R09. ggplot (3)

Cheatsheets & Taxonomy of graphics

Sim, Min Kyu, Ph.D.
mksim@seoultech.ac.kr

SEOULTECH 서울과학기술대학교 데이터사이언스학과

About

About

- 그래프를 그릴때는 익숙하고 머리속에 떠오르는 plot에서 벗어나 cheatsheet을 확인하면서 많은 대안을 떠올려보는 습관이 좋습니다.
- 그래서 이번 노트는 ggplot의 cheatsheet을 보면서 여러가지 geom들을 살펴보도록 하겠습니다.

Data visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

```
ggplot (data = <DATA>) +                                   required
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
 stat =<STAT>, position =<POSITION>) +                     Not required, sensible defaults supplied
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

ggplot(data = mpg, **aes(**x = cty, y = hwy**))** Begins a plot that you finish by adding layers to. Add one geom function per layer.

last_plot() Returns the last plot.

ggsave("plot.png", width = 5, height = 5**)** Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Aes Common aesthetic values.

color and **fill** - string ("red", "#RRGGBB")

linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")

lineend - string ("round", "butt", or "square")

linejoin - string ("round", "mitre", or "bevel")

size - integer (line width in mm)

shape - integer/shape name or a single character ("a")

RStudio

Geoms

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

GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank() and **a + expand_limits()**
Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend = "butt", linejoin = "round", linemitre = 1**)**
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(alpha = 50)**)** - x, y, alpha, color, fill, group, subgroup, linetype, size

b + geom_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1)**)** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)**)** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept = 0, slope = 1)**)**
b + geom_hline(aes(yintercept = lat)**)**
b + geom_vline(aes(xintercept = long)**)**

b + geom_segment(aes(yend = lat + 1, xend = long + 1)**)**
b + geom_spoke(aes(angle = 1:1155, radius = 1)**)**

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin"**)**
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly()
x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)**)**
x, y, alpha, color, fill, linetype, size, weight

discrete

d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

both continuous

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

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

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

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

e + geom_rug(sides = "bl"**)**
x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)**
x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

one discrete, one continuous

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

f + geom_col()
x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot()
x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, group, linetype, size, weight

both discrete

g <- ggplot(diamonds, aes(cut, color))

g + geom_count()
x, y, alpha, color, fill, shape, size, stroke

e + geom_jitter(height = 2, width = 2**)**
x, y, alpha, color, fill, shape, size

continuous bivariate distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density_2d()
x, y, alpha, color, group, linetype, size

h + geom_hex()
x, y, alpha, color, fill, size

continuous function

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

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

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv"**)**
x, y, alpha, color, group, linetype, size

visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))

j + geom_crossbar(fatten = 2**)** - x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar() - x, ymax, ymin, alpha, color, group, linetype, size, width
Also **geom_errorbarh()**.

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange() - x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map**)**
+ expand_limits(x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size

THREE VARIABLES

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z)**)**
x, y, z, alpha, color, group, linetype, size, weight

l + geom_contour_filled(aes(fill = z)**)**
x, y, alpha, color, fill, group, linetype, size, subgroup

l + geom_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE**)**
x, y, alpha, fill

l + geom_tile(aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size, width

Stats An alternative way to build a layer.

A stat builds new variables to plot (e.g., count, prop).

Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

geom to use | stat function | geommappings

i + stat_density_2d(aes(fill = ..level..), geom = "polygon"**)**

variable created by stat

c + stat_bin(binwidth = 1, boundary = 10**)**
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1**) x, y |** ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian"**)**
x, y | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T**)**
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins = 30**) x, y, fill |** ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100**)**
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t"**)**

l + stat_contour(aes(z = z)**) x, y, z, order |** ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max**)**
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean**)**
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5**)**
x, y | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area"**) x, y**
| ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40**) x, y |** ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9),
formula = y ~ log(x), method = "rq"**) x, y |** ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se = T,
level = 0.95**) x, y |** ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + xlim(-5, 5**) + stat_function(**fun = dnorm,
n = 20, geom = "point"**) x |** ..x.., ..y..

ggplot() + stat_qq(aes(sample = 1:100)**)**
x, y, sample | ..sample.., ..theoretical..

e + stat_sum() x, y, size | ..n.., ..prop..

e + stat_summary(fun.data = "mean_cl_boot"**)**

h + stat_summary_bin(fun = "mean", geom = "bar"**)**

e + stat_identity()

e + stat_unique()

Scales Override defaults with **scales** package.

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

n <- d + geom_bar(aes(fill = fl))

scale_ | aesthetic to adjust | prepackaged scale to use | scale-specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - Map cont' values to visual ones.
scale_*_discrete() - Map discrete values to visual ones.
scale_*_binned() - Map continuous values to discrete bins.
scale_*_identity() - Use data values as visual ones.
scale_*_manual(values = c()**)** - Map discrete values to manually chosen visual ones.
scale_*_date(date_labels = "%m/%d"),
date_breaks = "2 weeks"**)** - Treat data values as dates.
scale_*_datetime() - Treat data values as date times. Same as scale_*_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale.
scale_x_reverse() - Reverse the direction of the x axis.
scale_x_sqrt() - Plot x on square root scale.

COLOR AND FILL SCALES (DISCRETE)

n + scale_fill_brewer(palette = "Blues"**)**
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2,
end = 0.8, na.value = "red"**)**

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

o + scale_fill_distiller(palette = "Blues"**)**

o + scale_fill_gradient(low="red", high="yellow"**)**

o + scale_fill_gradient2(low = "red", high = "blue",
mid = "white", midpoint = 25**)**

o + scale_fill_gradientn(colors = topo.colors(6)**)**
Also: rainbow(), heat.colors(), terrain.colors(),
cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

p <- e + geom_point(aes(shape = fl, size = cyl))

p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7)**)**

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

p + scale_radius(range = c(1,6)**)**
p + scale_size_area(max_size = 6**)**

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5)**)** - xlim, ylim
The default cartesian coordinate system.

r + coord_fixed(ratio = 1/2**)**
ratio, xlim, ylim - Cartesian coordinates with fixed aspect ratio between x and y units.

ggplot(mpg, aes(y = fl)**) + geom_bar()**
Flip cartesian coordinates by switching x and y aesthetic mappings.

r + coord_polar(theta = "x", direction=1**)**
theta, start, direction - Polar coordinates.

r + coord_trans(y = "sqrt"**)** - x, y, xlim, ylim
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation = c(41, -74, 0)**)** - projection, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.).

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side.

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height.

e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting.

e + geom_label(position = "nudge")
Nudge labels away from points.

s + geom_bar(position = "stack")
Stack elements on top of one another.

Each position adjustment can be recast as a function with manual **width** and **height** arguments:
s + geom_bar(position = position_dodge(width = 1))

Themes

r + theme_bw()
White background with grid lines.

r + theme_gray()
Grey background (default theme).

r + theme_dark()
Dark for contrast.

r + theme_classic()

r + theme_light()

r + theme_linedraw()

r + theme_minimal()
Minimal theme.

r + theme_void()
Empty theme.

r + theme() Customize aspects of the theme such as axis, legend, panel, and facet properties.
r + ggtitle("Title") + theme(plot.title.postion = "plot")
r + theme(panel.background = element_rect(fill = "blue"))

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(cols = vars(fl))
Facet into columns based on fl.

t + facet_grid(rows = vars(year))
Facet into rows based on year.

t + facet_grid(rows = vars(year), cols = vars(fl))
Facet into both rows and columns.

t + facet_wrap(vars(fl))
Wrap facets into a rectangular layout.

Set **scales** to let axis limits vary across facets.

t + facet_grid(rows = vars(drv), cols = vars(fl),
scales = "free")
x and y axis limits adjust to individual facets:
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet label:

t + facet_grid(cols = vars(fl), labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(rows = vars(fl),
labeller = label_bquote(alpha ^ .(fl)**))**

α^{c} α^{d} α^{e} α^{p} α^{r}

Labels and Legends

Use **labs()** to label the elements of your plot.

t + labs(x = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
alt = "Add alt text to the plot",
<AES> = "New **<AES>** legend title"**)**

t + annotate(geom = "text", x = 8, y = 9, label = "A"**)**
Places a geom with manually selected aesthetics.

p + guides(x = guide_axis(n.dodge = 2)**)** Avoid crowded or overlapping labels with guide_axis(n.dodge or angle).

n + guides(fill = "none"**)** Set legend type for each aesthetic: colorbar, legend, or none (no legend).

n + theme(legend.position = "bottom"**)**
Place legend at "bottom", "top", "left", or "right".

n + scale_fill_discrete(name = "Title",
labels = c("A", "B", "C", "D", "E"**))**
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred):
t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20)**)**

With clipping (removes unseen data points):
t + xlim(0, 100**) + ylim(**10, 20**)**

t + scale_x_continuous(limits = c(0, 100)**) +**
scale_y_continuous(limits = c(0, 100)**)**

Basic

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

```
ggplot(data = <DATA>) +                          ┐ required
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),       ┘
  stat = <STAT>, position = <POSITION>) +        ┐
<COORDINATE_FUNCTION> +                          │
<FACET_FUNCTION> +                               │ Not required, sensible defaults supplied
<SCALE_FUNCTION> +                               │
<THEME_FUNCTION>                                 ┘
```

ggplot(data = mpg, **aes(**x = cty, y = hwy**))** Begins a plot that you finish by adding layers to. Add one geom function per layer.

last_plot() Returns the last plot.

ggsave("plot.png", width = 5, height = 5**)** Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

1. One variable
   - 1-1. Continuous
     i) `geom_area()`
     ii) `geom_density()`
     iii) `geom_dotplot()`
     iv) `geom_freqpoly()`
     v) `geom_histogram()`
     vi) `geom_geom_qq()`
   - 1-2. Discrete
     i) `geom_bar()`
2. Two variables (basics)
   - 2-1. Both continuous
     i) `geom_point()`
     ii) `geom_label()`
     iii) `geom_text()`
     iv) `geom_quantile()`
     v) `geom_rug()`
     vi) `geom_smooth()`
   - 2-2. One discrete, one continuous
     i) `geom_col()`
     ii) `geom_boxplot()`
     iii) `geom_dotplot()`
     iv) `geom_violin()`
   - (cont'd)
     - 2-3. Both discrete
       i) `geom_count()`
       ii) `geom_jitter()`
3. Two variables (advanced)
   - 3-1. Continuous bivariate
     i) `geom_bin2d()`
     ii) `geom_density_2d()`
     iii) `geom_text()`
   - 3-2. Continuous function
     i) `geom_area()`
     ii) `geom_line()`
     iii) `geom_step()`
   - 3-3. Visualizing error
     i) `geom_crossbar()`
     ii) `geom_errorbar()`
     iii) `geom_linerange()`
     iv) `geom_pointrange()`
   - 3-4. Maps
     i) `geom_map()`

1. One variable

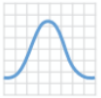
1. One variable

- 1-1. Continuous
  - i) `geom_area()`
  - ii) `geom_density()`
  - iii) `geom_dotplot()`
  - iv) `geom_freqpoly()`
  - v) `geom_histogram()`
  - vi) `geom_geom_qq()`
- 1-2. Discrete
  - i) `geom_bar()`

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

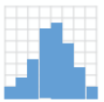
c + geom_area(stat = "bin"**)**
x, y, alpha, color, fill, linetype, size



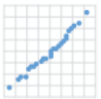
c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly()
x, y, alpha, color, group, linetype, size



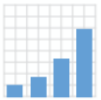
c + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight



c2 + geom_qq(aes(sample = hwy)**)**
x, y, alpha, color, fill, linetype, size, weight

discrete

d <- ggplot(mpg, aes(fl))



d + geom_bar()
x, alpha, color, fill, linetype, size, weight

1-1. Continuous

Dataset

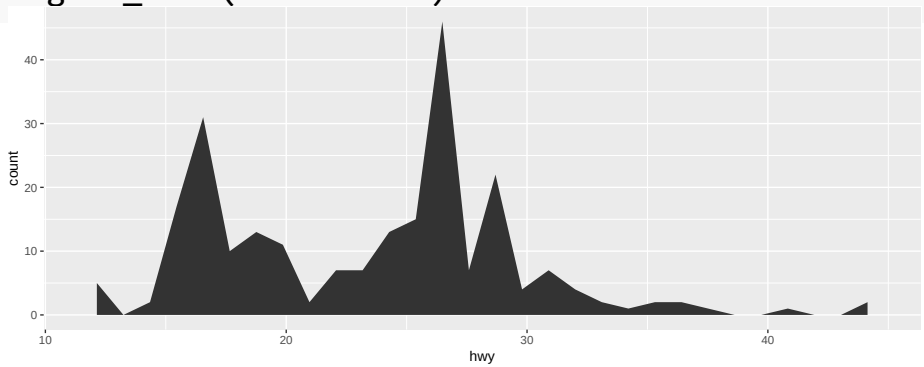
```
mpg$hwy %>% sample(10) # randomly choose 10 elements.
```

