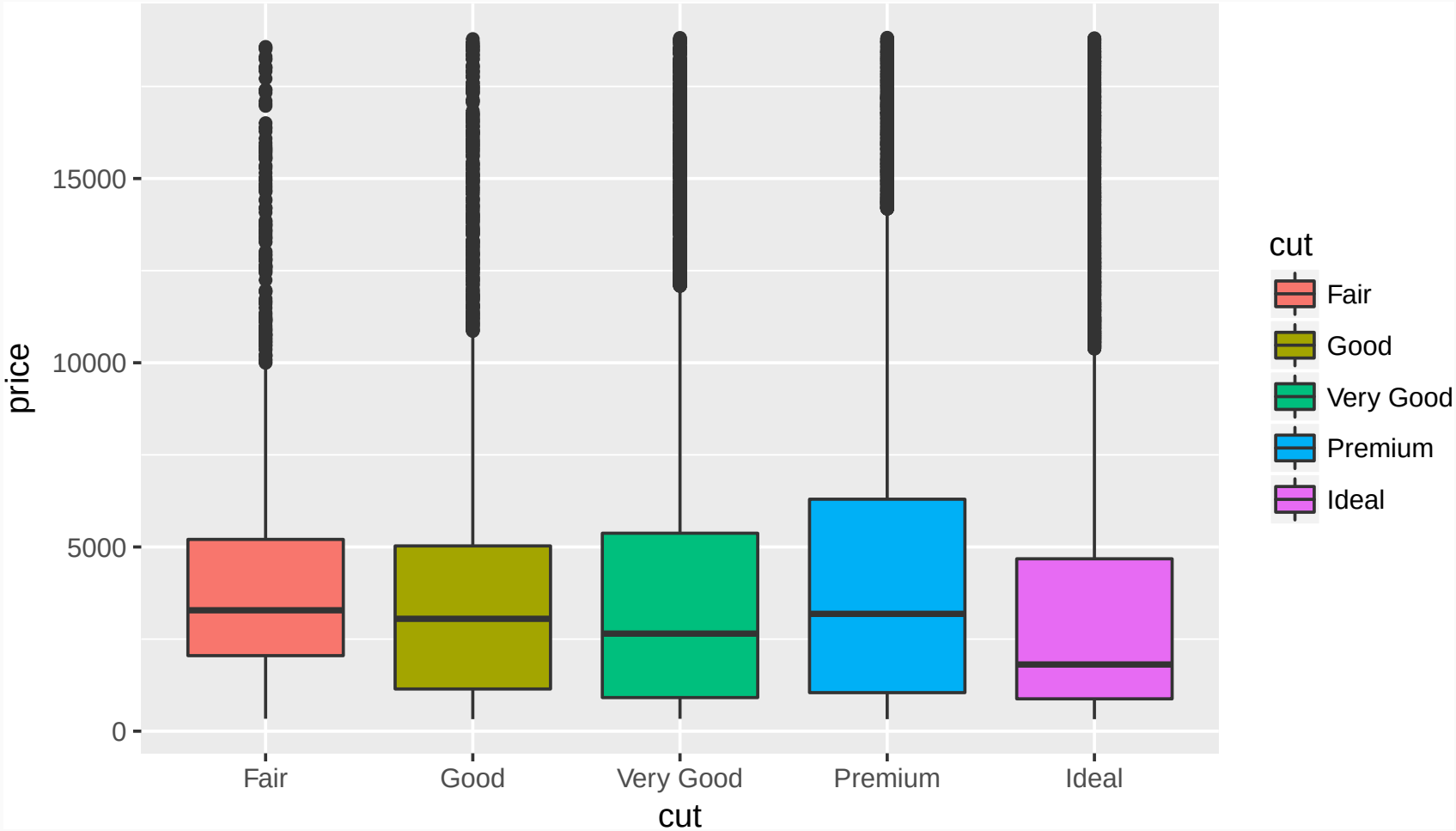
Density Chart

```
ggplot(diamonds, aes(x = price, fill = cut)) +
  geom_density(alpha = 0.5)
```



Boxplot

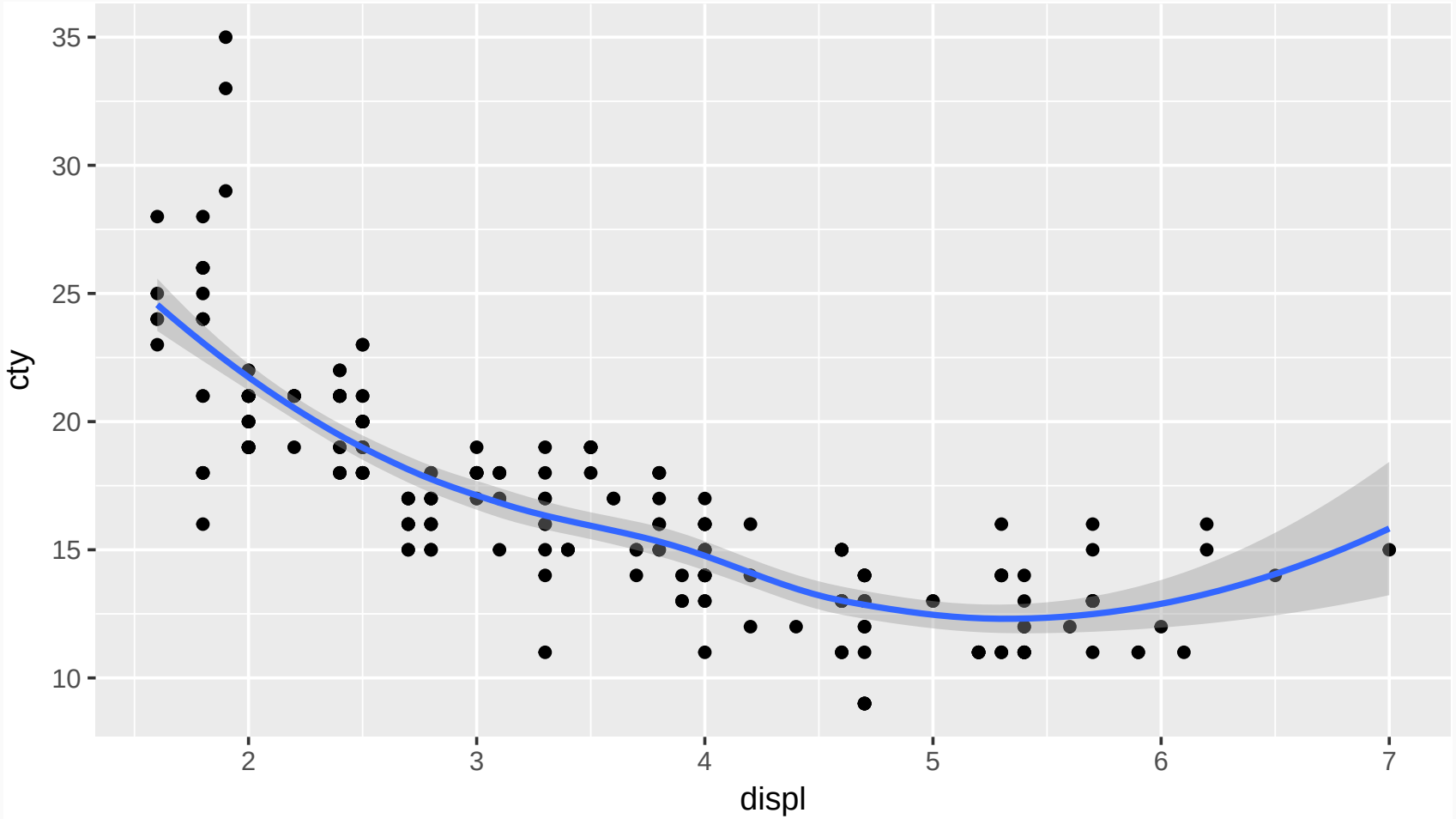
```
ggplot(diamonds, aes(x = cut, y = price, fill = cut)) +
  geom_boxplot()
```



Smoothing

Fitting a loess-function to the points

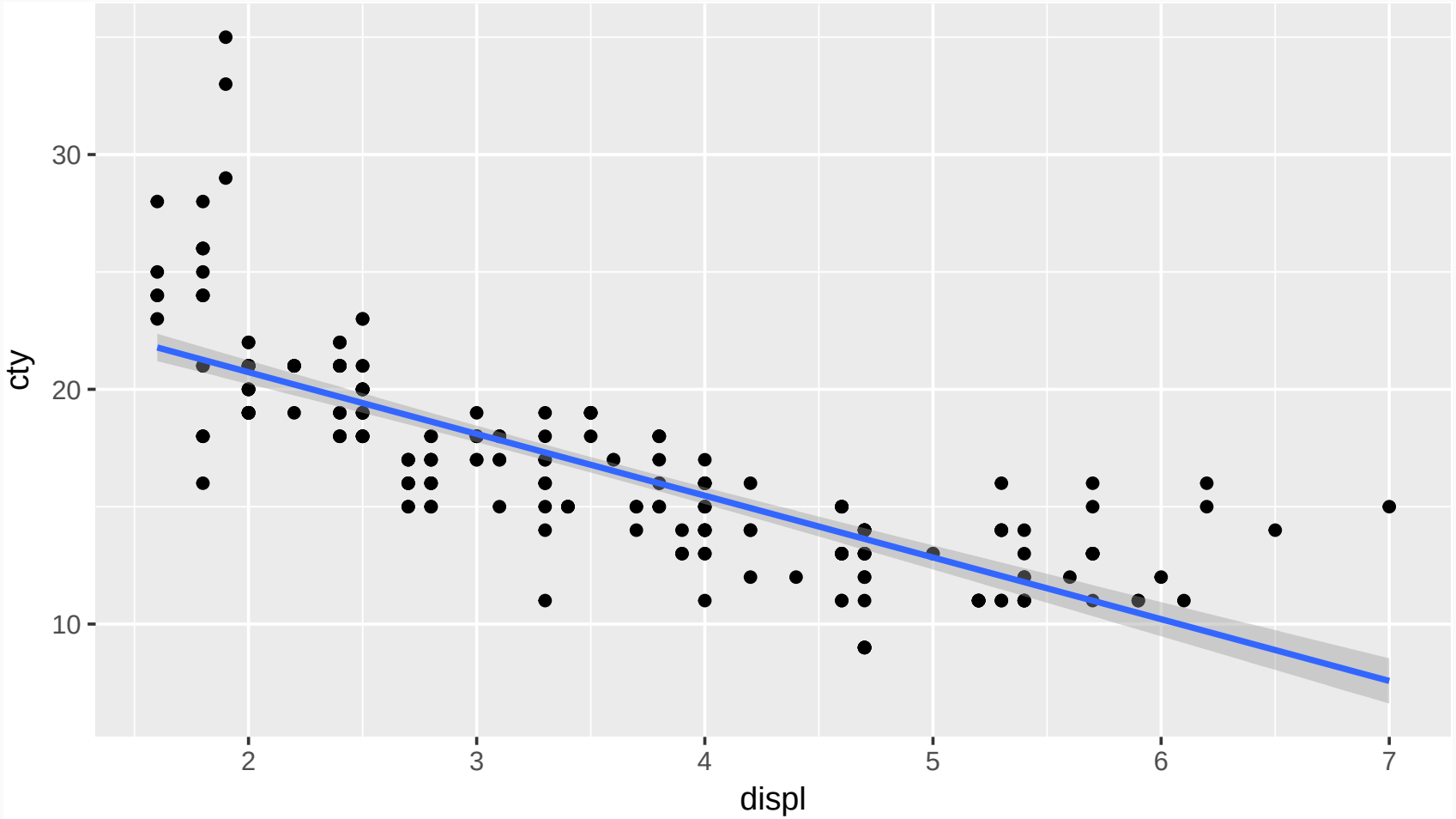
```
ggplot(mpg, aes(x = displ, y = cty)) + geom_point() +
  geom_smooth()
```



Smoothing cont'd

Fitting an OLS (linear model) to the points

```
ggplot(mpg, aes(x = displ, y = cty)) + geom_point() +
  geom_smooth(method = "lm")
```



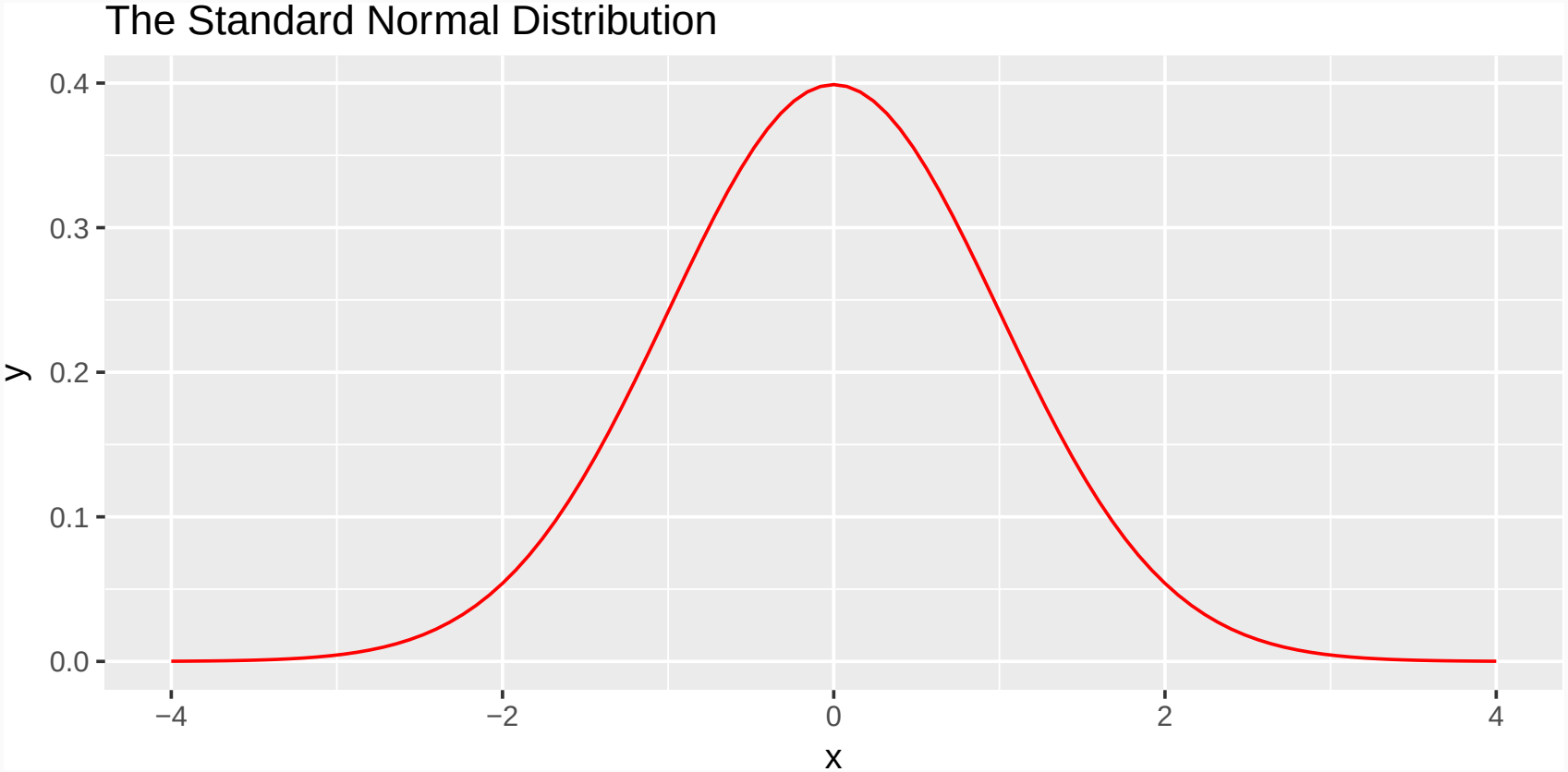
Exercises: Geometrics

Functions

Plotting Functions

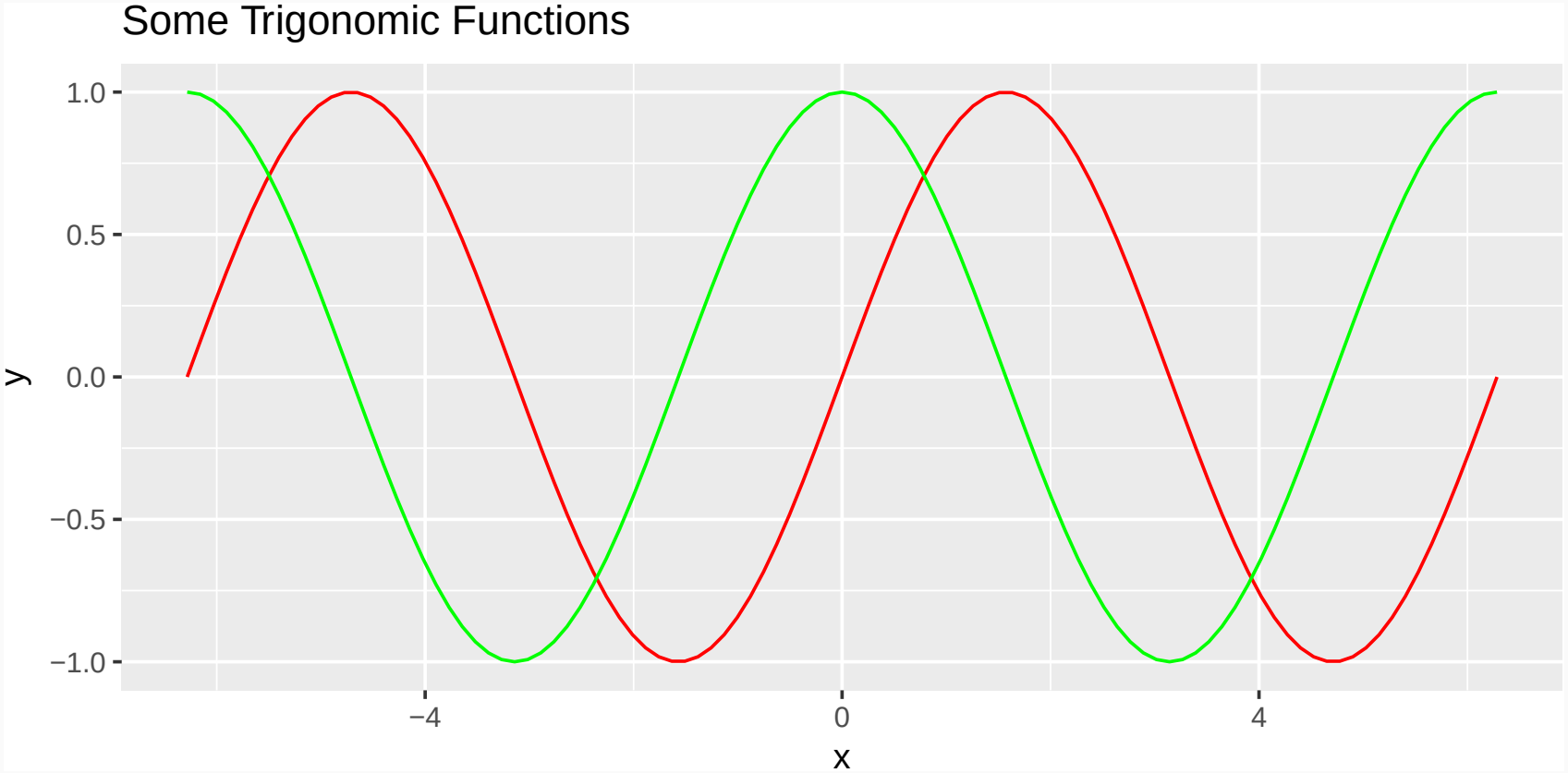
Using `stat_function`.

```
ggplot(data_frame(x = c(-4, 4)), aes(x = x)) +
  stat_function(fun = dnorm, color = "red") + #?dnorm
  labs(title = "The Standard Normal Distribution")
```



