Visualizing data

ADS2 week 9

Dmytro Shytikov (adapted from Chaochen Wang`s slides)

2023-11-13

dmytroshytikov@intl.zju.edu.cn

Lecture outline

- MAKING GRAPHS IN graphics
- MAKING GRAPHS IN ggplot2
- WORKING IN ggplot2
- DEALING WITH SEVERAL GROUPS ON THE SAME GRAPH IN ggplot2
- OTHER IMPORTANT POINTS
- MAKING GRAPHS IN lattice

Learning objectives

- Introduce the R key visualization tools: `graphics` and `ggplot2` packages

- Discuss data visualization choices

Data analysis workflow

1. **Present data**

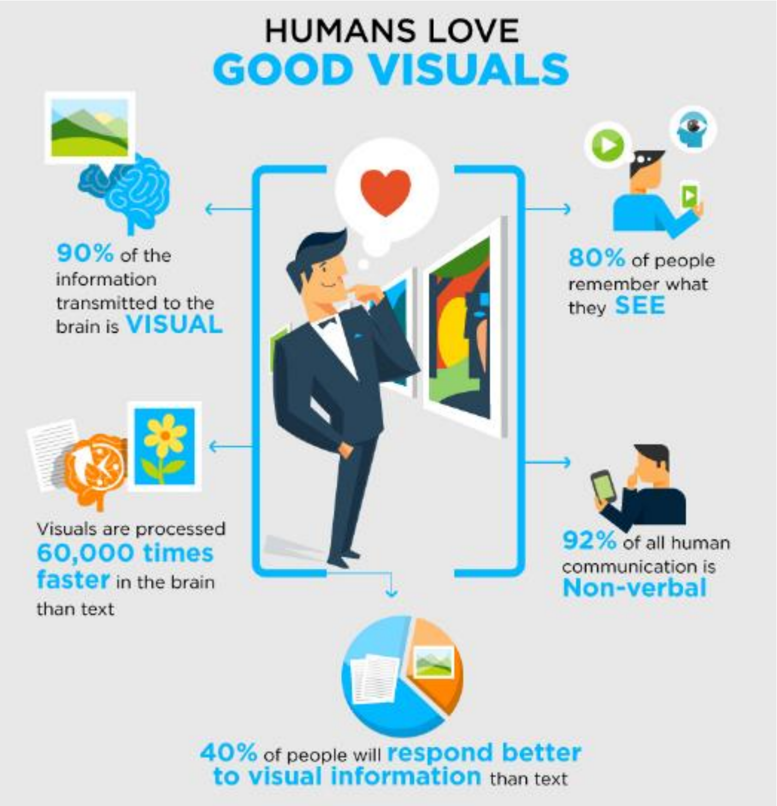
- Straightforward
- Present large data sets in a limited space

2. **Provide more information**

- Counts, Distribution, Trends, Irregularities…

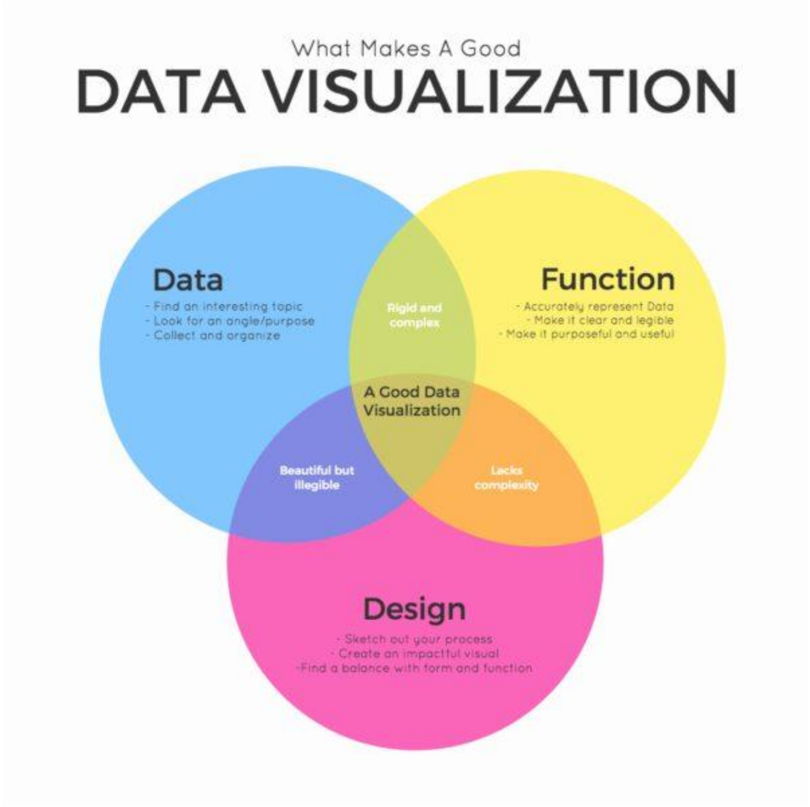
3. **Tell a story**

- Relationships among data
- Help find interesting regions
- Help make decisions


HUMANS LOVE
GOOD VISUALS
90% of the information transmitted to the brain is VISUAL
80% of people remember what they SEE
Visuals are processed 60,000 times faster in the brain than text
92% of all human communication is Non-verbal
40% of people will respond better to visual information than text

https://www.infographicdesignteam.com/blog/data-visualization-best-practices/

Good data visualization


What Makes A Good
DATA VISUALIZATION
Data
- Find an interesting topic
- Look for an angle/purpose
- Collect and organize
Function
- Accurately represent Data
- Make it clear and legible
- Make it purposeful and useful
Rigid and complex
A Good Data Visualization
Beautiful but illegible
Lacks complexity
Design
- Sketch out your process
- Create an impactful visual
-Find a balance with form and function

https://hiilite.com/information-visualization/

General comments about data visualization

- Make the design of your data visualization fit the data, not the other way around
- Don’t manipulate the data to make it fit your argument
- Cite the sources of your data
- Tell a story from the data
- Make your data clear and readbable!

Plotting systems in R

`graphics`

`ggplot2`

`lattice`

`plot3D`

MAKING GRAPHS IN `graphics`

Using the `graphics` package to create plots: `ToothGrowth` dataset

Plotting graphs step by step:

1. Arrange your data;
2. Set graphic parameters using `par()`;

```
head(ToothGrowth)
   len supp dose
1  4.2   VC  0.5
2 11.5   VC  0.5
3  7.3   VC  0.5
4  5.8   VC  0.5
5  6.4   VC  0.5
6 10.0   VC  0.5
par("bty" = "l") # Sets a different shape to the box around the plot
par("mai" = c(0.6, 0.75, 0.4, 0.4)) # Change the graph margins
```

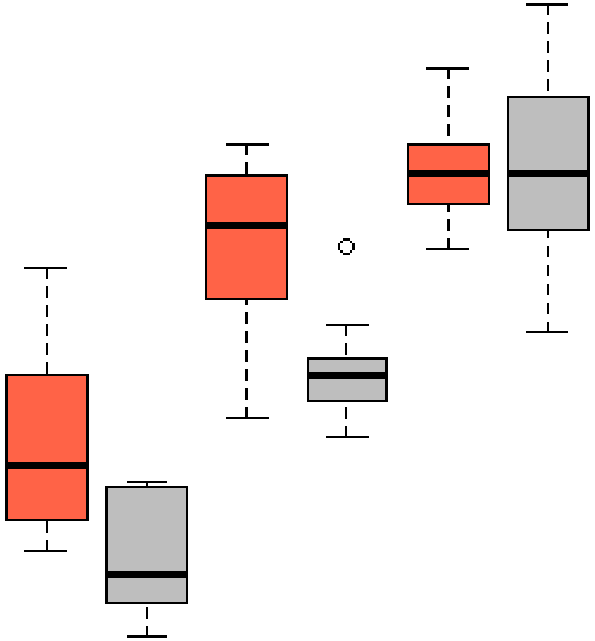
Using the `graphics` package to create plots: `ToothGrowth` dataset

Plotting graphs step by step:

1. Arrange your data;
2. Set graphic parameters using `par()`;
3. Call the major plotting function:

```
Histograms – hist(x, …)
Scatter plots – plot(x, y, …)
Bar plots – barplot(x, y, …)
Pie charts – pie(data, …)
Box plots – boxplot(data, …)
```

```
boxplot(len ~ supp*dose,
        data = ToothGrowth,
        col = c("tomato", "grey")
        axes = F, xlab = "", ylab = "")
```



Using the `graphics` package to create plots: `ToothGrowth` dataset

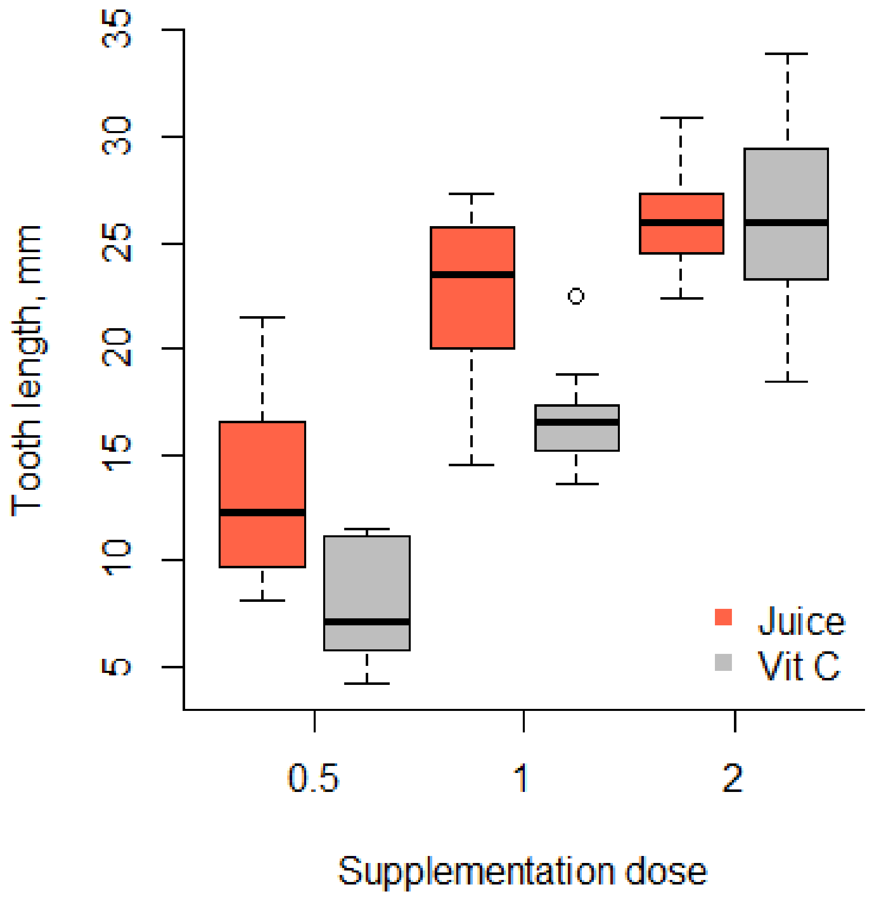
Plotting graphs step by step

1. Arrange your data;
2. Set graphic parameters using `par()`;
3. Call the major plotting function:

```
Histograms – hist(x, ...)
Scatter plots – plot(x, y, ...)
...
```