```
##  [1] 15 30 17 22 24 26 25 19 26 21
```

i) geom_area()

- x, y, alpha, color, fill, linetype, size

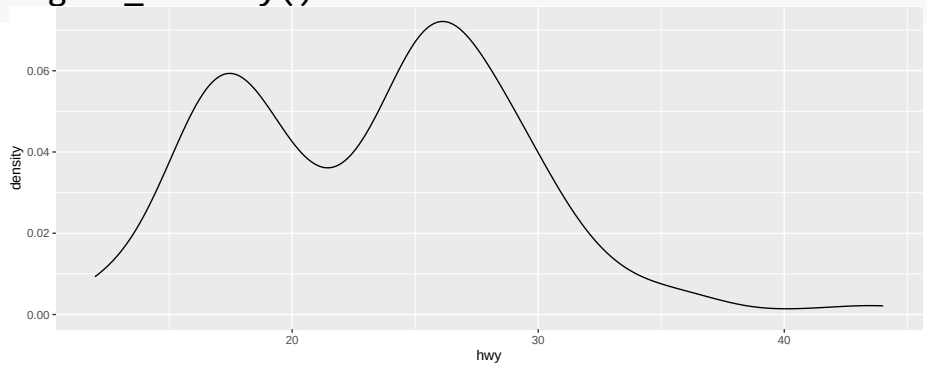
```
ggplot(mpg, aes(x=hwy)) +
  geom_area(stat="bin")
```



ii) geom_density()

- x, y, alpha, color, fill, group, linetype, size, weight

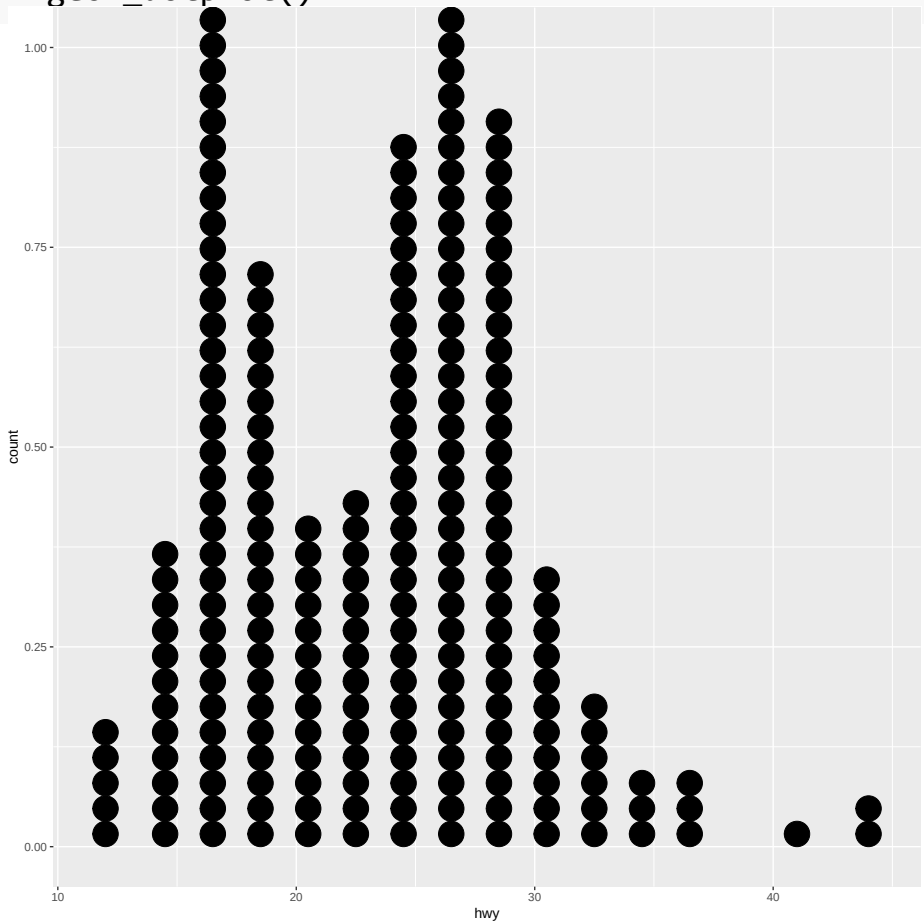
```
ggplot(mpg, aes(x=hwy)) +
  geom_density()
```



iii) geom_dotplot()

- x, y, alpha, color, fill

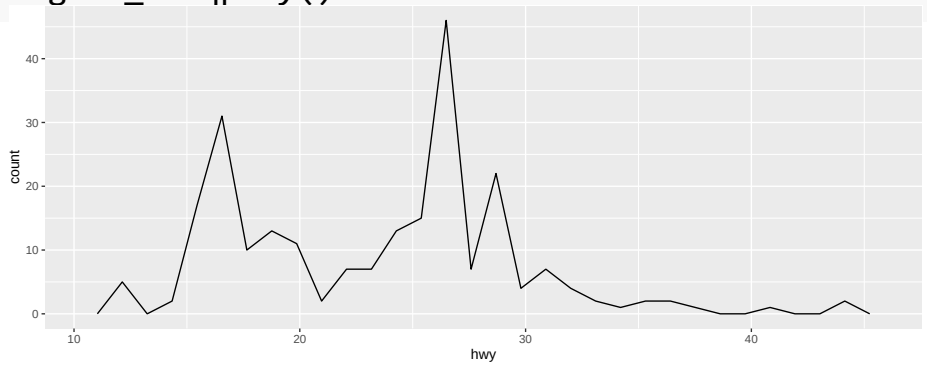
```
ggplot(mpg, aes(x=hwy)) +
  geom_dotplot()
```



iv) geom_freqpoly()

- x, y, alpha, color, group, linetype, size

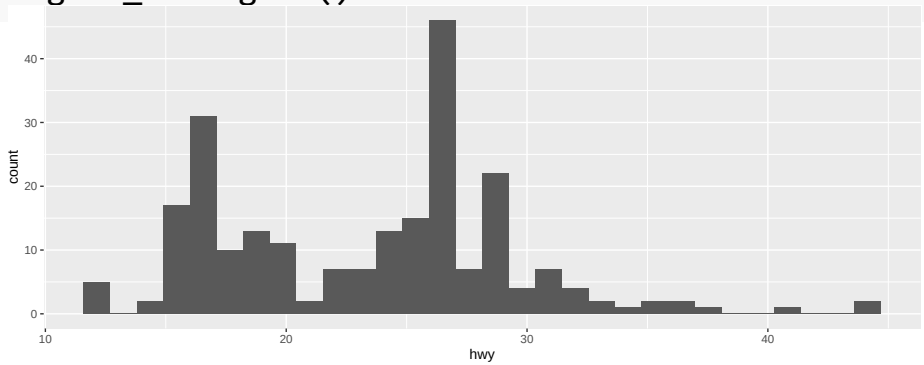
```
ggplot(mpg, aes(x=hwy)) +
  geom_freqpoly()
```



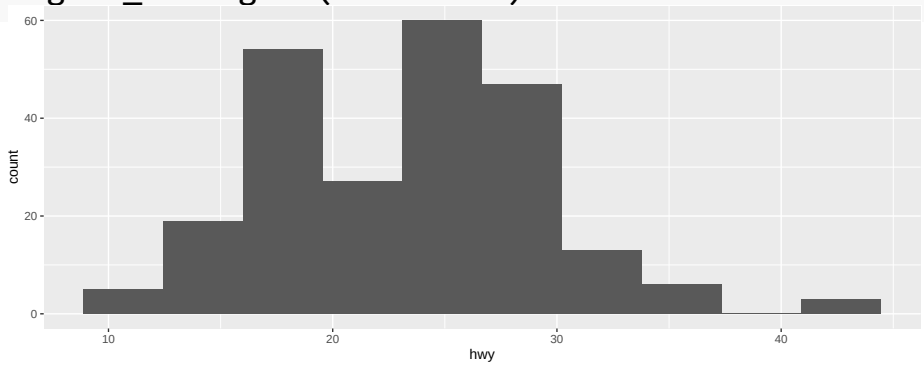
v) geom_histogram()

- x, y, alpha, color, fill, linetype, size, weight

```
ggplot(mpg, aes(x=hwy)) +
  geom_histogram()
```



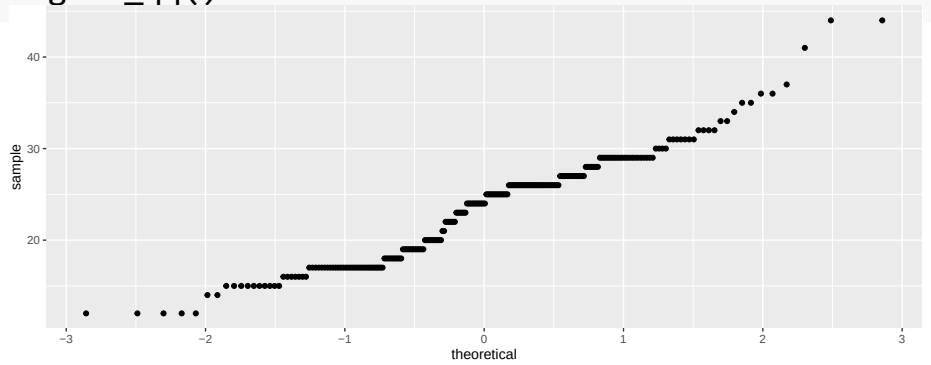

