Introduction to Data Visualization in R Using ggplot2

Guido Ropers

University of Mannheim (CDSS)

03.05.2019

Data visualization: Overview

1. `ggplot2` basics
2. Themes & output
3. Model visualization

Useful Resources

- **ggplot2 cheatsheet:** https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf
- **For inspiration:** https://www.r-graph-gallery.com/
- **Winston Chang (2109). "R Graphics Cookbook":** https://www.r-graphics.org
- **Claus O. Wilke: "Fundamentals of Data Visualization":** https://serialmentor.com/dataviz/

Plotting in R

1. base `plot` package (default)
   - most common
   - conceptually simple
   - vector based
2. `lattice`
3. `ggplot2`
   - unified interface
   - builds on `tidyverse` logic
   - designed to work with datasets

```
library(ggplot2)
```

A first glimpse at ggplot2 I

- We'll use the dataset diamonds included in ggplot2

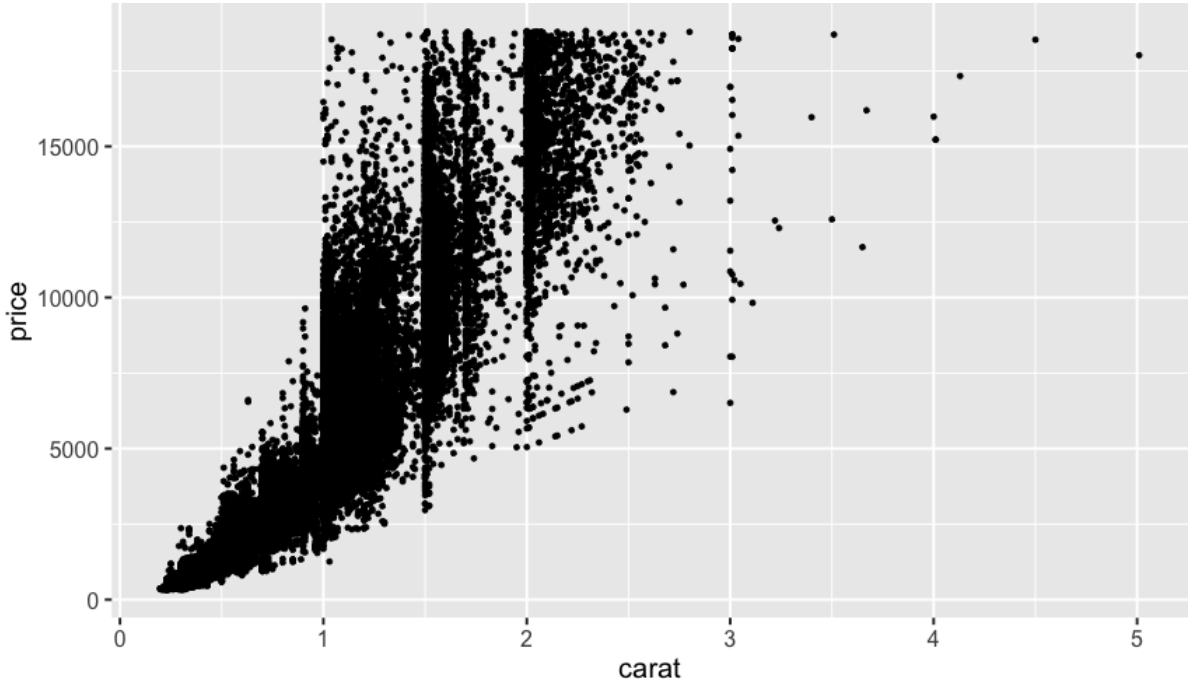
```
data(diamonds)
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut   color clarity depth table price     x
##   <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl>
## 1 0.23  Ideal E     SI2      61.5    55   326  3.95
## 2 0.21  Prem~ E     SI1      59.8    61   326  3.89
## 3 0.23  Good  E     VS1      56.9    65   327  4.05
## 4 0.290 Prem~ I     VS2      62.4    58   334  4.2
## 5 0.31  Good  J     SI2      63.3    58   335  4.34
## 6 0.24  Very~ J     VVS2     62.8    57   336  3.94
## # ... with 2 more variables: y <dbl>, z <dbl>
```

A first glimpse at ggplot2 II

- We'll use the dataset `diamonds` included in `ggplot2`
- Here's a scatter plot of the price of diamonds against carat.

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



The grammar of graphics

Required:

1. `data`: that's what we want to visualise
2. `aesthetics`: variables which are mapped to aesthetic attributes
3. `geoms`: visual marks drawn to represent data

Optional:

4. `stats`: statistical transformations
5. `scales`: controls the mapping of values in data to values in aesthetic space
6. `coord`: coordinate system, most of the cases cartesian
7. `facet`: facetting breaks up plot in several group-specific subplots

ggplot2

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

1. `ggplot()` is the main function and takes `data` as input
2. `mapping = aes()` as second key input
   - `aes()` sets the aesthetics, `x` and `y` variables but also other variables depending on the plot
3. `+` adds a new layer on top of the plot
4. `geom_point()` is a geometric object, it draws dots at the x-y-coordinates
5. No statistical transformations
6. `scales` are given on the axis
7. `coord` is cartesian
8. No facetting

Data

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

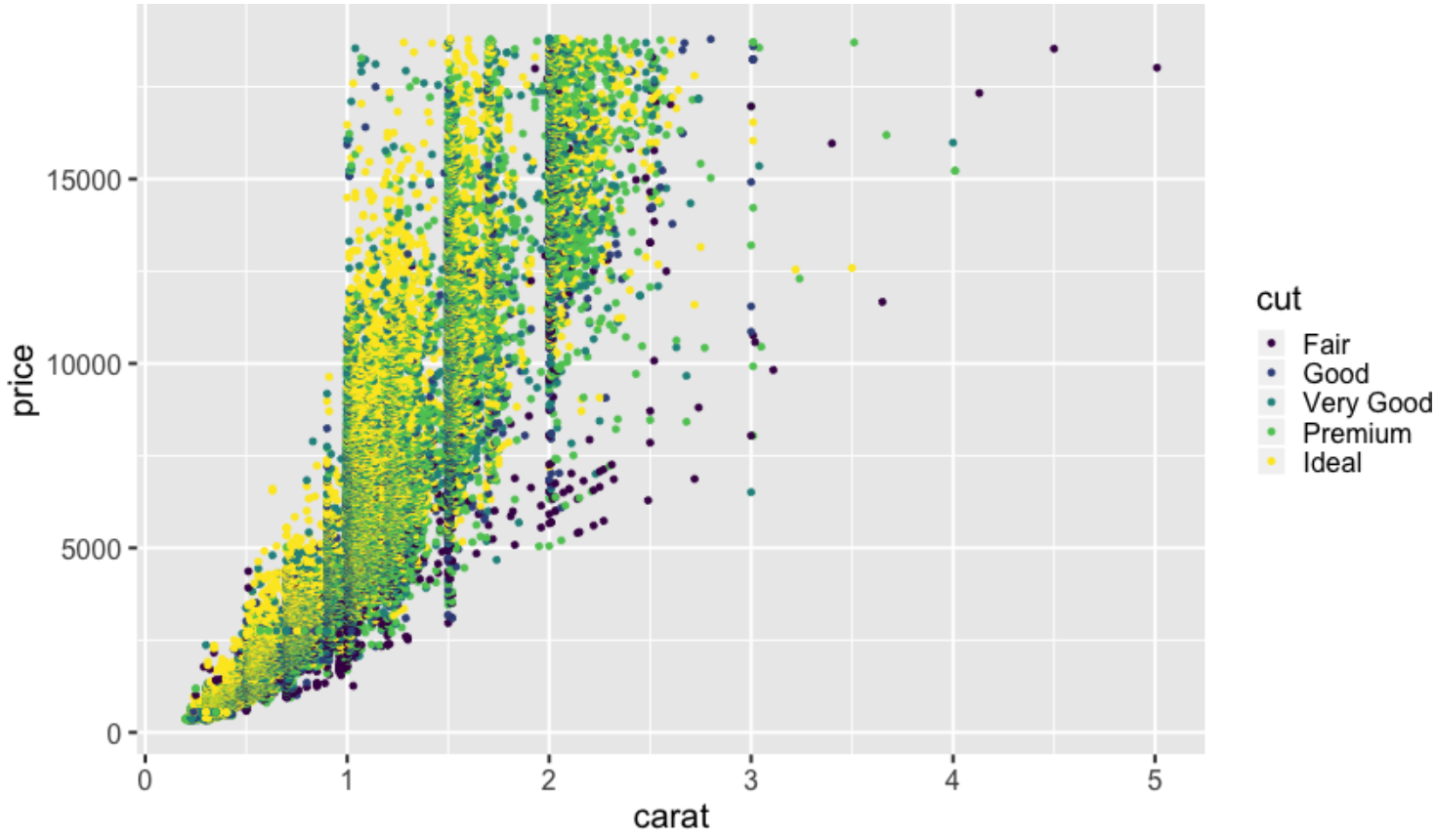
- `data` is the first argument in `ggplot()`
- the input needs to be a `data.frame` or `tibble`
- `geom_...()` can also take a `data.frame` as second argument