4. Add additional objects to the graph:
   - `lines()`, regression slopes (`abline`);
   - `arrows()`, `points()`, `rect()`, etc.
5. Adjust axes and add annotations (if needed):

```
axis(), title(), text(),
mtext(), legend(), etc
```

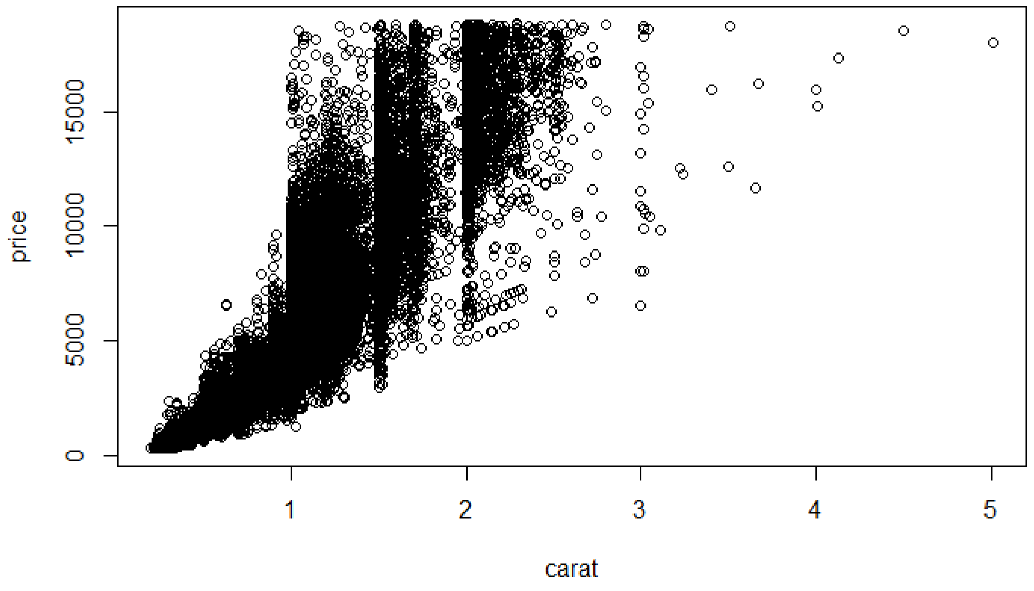


Using the `graphics` package to create plots: `diamonds` dataset

```
head(diamonds, 7)
# A tibble: 7 × 10
  carat cut       color clarity depth table price     x     y     z
  <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
dim(diamonds)
[1] 53940    10
```

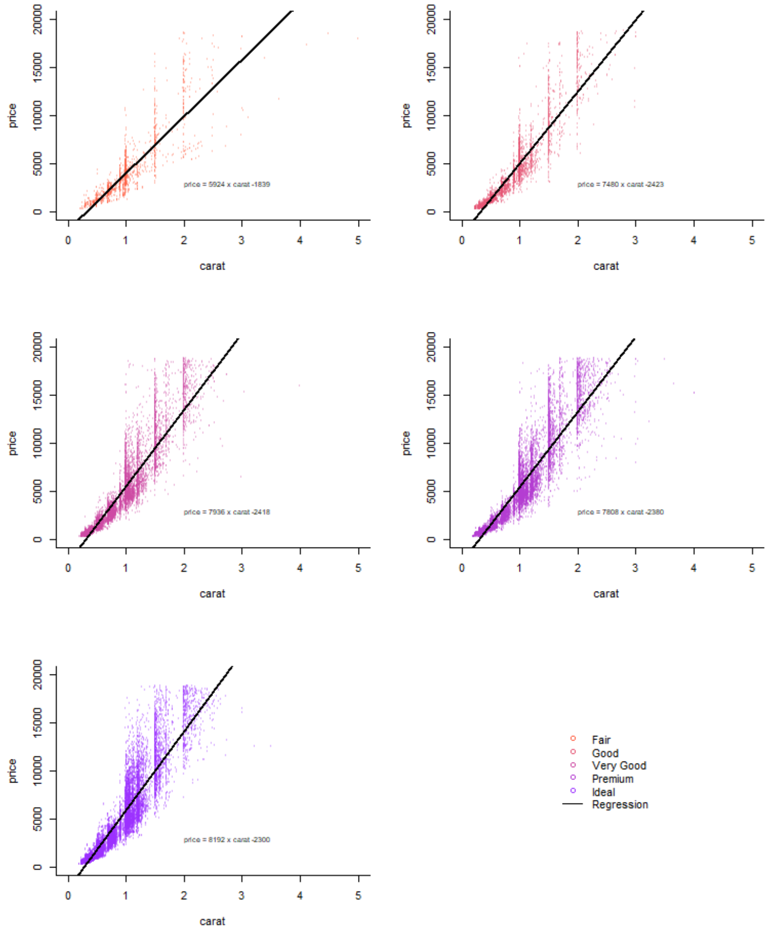
Using the `graphics` package to create plots: `diamonds` dataset

```
plot(price ~ carat, data = diamonds)
```



Using the `graphics` package to create plots: `diamonds` dataset

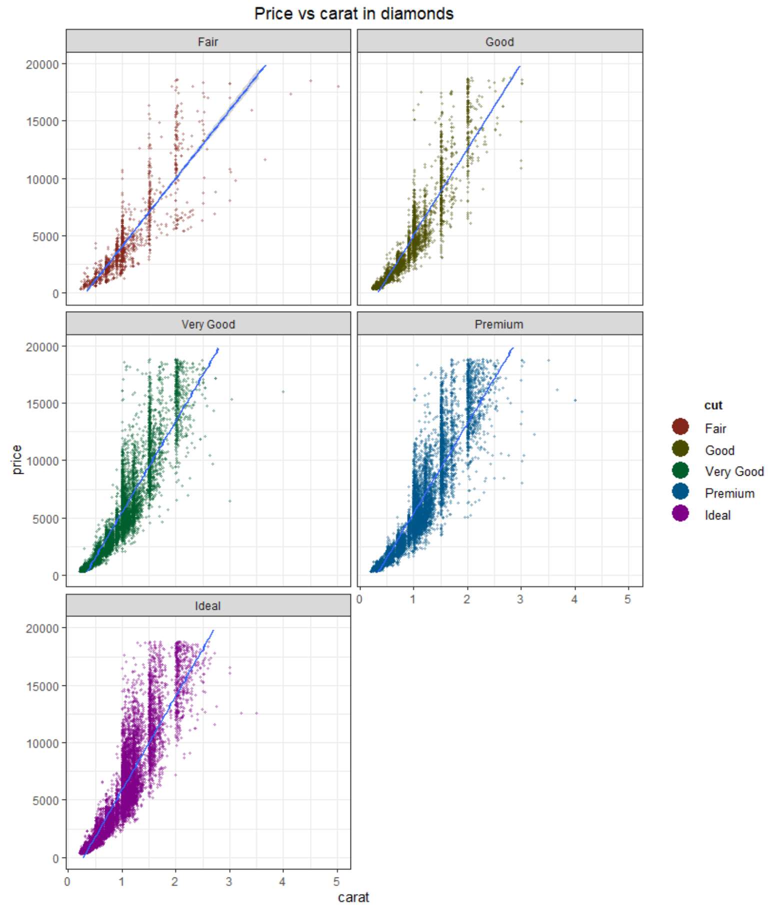
```
par("mfrow" = c(3, 2))
par("oma" = c(1,1,2,1))

new.palette <-
colorRampPalette(c("color.1",
"color.2"))(5)
palette(new.palette)

for(i in 1:length(levels(diamonds$cut))){
  datatoplot <- ... # Choose data
  plot(...) # Produce plot
  model_diamonds <- lm(...)
  abline(...) # Add additonals
  text(...) # Add annotation
}
plot.new() # Arrange the legend
legend(...)
```



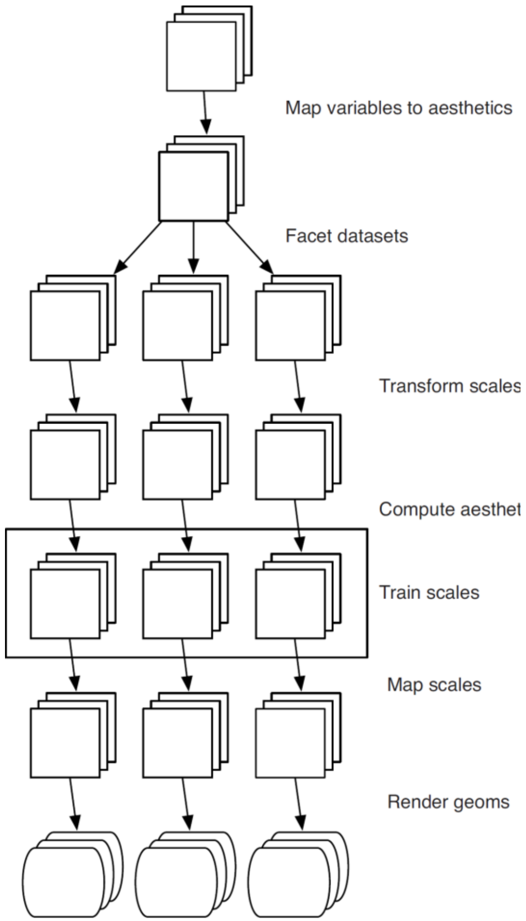
DOING GRAPHS IN `ggplot2`

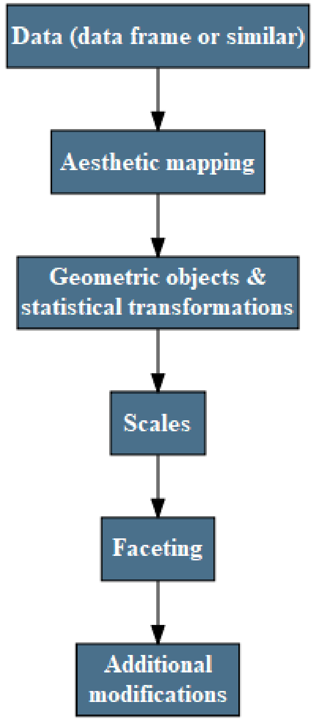
Using the `ggplot2` package to create plots: `diamonds` dataset

```
ggplot(data=diamonds,
       mapping =  aes(...)) +
  geom_point(stat = "identity",
             mapping = aes(...)) +
  facet_wrap(~cut, ncol = 2) +
  scale_color_hue(l=30, c=70) +
  scale_y_continuous(limits = c(0,
20000)) +
  geom_smooth(method = "lm") +
  labs(title = "Price vs carat in
diamonds") +
  theme_bw() +
  theme(...) +
  guides(color =
guide_legend(override.aes = ...))
```