Plotting Functions

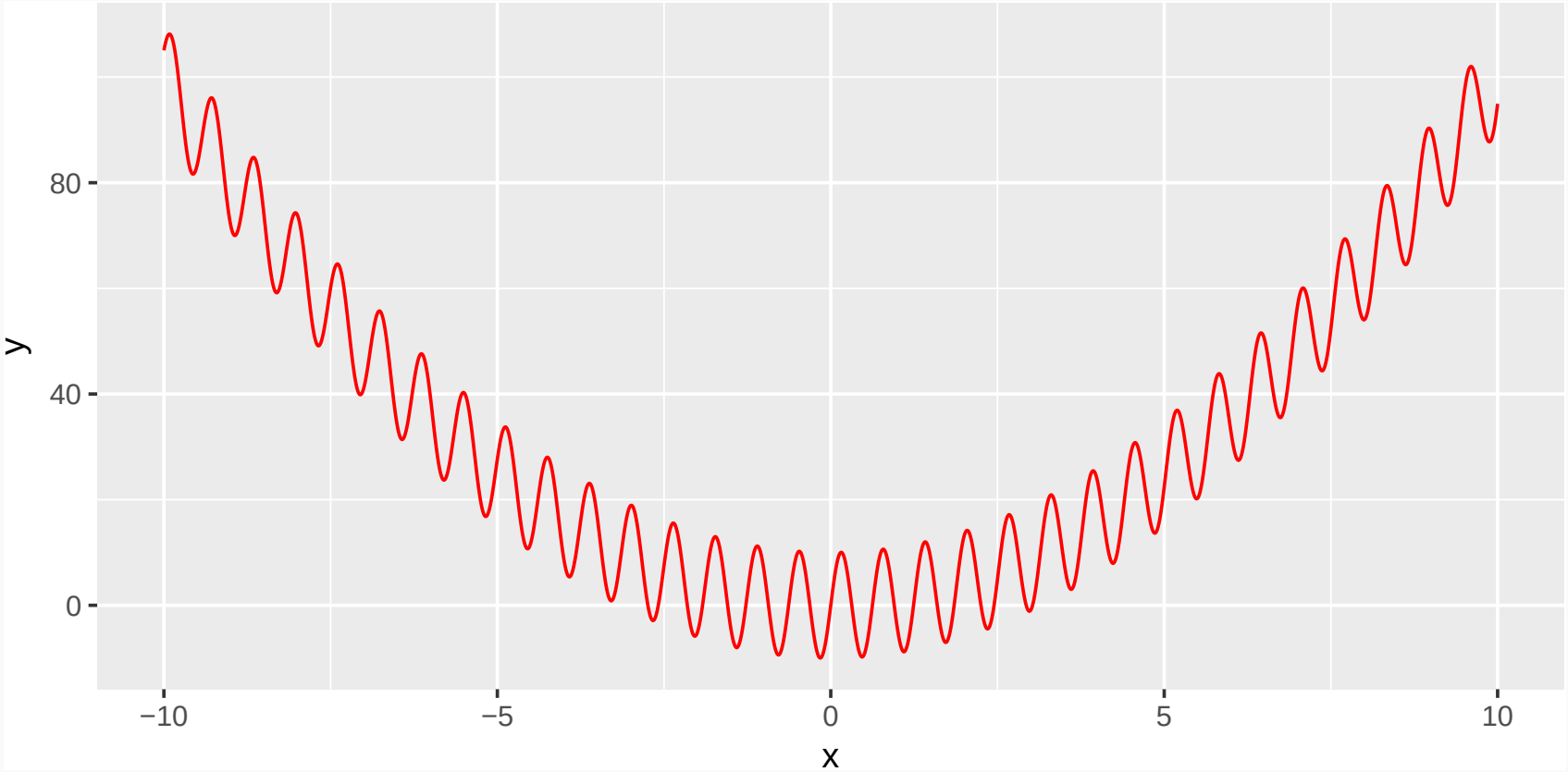
```
ggplot(data_frame(x = c(-2*pi, 2*pi)), aes(x = x)) +
  stat_function(fun = sin, color = "red") +
  stat_function(fun = cos, color = "green") +
  labs(title = "Some Trigonomic Functions")
```



Writing your own Functions

```
myfun <- function(x) x^2 + sin(x*10) * 10

ggplot(data_frame(x = c(-10, 10)), aes(x = x)) +
  stat_function(fun = myfun, color = "red", n = 10000)
```



Exercises: Functions

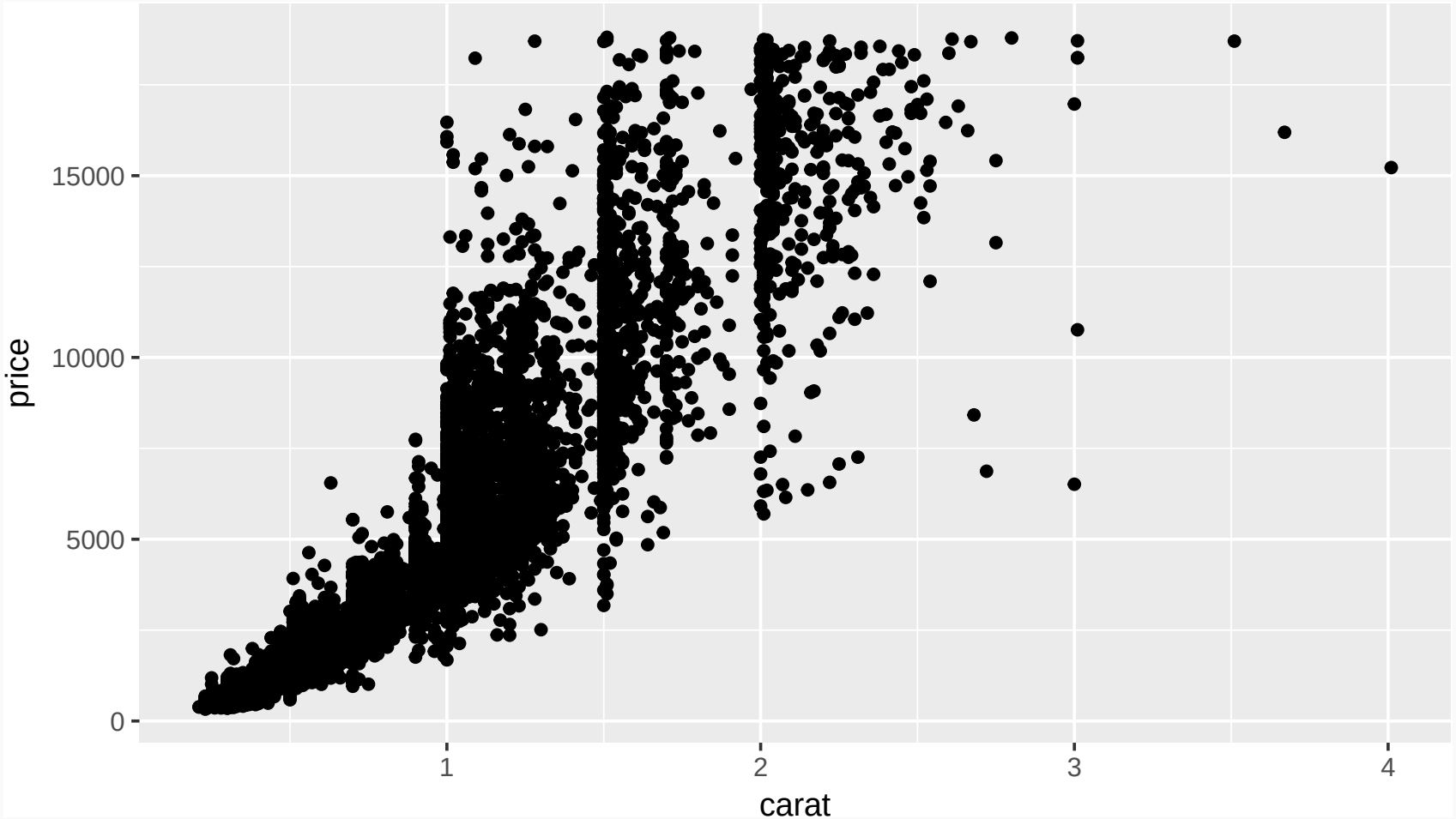
Scales

Possible Scales

* for the x-axis and the y-axis (no z-axis possible)
  - `scale_*_reverse` (i.e., `scale_x_reverse` and `scale_y_reverse`)
  - `scale_*_sqrt`
  - `scale_*_log10`
  - `scale_*_date`

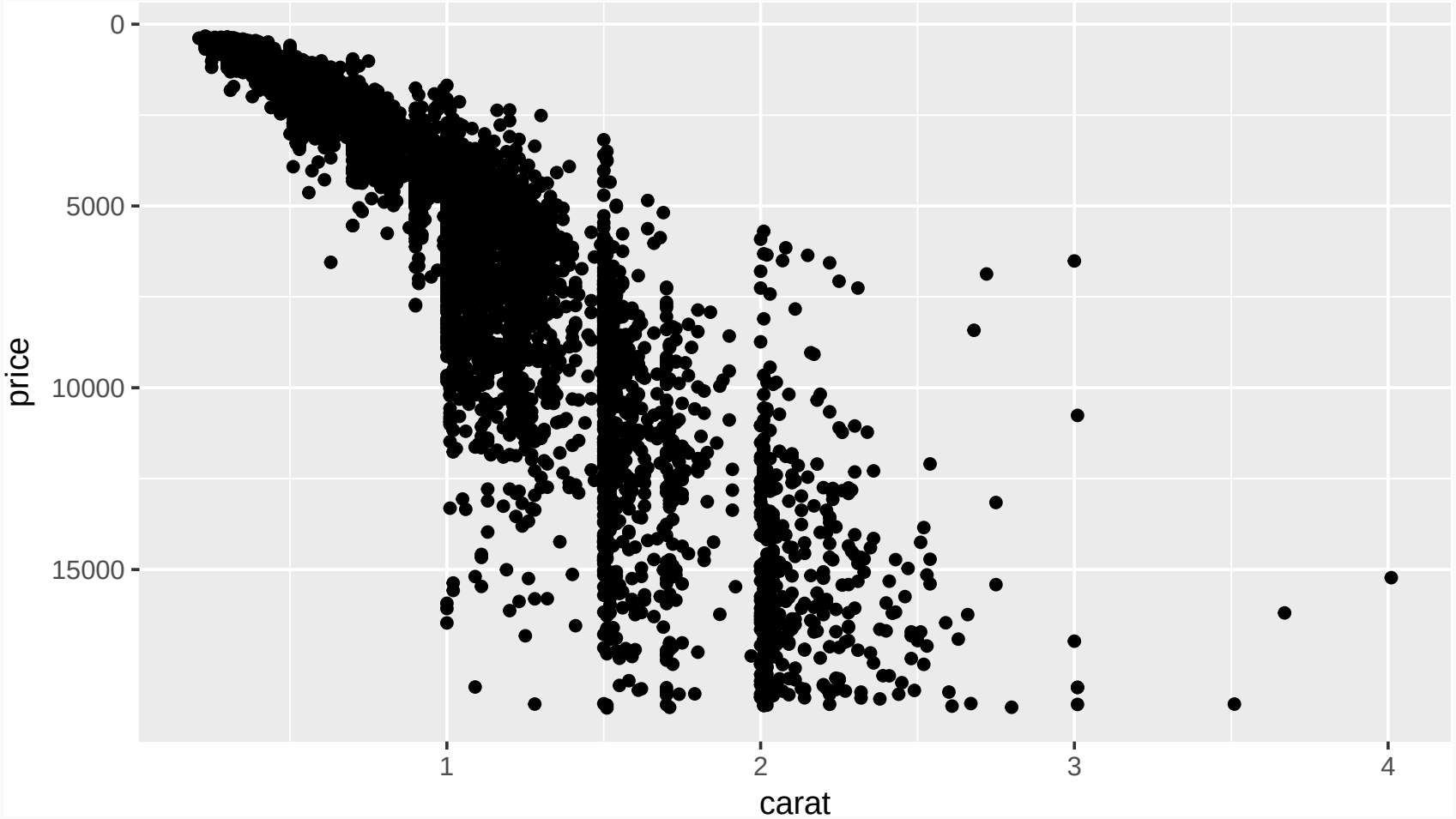
A Base Plot

```
df <- diamonds %>% sample_n(10000)
plot1 <- ggplot(df, aes(x = carat, y = price)) + geom_point()
plot1
```



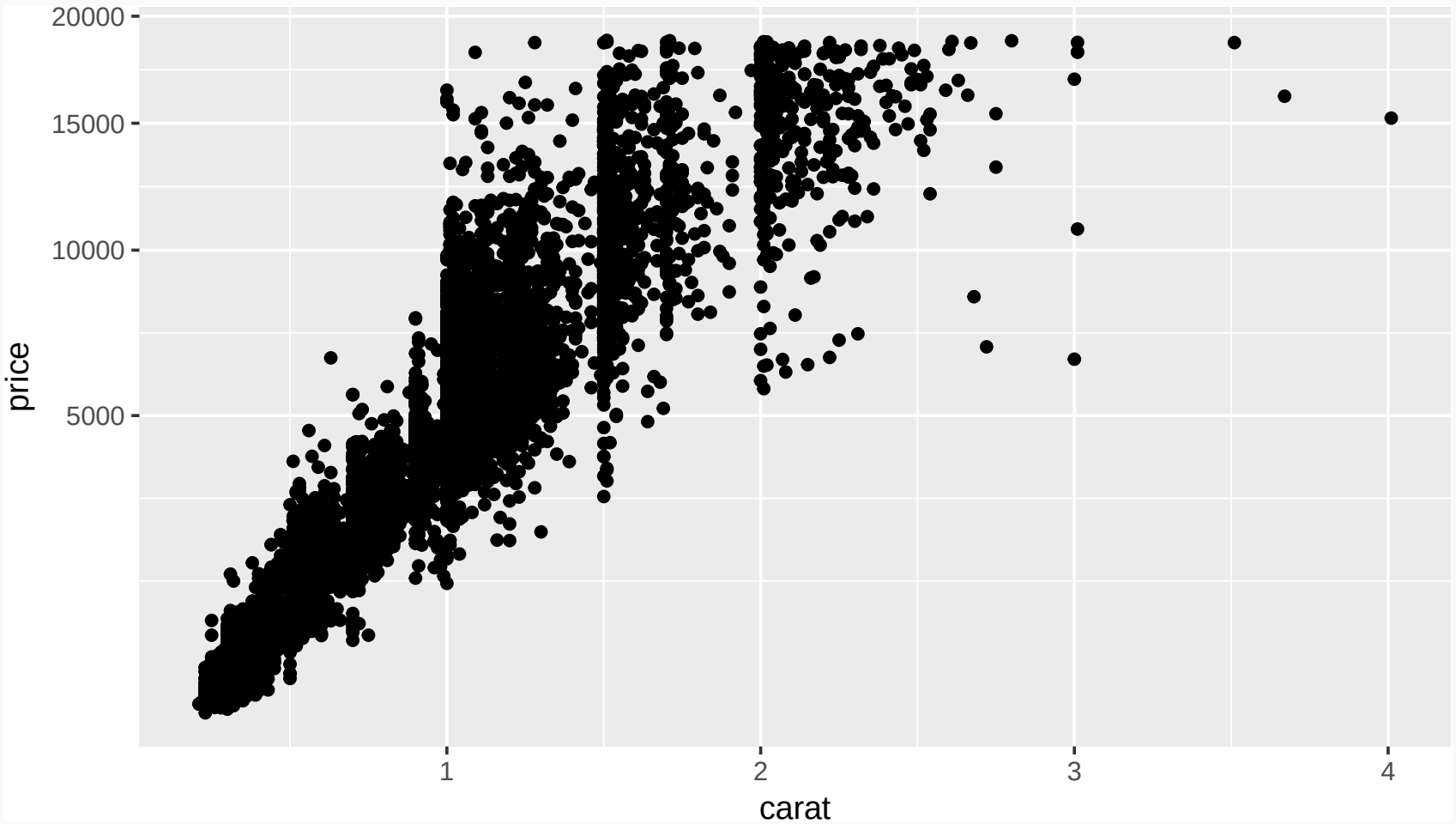
Scale Reversion

```
plot1 + scale_y_reverse()
```



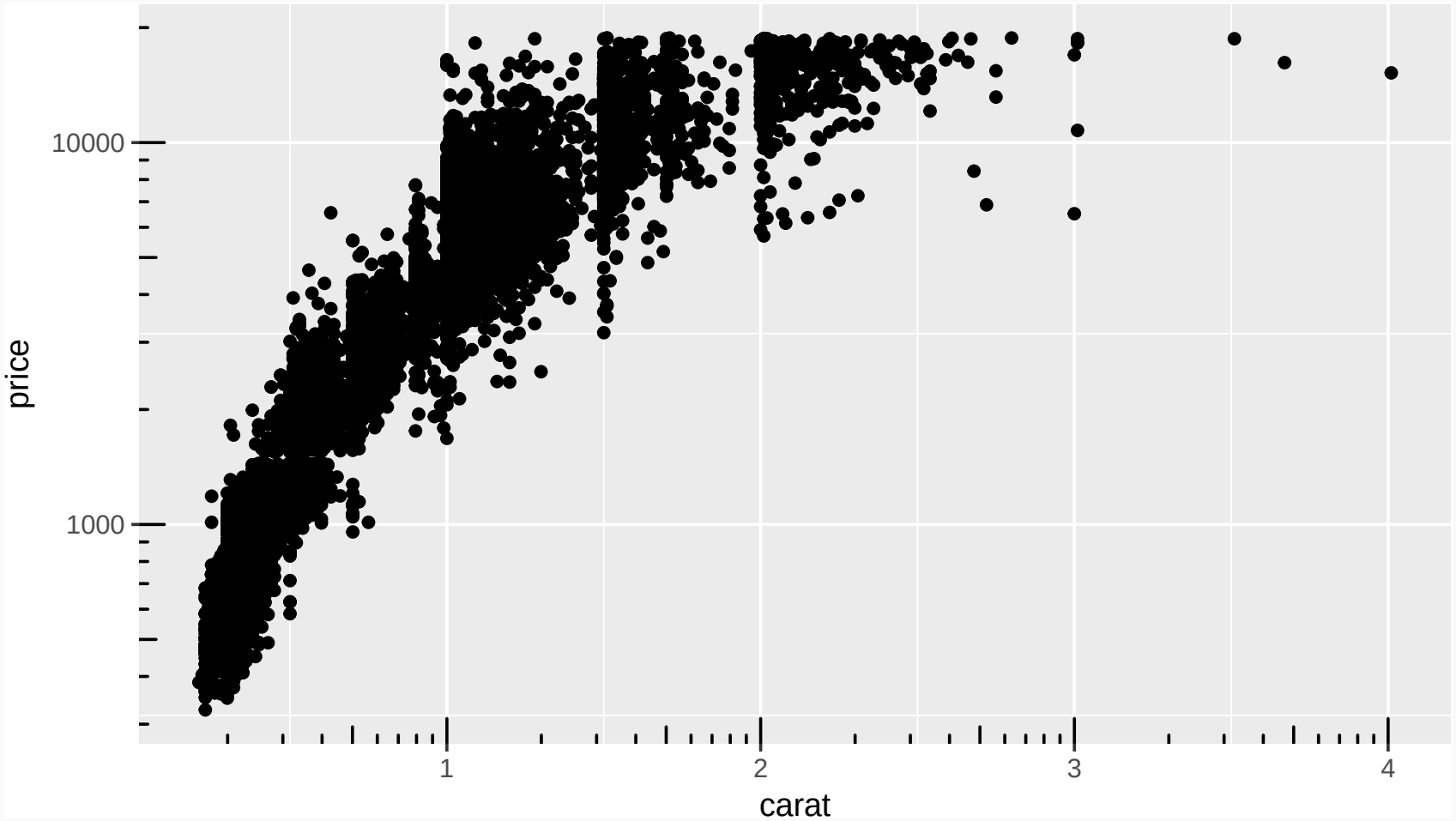
Square-Root Scale

```
plot1 + scale_y_sqrt()
```