```
ggplot(mpg, aes(x=hwy)) +
  geom_histogram(bins = 10)
```



vi) geom_geom_qq()

- x, y, alpha, color, fill, linetype, size, weight

```
ggplot(mpg, aes(sample=hwy)) +
  geom_qq()
```

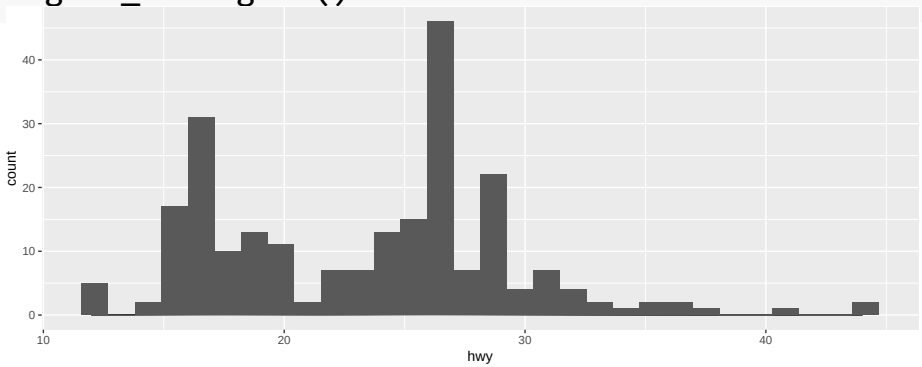


- qqplot: In statistics, a Q-Q plot (quantile-quantile plot) is a probability plot, a graphical method for comparing two probability distributions by plotting their quantiles against each other. (wiki)

Example: *geom_histogram()* + *geom_density()*

- Frequently, you may be tempted to overlay smoothing curve for a histogram.

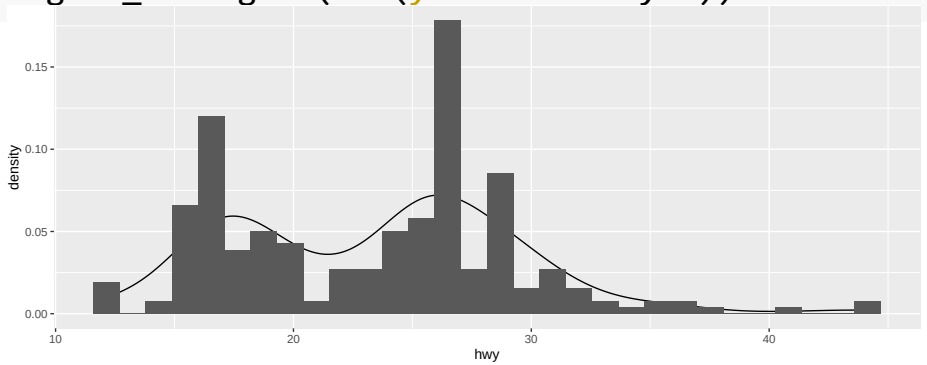
```
ggplot(mpg, aes(x=hwy)) +
  geom_density() +
  geom_histogram()
```



- 그림이 이상하게 나오는 원인
  - `geom_density()`는 0~1 사이의 pdf 값을 출력한다.
  - `geom_histogram()`은 각각의 bin의 관찰값의 갯수를 출력한다.
  - 따라서 y축의 범위가 다르다.

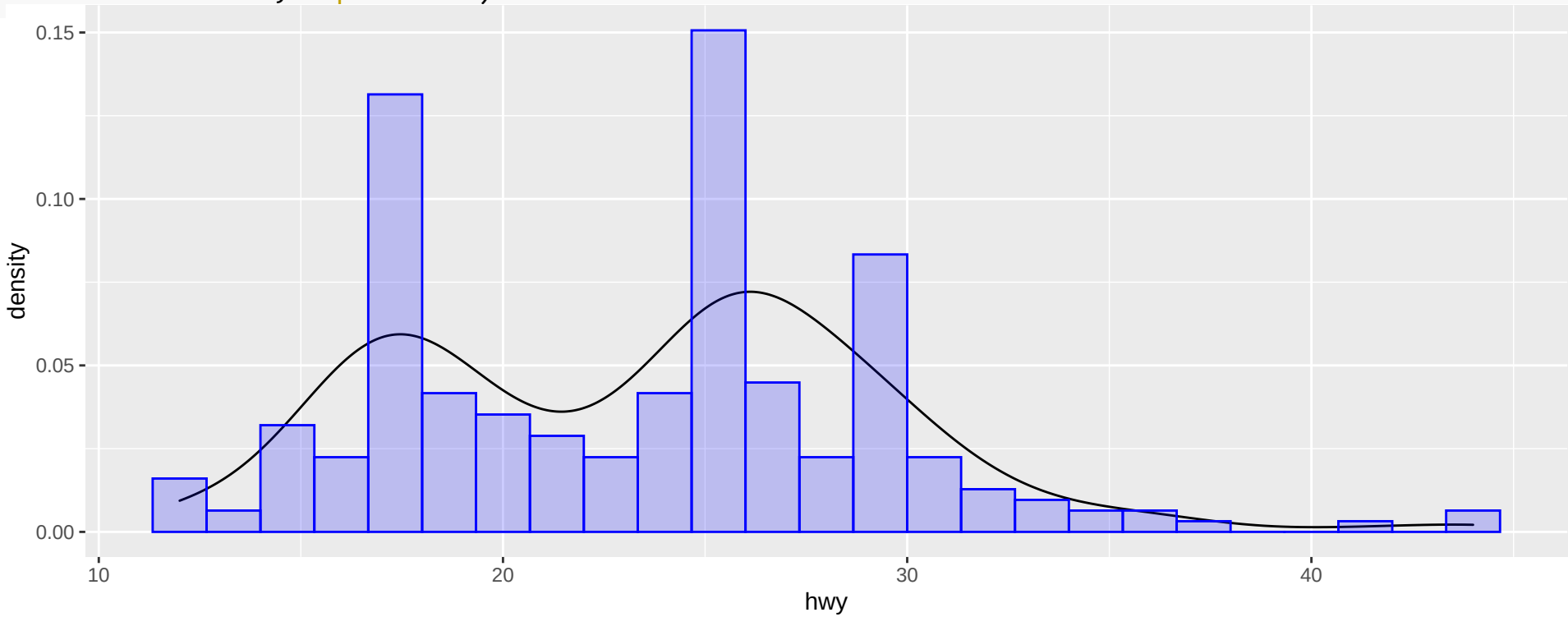
- 해결을 위해서 `geom_histogram()`에서 y의 값이 `density`에 대응되도록 출력하게 한다.

```
ggplot(mpg, aes(x=hwy)) +
  geom_density() +
  geom_histogram(aes(y = ..density..))
```



- Finalizing

```
ggplot(mpg, aes(x=hwy)) +
  geom_density() +
  geom_histogram(aes(y = ..density..),
    bins = 25, color = "blue",
    fill = "blue", alpha = 0.2)
```



1-2. Discrete

Dataset

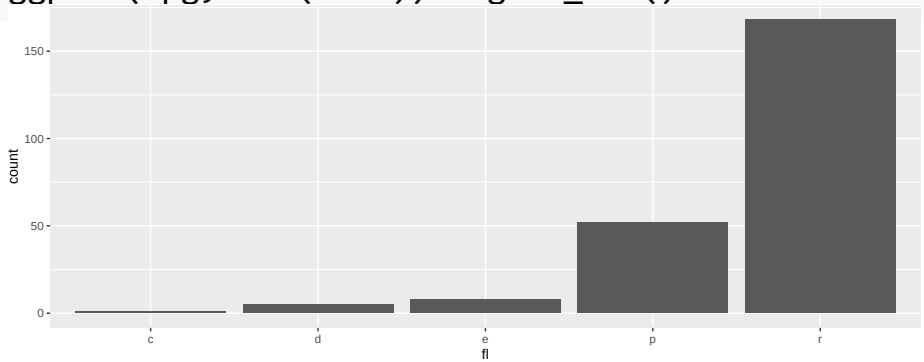
```
mpg$fl %>% sample(10) # randomly choose 10 elements.
```

```
##  [1] "e" "r" "r" "r" "p" "p" "r" "r" "p" "r"
```

i) geom_bar()

- x, y, alpha, color, fill, linetype, size, weight

```
ggplot(mpg, aes(x=fl)) + geom_bar()
```



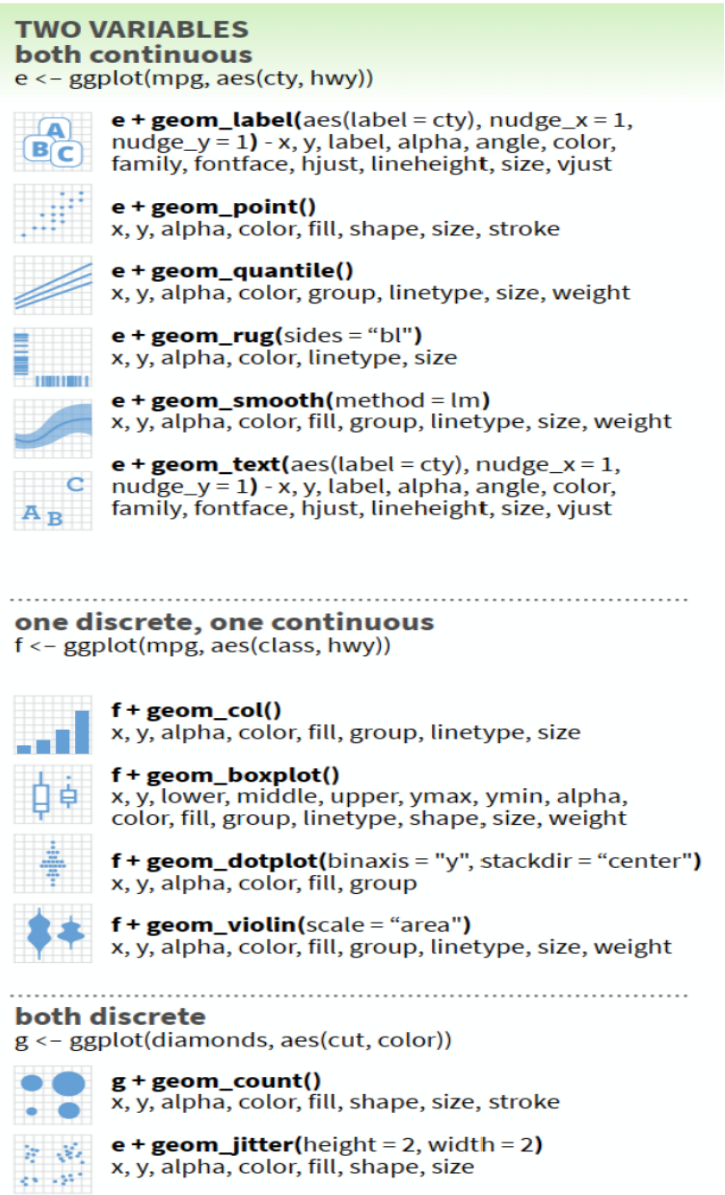
Exercise

- `mpg` 데이터셋의 `hwy` 변수에 대한 histogram을 그려라.
- 위의 histogram에 겹쳐서 normal curve를 그려라.
- 위의 `geom_histogram()`의 파라미터 `bins`의 값을 10으로 해보고 25로 해보라.
- `hwy` 변수는 정규분포와 유사하다고 주장할 수 있는가?

2. Two variables (basic)

2. Two variables (basics)

- 2-1. Both continuous
  - i) `geom_point()`
  - ii) `geom_label()`
  - iii) `geom_text()`
  - iv) `geom_quantile()`
  - v) `geom_rug()`
  - vi) `geom_smooth()`
- 2-2. One discrete, one continuous
  - i) `geom_col()`
  - ii) `geom_boxplot()`
  - iii) `geom_dotplot()`
  - iv) `geom_violin()`
- 2-3. Both discrete
  - i) `geom_count()`
  - ii) `geom_jitter()`



2-1. Both continuous

Dataset

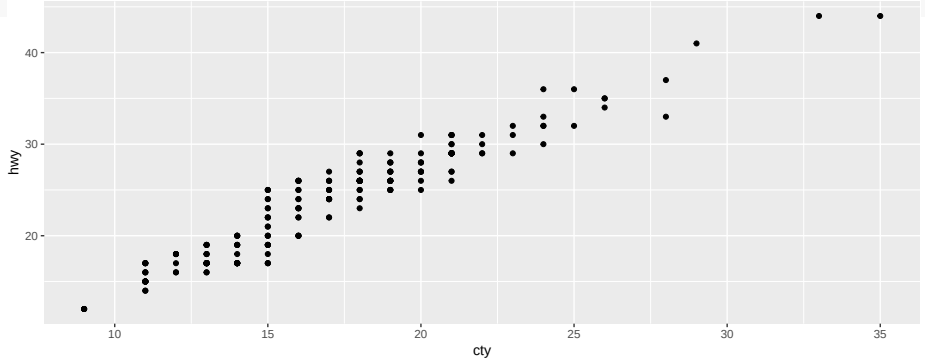
```
mpg %>% select(cty, hwy, manufacturer) %>% head()
```