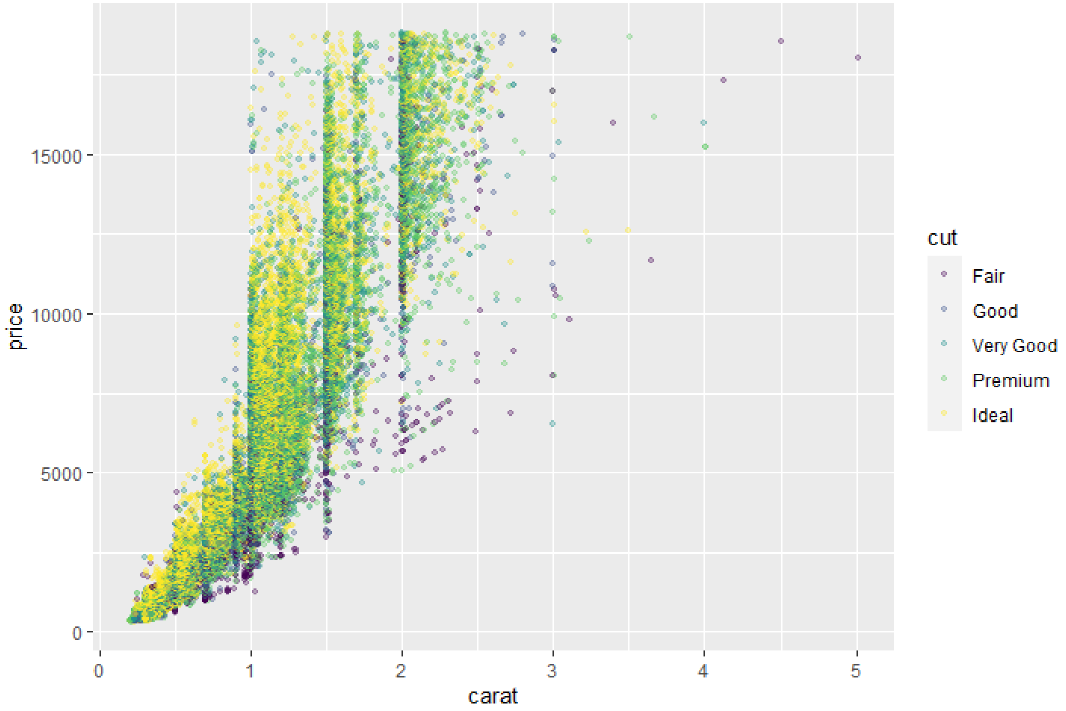
ggplot2 – layered grammar




Data (data frame or similar)
Aesthetic mapping
Geometric objects & statistical transformations
Scales
Faceting
Additional modifications

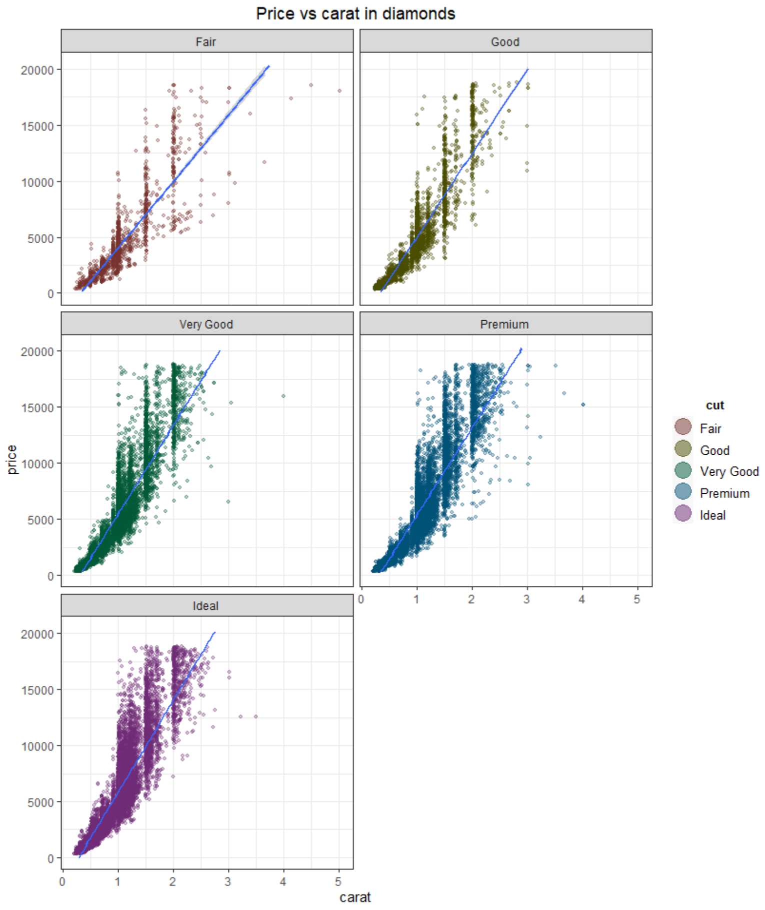
ggplot2 – adding layer by layer

```
g <- ggplot(data = diamonds, mapping =
aes(x = carat, y = price, group = cut))


g1 <- g + geom_point(stat = "identity",
aes(colour = cut), size = 1, alpha =
0.3)
```



ggplot2 – adding layer by layer

```
g <- ggplot(data=diamonds,
              mapping =  aes(...))
g1 <- g + geom_point(stat = "identity",
                       mapping = aes(...))
g2 <- g1 + facet_wrap(~cut, ncol = 2)
g3 <- g2 + scale_color_hue(l=30, c=70)
g4 <- g3 + scale_y_continuous(limits =
c(...))
g5 <- g4 + geom_smooth(method = "lm")
g6 <- g5 + labs(title = ...)
g7 <- g6 + theme_bw() + theme(...)

g8 <- g7 + guides(color =
guide_legend(override.aes = ...))
```



WORKING IN `ggplot2`

`ggplot` as an R object

Can be viewed by `summary()`

Can be saved (`save`) and loaded (`load`)

Data is stored inside the plot, so that if you change the data outside of the plot, and then redraw a saved plot, it will not be updated.

Geom can also be saved and applied to another `ggplot` object if the aesthetics still exist

```
summary(g8)

data: carat, cut, color, clarity, depth,
table, price, x, y, z

  [53940x10]

mapping:  x = ~carat, y = ~price, group =
~cut

scales:   y, ymin, ymax, yend,
yintercept, ymin_final, ymax_final,
lower, middle, upper, y0, colour

faceting: <ggproto object: Class
FacetWrap, Facet, gg>

    compute_layout: function

    draw_back: function

    draw_front: function

    draw_labels: function

    ...
```

Layers as building blocks

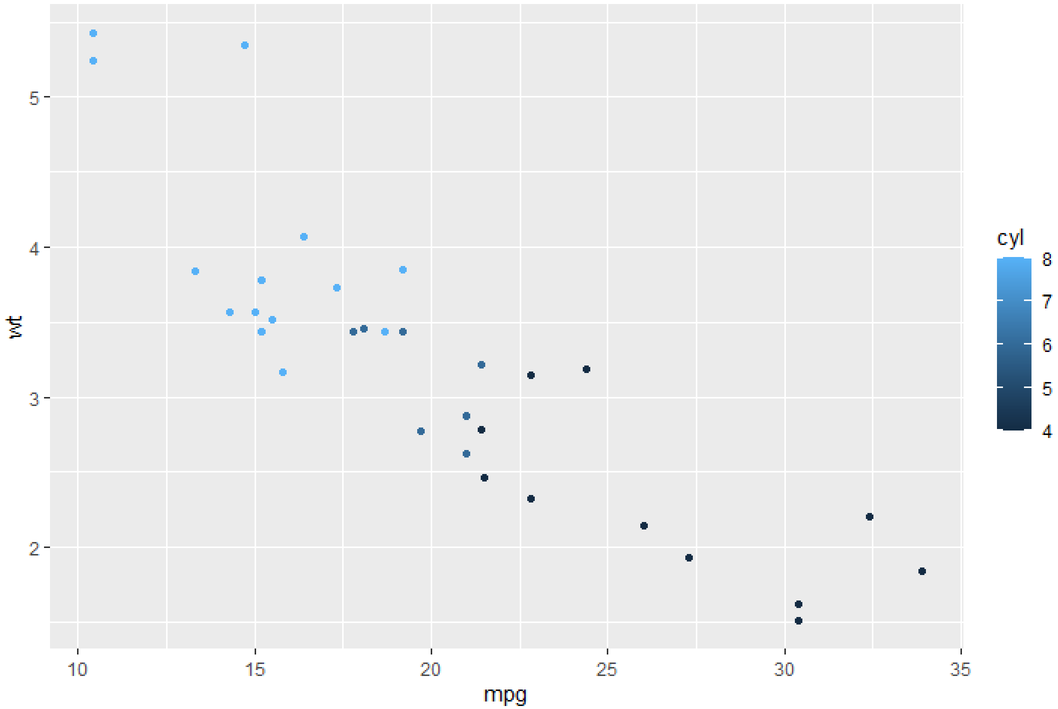
Layers are responsible for creating objects that we perceive on the plot. A layer is composed of four parts:

- data and aesthetic mapping,
- a statistical transformation (`stat`),
- a geometric object (`geom`)
- a position adjustment

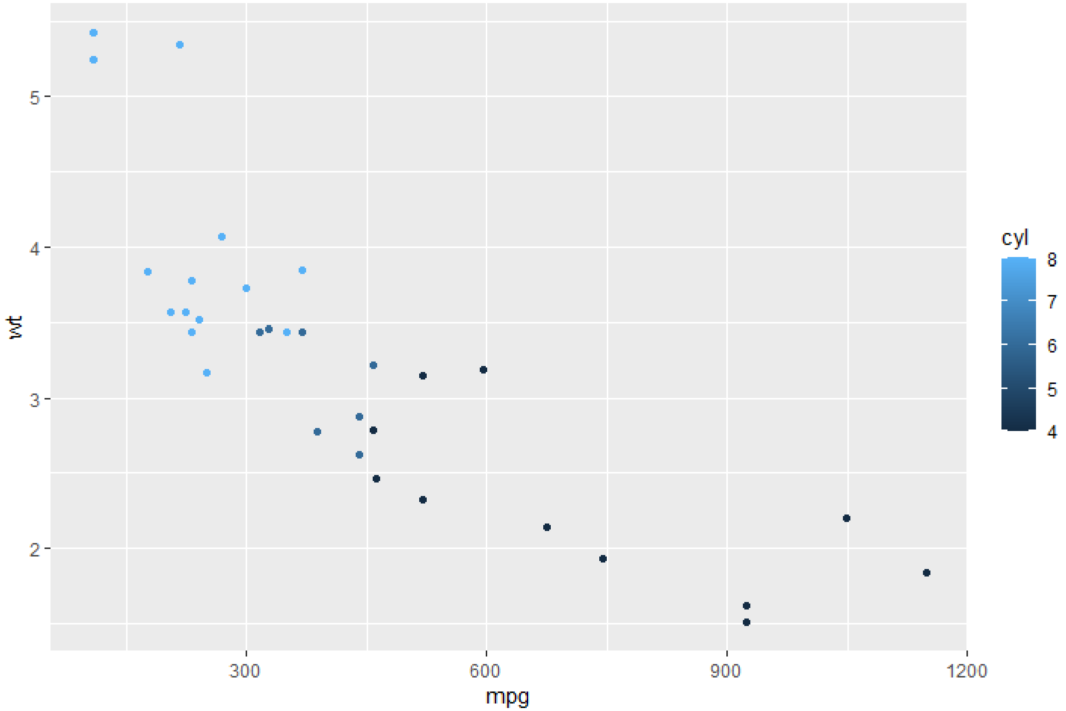
```
plot_cut <- ggplot(data = diamonds,
                   mapping = aes(x = cut, y = price, group = cut))

plot_cut + layer(
  geom = "boxplot",
  stat = "boxplot",
  position = "identity",
  mapping = NULL, data = NULL)

plot_cut + geom_boxplot()
```