Mapping

- `aes()` knows the following arguments:
  - `x` and `y` (also `xmin`, `xmax`, `xend`, `ymin`, `ymax`, `yend`)
  - `group` seperates the data into groups for separate geoms
  - `fill` for filling geometric objects with color
  - `color` for coloring dots and outlines of other objects
  - `shape` to determine the shape of e.g. `geom_point()`
  - `size` to determine the size of points or width of lines

Adding color

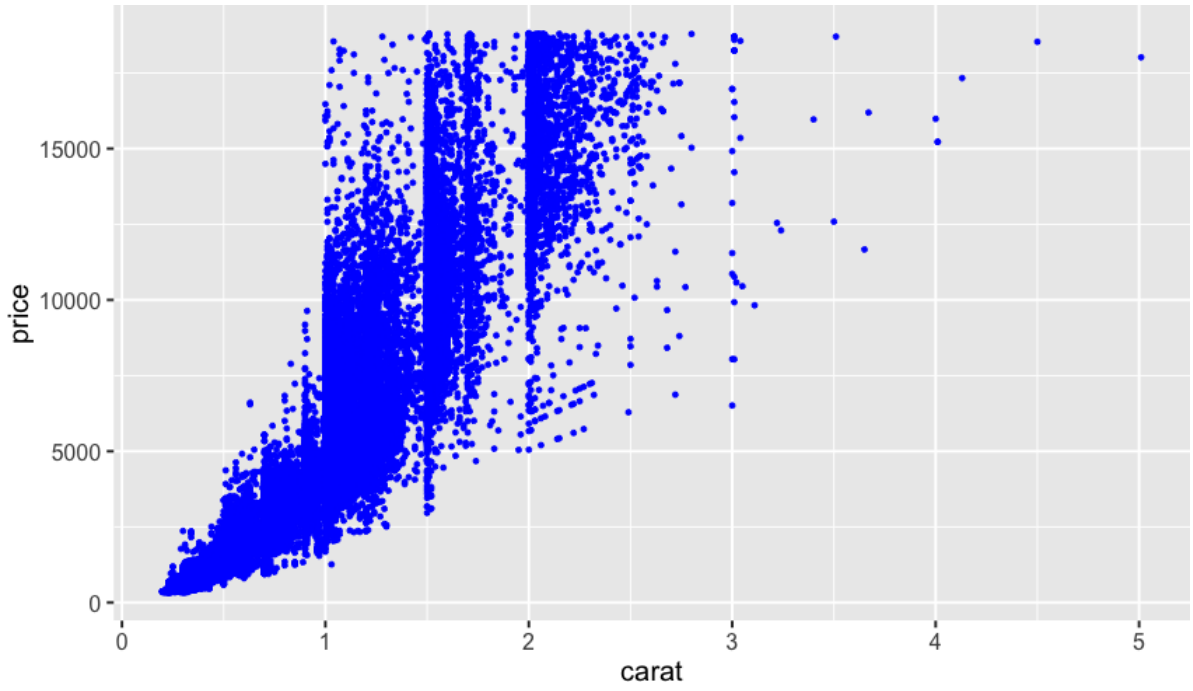
```
ggplot(data = diamonds,
  aes(x = carat, y = price, color = cut)) +
  geom_point()
```



Aesthetics vs. settings I

- Aesthetics represent mappings to visual marks based on variables in the data
- Settings specify the aesthetics without relying on data

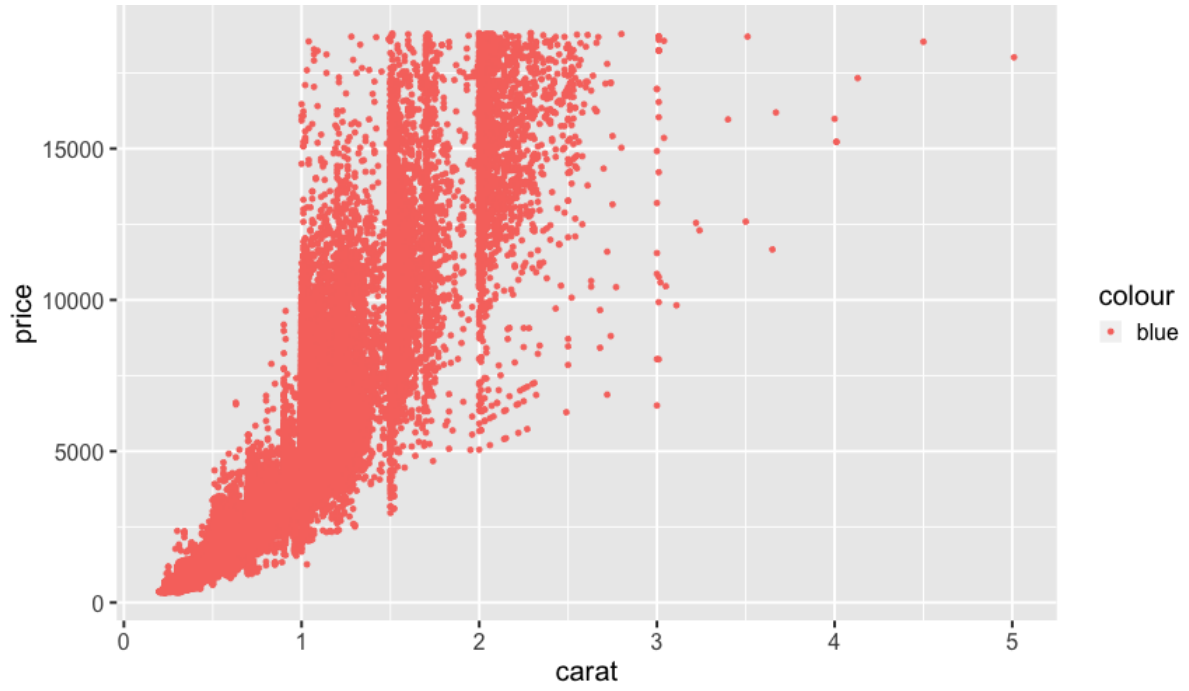
```
ggplot(diamonds, aes(x = carat, y = price)) +
    geom_point(color = 'blue')
```



Aesthetics vs. settings II

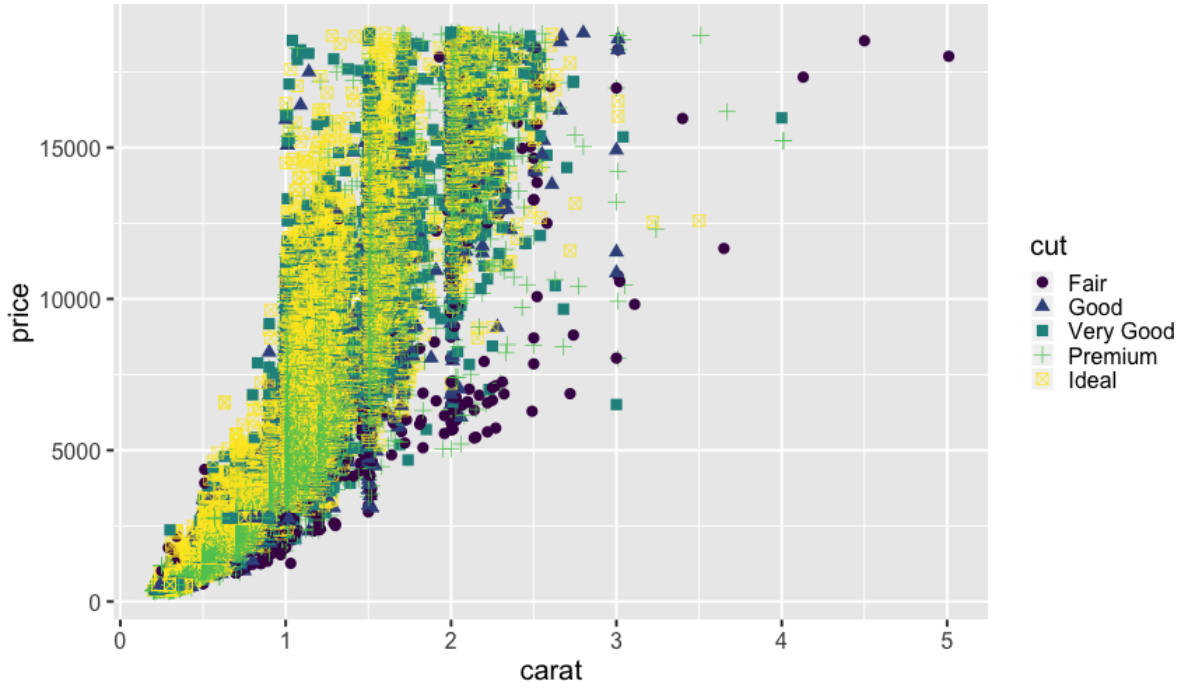
- Aesthetics represent mappings to visual marks based on variables in the data
- Settings specify the aesthetics without relying on data

```
ggplot(diamonds
       , aes(x = carat, y = price, color = 'blue')) +
  geom_point()
```