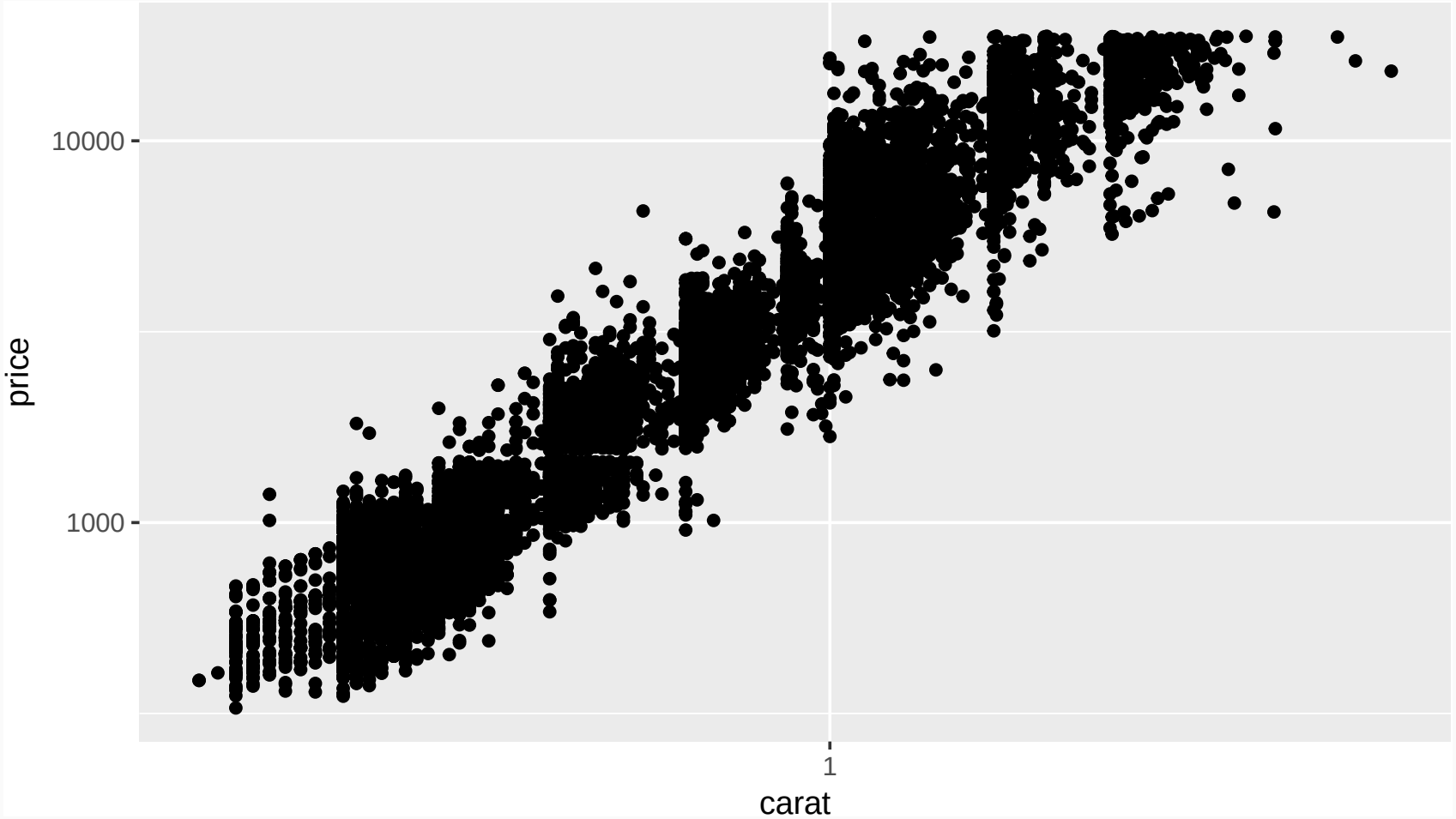
Logarithmic Scale

```
plot1 + scale_y_log10() + annotation_logticks()
```



Double Logarithmic Scale

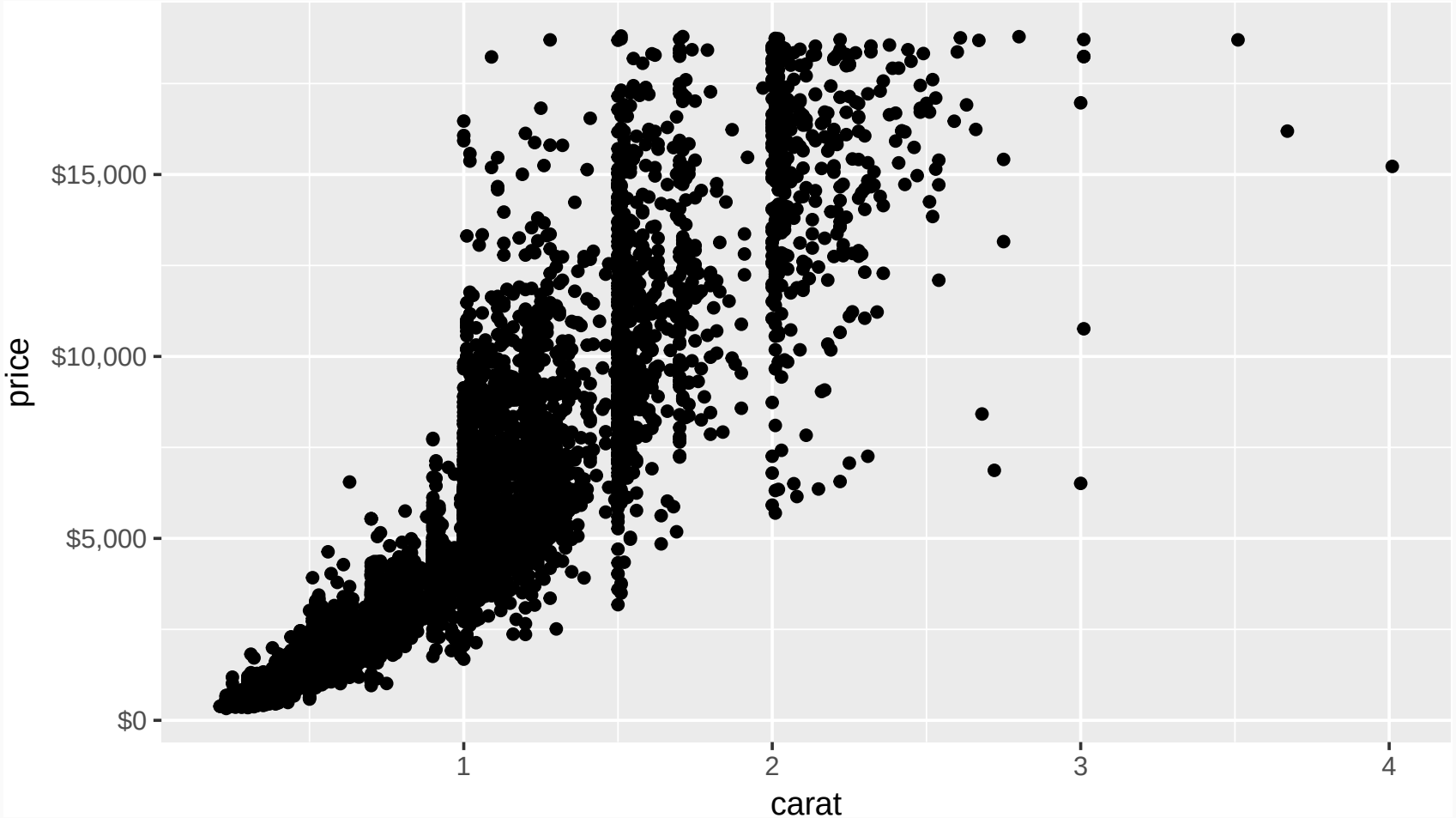
```
plot1 + scale_y_log10() + scale_x_log10()
```



Scale Formatting

Use: `library(scales)` for scale formatting

```
library(scales)
plot1 + scale_y_continuous(labels = dollar)
```



Scale Formatting cont'd

Library `scales` also includes:

- `comma`: 15,123.02
- `dollar`: $15,123.02
- `percent`: 12.45%
- `scientific`: 1.5e+04

Colors

Color Basics

- Hardcoded colors
  - Named colors: `red`, `green`, `blue`, ... `?colors`
  - HEX colors: `#ff0000`, `#00ff00`, `#0000ff`, ... -> Google...
- Using a library
  - `RColorBrewer`: sequential, diverging, and qualitative but limited colors
  - `viridis`: continuous with unlimited colors

ggplot changing Colors

- scale_*_manual
- scale_*_discrete
- scale_*_continuous

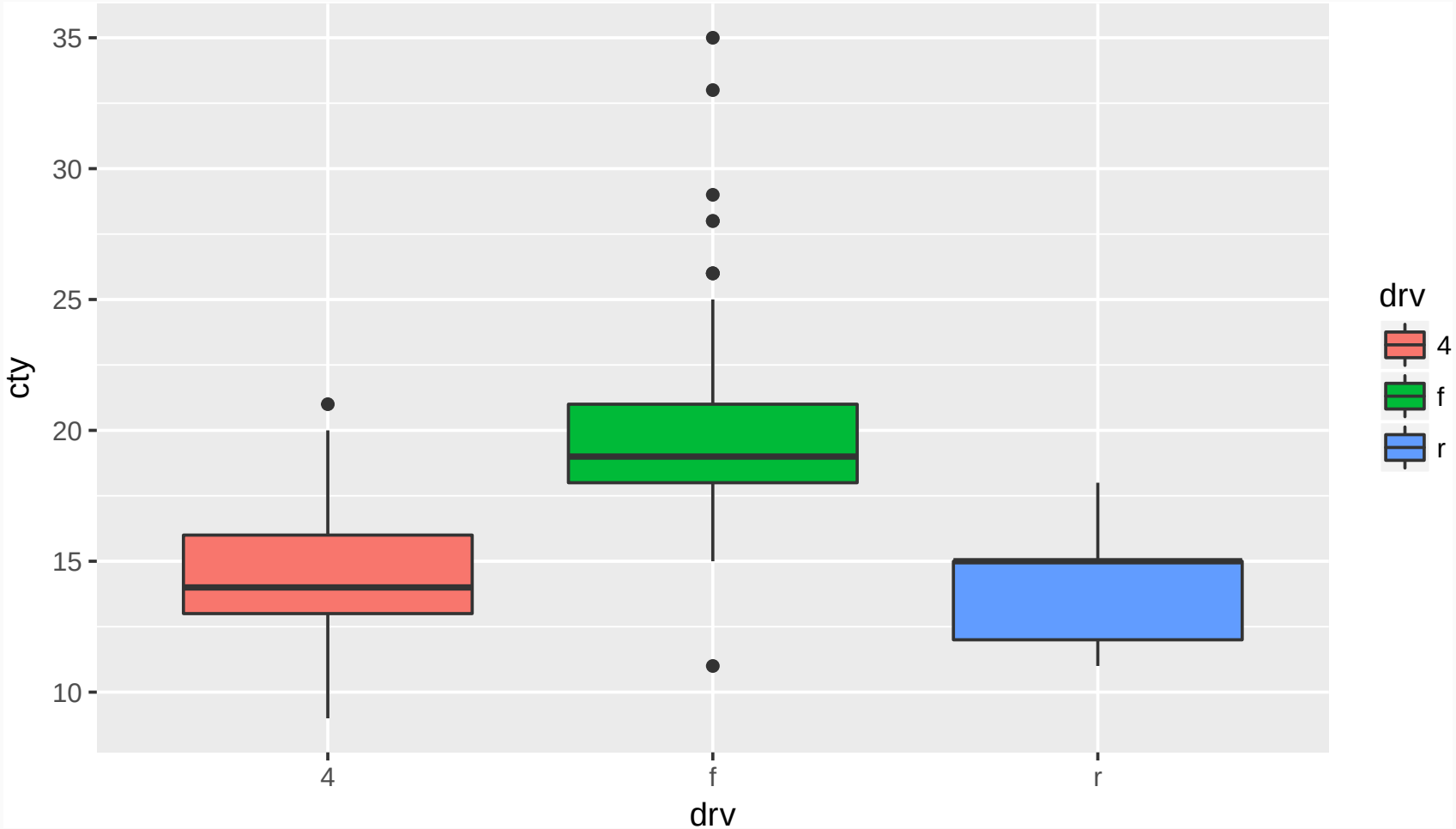
Important difference!

fill vs. color

```
ggplot(..., aes(..., fill = ...)) +
  scale_fill_manual(...)
# vs.
ggplot(..., aes(..., color = ...)) +
  scale_color_manual(...)
```

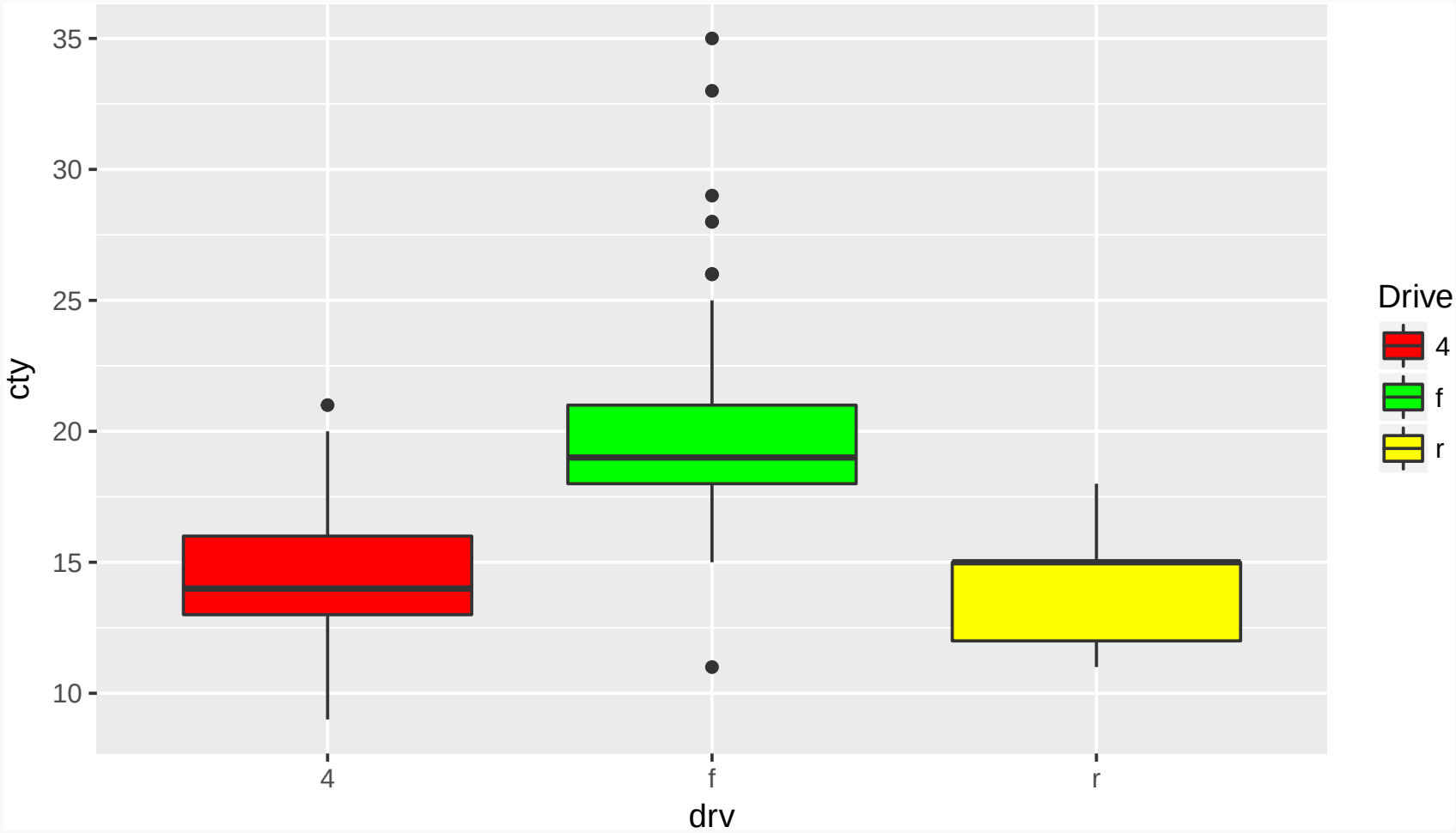
Colors

```
mpg_plot <- ggplot(mpg, aes(x = drv, y = cty, fill = drv)) +
  geom_boxplot()
mpg_plot
```



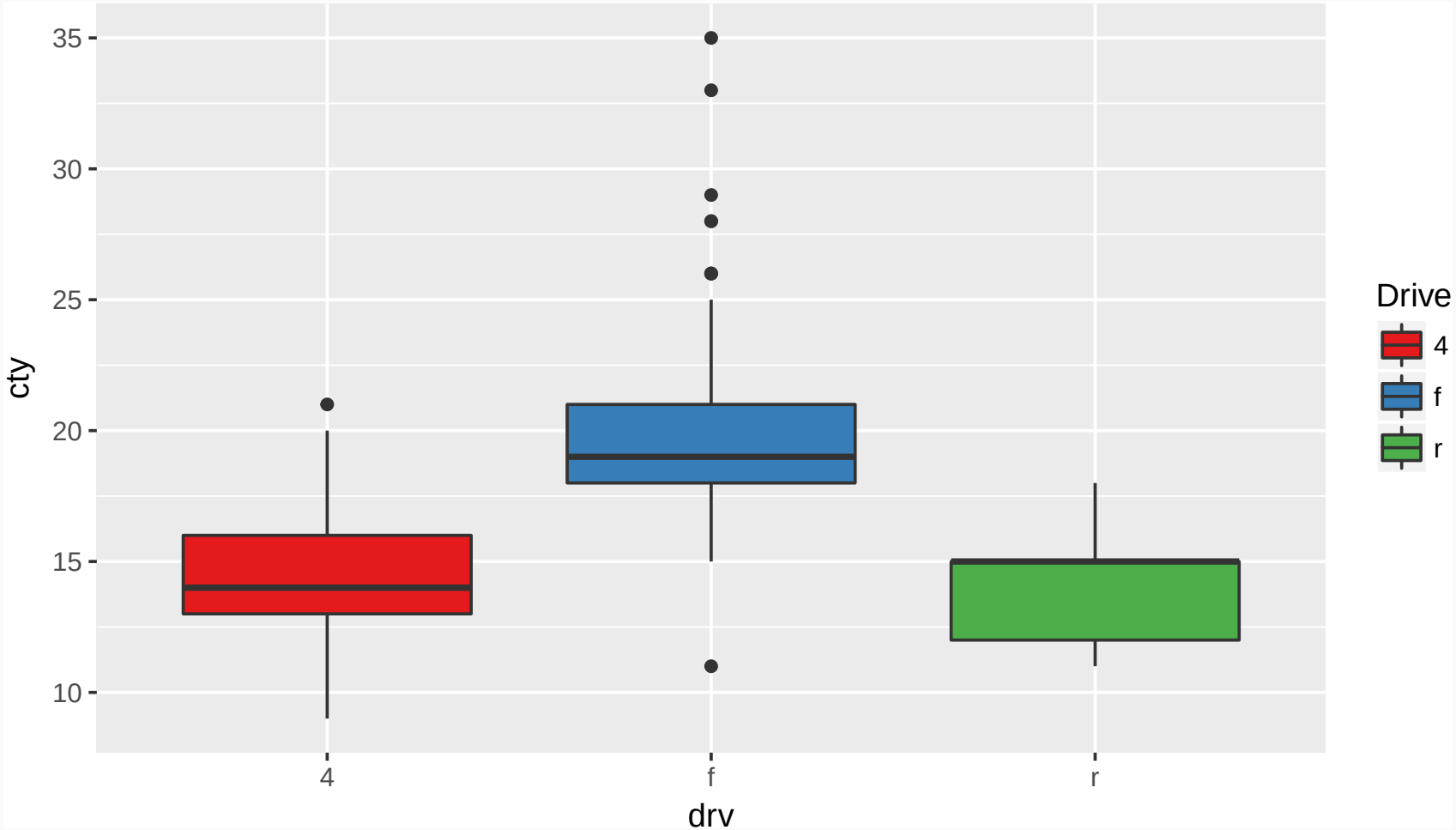
Hardcoded Named Colors

```
mpg_plot +
  scale_fill_manual(name = "Drive",
                    values = c("red", "green", "yellow"))
```