Updating datasets

```
p <- ggplot(mtcars, aes(mpg, wt, colour
= cyl)) + geom_point()
p
```



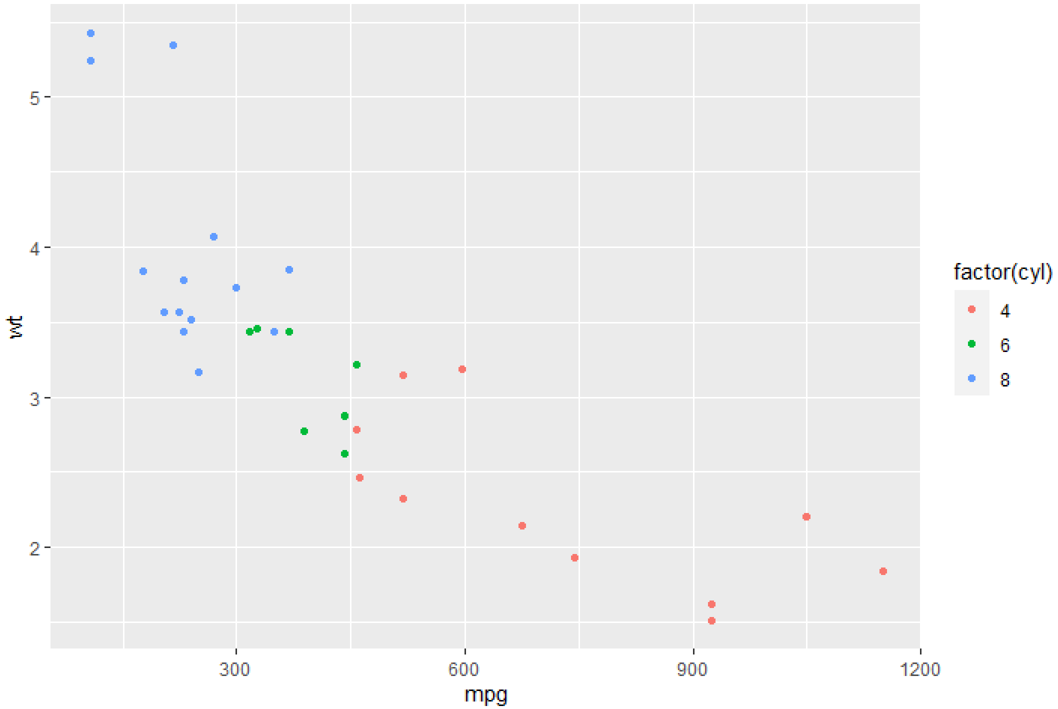
```
mtcars <- transform(mtcars, mpg = mpg ^ 2)

p %+% mtcars
```

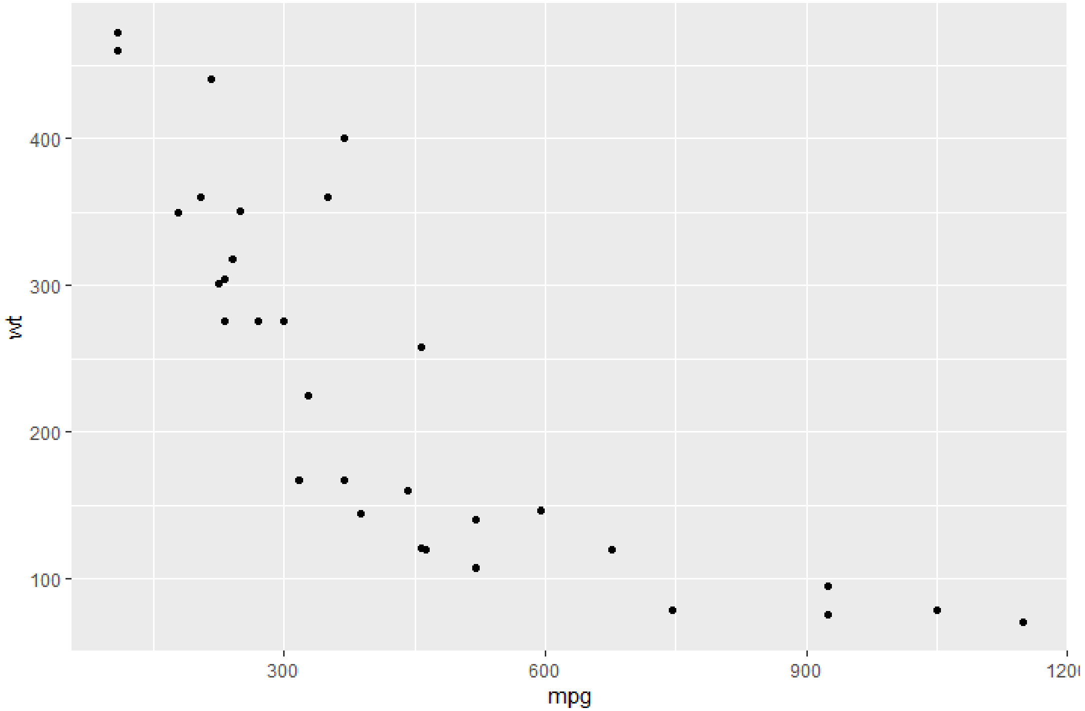


Aesthetic mapping

```
p <- ggplot(mtcars, aes(x = mpg, y = wt))
p + geom_point(aes(colour = factor(cyl)))
```

```
p + geom_point(aes(y = disp))
```

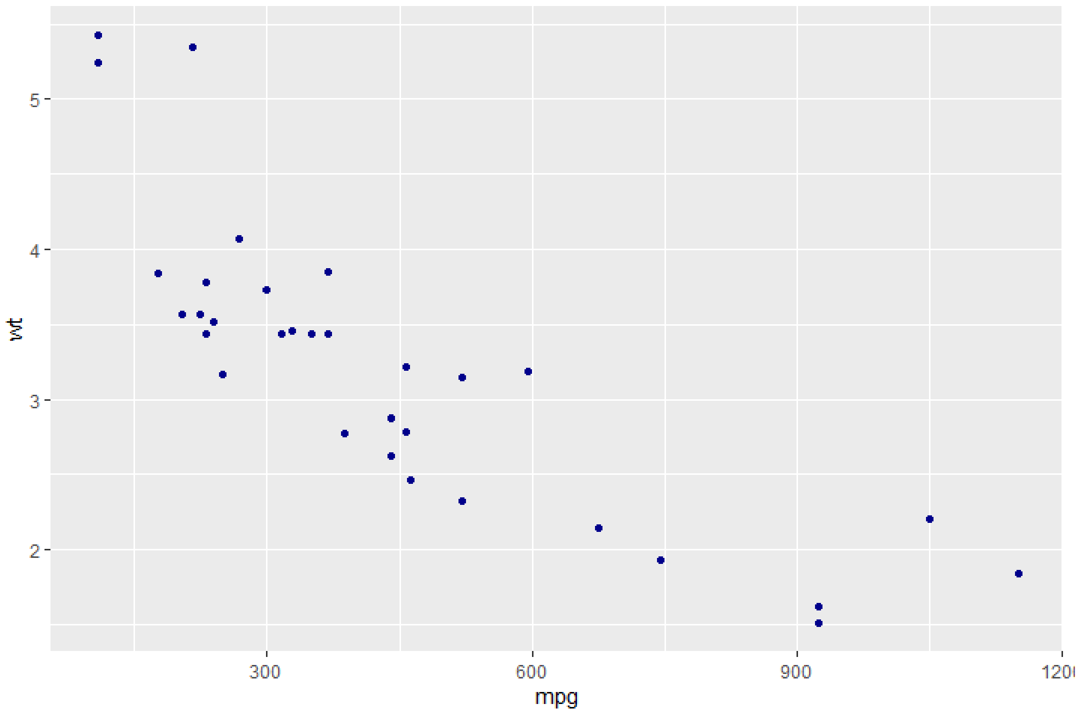




Setting vs. mapping

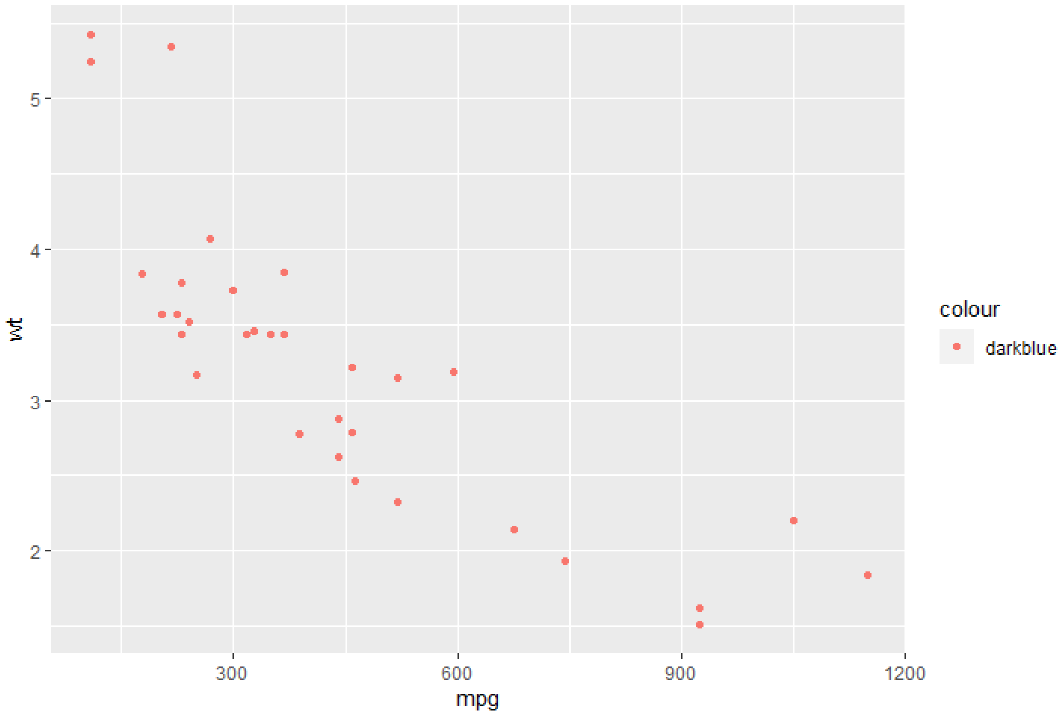
Setting parameters (color)

```
p <- ggplot(mtcars, aes(mpg, wt))
p + geom_point(colour = "darkblue")
```