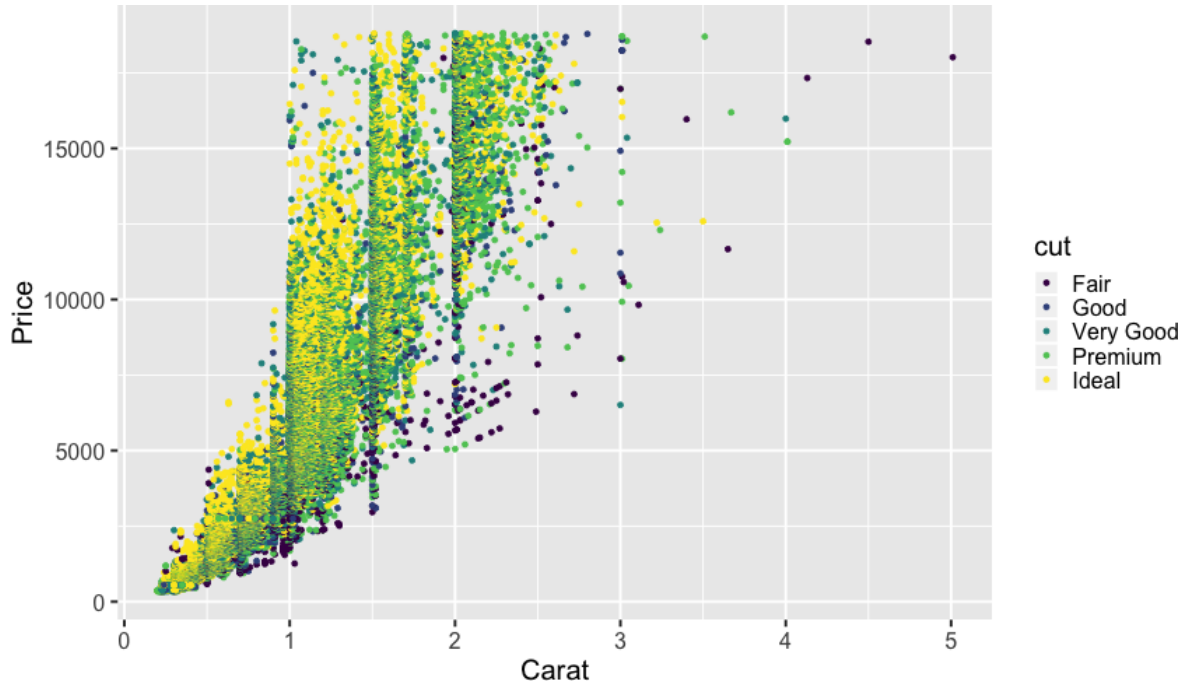
Modifying shapes

```
ggplot(diamonds
       , aes(x = carat, y = price, color = cut
             , shape = cut)) +
  geom_point(size = 3.5)
```



Labels I

```
#Let's make our plot look nicer
ggplot(diamonds
       , aes(x = carat, y = price, color = cut)) +
  geom_point() +
  xlab("Carat") + ylab("Price")
```



Labels II

```
#Let's make our plot look nicer
ggplot(diamonds, aes(x = carat, y = price, color = cut)) +
  geom_point() +
  labs(x = "Carat", y = "Price", title = "Scatterplot"
       , color = "Quality of the Cut")
```

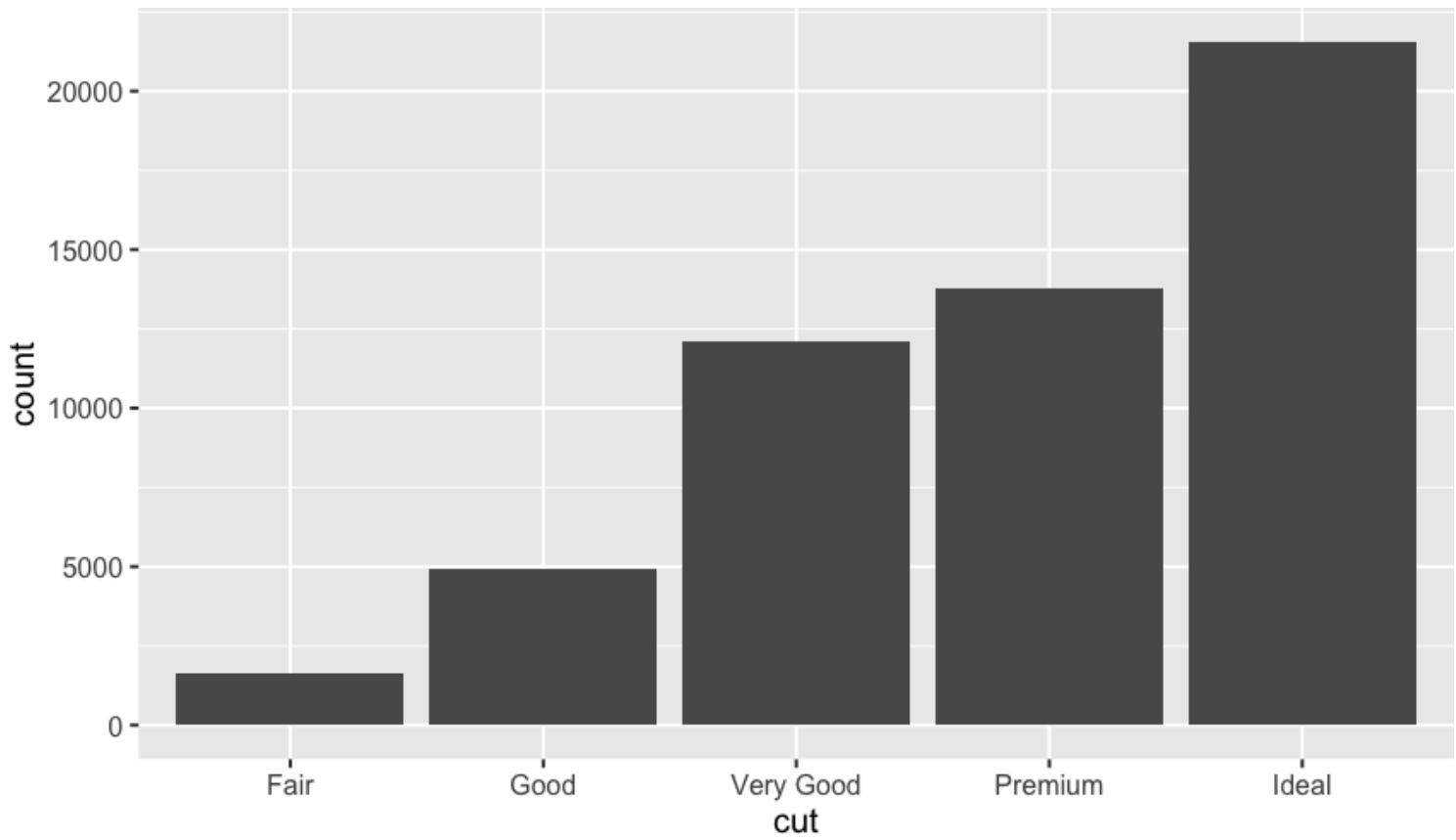
Geoms

Usage depends on number and type of variables

- Scatter plot: `geom_point()`
- Bar charts:
  - `geom_bar()` plots the count
  - `geom_col()` plots discrete variable against a continuous
- Distribution
  - `geom_histogram()`
  - `geom_boxplot()`
- Line plot
  - `geom_line` draws a line through the x-y-coordinates
- `geom_smooth()` for smoother functions such as LOESS
- `geom_ribbon()` useful for confidence intervals around lines
- there are many more `geom` functions
- stats, scales, coord and facetting later

Bar chart

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

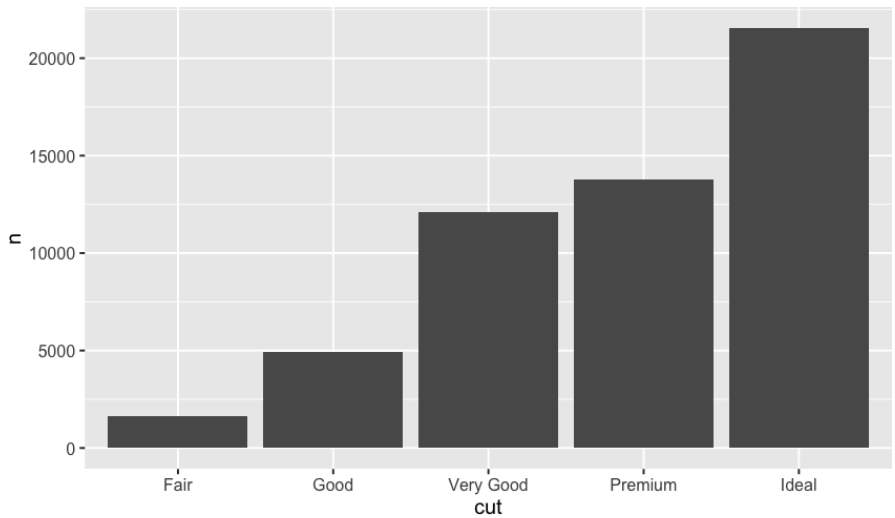


- `geom_bar()` does not need a `y`, `y` is defined by statistical transformation of `x`, simply the count of `x`

Bar chart

- You can also explicitly define a `y` defining the height and bars
- Set the `statistical` transformation to 'identity' (the default for most geoms)