```
## # A tibble: 6 x 3
##     cty   hwy manufacturer
##   <int> <int> <chr>
## 1    18    29 audi
## 2    21    29 audi
## 3    20    31 audi
## 4    21    30 audi
## 5    16    26 audi
## 6    18    26 audi
```

i) geom_point()

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

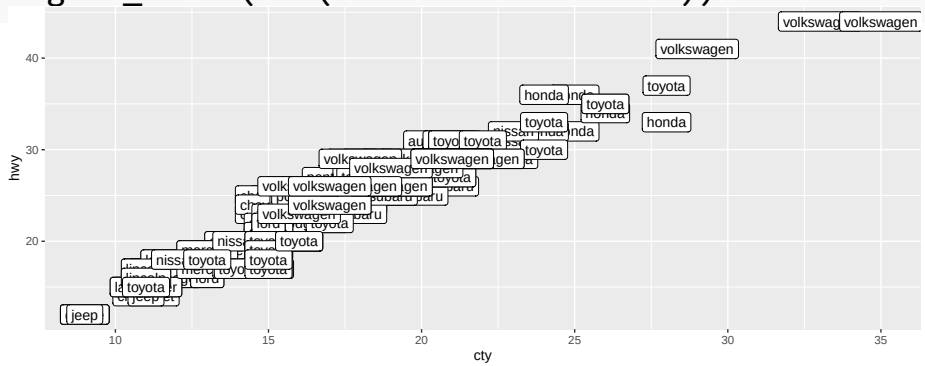
```
ggplot(mpg, aes(x=cty, y=hwy)) + geom_point()
```



ii) geom_label()

- x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

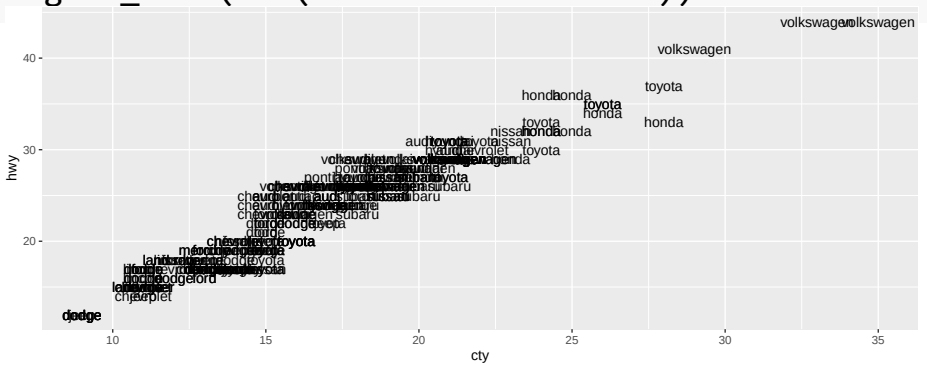
```
ggplot(mpg, aes(x=cty, y=hwy)) +
  geom_label(aes(label=manufacturer))
```



iii) geom_text()

- x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

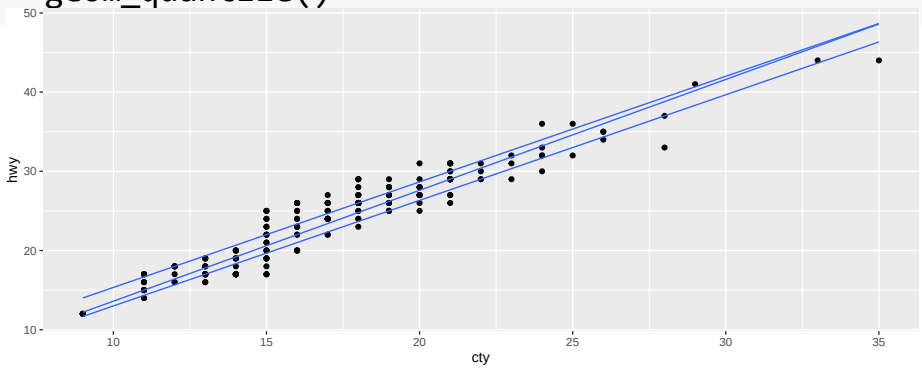
```
ggplot(mpg, aes(x=cty, y=hwy)) +
  geom_text(aes(label=manufacturer))
```



iv) geom_quantile()

- x, y, alpha, color, group, linetype, size, weight

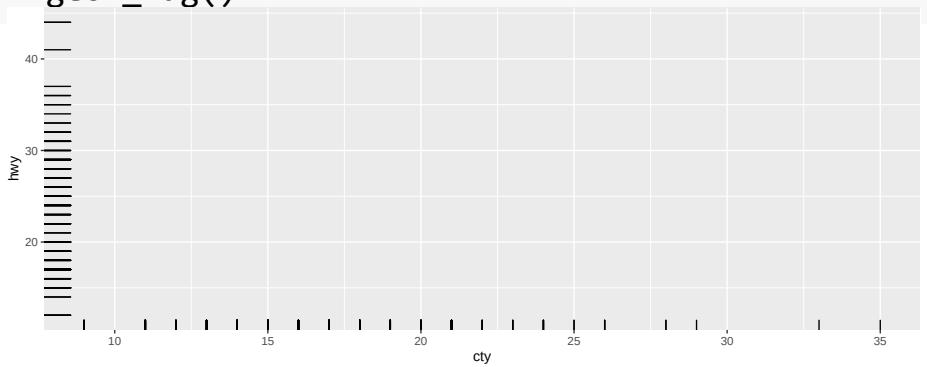
```
# install.packages("quantreg")
# (cf. Quantile regression)
ggplot(mpg, aes(x=cty, y=hwy)) +
  geom_point() +
  geom_quantile()
```



v) geom_rug()

- x, y, alpha, color, linetype, size

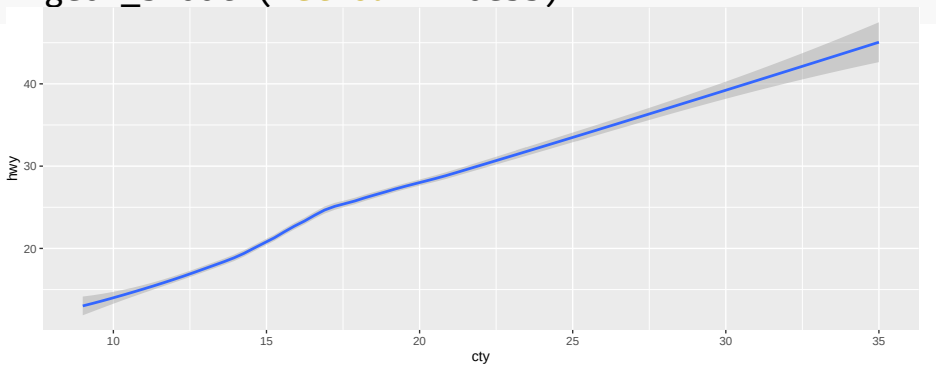
```
ggplot(mpg, aes(x=cty, y=hwy)) +
  geom_rug()
```



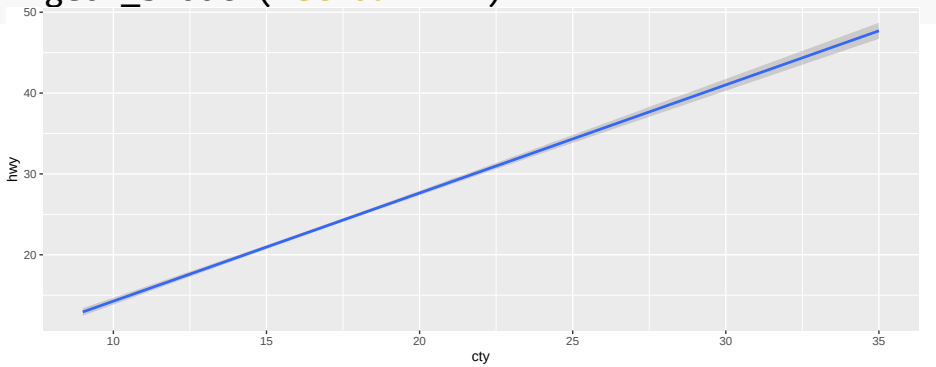
vi) geom_smooth()

- x, y, alpha, color, fill, group, linetype, size, weight