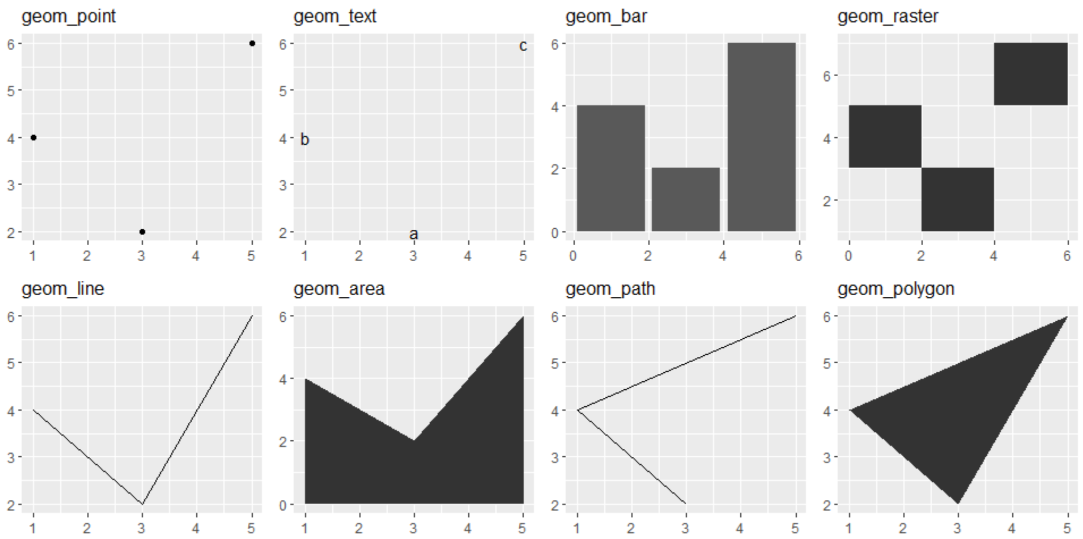
Creating a new variable

```
p + geom_point(aes(colour = "darkblue"))
```



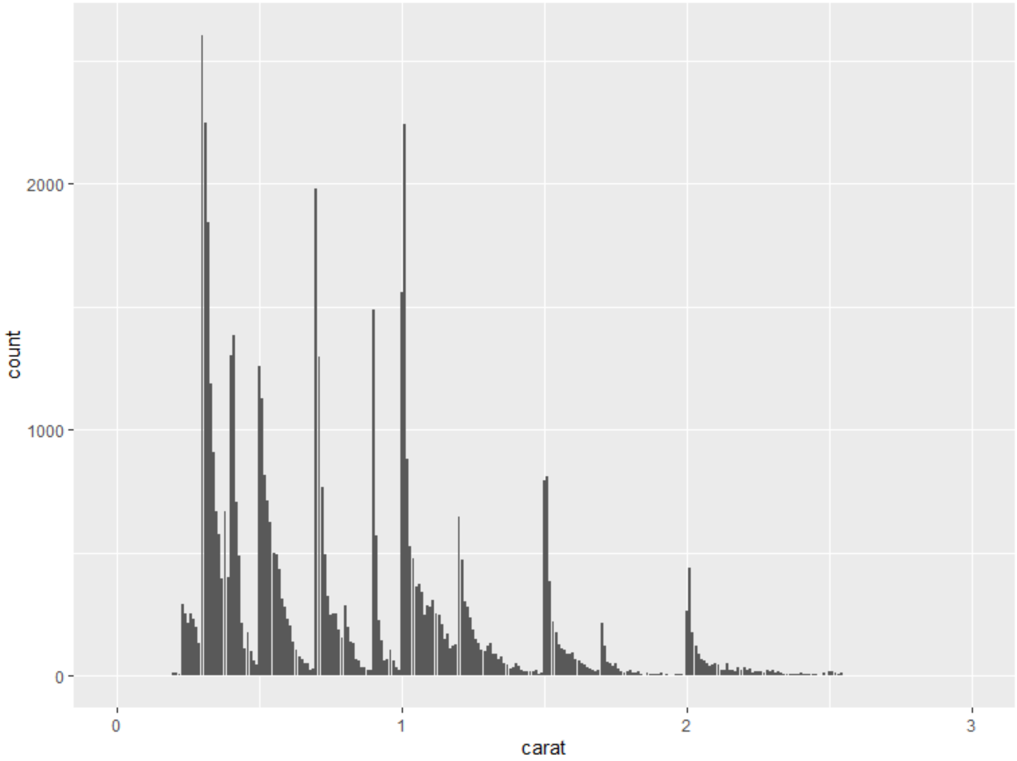
geoms

```
df <- data.frame(
  x = c(3, 1, 5),
  y = c(2, 4, 6),
  label = c("a","b","c"))


p <- ggplot(df, aes(x, y, label = label)) +
  labs(x = NULL, y = NULL) # Hide axis label

p1 <- p + geom_point() +
ggtitle("geom_point")
...


library(cowplot)

plot_grid(p1, ..., nrow = 2)
```

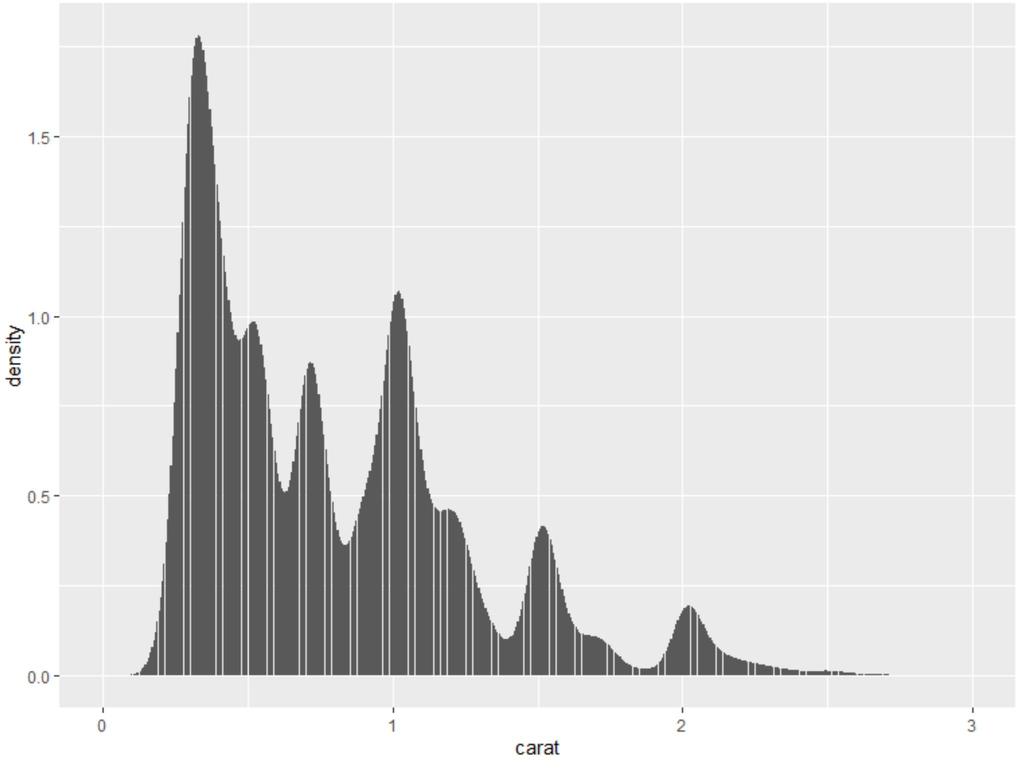


Modifying `geoms` with different `stats`

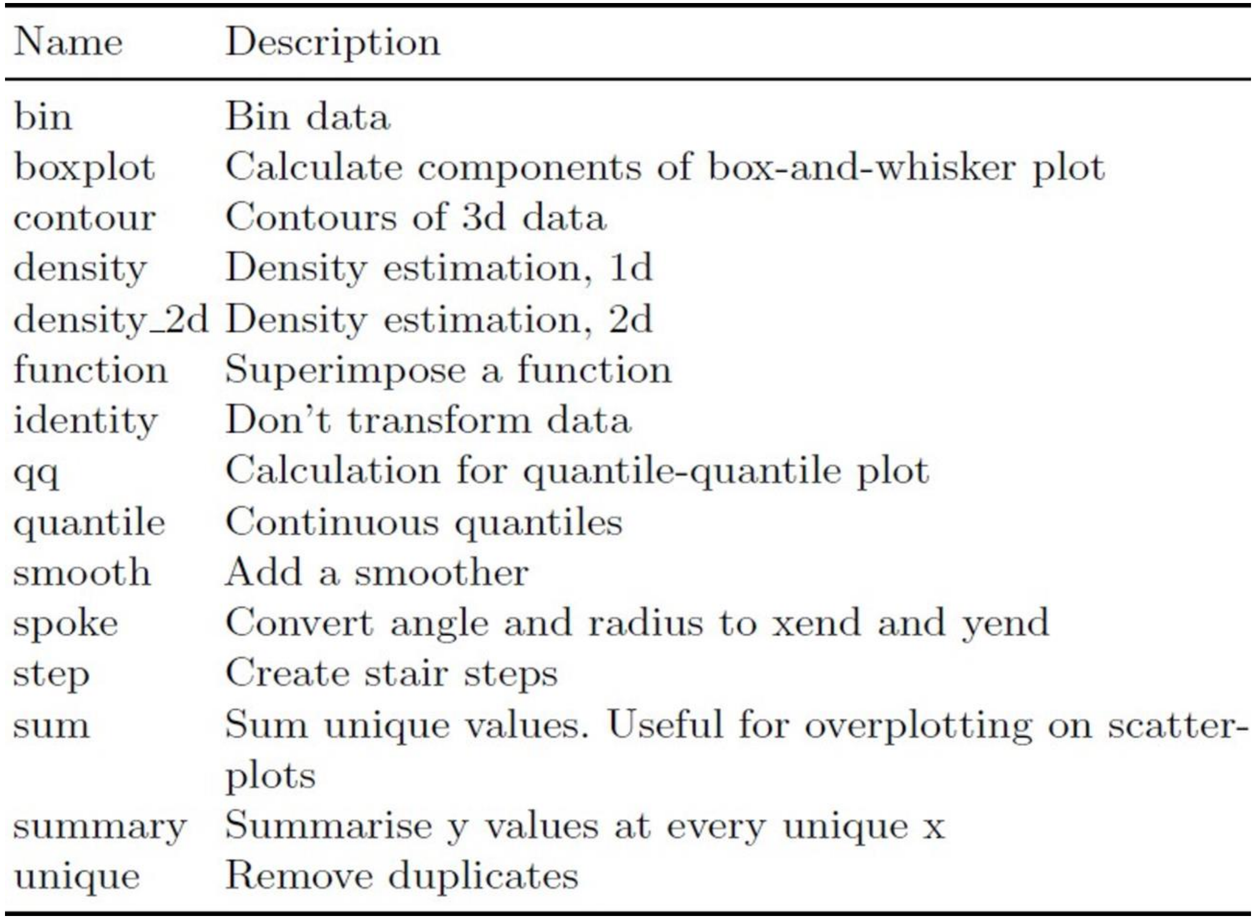
```
d <- ggplot(diamonds, aes(carat)) +
xlim(0, 3)
d  + geom_histogram(stat = "count")
```



```
d + geom_histogram(stat = "density")
```