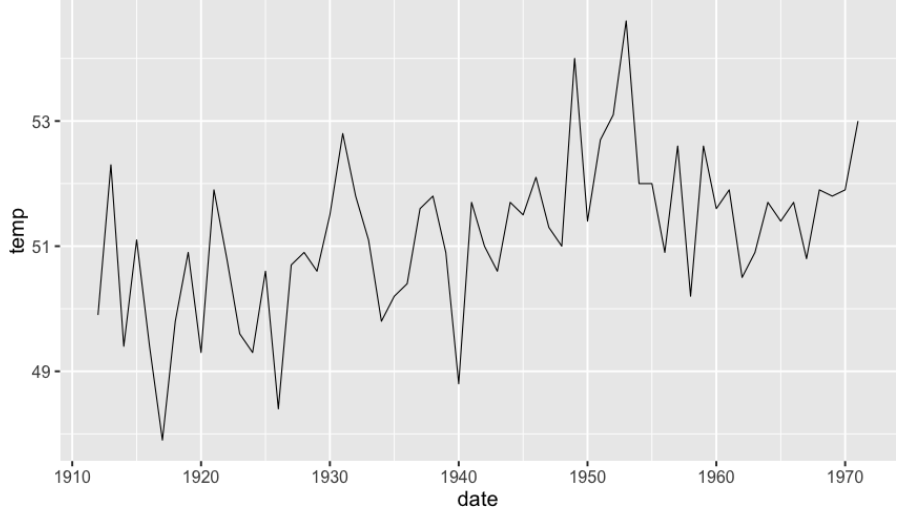
```
library(dplyr)
to_plot <- diamonds %>% group_by(cut) %>%
  summarise(n = n())
# plot
ggplot(to_plot, aes(x = cut, y = n)) +
  geom_bar(stat = "identity")
```



Line chart I

- Line charts are very helpful for plotting temporal developments

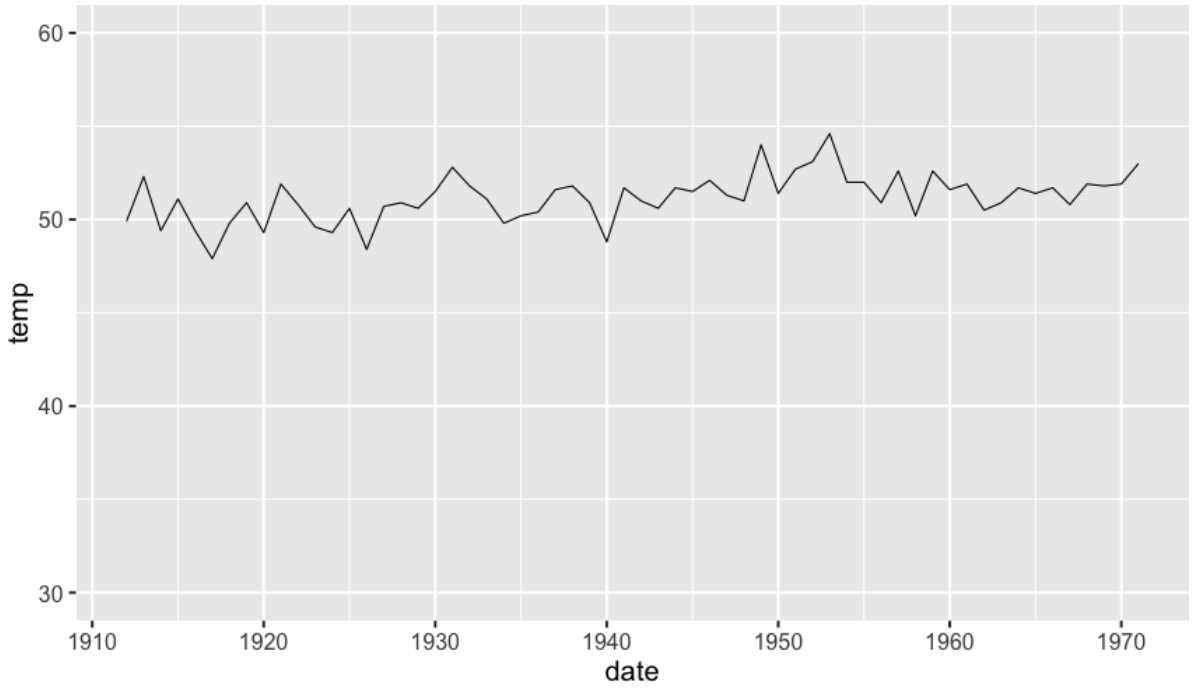
```
nhtemp <- as.data.frame(nhtemp) %>%
  mutate(temp = x, date = seq(1912,1971))

ggplot(nhtemp, aes(x = date, y = temp)) + geom_line()
```



Line chart II

- Scaling of the axis or of the group, color, ... 'dimension' (= y in this case) can be done with the requisite functions

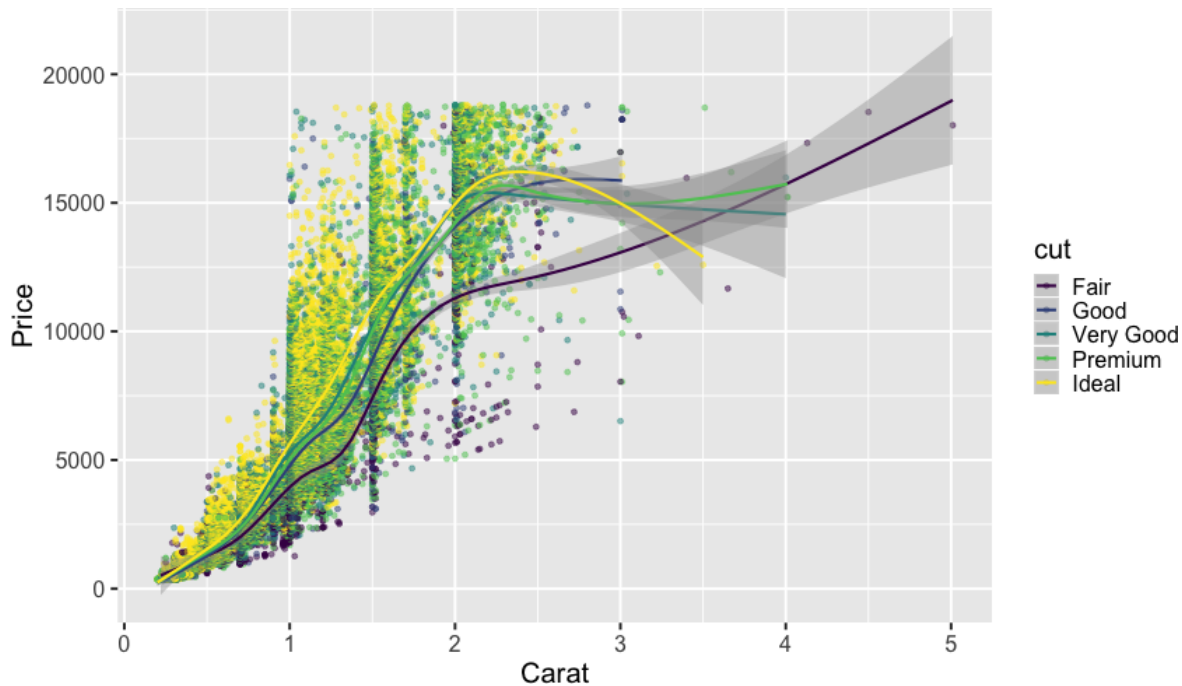
```
ggplot(nhtemp, aes(x = date, y = temp)) +
  geom_line() +
  scale_y_continuous(limits = c(30, 60))
```



Layers

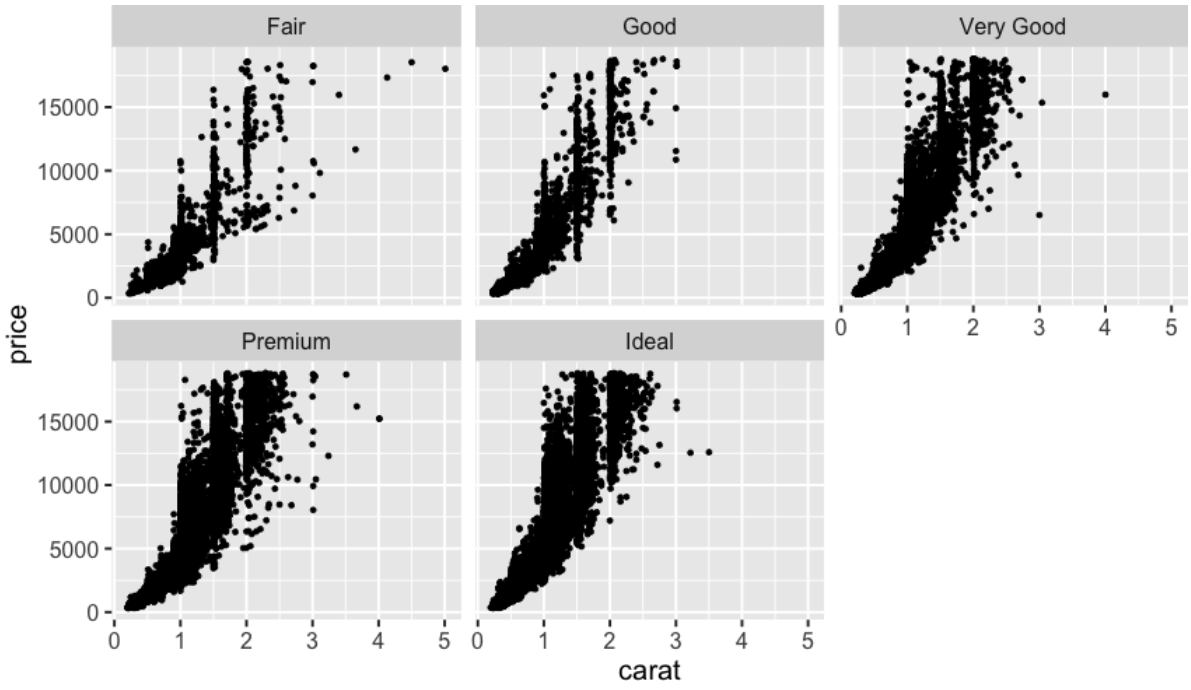
- We can overlay lots of plots one on top of one another.
- We just add a new function and precede it with a +.
- We add a loess curve with `geom_smooth()`

```
ggplot(diamonds, aes(x = carat, y = price, color = cut)) +
  geom_point(alpha = .5) +
  geom_smooth() +
  xlab('Carat') + ylab('Price')
```