```
ggplot(mpg, aes(x=cty, y=hwy)) +
  geom_smooth(method = loess)
```

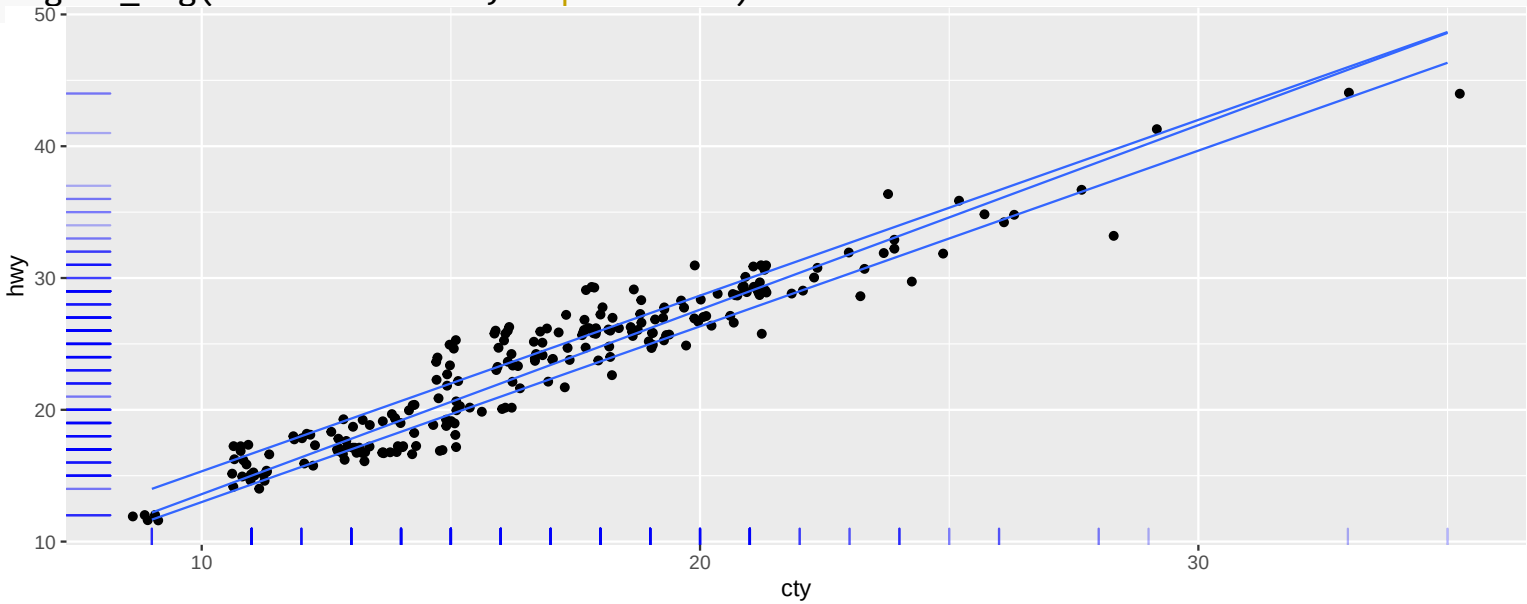



```
ggplot(mpg, aes(x=cty, y=hwy)) +
  geom_smooth(method = lm)
```



Example

```
ggplot(mpg, aes(x=cty, y=hwy)) +
  geom_point(position="jitter") +
  geom_quantile() +
  geom_rug(color = "blue", alpha = 0.3)
```



2-2. One discrete, one continuous

Dataset

```
dim(mpg)
```

```
## [1] 234  11
```

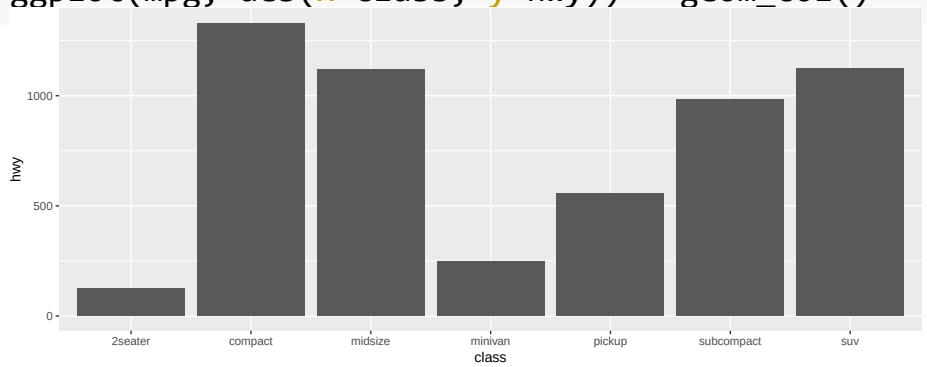
```
# `sample_n()` is from dplyr
mpg %>% select(class, hwy) %>% sample_n(10)
```

```
## # A tibble: 10 x 2
##    class        hwy
##    <chr>      <int>
##  1 suv           18
##  2 midsize       25
##  3 suv           19
##  4 minivan       24
##  5 subcompact    22
##  6 compact       29
##  7 minivan       23
##  8 pickup        16
##  9 suv           19
## 10 subcompact    27
```

i) geom_col()

- x, y, alpha, color, fill, group, linetype, size

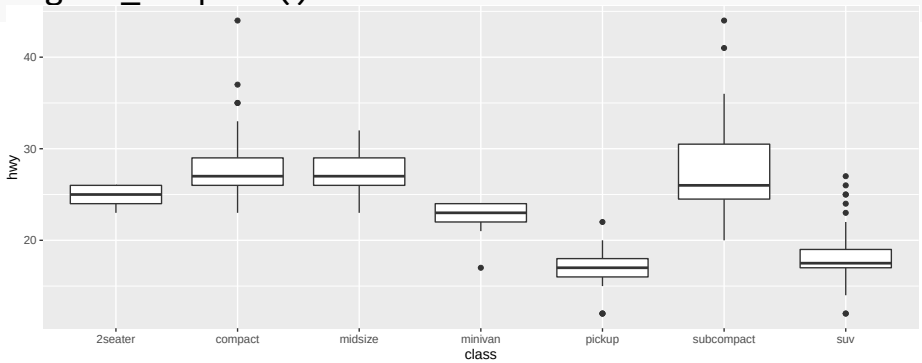
```
ggplot(mpg, aes(x=class, y=hwy)) + geom_col()
```



ii) geom_boxplot()

- x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

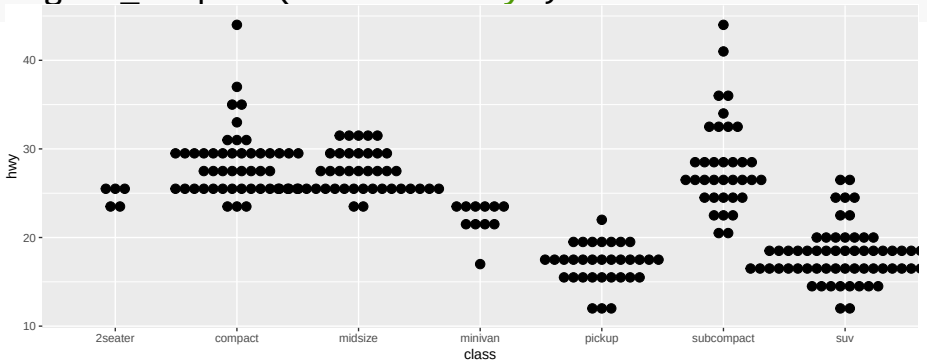
```
ggplot(mpg, aes(x=class, y=hwy)) +
  geom_boxplot()
```



iii) geom_dotplot()

- x, y, alpha, color, fill, group

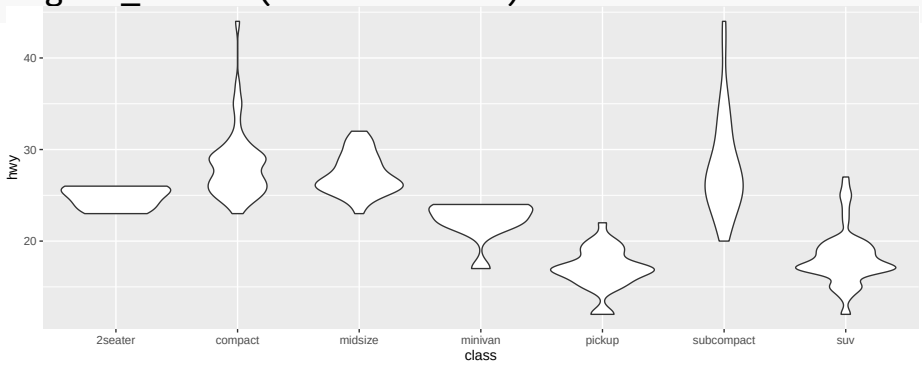
```
ggplot(mpg, aes(x=class, y=hwy)) +
  geom_dotplot(binaxis = "y", stackdir = "center")
```



iv) geom_violin()

- x, y, alpha, color, fill, group, linetype, size, weight

```
ggplot(mpg, aes(x=class, y=hwy)) +
  geom_violin(scale="area")
```



2-3. Both discrete

```
diamonds %>% select(cut, color) %>% sample_n(10)
```

```
## # A tibble: 10 x 2
##    cut       color
##    <ord>     <ord>
##  1 Very Good I
##  2 Ideal     D
##  3 Ideal     F
##  4 Ideal     G
##  5 Very Good H
##  6 Ideal     E
##  7 Premium   G
##  8 Ideal     H
##  9 Very Good G
## 10 Good      G
```

i) geom_count()

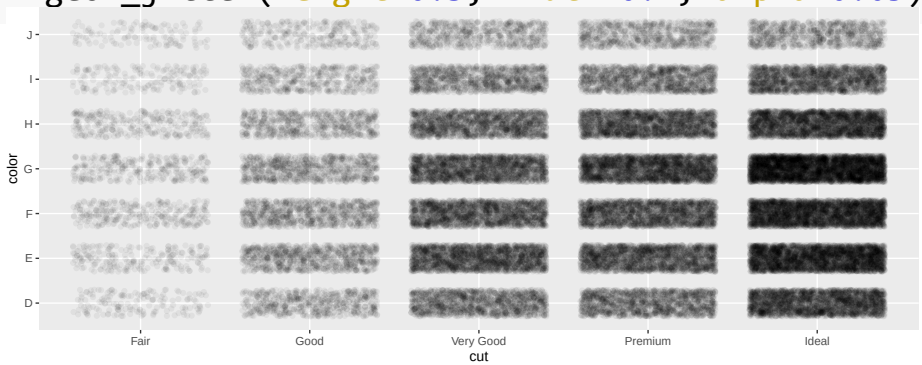
- x, y, alpha, color, fill, shape, size, stroke

```
ggplot(diamonds, aes(x=cut, y=color)) +
  geom_count()
```

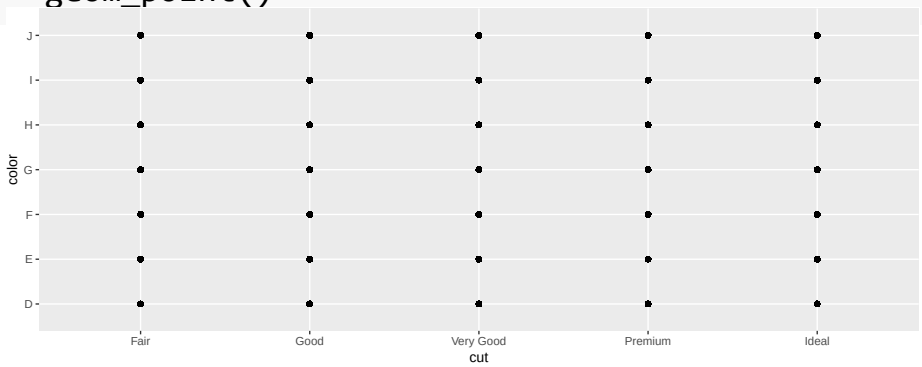
ii) geom_jitter()

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

```
ggplot(diamonds, aes(x=cut, y=color)) +
  geom_jitter(height=0.3, width=0.4, alpha=0.05)
```




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



Example

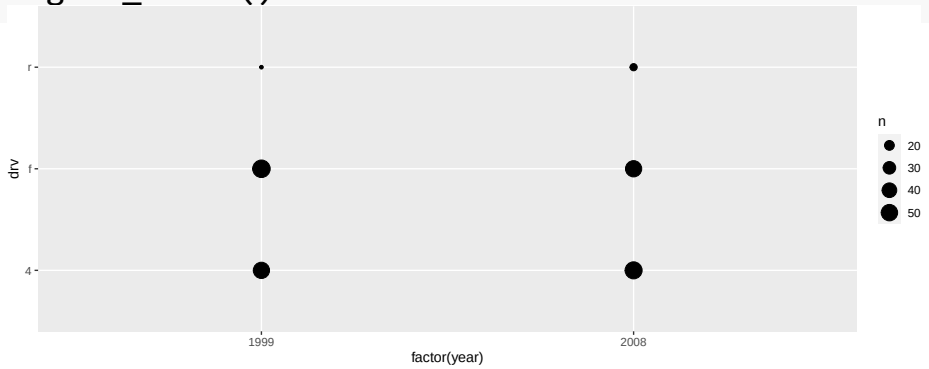
1. 아래의 summarized data frame을 시각화 한다면 어떻게 해야할 것인가?