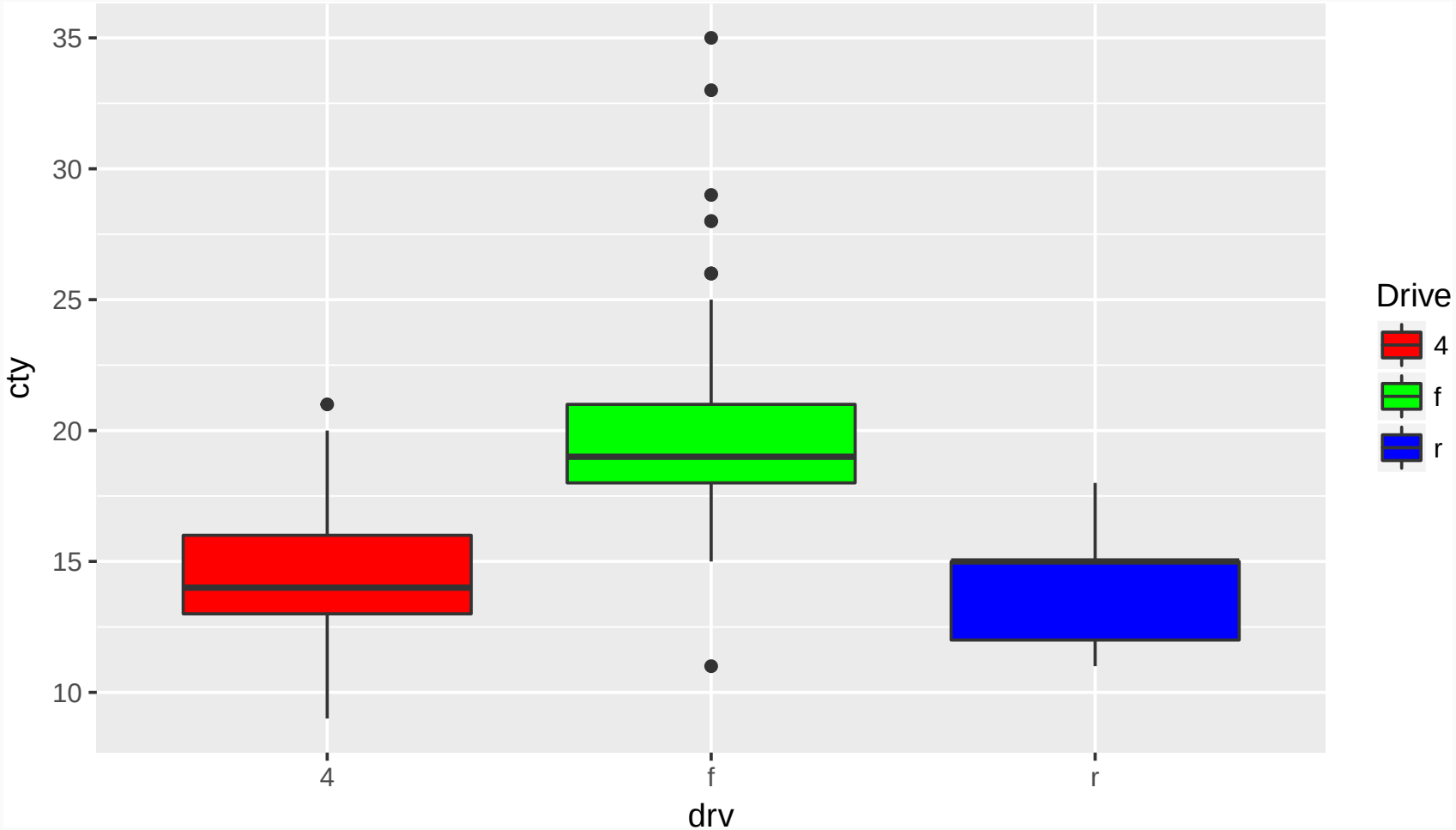
Hardcoded HEX Colors cont'd

```
mpg_plot +
  scale_fill_manual(name = "Drive",
                    values = c("#e41a1c", "#377eb8", "#4daf4a"))
```



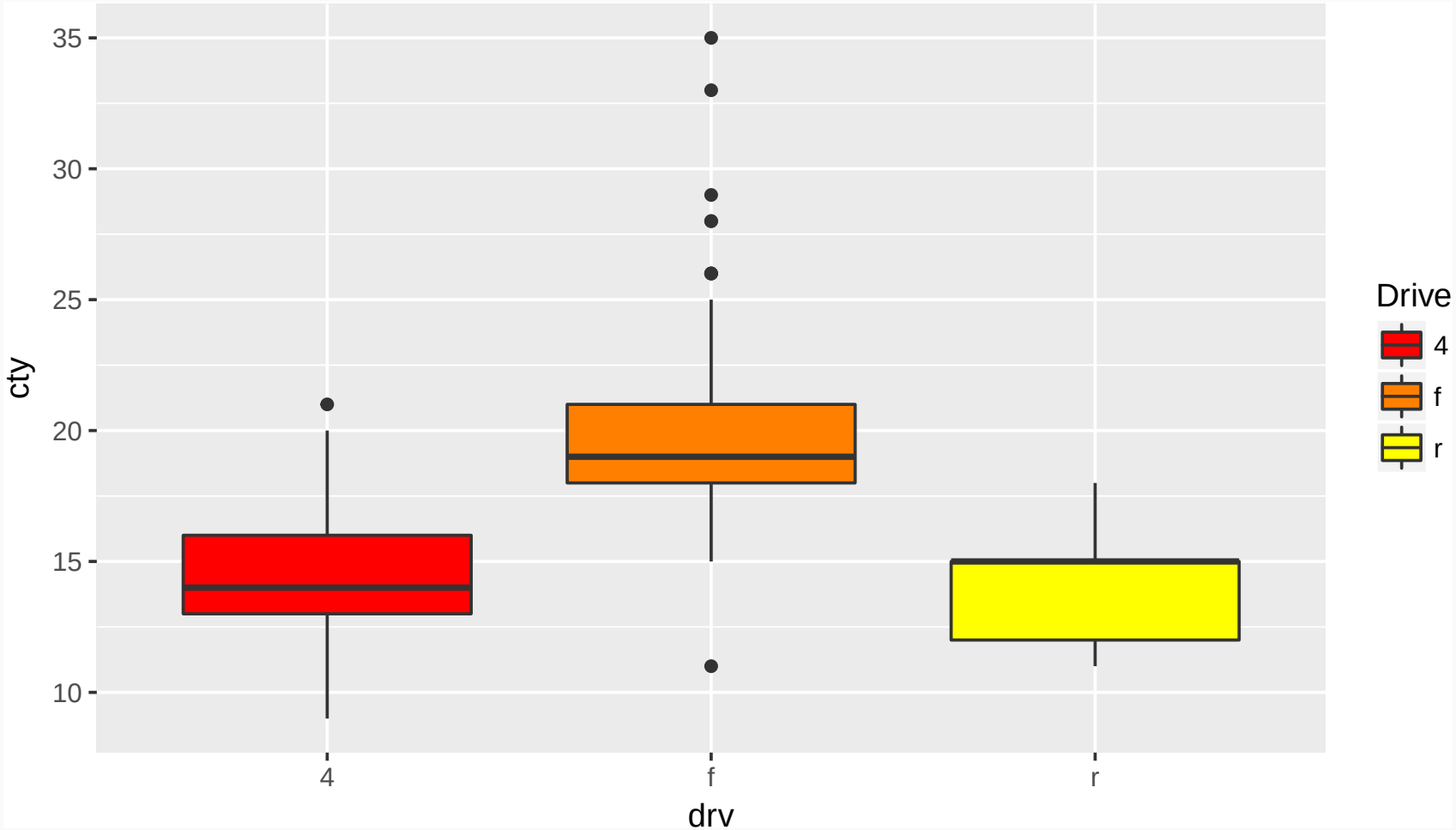
ggplot Colors

```
mpg_plot +
  scale_fill_manual(name = "Drive",
                    values = rainbow(3))
```



ggplot Colors

```
mpg_plot +
  scale_fill_manual(name = "Drive",
                    values = heat.colors(3))
```



RColorBrewer

ColorBrewer: http://colorbrewer2.org/

sequential, diverging, & qualitative

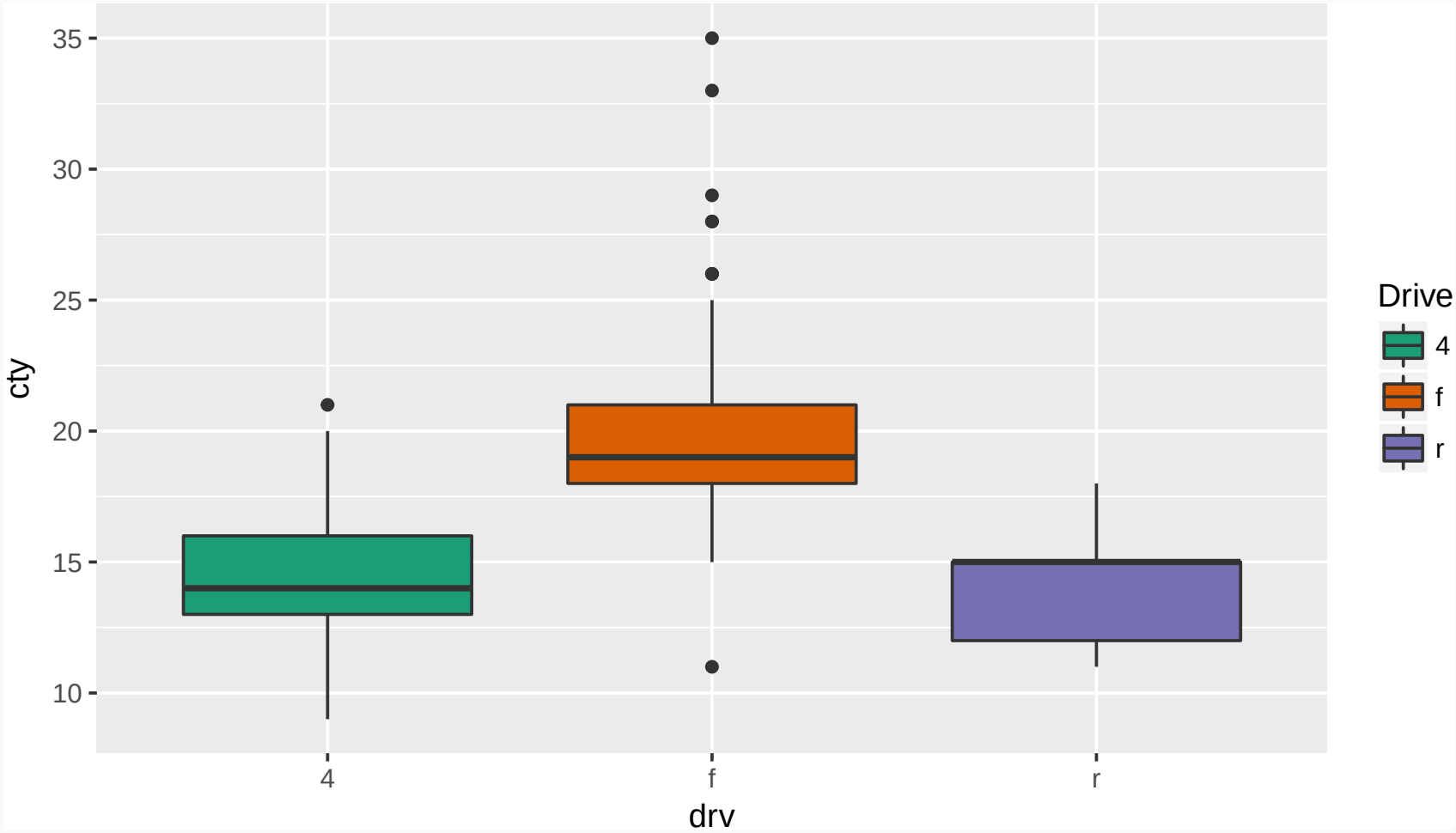
List all possible palettes `display.brewer.all()`

```
library(RColorBrewer)
brewer.pal(3, "Dark2")
```

```
## [1] "#1B9E77" "#D95F02" "#7570B3"
```

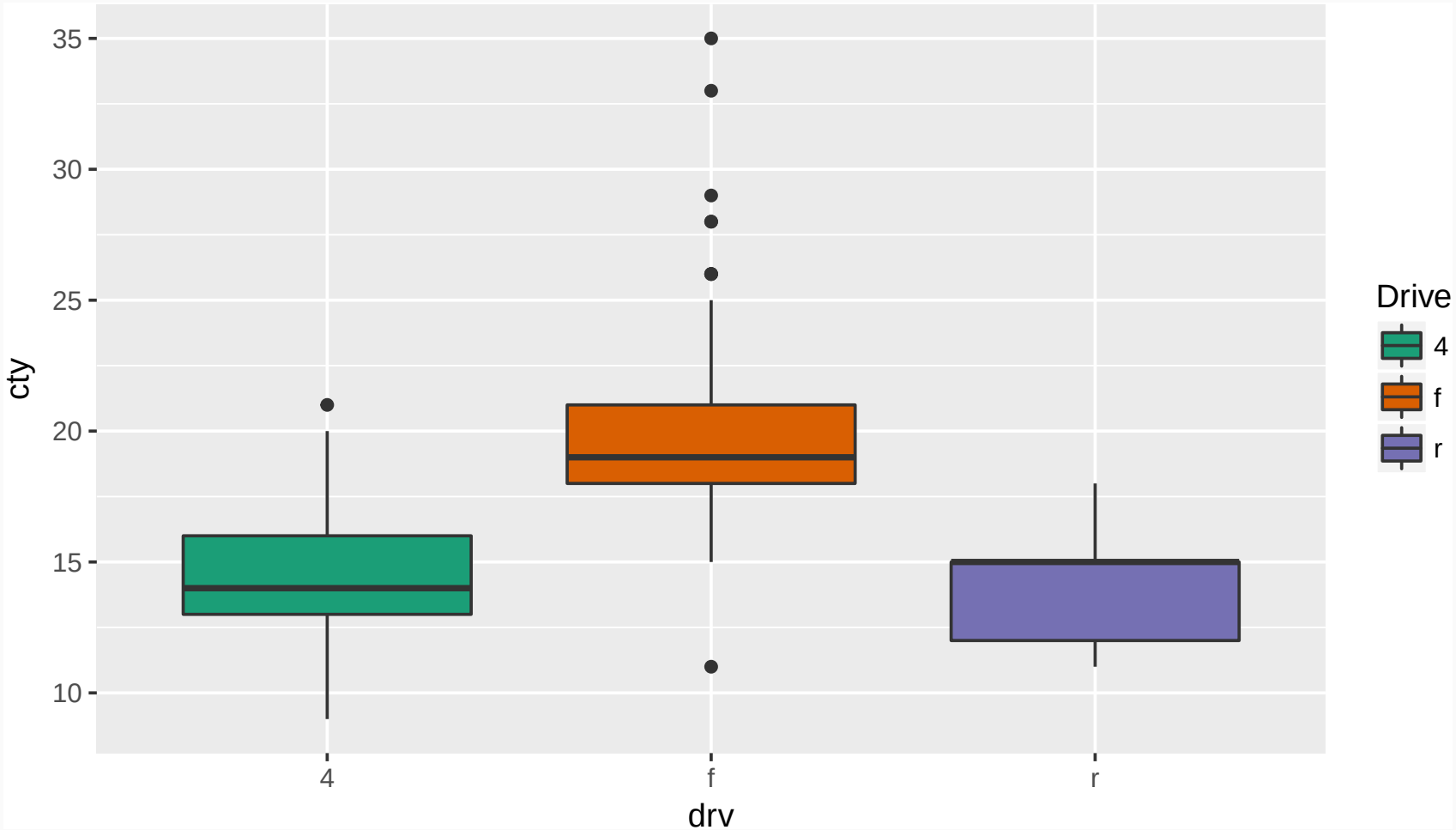
Using RColorBrewer

```
mpg_plot +
  scale_fill_manual(name = "Drive",
                    values = brewer.pal(3, "Dark2"))
```



Using RColorBrewer

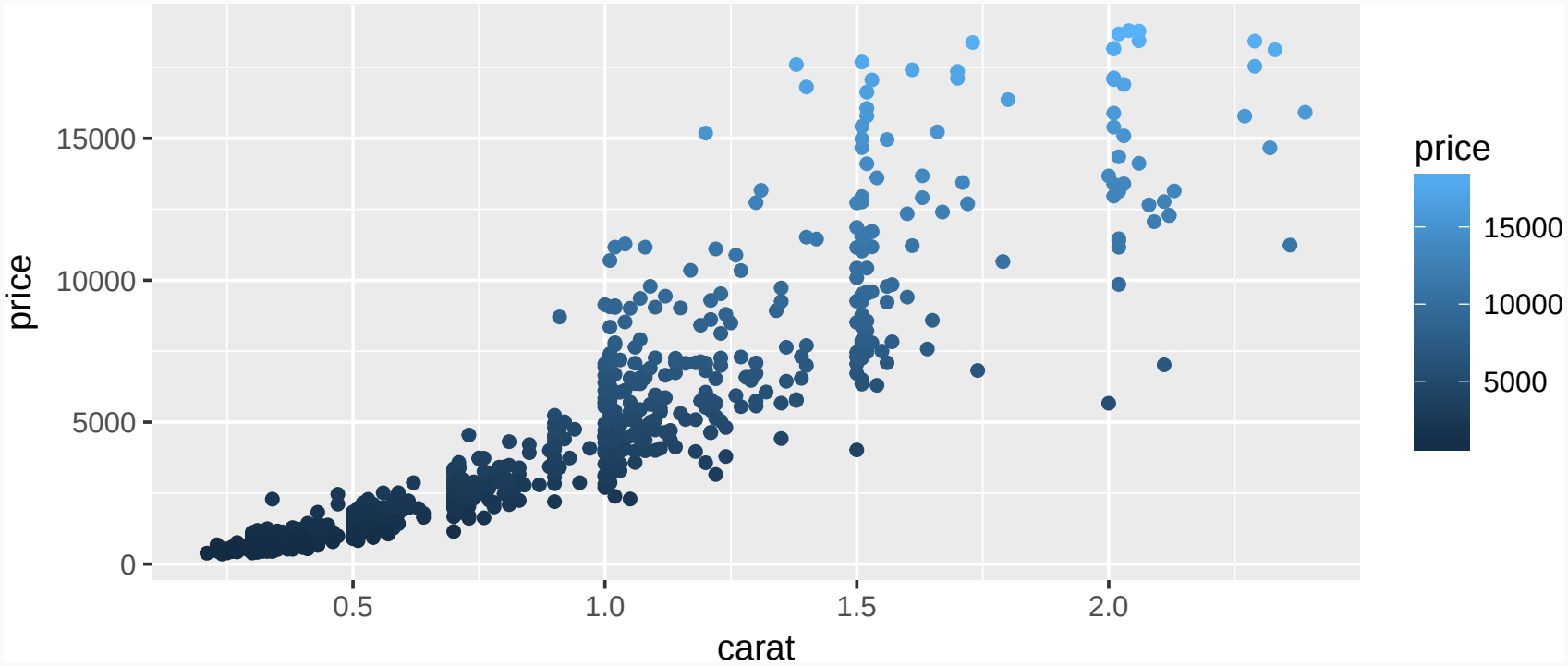
```
mpg_plot + scale_fill_brewer(name = "Drive", palette = "Dark2")
```



ggplot Contiuous Colors

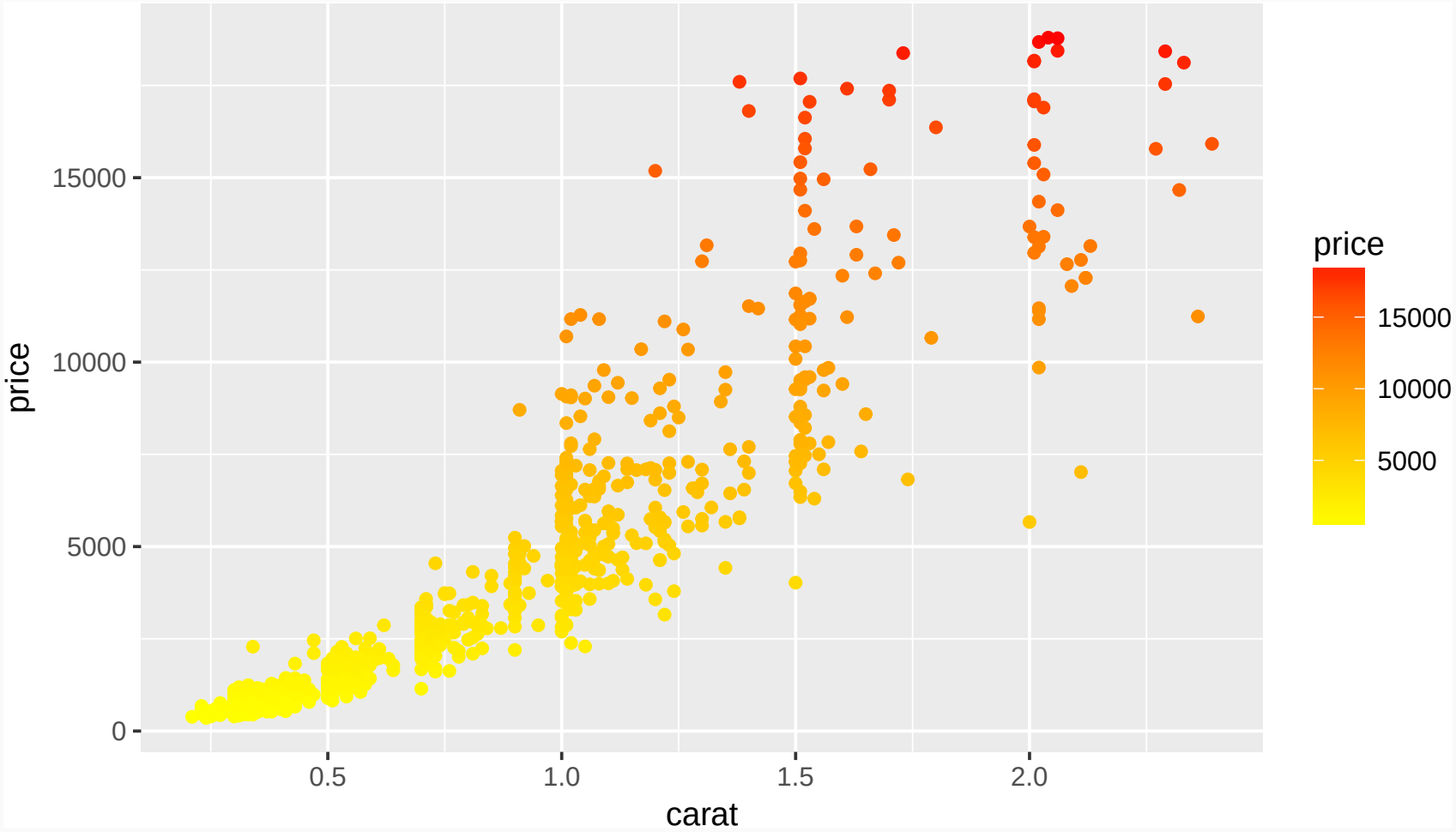
```
diamond_plot <-  ggplot(sample_n(diamonds, 1000),
                        aes(x = carat, y = price,
                            color = price)) +
  geom_point()

diamond_plot
```