Facetting

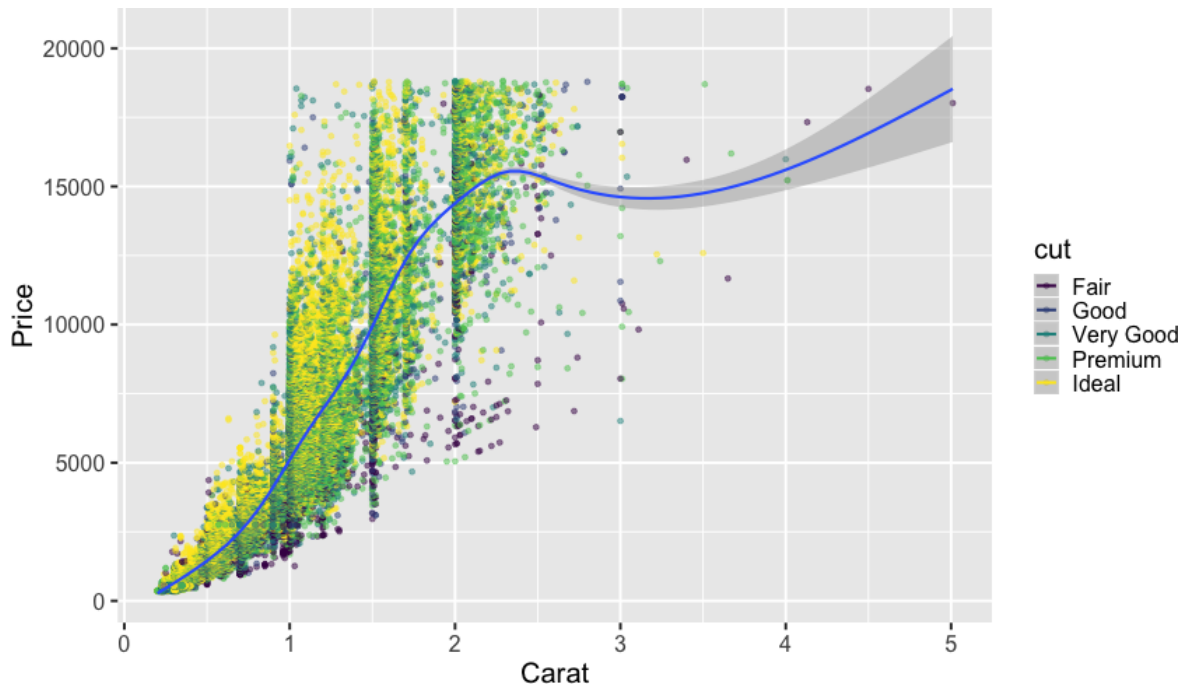
```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point() +
  facet_wrap(~cut)
```



Aesthetics

- What if we only want to have one loess curve based on the whole data?
- Simply override the aesthetic in `geom_smooth()`

```
ggplot(diamonds, aes(x = carat, y = price, color = cut)) +
  geom_point(alpha = .5) +
  geom_smooth(aes(color = NULL)) +
  xlab('Carat') + ylab('Price')
```



Data visualization II

Figures as objects

- ggplot figures can be saved as objects

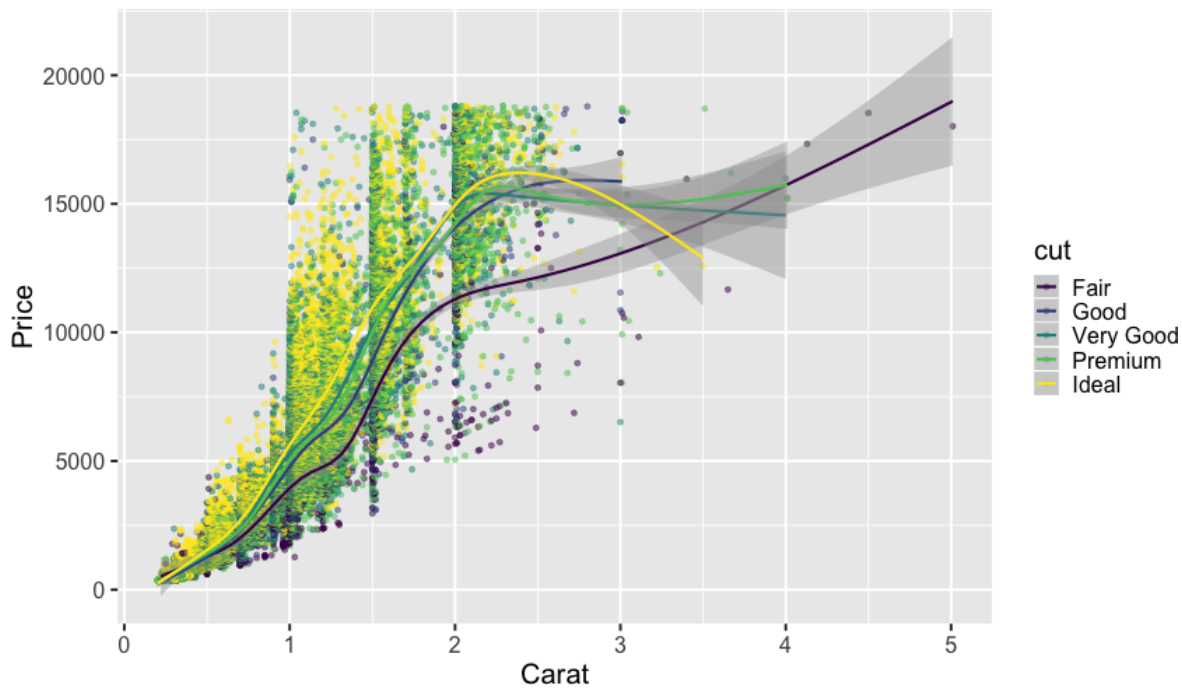
```
f <- ggplot(diamonds, aes(x = cut)) + geom_bar()
```

- These objects are self-contained. They are a list object containing:
  - the data
  - aesthetics
  - stats
  - scales
- If you chage the original data and redraw the plot object, the plot will not change

Layers

- You can also save a figure and build on it by adding further ggplot2 functions

```
f <- ggplot(diamonds
       , aes(x = carat, y = price, color = cut)) +
       geom_point(alpha = .5)

f + geom_smooth() + xlab('Carat') + ylab('Price')
```



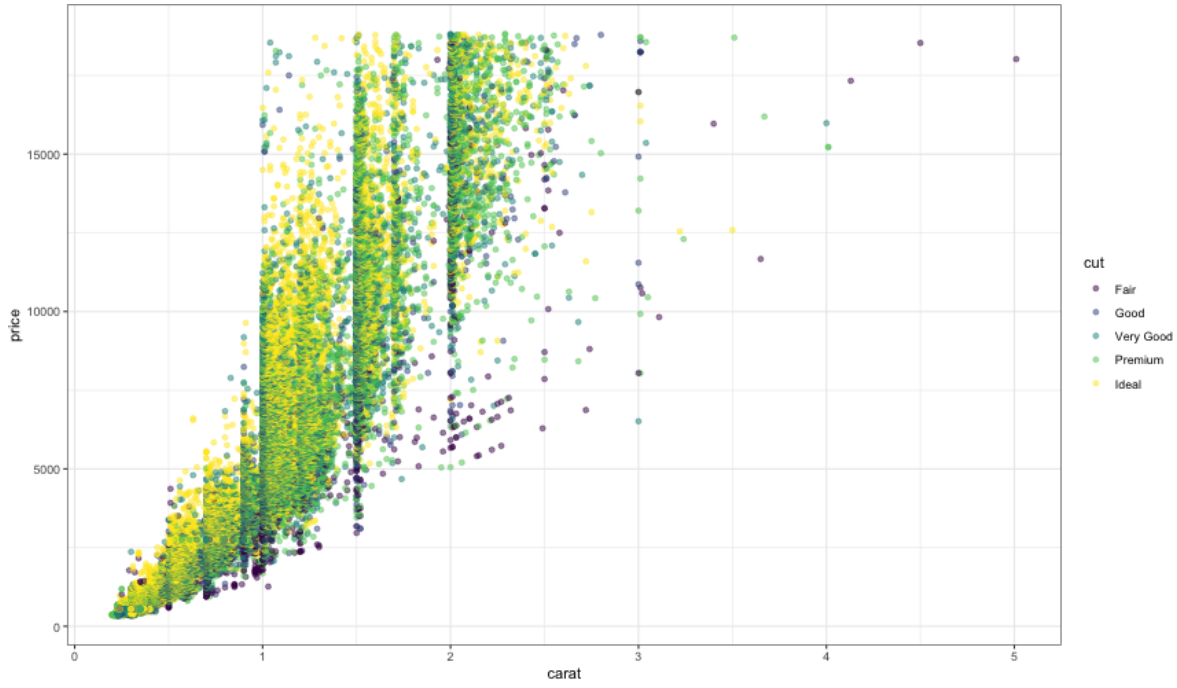
Themes

- Various themes are available
- For instance:
  - `theme_bw()`
  - `theme_minimal()`
  - `theme_light()`
- Further themes are available in the package `ggthemes`:
  - `theme_tufte` is minimal and elegant
  - there is also an Excel theme (`theme_excel`)

```
install.packages("ggthemes")
```