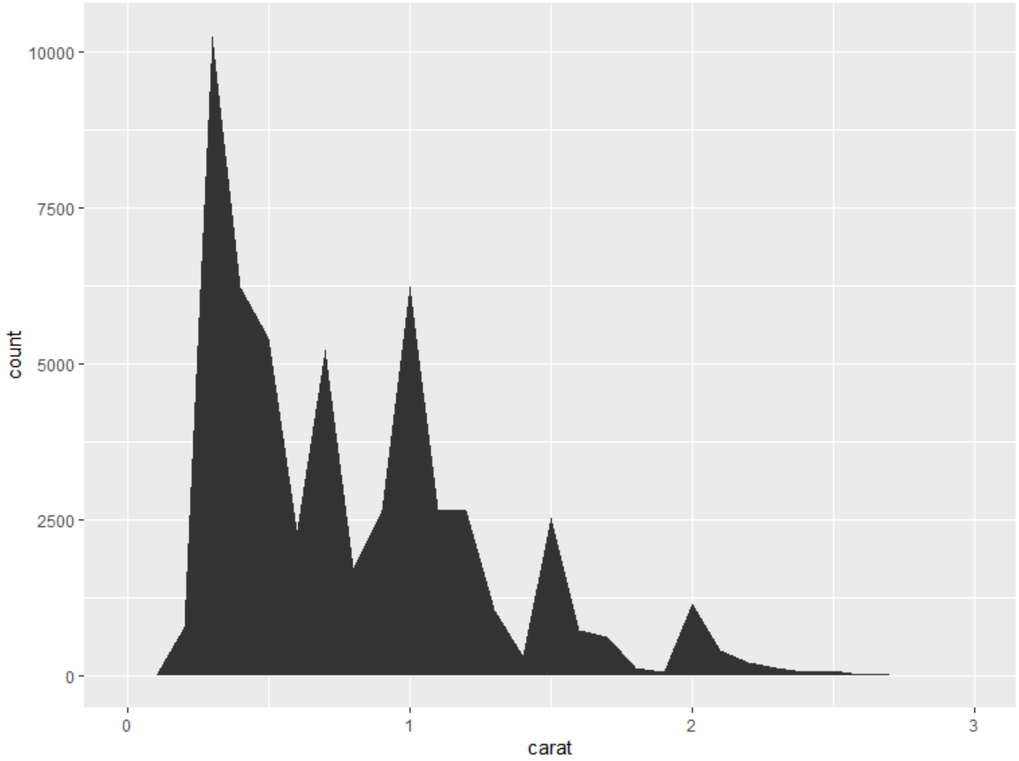
stats



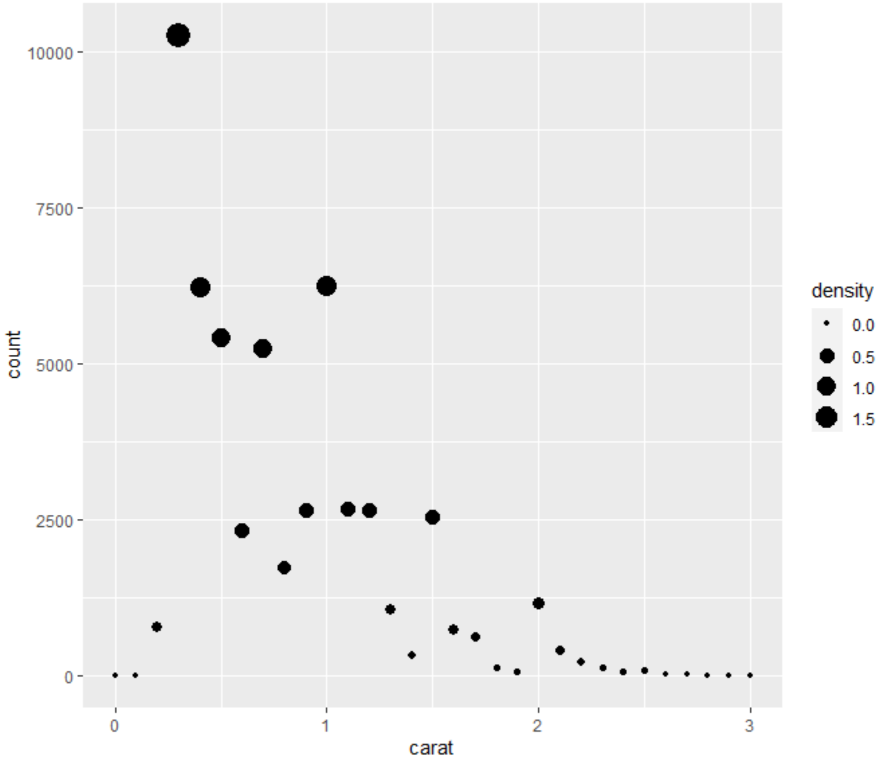
Name	Description
bin	Bin data
boxplot	Calculate components of box-and-whisker plot
contour	Contours of 3d data
density	Density estimation, 1d
density_2d	Density estimation, 2d
function	Superimpose a function
identity	Don't transform data
qq	Calculation for quantile-quantile plot
quantile	Continuous quantiles
smooth	Add a smoother
spoke	Convert angle and radius to xend and yend
step	Create stair steps
sum	Sum unique values. Useful for overplotting on scatterplots
summary	Summarise y values at every unique x
unique	Remove duplicates

Choosing different `stats`

```
d <- ggplot(diamonds, aes(carat)) +
xlim(0, 3)
d + stat_bin(aes(ymax = ..count..),
binwidth = 0.1, geom = "area")
```



```
d + stat_bin(aes(size = ..density..),
binwidth = 0.1, geom = "point",
position="identity" )
```



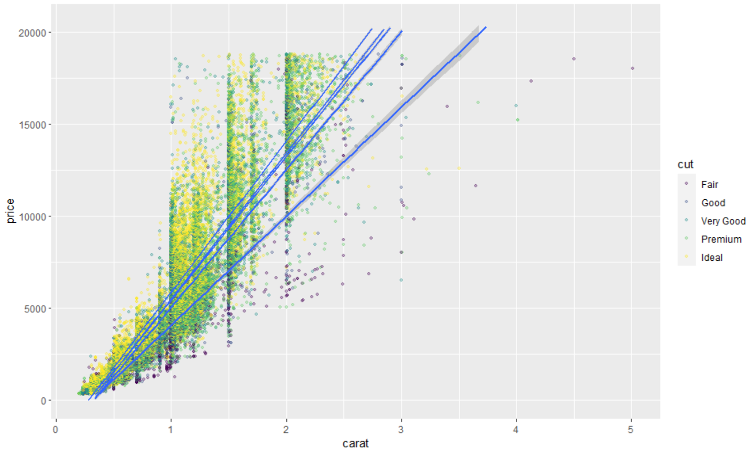
`geom_xxxx` and `stat_xxxx` are shortcuts for layer

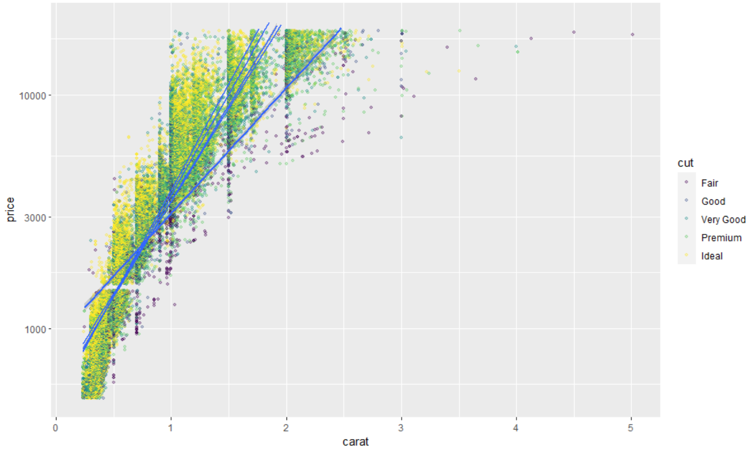
```r
p <- ggplot(diamonds, aes(x = carat))

p + layer(geom = "bar", stat = "bin", position = 'identity', params = list(fill =
"steelblue", binwidth=0.1))

p + geom_histogram(stat="bin", fill="steelblue", binwidth = 0.1)

p + stat_bin(geom="bar", fill="steelblue", binwidth = 0.1)
```

Axis scales

Possible modifications

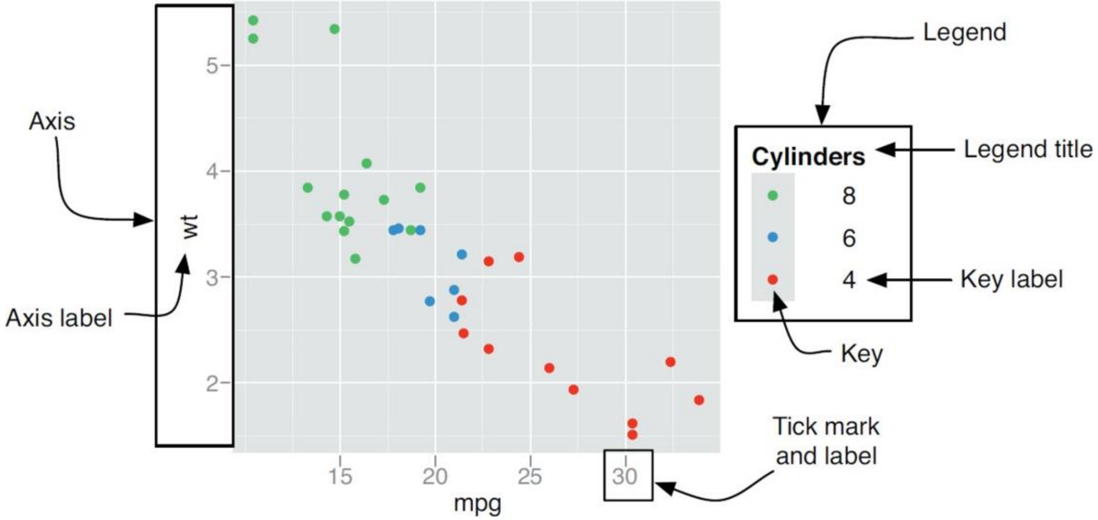
```
# Possible scales:
scale_<PARAMETER>_<SCALE_TYPE>

scale_x_continuous()
scale_y_log10()
scale_color_continuous()

...

# Example

g <- ggplot(data = diamonds, mapping = aes(...))

g1 <- g + geom_point(...)
g2 <- g1 + geom_smooth(...)
g3 <- g2 + scale_y_continuous(limits = c(...))

g3
g3 + scale_y_log10(limits = c(...))
```