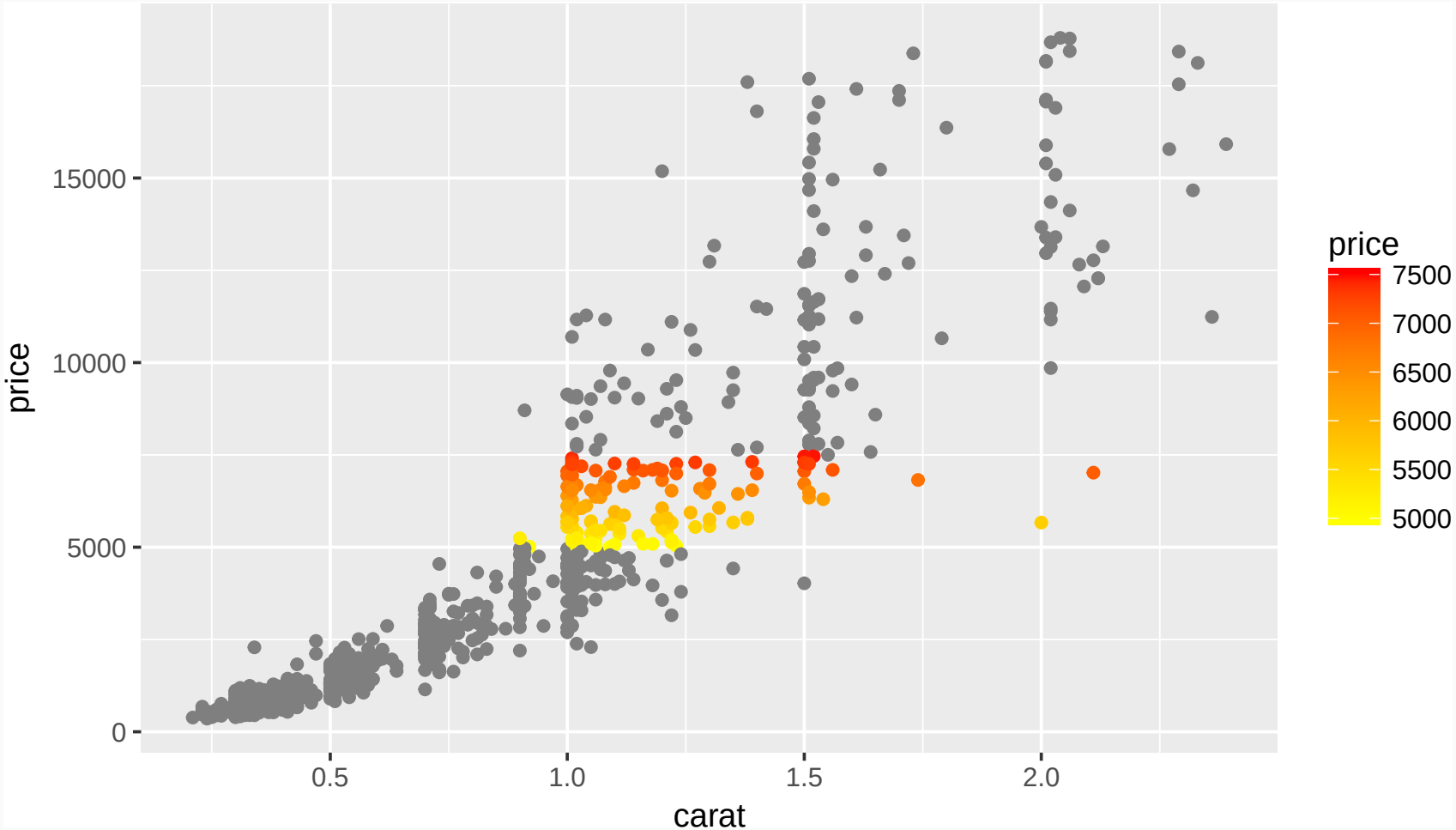
ggplot Continuous Colors

```
diamond_plot +
  scale_color_gradient(low = "yellow", high = "red")
```



ggplot Continuous Colors

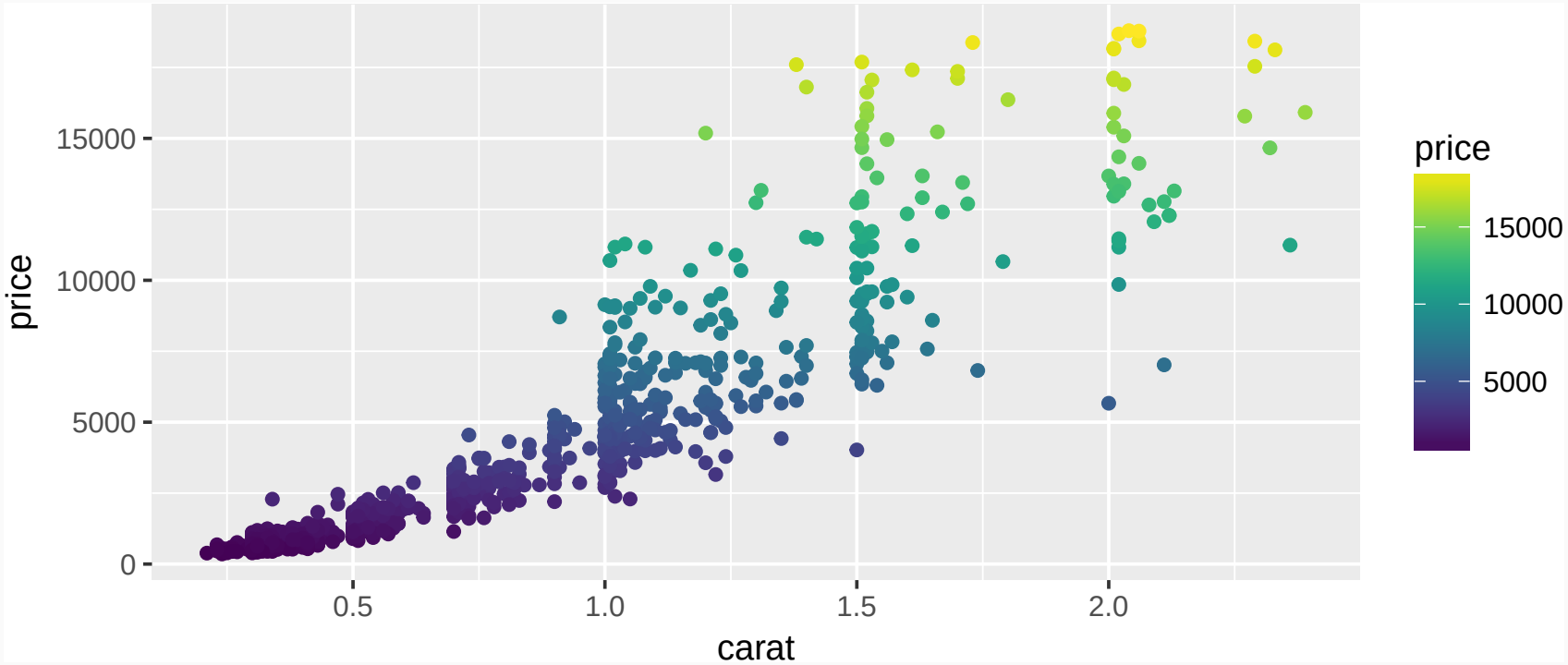
```
diamond_plot +
  scale_color_gradient(low = "yellow", high = "red",
                       limits = c(5000, 7500))
```



Viridis

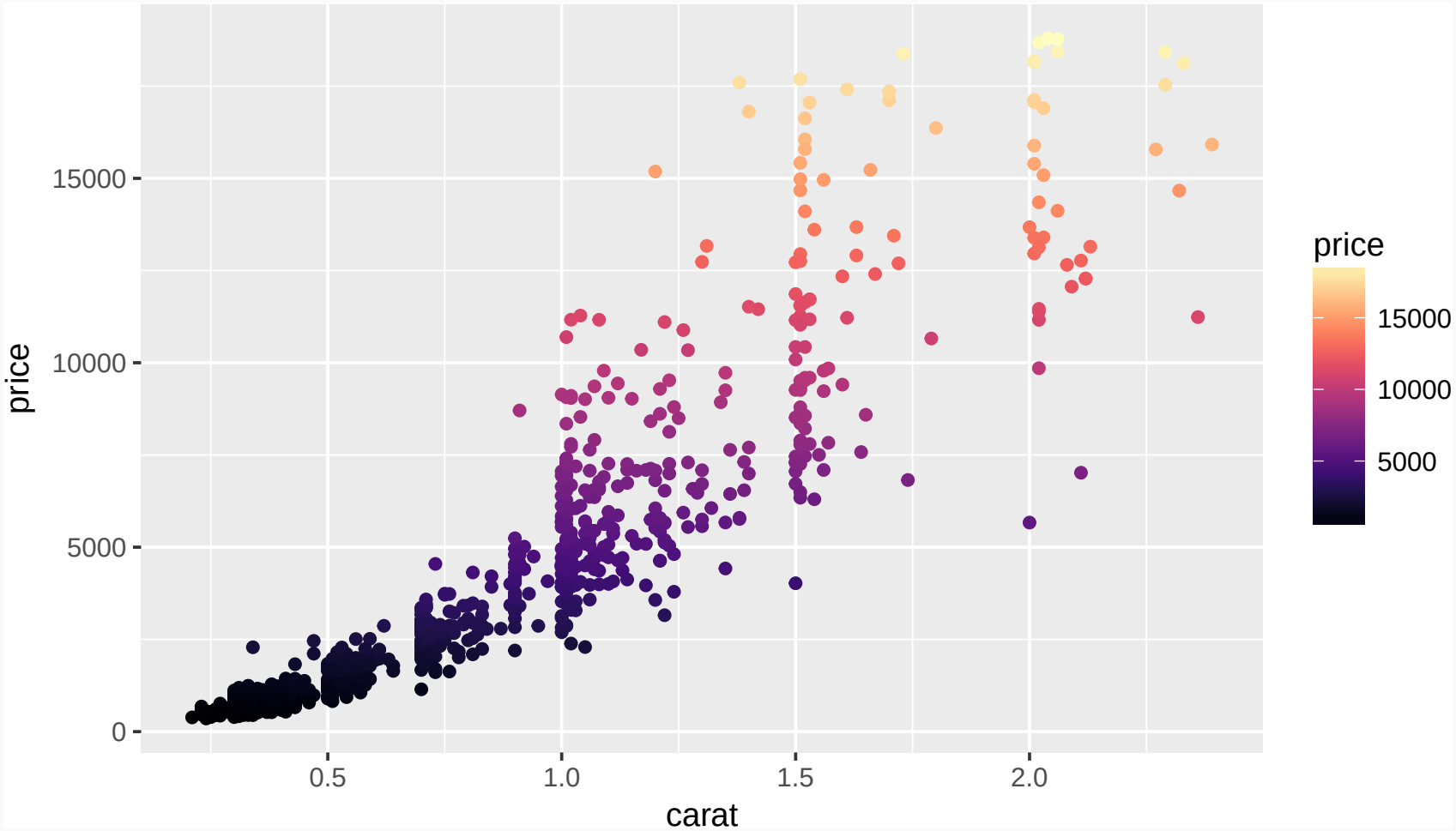
Viridis: https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html

```
library(viridis)
diamond_plot + scale_color_viridis()
```



Viridis cont'd

```
diamond_plot + scale_color_viridis(option = "magma")
```

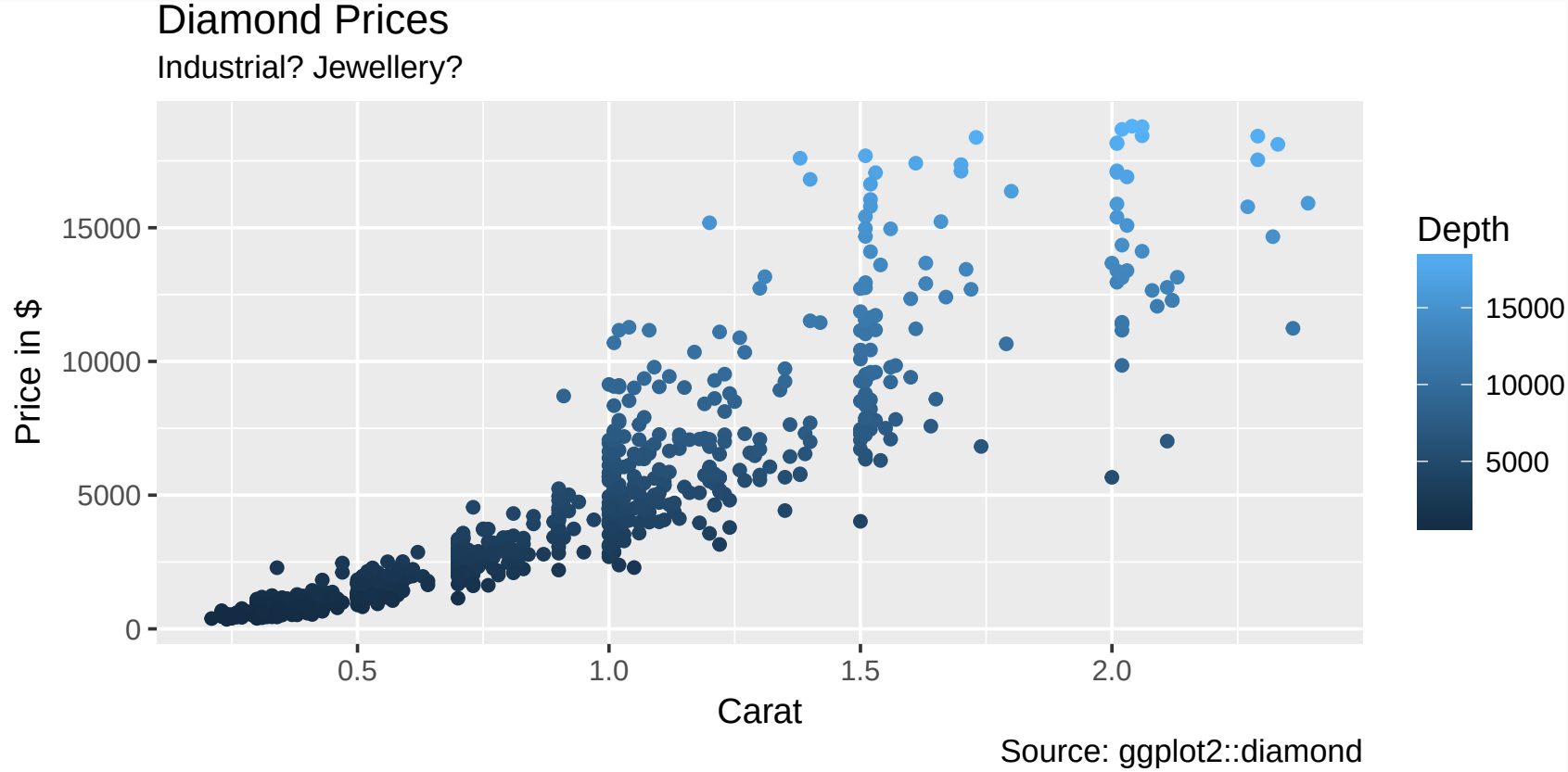


Exercises: colors

Labels

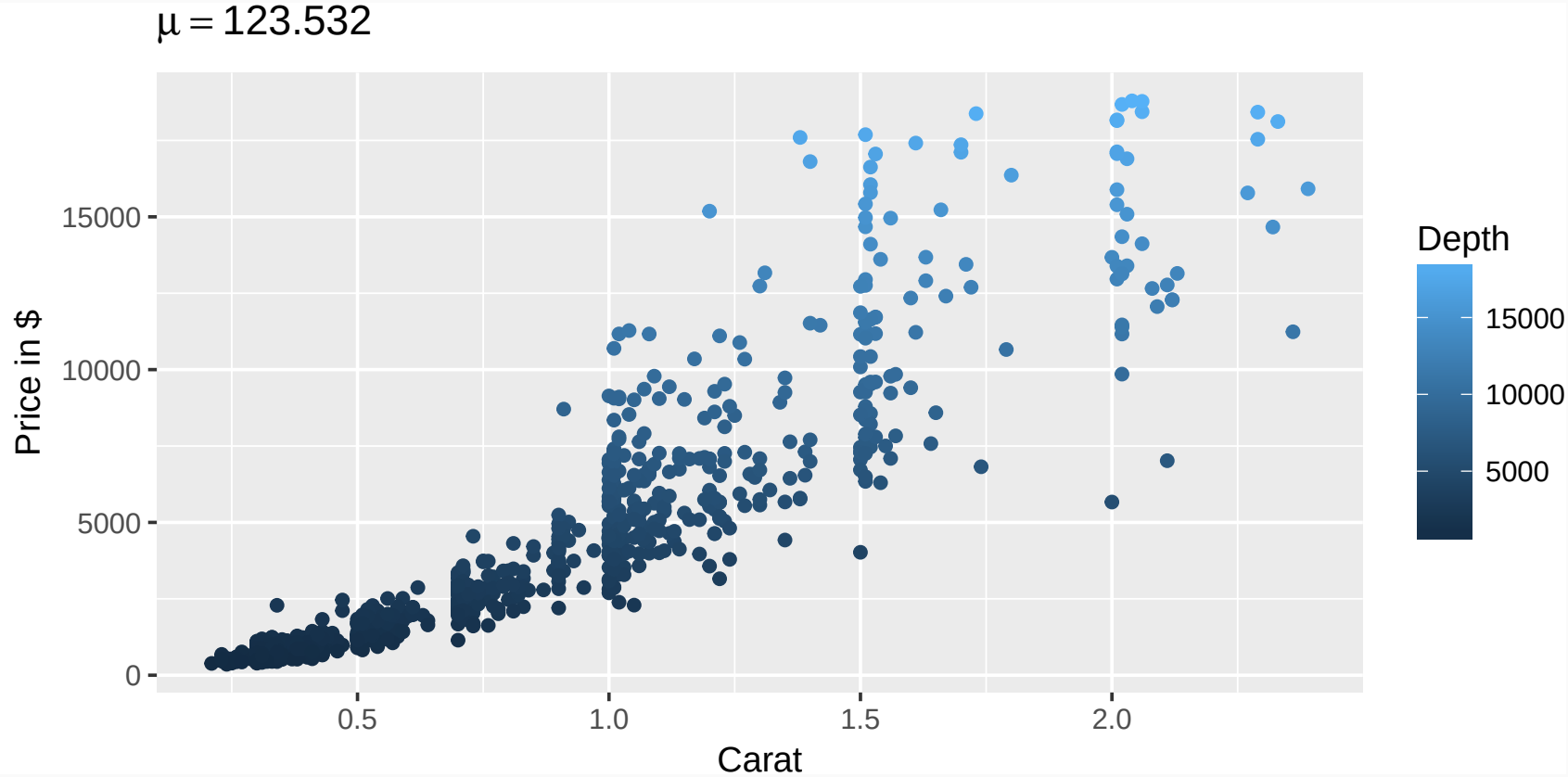
Labels

```
diamond_plot + labs(x = "Carat", y = "Price in $",
                    title = "Diamond Prices",
                    subtitle = "Industrial? Jewellery?",
                    caption = "Source: ggplot2::diamond",
                    color = "Depth")
```