```
mpg2 <- mpg %>%
  group_by(year=factor(year), drv=factor(drv)) %>%
  summarise(count=n())
mpg2
```

```
## # A tibble: 6 x 3
## # Groups:   year [2]
##   year  drv   count
##   <fct> <fct> <int>
## 1 1999  4        49
## 2 1999  f        57
## 3 1999  r        11
## 4 2008  4        54
## 5 2008  f        49
## 6 2008  r        14
```

2. `geom_count()`를 이용하면 가능하다.

```
ggplot(mpg, aes(x=factor(year), y=drv)) +
  geom_count()
```



- 충분히 informative 하지 않은것 같다. 차라리 `geom_text()`나 `geom_label()`을 사용하는 것이 낫지 않을까?

3. 많은 경우에 continuous를 위한 geom은 discrete에서 사용해도 무방하다.

geom_label()

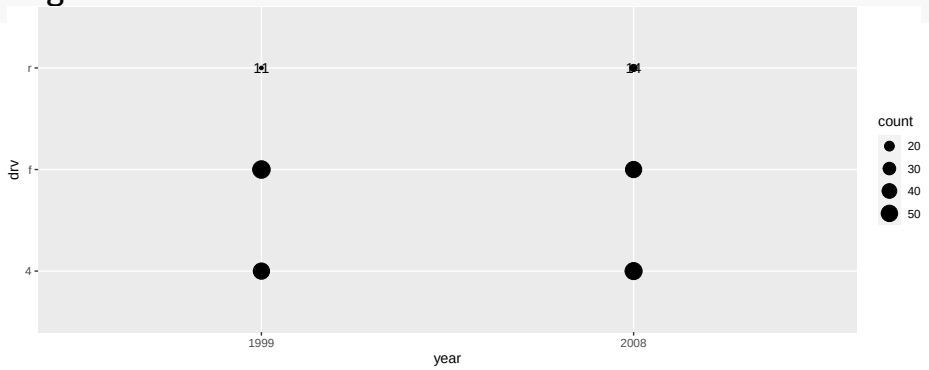
```
mpg2 %>% ggplot(aes(x=year, y=drv)) +
  geom_label(aes(label=count))
```

geom_text()

```
mpg2 %>% ggplot(aes(x=year, y=drv)) +
  geom_text(aes(label=count))
```

4. geom_point()와 geom_text()를 같이 사용해볼까?

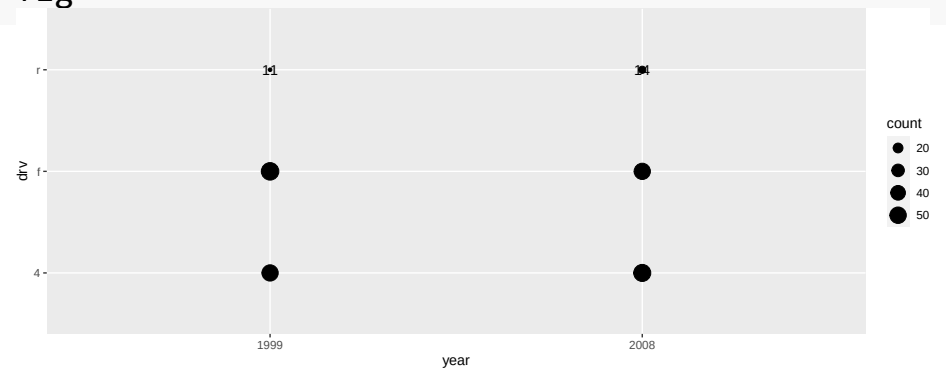
```
fig <- mpg2 %>% ggplot(aes(x=year, y=drv)) +
  geom_text(aes(label=count)) +
  geom_point(aes(size=count))
fig
```



5. fig의 개선사항은 아래와 같다.

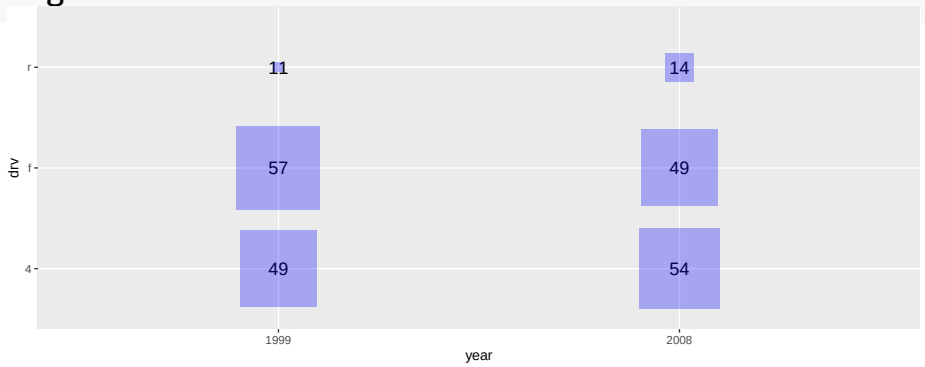
- 5.1 text의 크기가 너무 작다
- 5.2 점의 크기가 너무 작고 대비가 안된다.
- 5.3 text가 있다면 굳이 size에 대한 legend는 없어도 된다.

```
fig
```



6. 개선사항의 반영

```
fig <- mpg2 %>% ggplot(aes(x=year, y=drv)) +
  geom_text(aes(label=count), size=5) + # 5.1
  geom_point(
    aes(size=count), alpha=0.3,
    color="blue", shape="square") + # 5.2
  scale_size_continuous(range=c(3, 30)) + # 5.2
  theme(legend.position="none") # 5.3
fig
```

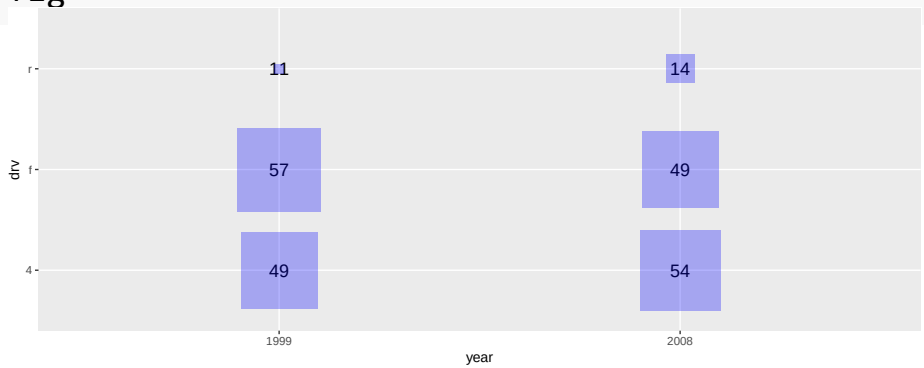


7. 코드 설명

```
fig <- mpg2 %>% ggplot(aes(x=year, y=drv)) +
  geom_text(aes(label=count), size=5) + # 5.1
  geom_point(aes(size=count), alpha=0.3, color="blue", shape="square") + # 5.2
  scale_size_continuous(range=c(3, 30)) + # 5.2
  theme(legend.position="none") # 5.3
```

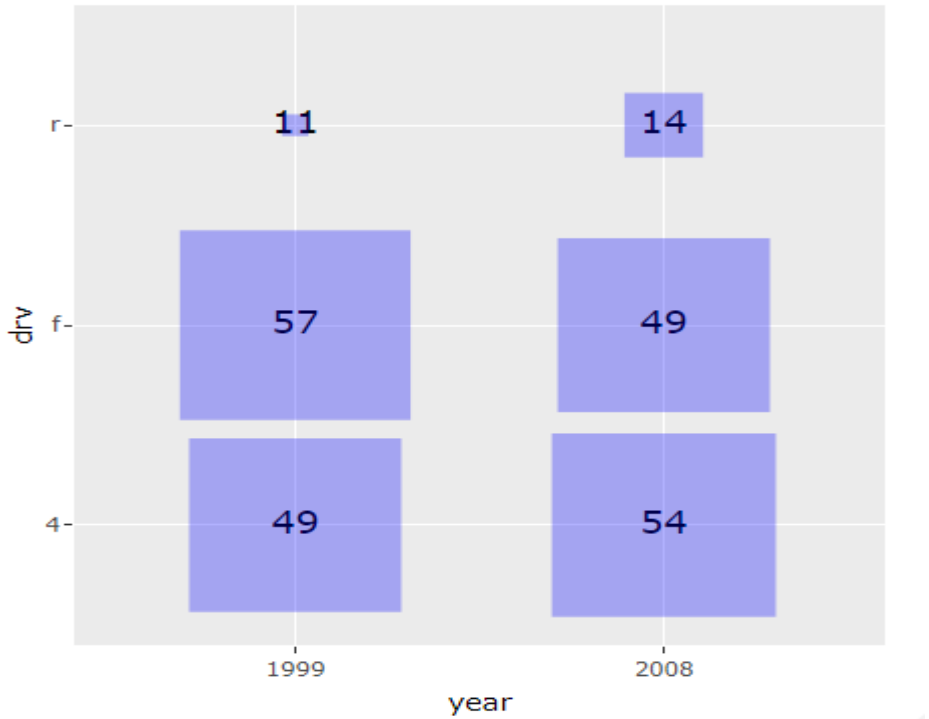
- `size=5`: text의 사이즈를 일괄적으로 크게 했음
- `alpha=0.3`: point를 투명하게 하여 text의 가독성을 높임
- `color="blue"`: point의 색을 바꾸어 text와의 대비를 높임
- `shape="square"`: point를 원이 아닌 사각으로 하여 text를 더 잘 수용하게 함
- `theme(legend.position = "none")`: legend를 없애버림
- `scale_size_continuous(range = c(3, 30))`: `size=count`의 효과를 강조