Axes, legends, lables, and theme elements

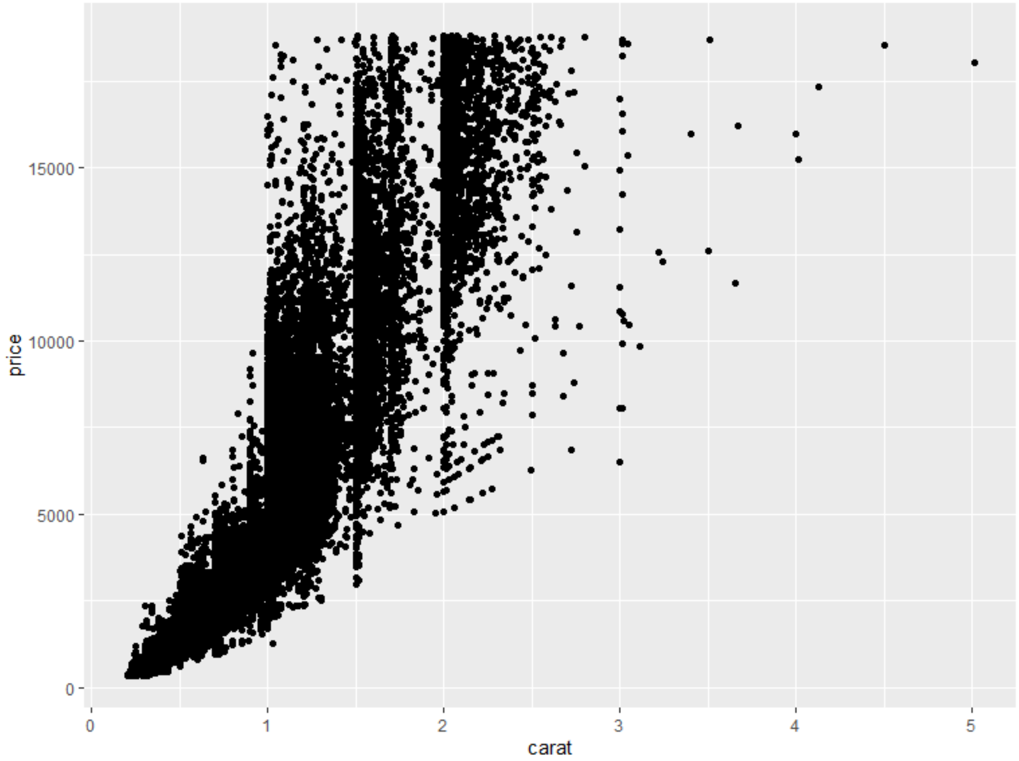
```
labs(title="...", y="...", x="...",
caption="...")
ggtitle("..."), xlab("..."),
ylab("...")

guides()

theme()

theme_bw(), theme_classic(),
theme_gray(), theme_minimal(), ...


theme_get()
```



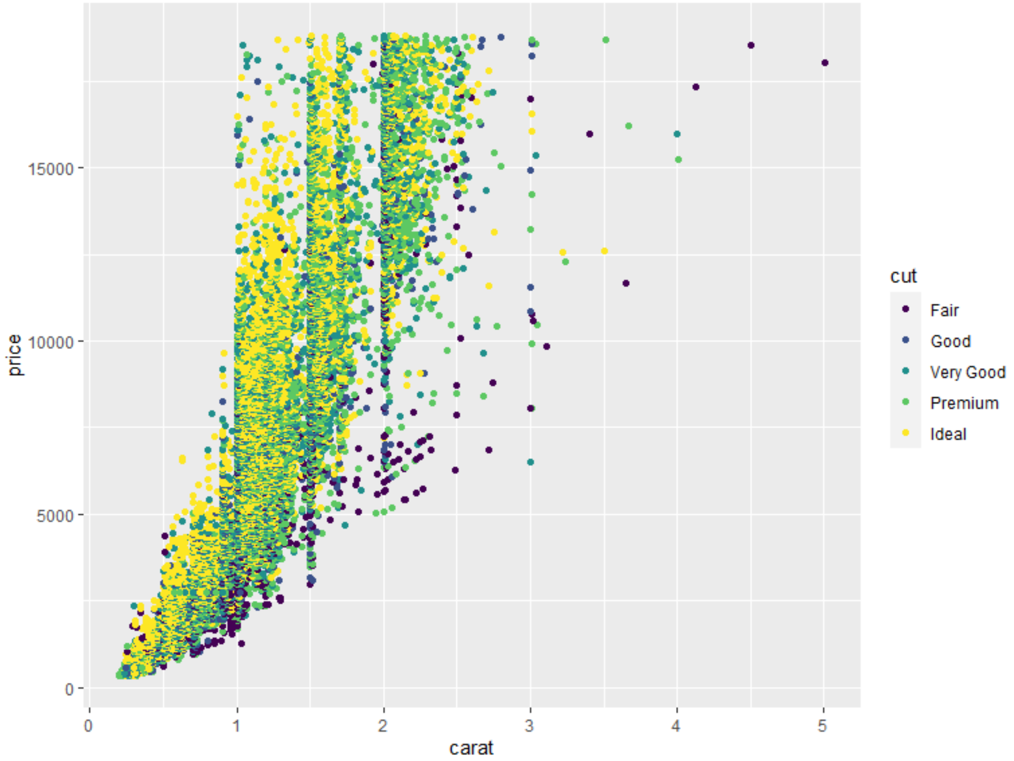
DEALING WITH SEVERAL GROUPS ON THE SAME GRAPH IN `ggplot2`

Grouping

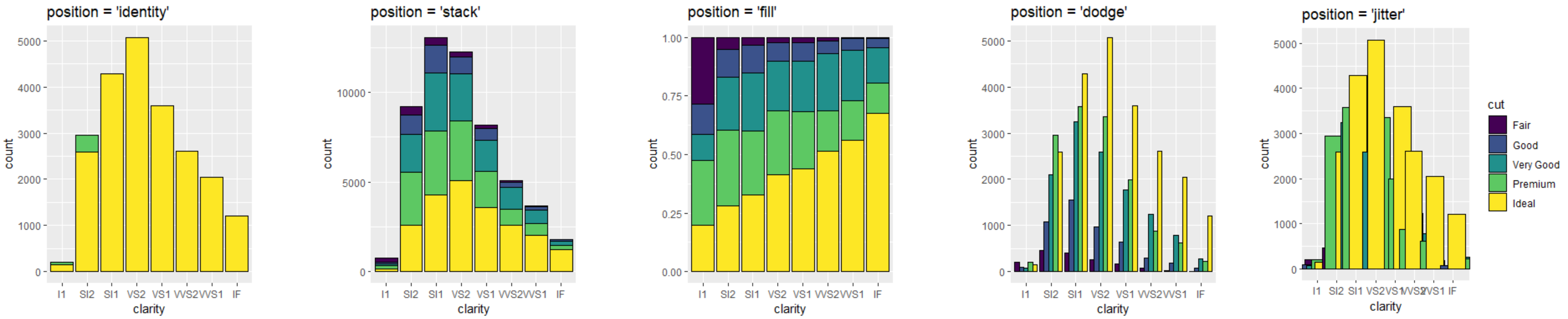
```
g <- ggplot(data = diamonds, aes(x =
carat, y = price))
g + geom_point()
```



```
g + geom_point(aes(group = cut, colour =
cut))
```



Position adjustment



position = 'identity'

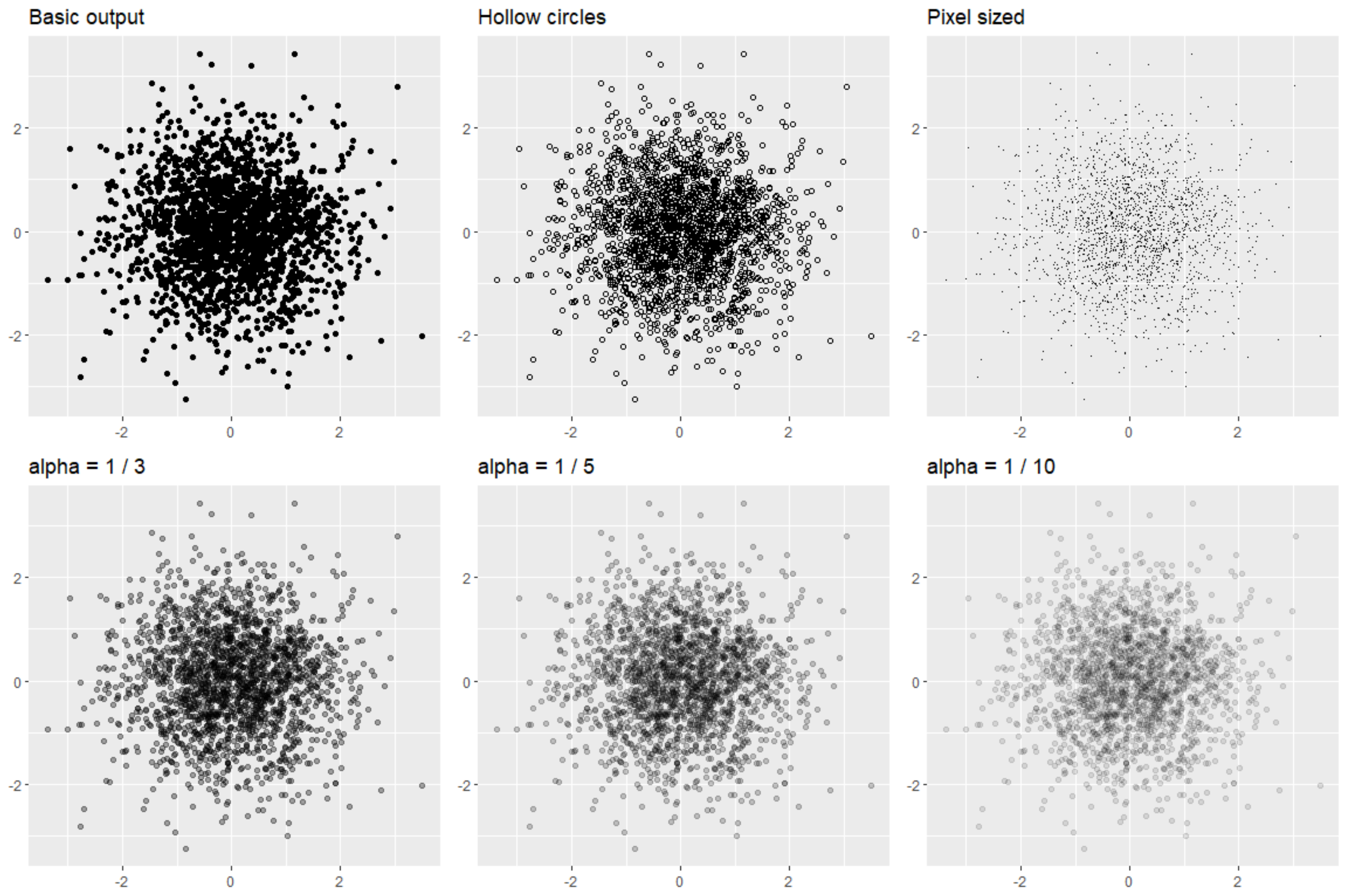
position = 'stack'

position = 'fill'

position = 'dodge'