Expressions

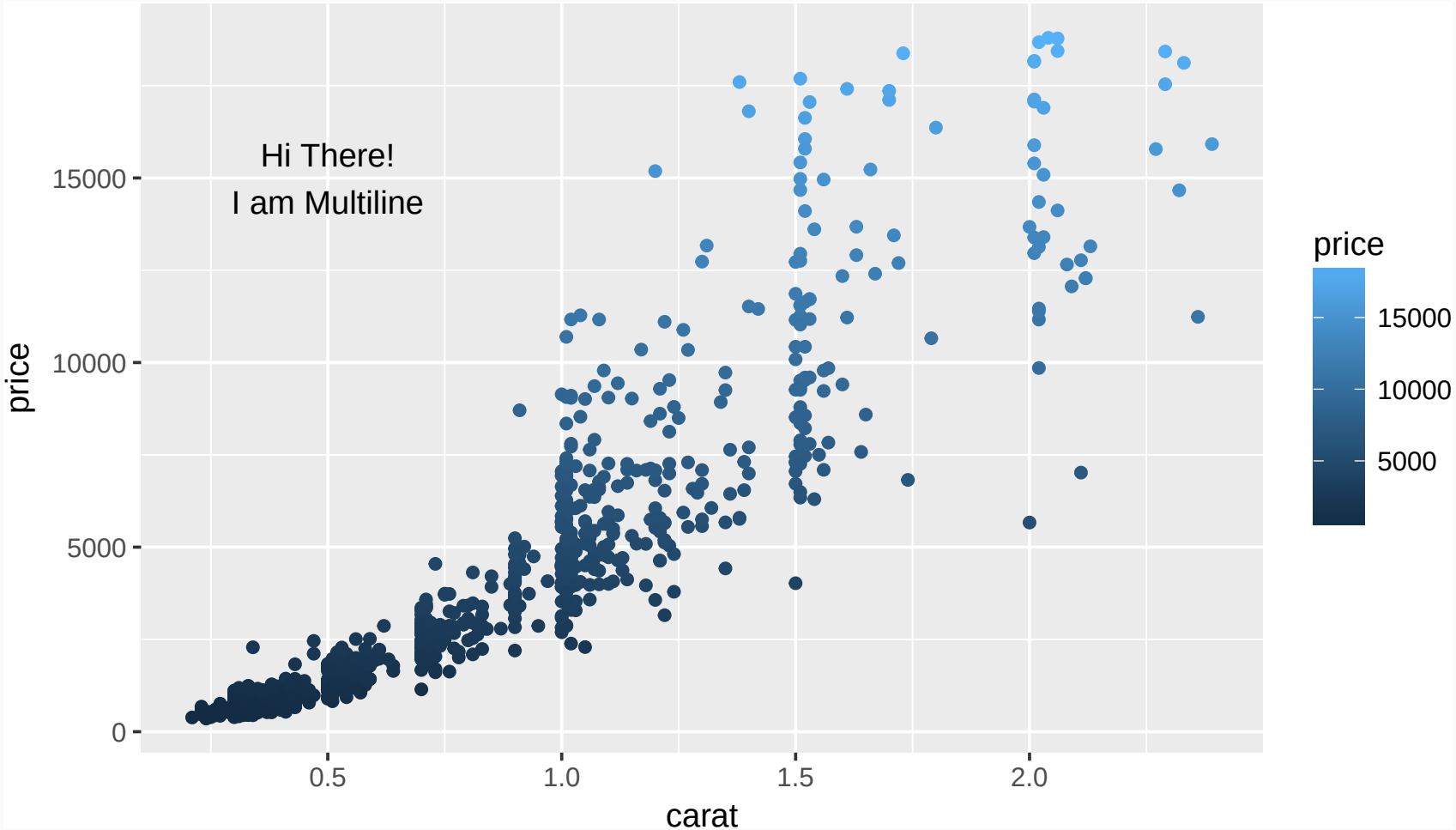
```
diamond_plot + labs(x = "Carat", y = "Price in $",
                    title = expression(mu==123.532),
                    color = "Depth")
```



More advanced expressions, see the `latex2exp`-library

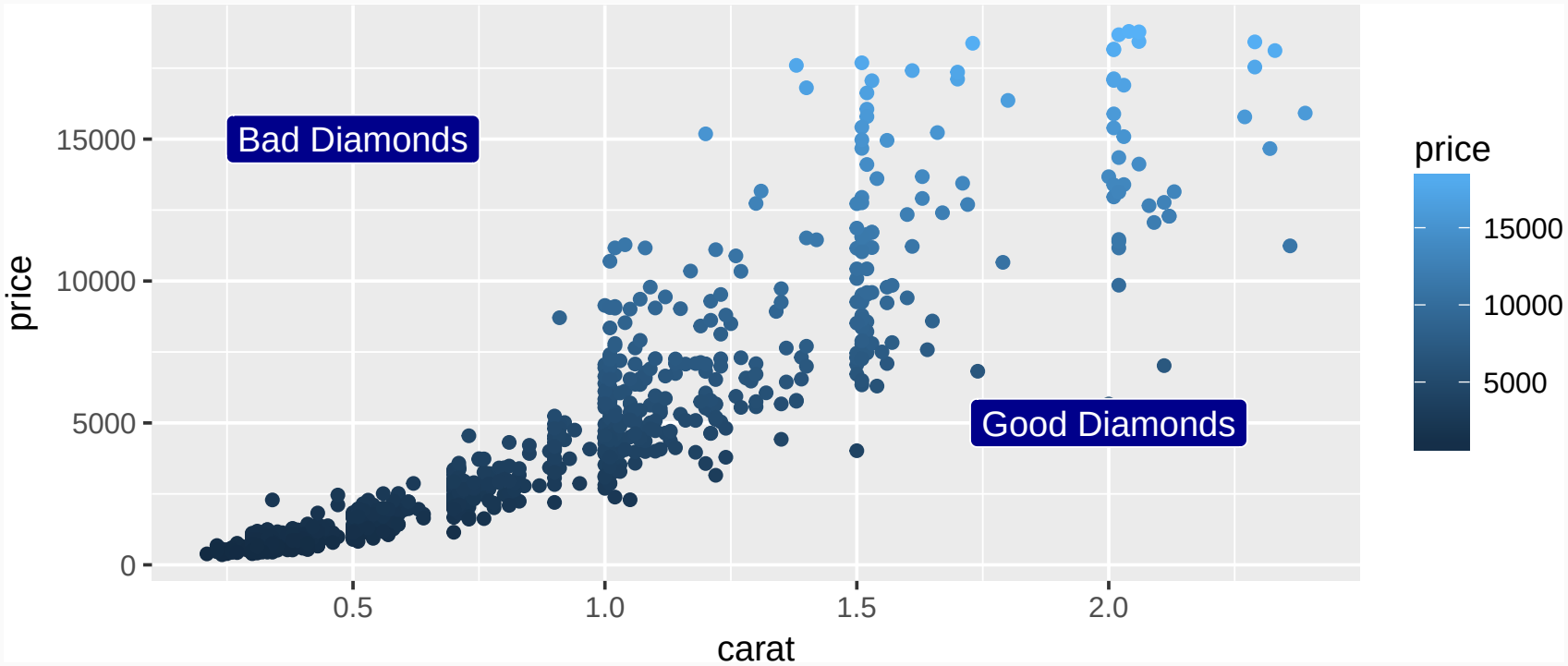
Annotation

```
diamond_plot +
  annotate("text", x = 0.5, y = 15000,
           label = "Hi There!\nI am Multiline")
```



Annotation

```
text_df <- tibble(carat = c(0.5, 2.0), price = c(15000, 5000),
                  text = c("Bad Diamonds", "Good Diamonds"))
diamond_plot +
  geom_label(data = text_df, aes(label = text),
             fill = "darkblue", color = "white")
```

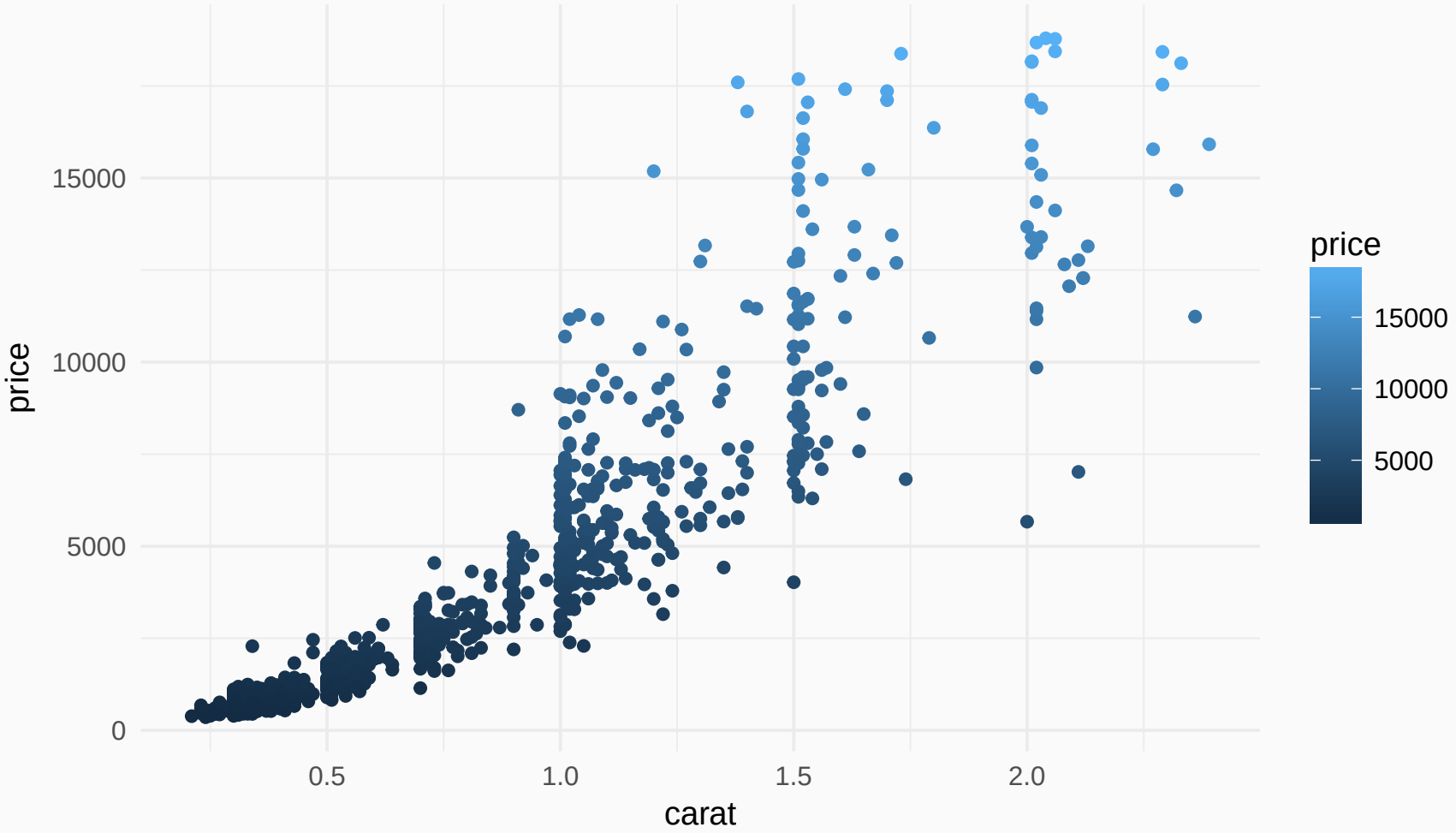


Exercises: Labels

Themes

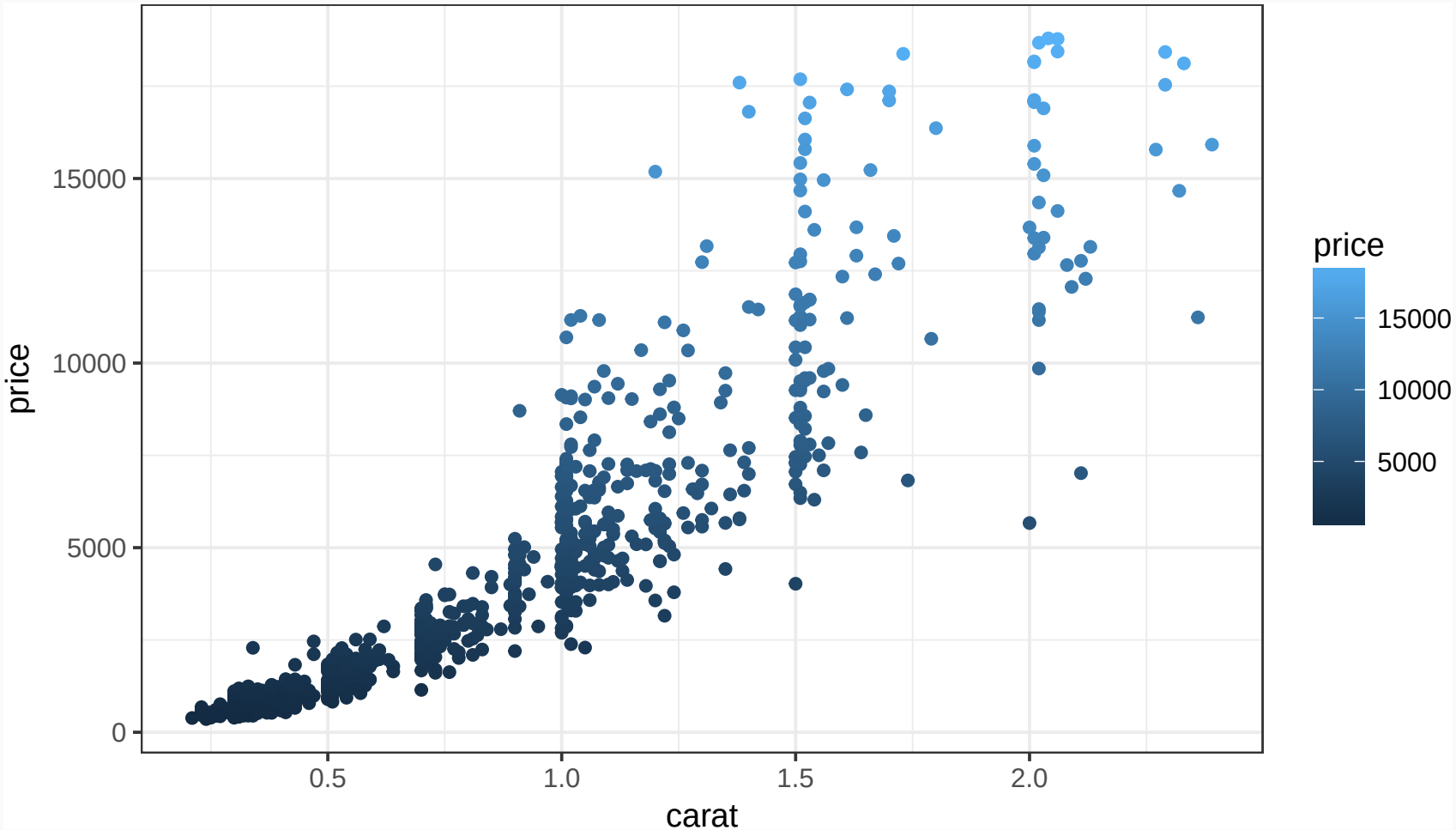
Using Themes in ggplot2

```
diamond_plot + theme_minimal()
```



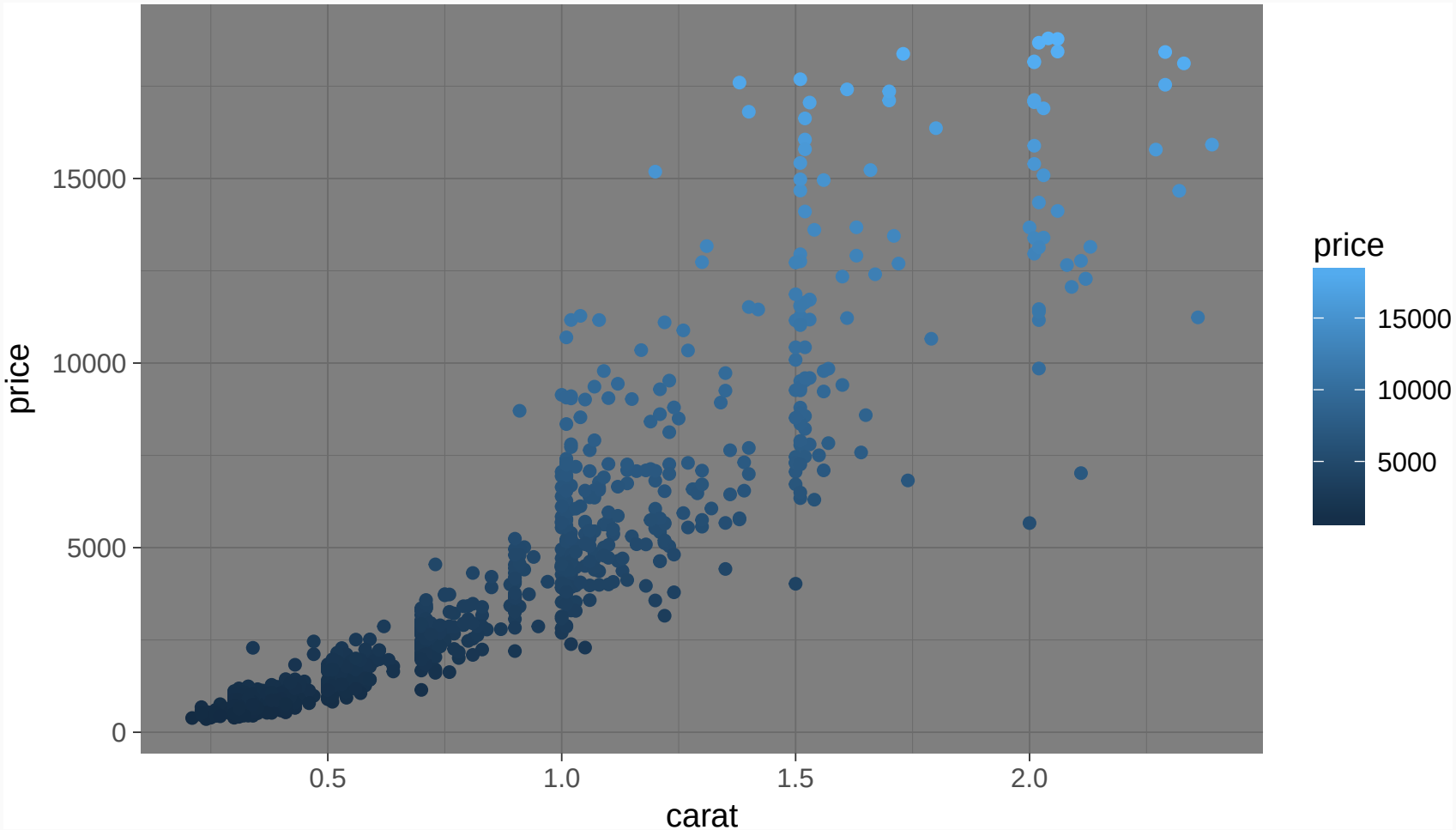
Using Themes in ggplot2 cont'd

```
diamond_plot + theme_bw()
```