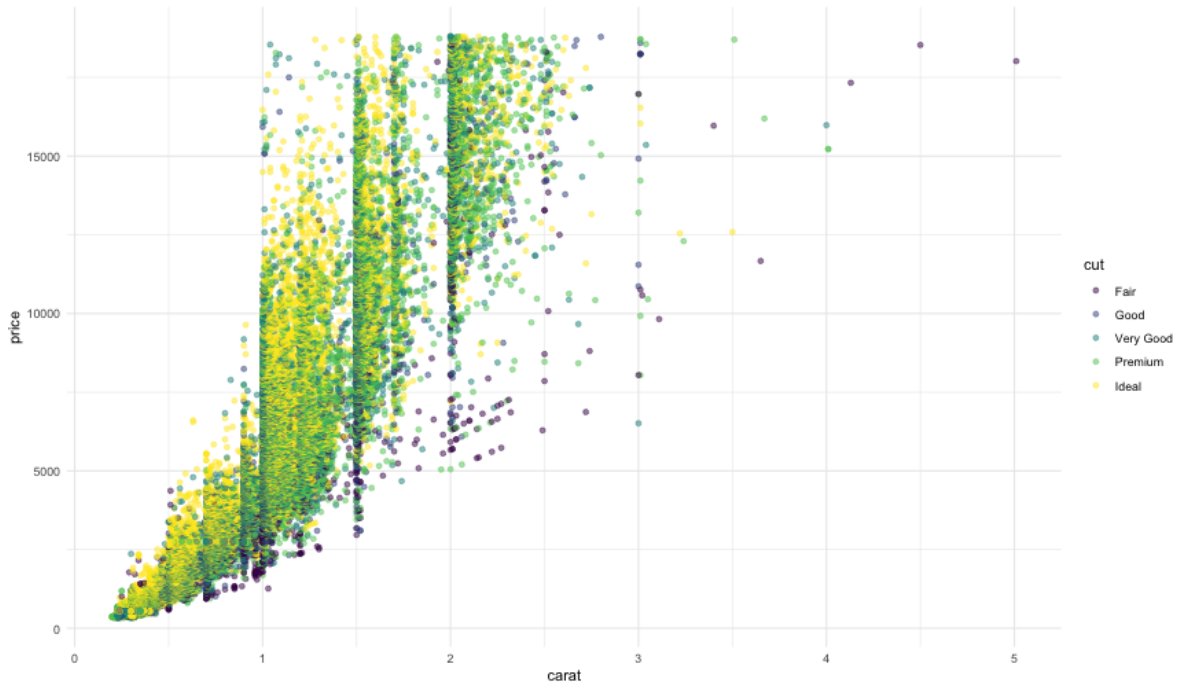
Themes

```
library(ggthemes)

f + theme_bw()
```