```
fig
```



8. `ggplotly()` 함수를 이용하면 더 보기좋은 그래픽을 얻을 수 있다.
   - 주의) `ggplotly()`에 의해서 생성된 객체는 `htmlwidget`에 속하기 때문에 rmarkdown에서 pdf나 docx로 렌더링 할때에는 아래의 코드가 작동하지 않는다.

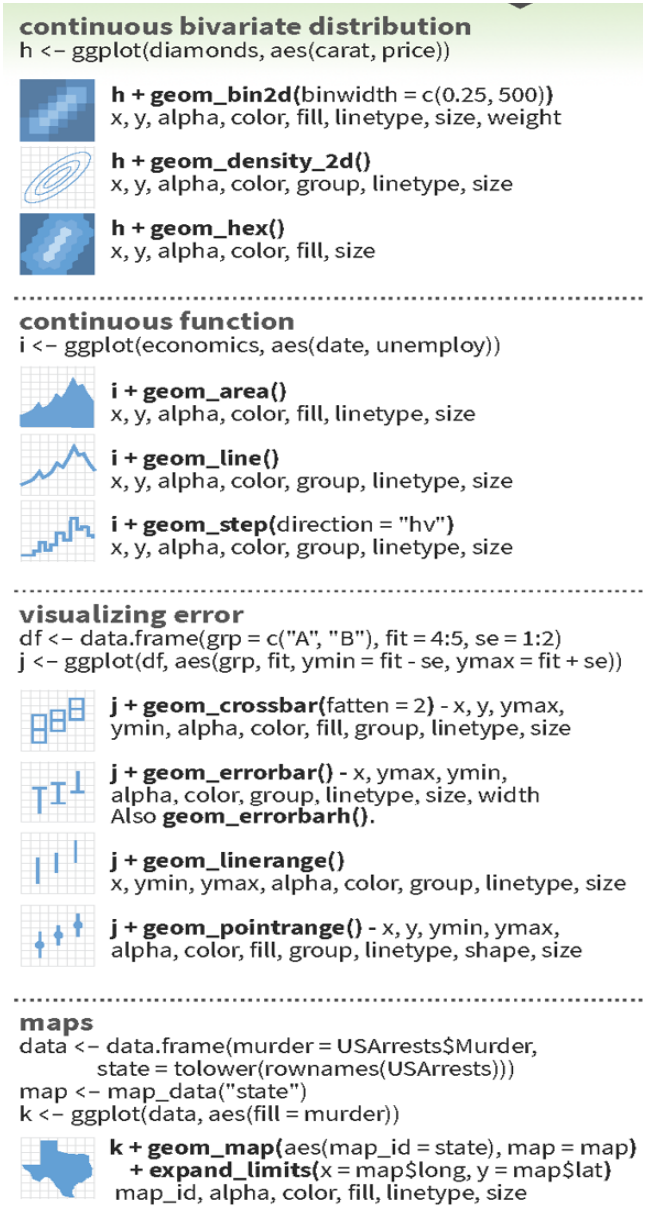
```
library(plotly)
ggplotly(fig)
```



3. Two variables (advanced)

3. Two variables (advanced)

- 3-1. Continuous bivariate
  - i) `geom_bin2d()`
  - ii) `geom_density_2d()`
  - iii) `geom_text()`
- 3-2. Continuous function
  - i) `geom_area()`
  - ii) `geom_line()`
  - iii) `geom_step()`
- 3-3. Visualizing error
  - i) `geom_crossbar()`
  - ii) `geom_errorbar()`
  - iii) `geom_linerange()`
  - iv) `geom_pointrange()`
- 3-4. Maps
  - i) `geom_map()`



3-1. Continuous bivariate distribution

A bivariate probability density function, typically notated as $f(x, y)$.

Dataset: `diamonds`

- A dataset containing the prices and other attributes of almost 54,000 diamonds.

```
colnames(diamonds)
```

```
## [1] "carat"   "cut"     "color"   "clarity" "depth"
## [8] "x"       "y"       "z"
```

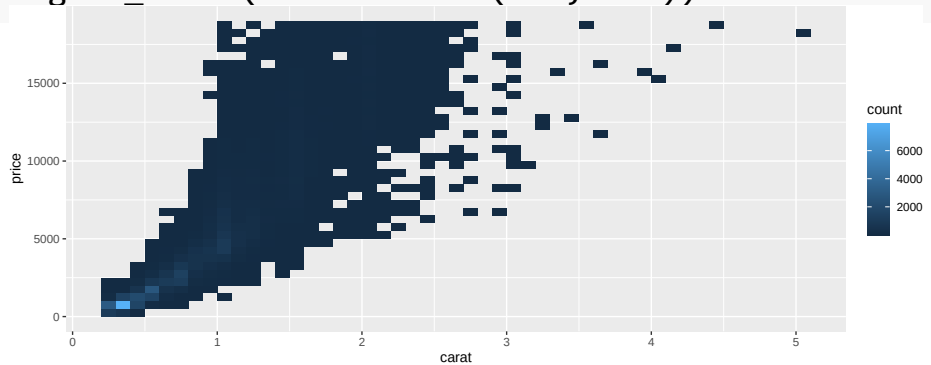
```
diamonds %>%
  select(carat, price) %>%
  sample_n(3)
```

```
## # A tibble: 3 x 2
##   carat price
##   <dbl> <int>
## 1  1.01  3461
## 2  0.32  1140
## 3  1.02  4381
```

i) `geom_bin2d()`

- `x, y, alpha, color, fill, linetype, size, weight`

```
ggplot(diamonds, aes(x=carat, y=price)) +
  geom_bin2d(binwidth = c(0.1, 500))
```

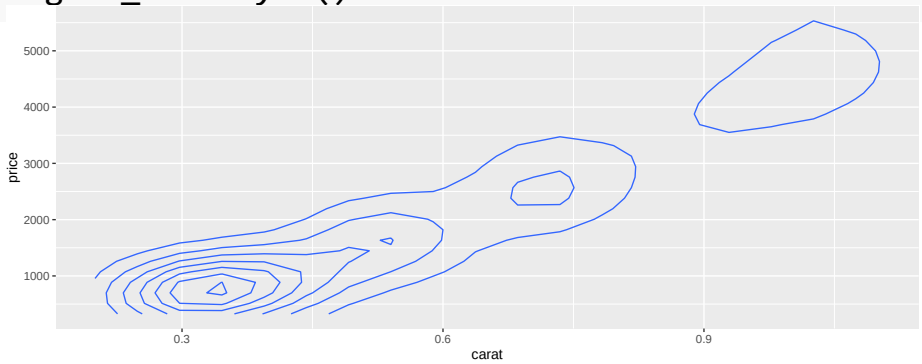


- `geom_count()`와 유사한 방식

ii) geom_density2d()

- x, y, alpha, colour, group, linetype, size

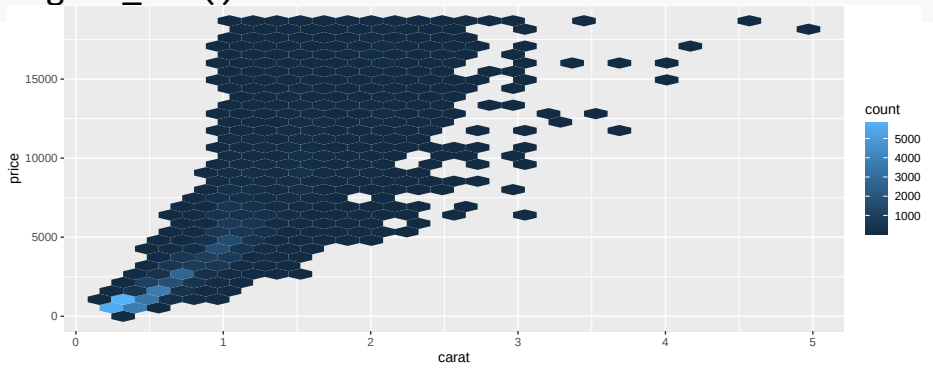
```
ggplot(diamonds, aes(x=carat, y=price)) +
  geom_density2d()
```



iii) geom_hex()

- x, y, alpha, colour, fill, size

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



3-2. Continuous function

Dataset: `economics`

- US economic time series data

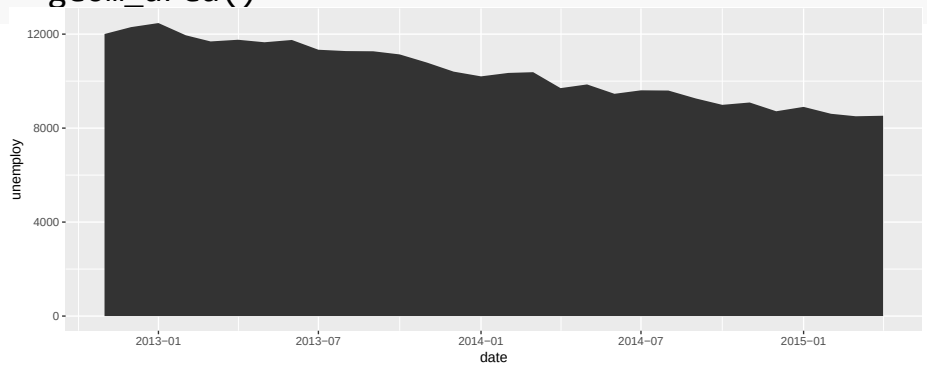
```
economics %>%
  select(date, unemploy) %>%
  head(3)
```

```
## # A tibble: 3 x 2
##   date       unemploy
##   <date>        <dbl>
## 1 1967-07-01     2944
## 2 1967-08-01     2945
## 3 1967-09-01     2958
```

i) `geom_area()`

- `x, y, alpha, color, fill, linetype, size`

```
economics %>% tail(30) %>%
  ggplot(aes(date, unemploy)) +
  geom_area()
```

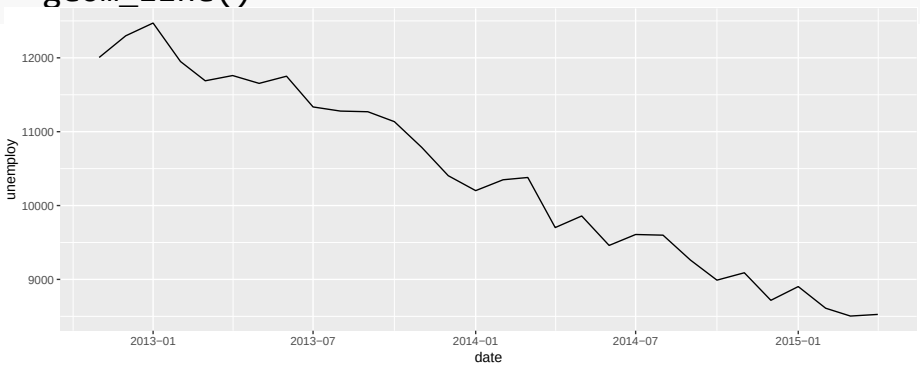


- **1. One variable**에서도 `geom_area()`가 있었음

ii) geom_line()

- x, y, alpha, color, group, linetype, size

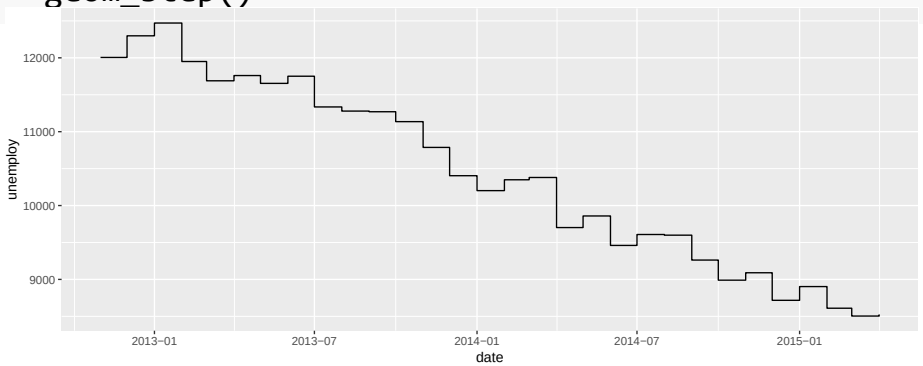
```
economics %>% tail(30) %>%
  ggplot(aes(date, unemploy)) +
  geom_line()
```



iii) geom_step()

- x, y, alpha, color, group, linetype, size