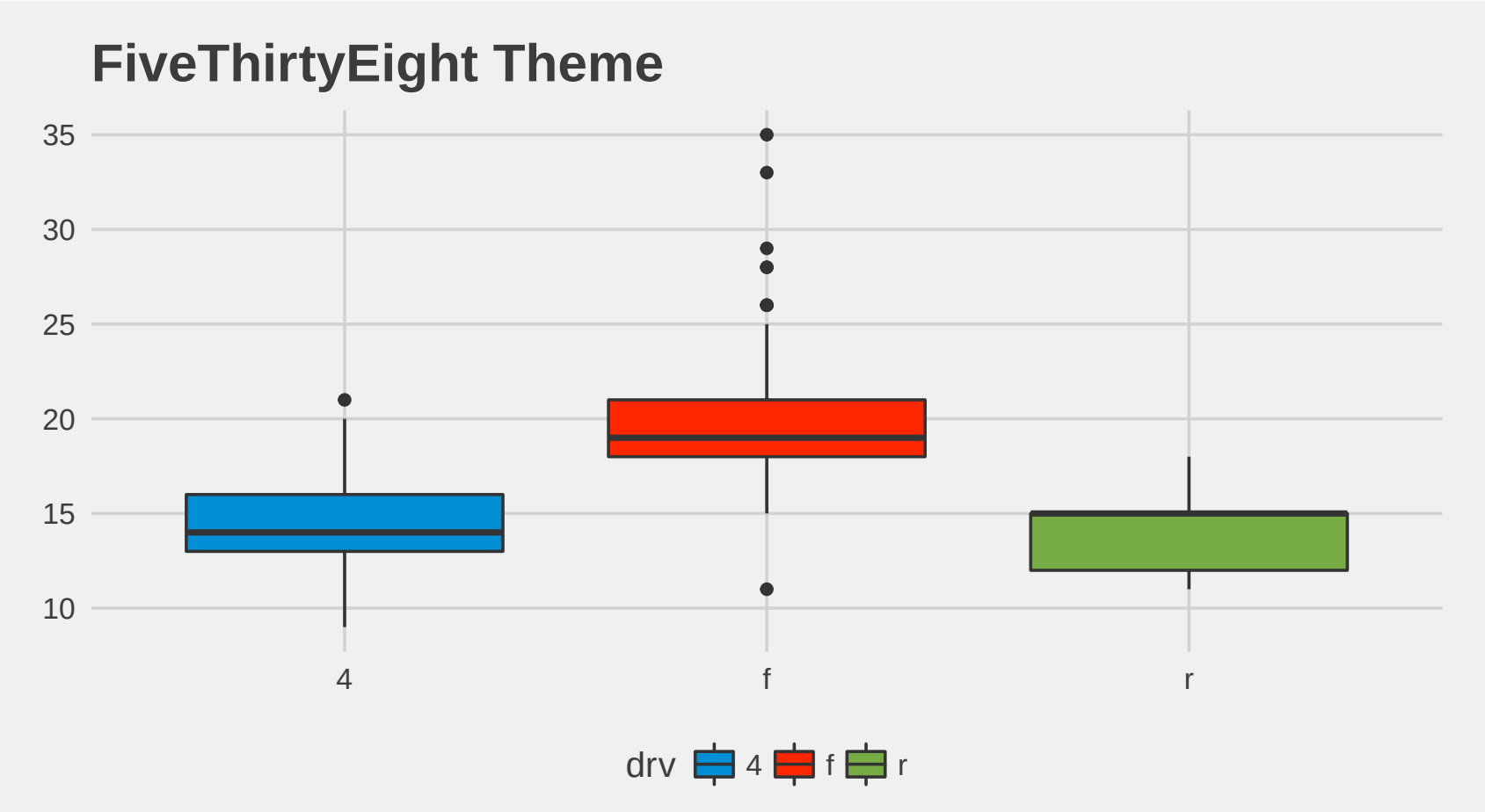
Using Themes in ggplot2 cont'd

```
diamond_plot + theme_dark()
```



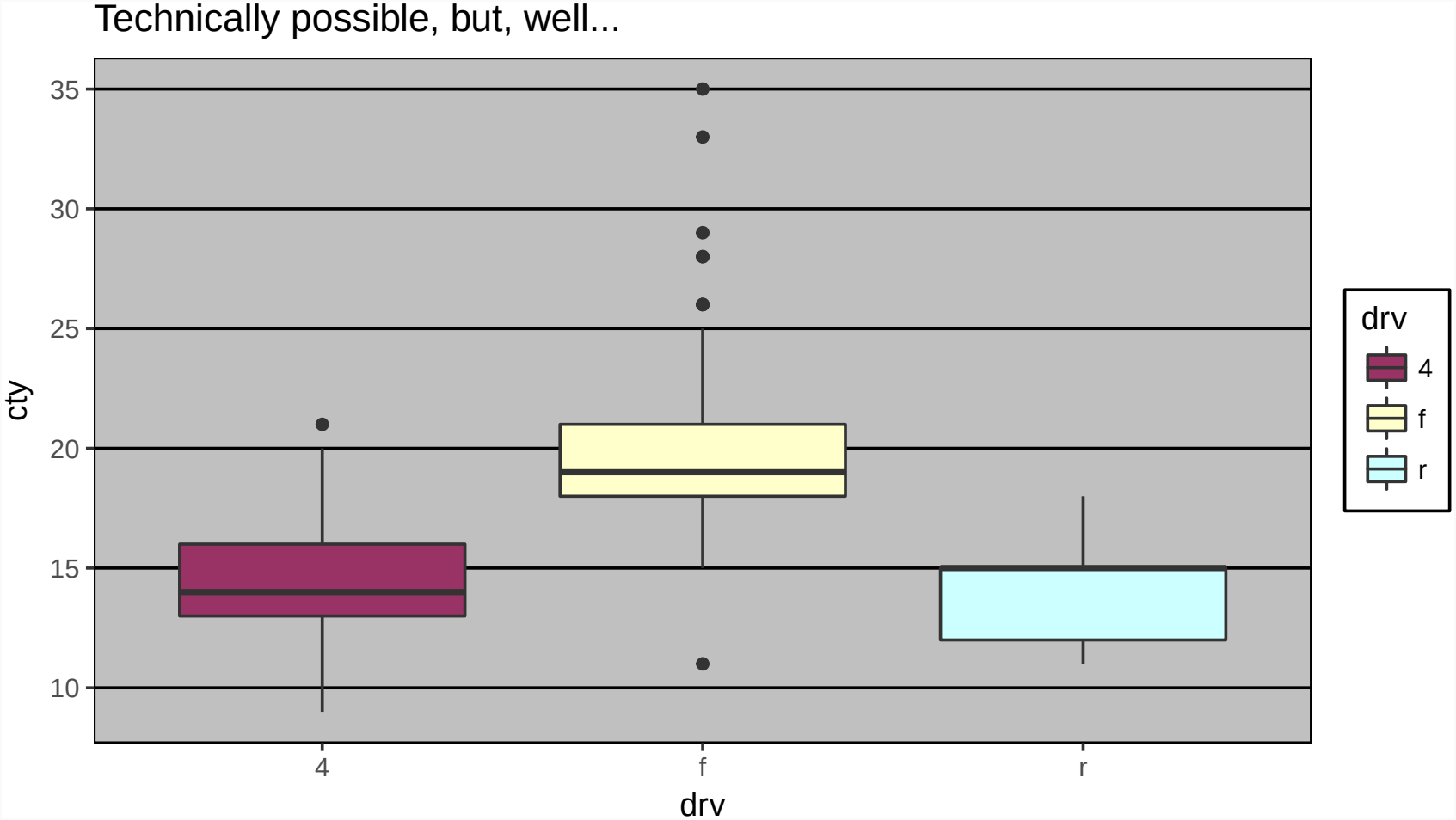
Additional Themes

```
library(ggthemes)
mpg_plot + theme_fivethirtyeight() +
  scale_fill_fivethirtyeight() +
  labs(title = "FiveThirtyEight Theme")
```



Additional Themes

```
mpg_plot + theme_excel() +
  scale_fill_excel() +
  labs(title = "Technically possible, but, well...")
```



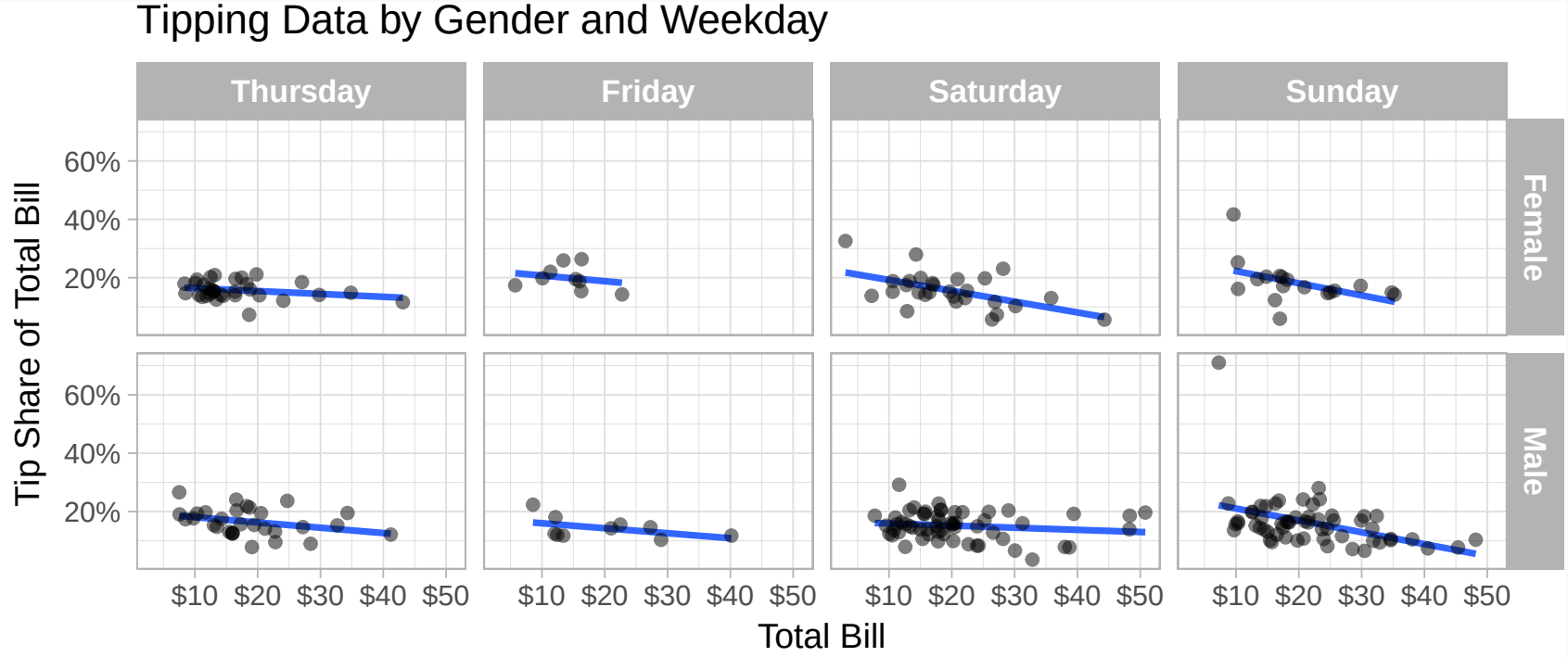
Exercises: Themes

Facetting

Idea

Facets are multiple plots side-by-side for different subgroups.

Example:

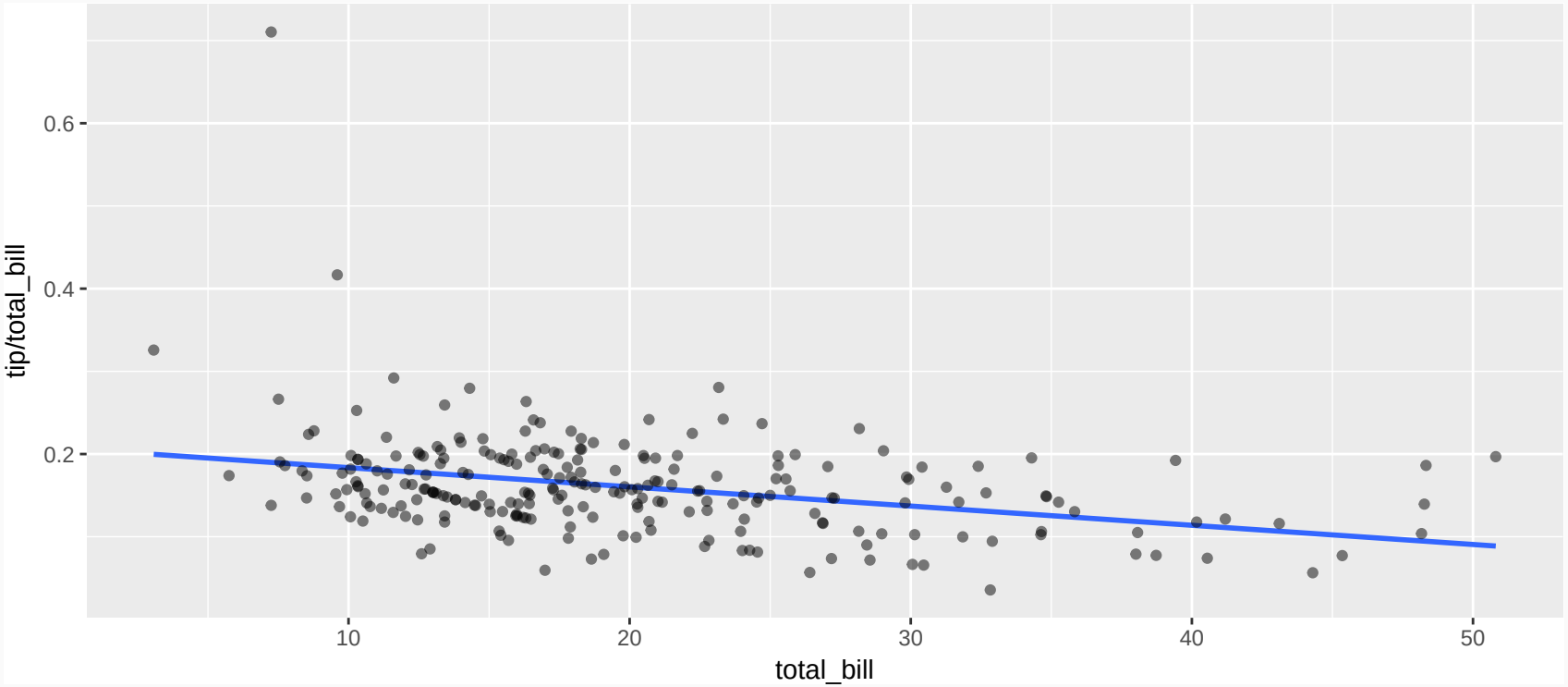


Further information:

http://www.cookbook-r.com/Graphs/Facets_(ggplot2)/

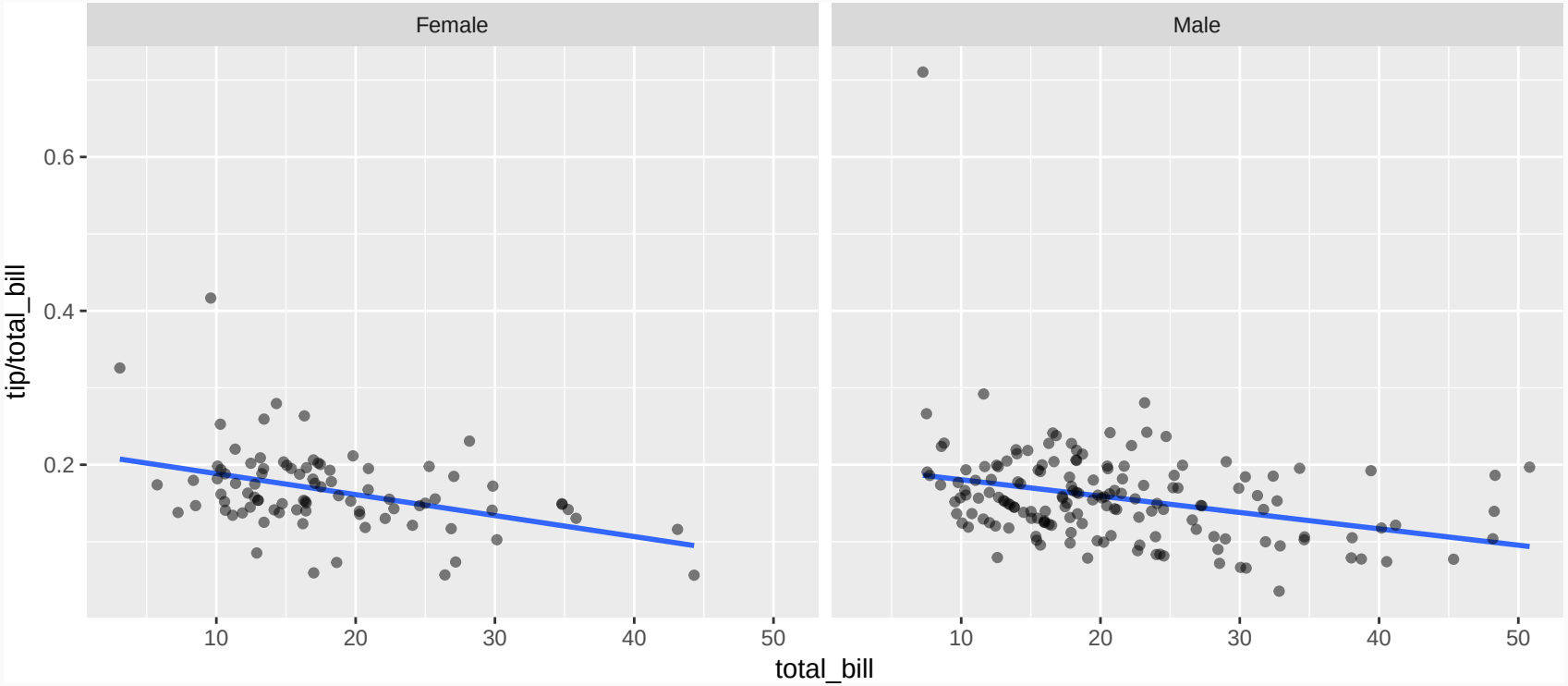
A new Plot

```
df <- as_data_frame(reshape2::tips)
tip_plot <- ggplot(df, aes(x = total_bill,
                           y = tip / total_bill)) +
  geom_smooth(method = "lm", se = F) +
  geom_point(alpha = 0.5)
tip_plot
```