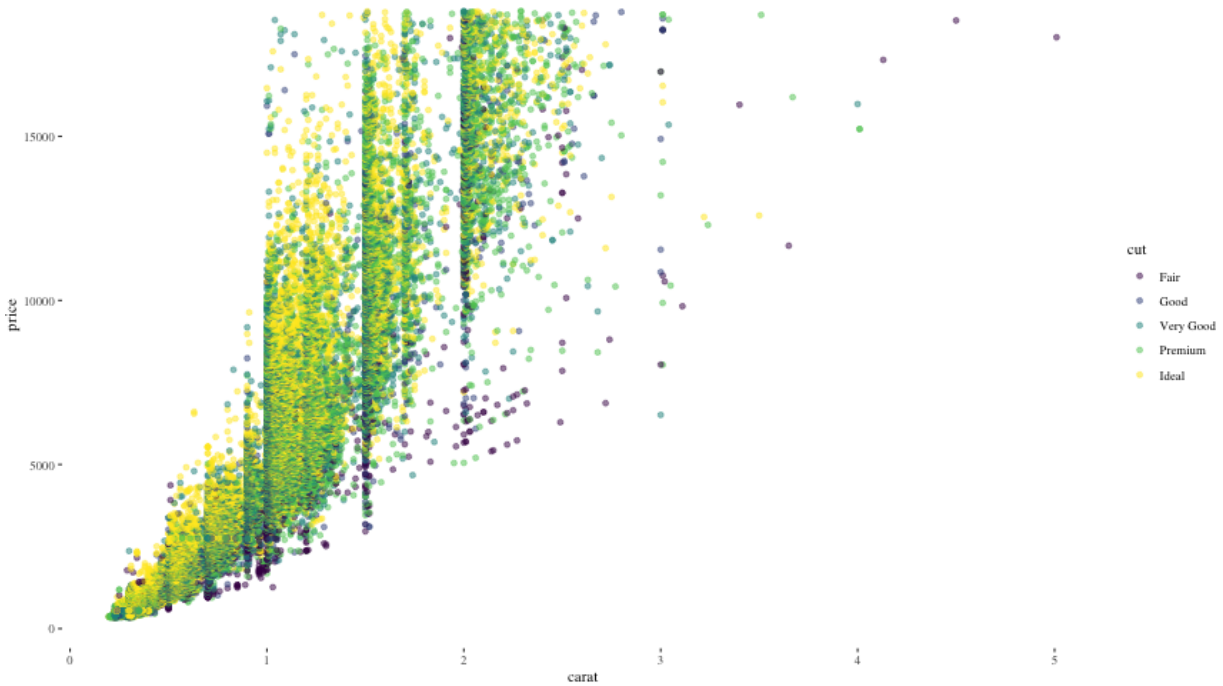
Themes

```
f + theme_minimal()
```



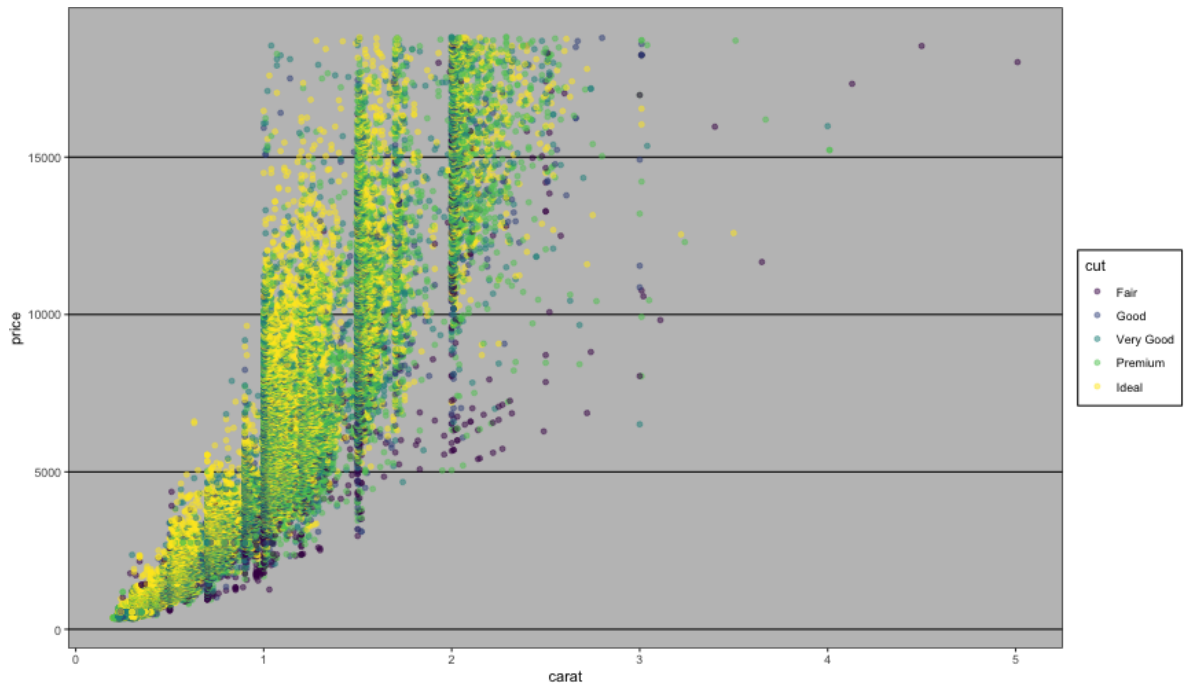
Themes

```
f + theme_tufte()
```



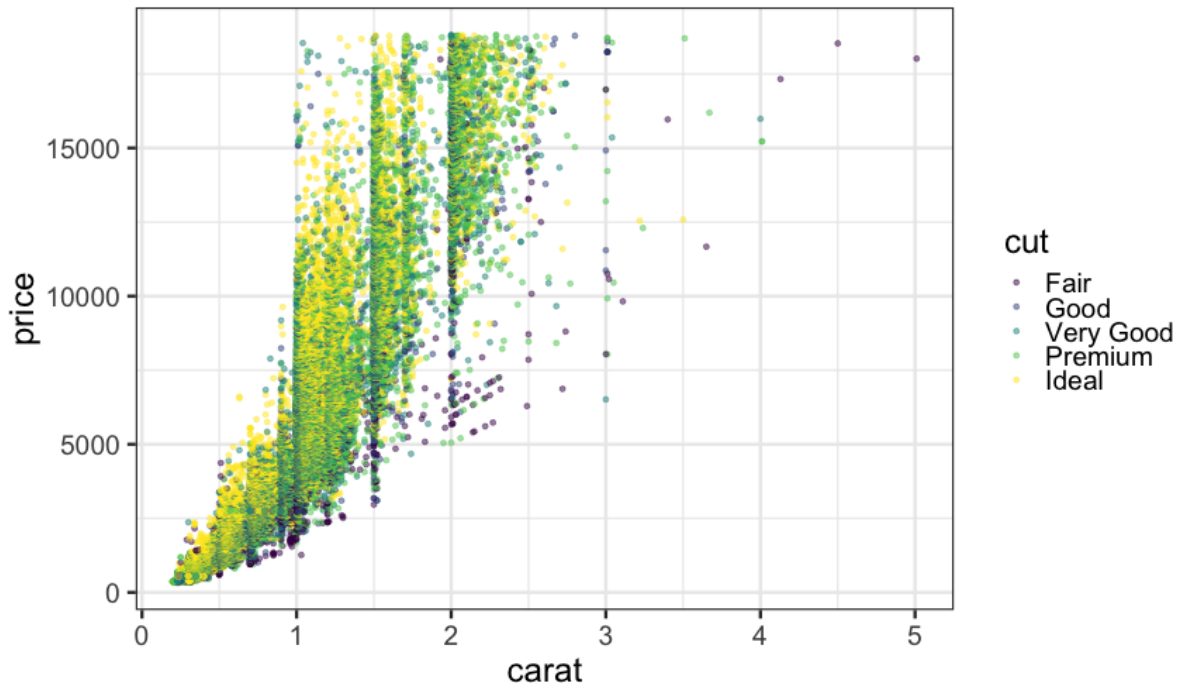
Themes

```
f + theme_excel()
```



Font size

```
f + theme_bw(base_size = 24)
```



Adding labels to a plot

```r
library(ggrepel)
set.seed(42)

dat <- subset(mtcars, wt > 2.75 & wt < 3.45)
dat$car <- rownames(dat)

p <- ggplot(dat, aes(wt, mpg, label = car)) +
  geom_point(color = "red")

# Standard ggplot2
p1 <- p + geom_text() +
  labs(title = "geom_text()")
# ggrepel
p2 <- p + geom_text_repel() +
  labs(title = "geom_text_repel()")
```

Adding labels to a plot

```
gridExtra::grid.arrange(p1, p2, ncol = 2)
```

Saving graphics

Of course plots can be saved to objects as well as to the hard drive.

```
fig <- ggplot(diamonds, aes(x = price, y = carat)) +
  geom_point() +   xlab('Price') + ylab('Carat') +
  theme_bw()
pdf('figures/fig.pdf')
fig
dev.off()
```

- or `png()`, `jpeg()`, `tiff()` followed by plot function or object and dev.off()
- or `ggsave()`

```
ggsave(filename = "figures/fig.pdf", plot = fig)
```

Saving graphics

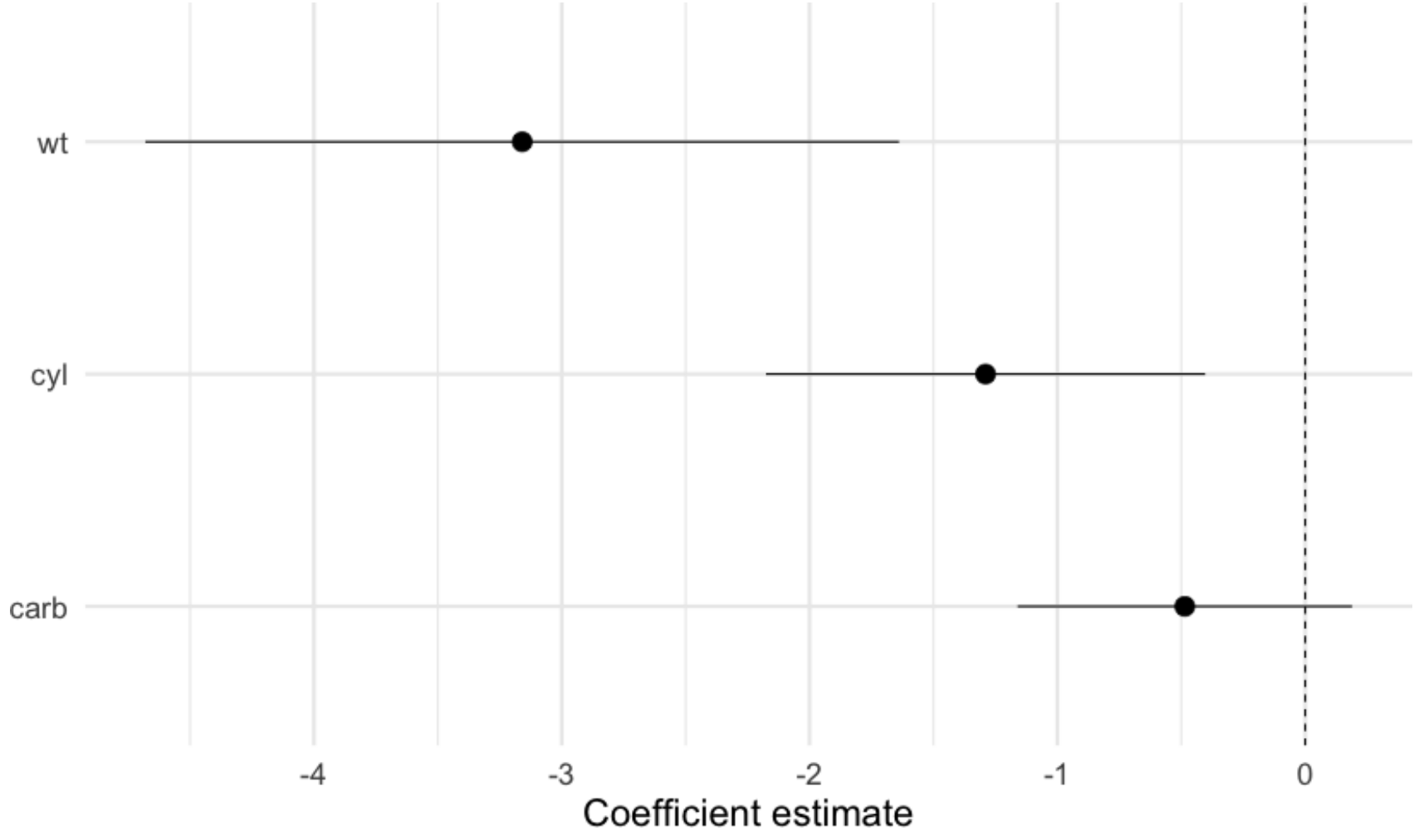
```
# set dimensions in inches for pdf
pdf('figures/fig.pdf', width = 5, height = 4)
fig
dev.off()
# and in pixels for png
png('figures/fig.png', width = 500, height = 400)
fig
dev.off()
```

Model visualization

Coefficient plot

```r
# Create a model to plot
m1 <- lm(mpg ~ wt + cyl + carb, data=mtcars)

coefs <- data.frame(names(coef(m1)), coef(m1), confint(m1))
names(coefs) <- c('var', 'coef', 'lwr', 'upr')
coefs <- coefs[-1,]

ggplot(coefs, aes(var, coef)) +
  geom_hline(yintercept = 0, linetype = 'dashed') +
  geom_point(size = 5) +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0) +
  xlab('') + ylab('Coefficient estimate') +
    coord_flip() + theme_minimal(base_size = 22)
```