```
economics %>% tail(30) %>%
  ggplot(aes(date, unemploy)) +
  geom_step()
```



3-3. Visualizing error

Dataset preparation

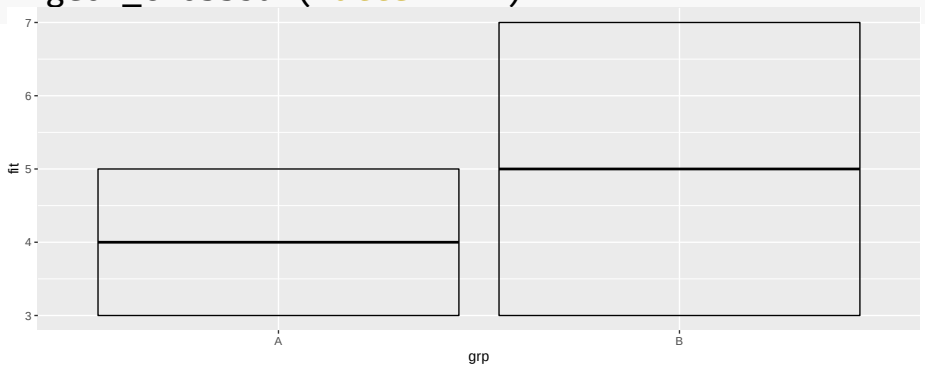
```
df <- data.frame(grp = c("A", "B"),
                 fit = 4:5,
                 se = 1:2)
df
```

```
##   grp fit se
## 1   A   4  1
## 2   B   5  2
```

i) `geom_crossbar(fatten = 2)`

- `x, y, ymax, ymin, alpha, color, fill, group, linetype, size`

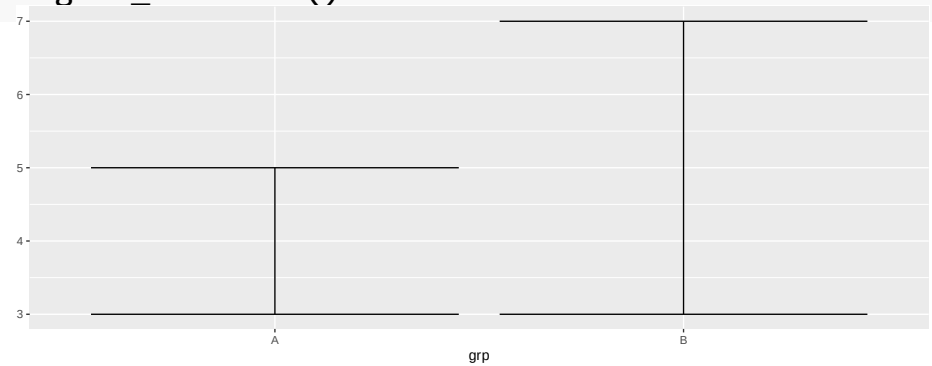
```
ggplot(df, aes(grp, fit,
               ymin = fit-se, ymax = fit+se)) +
  geom_crossbar(fatten = 2)
```



ii) geom_errorbar() (also geom_errorbarh())

- x, ymax, ymin, alpha, color, group, linetype, size, width

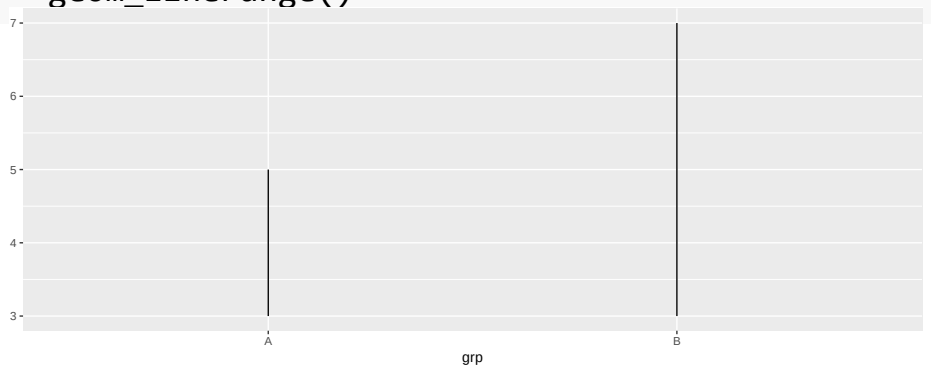
```
ggplot(df, aes(grp,
               ymin = fit-se, ymax = fit+se)) +
  geom_errorbar()
```



iii) geom_linerange()

- x, ymin, ymax, alpha, color, group, linetype, size

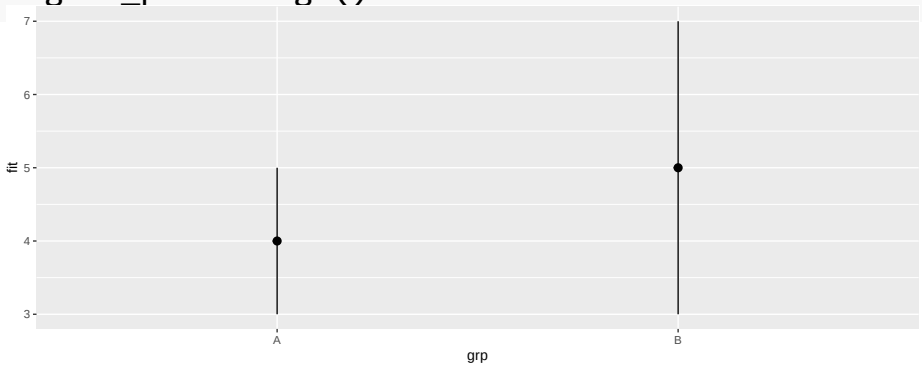
```
ggplot(df, aes(grp,
               ymin = fit-se, ymax = fit+se)) +
  geom_linerange()
```



iv) geom_pointrange()

- x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

```
ggplot(df, aes(grp, fit,
               ymin = fit-se, ymax = fit+se)) +
  geom_pointrange()
```

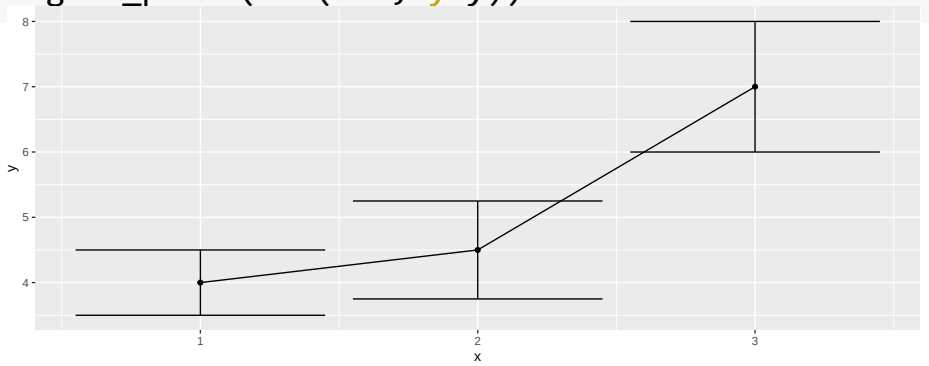


A realistic implementation

```
df <- data.frame(
  x = 1:3, y = c(4,4.5,7), se=seq(0.5,1,0.25))
df
```

```
##   x   y   se
## 1 1 4.0 0.50
## 2 2 4.5 0.75
## 3 3 7.0 1.00
```

```
ggplot(df) +
  geom_errorbar(aes(x=x, ymax=y+se, ymin=y-se)) +
  geom_line(aes(x=x, y=y)) +
  geom_point(aes(x=x, y=y))
```



3-4. Maps

Data preparation

통계데이터

```
data <- data.frame(
  murder = USArrests$Murder,
  state = tolower(rownames(USArrests)))
data %>% head(3)
```

```
##   murder   state
## 1   13.2 alabama
## 2   10.0  alaska
## 3    8.1 arizona
```

지리데이터

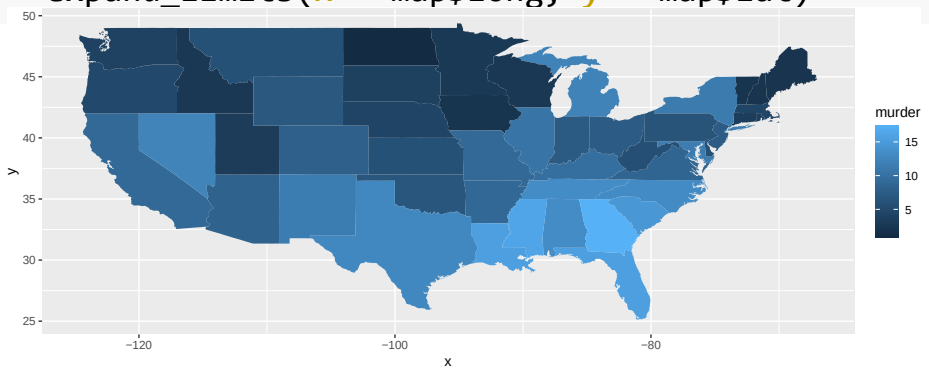
```
map <- map_data("state")
map %>% head(3)
```

```
##       long     lat group order  region subregion
## 1 -87.46201 30.38968     1     1 alabama      <NA>
## 2 -87.48493 30.37249     1     2 alabama      <NA>
## 3 -87.52503 30.37249     1     3 alabama      <NA>
```

i) geom_map()

- map_id, alpha, color, fill, linetype, size

```
ggplot(data, aes(fill = murder)) +
  geom_map(aes(map_id = state), map = map) +
  expand_limits(x = map$long, y = map$lat)
```



Exercise

- 직전의 차트의 color는 살인율이 높은 state를 더 옅게 표현하고 있다.
- 살인율이 높은 state가 더 짙은 빨간색으로 표현되도록 코드를 수정하라.

A quick summary

	geoms	variables
Barchart	`geom_bar()`	x: disc
Histogram	`geom_histogram()`	x: conti
Density	`geom_density()`	x: conti
Scatterplot	`geom_point()`	x: conti, y:conti
Bubblechart	`geom_point()` with `size`	x: conti, y:conti, `size`: numeric
Linechart	`geom_line()`	x: conti, y:conti
(fit line)	`geom_smooth(method="lm")`	x: conti, y:conti
Boxplot	`geom_boxplot()`	x: disc, y: conti
Violin	`geom_violin()`	x: disc, y: conti

```
"Tantum videmus quantum scimus."
```

```
## [1] "Tantum videmus quantum scimus."
```