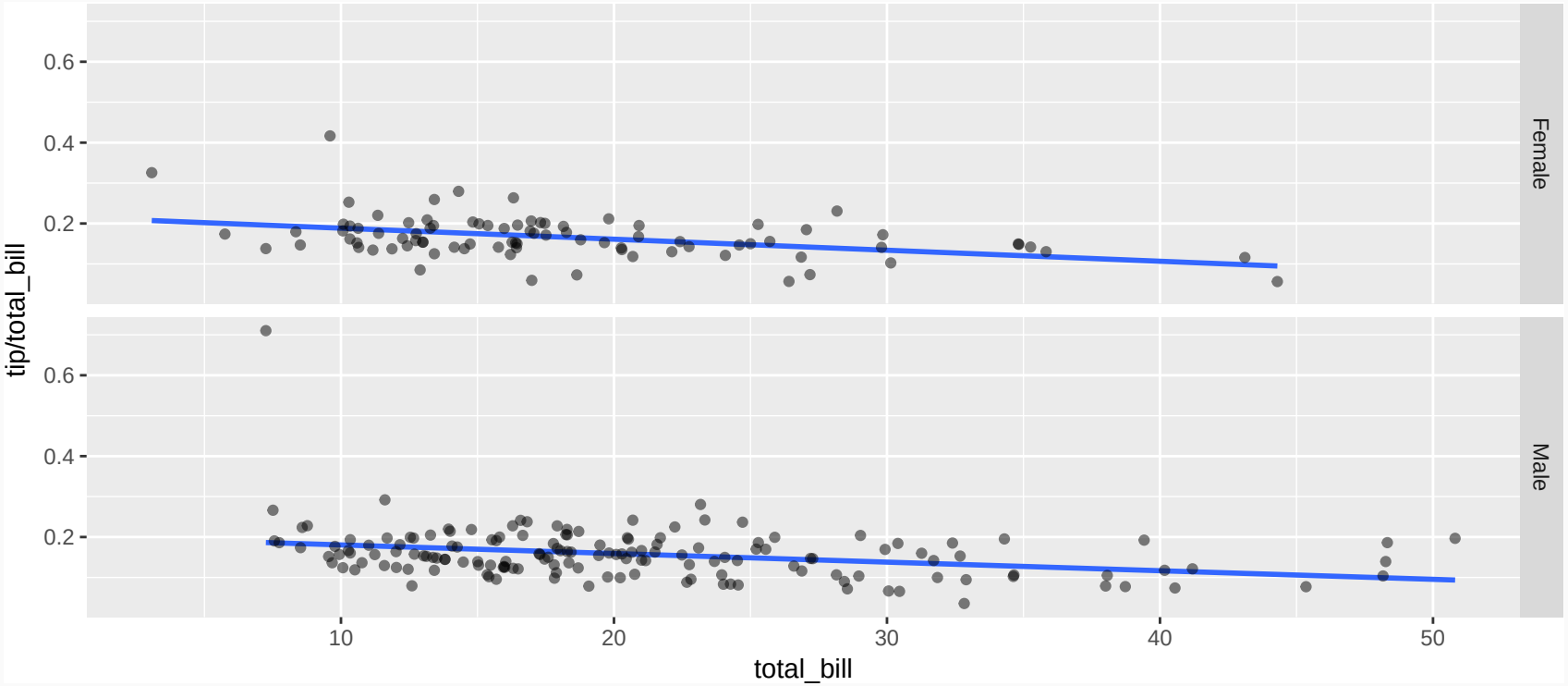
A Basic Facet

```
tip_plot + facet_wrap(~sex)
```



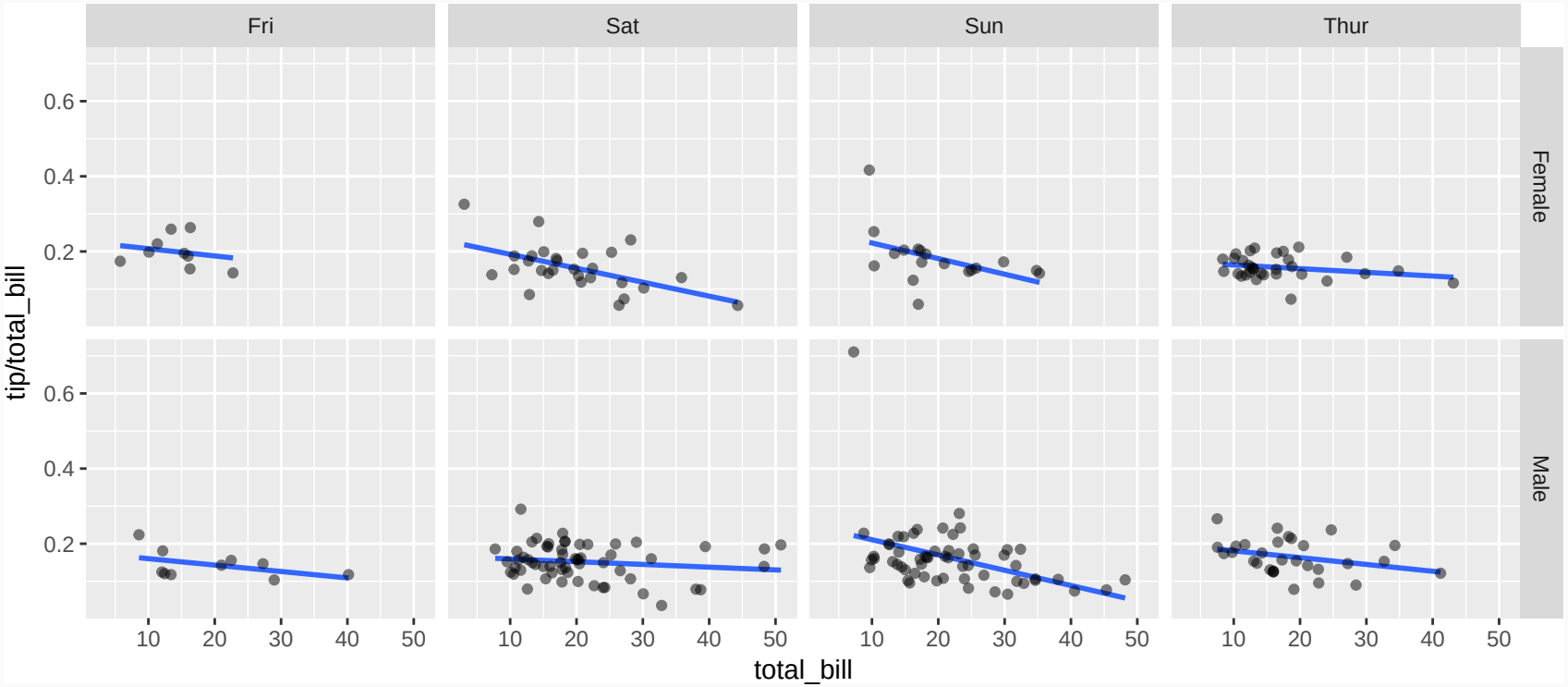
A Basic Facet-Grid

```
tip_plot + facet_grid(sex ~ .) # not facet_wrap anymore!
```



A Two-Dimensional Facet-Grid

```
tip_plot + facet_grid(sex ~ day)
```



Exercises: Facets

Maps

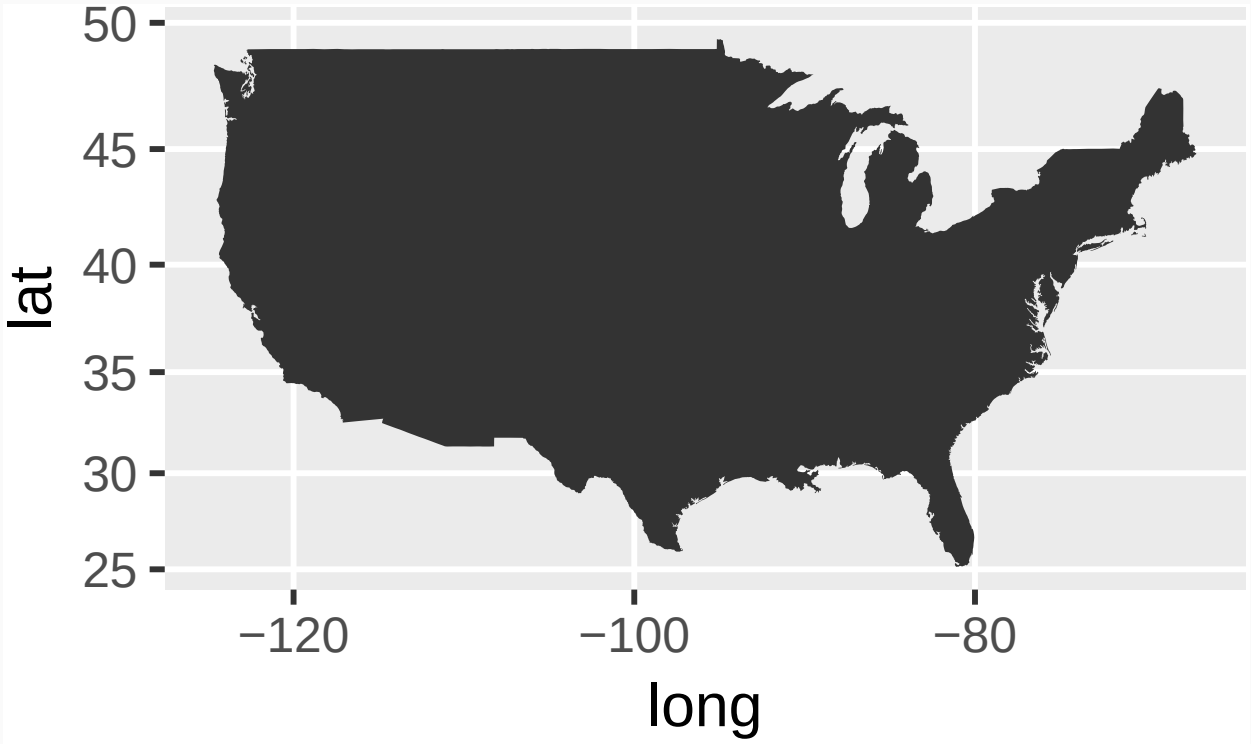
Preparing some Map Data

```
library(ggmap)
usa_df <- map_data("usa") %>% as_data_frame()
usa_df %>% head
```

```
## # A tibble: 6 × 6
##       long      lat group order region subregion
##      <dbl>    <dbl> <dbl> <int>  <chr>     <chr>
## 1 -101.4078 29.74224     1     1   main      <NA>
## 2 -101.3906 29.74224     1     2   main      <NA>
## 3 -101.3620 29.65056     1     3   main      <NA>
## 4 -101.3505 29.63911     1     4   main      <NA>
## 5 -101.3219 29.63338     1     5   main      <NA>
## 6 -101.3047 29.64484     1     6   main      <NA>
```

A basic Map

```
map_plot <- ggplot(usa_df, aes(x = long, y = lat)) +
  geom_polygon(aes(group = group)) + coord_map()
map_plot
```



Question: why the `group = group` inside of `geom_polygon()` and not `ggplot()`?

Expanding the Map: Data

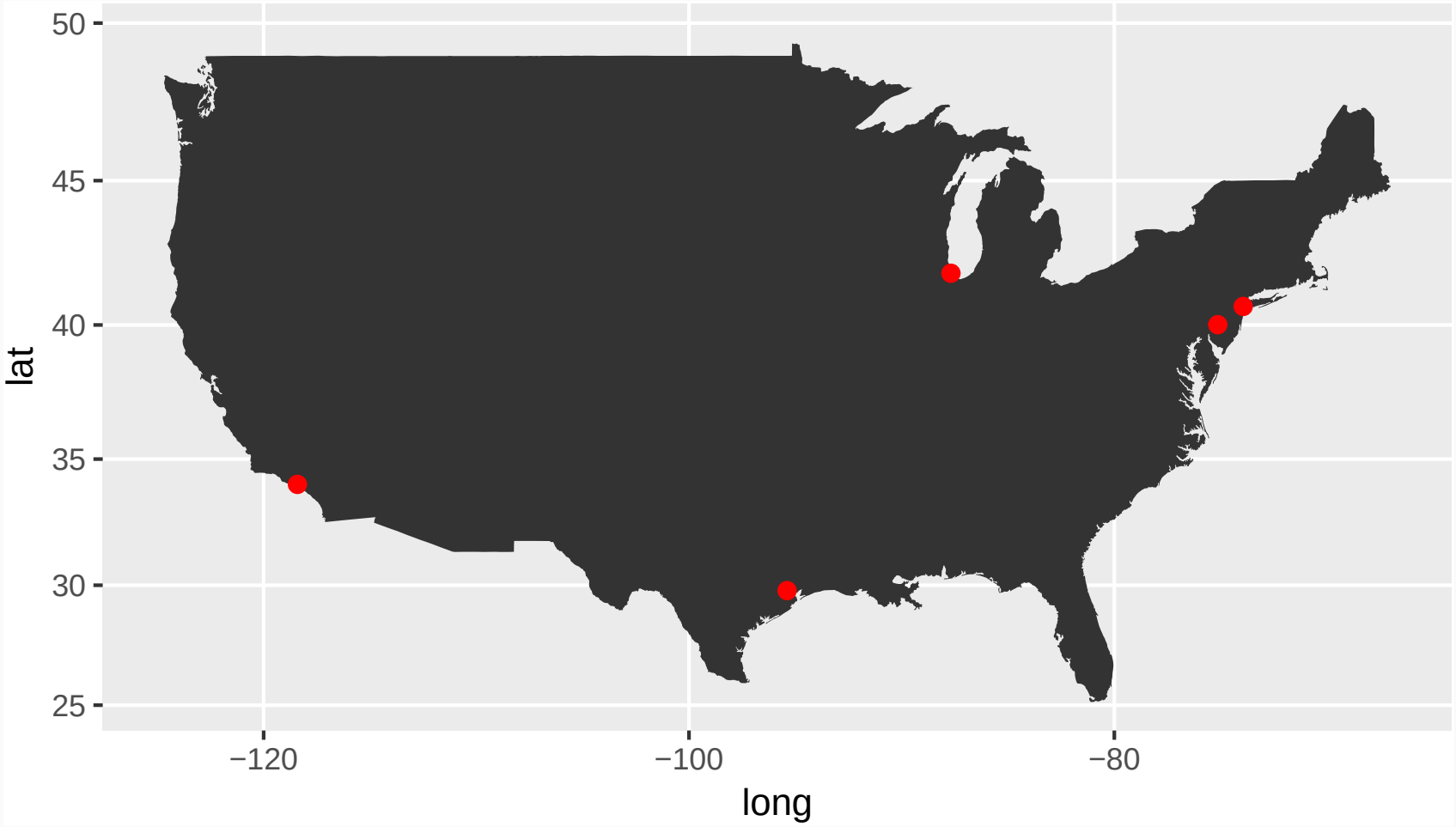
First we need to add some data, i.e., the 5 largest cities

```
cities <- c("New York", "Los Angeles", "Chicago", "Houston",
            "Philadelphia")
us_cities <- tibble(name = cities,
                    pop = c(8.550, 3.972, 2.271, 2.296, 1.567),
                    lat = c(40.6643, 34.0194, 41.8376, 29.7805,
                            40.0094),
                    lon = c(-73.9385, -118.4108, -87.6818,
                            -95.3863, -75.1333))
```

Source:
https://en.wikipedia.org/wiki/List_of_United_States_cities_by_population

Expanding the Map: Plot

```
map_plot +
  geom_point(data = us_cities, aes(x = lon, y = lat),
             color = "red", size = 2)
```



Expanding the Plot: Formatting

```
library(ggrepel)
map_final <- ggplot(usa_df, aes(x = long, y = lat)) +
  coord_map() + theme_void() +
  geom_polygon(aes(group = group), fill = "lightgrey") +
  labs(title = "Largest Cities in the US",
       subtitle = "Displaying the 5 largest cities as of 2015",
       caption = "Data source: Wikipedia",
       size = "Population\nin Mio") +
  geom_label_repel(data = us_cities,
                   aes(x = lon, y = lat, label = name),
                   fill = "#e0162b") +
  geom_point(data = us_cities, aes(x = lon, lat, size = pop),
             color = "#0052a5")
```

Expanding the Plot: Formatting

```
map_final
```

More Advanced Maps

Interactive Maps: Leaflet https://rstudio.github.io/leaflet/

Google & OSM: ggMaps https://blog.dominodatalab.com/geographic-visualization-with-rs-ggmaps/

Exercises: Maps

Additional Stuff

Open Questions? Feedback?

Contact

All Feedback welcome (in person, per mail, smoke signals or whatever you prefer.)

- Mail: david_j_zimmermann@hotmail.com or david.zimmermann@zu.de
- Blog: https://datashenanigan.wordpress.com/
- Github: https://github.com/DavZim

Next steps

- Fast computation on large datasets? -> `data.table`
- Want to dive into Machine Learning? Look at www.kaggle.com and sign up for a competition, look at good r-scripts. Understand the models used (might take some time; as in a lifetime). Repeat for other models. Also look into the libraries: `caret`, `e1071`, `h20` or `mxnet`
- Forecasting needs? -> `forecast`, have fun understanding the models
- Business context and databases? Learn `SQL` and use `dplyrs` sqlite3 connection
- Interactive Dashboards? Use `shiny` to create cool stuff (see https://shiny.rstudio.com/gallery/)
- Quantitative Finance? `RQuantLib` has got you covered also have a look at https://github.com/DavZim/OptionValuation
- Need for speed? I.e., High-Performance Computing (HPC), look into `C++` and the R library `RCpp` or into multi-threading using `parallel`, `snowfall`, or `foreach`
- and much much more...

Appendix

Overlaying ggplots

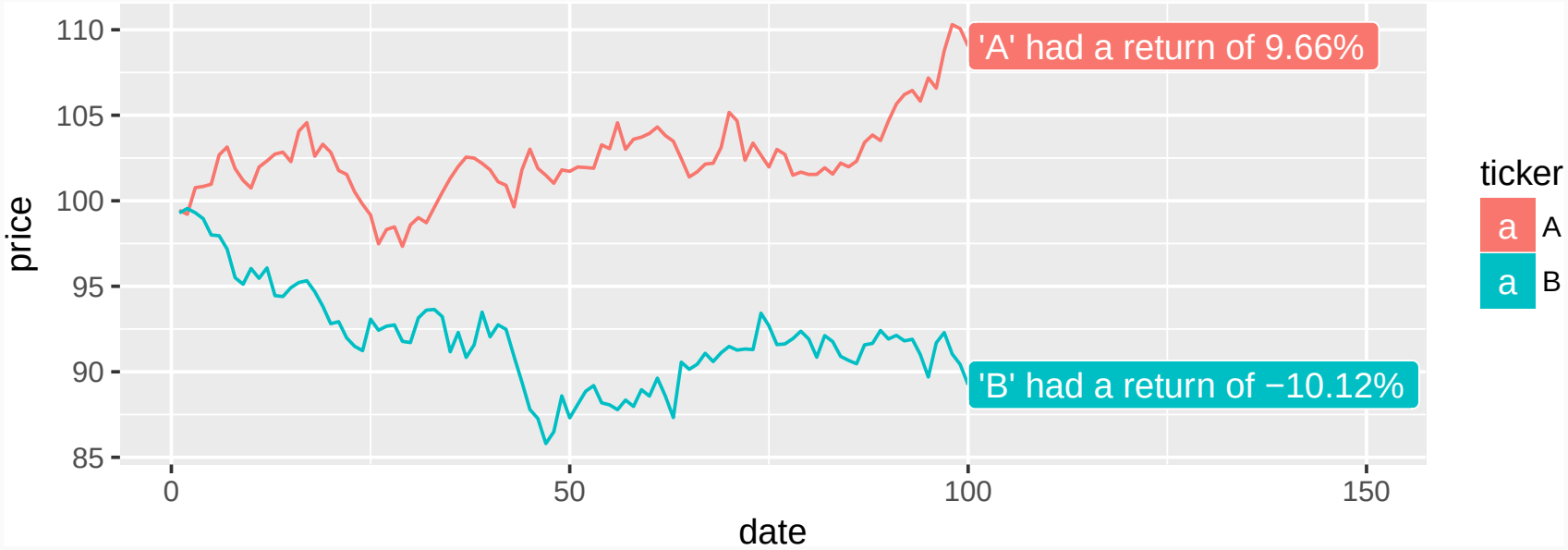
The function-call `ggplot(df, aes(x = x, y = y))` maps (globally for the plot) the variables. But we can also separate the mappings per geom. First, lets create some stock data and compute the overall return

```
set.seed(123);n <- 100
df_data <- tibble(date = rep(1:n, 2),
                  ticker = rep(c("A", "B"), each = n),
                  ret = rnorm(2*n)) %>%
  group_by(ticker) %>%
  mutate(price = 100 + cumsum(ret))

df_text <- df_data %>%   group_by(ticker) %>%
  summarise(overall_ret = price[n()] / price[1] - 1,
            last_price = price[n()],
            date = date[n()]) %>%
  mutate(text = sprintf("'%s' had a return of %.2f%%",
                        ticker, overall_ret * 100))
```

Overlapping ggplots cont'd

```
ggplot(df_data, aes(x = date, y = price, color = ticker)) +
  geom_line() +
  geom_label(data = df_text,
             aes(y = last_price, label = text, fill = ticker),
             hjust = 0, color = "white") +
  expand_limits(x = 150)
```



Overlapping ggplots cont'd

We can also shift everything to the geoms, but then we need to specify all information (x, y + ...) in each geometric (compare the `aes()` from before and now).

```
ggplot() +
  geom_line(data = df_data,
            aes(x = date, y = price, color = ticker)) +
  geom_label(data = df_text,
             aes(x = date, y = last_price, label = text, fill = ticker),
             hjust = 0, color = "white") +
  expand_limits(x = 150)
```