Coefficient plot



Coefficient plot with broom

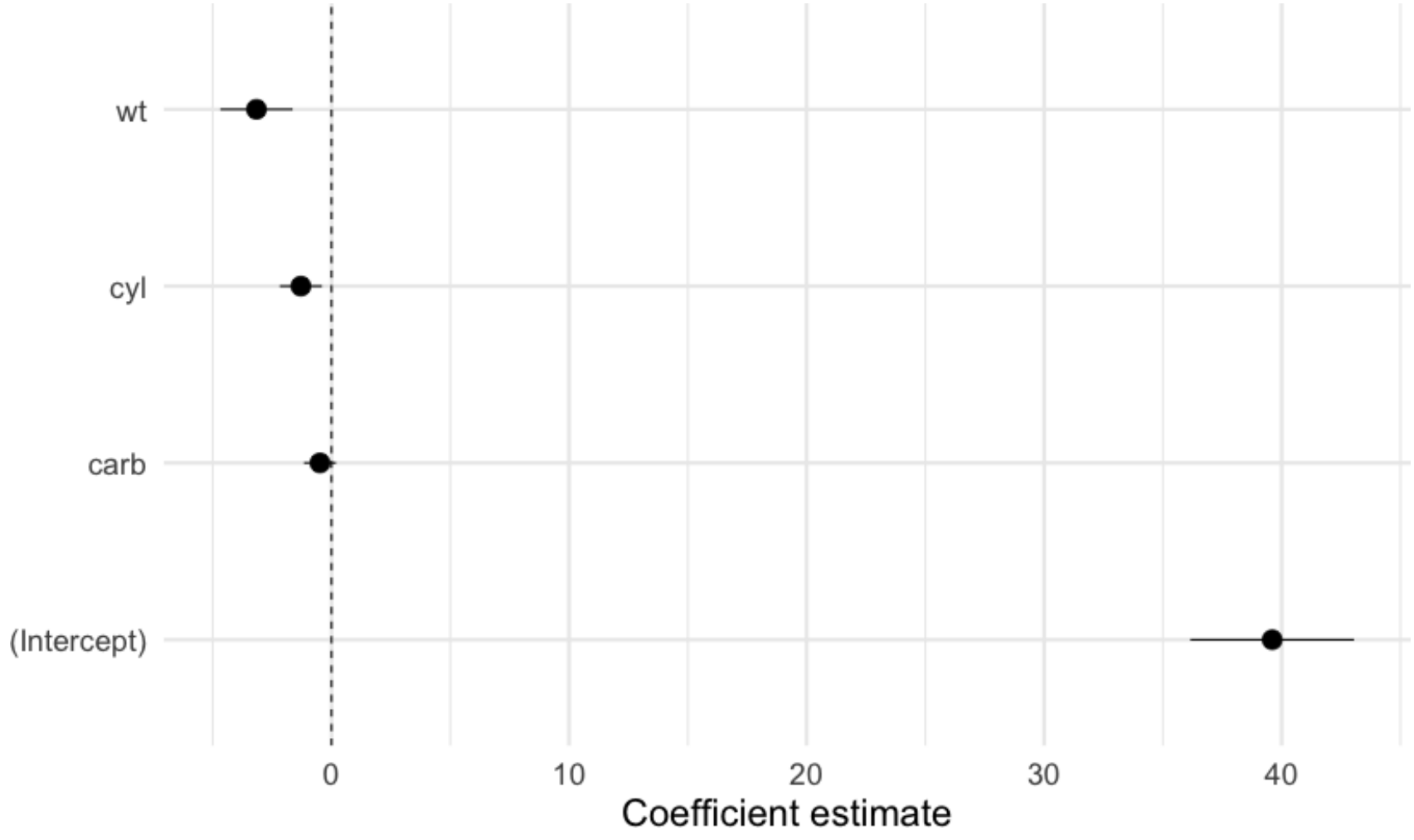
```
library(broom)

coefs <- tidy(m1)
coefs <- cbind(coefs,
               lwr = confint(m1)[,1],
               upr = confint(m1)[,2])
coefs
```

```
##                    term   estimate std.error statistic
## (Intercept) (Intercept) 39.6021403 1.6822639 23.540979
## wt                 wt -3.1594517 0.7423463 -4.256035
## cyl               cyl -1.2897877 0.4325975 -2.981496
## carb             carb -0.4857629 0.3294704 -1.474375
##                    p.value       lwr         upr
## (Intercept) 5.418679e-20 36.156179 43.0481018
## wt          2.107662e-04 -4.680079 -1.6388243
## cyl         5.880227e-03 -2.175923 -0.4036518
## carb        1.515365e-01 -1.160652  0.1891267
```

Coefficient plot with `broom` I

```
ggplot(coefs
       , aes(x = term, y = estimate, ymin = lwr, ymax = upr
  geom_hline(yintercept = 0, linetype = 'dashed') +
  geom_point(size = 5) +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0) +
  xlab('') + ylab('Coefficient estimate') +
    coord_flip() + theme_minimal(base_size = 22)
```

Coefficient plot with `broom` II

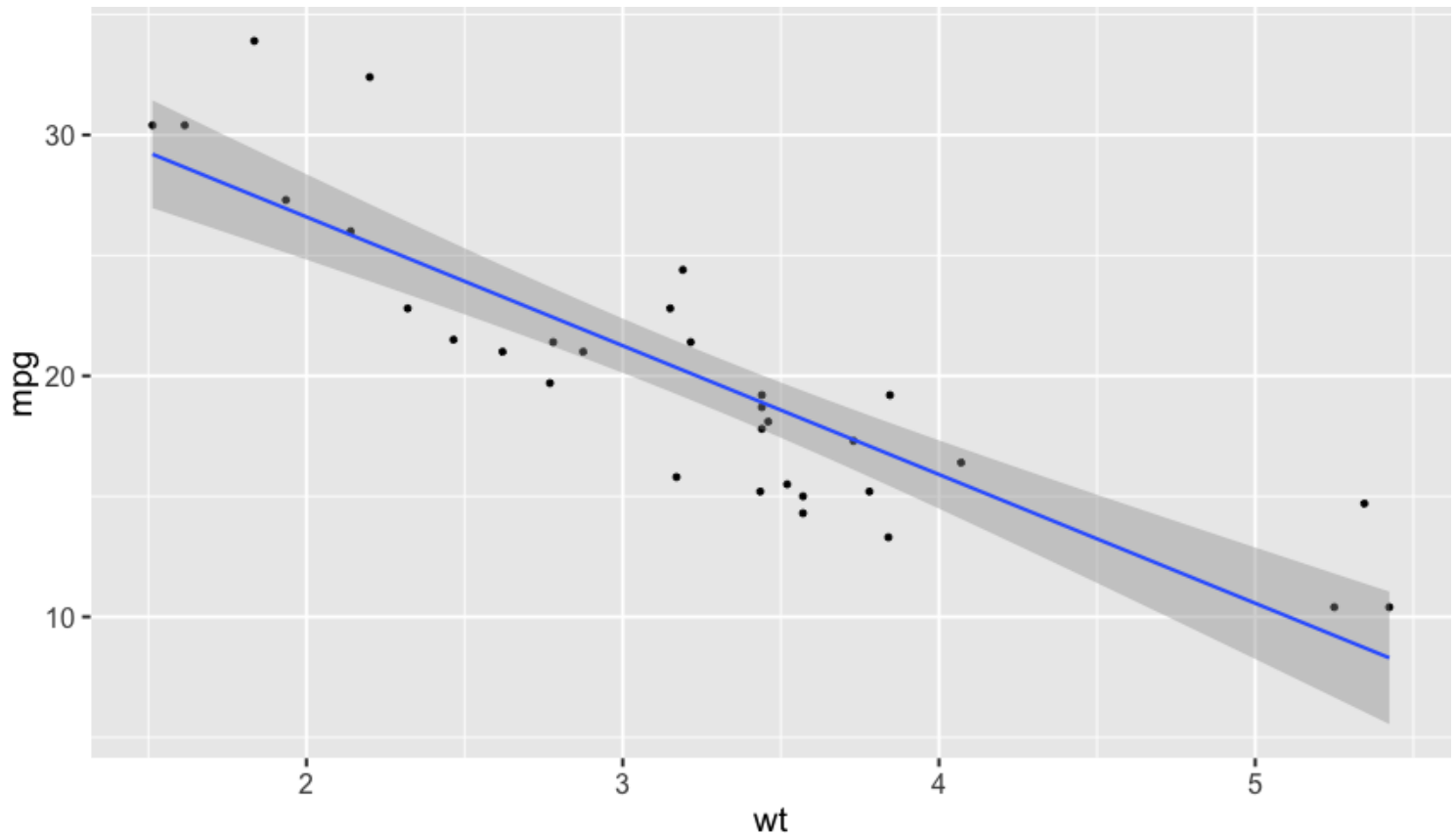


Appendix

Predicted values: `lm()`

- For a bivariate model it's simple
- Just use `geom_smooth(method = 'lm')`

```
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_smooth(method = 'lm')
```



Predicted values: `lm()`

```r
# Recall m1
m1 <- lm(mpg ~ wt + cyl + carb, data=mtcars)

tmp <- data.frame(wt = seq(min(mtcars$wt),
                           max(mtcars$wt), .1),
                  cyl = mean(mtcars$cyl),
                  carb = mean(mtcars$carb))

tmp$mpg_hat <- predict(m1, newdata = tmp)
tmp$lwr <- tmp$mpg_hat - 1.96 *
  predict(m1, newdata = tmp, se.fit = T)$se.fit
tmp$upr <- tmp$mpg_hat + 1.96 *
  predict(m1, newdata = tmp, se.fit = T)$se.fit
```

Predicted values: `lm()`

```
ggplot(tmp, aes(x = wt, y = mpg_hat)) +
  geom_ribbon(aes(ymin = lwr, ymax = upr),
              fill = 'maroon3', alpha = .5) +
  geom_line(color = 'maroon3') +
  theme_tufte(base_size = 22)
```