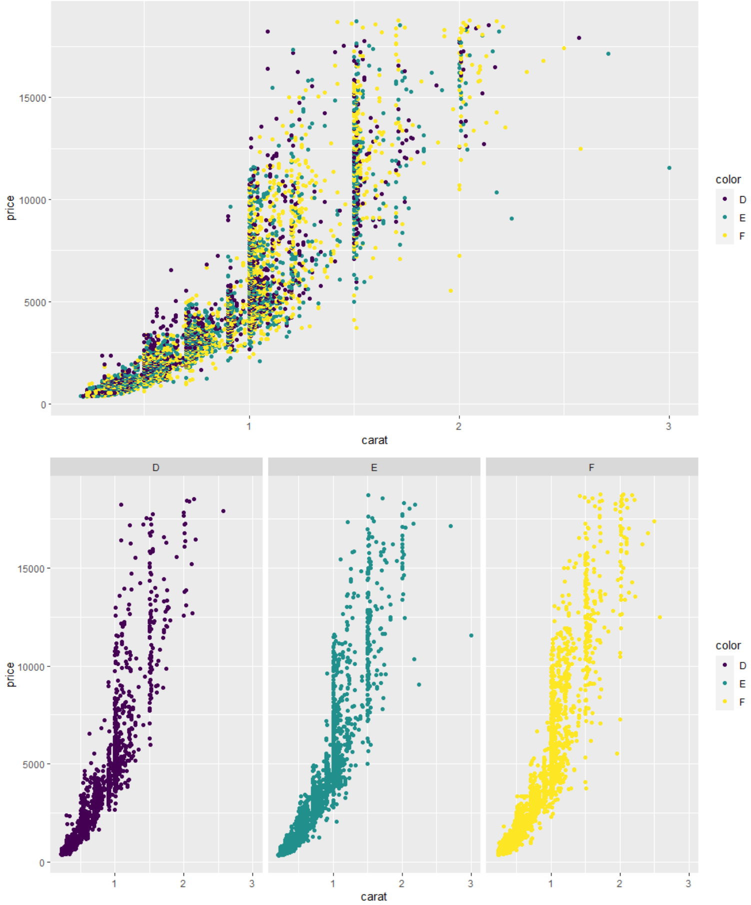
position = 'jitter'

Overplotting



Grouping vs faceting

```
p <- ggplot(data = filter(diamonds,
all_of(color == c("D", "E", "F"))), aes(x
= carat, y = price))

p_group <- p +
geom_point(aes(colour=color))

p_facet <- p +
geom_point(aes(colour=color)) +
facet_grid(. ~ color)

plot_grid(p_group, p_facet, nrow = 2,
labels = NULL)


# Functions for faceting

p + facet_grid(rows = vars(some_variable))
# facet_grid() requires you to have all
the variables you need

p + facet_wrap(vars(drv), nrow=3)
# facet_wrap() is a more flexible function
```



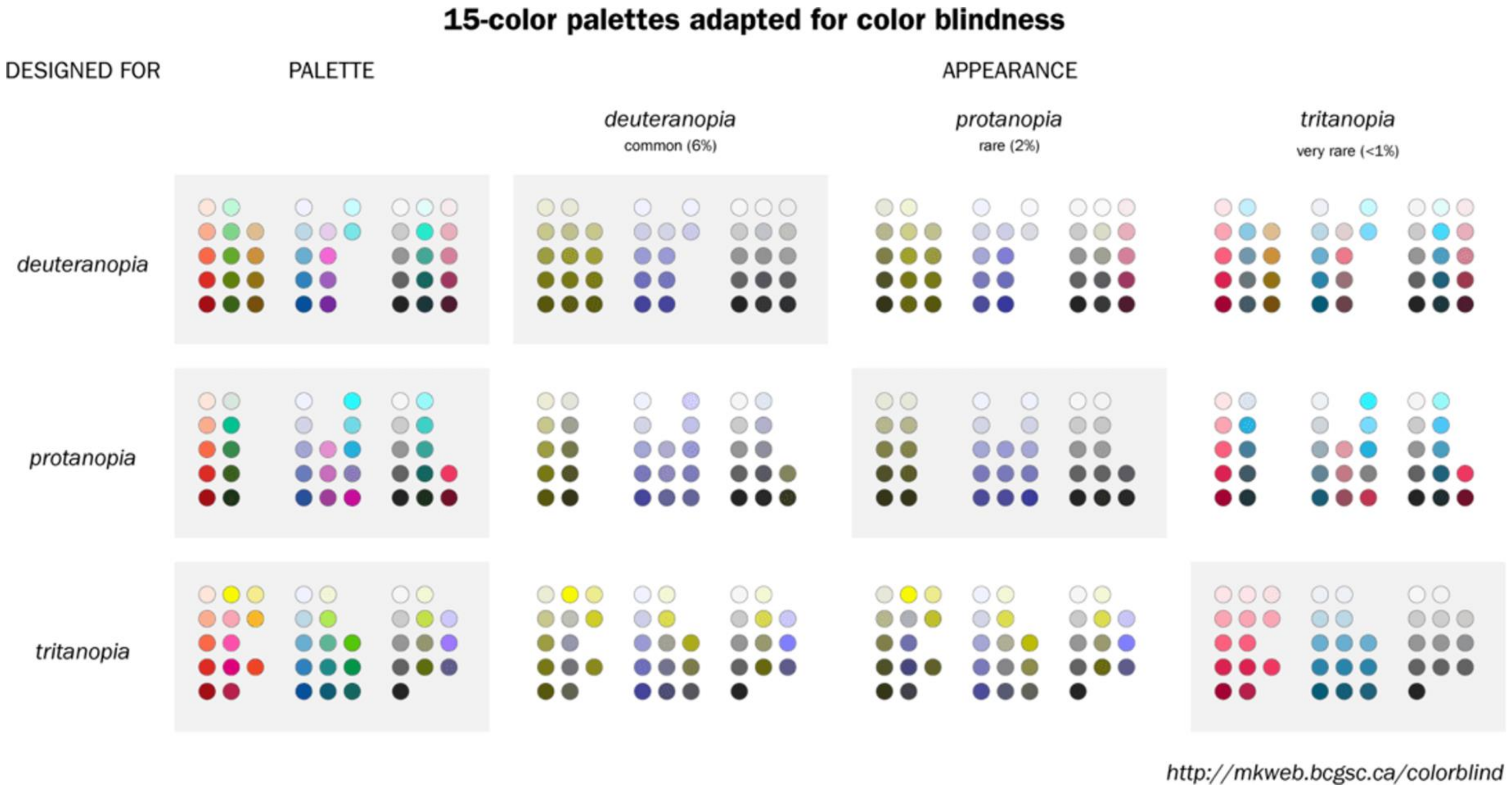
OTHER IMPORTANT POINTS

Saving your graph to a file

1. Open the graphical device (`png(...), bmp(...), jpg(...), tiff(...), pdf(...), windows/quartz/x11, svg(...), RStudioGD`) and specify its parameters (`file name, width, height, resolution, etc`);
2. Run the code to depict your graph;
3. Close the device.

```
png(file=“my_plot.png”, width=500, height=500, units=“px”)

d <- ggplot(diamonds, aes(carat)) + xlim(0, 3)

d + stat_bin(aes(size = ..density.., colour=..density..), binwidth =  0.1, geom =
"point", position="identity" )

dev.off()


ggsave(myplot, “path/filename.png”, width = 6, height = 4)
```

Color palettes



http://mkweb.bcgsc.ca/colorblind

Color palettes in R

```
colors() # Shows all the available in-built colors

palette2 <- colorRampPalette(c("tomato", "purple1"))(5)
palette(palette2)
palette()

[1] "tomato"  "#E65675" "#CD49A3" "#B33CD1" "purple1"

palette3 <- c(palette2[c(1:3,5)], "blue1")
palette(palette3)
palette()

[1] "tomato"  "#E65675" "#CD49A3" "purple1" "blue"

rgb(red = 0.9, green = 0.5, blue = 0.3)

[1] "#E6804D"

rainbow(7)

[1] "#FF0000" "#FFDB00" "#49FF00" "#00FF92" "#0092FF" "#4900FF" "#FF00DB"

library(RColorBrewer)
brewer.pal(3, "BuGn")

[1] "#E5F5F9" "#99D8C9" "#2CA25F"
```

Conclusions

By now, you should

1. Become more acquainted with `graphics`, `ggplot2`, and `lattice` packages.
2. Understand principles of making nice graphs
3. Be able to export your graphs and to change your color palette.

Further reading

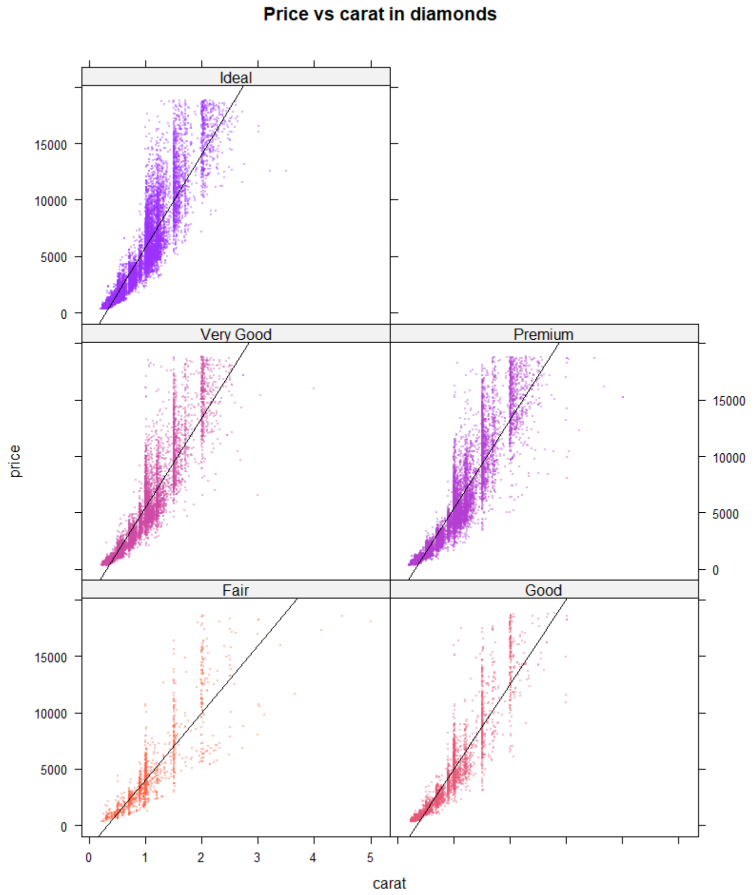
- `library(swirl)    # An R package for self-learning Exploratory data analysis course`
- Wickham H. Ggplot2: Elegant graphics for data analysis. 2nd ed. Cham, Switzerland: Springer International Publishing; 2016.
- Chang W. R graphics cookbook: Practical recipes for visualizing data. 2nd ed. O'Reilly Media; 2018.
- Murrell P. R Graphics. Philadelphia, PA: Chapman & Hall/CRC; 2006
- https://r-charts.com/
- https://stackoverflow.com/
- http://www.sthda.com/

THANK YOU FOR ATTENTION

MAKING GRAPHS IN `lattice`

Using the `lattice` package to create plots: `diamonds` dataset

```
library(lattice)

xyplot(price ~ carat | cut,
       data = diamonds,
       layout = c(2, 3),
       col = alpha(palette3, alpha = 0.4),
       groups = cut, type = c("p", "r"),
       pch = 16, cex = 0.3,
       col.line = "black",
       main = "Price vs carat in diamonds")
```


Price vs carat in diamonds
Ideal
Very Good
Premium
Fair
Good
price
carat